Vision-Based UAV Fault Detection and Diagnosis

From Data to Prediction

José Ignacio de Alvear Cárdenas

Technische Universiteit Delft

TU Delft

This page was intentionally left blank

Vision-Based UAV Fault Detection and Diagnosis

From Data to Prediction

by

José Ignacio de Alvear Cárdenas

to obtain the degree of Master of Science
at the Delft University of Technology,
to be publicly defended on 22nd December 2022 at 9:30

Student number:	4463196	
Date completed:	22nd December 2022	
Thesis committee:	Dr. ir. C. C. de Visser	TU Delft, Control and Simulation
	Dr. M. D. Pavel,	TU Delft, Control and Simulation
	Dr. ir. E. Mooij,	TU Delft, Space Engineering

TU Delft

Preface

This work marks the end of my long seven-year journey at Delft University of Technology, the faculty of Aerospace Engineering and the small but ”gezellige” city of Delft. From my 25 years of existence, this chapter of my life has left a huge footprint on the person I am today. I had the opportunity of working with smart and inspiring people from every corner of the world and enjoyed the unlimited number of opportunities offered at TU Delft. From travelling around Europe with the IDEA League Challenge Programme and carrying out my internship at the German Aerospace Centre in Munich, to being part of a Dream Team and co-founding a consulting organisation; I came seeking personal growth and academic excellence, and I have not been disappointed.

Despite having enjoyed multiple individual research projects in the past as part of the Honours Programme, this has been by far the greatest challenge of my studies. Pursuing an ambitious project coupled with the worst pandemic the world has seen in decades has led to a turbulent journey. Back in 2020, I returned from a visit to some friends in Silicon Valley with great plans for the following three years. Little I knew that everything I had built was going to vanish within a month. Away from all the hecticness of travelling, extracurriculars and side projects, the last two years have served as time for reflection, introspection and goal-setting.

My master thesis has been a great learning experience which has enabled me for the first time to develop an idea ”from data to prediction”: from developing a simulator for data collection to training and testing an algorithm for fault detection and diagnosis. For this, I would like to thank my supervisor Coen de Visser for his continuous guidance and support. Thank you for giving me the space and resources to develop my ideas from the moment we met, encouraging me to think out-of-the-box, for believing in those ideas and pushing me to keep the bigger picture in mind. Online and offline, it has been a pleasure to work with you and hear your advice. I am also grateful to count on the expertise of Sihao Sun and Rudi Schilder in the early stages of the thesis, whose previous work inspired this project.

Besides those who contributed to the present work, I would also like to thank Fabrizio Oliviero, Erik-Jan van Kampen and Bo Sun, who supervised my Bachelor and Master Honours research projects. They were the first ones to encourage me to develop an idea from concept to publication and they are the reason behind my confidence for tackling such an ambitious master thesis. I am also grateful to those lecturers and researchers whose personal advice and support (unknowingly) contributed to important decisions I have taken throughout the last seven years, namely Jacco Hoekstra, Sergio Turteltaub, Francesco Avallone and Gertjan Looye.

Thank you to all my friends with whom I have shared great memories and who supported me through thick and thin, especially in the past few years. *Grazie mille* Giulio for always been there; from the European Space Agency to the Mare Tranquillitatis, this journey would definitely not have been the same without you. Thank you Rafa for showing TU Delft to a kid with big dreams back in 2014, for your continuous support and intense tennis matches. Thank you Johannes and Stijn, two of the most ambitious people I have met, for all the past and future laughs and experiences together around Europe and beyond. Thank you Daniel for cheering me up during the thesis, being the best running mate and sharing your invaluable insights about green aviation. Thank you Bart for the great collaboration during the master, the discussions about future professional paths and always checking in despite the distance. Thank you to Laura, Lorenzo, Nikita, Alejandro and Tiago for those amazing board game nights, bouldering sessions and deep scientific discussions during the roughest parts of the pandemic. Thank you Wesley and Pierre for joining me in the adventure of learning about artificial intelligence and discussing future ambitions. Thank you to the running team of TU Delft, especially Akshit, Pablo, Tjeerd, Aniket and Garazi. Thank you Jacopo, Julia, Jasper, Jonatthas, Rohan, Sandro, Rano, Andrea, Lidia, Ramesh, Juan José, Carsten, Felix and many others. The list goes on and on, you know who you are: thank you very much from the bottom of my heart!

A special thank you to my girlfriend Lara, who never stopped rooting for me. You have always listened for hours to my ideas and given me the space and time I needed to finish this work. In spite of the distance, I always felt you were by my side believing in me.

My deepest gratitude goes to my fantastic family. To my parents and my little brother who supported me unconditionally during this project and, in general, during my studies. Thanks for being a source of inspiration everyday and for pushing me to achieve my dreams, regardless of how far they seem. Without you, nothing of this would have been possible. I know I can always count on you.

I have lived this experience to the fullest, to the point that I could say that I have obtained the title of ”*Vive TU Delft*”. This is the closure of a chapter in my life and the start of a new one. Beginning of 2023, I will start as a research scholar at NASA Ames Research Centre in California, definitely a dream come true. I have never discarded to do a PhD so, who knows, I may be back in academia in the (near) future; only time will tell!

José Ignacio de Alvear Cárdenas
Delft, December 2022

Contents

List of Figures

List of Tables

I

Thesis introduction

1 Introduction

With the advent of Smart Cities, Unmanned Air Vehicles (UAVs) have seen a surge in their number of applications, from package delivery [31, 32] to Urban Air Mobility (UAM) [33]. Most recently, as a response to the COVID-19 pandemic [34, 35], the implementation of UAVs for medical purposes has been accelerated. Zipline, a drone start-up in California (USA), has been granted permission for transporting medical supplies in North Carolina [36] and AVY, a start-up based in Amsterdam (The Netherlands), has received a grant from the European Commission for urgent medical transport between healthcare facilities [37, 38]. With Air Traffic Control programs under development for the management of drones, such as the U-Space in Europe [39], one of the main concerns of the future crowded urban airspace is safety [40].

Most of the research in this field has been focused on fault tolerant control [41], with companies such as Verity Studios, which successfully filed a patent in 2020 for a final product [42]. However, in order to improve the resilience of multi-rotor and hybrid drones to potential failures, work is also carried out in obstacle avoidance [43], upset recovery [44], system identification [28] or fault detection and diagnosis [45]; the latter consisting of the fault classification, as well as its location and magnitude identification. Fault detection and diagnosis (FDD) expands the envelope of the UAV's self-awareness and allows informed decisions when deploying emergency systems, such as a parachute, and switching between controllers or internal physics models to counteract a failure. Literature in FDD is very extensive but, as it will be shown later, it deals with a single failure type at a time and has been limited to the manipulation of signals from the Inertial Measurement Unit (IMU), namely the accelerometers and gyroscopes, or additional external sensors such as microphones, which add weight and complexity to the system. Cameras are nowadays ubiquitous in commercial UAVs and they have been ignored for this task, even though their information is already processed for navigation, such as Simultaneous Localisation and Mapping (SLAM) [46], and state estimation in GPS denied urban regions, such as Visual Inertial Odometry [22]. Visual information is very rich and it could potentially identify multiple failure types at once, as well as increase the accuracy when fused with the IMU sensors. This research gap widens when considering future advancements in UAV hardware and software, such as event-based cameras [47] and spiking neural networks [48]. Therefore, the main research objective of this thesis is stated below.

The objective of this research is to improve the accuracy of UAV actuator fault detection and diagnosis by developing a framework that fuses IMU and vision-based information.

For the development and performance assessment of vision-based FDD algorithms, it is required to have a dataset which includes IMU and camera output in nominal flight and in failure scenarios. Unfortunately, the current available datasets do not include IMU sensor information and do not have any recorded scenarios with failures. Gathering large quantities of data for knowledge-based fault detection models with a UAV is very time consuming, dangerous and expensive; data would have to be annotated, multiple failure modes would have to be induced in the vehicle and the flight environment, as well as the UAV, would have to be adapted to minimise the potential risk. Besides that, in an experimental physical setting it is very difficult to collect data from various environments and conditions. A suitable alternative is the simulation of the vehicle in a realistic environment, the storage of the sensor synthetic data for model training and the transfer learning to the real world UAV. It has been observed that the addition of large quantities of synthetic data to a smaller real dataset would lead to a performance increment when compared to the scenario in which only real data is collected [49].

Unfortunately, there does not exist an off-the-shelf simulator that enables the generation of synthetic flight data with mid-flight injected failures. Additionally, previous literature in the field of fault diagnosis has simplified the blade damage to a simple loss of control effectiveness or to the centrifugal forces caused by the shift in the centre of gravity. Such approaches neglect the vibrations in the moment signals, as well as the vibrations caused by the changed aerodynamics due to the centre of pressure displacement.

Hence, in order to achieve the proposed research objective, three main research questions have been identified; each tackling a different research project phase: blade damage modelling, data gathering and FDD framework development. With their respective research subquestions, they have been defined as follows:

1. How can high-fidelity blade damage be modelled?
2. How can data be gathered for the training and testing of the UAV fault detection and diagnosis framework?
   (a) What simulator can be used for the development of the data gathering pipeline?
   (b) What actuator faults are considered for the development of the UAV fault detection and diagnosis framework?
   (c) What drone physics model and controller should be used for obtaining the data?
3. How can the camera sensor information be fused with IMU data within a UAV fault detection and diagnosis framework?
   (a) How can features be extracted from the camera and IMU sensor data?
   (b) What method shall be used for UAV actuator fault detection?
   (c) What method shall be used for UAV actuator fault diagnosis?
   (d) How do the vision-based features contribute to the accuracy and computational load of the UAV fault detection and diagnosis framework?

The first question aims at developing a model capable of simulating blade damage more accurately than the alternatives available in literature. The second research question aims at understanding how a rich dataset can be built. The third research question aims at filling the research gap by tackling the FDD framework development and assessing its performance.

As far as the contents of the present report are concerned, an overview is provided hereby. First, Part II contains a literature study on existing FDD approaches, applications of visual data on board of autonomous vehicles and photo-realistic simulators; information leveraged throughout the rest of this project. Then, Part III presents two scientific papers — the main contribution of the research and the master thesis. Next, Part IV contains the thesis report, which explains in greater detail the methodology and results, easing the reproducibility of this work. Part V recaps the main findings of this research project and provides recommendations for further work in each of the project phases. Finally, Part VI encloses additional supporting work.

II

Literature study

2 Fault detection and diagnosis

Fault is defined as "*an unpermitted deviation of at least one characteristic feature of the system from the acceptable, usual, standard condition*" [50], reducing the capability of performing a required task. Failure and malfunctions are the result of the accumulation of one or more faults that lead to the permanent interruption or intermittent irregularity in the performance of a system function under the specified operating conditions.

Depending on their development through time, faults can be classified in 3 groups [50–52]: abrupt faults, that take place in a step-wise form such as impact faults, bias faults, stuck faults and loss of signal [53]; incipient faults, that show a drifting behaviour; and intermittent faults. Besides that, depending on where in the system they take place, they can be classified at the same time as sensor faults, actuator faults and plant faults [54], being the first two groups those that most literature aim at predicting. Sensor faults result from incorrect readings from the system instruments and sensors and they include constant bias faults (stuck sensor), drift fault (additive-type), constant gain faults (multiplicative-type) and outlier faults [6, 55].

At the beginning, faults were only compensated with hardware redundancy techniques due to the lack of effective algorithms for fault detection and the low computational power available. However, they add weight and are not robust against sensor noise or potential cyber-attacks [6, 56]. In the last 50 years, algorithms have been developed to detect these faults without the need of comparing duplicated signals provided from multiple sensors (Figure 2.1 [1]). Instead, information redundancy between the measured data and an explicit or implicit model is checked by a search engine. They are called analytical redundancy and they have been implemented in numerous systems, from chemical plants to aeronautics.

There exist numerous review documents in the subject of fault detection and diagnosis with analytical redundancy, being a field whose terminology has constantly evolved through time. Although fault classification, fault identification, fault isolation and fault diagnosis have been used interchangeably throughout literature, originally each term was tied to a different group of tasks [1, 51]:

1. **Fault classification**: fault type
2. **Fault isolation**: fault type and location
3. **Fault identification**: fault magnitude (size)
4. **Fault diagnosis**: fault type, size, location and time of detection

As can be observed, fault diagnosis encompasses all the other terms. Besides that, fault accommodation refers to the replacement or partial compensation of the faulty sensor measurement by a computed estimation.

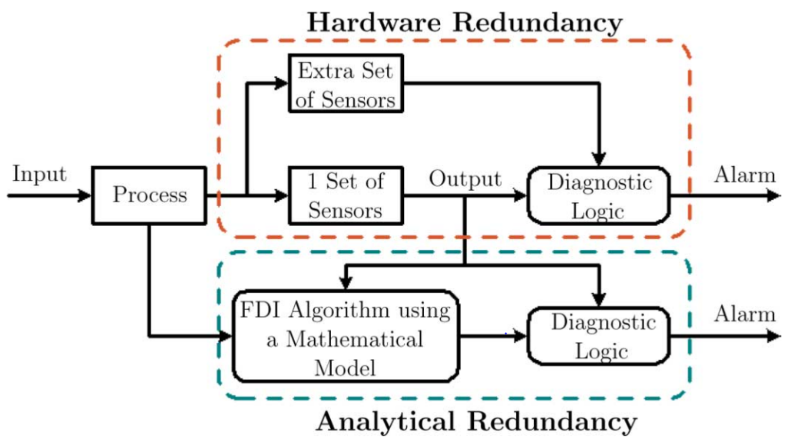


Figure 2.1: Graphical comparison between hardware and analytical redundancy [1].

In 1997, Isermann [51] provided an overview of model-based methods for fault detection that could be classified in parameter estimation, state estimation and parity equations. The introduction of human expert heuristic knowledge with approximate reasoning (e.g. fuzzy rules) in fault diagnosis was also discussed. Later in 2006, the same author provided a broader selection of fault detection methods [50]. Apart from the conventional limit value checking (e.g trend checking, deviation from mean or variance), the author extended the aforementioned model-based methods with non-linear process identification methods (e.g. Artificial Neural Networks), fault-detection with signal models (their transformation to the frequency domain and their analysis) and fault detection with Principal Component Analysis (PCA).

Two years later, Zhang [2] presented another classification of fault detection and diagnosis methods based on the a priori knowledge used by the algorithm, which can be observed in Figure 2.2 [2]. This lead to two large groups, namely: model-based methods and data-based methods. Model-based methods, also referred to as deep or causal, understand the process using first principles knowledge; the physics of the process. Their goal is to detect whether there is consistency between the behaviour of the model and the actual system, even in the presence of system input deviations and disturbances (robustness). For that purpose, the system input is fed to the generated model whose output is compared to the system output data in order to generate residuals. Then, those residuals are passed to a classifier that identifies whether there is a fault and its type. At the same time, this group can be further divided as qualitative and quantitative. In quantitative approaches, mathematical functional relationships are established between the input and the output of the system [57]. In contrast, qualitative algorithms exploit qualitative functions which capture the causal structure present in the process less rigidly than quantitative model-based approaches but in a more detailed manner than expert systems, and they can be built as causal models or abstraction hierarchies [58]. Expert systems are if-then-else based algorithms without physics understanding that mimic the human expert behaviour when solving a domain-specific problem, unable to work in new unexpected conditions.

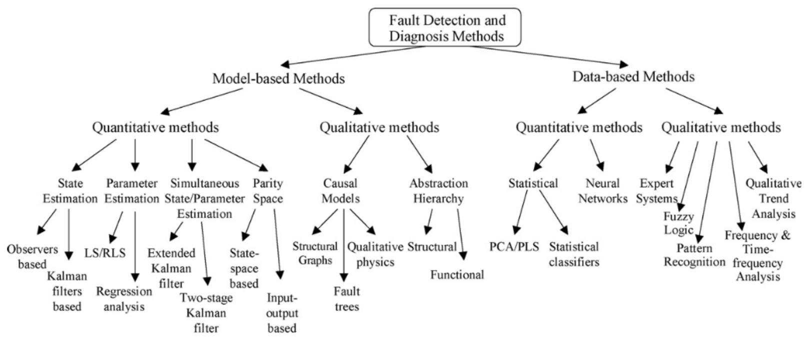


Figure 2.2: Fault detection and diagnosis classification based on [2].

On the other side of the spectrum are data-based approaches, originally called shallow, compiled, evidential or process history-based models, which obtain their knowledge from past experience with the process in the form of features extracted from large quantities of historical data. As can be seen, they are again subdivided in qualitative and quantitative approaches and the latter can be further broken down into statistical and non-statistical (Neural Networks) feature extraction methods [59].

Additionally, an alternative classification was presented based on whether the method was mostly exploited for the generation of residuals or for the fault decision making [1, 2]. The residuals are usually the difference between the system output and the developed model, and it should be zero under no-fault conditions.

Then, in 2013, Dai [3] proposed a classification that combined the initial insights of [50], which considered fault detection with signal models, and the aforementioned division between model-based and data-based approaches [2]. Based on how the data is processed, 3 groups are considered (Figure 2.3) [3]: model-based, signal-based and knowledge-based. The first approach is the same as discussed by [2], which compares the measured data with the predictions from an identified or derived from first principles model. It is the method which requires the least amount of data and the information redundancy is in the explicit model. It is also known as online data-driven.

In contrast, in the second approach the analytical redundancy is found in the fault-signal patterns relation which are based on a priori human understanding. They do not require an input-output model but more data (e.g. electronic signals and vibrations) is usually fed to these methods than to model-based approaches. The author further breaks down the signal-based approaches in those executed in the time domain, those in the frequency domain and those in the joint time frequency domain.

Finally, the last group is the equivalent to the data-based approach. The author considered the ”data-driven” nomenclature misleading since all the fault detection and diagnosis algorithms require historical and/or measured data. The goal of this approach is the development of a knowledge-based implicit representation of the system variables and it is mostly used when the process is too complex for a model-based strategy and the signal analysis does not lead to accurate results. When compared to the other two groups, knowledge-based methods require greater amounts of historic data in order to create its implicit models that encapsulate the information redundancy.

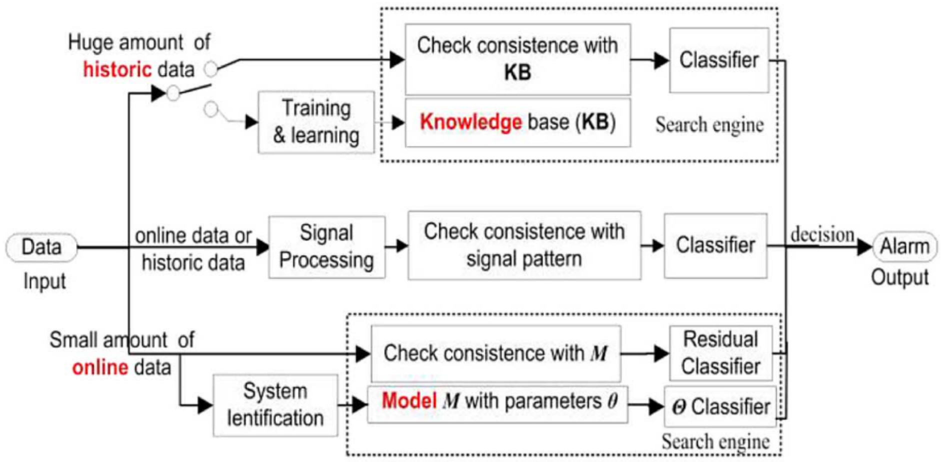

Huge amount of historic data
Training & learning
Check consistence with KB
Knowledge base (KB)
Classifier
Search engine
Data
Input
online data or historic data
Signal Processing
Check consistence with signal pattern
Classifier
decision
Alarm
Output
Small amount of online data
System lentification
Check consistence with M
Model M with parameters θ
Residual Classifier
θ Classifier
Search engine

Figure 2.3: Data flow in the knowledge-based, signal-based and model-based fault detection and diagnosis approaches [3].

Besides that, Dai [3] divides the knowledge-based approach in qualitative or symbolic AI and quantitative or machine learning. Additionally, the author distinguishes 3 new groups within machine learning, namely supervised learning, unsupervised learning and reinforcement learning, as can be observed in Figure 2.4 [3].

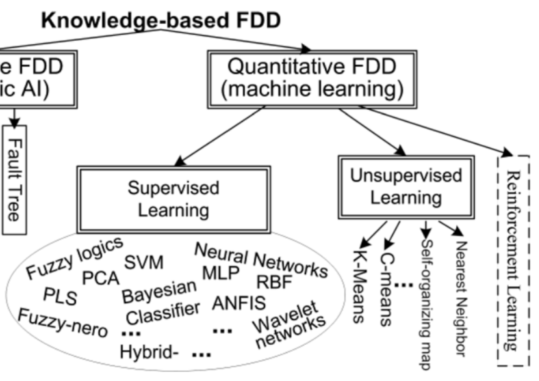


Figure 2.4: Knowledge-based fault detection and diagnosis approach classification [3].

More recent literature either maintains the discussed 3 group classification based on how the data is processed or forgets the signal-based approaches. In both cases, they enrich the classification by including more recent methods, such as the model-based Signed Bond Graph or the unsupervised knowledge-based Auto-Encoder [54, 60]. This shows that the terminology used in this field started to consolidate and become widely accepted. Additionally, previous literature has defined desirable characteristics for fault detection and diagnosis, such as detection speed or robustness, and employed it for the evaluation and comparison of the different approaches [2, 54, 59].

The combination of previous literature would lead to the complete and common classification framework presented in Figure 2.5. In UAV platforms, these algorithms are developed taking into consideration the limited computational capacity of micro-controllers.

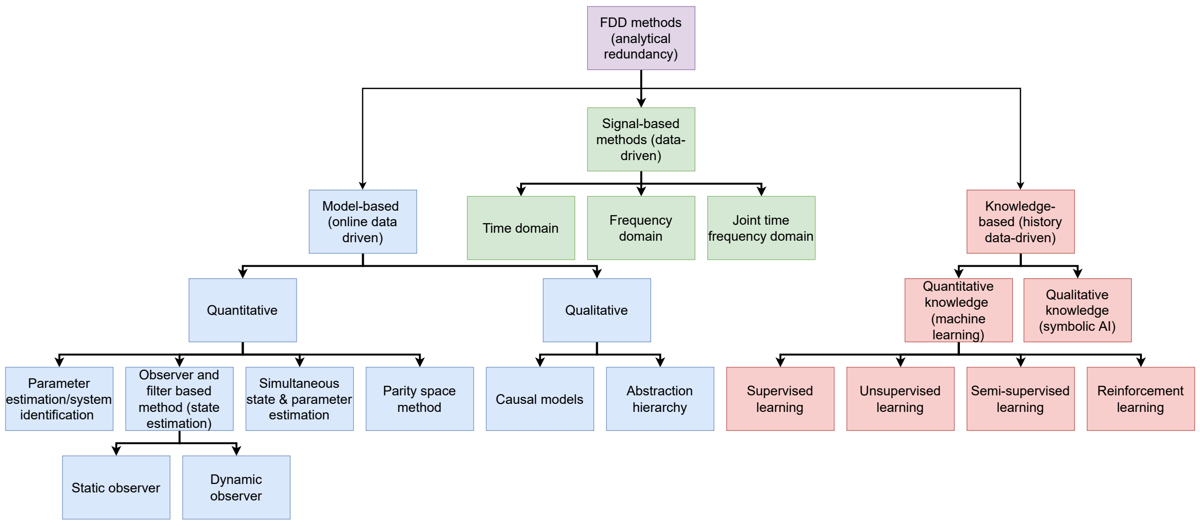

FDD methods (analytical redundancy)
Signal-based methods (data-driven)
Model-based (online data driven)
Time domain
Frequency domain
Joint time frequency domain
Knowledge-based (history data-driven)
Quantitative
Qualitative
Quantitative knowledge (machine learning)
Qualitative knowledge (symbolic AI)
Parameter estimation/system identification
Observer and filter based method (state estimation)
Simultaneous state & parameter estimation
Parity space method
Causal models
Abstraction hierarchy
Supervised learning
Unsupervised learning
Semi-supervised learning
Reinforcement learning
Static observer
Dynamic observer

Figure 2.5: Fault detection and diagnosis method classification.

In the next sections only the knowledge-based approaches will be presented in detail, including multiple examples and architectures from literature since they have historically shown to be the most suitable for dealing with high-dimensional visual data. In contrast with model-based approaches, they do not make use of the physical properties of the system and do not build a mathematical model for detection. As can be observed in Figure 2.6, they can be divided in quantitative methods, more commonly known as machine learning, and qualitative methods, so called symbolic AI. Quantitative methods consider the detection as a classification problem, which can be supervised, unsupervised, semi-supervised or reinforcement learning. These algorithms are exploited in modern fault detection systems because they do not require a lot of computations, allowing their use online in real-time. Before applying these approaches, pre-processing of data is required.

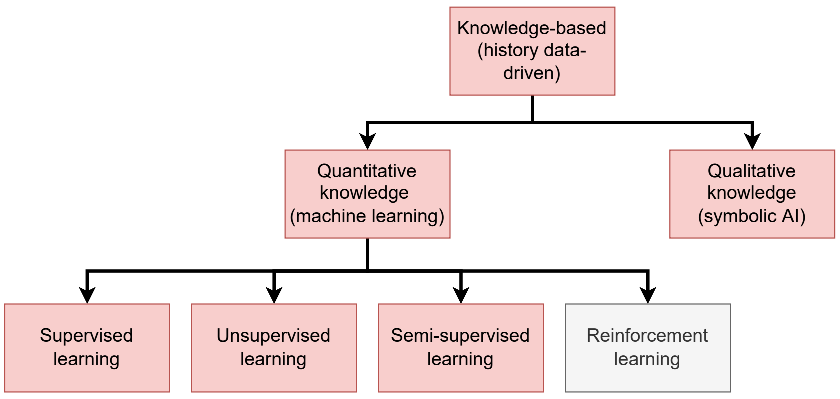

Knowledge-based (history data-driven)
Quantitative knowledge (machine learning)
Qualitative knowledge (symbolic AI)
Supervised learning
Unsupervised learning
Semi-supervised learning
Reinforcement learning

Figure 2.6: Knowledge-based approach classification.

This chapter is structured as follows. First, section 2.1 presents the supervised learning, building from the simple building blocks to the more complex strategies. Then, section 2.2 discusses the unsupervised alternatives. Since semi-supervised learning approaches usually undergo an initial unsupervised learning phase, some of its most important algorithms are included within this section. Finally, section 2.3 briefly discuses the most important qualitative knowledge approaches. Reinforcement learning is not covered since it is a technique mostly exploited in the field of control and not fault detection and diagnosis.

2.1. Quantitative knowledge: supervised learning

Supervised learning exploits large amounts of labelled historical data in order to train the algorithms. For that purpose, the data points have already been classified according to the system conditions at the time of measurement, namely healthy or faulty and the corresponding type of fault. The goal of the algorithms is to find the data patterns that can be connected to the different types of labelled faults. Figure 2.7 shows some of the most common methods within supervised learning used for fault detection and diagnosis. In the coming sections, the highlighted methods will be described in detail with examples from literature.

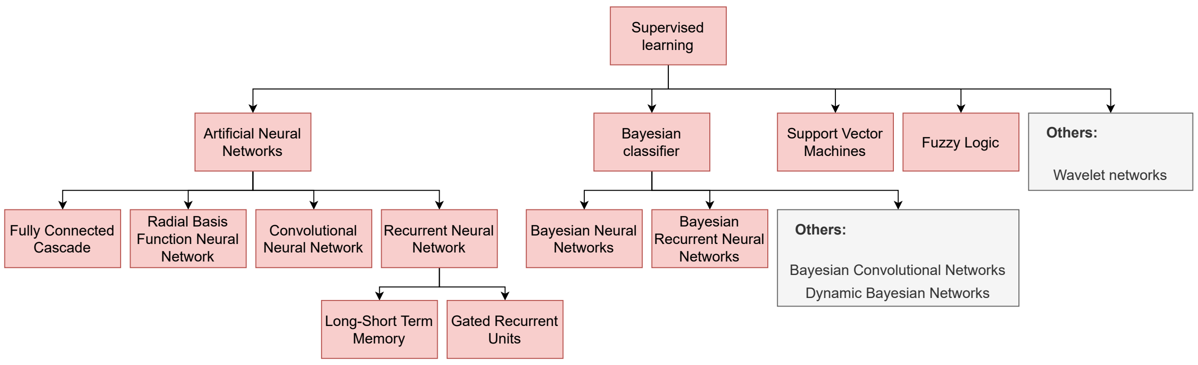


Figure 2.7: Supervised learning knowledge-based fault detection and diagnosis methods.

2.1.1. Artificial Neural Networks

Artificial Neural Networks (ANNs), also known as deep networks, are an efficient knowledge-based tool capable of representing an arbitrary function $y = \text{ANN}(x)$, which is exploited in the modelling and identification of nonlinear systems thanks to its flexible structure and approximation capabilities [14, 61]. It is inspired by how the brain works and each node in the network is aimed at mimicking the behaviour of a neuron.

Although there has been a surge of many variations during the last 30 years, from Convolutional Neural Networks (CNNs) in the 1990s [62] to the most recent advancements of Transformers [19], the basic concept remains constant. An ANN is composed of a collection of weighted activation functions (neurons) distributed among 3 types of computational layers: input layer, hidden layer and output layer. Even though the terms ANN and deep network are used interchangeably, an ANN is considered to be a deep network when the model contains multiple hidden layers.

Figure 2.8 [4] shows a clear representation of a neuron. Each neuron is a computational unit that receives one or multiple inputs (x_i) multiplied by their respective weights (W_i), adds them up to an external term known as the bias (b) and passes them through a nonlinear function before providing the result as output. There exist multiple types of nonlinear functions (e.g. ReLU, sigmoid, softmax) and their use greatly depends on the problem at hand. From the simplicity in the structure of a neuron, it can be deduced that modularity is at the core of ANNs since neuron can be connected to each other shaping diverse architectures.

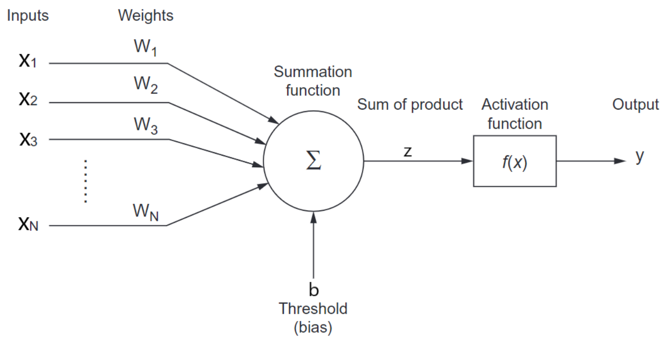


Figure 2.8: Artificial Neuron computational diagram [4].

In its simplest form (feedforward NN), each layer l_j in the ANN applies a non linear operation to a linear transformation ($f_i(\cdot)$) of the previous layer [4, 14]:

$$\begin{aligned} &l_0 = x, \\ &l_j = f_j(\mathbf{W}_j l_{j-1} + b_j) \quad \forall j \in [1, n], \\ &y = l_n \end{aligned} \tag{2.1}$$

Once the last layer is reached, the output is compared to the target output in a loss function. As with the nonlinear functions, there are multiple types of loss functions (e.g. mean squared error, categorical cross entropy) and their use greatly depends on the problem being solved. Finally, the parameters of the network (the weights and the biases) are updated with the derivative of the loss function with respect to the parameters in order to minimise the loss through a process which exploits the chain rule, known as backpropagation [10, 63].

ANNs can be used in a supervised setting where the targets are provided next to the inputs during the learning phase. For instance, Heo [64] considers two types of neural networks with labelled data for the tasks of fault detection and classification of a benchmark chemical process. In the first option, a neural network is trained for a binary classification task in which the data is labelled as normal or with a specific type of fault (fault detection). In the second option, the classification is performed by a neural network trained with normal operation data and data from various faulty operating conditions, performing a multi-class classification task. In order to assess the performance of the neural networks, 3 indices where compared, namely the accuracy, the fault detection rate (FDR) and the false alarm rate (FAR), which are defined as follows:

$$\text{Accuracy} = \frac{\text{\# of samples with correct label}}{\text{\# of samples}} \tag{2.2}$$

$$\text{FDR} = \frac{\text{\# of faulty samples with fault label}}{\text{\# of faulty samples}} \tag{2.3}$$

$$\text{FAR} = \frac{\text{\# of normal samples with fault label}}{\text{\# of normal samples}} \tag{2.4}$$

Iannace [65] used ANNs for the identification of a fault in the blades of a drone flown indoors exploiting the noise produced by the blade rotation as input. The author extracted the important features from the noise through frequency analysis and the ANN had to distinguish between a balanced and 2 unbalanced blade scenarios.

Furthermore, Hussain [5] has developed a sensor failure detection, identification and accommodation (SFDIA) scheme for the aircraft roll, pitch and yaw sensors based on a fully connected cascade (FCC) NN architecture trained using the neuron by neuron (NBN) learning algorithm. This NN differs from the Multi-Layer Perceptron (MLP) in that it allows connections across layers, as can be observed in Figure 2.9 [5]; an architecture that reduces the number of required neurons. NBN is an improved method of the 2^{nd} order Levenberg-Marquadt algorithm [66, 67] and, in contrast with backpropagation, it can train fewer neurons faster and more efficiently [68].

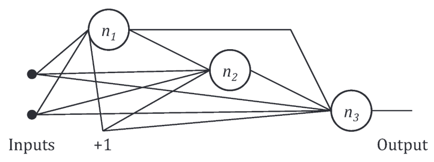


Figure 2.9: FCC NN architecture [5].

Aboutalebi [6] proposes a neural network adaptive structure (NNAS) to detect faults in the sensors of a quadrotor, whose equations of motion can be found in [69]. NNAS consists of 2 components: a nonlinear model-based observer and a NN whose weights are updated with an Extended Kalman Filter (EKF). The nonlinear observer computes the expected output and the NN approximates the change in behaviour due to the presence of a fault. As can be observed in Figure 2.10 [6], the NN is fed the error between the expected value from the nonlinear observer and the sensor observation as input, as well as previous outputs of the NN. The algorithm allows the user to tune two hyper-parameters that define the number of past time-steps of the observation-expectation error and NN outputs that are given as input to the NN. It was demonstrated that the proposed EKF technique for NN weight update helps to improve its learning rate.

In contrast with other ANN implementations that simply discern whether there is or not a fault (classifier), the NNAS exploits the ANN to approximate the deviation caused by a fault (function approximator). Similar application was found in [70], which uses a NN to estimate the sensor faults of a nonlinear hybrid system subjected to unmodelled actuator and sensor dynamics, as well as plant uncertainties. In the same manner, Talebi [71] uses a NN for fault detection and recovery of a reaction wheel actuator exploited in satellite attitude control systems, being able to reconfigure the controller with a fault. In both aforementioned works, the presence of a sensor fault can be detected when there is a spike in the NN output, since it is expected to produce zero output in non-faulty conditions. Also, the output can be used to classify the type of fault encountered.

Similarly, in [72] an ANN disturbance observer is developed for tackling the system uncertainty and nonlinear actuator (gain) faults of a 3 degree-of-freedom (DOF) model helicopter; together with another disturbance observer that estimates the external disturbance and the ANN approximation errors. A Radial Basis Function Neural Network (RBFNN) is implemented as observer, an ANN with radial basis functions as activation functions. The information from both observers is used by a backstepping based adaptive neural fault-tolerant controller in order to track the desired system output. Unfortunately, the fault tolerant controller was not applied to a 6 DOF model and can not counteract sensor faults.

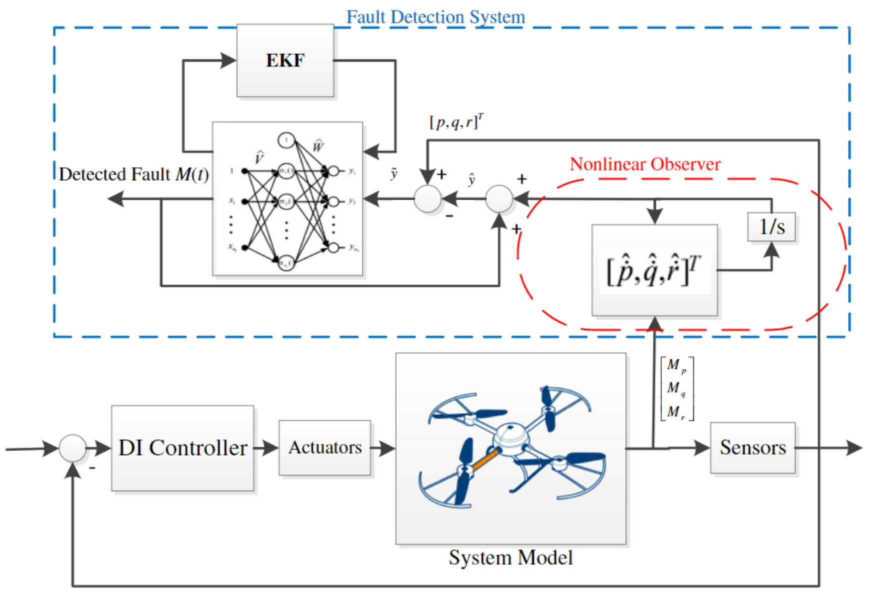


Figure 2.10: Block diagram of the NNAS fault detection implementation in a quadrotor [6].

In order to detect sensor faults in UAVs, Samy [73] proposes an extended minimum resource allocating radial basis function NN (EMRAN-RBFNN) for sensor fault detection, isolation and accommodation. It is an adaptive scheme because it adds hidden units only if they are necessary and removes those that do not contribute significantly to the output (pruning). Before adding a hidden unit, it verifies 3 conditions: the estimation error is below a pre-defined threshold, the RMS error of the previous estimations within a predefined time horizon are below a set threshold and the distance between the input vector and the centres of the hidden Gaussian RBF units is small enough. If any of the conditions is not met, another hidden unit is added. Additionally, the input layer can also be filtered by deleting those inputs that do not contribute to the error reduction. For that purpose, the error before and after the introduction of a parameter is compared. As a result, the NN structure and the execution time are reduced, which is beneficial due to the limited computational capability present in most UAVs.

In [7] an FDI strategy is proposed for cooperative robotic manipulators in which a NN is used to model the velocities of the system. The NN estimated and the manipulator observed velocities are subtracted to generate a residual which is fed to a RBFNN to classify the type of actuator fault given pre-defined fault criteria, as can be observed in Figure 2.11 [7]. The FDI scheme was applied in the presence of swinging joint faults and was able to detect and isolate them in trajectories that had not been presented before. Similar work and results were observed in [74].

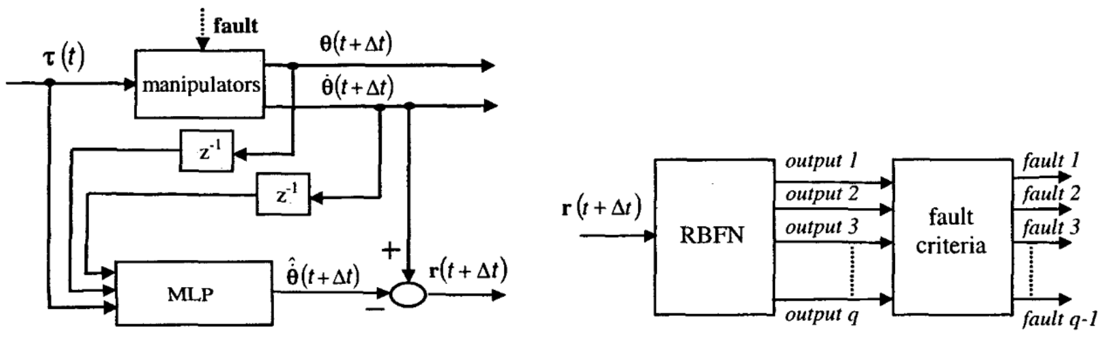


Figure 2.11: Robot manipulator FDI strategy [7].

2.1.1.1. Convolutional Neural Networks

Another well-known NN architecture which has shown great success in the analysis of images is the Convolutional Neural Network (CNN). Its role is to reduce images large pixel space into a dimension that is easier to process without losing information about important features. As can be observed in Figure 2.12 [8], CNNs are composed of four main types of layers: convolutional layers, nonlinearity layers, pooling layers (also known as sub-sampling layers [75]) and fully connected layers. The first type contains a set of filters and its duty is to perform a convolution operation between the input images and those filters in order to create feature maps. The nonlinearity layers usually follows or is considered to be part of the convolutional layer and performs a nonlinear mapping to its output. The pooling layer contributes to the reduction in the spatial representation and compresses the information. Finally, the fully connected layers are positioned at the end and they behave in the same manner as the hidden and output layers of a NN. The weights to be learnt by this algorithm are the filters of the convolutional layers. The reader can find more detailed information about the components, structure and applications of this architecture in [8, 10].

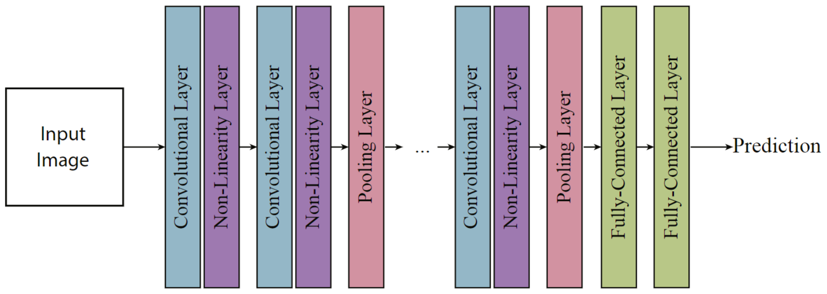


Figure 2.12: CNN general architecture [8].

An example that exploits this architecture can be found in the work by Janssens [76], which uses CNNs for fault detection in rotating machinery, more specifically in bearing fault detection. For that purpose, it feeds images of the Discrete Fourier Transformed (DFT) vibration signals (frequency spectrum) measured by the accelerometers to a shallow single convolutional layer CNN. At the end, the results of this form of feature learning show a better classification accuracy than those obtained through engineered features, which require expert knowledge.

Fault diagnosis in the aerospace sector has also seen the application of CNNs. Guo [9] proposes a hybrid feature model and deep learning framework capable of detecting faults from different sensors. As in the previous work, signals are first translated to the frequency domain, in this case short time Fourier transform (STFT) is applied to the state residuals obtained from the difference between the observations and the estimations from an EKF. Then, a CNN was used to extract the relevant features for fault diagnosis. An interesting feature from this implementation is the training of 9 independent CNNs, each corresponding to a different sensor signal, that are coupled at the end by a common fully connected layer which identifies the normal condition or faulty sensor. The complete pipeline can be observed in Figure 2.13 [9].

Instead of extracting features and then classifying the type of fault, there exist end-to-end learning approaches that aim at avoiding suboptimal hand-crafted features that require an additional computation cost. Ince [75] proposes an adaptive real-time fault detection and classification system that applies 1-D CNNs [77] directly to the raw current signals of a motor. The convolutional, nonlinearity and pooling layers are implicitly in charge of the feature extraction, whereas the full connected layers carry out the fault classification. It is considered to have an adaptive CNN topology because the last pooling layer reduces the information to one data point per neuron. As an example, if after the last convolution and nonlinearity layers each neuron has 8 data points, the last pooling layer will have a window of size 8 in order to downsize it to 1 data point per neuron before feeding it as input to the fully-connected layers. As a result, the CNN architecture allows any input layer dimension and any number of CNN layers before the fully-connected section.

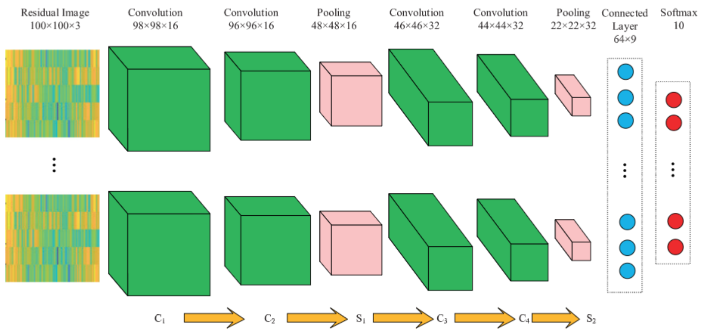


Figure 2.13: Hybrid feature model and deep learning pipeline for fault detection and classification [9].

2.1.1.2. Recurrent Neural Networks

2D CNNs are used for image data, whereas 1D CNNs can be exploited for the analysis of 1D signals. Another architecture that excels at 1D temporal data and can handle any input/output lengths are Recurrent Neural Networks (RNNs). In contrast with previous NNs architectures and instead of just considering a small number of neighbours around the input, as it is done by 1D CNNs, RNNs take into account the history of previous observations in order to exploit their sequential relationships when carrying out the computations with the current observation. Information about their dynamic correlations is stored or ”remembered” in the form of ”memory cells” over a span of time, forcing the model to take them into consideration in new computations [10, 11].

One of the key concepts of RNNs is the sharing of model parameters (weights) across a certain time span, allowing the model to process sequences of different length than presented during training. Besides that, it introduces the concept of cycles, which represent the influence of the present variable value on itself in future time steps. As can be observed in Figure 2.14 [10, 11], the intermediate hidden variable h at time t-i affects the corresponding h value at time t-i+1.

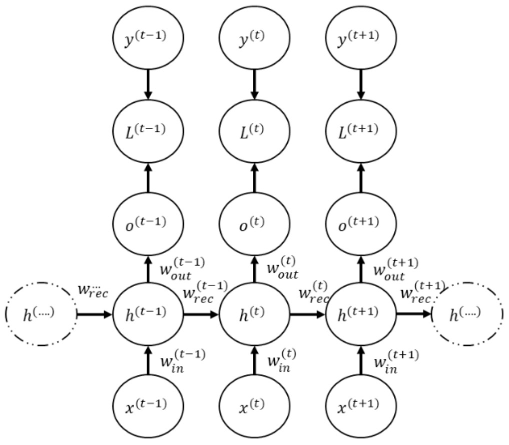


Figure 2.14: RNN representation in its folded (left) and time-unfolded (right) forms [10, 11].

The value of the hidden layer h_t is the result of a non-linear element-wise transformation $f(\cdot)$ of a linear combination of the input at the current time step x_t, the value of the same hidden layer at the previous time-step h_{t-1} and the bias b; whereas the output $\hat{y}_t$ involves another non-linear element-wise transformation $g(\cdot)$ of the linear combination of the value of the hidden layer at the current time step h_t and another bias c:

$$h_t = f\left(w_{\text{in}} x_t + w_{\text{rec}} h_{t-1} + b\right) \tag{2.5}$$

$$\hat{y}_t = g\left(w_{\text{out}} h_t + c\right) \tag{2.6}$$

There exist numerous variations in the RNN architecture depending on when an output is provided by the network or where are the recurrent connections (cycles) located. More detailed information about RNNs can be found in [10].

Two RNN architectures that have become very popular in recent years are Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). The LSTM is composed of the following 4 main parts [78]:

1. **Input gate**: determines what information provided in the current input and previous output should be stored. It is described as follows:

$$\Gamma_u = \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right), \tag{2.7}$$

   where σ is the sigmoidal activation function, W_u and U_u are the weights for the current input x_t and previous LSTM cell output h_{t-1}, and b_u is the bias of the input gate.

2. **Forget gate**: identifies the stored information from historical data that should be preserved and passed on in time. It is defined as follows:

$$\Gamma_f = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right), \tag{2.8}$$

   where all the parameters have the same meaning and serve the same purpose as in the input gate.

3. **Output gate**: its main task is to filter the memory to generate the output. It is defined as follows:

$$\Gamma_o = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right). \tag{2.9}$$

4. **Memory cell**: in charge of ”memorising” the important long term dependencies. For that purpose, it identifies the new information that should be stored, as well as the knowledge from the previous memory state that should be preserved. It is composed of the input and forget gates:

$$\tilde{c}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{2.10}$$

$$c_t = \Gamma_u \odot \tilde{c}_t + \Gamma_f \odot c_{t-1}, \tag{2.11}$$

   where $\odot$ defines the element-wise multiplication.

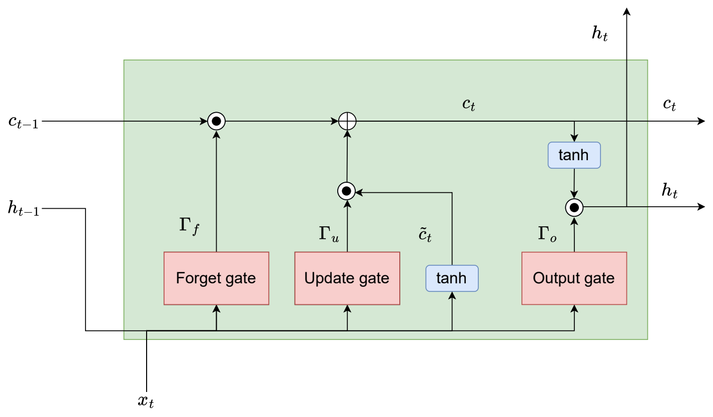
Having computed the output gate and the memory at the current time step, it is possible to compute the LSTM cell output as follows:

$$h_t = \Gamma_o \odot \tanh\left(c_t\right) \tag{2.12}$$

The aforementioned architecture of the LSTM can be observed in Figure 2.15 [11]. Their main advantages when compared to the vanilla RNN are their capability of storing useful information for a long time; they do not suffer from the long-term dependence problem [79].

When compared to the LSTM, the GRU is slightly simpler since it only contains two gates, namely the update and reset gates, and it works directly with the hidden content without using a memory unit. The update gate is in charge of determining what new data should be kept for the future and the reset gate determines how much of the previous information should be forgotten. Its analytical and graphical representation is very similar to the LSTM; therefore, the reader is encouraged to find more details about GRUs in [10, 11].

RNNs and their variations have seen an ample implementation in literature due to their ability to process sequential data. In the field of FDI, time series data from inputs, states or outputs are exploited. For instance, Talebi [80] uses RNNs as neural observers in order to detect and isolate actuator and sensor faults in nonlinear systems without relying on the presence of full state measurements. It



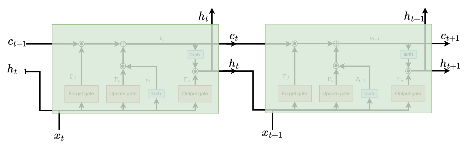


Figure 2.15: Unfolded LSTM cell architecture [11] (left) and how the LSTMs pass recurrent information through time (right).

adopts two RNNs for estimating the actuator and sensor fault vectors, respectively. When their outputs are nonzero and above a certain predefined threshold, a fault is detected and they are used to isolate the fault. The proposed implementation on a satellite attitude control subsystem (ACS) can not detect faults in actuators and sensors simultaneously but is capable of identifying multiple actuator or sensor faults at the same time. The stability of the complete FDI strategy is verified via the Lyapunov's direct method and the backpropagation algorithm is extended with a regularization term that adds extra damping and stability margin to the weight update.

Chen [79] implements LSTMs for the prediction of compressor failure in heavy duty trucks and compared the results to a Random Forest (RF) implementation, learning that LSTMs are more stable, robust and are able to capture temporal relationships better. Zhao [81] also takes advantage of LSTMs in order to extract dynamic information from data for simultaneous FDI within chemical processes (TEP). Here, batch normalization is used in order to reduce the internal covariate shift, situation where the training and test input features of the network present very different distributions, and accelerate the convergence of the algorithm. The results show superior performance when compared to other statistical approaches, such as DPCA + SVM or deep neural networks.

Furthermore, the performance of LSTMs and GRUs is compared in the work by González [11], who proposes a fault detection and classification algorithm for chemical processes in a large industrial plant, using the TEP as testing benchmark. The goal of the author is to increase the dectectability by introducing deep RNNs and the classification accuracy by developing a hierarchical structure and the implementation of a pseudo-random signal (PRS). Moreover, batch normalization is included in order to accelerate the training and avoid the covariate shift problem.

The introduction of a hierarchical structure aims at solving the problem of distinguishability, namely the issue that different faults have a similar effect on the measured variables. For that purpose, it creates a second classification model that focuses on a smaller group of faults which are difficult to classify, dedicating its whole learning power to find and augment their small differences. This is analogous to assigning preferential weights to the difficult to classify fault types. Normalizing the difficult to classify data separately increases its variance when compared to the normalization in which it is embedded within the data of the rest of the faults, improving fault classification. Besides that, the addition of external excitation through the introduction of PRS in the input data also improves the classification accuracy.

At the end, the proposed algorithm shows superior performance when compared to other statistical schemes such as Dynamic PCA (DPCA). Additionally, it was shown that LSTM performs slightly better than GRU in the TEP benchmark.

Finally, Rengasamy [82] provides an overview of deep learning approaches and architectures for actuator fault detection in the field of aircraft operations, specifically in aircraft maintenance, repair and overhaul (MRO). It concludes that the knowledge-based methods presented, such as CNNs or LSTMs, outperform model-based alternatives and that there is a lack of literature in the application of Bayesian Neural Networks and hybridisations of deep learning with fuzzy logic.

2.1.2. Bayesian classifier

The presented ANN and RNN approaches receive information from a single or multiple sensors, are able to detect whether a fault is present and, if that is the case, determine which type of fault within a pre-established group of classes. The problem is that it can not detect other types of faults than the ones inferred during training and there are no probability estimates provided of how uncertain is the algorithm about its decision. Besides that, when multiple processes can cause the same fault, the underlying origin can not be identified [11].

Even though some classifiers can provide a degree of confidence (e.g. those that include a soft-max activation function in the final layer) and multiple regularisation techniques have been developed [83] (e.g. weight decay, early stopping and dropout), the modern NNs tend to show overconfidence or underconfidence (Figure 2.16 [12]). The degree to which the confidence matches the accuracy of a prediction is called the calibration and there is a line of research that aims at studying recalibration techniques and calibration measures in order to improve the confidence estimation of classifiers. Although the Expected Calibration Error (ECE) and the Maximum Calibration Error (MCE) are the most common calibration measures in which the confidence space is evenly divided in bins [12, 84], such as the one shown in Figure 2.16 [12], alternatives have been proposed with adaptive binning schemes, such as the Adaptive Calibration Error (ACE) [85]. Besides that, while ECE and MCE use only the predicted class probability (the class with the highest probability), ACE aims at calibrating the error across all the classes. Sources of poor calibration are the hyper-parameters used to build the learning architecture, such as the number layers, the number of neurons, the regularisation technique or the batch normalisation [12, 86].

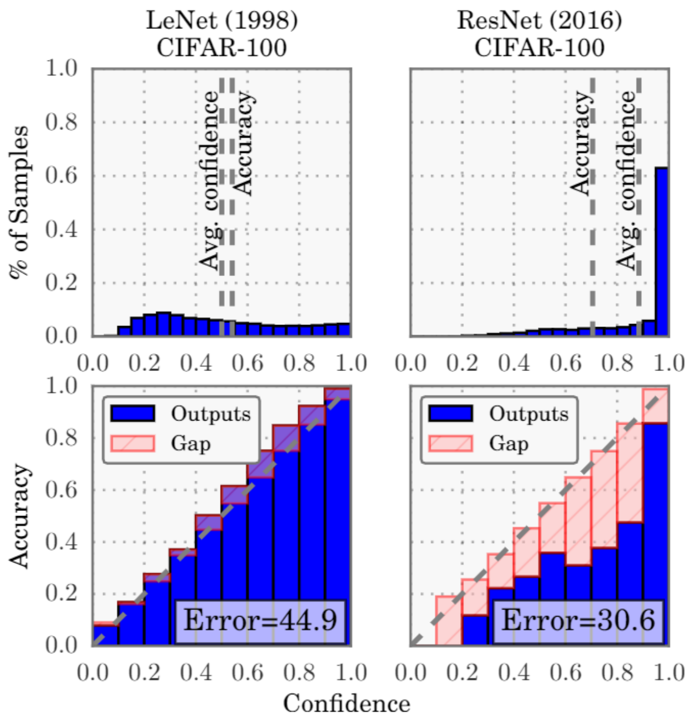


Figure 2.16: Confidence histograms (top) and reliability diagrams (bottom) for early (left) and modern (right) neural networks [12].

2.1.2.1. Bayesian inference framework

To tackle these problems in conventional ANNs and RNNs, the Bayesian inference framework could be used for training these machine learning architectures. According to the Bayesian paradigm, ANNs and RNNs can be treated as models with parameters (θ) which are considered random variables. These parameters are assigned a probability distribution which describes how well do they fit the data and characterises the uncertainty in our current knowledge of θ. In contrast with the frequentist paradigm, it is possible to include prior knowledge and incorporate new experiences/data when quantifying the ”belief” or probability of a parameter. In the Bayesian analysis there are three main components [87]:

1. **Prior distribution**: encode prior knowledge of the parameters before using the data. Usually a flexible probability distribution which is easy to work with is chosen. In order to choose prior beliefs, solid prior information from scientific knowledge, previous studies or pilot data is used. Sensitivity analysis can be used as a tool for choosing a prior by analysing their impact on the results and conclusions. In the case that no prior knowledge of the parameters is available, non-informative priors can be used.

2. **Likelihood**: encodes the information found in the data about the parameters and measures how well does the model fit the data.

3. **Posterior distribution**: the update of the beliefs regarding θ resulting from the combination of the prior distribution and the likelihood; it is a compromise between both sources of information. It updates the uncertainty quantification and it can be exploited for estimation and inference about the parameters.

These concepts come together in Bayes' theorem. If we consider two events (A and B) and we assume that the independent probability of A and B is greater than 0, namely P(A)>0 and P(B)>0, then their conditional probabilities can be expressed as follows:

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \tag{2.13}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2.14}$$

Combining both expressions, the simplest form of the Bayes's theorem is obtained, providing the formula for obtaining the posterior distribution:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}, \tag{2.15}$$

where P(A) is the prior distribution, P(A|B) is the likelihood and P(B|A) is the posterior distribution. If instead of the events A and B, the parameters of a model and the data are considered, the Bayes' theorem can be translated to:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} = \frac{P(D|\theta)P(\theta)}{\int_\theta P(D|\theta')P(\theta')d\theta'} \propto P(D|\theta)P(\theta) \tag{2.16}$$

In contrast with the Maximum Likelihood Estimate (MLE) which assumes that the data is the only source of uncertainty, Bayesian learning also considers the uncertainty in the parameters θ thanks to the prior (P(θ)); encoder of subjective initial beliefs. As can be observed, the denominator is not a function of θ, so it can be considered as a normalising constant which ensures that the integrated posterior distribution equals 1. In the case the posterior (predictive) distribution is required for a complex model, the marginalization over the parameters, the integral in the denominator in Equation 2.16, is computationally infeasible due to the high-dimensional parameter space. In scenarios where different models' posterior distributions need to be compared, this information prevents the unnecessary calculation of the integral in the denominator.

With the posterior distribution, it is possible to compute the posterior predictive distribution; the prediction of a future observation. Given a dataset D, a model trained on it θ and a new data point x, the corresponding y value can be computed as follows with the posterior probability:

$$P(y|x,D) = \int_{\theta} P(y|x,\theta')P(\theta'|D)d\theta' = \mathbb{E}_{P(\theta|D)}\left[P(y|x,\theta)\right] \tag{2.17}$$

It can be considered as the average of the product between the probability of each model θ given the data and the probability of each output given each model and the new data point. Unfortunately, the computation of the posterior predictive distribution for a complex model is infeasible due to the high-dimensional parameter space. In the case of a neural network, since the integral is taken about the parameters, it would be equivalent to using an infinite number of neural networks with the same architecture and different weight values, which is intractable for any NN of considerable size. Due to this limitation, researchers have found alternatives in order to achieve an approximation to the inference problem. There exist two groups of approximate algorithms:

1. **Sampling methods**: generate answers by repeatedly generating random numbers from a distribution of interest. They can be used for performing marginal and maximum a posteriori (MAP) probability inference queries, as well as computing other interesting characteristics of the distribution. Most algorithms are based on Markov Chain Monte-Carlo (MCMC) [88], such as Gibbs sampling [89], differential evolution (DE) [90] and Metropolis-Hastings [89]. Monte-Carlo (MC) methods are algorithms capable of building solutions from many samples obtained from a certain distribution. As an example, the expectation presented in Equation 2.18 would be computed as follows:

$$\mathbb{E}_{P(\theta|D)}\left[P(y|x,\theta)\right] \approx I_T = \frac{1}{T}\sum_{t=1}^{T} P(y|x,\theta^t), \tag{2.18}$$

where θ^1, ..., θ^t are samples drawn from P(θ|D) and I_T is an unbiased estimator of $\mathbb{E}_{P(\theta|D)}\left[P(y|x,\theta)\right]$.

2. **Variational methods**: consider inference as an optimization problem. These alternatives emerged due two main disadvantages in sampling methods. First, even though sampling methods are guaranteed to find the global optimum when time tends to infinity, it is very difficult to see when a ”good-enough” solution has been achieved. Second, the speed of convergence highly depends on the sampling technique choice, which in itself is an art.

   Variational methods will try to find a tractable distribution depending on some hyper-parameters θ, $q(\mathbf{w}|\theta) \in Q$, that is as close as possible to the intractable probability distribution $P(\mathbf{w}|$D) [1]. Then, all the required computations will be done with $q(\mathbf{w}|\theta)$ in order to obtain an approximate solution. For that purpose, a tractable distribution $q(\mathbf{w}|\theta)$ needs to be chosen, which is usually parametrised by a model (e.g. NN, Gaussian processes or latent variable models), as well as an optimisation objective J($q(\mathbf{w}|\theta)$) which minimises the differences of $q(\mathbf{w}|\theta)$ with $P(\mathbf{w}|$D). A common choice used in literature to compute the degree of similarity between $q(\mathbf{w}|\theta)$ and $P(\mathbf{w}|$D) is the Kullback-Leibler (KL) divergence [91]:

$$KL(q(\mathbf{w}|\theta)||P(\mathbf{w}|D)) = \int_x q(\mathbf{w}|\theta)\log\frac{q(\mathbf{w}|\theta)}{P(\mathbf{w}|D)} \tag{2.19}$$

   The KL has a value higher or equal to zero for all $P(\mathbf{w}|$D)-$q(\mathbf{w}|\theta)$ combinations and it is zero if and only if $P(\mathbf{w}|$D) and $q(\mathbf{w}|\theta)$ are equal. Since it tends to infinity when $P(\mathbf{w}|$D)=0 and $q(\mathbf{w}|\theta)$>0, KL tends to force $q(\mathbf{w}|\theta)$ to zero (zero-forcing). Given that the posterior distribution within the denominator of the integral is proportional to the likelihood times the prior (Equation 2.16), it can be simplified to the following expression:

$$KL(q(\mathbf{w}|\theta)||P(\mathbf{w}|D)) = \int_x q(\mathbf{w}|\theta)\log\frac{q(\mathbf{w}|\theta)}{P(D|\mathbf{w})P(\mathbf{w})} = \int_x q(\mathbf{w}|\theta)\log\frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})} - \mathbb{E}_{q(\mathbf{w})}\left[\log P(D|\mathbf{w})\right] = \\ KL(q(\mathbf{w}|\theta)||P(\mathbf{w})) - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(D|\mathbf{w})\right] \tag{2.20}$$

[1]Derivation to minimise the difference between $q(\mathbf{w}|\theta)$ and $P(\mathbf{w}|$D): `https://youtu.be/uaaqyVS9-rM?t=19m42s`

The minimization of the KL divergence is achieved by maximizing the variational lower bound or evidence lower bound (ELBO) $\mathcal{L}(q(\mathbf{w}|\theta))$ of the marginal likelihood of the data [92]:

$$\mathcal{L}(D,\theta) = -KL(q(\mathbf{w}|\theta)||P(\mathbf{w})) + L_D(\theta) = -KL(q(\mathbf{w}|\theta)||P(\mathbf{w})) + \sum_{(x,y)\in D} \mathbb{E}_{q(\mathbf{w}|\theta)}\,[\log P(y|x,\mathbf{w})], \tag{2.21}$$

where $L_D(\theta)$ is the expected log-likelihood. The sum of $\mathcal{L}$(D, θ) and $KL(q(\mathbf{w}|\theta)||P(\mathbf{w}|D))$ equals the (conditional) marginal log-likelihood:

$$\mathcal{L}(D,\theta) + KL(q(\mathbf{w}|\theta)||P(\mathbf{w}|D)) = \sum_{(x,y)\in D} \log P(y|x) \tag{2.22}$$

Since the sum of both terms is constant with respect to changes of the tractable distribution q($\mathbf{w}$), the maximization of the ELBO leads to the minimization of the Kullback-Leibler divergence.

In contrast with sampling methods, the probability variational methods of finding a global optimum is very low, it is always possible to know when they have converged (sometimes with bound on the accuracy), and they can easily be scaled.

Sampling methods are the group that has prominently been used in literature but variational methods have gained a lot of interest in the last 15 years, showing superior performance [87].

2.1.2.2. Bayesian Neural Networks

Neural Networks can be considered probabilistic models P(y|x, $\mathbf{w}$) that are capable of predicting an output given an input and the model weights. In the frequentist (conventional) approach, the goal is to update the weights such that they are able to best explain the data by maximizing the likelihood P(D|$\mathbf{w}$), through the Maximum Likelihood Estimate (MLE):

$$\mathbf{w}^{\text{MLE}} = \arg\max_{\mathbf{w}} P(D|\mathbf{w}) = \arg\max_{\mathbf{w}} \prod_i P(y_i|x_i,\mathbf{w}) \tag{2.23}$$

In contrast, the Bayesian approach aims at maximizing the posterior P($\mathbf{w}$|D) with Maximum a Posteriori (MAP) learning, a method which views the weights as random variables and the data as being fixed. The result is the same as MLE with the introduction of a regularization term in the form of the weights prior distribution:

$$\mathbf{w}^{\text{MAP}} = \arg\max_{\mathbf{w}} P(\mathbf{w}|D) = \arg\max_{\mathbf{w}} P(D|\mathbf{w})P(\mathbf{w}) = \arg\max_{\mathbf{w}} \,(\log P(D|\mathbf{w}) + \log P(\mathbf{w})) \tag{2.24}$$

Bayesian Neural Networks (BNN) are the result of applying the Bayesian framework to Stochastic Neural Networks. Stochastic Neural Networks are ANN with stochastic transfer functions and/or weights where a distribution is learned instead of point estimates (Figure 2.17 [13]). They are defined by the variables that behave stochastically in the NN and their a priori distributions. The randomness introduced by this parameters brings exploration and help this NN architectures escape local minima [13]. However, as training progresses, the uncertainty is decreased and the environment is better understood, leading to more deterministic NN decisions (exploitation). As a result, BNNs are considered a great tool for knowledge representation and reasoning in the presence of uncertainties.

As can be seen in Equation 2.24, the Bayesian inference framework aims at calculating the posterior distribution of the weights with the provided training data in order to make predictions (Equation 2.17). In other words, with the observed data, it updates the initial beliefs of the parameters encapsulated in the prior with an updated belief in the form of the posterior distribution. Besides that, the uncertainty introduced in the model shows the regions with little or no data that require attention and further training, leading to an improvement in the predictive performance.

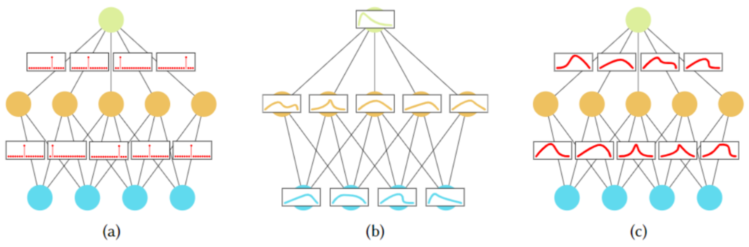


Figure 2.17: Visual comparison between a conventional ANN and a stochastic NN [13]: a) corresponds to an ANN in which point estimates are computed instead of distributions, b) corresponds to a Stochastic NN in which a probability distribution over the transfer functions is learnt, and c) corresponds to a Stochastic NN in which a probability distribution is learnt over the weights of the NN.

In BNN, the possibility that with every run some parameters acquire a different value can be regarded as simulating multiple different models, each with a different probability of appearance depending on the probability distributions of the stochastic parameters. As a result, instead of having a single model, it can be considered that multiple models are trained and their results are aggregated to solve a particular problem. This phenomenon is called ensemble learning [93], which refers to the combination of classifiers that correct each other errors and whose success relies on their diversity; if all classifiers would always vote the same option, they would not be filling the gaps of each other weaknesses. Traditionally, ensemble learning has been achieved using different classifiers, using different training parameters for the model or different datasets with bootstrapping or bagging. In this case, the first two options are achieved by BNNs thanks to the stochasticity of its weights and activation functions. The result is an ANN with confidence estimation, a classifier that estimates how sure it is about its own output, and learnt representations which are robust to perturbations in the weights and show the variability present in the training data. In contrast with traditional ensemble learning methods, BNN only doubles the number of parameters.

In most cases, the weights are represented by Gaussian distributions, whose mean and standard deviation are the unknown parameters ($\theta = (\mu, \sigma^2)$) that shall be obtained during training [92]. However, in order to prevent obtaining a negative value for the standard deviation, it is parameterised as:

$$\sigma = \log(1 + e^{\rho}), \tag{2.25}$$

leading to the new unknown parameters $\theta = (\mu, \rho)$. A Gaussian distribution is chosen due to its simplicity, flexibility and convenience. During the forward propagation in a BNN with Gaussian distributions, a sample is taken from a standard Gaussian distribution ($\epsilon = \mathcal{N}(0,1)$) for each weight i and their values are computed as follows [92]:

$$w_i = \mu_i + \sigma_i \cdot \epsilon_i \tag{2.26}$$

Alternatively, a reparameterization trick [92] can be applied such that a sample from a standard Gaussian distribution (ϵ) only has to be computed once per neuron and not per weight. The impact of this trick can be easily observed with an example. In a NN with 1000 input neurons and 1000 hidden neurons, the weight matrix that connects both layers has a size of 1000×1000. Instead of computing ϵ 10^6 times, which is the number of weights, it is sampled only 10^3 times, which is equivalent to the number of hidden neurons. Additionally, the local reparameterization trick leads to an estimator that has a lower variance.

With the weights, the BNN behaves the same as an ANN during forward propagation. The next component for the BNN to work is the loss function. Bayes by Backprop[2] [94] provides a derivation from Equation 2.20 for the objective function, which requires Monte Carlo sampling (N samples):

[2]BNN with Bayes by Backprop Python implementation applied to FMNIST dataset: `https://www.nitarshan.com/bayes-by-backprop/`

$$J(D,\theta) = \sum_{j=1}^{N} \log q(\mathbf{w}^{(j)}|\theta) - \log P(\mathbf{w}^{(j)}) - \log P(D|\mathbf{w}^{(j)}), \tag{2.27}$$

where $\mathbf{w}^{(j)}$ corresponds to the jth Monte Carlo sample obtained from the variational posterior q($\mathbf{w}^{(j)}|\theta$). The first two components within the sum correspond to the complexity cost whereas the last component denotes the likelihood cost which is influenced by the data. In the case that minibatches are used instead of fully batched gradient descent or fully stochastic gradient descent, Graves [95] proposes to minimise the minibatch cost for minibatch k = 1,2,...,M:

$$J_k(D_k,\theta) = \frac{1}{M} KL\left[q(\mathbf{w}|\theta)||P(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(D_k|\mathbf{w})\right] \tag{2.28}$$

$$\sum_k J_k(D_k,\theta) = J(D,\theta) \tag{2.29}$$

Alternatively, the complexity cost can be weighted non-uniformly across the different minibatches:

$$J_k(D_k,\theta) = \pi_k KL\left[q(\mathbf{w}|\theta)||P(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(D_k|\mathbf{w})\right], \tag{2.30}$$

where $\pi \in [0,1]^M$ and $\sum_{k=1}^{M} \pi_k = 1$. Blundell [94] found $\pi_k = \frac{2^{M-k}}{2^M-1}$ to work well: the cost is heavily influenced at the beginning by the complexity cost while the data acquires higher importance with later minibatches. This scheme can have great impact during training since at the beginning the impact of the data is small whereas, with more experienced minibatches, the data becomes more influential than the prior.

Furthermore, in order to compute the prior of the weights $P(\mathbf{w})$, Blundell [94] proposes a scale mixture prior which is a scaled mixture of two zero mean gaussians with different variances and is not modified in training:

$$P(\mathbf{w}) = \prod \pi\mathcal{N}(\mathbf{w}_j|0,\, \sigma_1^2) + (1-\pi)\mathcal{N}(\mathbf{w}_j|0,\, \sigma_2^2) \tag{2.31}$$

$$\log P(\mathbf{w}) = \sum_j \log\left(\pi\mathcal{N}\left(\mathbf{w}_i|0,\, \sigma_1^2\right) + (1-\pi)\mathcal{N}\left(\mathbf{w}_i|0,\, \sigma_2^2\right)\right), \tag{2.32}$$

where $\mathbf{w}_i$ is the ith weight of the network and σ_1^2 and σ_2^2 are the variances of the mixture components. Usually, the first component is given a larger variance than the second ($\sigma_1 > \sigma_2$) and the second component has a small variance ($\sigma_2 \ll 1$). The result is a spike-and-slab prior with a heavy tail and most of the weights concentrated a priori around 0. It is also possible to use a previously learnt posterior as the prior when new data is acquired, making BNNs suitable for online learning [96].

Once the loss function has been computed, backpropagation is used in order to update the weights. In contrast with the conventional ANN where the gradient of the loss function with respect to the weights and biases was enough ($\frac{\delta J(D,\theta)}{\delta \mathbf{w}}$), here an additional step needs to be taken for backward propagation:

$$\Delta\mu = \frac{\delta J(D,\theta)}{\delta \mathbf{w}}\frac{\delta \mathbf{w}}{\delta \mu} + \frac{\delta J(D,\theta)}{\delta \mu} = \frac{\delta J(D,\theta)}{\delta \mathbf{w}} + \frac{\delta J(D,\theta)}{\delta \mu} \tag{2.33}$$

$$\mu \leftarrow \mu - \alpha\Delta\mu \tag{2.34}$$

$$\Delta\rho = \frac{\delta J(D,\theta)}{\delta \mathbf{w}}\frac{\delta \mathbf{w}}{\delta \rho} + \frac{\delta J(D,\theta)}{\delta \rho} = \frac{\delta J(D,\theta)}{\delta \mathbf{w}}\frac{\epsilon}{1+e^{-\rho}} + \frac{\delta J(D,\theta)}{\delta \rho} \tag{2.35}$$

$$\rho \leftarrow \rho - \alpha\Delta\rho \tag{2.36}$$

Once the BNN has been trained, in the testing phase it is possible to carry out a prediction, as well as computing its uncertainty. In the case of a regression problem, the prediction is obtained from the output average of different models obtained from sampling different weights $\mathbf{w}_j \in \mathbf{W}$ from the converged hyper-parameters θ [97]:

$$\hat{y} = \frac{1}{N} \sum_{\mathbf{w}_j \in \mathbf{W}} NN_{\mathbf{w}_j}(x) \tag{2.37}$$

This process is known as ensembling and the uncertainty can be quantified as follows:

$$\Sigma_{y|x,D} = \frac{1}{N-1} \sum_{\mathbf{w}_j \in \mathbf{W}} \left(NN_{\mathbf{w}_j}(x) - \hat{y}\right)\left(NN_{\mathbf{w}_j}(x) - \hat{y}\right)^T \tag{2.38}$$

In the case that classification is performed, the average model prediction of each class probability can be considered as a measure of uncertainty and the prediction is computed as follows:

$$\hat{p} = \frac{1}{N} \sum_{\mathbf{w}_j \in \mathbf{W}} NN_{\mathbf{w}_j}(x) \tag{2.39}$$

$$\hat{y} = \arg\max_{l} p_l \in \hat{p} \tag{2.40}$$

Figure 2.18 shows a summary of the BNN workflow with the design, training and testing stages [13]. In the training section, apart from classifying the algorithms in the sampling and variational methods, it also distinguishing between those which are specialised in deep learning and those that are more generic.

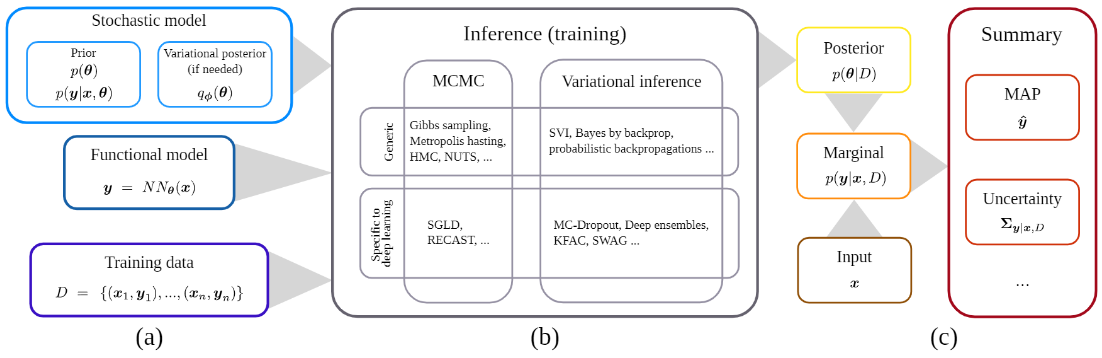

Stochastic model
Prior
$p(\boldsymbol{\theta})$
$p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$
Variational posterior (if needed)
$q_\phi(\boldsymbol{\theta})$
Functional model
$\boldsymbol{y} = NN_{\boldsymbol{\theta}}(\boldsymbol{x})$
Training data
$D = \{(\boldsymbol{x}_1,\boldsymbol{y}_1),...,(\boldsymbol{x}_n,\boldsymbol{y}_n)\}$
(a)
Inference (training)
MCMC
Variational inference
Generic
Gibbs sampling, Metropolis hasting, HMC, NUTS, ...
SVI, Bayes by backprop, probabilistic backpropagations ...
Specific to deep learning
SGLD, RECAST, ...
MC-Dropout, Deep ensembles, KFAC, SWAG ...
(b)
Posterior
$p(\boldsymbol{\theta}|D)$
Marginal
$p(\boldsymbol{y}|\boldsymbol{x},D)$
Input
$\boldsymbol{x}$
Summary
MAP
$\hat{\boldsymbol{y}}$
Uncertainty
$\Sigma_{\boldsymbol{y}|\boldsymbol{x},D}$
...
(c)

Figure 2.18: BNN workflow that includes the design, train and test stages [13].

Moreover, Figure 2.19 [14] shows the relation between the general framework for fault detection and the Bayesian approach. Conventionally, a model is built to characterise the Normal Operating Conditions (NOC) of the system such that its predictions are compared with the real observations, their deviation is measured and a decision is made about the presence of a fault. In the Bayesian approach, a Bayesian model of the system is developed and the deviation level from the NOC region is assessed with the probability estimate of the observations with respect to the inferred posterior distribution. Finally, the fault is detected when the aforementioned probability exceeds a certain threshold which has been designed such that the number of false alarms does not exceed a required limit.

When compared to the conventional point estimates ANNs, the BNN implementation allows the visualisation of uncertainty in the parameters and has shown better quantification of the uncertainty in the classification task (calibration), meaning that the estimated confidence is more congruent with the errors observed and there are less scenarios of over- or underconfidence [84, 98, 99]. For instance, Kristiadi [99] demonstrates that the overconfidence problem in ReLU classification networks [100] with data points far away from the training dataset, out-of-distribution (OOD) examples, can be solved by having a network that is ”a bit Bayesian”, meaning that last-layer Bayesian approximation is sufficient condition for achieving calibrated uncertainty. Additionally, it was shown that Bayesian Deep Neural Networks show less sensitivity to out of sample instances caused by noise present in the data and/or in the parameters (higher reliability) [84].

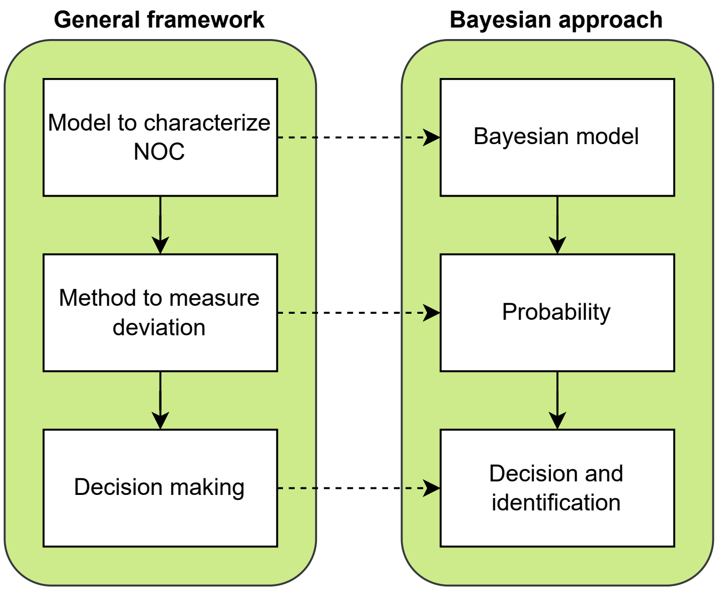

General framework
Bayesian approach
Model to characterize NOC
Method to measure deviation
Decision making
Bayesian model
Probability
Decision and identification

Figure 2.19: Bayesian fault detection compared to the general framework [14].

Furthermore, the use of Bayesian Neural Networks allows to establish a distinction between 2 types of uncertainty: epistemic and aleatoric uncertainty. The first one refers to the uncertainty present due to the lack of knowledge, whose effects can be dampened with the use of more data, whereas the second is caused by the (partial) random nature of a phenomenon and can not be solved by the introduction of more data [101, 102]. Understanding where in the high dimensional space of data there is epistemic uncertainty enables the system to decide what new data points should be labelled in order to reduce it. As a result, BNNs are a very data efficient active learning framework [103, 104] which can learn from a small dataset preventing overfitting.

2.1.2.3. Bayesian Recurrent Neural Networks

An advanced approach is based on using Bayesian Recurrent Neural Networks (BRNN) [105] with variational dropout, that was applied to the fault detection in chemical processes using the benchmark Tennessee Eastman process (TEP) [14]. In order to understand how fault detection and classification was performed, it is necessary to understand how the Bayesian framework is applied to RNNs, as well as the implications of variational dropout.

BRNN, as with Bayesian Neural Networks, aim at providing probabilistic distributions over the outputs by combining statistical modelling (the Bayesian framework) and RNN. As previously explained, the output of an RNN is a linear combination of the current state (s$_t$) and output bias (b$_y$), whereas the current state is computed by applying a nonlinear function to a linear combination of the current input (x$_t$), the previous state (s$_{t-1}$) and the state bias (b$_s$). In the case of a BRNN, the model parameters θ = (W$_s$, W$_y$, U$_s$, b$_s$ and b$_y$) are considered random variables from the prior distribution. Fortunato [105] modifies Equation 2.21 in order to account for the change in architecture introduced by the RNN. Given that the number of mini-batches is B and the number of truncated sequences in a mini-batch is C, then the ELBO is computed as follows:

$$\mathcal{L}(D,\theta) = -KL(q(\mathbf{w}|\theta)\|P(\mathbf{w})) + \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log\prod_{b=1}^{B}\prod_{c=1}^{C} p\left(y^{(b,c)}|\theta, x^{(b,c)}\right)\right]. \tag{2.41}$$

Then, the ELBO of mini-batch b and truncated sequence c can be expressed as:

$$\mathcal{L}_{(b,c)}(D,\theta) = -w_{KL}^{(b,c)} KL(q(\mathbf{w}|\theta)\|P(\mathbf{w})) + \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log p\left(y^{(b,c)}|\theta, x^{(b,c)}, s_{\text{prev}}^{(b,c)}\right)\right], \tag{2.42}$$

where $w_{KL}^{(b,c)}$ is the responsibility of the KL cost for each of the sequences of each minibatch and $s_{\text{prev}}^{(b,c)}$ refers to the initial state of minibatch x$^{(b,c)}$. In order to divide the penalty equally among all the b-c combinations, then $w_{KL}^{(b,c)} = \frac{1}{BC}$. In the case that subsequent minibatches follow the same order as the time series data, then $s_{\text{prev}}^{(b,c)}$ is set to the last state of the RNN for x$^{(b,c-1)}$ (the last state of the previous sequence). Once the ELBO of a complete minibatch is computed, the weights' parameters are updated and new weights are sampled for the next minibatch.

Sun [14] proposes the use of a normal distribution for the likelihood distribution, zero-mean Gaussian priors and variational dropout for approximating the BRNN inference. Apart from variational dropout [106] and Bayes by Backprop [94, 105, 107], alternative approximate inference methods for BRNNs are multiplicative normalizing flows [108] and probabilistic backpropagation [109]; however, Sun [14] chose for variational dropout due to is simplicity and generalization capability.

Unlike other approximation methods, variational dropout does not require modifications in the training method or in the BRNN architecture. As other variational methods, it starts by proposing a variation parameterized distribution $q(\mathbf{w}|\theta)$, which in this case it is made of two Gaussian distributions with small variances:

$$q(\mathbf{w}|\theta) = p\mathcal{N}\left(\mathbf{w}|0, \sigma^2\mathbf{I}\right) + (1-p)\mathcal{N}\left(\mathbf{w}|\mathbf{m}, \sigma^2\mathbf{I}\right), \tag{2.43}$$

where p is the dropout probability and the precision parameter (σ^2) together with the variational parameter ($\mathbf{m}$) shape $\theta = (\sigma^2, \mathbf{m})$. Once the tractable function is defined, as with a BNN, the goal of the BRNN training process is to minimise the KL divergence $KL\left[q(\mathbf{w}|\theta)\|P(\mathbf{w}|D)\right]$. Gal [106] demonstrates that the loss function optimisation with dropout is equivalent to the KL divergence minimisation. However, applying the conventional dropout to the unrolled BRNN, meaning that at every time step the probability of dropping out a weight is assessed, would lead to model instabilities and the model incapability to learn a sequence. To solve this issue, Gal [110] proposes to keep fixed the dropout mask between time steps for a complete sequence; a difference with respect to the general dropout approach which can be observed in Figure 2.20 [14]. It has been shown that this novel approach also serves as regularization technique to prevent overfitting [106].

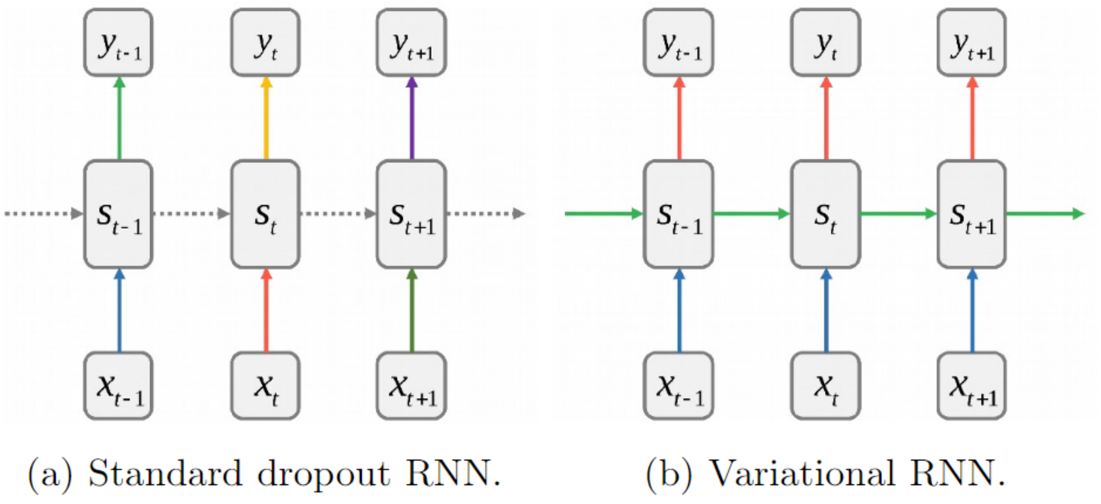


Figure 2.20: Comparison between standard and variational dropout applied to RNN [14]. Both graphs show the RNN unfolding in time with the inputs at the bottom, the state units in the middle and the outputs at the top. The vertical lines are the input-output connections while the horizontal lines are recurrent connections. The dotted lines represent those connections without dropout being applied while each of the colours in the solid lines represent connections with different dropout masks.

In the case that variational dropout is used during the testing phase, multiple outputs can be obtained with the same input by running forward the model multiple times. With the collected samples, it is possible to define a probability distribution and obtain the posterior predictive statistics, such as the mean, standard deviation and covariance. Additionally, since they are obtained from independent forward model passes, multiple realizations can be executed concurrently and these statistics could be computed online.

Having understood the BRNN with variational dropout, Sun [14] exploits the scheme presented in Figure 2.19 for fault detection. First, it builds a model to reproduce the dynamics in time with historical NOC data. With the current observation (x_t) and the past experienced context represented by the current state (s_t), the model is able to predict the observation at the next time step ($\hat{x}_{t+1}$). Since variational dropout is being applied concurrently N times, the predicted samples ($\{\hat{x}_{t+1}(i)\}_{i=1,...,N}$) are used to approximate the posterior predictive distribution of the next time step. Then, when the next time step observation is received (x_{t+1}), it is compared with the generated distribution and a fault is detected when the observation is significantly deviated from the posterior predictive distribution.

In order to assess the deviation magnitude of the observation at the next time step and the posterior predictive distribution, Sun [14] proposes two methods: the squared Mahalanobis distance for Gaussian predictive distributions and the local density ratio for non-Gaussian predictive distributions. The first method is limited to the implementation of a Gaussian predictive posterior distribution and the deviation of the observation from the NOC model is parameterized by the squared Mahalanobis distance:

$$M^2 = (x_t - \mu_t)^T \, \sigma_t^{-1} \, (x_t - \mu_t) \, , \tag{2.44}$$

where μ_t and σ_t are the mean and the covariance of the predictive posterior distribution. The higher the squared Mahalanobis distance, the higher the likelihood that the next observation corresponds to a fault. The threshold (M^2_{th}) above which a data point is called as a fault ($M^2 > M^2_{\text{th}}$) is the (1-α) percentile of the M^2 computed with a validation dataset.

The second method uses the local density ratio (LDR) to detect whether an observation is considered an outlier. The LDR computes an estimate of the density around the observation using its k-nearest-neighbors (k-NNs) set $\mathcal{N}_k(x)$ from the N variational dropout samples used to define the predictive posterior distribution. For that purpose, it computes the k-NN local density estimate $\hat{f}(x)$ as follows:

$$\hat{f}(x) = \frac{k}{\sum_{p \in \mathcal{N}_k(x)} d(p, x)}, \tag{2.45}$$

where $\mathcal{N}_k(x)$ is the set of k-NNs of x and $d(p,x)$ is the Euclidean distance between x and a point $p \in \mathcal{N}_k(x)$. The higher the local density estimate, the closer is the observation to its k-NNs. Then, the LDR can be computed as the ratio of the average local density estimates of the k-NNs and the local density estimate of the observation x_t:

$$LDR(x_t) = \frac{\frac{1}{k} \sum_{p \in \mathcal{N}_k(x_t)} \hat{f}(p)}{\hat{f}(x_t)} \tag{2.46}$$

The higher the LDR(x), the further away is the observation from the points of the prediction posterior distribution, indicating a higher possibility of a fault.

In this second method, the choice of the number of k-NNs is crucial since a high number would lead to detection insensitivity whereas a low number causes instability. Besides that, the tuning of the threshold for the LDR fault detection is carried out in the same manner as the Mahalanobis distance.

Finally, fault identification consists of discovering the observation variables that have caused the fault. The BRNN with variational dropout proposed in [14] applies the discussed fault detection scheme to each variable. As a result, it is possible to immediately discern which variables are behaving abnormally by highlighting those whose deviation criterion exceeds the pre-computed per variable threshold. Additionally, fault identification plots can show the deviation of each variable throughout time for the analysis of the fault propagation path throughout the system. The fault identification methodology is very similar to the fault detection, so the reader can find the detailed description in [14]. A general overview of the complete fault detection and identification process can be observed in Figure 2.21 [14].

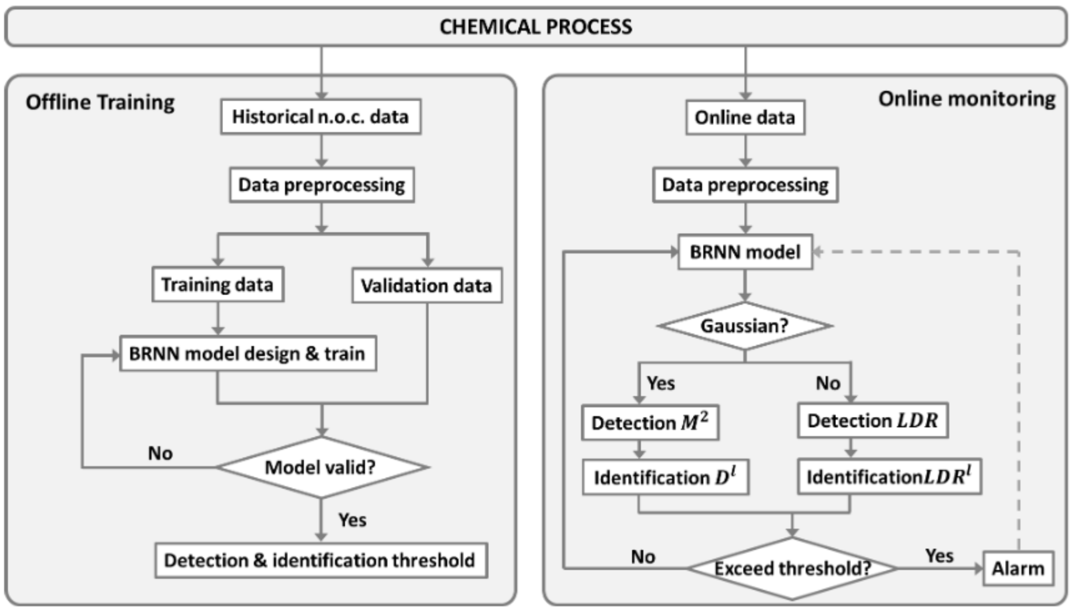


Figure 2.21: BRNN fault detection and identification process for the chemical process presented in [14].

As mentioned at the start of the section, ANNs and RNNs are not able to detect other types of faults than the ones provided during training, cannot discern between processes that cause the same fault and they do not provide a degree of uncertainty in the prediction. BRNNs solves the first 2 problems by providing a per-variable analysis without pre-defined classes, allowing to track the fault from its origin throughout the system. The last problem is tackled by applying the Bayesian framework which computes a probability distribution as predictive posterior. The distribution provides information about the prediction uncertainty and about the observation deviation from the NOC.

2.1.3. Fuzzy logic

The term ”fuzzy logic” (FZ) was coined by Lotfi Zadeh in 1960’s [111, 112] and it is a form of logical reasoning capable of describing vagueness or impreciseness through its linguistic representation of human knowledge; it is able to include the uncertainties and possibilities that can be found in human decision making and reason under uncertainty. In contrast, with classic or Boolean logic in which an element is member or not of a set, meaning that the membership of that element to the set is either 0 or 1, fuzzy logic uses fuzzy sets that allows memberships in the closed interval [0,1]. As a result, elements can be part of a fuzzy set to a certain degree, preserving information in those scenarios in which membership is not certain [113].

Fuzzy logic consists of three main components: membership functions which define the degree of membership to fuzzy sets, fuzzy set operations such as union or intersection for fuzzy sets, and fuzzy rules which, as qualitative knowledge-based approaches such as expert systems and fault trees, exploits the if-then reasoning rule. The fuzzy rules are the intersection between the human reasoning and the mathematical representation.

These 3 elements of fuzzy logic allow fuzzy inference in 3 steps, as can be observed in Figure 2.22 [15]. The first step is called fuzzification in which the historical data is used to build the membership functions for healthy and faulty conditions, and the crisp input values are assigned to the fuzzy sets with some degree of membership. The second step is to build the if-then fuzzy rules which establish the relationships between the data and the faults. Finally, the last step is called defuzzification. Given new data, the membership functions and the rules, the fuzzy information is used to infer a healthy or faulty system and the corresponding fault type. The reader can find more information and examples of all the theory of fuzzy logic, such as fuzzification and defuzzification techniques, in [114–119].

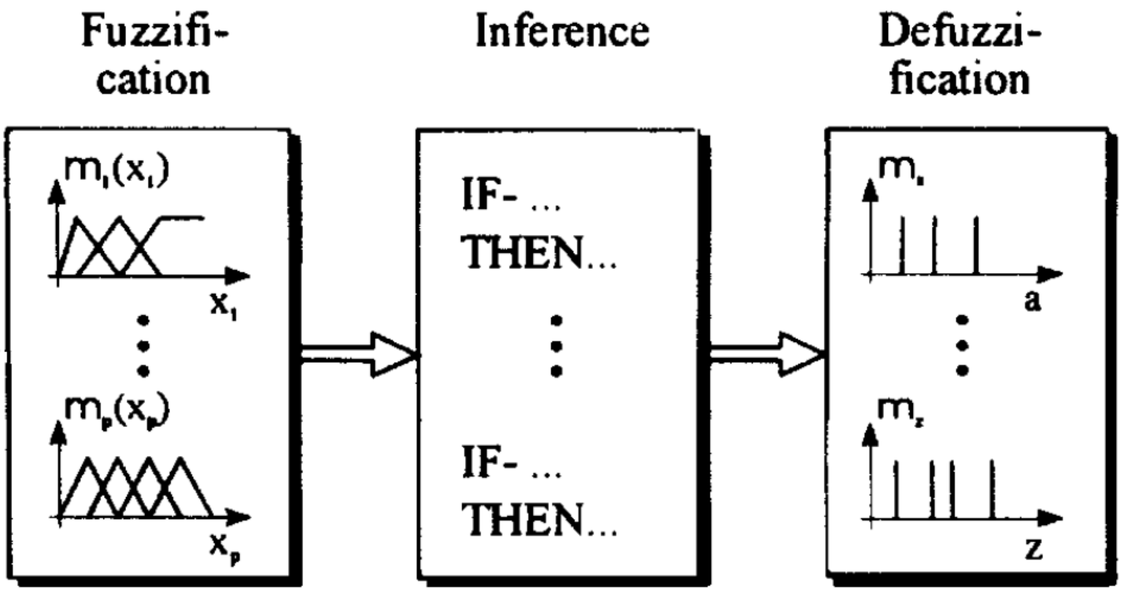


Figure 2.22: Fuzzy inference according to the Mamdami inference method [15].

For fault detection, Sauter [120] proposes an adaptive thresholding that is altered using fuzzy relations. Depending on operating conditions, such as the position or the velocity of the system under study, the change in the threshold is different. Furthermore, Ribeiro [121] provided a detailed explanation and architecture on Mamdami-type Fuzzy Inference Systems (FIS) [122] and assessed the suitability of fuzzy logic for fault detection in space applications. For that purpose, it discussed its operation in 2 ESA related projects: fault detection of the ENVISAT satellite gyroscopes and the development of a fault diagnosis tool for a drill and sampling system of a Mars rover (MODI project). The author concluded that fuzzy logic is appropriate in scenarios with imprecise knowledge or when mathematical modelling is not possible due to complex behaviour, enabling the possibility of leveraging expert knowledge. The same author also compared the Mamdami-type FIS with the (Takagi, Sugeno, Kang)-type (TSK) [123] for the ENVISAT gyroscopes fault detection problem and concluded that the TSK-type outperforms the Mamdami system for MISO systems [124]. Given the TSK superior performance in literature, Li [125] approximated nonlinear systems by a set of TSK fuzzy models for the design of a robust fuzzy observer-based fault detection for general nonlinear systems.

Moreover, Lo [126] applied a fuzzy-genetic algorithm for fault detection and classification in aircraft actuators, where the genetic algorithm is exploited as a human expert for the generation of the optimal fuzzy rule set. The chromosomes in the genetic algorithm are evaluated in parallel by the fuzzy evaluation system which checks whether the estimated and simulated faults are the same given the genetically optimised fuzzy rule table.

For fault isolation, Chin [127] uses a hybrid Cause-Effect network (CE-net) and fuzzy logic for the identification of fault sections within power systems. The CE-net is a graphic modelling tool that allows the representation of faults and the corresponding chain of events. The proposed strategy is to use fuzzy logic to describe the relations between the nodes in the network, as well as fuzzy reasoning and operators instead of search tools. As a result, the fault section is the candidate CE-net path with the maximum value in the inferred fuzzy set.

More recently, Ramos [128] applied fuzzy logic for multiple fault detection and isolation. For fault detection, it compares the output of the system with the output of its model in normal operating conditions in order to generate residuals. Then, for fault isolation, individual models of the system were built for each of the faulty conditions and residuals for each type of fault were computed. The fuzzy sets were defined asymmetrically, giving more weight to the set that represented the fault in order to improve the certainty of fault occurrence. Given that the presence of each type of fault is assessed individually, the defuzzification step is not required. Besides that, the algorithm is able to detect the presence of unknown faults since fault detection is not constrained to the identified fault types. Finally, in order to improve the robustness, the data was pre-processed with a wavelet transform that facilitated the removal of noise.

Since the computations in a fuzzy model have a layered structure, a hybrid approach has emerged that combines fuzzy logic and neural networks. This method aims at the optimisation of the antecedent membership functions by gradient-descent learning algorithms common in neural networks, such as back-propagation [117]. For instance, Ayoubi [16] considers a hybrid scheme with 3 layers, as can be seen in Figure 2.23 [16]. The antecedent layer performs the fuzzification of the crisp inputs to the network with radial neurons. Then, the relation layer represents the rule layer that applies the fuzzy operations and sums the antecedent possibilities. The last layer is known as the conclusion layer and performs the defuzzification.

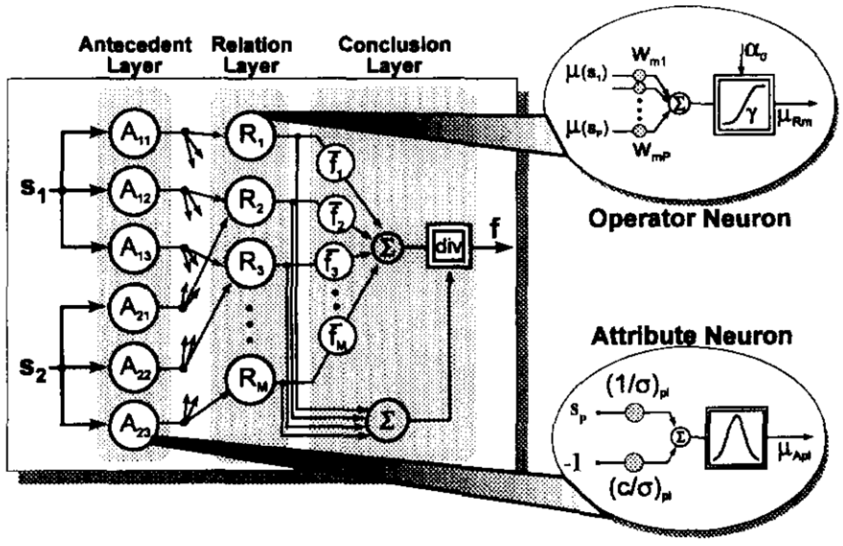


Figure 2.23: Neuro-fuzzy network architecture with 2 inputs and 1 output [16].

Finally, Chen [129] built a fault monitoring and time-evolution tracking method that uses adaptive neuro-fuzzy inference systems (ANFIS) with high-order particle filtering that updates the system states online in order to take into account the changing system dynamics. The author also explained an adaptive recurrent neuro-fuzzy inference system (ARNFIS), applied both approaches to helicopter faults and concluded that ANFIS with a high-order particle filter outperforms classical predictors.

2.1.4. Support Vector Machines

Developed in 1992 at AT&T Bell Laboratories [130, 131], Support Vector Machines [3] (SVM) is a learning algorithm that aims at maximizing the margin between the decision boundary and the different classes in which it is desired to split the training dataset. The main intuition is that the larger the margin, the lower the generalization error of the classifier.

[3]Great online resources to understand the intuition behind SVM and its derivation are:
`https://www.youtube.com/watch?v=_PwhiWxHK8o` (MIT 6.034 AI, Fall 2010),
`https://www.coursera.org/learn/machine-learning` (Machine Learning Stanford Coursera course).

Additionally, the optimisation problem depends on a hyper-parameter that applies a trade-off between maximizing the margin and allowing samples to lie on the wrong side of the boundary. Being lenient with misclassifications reduces its sensitivity to outliers (overfitting). In the case of a 2 class classification problem, the goal is to find the hyperplane D(x) such that one of the two classes is predicted (y_k = 1) when D(x_k)>0 and the other class (y_k = -1) is predicted otherwise. This hyperplane or decision function has to be linear in its parameters but does not have to be linear with respect to x, meaning that cross-terms are allowed. Besides that, it can be expressed either in direct space or in dual space where different kernel functions can be used, such as the Radial Basis Function.

Considering the simple scenario shown in Figure 2.24 [17] with a linearly-separable dataset, SVM will find the optimal hyperplane such that its distance to the closest point of each class (margin) will be equal and as large as possible. Those points are called support vectors and they represent each data class in the decision boundary location process. In the case that the 2 categories can not be separated by a linear classifier, the SVM with kernel functions moves the data into a higher dimensional space that allows the classification of the observations by a higher dimensional hyperplane; the dimensionality of the problem can be increased until a solution can be found.

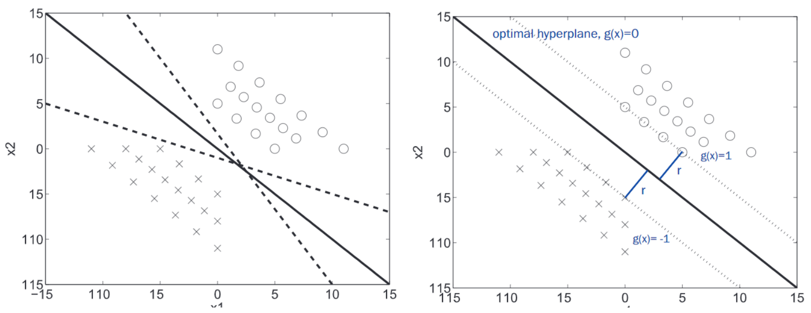


Figure 2.24: Visual representation of the action of the SVM in a linear-separable 2 class problem [17].

When compared to Neural Networks, SVM obtain a similar accuracy when trained with the same dataset. However, when a lot of data and computational power is available NNs tend to outperform SVMs. In terms of data required for training and its required time, SVMs are less data hungry and can be trained faster. Finally, SVMs are less sensitive to the weight/parameter initialisation when compared to NNs since it is a convex method that provides an optimal global solution (immune to local maxima) [17, 60].

Previous work ([132]) has compared the performance of SVMs for classification against 16 other methods with 21 datasets and for regression against 9 approaches with another 12 data sets. Results show that it has good performance but it does not hold overall superiority. For instance, NN and random forests tend to outperform SVM in regression tasks.

SVMs have been widely applied for fault detection and diagnosis in a wide range of fields. In [133], SVMs were used for helicopter rotor blade damage detection using the vibrations in the hub as features. The output specified the damage class with the required action from the operators. In order to deal with more than 2 classes (k classes), the ”one to others” approach was used [134] that repeatedly aims at separating the data from one class from the data of the k-1 classes. The process is repeated with the data corresponding to the k-1 classes until 2 classes remain.

In contrast, Kurek [135] used a single-class SVM (it has 2 classes: healthy and faulty) for fault detection and a ”one-against-one” SVM approach [136] for the fault diagnosis of the rotor bars within an induction motor using the FFT of the phase current, voltage and shaft field in steady state as features. The last approach aims at obtaining the fault type within M classes by training M(m-1)/2 SVM classifiers that distinguish every 2 classes of data. The chosen class is the winner of the most 2-class SVM classifications. Similarly, Santos [17] applied ”one-against-one” SVMs with different kernels (linear, Gaussian, perceptron and stump kernels) for the fault detection in wind turbines using the vibration and electrical signals from multiple sensors. In [137] a simple single-class SVM was implemented to detect whether a small UAV suffered from actuator loss of effectiveness fault or it was operating in nominal flight conditions. In the same line, Jeong [138] used SVMs to detect whether a fault was present in vehicle suspensions.

Finally, another approach was followed by [139] that aims at the sensor fault detection, diagnosis and signal reconstruction for small UAVs. The author proposes to use a least squares support vector machines (LS-SVM) as predictive model that generates sensor estimates for fault detection. Then, principal component analysis (unsupervised learning method analysed in section 2.2) is used for fault isolation and the output signals of LS-SVM, in the case of sensor fault, are used instead of the real observations. A modified version of LS-SVM, namely online sparse LS-SVM (OS-LSSVM), is explained in detail in [140].

2.2. Quantitative knowledge: unsupervised learning

In contrast with supervised learning that enjoys the information found in large quantities of labelled historical data, unsupervised learning searches for patterns within non-human labelled data. In most cases, unsupervised learning methods are used for clustering or data-dimensionality reduction, the latter also known as feature extraction. This section also includes forms of semi-supervised learning, an approach found between unsupervised and supervised training which mostly extracts latent features from large corpus of unlabelled (unsupervised) data and is later fine-tuned with a small amount of labelled data (supervised) to perform a specific task.

In Figure 2.25, some of the most common methods of unsupervised learning within the field of fault detection and diagnosis can be found. In the coming sections, the highlighted methods will be described in detail with references to literature.

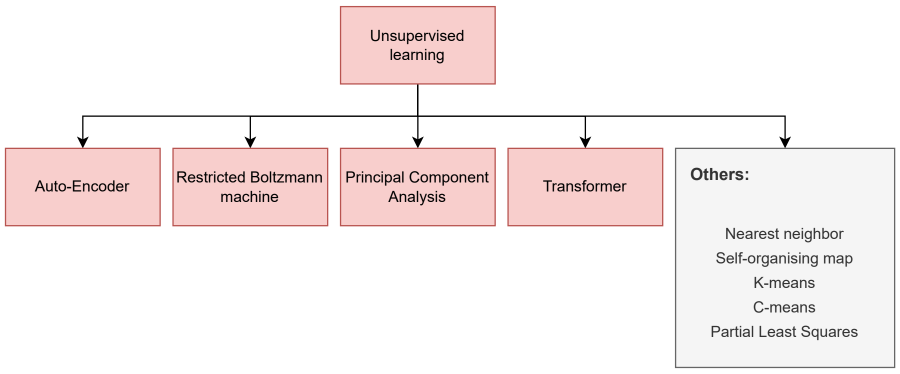


Figure 2.25: Unsupervised learning knowledge-based fault detection and diagnosis methods

2.2.1. Auto-Encoders

Auto-encoders (AE) are part of a subgroup within unsupervised learning known as self-supervised learning (SSL). Methods within this category can automatically generate labels from data, its own supervisory signal, and learn the corresponding mapping. AEs are a great example that exploit NNs (Figure 2.26 [18]). It is a NN architecture that aims to copy its input to its output with two components: an encoder that enforces a bottleneck in the network in order to create a compressed knowledge representation of the original input, reducing the data in the observed space to the latent space, and a decoder that aims at reconstructing the original input from the compressed information. In order to prevent a linear mapping to the input, where x is mapped to x everywhere, AEs are constrained such that they are forced to prioritise useful properties in the data [10]. Unfortunately, large initial weights cause AE to find poor local minima and small initial weights cause that the gradients are so small that training an AE with multiple layers is infeasible. As a result, Hinton [141] proposes a pre-training procedure based on RBMs that finds initial weights close to the good solution.

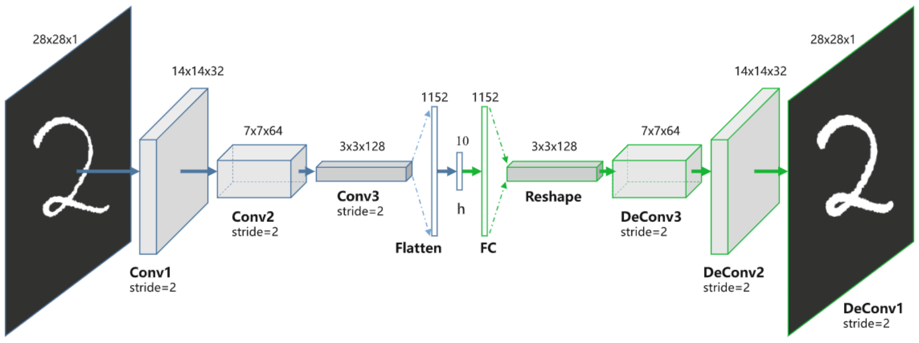


Figure 2.26: Convolutional Auto-Encoder representation [18].

An example that exploits this compressed representation is the previously discussed work by González [11], which uses hierarchical LSTMs with pseudo-random signals for FDI in the TEP benchmark. Here, the training is carried in two different parts, namely a self-supervised stacked auto-encoder of LSTM units stacked together and a softmax classifier later trained with supervised learning. Once the first part of the training is achieved, the decoder is thrown away and the softmax classifier is added at the end of the network before starting the second learning phase. The initial AE compresses the data and provides weight values that serve as good initial guesses for the second training part.

An approach that also combines the joint effort of AE + LSTM for the TEP problem is [142]. However, in this case they are separate components. The AE is trained offline with normal data and is in charge of fault detection. When data containing a fault is given to the AE, its reconstructed output will not match the input. Then, the intermediate compressed representation is passed to the LSTM for fault classification.

In [143], a unified training method is developed that can be applied to PCA and AE architectures for the detection of faults. This method is called the robust self-supervised model and it is based on the addition of Gaussian noise to the input before being fed to the self-supervised model, whose task is to reconstruct the clean input. It demonstrates that this method is equivalent to the regularization in supervised learning and that there is a higher fault sensitivity when compared to the vanilla self-supervised algorithms. A fault is detected when the prediction error shows a deviation from the normal data.

2.2.2. Restricted Boltzmann Machines

Another unsupervised strategy are Restricted Boltzmann Machines (RBM), which extract information from the observation to the latent space, analogous to the AE. The main difference can be found in the architecture, the training and the representation of the latent space. In contrast with AE that compresses the data into point estimates, RBM are trained to predict distributions.

A clear derivation of RBM, as well as their inner workings and hyper-parameter tuning is presented in detail by Hinton [144]. Originally, RBMs are composed of two layers of neurons, namely the input or visible layer and the hidden layer, which modify each other during the forward and backward passes in the network. It is a symmetrical bipartite graph since the nodes of a layer are not connected to each other and all the nodes of one layer are connected to all the nodes of the next. The goal is to minimize the reconstruction error, which is defined as the difference between the initial values of the visible layer (input data) and their modified values generated during the backward pass. For that purpose, the weights of the network are updated with an algorithm called Contrastive Divergence (CD) [145], which is defined as an approximate difference of two Kullback-Liebler divergences. From the definition presented in subsubsection 2.1.2.1, it can be considered as the area difference between the probability distributions of the input and its reconstruction. During the forward and backward Gibbs sampling, the weights are the same. A graphical representation of an RBM can be observed in Figure 2.27 [4].

[4]Source:https://wiki.pathmind.com/restricted-boltzmann-machine

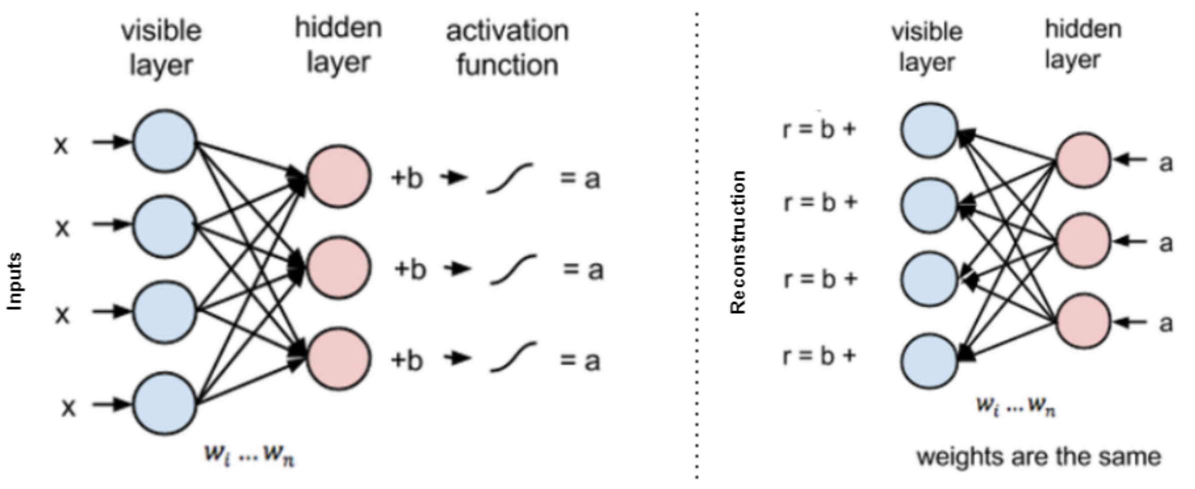


Figure 2.27: Forward and backward RBM pass. See footnote 4 for details.

The combination of multiple RBMs or AEs leads to a class of deep neural network known as deep belief network (DBN) which is composed of a single input layer and multiple hidden layers of latent variables. Each layer communicates with the previous and next layers. It can be studied as multiple stacked RBMs where each hidden layer is considered as an input layer for the next and training is carried out by applying CD to each input-hidden layer subnetwork, starting from the ”real” input layer. First, it is trained without supervision in order to reconstruct the inputs probabilistically, as done in the RBM, and then it is fine tuned with supervised classification training.

DBNs have been recently applied in the field of fault detection to multiple different systems. Huang [146] combines DBNs and Global Back-Reconstruction (GBR) for determining the presence of early cracks in turbine blades. GBR reduces the feature degradation by building layer skipping connections that link the hidden layers with the visible layer, analogous to residual connections in Residual NNs. For that purpose, the 3D position of a fixed point within the blade is fed to the algorithm since the presence of fatigue cracks would deform the blade surface leading to a 3D displacement. Furthermore, Dash [147] used DBNs for robotic manipulator execution failure and Xing [148] proposes a distribution-invariant DBN (DIDBN) for carrying out fault diagnosis of machines (gearboxes) analysing vibration data. The DIDBN consists of 3 layer types: locally connected RBM (LCRBM) layer that extracts features from local segments of the vibration signals, fully connected RBM (FCRBM) and mean-discrepancy maximum RBM (MDM-RBM). The latter is able to extract features with similar distributions under different conditions. Finally, Chen [149] designed a DBN for early warning and fault detection (stuck and constant deviation faults) for rotor UAV.

2.2.3. Principal Component Analysis

Principal Component Analysis (PCA) is a multivariate statistical analysis method that aims at the reduction of data dimensionality for essential information retention by orthogonal projection on a linear subspace of lower or equal dimensionality. It can be assumed that the small changes in the data are caused by noise and the reduced representation only keeps the big data variations (feature selection). For that purpose, it constructs an optimal subspace where the variance of the orthogonally projected data is maximised, which is equivalent to finding the subspace such that the mean-squared distance of each data point to its projection is minimised (minimisation of the squared reconstruction loss). Therefore, the new coordinate system is centred around the mean with the first axis aligned with most variance in the data, the second axis orthogonal (uncorrelated) to the first and preserving most of the remaining variance, and the rest of the axes orthogonal to the previous ones and preserving each time most of the remaining variance [150]. When the data is projected to the Principal Components it appears less noisy, enhancing the tasks of pattern recognition and clustering.

The algorithm consists of 3 main steps:

1. Computing a D-dimensional PCA subspace from an M-dimensional dataset $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]$ with N data points:

   (a) Compute the mean $\boldsymbol{m}$ and subtract it from data to obtain $\boldsymbol{X} = [\bar{\boldsymbol{x}}_1, \dots, \bar{\boldsymbol{x}}_N]$:

   $$\boldsymbol{m} = \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{x_i} \quad \forall j : \bar{\boldsymbol{x}}_{\boldsymbol{j}} = \boldsymbol{x_j} - \boldsymbol{m} \tag{2.47}$$

   (b) Compute the M×M covariance matrix $\boldsymbol{C}$ on zero-mean data $\bar{\boldsymbol{X}}$:

   $$\boldsymbol{C} = \frac{1}{N}\sum_{j=1}^{N} \bar{\boldsymbol{x}}_{\boldsymbol{j}} \cdot \bar{\boldsymbol{x}}_{\boldsymbol{j}} = \frac{1}{N}\left(\bar{\boldsymbol{X}} \cdot \bar{\boldsymbol{X}}^T\right) \tag{2.48}$$

   (c) Compute the unit-length eigenvectors $\boldsymbol{w_i}$ and scalar eigenvalues λ_i of $\boldsymbol{C}$:

   $$\boldsymbol{C} \cdot \boldsymbol{w_i} = \lambda_i \cdot \boldsymbol{w_i} \tag{2.49}$$

   (d) Sort the eigenvalues and corresponding eigenvectors from the highest to the lowest eigenvalue. Since the amount of data variance maintained in the PCA dimension i is proportional to λ_i, we will maintain those dimensions with the highest eigenvalue.

   (e) The first D eigenvectors are the principal components of $\boldsymbol{X}$ and define the PCA transformation $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w_D}]$. The percentage of variance conserved is the ratio of the sum of kept D eigenvalues with the sum of all the M eigenvalues.

2. Project the data $\boldsymbol{x_j} \in R^M$ onto the PCA constructed subspace $\boldsymbol{x}_{\boldsymbol{j}}^* \in R^D$:

   $$\boldsymbol{x}_{\boldsymbol{j}}^* = \boldsymbol{W}^T \cdot \left(\boldsymbol{x_j} - \boldsymbol{m}\right) \tag{2.50}$$

3. Back-project from PCA subspace $\boldsymbol{x}_{\boldsymbol{j}}^* \in R^D$ back to $\boldsymbol{x}_{\boldsymbol{j}}' \in R^M$, being the reconstruction $\boldsymbol{x}_{\boldsymbol{j}}'$ an imperfect approximation of the original $\boldsymbol{x_j}$:

   $$\boldsymbol{x}_{\boldsymbol{j}}' = \boldsymbol{W} \cdot \left(\boldsymbol{x}_{\boldsymbol{j}}^* + \boldsymbol{m}\right) \tag{2.51}$$

Since the exact value of the eigenvalues is not important, but their relative value from one to another, the scale in the computation of $\boldsymbol{C}$ can be ignored:

$$\boldsymbol{C} = \left(\bar{\boldsymbol{X}} \cdot \bar{\boldsymbol{X}}^T\right) \tag{2.52}$$

Over the last decades, different versions of the original PCA have been developed, such as the kernel-PCA, the robust-PCA and the weighted adaptive recursive PCA (WARP). In order to circumvent the constrain that the variables have to be linearly correlated, the kernel-PCA [151] maps the nonlinear original data onto a higher-dimensional feature space to get linear mapped data using a kernel function. Robust PCA (RPCA) consists of a series of approaches, such as Principal Component Pursuit [152], that modify the original PCA to make it robust against multiple corrupted observations. Finally, WARP [153] aims at reducing the increasing false alarm rates caused by the natural deterioration of processes or systems. Other PCA variants include the moving window kernel PCA [154], the sparse PCA [155] or the shrinking PCA [156].

In literature, PCA methods have been used as core components in fault diagnosis approaches, as well as pre-processing components in which data's dimensionality is reduced before being fed to a classifier. Examples include [157] which uses the PCA reconstruction error as self-supervised method of fault detection and [158] which measures the fit of a new sample to a PCA model.

2.2.4. Transformers

Back in 2017, Google DeepMind published the paper ”Attention is all you need” [19] with the goal of bringing attention mechanisms, a concept that aims at building algorithms able to highlight the key information from the input space while ignoring the rest in order to efficiently use computing power, to deep learning; more specifically, to Natural Language Processing (NLP). It builds on the work of [159], one of the main innovators in the field of Neural Machine Translation. Although Transformer models undergo unsupervised pre-training, they are supervised fine-tuned with labelled data (semi-supervised training).

Conventionally, sequence to sequence translation was based on the encoding of the whole sentence from which the translation is decoded, leading to problems when translating long sentences. In 2015, Bahdanau [159] proposed a soft-search approach in which the decoder could choose what part of the input was useful in the translation by tapping into the hidden states of a bidirectional encoder, recurrent structure that allowed the observation of the following and preceding words. Until then, RNNs, LSTMs and GRUs where the state-of-the-art recurrent approaches in sequence modelling; however, they were not able to keep information and dependencies over long sequences, sequential processing prohibits parallelisation and there is no hierarchy of importance of all the previous inputs when making the next prediction. CNNs were also applied to NLP but convolutions can only exploit local dependencies and, in order to relate features far from each other, many layers are required.

Google DeepMind’s model, the Transformer, aims at solving all these problems by introducing the ”*first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution*” [19]. Apart from the original paper and its open-source code [5], there are multiple sources that visually explain the Transformer [6][7][8].

The Transformer consists of two main parts, the encoder and the decoder, as can be observed in Figure 2.28 [19]. The encoder is in charge of transforming the input to latent variables whereas the decoder takes those latent variables and generates the output.

The encoder consists of two main sub-layers, namely the multi-head attention and a feed forward (single hidden layer) neural network, as well as residual connections around these sub-layers followed by layer normalization. As can be seen in Figure 2.28 and Figure 2.29 [19], 3 identical signals are fed to the multi-head attention and they are multiplied in a linear stage by 3 different trainable matrices, resulting in 3 different inputs to the scaled dot-product attention block: a value (V), a key (K) and a query (Q).

These abstractions are used within the scaled dot-product attention block for the creation of a weight matrix for each input. The weight matrix denotes the attention to be paid to other elements of the input sentence (context) when encoding the current word. For that purpose, the similarity of its query is compared to the key of each of the other inputs by means of the dot product. Before applying this weight matrix to the value of the input, it is normalised by the dimension of the keys (gradient stability) and it is passed through a softmax function such that the sum of all the weights adds up to 1; as can be observed in Figure 2.30 [19] and whose matrix form is denoted as Equation 2.53. This weight matrix embodies the attention mechanism by providing a higher value to those inputs that contain most of the information for the current prediction (the inputs to be focused on) and eliminates the influence of irrelevant inputs. Although each input follows its own path through the encoder, there are dependencies between the other input paths within the multi-head attention block due to the query-key dot product. The reason why each input is mapped to a distinct query and key value is for allowing relations between inputs that are not symmetrical. Word *x* may require information from word *y* but that relation might not exist the other way around, leading to non-symmetric around the diagonal weight matrices.

[5] `https://github.com/google/trax`
[6] `http://jalammar.github.io/illustrated-transformer/`
[7] `http://nlp.seas.harvard.edu/2018/04/03/attention.html`
[8] `https://www.youtube.com/watch?v=rBCqOTEfxvg`

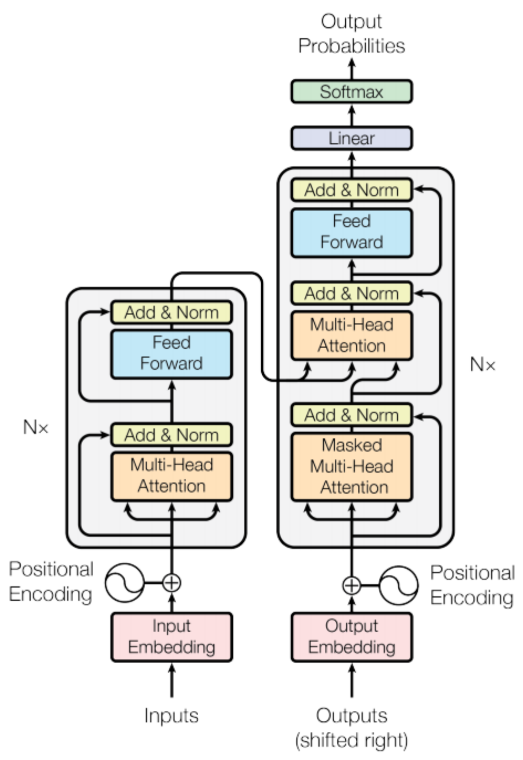
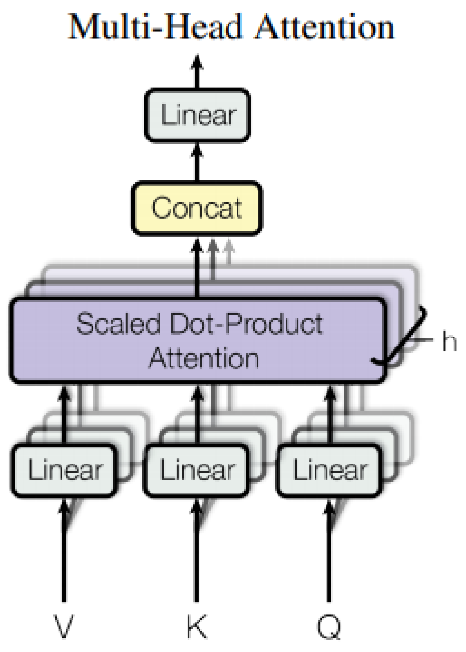


Figure 2.28: Transformer model architecture [19].



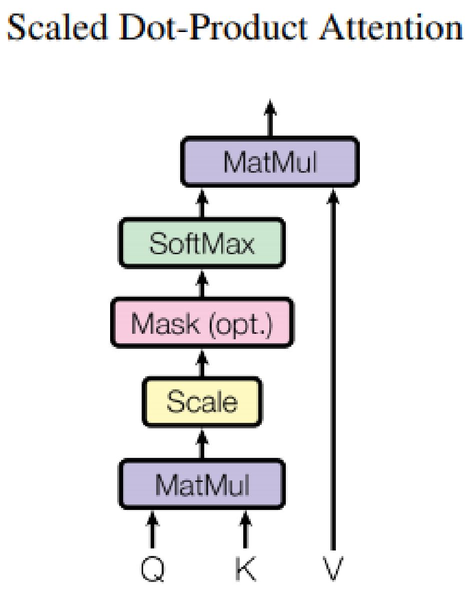


Figure 2.29: Multi-head attention [19]

Figure 2.30: Scaled dot-product attention [19]

$$\text{Dot product attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.53}$$

As can be observed in Figure 2.29 [19], h linear mappings and scaled dot-product attention blocks are run in parallel. Different W_Q, W_K and W_V matrices are learnt that allow the algorithm to pay attention to different input elements. This is called Multi-Head Attention. For instance, in NLP, there might be a group of (W_Q, W_K, W_V) matrices that attend to different syntactic and semantic structures, such as identifying the subject or the direct object of the sentence. Then, the outputs are concatenated and a linear mapping is applied in order to meet the next input's dimension requirement.

Having analysed the encoder architecture, the decoder is very similar, since the the only difference can be found in the attention blocks. The first decoder attention block is masked, meaning that each input is only able to pay attention to the previous ones. In the case of NLP, a word can only see the previous words and not all the words in the sample sentence. The rest of the decoder attention blocks receive as input the output of the previous decoder block as queries and the encoder output as the keys and values.

Both encoder and decoder have an embedding block and positional encoding. Since the Transformer was originally designed for NLP, the input text had to be encoded into numerical vectors. Besides that, in order to encode information about the absolute and relative position of each word within the sentence, sinusoidal signals of different frequencies were added to the embeddings; more details about positional encoding can be found in the original paper [19].

Finally, N_x encoders and decoders could be stack one after the other. After the last decoder, a linear mapping and softmax function is applied to the output, yielding the probabilities of the next element in the sequence; in NLP, the next word. Once the next element is chosen, it is fed as input to the decoder for the prediction of the following element.

In contrast with recurrent layers that require $\mathcal{O}(n)$ sequential operations to connect all elements of a sequence, the Transformer is capable of doing it in constant number of sequential operations. Besides that, whereas the complexity per recurrent layer is $\mathcal{O}(n \cdot d^2)$, in the Transformer it is $\mathcal{O}(n^2 \cdot d)$. As a result, Transformer layers will be faster if the number of inputs (words in a sentence) is lower than the number of dimensions (embedding vector dimensions). In the case of convolutions, they are more expensive to compute than recurrent layers and multiple layers are required to connect all input elements.

Apart from a benefit in lower required computation, the use of Transformers can lead to more interpretable models as it is possible to observe the attention that each input pays to its neighbours. The original paper provides Figure 2.31 [19] as an example in which it is possible to observe the attention given by each of the heads to each of the words of the sentence when providing the word ”making” as input.

During the last years the Transformer architecture has grown in popularity in the machine learning community, it has shown better results than the state-of-the-art in NLP and they are the backbone of the Generative Pre-trained Transformers GPT-2 and GPT-3, huge general-purpose learners that can perform multiple NLP tasks, such as translation or text summarization, without having been trained to perform any of them. Transformers have shown that the larger the model, the higher the accuracy (they do not saturate) and the more interesting behaviour emerges. More recently, improvements to the Transformer have been proposed with modified versions such as the Reformer [160] [9] and the Linformer [161]. The former uses local sensitive hashing (LSH) to reduce the time complexity of the attention layer and reversible residual layers [162] to reduce the memory required to store the activations for backpropagation. The latter is able to reduce to time complexity to $\mathcal{O}(n)$ by exploiting the observation that the weight matrix is low-rank.

In the last 3 years, multiple papers have appeared exploiting Transformers for Computer Vision tasks. In 2018, Parmar [163] translated self-attention to images obtaining state-of-the-art results for image super-resolution and generative image modelling. Later, in 2019, Cordonnier [164] showed that multi-head attention layers with relative positional (quadratic) encoding with sufficiently number of heads can be as expressive as convolutional layers and Bello [165] proposed the augmentation of

[9] `https://github.com/google/trax/tree/master/trax/models/reformer`

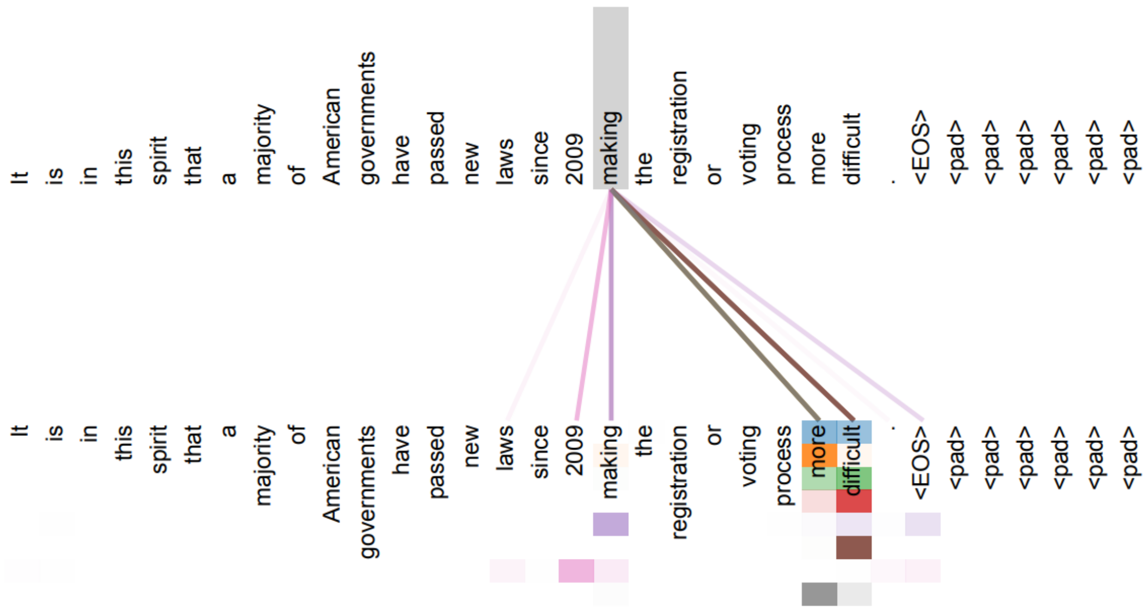


Figure 2.31: Visual representation of the attention mechanism when provided the verb "making" as input. Each of the 8 heads is represented by a colour and the more transparent, the lower the attention [19].

convolutional networks with attention mechanisms by concatenating CNN and self-attention features. The experiments show that attention augmentation leads to better results in image classification and object detection, even in computing power constrained models, while maintaining a similar number of parameters. It also shows how to apply attention to images. Unfortunately, the proposed method increases the inference and training time in 25% while improving the accuracy in 1.3% when compared to the baseline. Carion [166] and Ramachandran [167] also noticed the higher training and inference time due to the lack of optimised kernels for attention. The latter also performed a study of how and when is beneficial the attention-convolutional combination. Finally, Khan [168] provides a survey of how transformers have been applied to computer vision, highlighting that attention currently suffers from a few challenges, such as the high computational cost, the high data cost because it does not encode prior knowledge, and the lack of hardware efficient designs in comparison with GPUs that are ideal for CNNs.

Attention mechanisms have started to be applied to challenges involving video, such as video classification [169]. Transformers applied to the field of fault detection is a subject still in its infancy as [170] is the only paper that approaches this field by detecting earthquakes with a deep neural architecture that uses transformers and local attention mechanisms.

2.3. Qualitative knowledge (symbolic AI)

Symbolic AI, also known as Good Old Fashion AI (GOFAI), represents a series of methods popular in the second half of the 20th century which are heavily based on logic, search and a symbolic representation of the problems.

Within this category we can distinguish 3 main groups, namely fault trees, signed diagraph and expert systems [3]. The following sections will briefly discuss their main intuition together with examples of their application in the fault detection and diagnosis literature.

2.3.1. Fault trees

Fault tree analysis (FTA) consists of the creation of a tree-like structured logic diagram that represents the fault behaviour of a physical system. It provides a visual representation of the system elements involved and their relationships by means of event and logic symbols. Through quantitative and qualitative analysis the root cause of the failure can be found. An FTA usually consists of 4 steps [171]:

1. System definition
2. Fault-tree construction
3. Qualitative evaluation
4. Quantitative evaluation

Lee [171] provides a detailed overview of FTA with the steps for system definition to different methods of fault-tree construction and evaluation. In most cases, each failure type requires a different fault tree for which the failure causes are listed in order of occurrence with their respective probabilities based on collected data. After constructing the fault tree, it is evaluated seeking any potential improvement. Finally, based on the observations and the probabilities, hazards and their corresponding failures can be identified.

From literature a few examples of FTA can be mentioned. Kladis [172] carries out a quantitative and qualitative analysis of fault trees for the fault diagnosis of an electrically powered UAV. Shi [173] presents an approach for the fault-tree construction applied to advanced process control systems. Finally, Abdallah [20] exploits the power of FTA for the assessment of communication reliability within a UAV fleet formation. To visualise the structure of a fault tree, an example of a drone crash can be observed in Figure 2.32 [20].

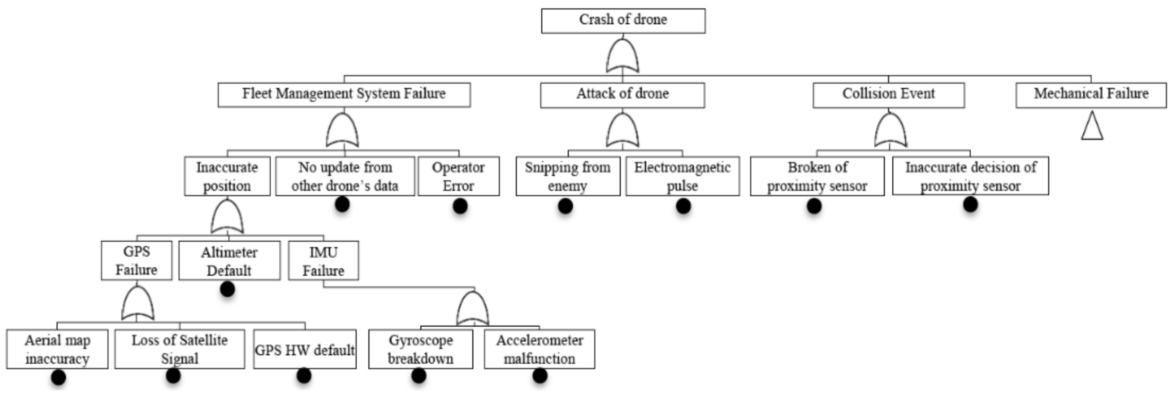


Figure 2.32: Drone crash FTA [20].

2.3.2. Signed digraph

Signed digraphs or diagraphs (SDG) are a graphical representation of qualitative models or cause-effect relationships. It is represented by nodes that contain the cause or effect state and edges or directed arcs that define their causal relationship. There are 3 types of nodes: the inputs or those that only have outgoing edges, the outputs that only have incoming edges and the process variables that have both types of directed arcs. The edges also contain a sign (+/-) which represents whether the cause and effect move in the same direction. If the sign is positive, when the cause increases its value, so those the effect; and vice versa. Venkatasubramanian [58] provides the progress of literature in the development of this graphical representation.

This qualitative knowledge method has been widely used for fault diagnosis in the chemical field. Chang [174] proposes a hybrid approach that combines SDG and SVM for the fault diagnosis within the chemical Tennessee Eastman Process model. SDG provides causal information about the different variables and the SVM establishes the relation using the data available. Similarly, Lee [175] combines SDG and Dynamic Partial Least Squares for the diagnosis of chemical processes. Finally, Xiaolei [176] proposes an Improved SDG (ISDG) for the diagnosis of multiple faults in an aero-engine.

2.3.3. Expert systems

Expert systems (ES) are rule-based approaches that present expert human experience into a set of rules that a computer can interpret. In contrast with quantitative knowledge-based system that require a lot of data and training with trial and error, expert systems makes conclusions based on logic and reason. Besides that, a human can interpret and explain the decisions of the system, as well as modify its propositional logic. An overview of these type of approaches and their applications can be found in [177].

Expert systems had a strong presence in the 20th century; however, their importance within the research community has been decaying during the last two decades. In the aerospace field, Bo [178] presents an expert system approach for the diagnosis of UAV control surface damage. In the chemical field, Zhang [179] provides a method for the formulation of diagnostic rules for an expert system model applied to a chemical tank stirring process. Finally, current research combines expert systems with other approaches, such as fuzzy logic [180].

3 Vision

As mentioned in the introduction, the camera is ubiquitous among commercial drones and is not exploited for fault detection and diagnosis even though its information is being processed for navigation or control purposes. The goal of this section is to explore some of the key applications of visual information and how features are extracted from this high-dimensional input space.

This chapter discusses 3 main fields in which visual information is exploited, namely optic flow, visual inertial odometry and next frame prediction. Optical flow is presented in section 3.1 and, among its numerous applications, it is used for estimating the motion between multiple frames. Then, in section 3.2, VIO is explained; method which combines the information from the camera and the IMU for navigation. Finally, section 3.3 summarises research in the field of next frame prediction, subject which sparked the interest for this line of fault detection and diagnosis research.

3.1. Optic flow

Optic flow is defined as ”the pattern of motion induced on the retina of a moving observer” [181], resulting in the apparent relative motion between the observer and the objects within the scene. The goal of optical flow methods is to find the motion between two frames through the displacement of intensity patterns by assuming that the intensity of moving pixels remains constant during displacement. This information allows an agent or vehicle to understand the location of the Focus of Expansion or its direction of motion for navigation, its relative velocity or its time-to-contact in landing scenarios. The derivation of the optical flow using the pinhole camera model can be found in [182]. Whereas the aforementioned derivation exploits 3D geometry, Horn [183] proposes an alternative method based on the 2D rate of change of image brightness.

There exist 2 types of optic flow visualisation, namely sparse and dense optic flow. Although the sparse optical flow, represented by the left image in Figure 3.1 [21], provides only the flow vectors of interesting features by means of arrows, it shows an intuitive perception of the physical motion. For the computation of sparse optical flow, the Lucas-Kanade technique [184] is the most common implementation, included in libraries such as OpenCV [1]. It is a local differential method which assumes that the optic flow vector of a pixel will be similar to those of its surroundings. In order to define the points of interest, a corner detector can be used, such as Harris [185], FAST [186] or Shi-Tomasi [187]. On the other hand is the dense optic flow, represented by the right image in Figure 3.1 [21], which provides the optic flow of the complete frame. The colour hue represents the direction whereas the saturation speaks for the vector magnitude. In this case, it allows the visualisation of small differences between pixels and smooth transitions. Software libraries such as OpenCV uses the polynomial expansion method [188] for its computation.

[1]https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

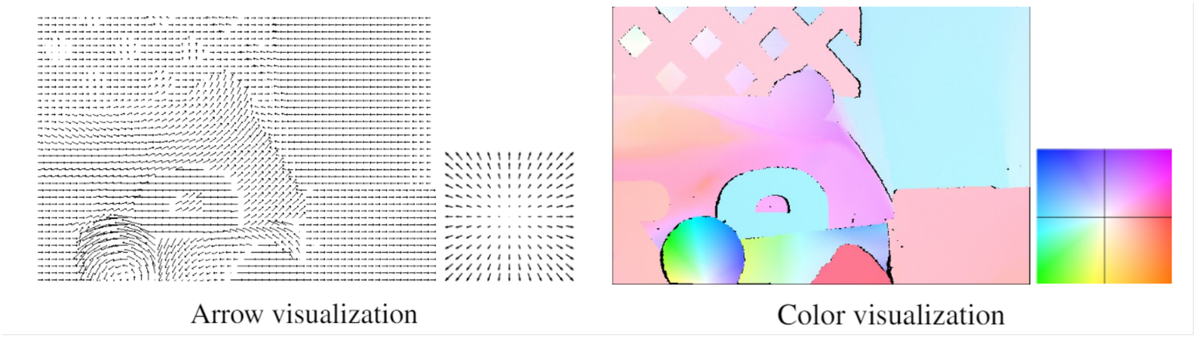


Figure 3.1: Optical flow visualisations: sparse optical flow (left) and dense optical flow (right) [21].

Optical flow has been widely used for UAV landing purposes. Hamel [189] used the average optical flow on a spherical camera for the vertical landing of a VTOL vehicle on a moving platform. In this scenario, the IMU information was only exploited for the derotation of the flow, resulting in an visual-based only control approach. This work highlights the difficulty of the proposed approach in the presence of aggressive motion of the environment and that it could be improved by increasing the measurement rate of the visual sensors. One year later, Croon [190] introduced an efficient optical flow based algorithm for UAV landing with a downward pointing camera. Apart from the standard information extracted from optic flow (ventral flow and time-to-contact), this approach provides an second-order approximation of the landing surface flatness and slope. The authors of this paper mention the potential benefit of introducing a Bayesian perspective in order to cope with noise or inaccurate derotation. An alternative landing approach based on optical flow and inspired in nature, namely honeybees, is constant divergence landing. The work in [191] and [192] corrects the landing instability found in previous literature which implemented this method by proposing an adaptive controller that exponentially modifies its gains during landing. This can be done by understanding and following the self-induced oscillations that the drone experiences in its approach to the ground. Before initialising the descent manoeuvre, the UAV is induced an oscillatory motion that allows the computation of the optimum initial gain. Finally, Hordijk [193] implemented the previous ideas using an event-based camera and showed that it enables the UAV to perform very fast landing manoeuvres.

Apart from landing manoeuvres, optic flow can be used at different stages of the autonomous pipeline of UAVs. Farid [194] applies it for scene mapping and autonomous localization in order to achieve a vision-based autopilot in a GPS denied environment. Valenti [195] uses optical flow for altitude estimation and the UAV movement in the scene. For UAV position estimation, Arreola [196] fuses the dense optic flow information with GPS and commercial autopilot sensors, whereas More [197] fuses optic flow with ultrasonic sensors in order to translate 2D motion to 3D and Rosser [198] exploits low-resolution long-wave infrared sensor visual information in order to use optic flow in darkness. Optic flow is also applied for obstacle avoidance [199] [200], velocity estimation [201], target tracking [202] or object motion estimation [203]. Dedicated hardware has also been developed for efficient computation: the optical flow sensor [204].

3.2. Visual Inertial Odometry

Visual Inertial Odometry or VIO is ”the process of estimating the state (pose and velocity) of an agent by using only the input of one or more cameras plus one or more Inertial Measurement Units (IMUs) attached to it” [22]. The goal is to combine the strengths of both sensors in order to accurately perform state estimation of the 6-DoF vehicle pose, velocities and IMU biases. Even though the IMU has a high refresh rate, it suffers from poor signal-to-noise ratio at low accelerations and rotations, as well as from drift accumulation due to sensor biases. These disadvantages can be corrected by the camera visual information which has 10x lower output rates and, in contrast with the IMU, has a higher accuracy the lower the motion speed. In order to achieve good results, it is key to perform spatial and temporal calibration of both sensors.

VIO can be performed in two ways depending on whether the features from the visual sensors are pre-processed to obtain motion estimates: loosely and tightly coupled VIO approaches. The first one aims at obtaining motion estimates of the IMU and the camera which are later fused. In contrast, the latter approach exploits the IMU motion information to predict the 2D feature locations in the next frame in order to improve the feature tracking of the vision information pipeline. This leads to better results than the loosely coupled VIO. Both concepts can be easily understood thanks to the diagram illustrated in Figure 3.2 [22].

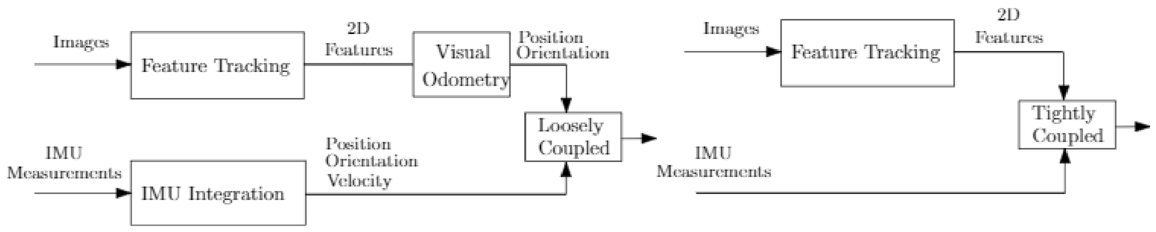


Figure 3.2: Comparison between the loosely coupled (left) and tightly coupled (right) VIO approaches [22].

There are 3 types of state estimation techniques which vary on the number of previous states that are recomputed at every point in time given the new sensor measurements. Filtering methods only estimate the latest state, hence they forget all previous states, have a low computational demand and are very efficient. As a result they suffer from linearisation errors and are strongly affected by outliers. Sliding window estimators or fixed-lag smoothers update a window of previous states, being more accurate but less efficient. They are robust to outliers since it is possible to incorporate outlier rejection mechanisms; however, they still suffer from linearisation errors. Finally, full smoothers exploit the complete pose history to provide a new update. It is the most accurate approach but is computationally demanding. Different modifications have been proposed in order to improve the efficiency, such as key frame storage selection [205, 206].

A summary and benchmark of different monocular VIO algorithms can be found in [207]. A great tutorial about the motion estimation from the camera information has been made available by Davide Scaramuzza [2] from the Robotics and Perception Group a the University of Zurich [3]. This estimation of motion from only visual information is known as Visual Odometry (VO) and there exists efficient approaches that are based on image intensities which skip the feature extraction and matching, such as semi-direct VO (SVO) [208]. Finally, although very similar to SLAM for navigation, VIO and VO do not generate a map of the environment in order to reach loop closures or include the extracted visual features in the state vector; the visual measurements are only used for creating motion constraints [209].

3.3. Next frame prediction

The paper that motivated the proposed line of research for fault detection and diagnosis is ”Geometry-based next frame prediction from monocular video” [23]. Most of the literature on fault detection and diagnosis is based on the prediction of the next time step of one or multiple signals and compare it with the actual next time step measurement in order to identify a potential failure based on the error. However, there is no available work that exploits information from the camera in order to identify a potential failure. Therefore, upon encountering this paper, the author of this document started to question why that was the case and whether the introduction of visual information in the fault detection and diagnosis framework could lead to more accurate results without leading to a computational load not available in most commercial UAVs. The wealth of information available from the camera could improve all aspects of the fault detection and diagnosis pipeline at a minimum computational cost if the extracted features are already been exploited for another purpose, such as navigation with VIO.

[2]http://rpg.ifi.uzh.ch/visual_odometry_tutorial.html
[3]https://www.ifi.uzh.ch/en/rpg/research/research_vo.html

The authors of [23] implemented a convolutional recurrent neural network to obtain depth from monocular video, leveraging the information within sequences of images. Figure 3.3 [23] presents the depth prediction neural network based on convolutional LSTMs. Given the depth information, the current frame and the displacement of the camera, it is possible to generate the next frame prediction, including depth information, using generative computer graphics (Figure 3.4 [23]). As a result, it uses a model-free method for depth estimation and a model-based approach for next frame computation. This combination is not only able to predict the next frame but multiple frames in the future in exchange of an increasing error by predicting the vehicle motion.

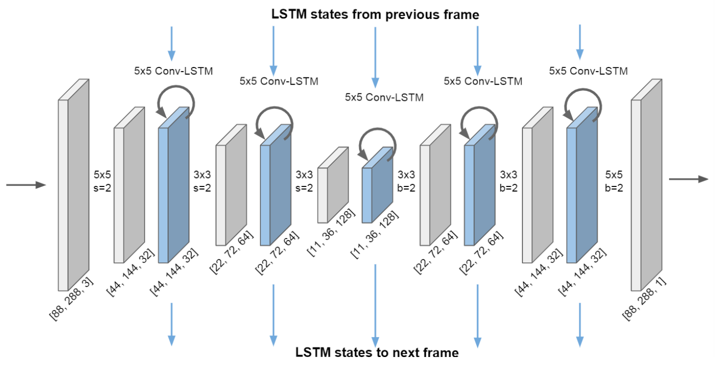


Figure 3.3: Convolutional LSTM encoder-decoder architecture for depth prediction [23].

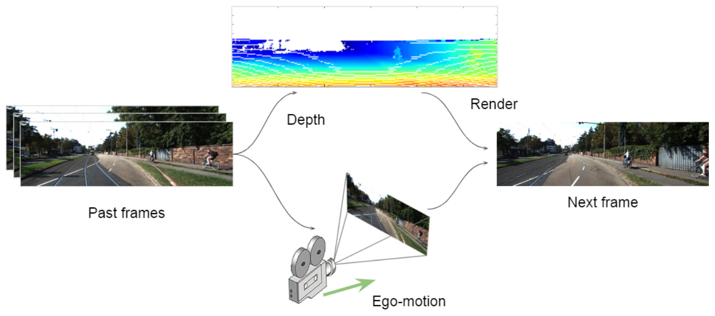


Figure 3.4: Next frame prediction proposed pipeline [23].

The authors tested its superior performance on the KITTI dataset and observed that the accuracy improved as the sequence of images used for depth prediction increased.

Further work has been carried out within the same research group. In 2018, they proposed an unsupervised algorithm for depth and ego-motion estimation from monocular video [24]. Based on an estimation of depth, they estimate a point cloud. Then, with an estimate of ego-motion (translation and rotation) from the previous to the current time step they transform the point cloud one time step back for comparison with the 3D point cloud created in the previous time step. With the transformed 3D point cloud, the authors also generate the expected image that the vehicle should have observed in the previous time step. Their main contribution was the novel loss function which does not only consider the photometric losses (2D image errors) from the reconstructed image, but also the 3D point cloud alignment loss based on the Iterative Closest Point (ICP) algorithm. Although it is not possible to take the derivative of the loss function for backpropagation through the NN that generated the depth and ego-motion estimates, the authors demonstrate that the ICP optimal transformation can be used as negative gradient for the ego-motion estimator whereas the remaining 3D point cloud misalignment after the ICP correction can be the negative gradient for the depth estimator. The authors show better performance when compared to similar methods. The complete pipeline can be observed in Figure 3.5 [24]. The authors have made their code and datasets available online [4].

[4]`https://sites.google.com/view/vid2depth`

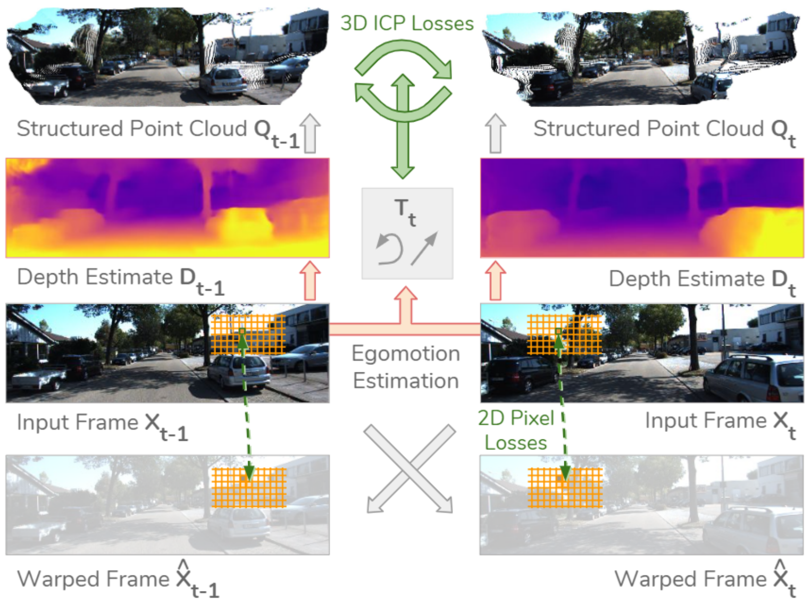


Figure 3.5: Overview of depth and ego-motion estimation framework from monocular video [24].

Later, in [210] they developed a model which is robust to moving objects by modelling their 3D motion and can be transferred to domains different than the trained environment. In 2019, they built upon all previous work developing a pipeline that can be trained in an unsupervised manner from arbitrary videos since the algorithm is able to estimate the camera intrinsic parameters [211]. The code of both papers was also made available [5] [6]. Finally, Castell [212] carries out a comprehensive survey of the field of future frame video prediction.

[5]https://sites.google.com/view/struct2depth
[6]https://github.com/google-research/google-research/tree/master/depth_from_video_in_the_wild

4 Photo-realistic simulator

As discussed in chapter 2, the different fault detection and diagnosis approaches require historic data in order to build their models. Gathering large quantities of data for fault detection with a UAV is very time consuming, dangerous and expensive; data would have to be annotated, multiple failure modes would have to be induced in the vehicle and the flight environment, as well as the UAV, would have to be adapted to minimise the potential risk. Besides that, in an experimental physical setting it is very difficult to collect data from various environments and conditions. A suitable alternative is the simulation of the vehicle in a realistic environment, the storage of the sensor synthetic data for model training and the transfer learning to the real world UAV. Also, it has been observed that the combination of synthetic and real information can reduce the required data when compared to the scenario in which only real data is collected, and the performance would be increased [49].

The selection of the right simulation for the application is key in order to minimise the gap between the simulation and the real world (Sim2Real gap), especially in computer vision applications. If the visual data in the simulator is very different when compared to the information registered by the camera on board of the drone, the validation performance will be lower than the performance observed during training with simulator data.

In this regard, there has been an effort in developing high-fidelity simulators for computer vision tasks in the last 5 years. In 2016, Skinner [25] highlighted the importance of training the algorithms in dynamic environments with time-varying lighting conditions. Therefore, they developed a reduced-size rural and photo-realistic 3D environment in Unreal Engine 4 (UE4), as shown in Figure 4.1 [25], in order to assess the performance of 3 computer vision tasks, namely robust place recognition, object recognition and SLAM. It was concluded that the development of a high-fidelity environment is very time-consuming and it is very difficult to maintain the scale of the different objects within the scene. Besides that, the change in the lightning conditions could not be automated. However, given the built environment and the path to follow by the camera, the generation of new scenarios given a change in the camera location, orientation or sample rate could be done programmatically. It concluded that it was necessary the development of tools that automated the generation of environments, considered one of the main bottlenecks.

One year later, in 2017 Microsoft launched AirSim [26], an open-source simulator built on Unreal Engine 4 for AI research, connected to autonomous drones and ground vehicles with realistic physics and visual cues. AirSim allows software-in-the-loop simulations with Ardupilot and PX4, as well as hardware-in-the-loop with Px4. It also includes C++, Python, Java and C# APIs that allow the interaction of the user programmatically with the vehicle for the extraction of state and sensor information, for providing control inputs or changing the weather effects. Apart from the APIs, the user can interact with the vehicle with external controllers, such as the keyboard, radio controller, steering wheel, pedals, etc. In the case that only visual data has to be retrieved, there is a computer vision mode in which the physics are disabled and the user is in control of a camera. It also provides true object segmentation and depth maps. The architecture of the system can be observed in Figure 4.2 [26].

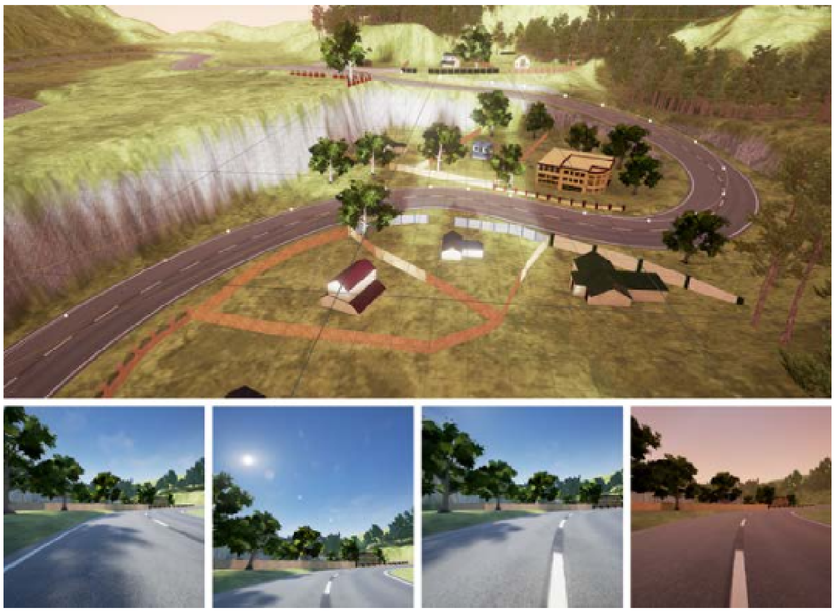


Figure 4.1: Rural 3D environment [25]

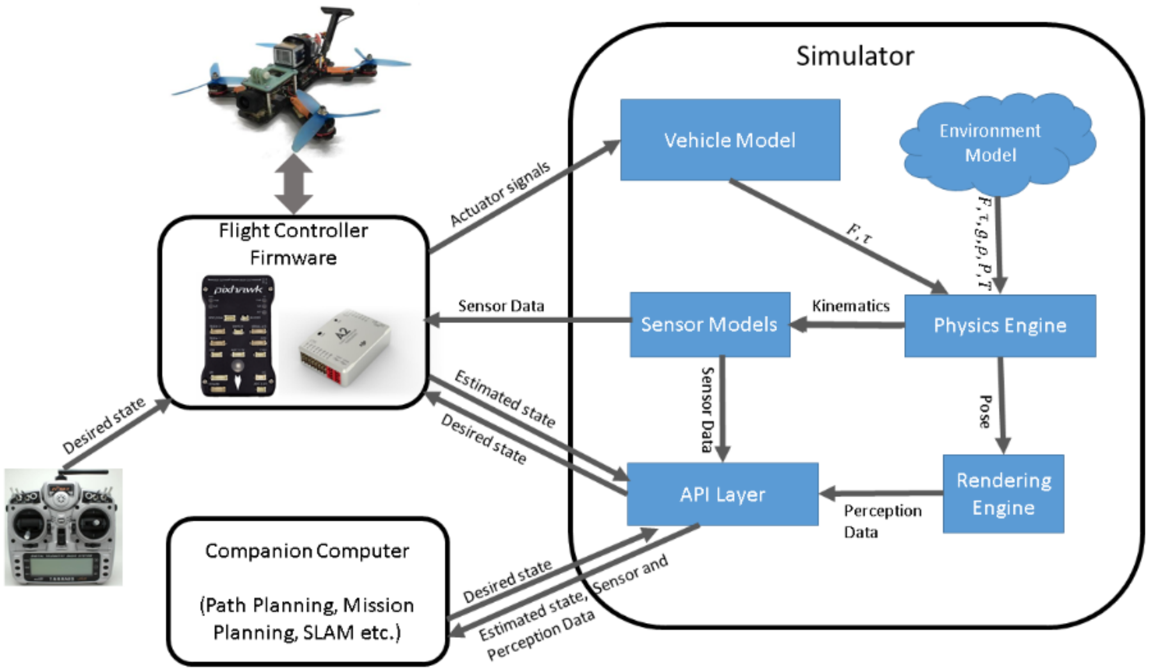


Figure 4.2: AirSim architecture with main components and interactions [26].

AirSim already provides 11 ready-to-use environments [1] but the researcher can buy or freely obtain any environment from the UE4 Marketplace and integrate AirSim. Unfortunately, the AirSim provided environments already have pre-compiled binaries and they can not be modified; the user can only interact with the vehicle through APIs. Regarding the vehicle's modularity, its rigid body physics and sensors as well as its appearance can be modified.

[1]https://github.com/Microsoft/AirSim/releases

A promising tool for the generation of real environments such as cities is CityEngine [2]. It allows the definition of a geographical area and its translation to UE4 format. Having created the environment, AirSim can be easily integrated and the drone can be trained with synthetic flight data around a real geographical location. Besides that, although the original AirSim version offered sensor models for gyroscopes, accelerometers, barometers, magnetometers, GPS and cameras, the developers and community are working towards new sensors such as LIDAR or radar.

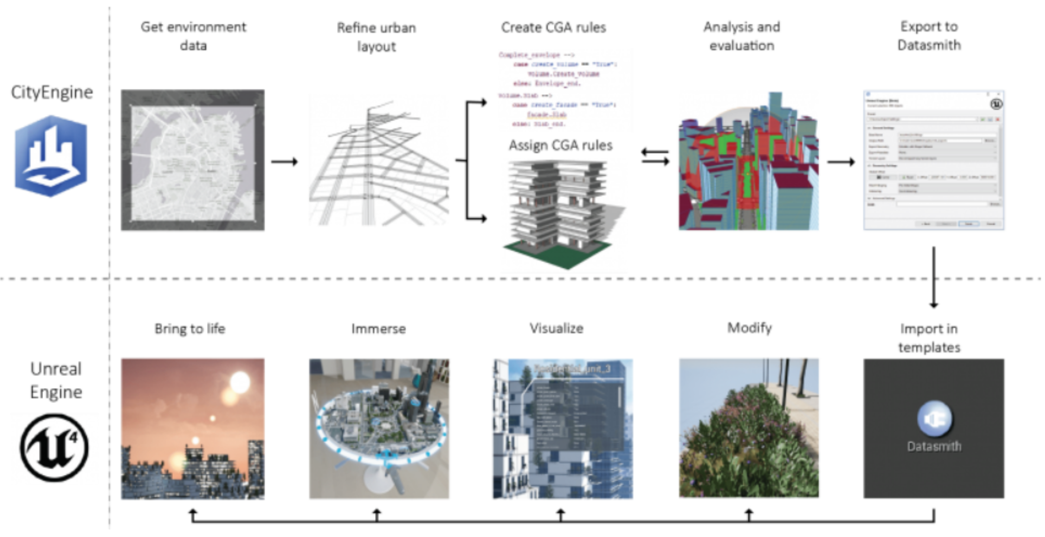


Figure 4.3: CityEngine to Unreal workflow [3].

Interesting extensions for this project include a wrapper for generating event-based camera information [4] or a voxel grid construction of the UE4 environment [5]. The latter would allow the definition of environment obstacles and the automatic generation of feasible drone flight paths for training data generation. Finally, AirSim has extensive documentation and tutorials [6].

In 2020, Microsoft, Stanford and ETH developed the AirSim Drone Racing Lab [213], a simulation framework based on AirSim for drone racing which hosted the NeurIPS 2019 simulation based drone racing competition. For that purpose, it extended the functionalities of the original AirSim with, for example, new input modalities, such as optical flow or APIs that provide the environment ground truth.

From 2017, UnrealCV [27] is another extension of UE4 for computer vision research which consists of a set of Python and Matlab plugins for interaction with the virtual world, allowing the generation of new environments and the communication with machine learning programs such as or OpenAI Gym for visual reinforcement learning applications. Apart from the camera image, UnrealCV provides the true depth map, segmentation and surface normal direction, as well as stereoscopic vision. Unfortunately, in contrast with the other robotic simulators, it does not include vehicles, such as UAVs. As a result, vehicle physics, renders and interaction through APIs have not been developed.

The researcher can easily obtain information about the objects in the environment, such as their location, by sending UnrealCV commands to the virtual world. It works thanks to a client-server communication using TCP, in which the server uses the C++ UE4 API to obtain information from the environment and the client interacts with the server sending commands and receiving information. This communication can be observed in Figure 4.4 [27].

[2] `https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview`
[3] Source: `https://www.youtube.com/watch?v=vMAVRnEWPEU`
[4] `https://github.com/microsoft/AirSim/pull/3202`
[5] `https://github.com/microsoft/AirSim/pull/3209`
[6] `https://microsoft.github.io/AirSim/`

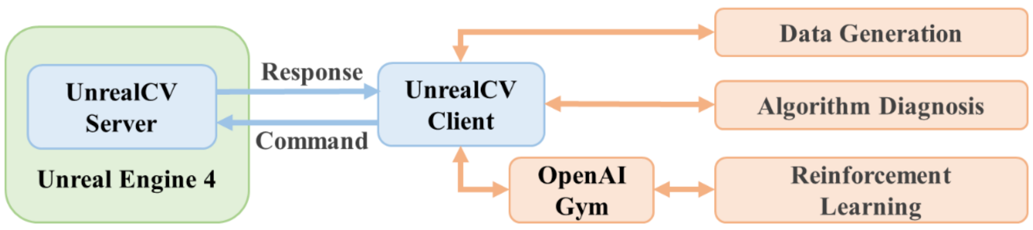


Figure 4.4: UnrealCV server-client communication [27]

As AirSim, UnrealCV releases six default virtual worlds in their pre-compiled binaries for data generation. If the researcher requires the modification of those environments, they would have to be purchased in the UE4 Marketplace. Besides that, UnrealCV has extensive documentation and encourages its community to contribute to UnrealCV and share its virtual works through their ”model zoo” platform. However, the latest release is from 2017.

In 2018, another simulator based on the Unreal Engine was released open-source, Sim4CV [214]. Its main added value is the creation of Python, C++ and Matlab Socket interfaces, multi-object logging, replay system and a simple interface for the drag-drop outdoor world generation. Unfortunately, the world generation tool only includes 2 default maps: 1 race drone track for deploying UAVs and 1 rural neighbourhood environment for deploying ground vehicles. Besides that, the tool only accepts .trk map files which differ from the .umap files generated by the UE4. Therefore, external software is required to generate an initial template and objects, and it is not possible to exploit the maps available at the UE4 Marketplace.

Finally, the Sim4CV team was planning to release open-source a developer version of their product in 2018. However, by February 2021 that has not been the case and the user is limited to the default environments, vehicles, physics and sensors. As a result, there is no ”live” version of the product or community around it. The original developers of Sim4CV are not all part of KAUST anymore, which could mean that the project development has stagnated.

In 2020, NVIDIA launched Isaac Software Development Kit (SDK) and Isaac Sim [7], a toolbox for the development and deployment of artificial learning applied to robotics in the Omniverse simulator environment of NVIDIA [8]. Apart from providing high-fidelity photo-realistic environments, it allows online collaboration, it can be connected to other software (e.g. Blender or UE4) through plugins and it includes state-of-the-art navigation and perception algorithms. In contrast with the other simulators based on EU4, Omniverse requires a minimum available storage of 500 GB and a RTX GPU; requirements beyond the specifications of most workstations. Besides that, Isaac and Omniverse are still an early access product in the beta phase released in 2020.

At the end of 2020, the Robotics and Perception Group released Flightmare [215], a modular quadrotor simulator that allows the user to perform a trade-off between accurate dynamics and photo-realistic rendering using Unity by decoupling both components. This allowed to speed up the simulation with parallel programming. Apart from RGB visual information from the camera, it provides depth, semantic segmentation and 3D point-clouds of the environment. Besides that, reinforcement learning tasks can be connected to the OpenAI Gym framework through a Python wrapper and the simulator allows the interaction with the Oculus virtual-reality headset.

Flightmare is released open-source with 5 default environments, but the user can create or purchase more in the Unity Asset Store, system analogous to the UE4 Marketplace. The simulator also offers C# scripts that allow the interaction of the user with the environment, vehicle and simulation. Regarding the modelling of the dynamics, Flightmare has a flexible framework that allows the user to choose between 3 quadrotor dynamics for different fidelity, purpose and computation capability.

[7]`https://developer.nvidia.com/isaac-sim`
[8]`https://www.nvidia.com/en-us/design-visualization/omniverse/`

Being an open-source live project, the creators aim at building a community that contributes to its future maintenance and development. Besides that, documentation is available but multiple sections are still under construction.

Finally, Flightmare was inspired by FlightGoggles [216], another photo-realistic simulator developed by MIT for high-speed drone races which decouples vision from physics. FlightGoggles also proposes a different simulation paradigm known as virtual reality in which the simulated environment synthetic visual information is fed to a vehicle flying in a real test environment within a motion capture system. In this way, real drone dynamic information is obtained while flying in different environments.

Table 4.1 summarises the characteristics of the main photo-realistic simulators mentioned in this document. As can be observed, from the functionality and support, AirSim seems to be the most suitable option for the development of a data collection pipeline. However, lessons can be learn from Flightmare in terms of decoupling the dynamics and the vision in order to develop an accurate drone simulator. Also, FlightGoggles virtual reality could be an alternative approach.

Table 4.1: Simulators comparison

	AirSim [26]	**UnrealCV** [27]	**Sim4CV** [214]	**Flightmare** [215]
Photo-realistic	Yes	Yes	Yes	Yes
Dynamics	PhysX UE4	PhysX UE4	PhysX UE4	Flexible Unity
Environments	Default maps Unlimited market	Default maps Unlimited market	Default maps Limited creation	Default maps Unlimited market
Vehicles	Adaptable	None	Not adaptable	Adaptable
Vehicle physics	Yes	None	Yes (Poor)	Yes
API/plugins	C++ Python	Matlab Python	C++ Matlab Python	C++ C# Python
Documentation	Extensive	Extensive	Poor	Medium
Community	Active	Active	None	Medium
C&S preliminary work	Yes	No	No	No
Default camera views	FPV Depth Segmentation	FPV Depth Segmentation Surface normal Stereoscopic	FPV	FPV Depth Segmentation 3D point cloud
Last version	2021	2017	2017	2020

Although Blender's Game Engine was also used for the development of simulators, such as MORSE [217] in 2011-2016, UE4 and Unity have become the new state-of-the-art. Besides the aforementioned photo-realistic frameworks, the research community has been using other alternatives for quadrotor simulation. Gazebo has been one of the main choices due to its high-fidelity physics engine, leading to simulators such as RotorS [218] and Hector [219]. The faculty of Aerospace Engineering at TU Delft has been using the Paparazzi autopilot within the Gazebo environment for its research. The DroneSimLab [220] defines a different approach which aims at connecting multiple simulation engines (including UE4) in order to exploit their individual benefits.

5 Literature study conclusion

In this document a literature study has been presented for the Master thesis called ”Vision-based UAV Fault Detection and Diagnosis Framework”. First, the motivation for this line of research was introduced with the research questions and timeline of the project. After that, a general outline of all fault detection and diagnosis algorithms available in literature was discussed, followed by a detailed explanation of the most important knowledge-based approaches, with especial focus on supervised and unsupervised learning. Then, 3 key application of visual data in modern robotics were discussed, since they highlight different methods of visual data processing. Finally, a summary of the most prominent photo-realistic simulators was performed.

From literature it is clear that visual information from the camera sensor, ubiquitous in most robotic systems, has not been exploited for fault detection and diagnosis. It has been constrained to the anomaly detection of signals from accelerometers, gyroscopes or other sensors that add weight to the vehicle. Although it could be argued that the processing of image data could be computationally expensive when compared to other FDD alternatives, this kind of information is already been processed by vehicles for navigation or state estimation. As a result, there exists the potential of tapping into the pipeline of these applications of visual information in order to improve the performance of fault detection and diagnosis systems without increasing the computational load, heavily constrained in most commercial UAVs.

Potential FDD frameworks could exploit end-to-end machine learning with combinations of convolutional and recurrent neural networks. Additionally, Bayesian inference could be incorporated to the framework in order to determine the uncertainty of the estimation, empowering better informed decisions. An alternative and novel approach is the implementation of attention mechanisms instead of the (convolutional and) recurrent neural networks, having the potential of being more accurate with a lower computational load. Finally, instead of tapping directly to the pixels, lessons could be learnt from previous applications of visual information and pre-process the images from the camera sensor before being fed to the FDD algorithm.

In order to train any knowledge-based approach, large quantities of failure and healthy data are required. Unfortunately, the collection of real drone data in which different failure modes are induced is very expensive and infeasible for the desired quantities of data. Therefore, it will have to be gathered from a simulation environment and, since the camera information is the centre of the researched FDD framework, a photo-realistic simulator is the most-sensitive choice. From the presented options, due to its documentation, support and flexibility, AirSim is the most suitable option at the moment of writing. If in the future Flightmare receives a similar support and the authors develop a comprehensive documentation, it could be a competitive alternative. Another approach to training from simulator data would be a hybrid approach in which a small real dataset is shuffled with a large simulation dataset.

To conclude, FDD is a field which will become of paramount importance when autonomous vehicles will conquer the skies in urban regions. The goal of this master thesis project will be to exploit the rich information provided by cameras, nowadays ubiquitous in modern UAVs, in order to improve the performance of current FDD approaches maintaining a competitive computational load. Since this work is the first of its kind, it will lay the groundwork for other researchers to design visual-based FDD approaches and pave the path for event-based cameras.

III

Scientific papers

6
Scientific Paper 1: Blade Element Theory Model for UAV Blade Damage Simulation

Blade Element Theory Model for UAV Blade Damage Simulation

José Ignacio de Alvear Cárdenas* and Coen de Visser†

From fault-tolerant control to failure detection, blade damage simulation has been an essential tool for the development and testing of failure resilient modern unmanned aerial vehicles before their entry into service. Current approaches assume partial loss of rotor effectiveness or reduce the problem to the centrifugal forces caused by the shift in the propeller centre of gravity. This work proposes a white-box blade damage model based on Blade Element Theory which combines the emerging mass and aerodynamic effects of blade damage. The model serves as plug-in to the nominal system model, enables the simulation of any degree of blade damage and does not require costly experimental data from failure cases. A complementary methodology for the identification of the airfoil lift and drag coefficients is also presented. Both contributions were demonstrated with the Bebop 2 drone platform and validated with static test stand wrench measurements obtained at 3 levels of blade damage (0%, 10%, 25%) from a dedicated wind tunnel experimental campaign with velocities up to 12 m/s. Results show high accuracy when simulating a healthy propeller. In the presence of blade damage, at high propeller rotational speeds the model shows a relative error between 5% and 24%. At low propeller rotational speeds, the relative error oscillates between 15% and 75%.

I. Nomenclature

BD	=	Blade damage, %
BL, BS	=	Blade and blade section
C_d, C_l	=	Airfoil drag and lift coefficients
c_c, c_r, c_t	=	Longest chord length, root chord and tip chord, m
D	=	Drag force, N
$\overrightarrow{d}$	=	Conversion matrix from rotational rates to linear velocities
d_r	=	Blade section length, m
F	=	Force, N
g	=	Gravitational acceleration, m/s^2
h	=	Trapezoid height, m
i_p	=	Propeller incidence angle, °
k_x, k_y	=	Linear inflow weighting factors
L	=	Lift force, N
l, b	=	Distance from the propeller centre of rotation to the body coordinate frame x- and y-axes, m
M	=	Moment, Nm
m	=	Mass, kg
$\dot{m}$	=	Mass flow, kg/s
N	=	Number of data samples
n_b, n_{bs}	=	Number of blades and blade sections
n_t	=	Number of trapezoids in which a blade is divided
P	=	Propeller
Q	=	Torque, Nm
R	=	Propeller radius, m
r_{CG}	=	Distance between the propeller centre of rotation and centre of gravity, m
T	=	Thrust, N
V	=	Linear velocity, m/s
V_A	=	True airspeed, m/s

*Graduate Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology
†Associate Professor, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology

V_R	=	Airspeed at the rotor, m/s
V_w	=	Wind speed, m/s
v_0, v_i	=	Uniform and linear induced velocities, m/s
$\bar{y}_c$	=	Span-wise centroid location, m
α	=	Angle of attack, rad
α_d	=	Angle of attack of the rotor disk relative to the oncoming flow, rad
γ	=	Gradient-descent optimisation learning rate
ϵ	=	Model error
ζ	=	Rotation direction boolean
θ	=	Pitch angle, rad
θ_{tw}	=	Blade twist rate per rotor radius, rad/mm
λ_j	=	Angle between the blade j with its propeller's x-axis
μ_x	=	Tip speed ratio or advanced ratio
ξ_{jk}	=	Damage indicator boolean
ρ	=	Air density, kg/m^3
σ	=	Standard deviation
ϕ	=	Roll angle, rad
χ	=	Wake skew angle, rad
ψ	=	Yaw angle, rad
ψ_k	=	Blade section azimuth angle, rad
ω	=	Propeller rotational speed, rad/s
Ω	=	Vehicle angular velocity, rad/s

II. Introduction

Fault is defined as "*an unpermitted deviation of at least one characteristic feature of the system from the acceptable, usual, standard condition*" [1], reducing its capability of performing a required task. Failure and malfunctions are the result of the accumulation of one or more faults that lead to the permanent interruption or intermittent irregularity in the performance of a system function under the specified operating conditions.

Depending on where in the UAV they take place, failures can be classified as sensor faults, actuator faults and plant faults [2], being the first two groups those that most literature aim at predicting. On the one hand, sensor faults result from incorrect readings from the system instruments and sensors, and they include constant bias faults (stuck sensor), drift fault (additive-type), constant gain faults (multiplicative-type) and outlier faults [3, 4]. On the other hand, actuator failures are caused by total loss or degradation of the propeller, motor or electronic speed controllers [5] and they can be classified in four categories, namely actuator saturation, actuator lock, actuator fly-off and propeller damage. Propeller damage is the most challenging actuator failure to simulate. Whereas other propeller failure modes are symmetrical failures around the rotation axes in which the rotational velocity of the propeller is fixed at a certain value or the complete propeller has flown off, propeller damage caused by the chipping or breaking of a blade leads to asymmetrical forces and moments acting on the system that go beyond the change in thrust.

In order to improve the resilience of multi-rotor and hybrid unmanned aerial vehicles (UAVs) to potential failures, work is carried out in multiple fronts, e.g. obstacle avoidance [6], upset recovery [7], fault-tolerant control [8–10] or fault detection and diagnosis [11]; the latter consisting of the fault classification, as well as its location and magnitude identification. For all these tasks, researchers use models to simulate their systems and failures for the training or testing of their approaches before deployment. Due to the complexity of aerospace systems, gray- or black-box models are usually obtained through system identification [12]. As a result, simulations are constrained to the failure cases within the flight envelope of the collected data, which is usually obtained from costly wind tunnel experimental campaigns. Additionally, acquiring enough data for system identification from highly damaged cases is challenging due to concerns regarding the operator's safety and system survivability upon a potential system's loss of control. Hence, the range of captured failures that later can be simulated is limited and it is based on the interpolation and extrapolation of the few experimentally tested scenarios.

Previous literature in the field of fault diagnosis have exploited simplifications of the simulation of blade damage. Avram et al. [13] consider quadrotor actuator faults, such as structural damage to the propellers or degradation of the rotors, as partial loss of effectiveness — a partial loss of thrust generated by the damaged rotor. This is simulated by multiplying the commanded rotor angular velocity by a factor lower than one in order to obtain the "true" rotor angular velocity. The main drawback of this approach is that vibrations in the system due to the unbalance of forces and moments are ignored.

Another approach is proposed by Ghalamchi et al. [14], which introduce sinusoids in the force signals to simulate the vibrations caused by the propeller unbalance. The sinusoids only consist of the decomposition of the centrifugal force in the x and y components caused by the displacement of the propeller centre of gravity due to blade damage. Unfortunately, this approach does not consider the vibrations in the moment signals, as well as the vibrations caused by the changed aerodynamics due to the displacement of the centre of pressure.

The development of more accurate blade damage models could contribute to the creation of more realistic simulations that will foster the potential discovery of emerging subtle data features able to improve the current UAV on-board failure detection and diagnosis capabilities. A technique that has been used for the modelling of forces and moments in helicopters [15], UAVs [16–18] and wind turbines [19] is Blade Element Theory (BET). Here, the propeller is discretised radially into a finite number of segments of length δr, each producing a differential thrust and torque. BET is based on the assumption that the wrenches generated by a (rotor) blade can be computed by the addition of the individual contributions of each of its span-wise elements. For this purpose, 2D airfoil characteristics are exploited whereas 3D effects are ignored. Previously, this approach has been used to model propeller thrust [20]. However, it has never been explored for blade damage modelling.

In this paper, a white-box blade damage simulation model based on Blade Element Theory that complements the identified healthy UAV model is proposed, implemented and validated. To this end, the developed approach provides the difference in forces and moments with respect to the nominal system. In contrast with existing methods, the effects from both shifts in the centres of gravity and pressure are considered. The approach allows the injection of any level of failure without the need of added costly and dangerous system identification experiments. To the authors' knowledge, this is the first time BET is used for UAV blade damage simulation and the first time mass and aerodynamic effects are modelled together in order to shift research towards more realistic white-box blade damage models. Furthermore, this paper also presents a method for identifying the (mostly unknown) UAV blade lift and drag curves with respect to the angle of attack using BET, an approach never tried before in literature.

The proposed model has been applied to the Parrot Bebop® 2 UAV and it has been validated by comparing its predictions to the wrench signals of a damaged propeller at multiple degrees of failure. The validation data was gathered during a dedicated wind tunnel experimental campaign at the Open Jet Facility at Delft University of Technology which allowed the controlled variation of environmental variables such as the wind speed (between 0 and 12 m/s) and the propeller incidence angle (between 0 and $\pi/2$ rad).

The paper is organised as follows. First section III, section IV and section V describe the methodology, where the first two explain the mass and aerodynamic effects and the third the identification of the propeller lift and drag curves. The flow of these computations and the complete approach is illustrated in advance for the reader in Fig. 1. Then, the proposed approach is applied to the Bebop 2 UAV in section VI to show its potential in characterising blade damage on a real platform. Next, section VII validates the results with static test stand wind tunnel experiments, highlighting some limitations of the proposed model and test set-up. Finally, section VIII presents the conclusions of this research and recommendations for future work.

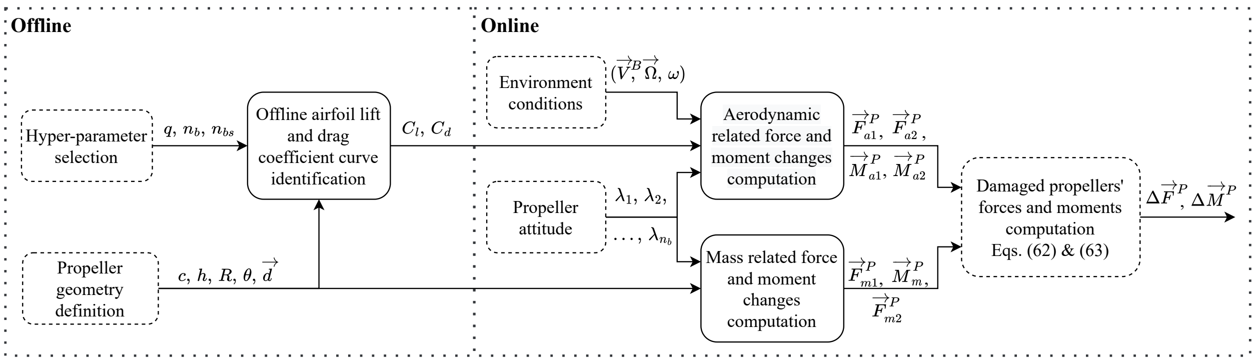

Offline
Online
Hyper-parameter selection
q, n_b, n_{bs}
Offline airfoil lift and drag coefficient curve identification
C_l, C_d
Environment conditions
$(\vec{V}^B, \vec{\Omega}, \omega)$
Aerodynamic related force and moment changes computation
$\vec{F}^P_{a1}$, $\vec{F}^P_{a2}$, $\vec{M}^P_{a1}$, $\vec{M}^P_{a2}$
Propeller attitude
λ_1, λ_2, $\ldots$, λ_{n_b}
Damaged propellers' forces and moments computation Eqs. (62) & (63)
$\Delta\vec{F}^P$, $\Delta\vec{M}^P$
Propeller geometry definition
c, h, R, θ, $\vec{d}$
Mass related force and moment changes computation
$\vec{F}^P_{m1}$, $\vec{M}^P_m$, $\vec{F}^P_{m2}$

Fig. 1 Flowchart of the damaged propeller offline and online computations. The lift and drag coefficient curves identification takes place offline, whereas the computation of forces and moments due to propeller damage are performed online. The blocks with a solid edge line are further expanded in the methodology sections.

III. Mass effects

This section discusses the moments and forces that emerge from the change in propeller mass and its corresponding shift in its centre of gravity for a single propeller with blade damage. They are all measured in the propeller's reference frame, which is the equivalent to the body reference frame translated to the centre of the propeller's hub.

The first forces to be obtained are those caused by the loss of mass. Since the goal is to compute the forces and moments that have to be added to those resulting from the physics model when there is no failure, the force required to be added is in the opposite direction of the gravity vector, as can be seen in Eq. (1). Here, m_{loss} is the lost mass and $\overrightarrow{R}_{PI}$ is the transformation matrix from the inertial to the propeller coordinate frame, which can be seen in Eq. (2). Depending on the drone attitude, the gravity vector can have a value in all three dimensional components of the propeller coordinate frame.

$$\overrightarrow{F}^{P}_{m1} = \overrightarrow{R}_{PI} \begin{bmatrix} 0 \\ 0 \\ -g m_{\text{loss}} \end{bmatrix} \tag{1}$$

$$\overrightarrow{R}_{PI} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{2}$$

Second, the shift in the centre of gravity (CG) causes the appearance of moments around the centre of rotation of the propeller. In order to compute these moments, the arm from the propeller central hub to the new CG location must be computed. For that purpose, the blade has been modelled as a group of trapezoids. As can be observed in Fig. 2, in the case of the Bebop 2 propeller, its blade can be split up in two trapezoids connected at their base which is situated at the location of the largest blade chord. c_r, c_t and c_c are the chords lengths at the root, tip and the location of longest chord, respectively.

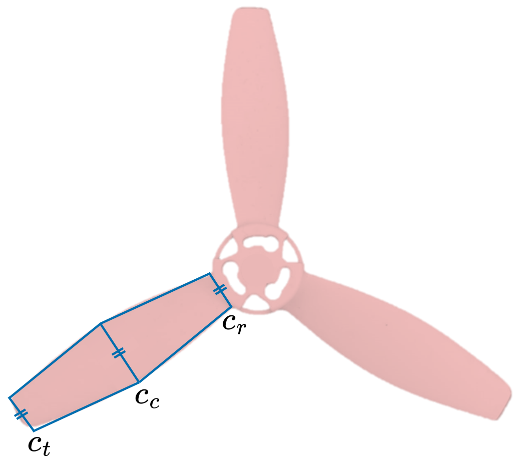


Fig. 2 Bebop 2 propeller top view and trapezoid simplification.

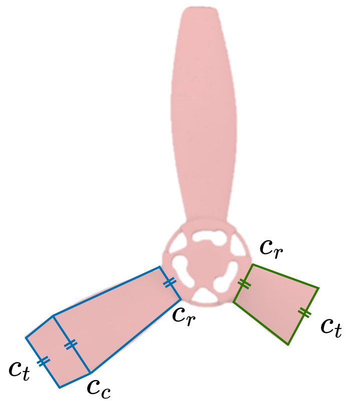


Fig. 3 Damaged Bebop 2 propeller top view and trapezoid simplification.

The CG of each blade is computed separately depending on whether it has damage or not. In the case that there is damage, the tip chord will move along the span of the blade toward the central hub. In the case that damage causes partial blade loss closer to the central hub than the location of c_c, then there would be only one trapezoid in the blade planform and c_c would disappear, as can be seen in Fig. 3.

For the computation of the centre of gravity, it is assumed that the density of the blades is constant, so that the CG will coincide with the centroid of the blade. Hence, the centroid of each trapezoid is computed with Eq. (3) and they are weighted together with their respective areas with Eq. (4). n_t stands for the number of trapezoids within a blade and Fig. 4 illustrates the trapezoid geometrical variables.

$$\bar{y}_{c_{\text{trapezoid}_i}} = \frac{h_i}{3}\frac{2c_{i+1}+c_i}{c_{i+1}+c_i} \tag{3}$$

$$\bar{y}_{\text{CG}_{BL}} = \bar{y}_{c_{BL}} = \frac{\sum_i^{n_t} \bar{y}_{c_{\text{trapezoid}_i}}(c_i+c_{i+1})h_i/2}{\sum_i^{n_t}(c_i+c_{i+1})h_i/2} \tag{4}$$

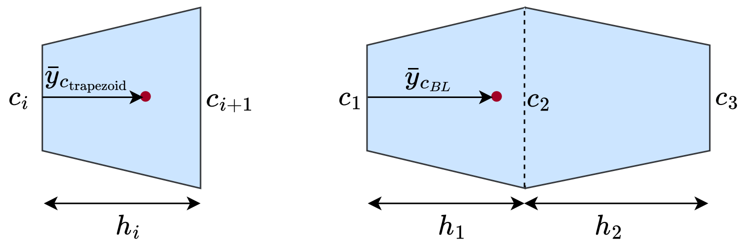


Fig. 4 Blade and trapezoid geometry and centroid.

The centroids of each of the blades are weighted with their respective areas in order to find the centroid of the complete propeller. With the location of the CG computed as in Eq. (5) and the gravitational force calculated in Eq. (1), the moments caused by the gravity force are computed in Eq. (6).

$$\vec{r}^{\,P}_{CG} = \begin{bmatrix} x^P_{CG} \\ y^P_{CG} \\ 0 \end{bmatrix} \tag{5}$$

$$\vec{M}^P_m = \vec{r}^{\,P}_{CG} \times \vec{F}^P_{m1} \tag{6}$$

Third, thanks to the shift of the centre of gravity and the rotation of the propeller, a centrifugal force is created. The magnitude of the centrifugal force is computed with Eq. (7), where m_P is the propeller mass, ω is the rotational velocity and r_{CG} is the distance between the centres of rotation and gravity, as computed in Eq. (8).

$$F_{m2} = m_P\omega^2 r_{CG} \tag{7}$$

$$r_{CG} = \sqrt{x^2_{CG} + y^2_{CG}} \tag{8}$$

The centrifugal force is later decomposed in the x and y components, leading to the vector shown in Eq. (9). θ_{CG} is the angle that $\vec{r}_{CG}$ creates with the propeller coordinate frame and is computed with Eq. (10). This centrifugal force is illustrated in Fig. 5.

$$\vec{F}^P_{m2} = \begin{bmatrix} F_{m2}\cos\theta_{CG} \\ F_{m2}\sin\theta_{CG} \\ 0 \end{bmatrix} \tag{9}$$

$$\theta_{CG} = \arctan\frac{y^P_{CG}}{x^P_{CG}} \tag{10}$$

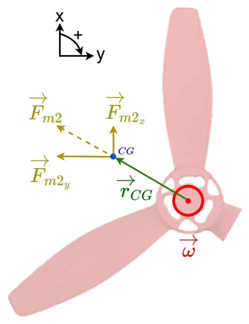
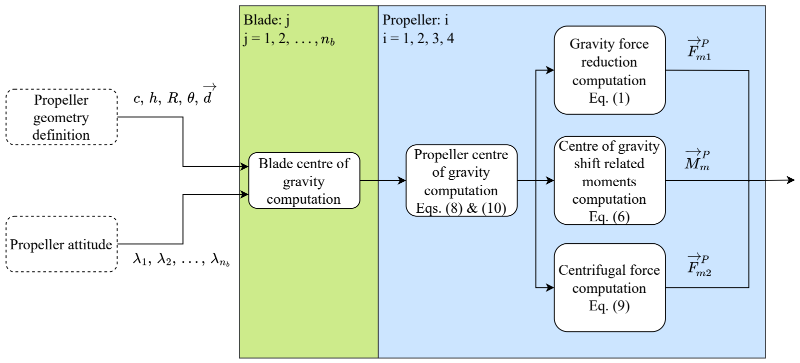


Fig. 5 Centrifugal force of a damaged propeller.



Blade: j
j = 1, 2, . . . , n_b
Propeller: i
i = 1, 2, 3, 4
Propeller geometry definition
$c, h, R, \theta, \vec{d}$
Propeller attitude
$\lambda_1, \lambda_2, \ldots, \lambda_{n_b}$
Blade centre of gravity computation
Propeller centre of gravity computation Eqs. (8) & (10)
Gravity force reduction computation Eq. (1)
Centre of gravity shift related moments computation Eq. (6)
Centrifugal force computation Eq. (9)
$\vec{F}^P_{m1}$
$\vec{M}^P_m$
$\vec{F}^P_{m2}$

Fig. 6 Flowchart of the computation of the damaged propeller mass related forces and moments at a single time step during simulation.

The discussed computation of the damaged propeller mass related forces and moments is illustrated in Fig. 6 with a flowchart.

IV. Aerodynamic effects

This section discusses the moments and forces that emerge from the change in aerodynamics for a single propeller upon blade damage. For their computation, the Blade Element Theory mathematical process is exploited. This method discretises the blade along its span in sections of equal length, determines their individual generated moments and forces, and adds all of them in order to obtain those generated by the complete propeller. When a blade is damaged, the forces and moments that would have been generated by the missing blade sections would be subtracted from those computed by the physics model in the healthy state.

A. Blade Element Theory

The goal of the BET method is the computation of the thrust and torque generated by the complete blade through the sum of the contributions of all its sections. For that purpose, the lift and drag equations, which can be seen in Eq. (11) and Eq. (12), are applied to each of the blade sections k of span length equal to dr. For the rest of the paper, the subscript i stands for the propeller, j for the blade and k for the blade section.

$$\Delta L_k(r_k,\psi_k) = \frac{1}{2}C_{l_k}(\alpha_k(r_k,\psi_k))\rho V_{A_k}^2(r_k,\psi_k)c_k(r_k)dr \tag{11}$$

$$\Delta D_k(r_k,\psi_k) = \frac{1}{2}C_{d_k}(\alpha_k(r_k,\psi_k))\rho V_{A_k}^2(r_k,\psi_k)c_k(r_k)dr \tag{12}$$

ρ is the air density which depends on the altitude at which the drone flies with respect to the sea level. Furthermore, c_k is the blade section average chord. V_{A_k} is the airspeed seen by the blade section perpendicular to its span and C_{l_k} and C_{d_k} are the lift and drag coefficients of the 2D blade airfoil, respectively. As can be seen, these parameters are a function of the angle of attack (α_k), the distance from the blade section centroid to the centre of rotation (r_k) and the blade section azimuth angle (ψ_k). The last one is an angle measured on the propeller plane and it is defined to have a value of zero degrees (ψ=0) in the direction of the drag, increasing its value in the direction of rotation. The r and ψ definitions can be visualised in Fig. 7 and Fig. 8. When a variable is a function of r and ψ, it will be represented by $(\cdot)$ for readability purposes.

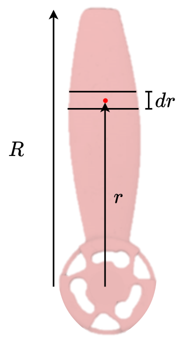


Fig. 7 Blade geometrical parameters.

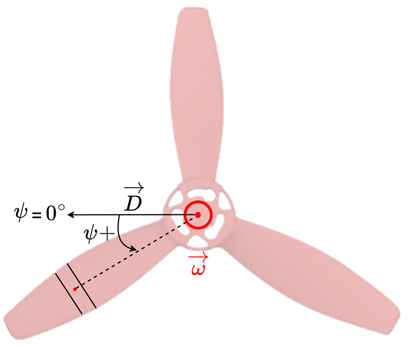


Fig. 8 Azimuth angle visualization.

The airspeed at each blade section (V_{A_k}) has to be computed taking into account three main components: the combined linear and angular velocities, the propeller rotational velocity and the induced velocity. First, Eq. (13) is used to compute the linear velocity of the propeller assembly ($\vec{V}^P$) from the body linear ($\vec{V}^B$) and angular velocities ($\vec{\Omega}$). The $\vec{d}$ matrix presented in Eq. (14) is used to convert the rotational rates of the vehicle to linear velocities, exploiting the known drone geometry shown in Fig. 9 [12]. Each row of the d matrix ($\vec{d}_i$) corresponds to each of the drone propellers. The first row corresponds to the front-left propeller and the following rows to the other propellers moving clockwise from a top-down view of the drone.

$$\vec{V}_i^P = \vec{\Omega} \times \vec{d}_i^T + \vec{V}^B \tag{13}$$

$$\vec{d} = \begin{bmatrix} l & -b & 0 \\ l & b & 0 \\ -l & b & 0 \\ -l & -b & 0 \end{bmatrix} \tag{14}$$

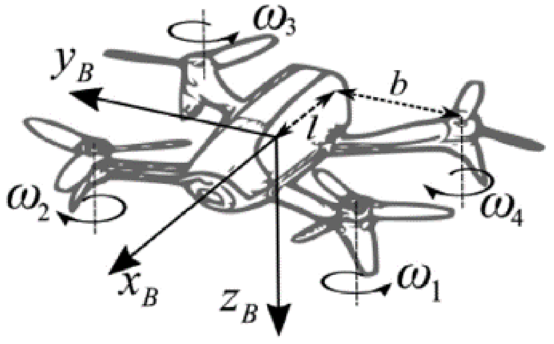


Fig. 9 Drone geometry [12].

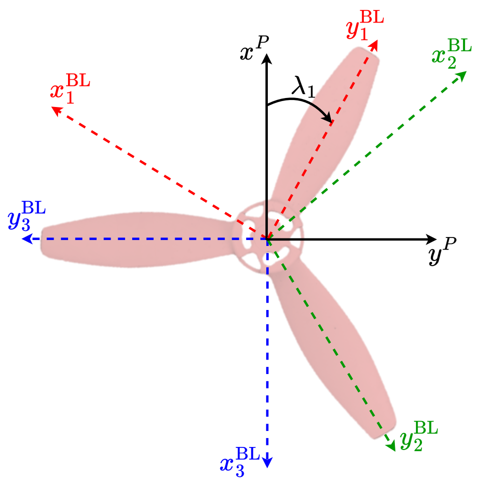


Fig. 10 Blade coordinate frame.

Then, the propeller linear velocity is translated to the blade coordinate frame (BL), which rotates with the respective blade, as can be seen in Fig. 10. The angle of the blade with the propeller x-axis is λ_j and it is used in Eq. (15) for the coordinate frame transformation. As can be observed, a minus sign precedes the transformation matrix because the airspeed vector is opposite to the displacement direction. It is assumed that this value of airspeed, which is a function of the vehicle linear and angular velocities, does not depend on the position along the blade.

$$\vec{V}^{BL}_{A_{ijk_1}} = \vec{V}^{BL}_{A_{ij}} = -\begin{bmatrix} \sin\lambda_j & -\cos\lambda_j & 0 \\ \cos\lambda_j & \sin\lambda_j & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{V}^{P}_{i} \tag{15}$$

Second, the component of the velocity due to the rotation of the propeller is the product of the distance of the blade section centroid to the centre of rotation (r_{ijk}) and the rotational velocity of the propeller (ω_i), as can be seen in Eq. (16). The main benefit of the chosen blade coordinate frame is that this velocity component only exists along the x-axis. ζ_i is a variable which acquires a value of 1 if the i^{th} propeller is rotating clockwise and -1 if it is rotating counter-clockwise.

$$\vec{V}^{BL}_{A_{ijk_2}}(r_{ijk}) = \begin{bmatrix} \zeta_i \omega_i r_{ijk} \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

Third, there exist multiple approaches in literature for computing the induced velocity field across the rotor disk, most of the them based on estimates and empirical tests. The work of Gill et al. [21] assumes ideal propeller geometry, considering a constant uniform induced velocity along the propeller which is mostly not the case in forward flight. The approach followed for the present research is the same one used by Niemiec et al. [22] and that is thoroughly explained by Leishman et al. [23], which combines an initial uniform inflow estimation for the complete propeller with local (blade section) linear inflow model corrections.

For the computation of the uniform induced velocity (v_0*), the Glauert formula presented in Eq. (19) is derived from the combination of the mass flow and the propeller thrust equations, shown in Eq. (17) and Eq. (18), respectively. According to the principles of momentum and energy conservation, the far wake velocity equals the airspeed before the rotor plus two times the induced velocity [23], leading to a change in velocity across the rotor of $\Delta V = 2v_0$.

$$\dot{m} = \rho \pi R^2 V_R \tag{17}$$

$$T = \dot{m}\Delta V = \dot{m}(V_{A_i} + 2v_0 - V_{A_i}) = 2\dot{m}v_0 \tag{18}$$

$$v_0 = \frac{T}{2\rho\pi R^2 V_R} \tag{19}$$

*Since it will be constantly referred to the same single propeller, the subscript i to denote a specific propeller is dropped for the rest of the paper for readability purposes.

Given that the airspeed at the rotor (V_R) equals the propeller airspeed plus the induced velocity along the z-axis direction, it can be expressed as shown in Eq. (20) using the translational velocity found in Eq. (13). The final Glauert equation does not have a closed form, so the induced velocity can be computed using an iterative optimisation technique, such as Nelder-Mead. Alternatively, since the induced velocity needs to be computed in simulation at every time step, a tailored and efficient gradient-descent (1D first order derivative) algorithm is discussed in appendix A.

$$V_R = \sqrt{V_x^P V_x^P + V_y^P V_y^P + (-V_z^P + v_0)^2} \tag{20}$$

Once the uniform inflow velocity is obtained, it can be used as the basis for the computation of the linear inflow model. There are multiple estimation models for the computation of the linear inflow, such as those proposed by Howlett [24], Pitt & Peters [25] and White & Blake [26]. However, for the present research, the model developed by Drees [27] will be used since it is one of the best representations when compared to empirical data [23]. Equation (21) models the induced velocity using the uniform inflow as basis and modifying it with the k_x and k_y weighting factors, which are computed in Eq. (22) and Eq. (23), respectively.

$$v_i(\cdot) = v_0(1 + k_x r \cos\psi + k_y r \sin\psi) \tag{21}$$

$$k_x = \frac{4}{3}\frac{(1 - \cos\chi - 1.8\mu_x^2)}{\sin\chi} \tag{22}$$

$$k_y = -2\mu_x \tag{23}$$

χ is the wake skew angle or the angle that the wake creates with respect to the z-axis of the propeller. It is illustrated in Fig. 11 and it is computed using the propeller airspeed and the uniform induced velocity, as can be seen in Eq. (24). Furthermore, μ_x is the tip speed ratio or advanced ratio and it is defined as the airspeed projected on the x-y plane in the propeller coordinate frame normalised with the blade length and propeller rotational velocity (Eq. (25)). The resulting inflow can be observed in Fig. 12, which shows how the induced velocity changes per blade section with respect to its distance to the centre of rotation and the azimuth angle.

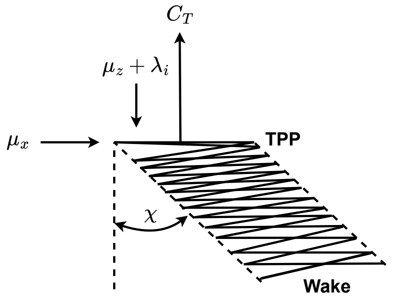


Fig. 11 Illustration of the wake skew angle [23].

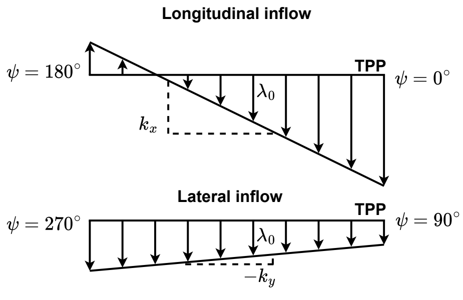


Fig. 12 Linear inflow model [23].

$$\tan\chi = \frac{\sqrt{V_x^P V_x^P + V_y^P V_y^P}}{v_0 - V_z^P} \tag{24}$$

$$\mu_x = \frac{\sqrt{V_x^P V_x^P + V_y^P V_y^P}}{\omega_i R} \tag{25}$$

Having computed the three components of the blade section airspeed for the lift and drag equations, they are summed in Eq. (26).

$$\overrightarrow{V}^{BL}_{A_{jk}}(\cdot) = \overrightarrow{V}^{BL}_{A_{jk_1}} + \overrightarrow{V}^{BL}_{A_{jk_2}}(r_{jk}) + \begin{bmatrix} 0 & 0 & v_{i_{jk}}(\cdot) \end{bmatrix}^T \tag{26}$$

Given the airspeed, the angle of attack seen by the blade can be obtained from Eq. (27). Here, θ_{jk} is the average twist of the blade section and it is a linear function of the distance from the centre of rotation (Eq. (28), where θ_{tw} is the blade twist rate per radius of the rotor and θ_0 is the blade twist at the blade root). The closer to the root, the higher the twist in order to compensate for the lower tangential velocity from the propeller rotation. When the tangential velocity decreases, the velocity along the propeller z-axis has a higher impact on the definition of the angle of attack. In the case of forward flight, that causes a reduction of the angle of attack at the root.

$$\alpha_{jk}(\cdot) = \theta_{jk}(r_{jk}) - \arctan \frac{V^{BL}_{A_{jk_z}}(\cdot)}{V^{BL}_{A_{jk_x}}(\cdot)} \quad (27) \qquad \theta_{jk}(r_{jk}) = \theta_0 - r_{jk}\theta_{tw} \quad (28)$$

Finally, from the blade section lift and drag contributions determined in Eq. (11) and Eq. (12), it is possible to determine the generated thrust (ΔT) and torque (ΔQ) by each blade section with Eq. (29) and Eq. (30), respectively[†]. Here, ϕ_{jk} is the blade section flow angle and it is defined as the difference between the local twist angle θ_{jk} and the local angle of attack α_{jk}, as can be seen in Eq. (31).

$$\Delta T = \Delta L \cos\phi - \Delta D \sin\phi \quad (29) \qquad \Delta Q = r\Delta L \sin\phi + r\Delta D \cos\phi \quad (30)$$

$$\phi_{jk}(\cdot) = \theta_{jk}(r_{jk}) - \alpha_{jk}(\cdot) \quad (31)$$

B. Aerodynamic forces and moments

Once the percentage of blade damage to be simulated is known, the corresponding "lost" blade sections are identified. For instance, if a 30% blade damage is considered in a blade composed of 100 sections, the 30 sections closest to the blade tip are the ones lost. Then, their forces and moments are added to later be subtracted from those of the nominal operating conditions obtained from the healthy blade model output.

Equation (29) can be used for the computation of the thrust of a single blade section. The added thrust value of all the lost blade sections leads to $F^P_{a1_z}$, as given by Eq. (32). ξ_{jk} is a boolean which has a value of 1 when the blade section is damaged and 0 when it is in its nominal state. The moments emerging about the propeller x- and y-axes from the change in the centre of pressure location are computed by decomposing the moment generated by the blade section thrust around the centre of rotation with Eq. (33) and Eq. (34).

$$F^P_{a1_z} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} \xi_{jk}\Delta T_{jk} \quad (32)$$

$$M^P_{a2_x} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} -\xi_{jk}\Delta T_{jk} r_{jk}\sin\lambda_j \quad (33) \qquad M^P_{a2_y} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} \xi_{jk}\Delta T_{jk} r_{jk}\cos\lambda_j \quad (34)$$

Furthermore, Eq. (30) can be used for the computation of the torque of each blade section. The integral torque of the lost blade sections leads to the moment about the z-axis, as shown in Eq. (35). The blade section force in the rotor plane can be obtained by dividing the torque by the magnitude of the moment arm. This force is then decomposed in the x- and y-directions, as shown in Eq. (36) and Eq. (37), in order to obtain the last aerodynamic forces.

$$M^P_{a1_z} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} \xi_{jk}\Delta Q_{jk} \quad (35)$$

$$F^P_{a2_x} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} \xi_{jk}\frac{\Delta Q_{jk}}{r_{jk}}\cos\left(\lambda_j - \zeta\frac{\pi}{2}\right) \quad (36) \qquad F^P_{a2_y} = \sum_{j=1}^{n_b}\sum_{k=1}^{n_{bs}} \xi_{jk}\frac{\Delta Q_{jk}}{r_{jk}}\sin\left(\lambda_j - \zeta\frac{\pi}{2}\right) \quad (37)$$

The sum of the force and moment vectors corresponds to the aerodynamic effects that need to be subtracted from the nominal physics model output. A flowchart visually illustrating the computation of these forces and moments can be seen in Fig. 13.

V. Airfoil lift and drag coefficients identification

Unfortunately, for most commercial drones, an aerodynamic model of the propeller's blade lift and drag does not exist. Previous literature [20] has taken a Hybrid Blade Element Momentum Theory approach for the computation of the induced velocity and aerodynamic coefficients. This approach is based on equating the thrust derived from BET (blade geometry) and the thrust derived from Momentum Theory (interacting flow characteristics). A two step method is

[†]Since every term of these equations is meant for a particular blade section, the subscripts jk have been left out to enhance readability. For the same reason, the variables each of the terms are a function of have also been removed, since they are all a function of (ψ_{jk}, r_{jk}).

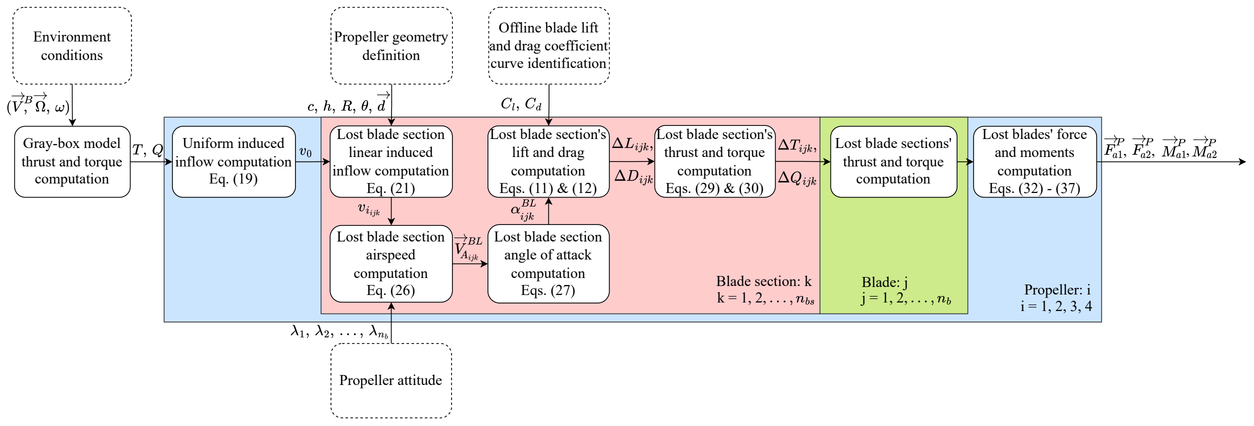

Environment conditions
$(\vec{V}^B, \vec{\Omega}, \omega)$
Gray-box model thrust and torque computation
T, Q
Uniform induced inflow computation Eq. (19)
v_0
Propeller geometry definition
$c, h, R, \theta, \vec{d}$
Lost blade section linear induced inflow computation Eq. (21)
$v_{i_{ijk}}$
Lost blade section airspeed computation Eq. (26)
$\vec{V}^{BL}_{A_{ijk}}$
Offline blade lift and drag coefficient curve identification
C_l, C_d
Lost blade section's lift and drag computation Eqs. (11) & (12)
α^{BL}_{ijk}
Lost blade section angle of attack computation Eqs. (27)
$\Delta L_{ijk}, \Delta D_{ijk}$
Lost blade section's thrust and torque computation Eqs. (29) & (30)
$\Delta T_{ijk}, \Delta Q_{ijk}$
Blade section: k k = 1, 2, ..., n_{bs}
Lost blade sections' thrust and torque computation
Blade: j j = 1, 2, ..., n_b
Lost blades' force and moments computation Eqs. (32) - (37)
Propeller: i i = 1, 2, 3, 4
$\vec{F}^P_{a1}, \vec{F}^P_{a2}, \vec{M}^P_{a1}, \vec{M}^P_{a2}$
$\lambda_1, \lambda_2, \ldots, \lambda_{n_b}$
Propeller attitude

Fig. 13 Flowchart of the computation of the damaged propeller aerodynamic forces and moments at one time step during simulation.

described in which first, the induced velocity is estimated and then, the aerodynamic coefficients. However, this method has two main drawbacks. First, it does not explain how the induced velocity is computed without having previously computed the aerodynamic parameters, since they are required for the computation of the lift and the drag of each blade section in the computation of the thrust from BET. Second, the described optimisation method takes between 2 to 10 hours with only 20 data points. Such long computation times renders the simulation of blade damage scenarios, in which the induced velocity must be computed for every time step, unfeasible. This section will thoroughly explain an alternative optimisation approach for identifying the lift and drag coefficient curves with much shorter computation times.

The proposed method is based on equating the thrust and torque obtained from the healthy UAV model (or experimental data) to the sum of the blade sections' moment and forces computed following BET. To that end, given a drone geometry (R, and l and b from Eq. (14)), combinations of vehicle body linear ($\vec{V}^B$) and angular velocities ($\vec{\Omega}$), as well as propeller rotational velocities (ω), are fed as input to both models. Then, their outputs are combined in order to solve a constrained minimisation problem that finds the lift and drag coefficient functions. C_l and C_d are modelled as polynomials of m and n degree as a function of the angle of attack, as can be seen in Eq. (38) and Eq. (39), respectively.

$$C_{l_{jk}} = x_0 + x_1\alpha_{jk} + x_2\alpha_{jk}^2 + \cdots + x_m\alpha_{jk}^m \quad (38) \qquad C_{d_{jk}} = y_0 + y_1\alpha_{jk} + y_2\alpha_{jk}^2 + \cdots + y_n\alpha_{jk}^n \quad (39)$$

Using these lift and drag coefficient polynomials, the BET thrust and torque can be computed with Eq. (40) and Eq. (41) .

$$\begin{aligned} T &= \left(\sum_j^{n_b}\sum_k^{n_{bs}} \frac{1}{2}\rho C_{l_{jk}} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} dr\right) - \left(\sum_j^{n_b}\sum_k^{n_{bs}} \frac{1}{2}\rho C_{d_{jk}} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} dr\right) \\ &= \frac{1}{2}\rho dr \left[\left(\sum_o^m x_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \cos\phi_{jk}\right) - \left(\sum_o^n y_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \sin\phi_{jk}\right)\right] \end{aligned} \quad (40)$$

$$Q = -\zeta\frac{1}{2}\rho dr \left[\left(\sum_o^m x_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk}\right) + \left(\sum_o^n y_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk}\right)\right] \quad (41)$$

With the previous definitions of the thrust and the torque, it is possible to create a system of the $\vec{A}\vec{x} = \vec{b}$ form. Each pair of rows of the A matrix and the b vector corresponds to the thrust and torque of a data point from the BET and healthy UAV models, respectively. A data point refers to a set of conditions ($\vec{V}^B$, $\vec{\Omega}$, ω) that are provided as input to both models. Equations (42) to (48) show the different components of the system with q data points.

$$l_1^o = \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} \quad (42) \qquad l_2^o = -\sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} \quad (43)$$

$$l_3^o = -\zeta \sum_j^{n_b} \sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} \quad (44) \qquad l_4^o = -\zeta \sum_j^{n_b} \sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} \quad (45)$$

$$\overrightarrow{A} = \frac{1}{2}\rho dr \begin{bmatrix} (l_1^0)_1 & (l_1^1)_1 & \cdots & (l_1^m)_1 & (l_2^0)_1 & (l_2^1)_1 & \cdots & (l_2^n)_1 \\ (l_3^0)_1 & (l_3^1)_1 & \cdots & (l_3^m)_1 & (l_4^0)_1 & (l_4^1)_1 & \cdots & (l_4^n)_1 \\ (l_1^0)_2 & (l_1^1)_2 & \cdots & (l_1^m)_2 & (l_2^0)_2 & (l_2^1)_2 & \cdots & (l_2^n)_2 \\ (l_3^0)_2 & (l_3^1)_2 & \cdots & (l_3^m)_2 & (l_4^0)_2 & (l_4^1)_2 & \cdots & (l_4^n)_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ (l_1^0)_q & (l_1^1)_q & \cdots & (l_1^m)_q & (l_2^0)_q & (l_2^1)_q & \cdots & (l_2^n)_q \\ (l_3^0)_q & (l_3^1)_q & \cdots & (l_3^m)_q & (l_4^0)_q & (l_4^1)_q & \cdots & (l_4^n)_q \end{bmatrix} \quad (46)$$

$$\overrightarrow{x} = \begin{bmatrix} x_0 & x_1 & \cdots & x_m & y_0 & y_1 & \cdots & y_n \end{bmatrix}^T \quad (47)$$

$$\overrightarrow{b} = \begin{bmatrix} (T)_1 & (Q)_1 & (T)_2 & (Q)_2 & \cdots & (T)_q & (Q)_q \end{bmatrix}^T \quad (48)$$

The output of the BET model, namely the $\overrightarrow{A}$ matrix, has to be averaged over a rotation of the propeller, integrating over the azimuth angle [20]. Equation (49) shows how this is done for l_1^0; the same procedure can be applied to l_2^0, l_3^0 and l_4^0. In practice, instead of integrating, a specific discrete number of rotation angles are chosen, and their contributions to the $\overrightarrow{A}$ matrix are computed and averaged.

$$l_1^o = \frac{1}{2\pi} \int_{\psi=0}^{2\pi} \sum_j^{n_b} \sum_k^{n_{bs}} \alpha_{jk}^o(\cdot) c_{jk}(r_{jk}) V_{A_{jk}}^2(\cdot) \cos\phi_{jk}(\cdot)\, d\psi \quad (49)$$

In order to guarantee that the lift and drag curves with respect to the angle of attack have their characteristic recognisable shape, a constrained optimisation problem is posed in order to include airfoil aerodynamic knowledge in the solver. Given that there is a large difference in scale between the thrust and torque values, the Normalised Root Mean Squared Error (NRMSE) is used as objective function (Eq. (50) and Eq. (51)); the standard deviation of the aerodynamic gray-model output (σ_b; the standard deviation of the observations) is the normalisation factor. Instead of computing the error for the thrust and torque equations together and computing the standard deviation of the complete $\overrightarrow{b}$ vector, their errors and their respective observations' standard deviations were computed separately. The figure that the optimisation function aims at minimising is the averaged thrust and torque NRMSE, as can be seen in Eq. (52). Hence, the objective function for the computation of the aerodynamic parameters $\overrightarrow{x}^*$ is defined in Eq. (53).

$$\overrightarrow{\epsilon} = \overrightarrow{b} - \overrightarrow{A}\overrightarrow{x} \quad (50) \qquad \text{NRMSE} = \frac{\sqrt{\left(\overrightarrow{\epsilon}^T \cdot \overrightarrow{\epsilon}\right)/N}}{\sigma_b} \quad (51)$$

$$\text{NRMSE}_{\text{total}} = \frac{\text{NRMSE}_T + \text{NRMSE}_Q}{2} \quad (52) \qquad \overrightarrow{x}^* = \underset{\overrightarrow{x}}{\arg\min}\ \text{NRMSE}_{\text{total}} \quad (53)$$

Furthermore, the following lenient constraints were used to achieve the recognisable shape of the lift and drag coefficient curves:

1) The maximum lift coefficient can not be higher than 5 within the angle of attack range of -30 to 30 degrees: $C_l(\alpha) < 5, \quad \forall\alpha \in [-30°, 30°]$.
2) The lift coefficient curve should have a negative slope within the angle of attack range of 25 to 30 degrees: $dC_l(\alpha)/d\alpha < 0, \quad \forall\alpha \in [25°, 30°]$.
3) The lift coefficient curve should have a positive slope within the angle of attack range of 0 to 7 degrees: $dC_l(\alpha)/d\alpha > 0, \quad \forall\alpha \in [0°, 7°]$.
4) The lift coefficient curve should intersect the angle of attack axis within the angle of attack range of -10 to 10 degrees: $\min C_l(\alpha) < 0, \quad \forall\alpha \in [-10°, 10°]$.
5) The drag coefficient curve can not be negative within the angle of attack range of -30 to 30 degrees: $C_d(\alpha) > 0, \quad \forall\alpha \in [-30°, 30°]$.

For the declaration of these constraints, matrix $\overrightarrow{C}(\overrightarrow{\alpha}) \in R^{s\times(mn)}$ is created with the range of angles of attack mentioned in each constraint definition. The matrix has the same number of rows as integer angles within the constraint range, namely $s = \alpha_{\max} - \alpha_{\min} + 1$ for $\alpha \in [\alpha_{\min}, \alpha_{\max}]$; each row corresponds to an angle of attack. The number of columns equals the length of the parameter vector $\overrightarrow{x}$. Its input is an angle of attack vector which is a function of the $\alpha_{\min}$ and $\alpha_{\max}$, as can be deduced from its definition in Eq. (54) and Eq. (55).

$$\overrightarrow{\alpha}(\alpha_{\min}, \alpha_{\max}) = \begin{bmatrix}\alpha_0 & \alpha_1 & \cdots & \alpha_\eta\end{bmatrix}^T \quad (54) \qquad \alpha_\eta = \alpha_{\min} + \eta, \quad \eta = 0, \ldots, s-1 \quad (55)$$

Since a constraint regarding the lift coefficient curve does not require information about the drag coefficient parameters, the last n columns will be full of zeros (Eq. (56)) for constraints 1 and 4. In the case of the drag constraint (constraint 5), it would be the first m columns that would be full of zeros, as shown in Eq. (57). Constraints 2 and 3 impose a limit in the gradient of the curve, so the derivative of $\overrightarrow{C}_{C_l}$ with respect to the angle of attack is taken in Eq. (58).

$$\overrightarrow{C}_{C_l}(\overrightarrow{\alpha}) = \left[\begin{array}{ccccc|c} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n & \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^n & \overrightarrow{0}_{s\times m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \\ 1 & \alpha_{s-1} & \alpha_{s-1}^2 & \cdots & \alpha_{s-1}^n & \end{array}\right] \quad (56) \quad \overrightarrow{C}_{C_d}(\overrightarrow{\alpha}) = \left[\begin{array}{c|ccccc} & 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n \\ \overrightarrow{0}_{s\times n} & 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^n \\ & \vdots & \vdots & \vdots & \cdots & \vdots \\ & 1 & \alpha_{s-1} & \alpha_{s-1}^2 & \cdots & \alpha_{s-1}^n \end{array}\right] \quad (57)$$

$$\frac{d\overrightarrow{C}_{C_l}(\overrightarrow{\alpha})}{d\alpha} = \left[\begin{array}{ccccc|c} 0 & 1 & 2\alpha_0 & \cdots & m\alpha_0^{m-1} & \\ 0 & 1 & 2\alpha_1 & \cdots & m\alpha_1^{m-1} & \overrightarrow{0}_{s\times m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \\ 0 & 1 & 2\alpha_{s-1} & \cdots & m\alpha_{s-1}^{m-1} & \end{array}\right] \quad (58)$$

The $\overrightarrow{C}$ matrix or its derivative is multiplied with the parameter vector $\overrightarrow{x}$ and the maximum or minimum value from the output is taken for the definition of the inequality constraints. As a result, the following constrained optimisation problem is posed:

$$\begin{aligned} \min_{\overrightarrow{x}} \quad & \text{NRMSE}_{\text{total}} \\ \text{s.t.} \quad & \max\left(\overrightarrow{C}_{C_l}(\overrightarrow{\alpha})\overrightarrow{x}\right) - 5 < 0, \ (\alpha_{\min}, \alpha_{\max}) = (-30, 30) \\ & \max\left(\frac{d\overrightarrow{C}_{C_l}(\overrightarrow{\alpha})}{d\alpha}\overrightarrow{x}\right) < 0, \ (\alpha_{\min}, \alpha_{\max}) = (25, 30) \\ & \min\left(\frac{d\overrightarrow{C}_{C_l}(\overrightarrow{\alpha})}{d\alpha}\overrightarrow{x}\right) > 0, \ (\alpha_{\min}, \alpha_{\max}) = (0, 7) \\ & \min\left(\overrightarrow{C}_{C_l}(\overrightarrow{\alpha})\overrightarrow{x}\right) < 0, \ (\alpha_{\min}, \alpha_{\max}) = (-10, 10) \\ & \min\left(\overrightarrow{C}_{C_d}(\overrightarrow{\alpha})\overrightarrow{x}\right) > 0, \ (\alpha_{\min}, \alpha_{\max}) = (-30, 30) \end{aligned} \quad (59)$$

Figure 14 visually illustrates the lift and drag coefficient identification with a flowchart.

VI. Results

The department of Control & Simulation at Delft University and Technology developed in 2019 a gray-box aerodynamic model of the Bebop 2 drone based on wind tunnel experiments [28]. To demonstrate the presented methodology, it will be used as the healthy UAV identified model, turning the Bebop 2 into the platform of choice.

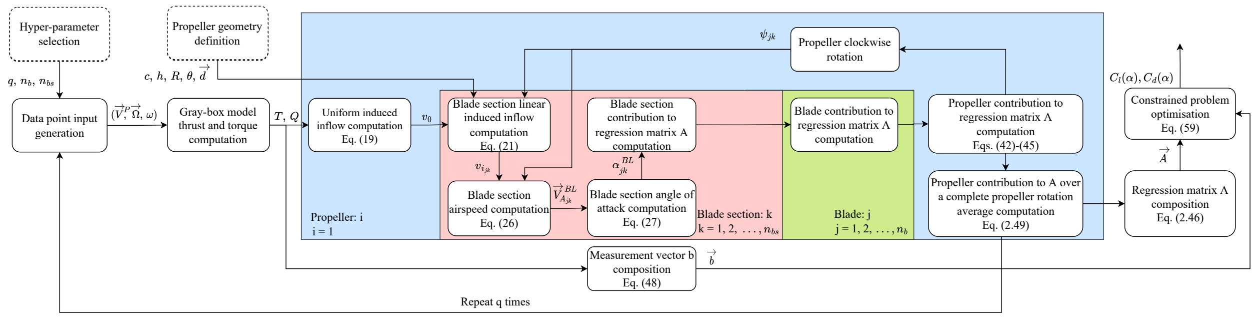

Hyper-parameter selection
Propeller geometry definition
q, n_b, n_{bs}
c, h, R, θ, $\vec{d}$
Data point input generation
$(\vec{V}^P, \vec{\Omega}, \omega)$
Gray-box model thrust and torque computation
T, Q
Uniform induced inflow computation Eq. (19)
v_0
Blade section linear induced inflow computation Eq. (21)
$v_{i_{jk}}$
Blade section airspeed computation Eq. (26)
$\vec{V}^{BL}_{A_{jk}}$
Blade section angle of attack computation Eq. (27)
α^{BL}_{jk}
Blade section contribution to regression matrix A computation
Blade contribution to regression matrix A computation
Propeller clockwise rotation
ψ_{jk}
Propeller contribution to regression matrix A computation Eqs. (42)-(45)
Propeller contribution to A over a complete propeller rotation average computation Eq. (2.49)
Propeller: i
i = 1
Blade section: k
k = 1, 2, ..., n_{bs}
Blade: j
j = 1, 2, ..., n_b
Measurement vector b composition Eq. (48)
$\vec{b}$
Regression matrix A composition Eq. (2.46)
$\vec{A}$
Constrained problem optimisation Eq. (59)
$C_l(\alpha), C_d(\alpha)$
Repeat q times

Fig. 14 Flowchart of the offline lift and drag coefficient curves identification.

A. Bebop 2 lift and drag coefficients identification

Table 1 summarises the geometry of the Bebop 2 drone and propeller. Besides that, in section V it was mentioned that the aerodynamic gray-box and BET models require as input for each data point a set of conditions ($\vec{V}^B$, $\vec{\Omega}$, ω) beyond the drone geometry. The range of those input conditions is also explained next:

Table 1 Geometrical properties of the Bebop 2 drone and propeller.

Drone geometry		Propeller geometry							
b	l	c_r	c_c	c_t	h_1 & h_2	R	n_b	θ_0	θ_{tw}
(mm)	(mm)	(mm)	(mm)	(mm)	(mm)	(mm)	(-)	(°)	(°/mm)
115	87.5	13	20	8	32	75	3	27	0.29

1) The drone linear velocity is constrained in the x-z plane (V^B_y=0). Its value in the z-direction (V^B_z) is sampled from a uniform distribution over the closed interval [-2, -0.5] m/s, avoiding positive velocities in the z-axis that could cause nonlinear behaviour, e.g. Vortex Ring State, that was not accounted for by the aerodynamic gray-box model.
   Furthermore, the value of the drone linear velocity in the x-direction (V^B_x) is also sampled from a uniform distribution, in this case over the closed interval [-3, 3] m/s. As a result, for the identification of the lift and drag curves only scenarios in which the drone is ascending, flying forward or backwards are considered.
2) The angular velocity of the drone is always zero ($\vec{\Omega} = \vec{0}$).
3) The rotation velocity of the propeller (ω) is sampled from a uniform distribution over the closed interval [300, 1256] rad/s.

Furthermore, the drag and lift curves were approximated with second degree polynomials (m and n equal 2) and the integral with respect to the azimuth angle presented in Eq. (49) was approximated with 10 discrete equally spaced azimuth angles starting at 0°, namely [0°, 36°, 72°, ..., 288°, 324°]. The constrained optimisation method is posed using Python's Scipy package with the trust-region interior point method ("trust-constr") [29] solver. Additionally, the number of blade sections (n_{bs}) and the number of data points used for the identification (q) were 100 and 16,000, respectively.

Using the aforementioned chosen hyper-parameters, the data gathering and optimisation took 10,707 seconds (2 hours and 58 minutes) in a consumer laptop with an Intel Core i7-9750H CPU running Python 3.7. Equations (60) and (61) show the identified polynomials that define the airfoil lift and drag curves with respect to the angle of attack. Plots of these polynomials in Fig. 15 and Fig. 16 show the characteristic shape expected from those aerodynamic curves. Only the 5th constraint in section V is limiting in the solution, namely that the drag coefficient cannot be negative.

$$C_l = 0.24 + 5.15\alpha - 12.25\alpha^2 \quad (60) \qquad C_d = 0.0092 - 0.79\alpha + 15.13\alpha^2 \quad (61)$$

The identified aerodynamic model is validated in appendix B by verifying that the model residuals approximate zero mean white noise.

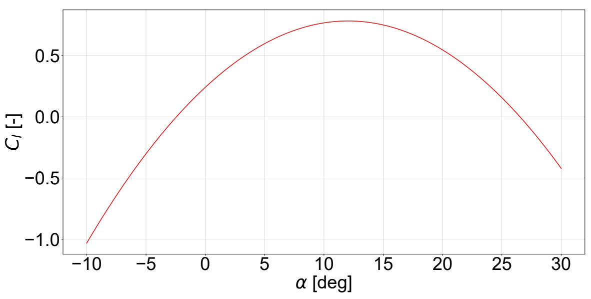


Fig. 15 Identified airfoil lift coefficient curve with respect to the angle of attack from the aerodynamic gray-box model [12] data. Illustration of Eq. (60)**.**

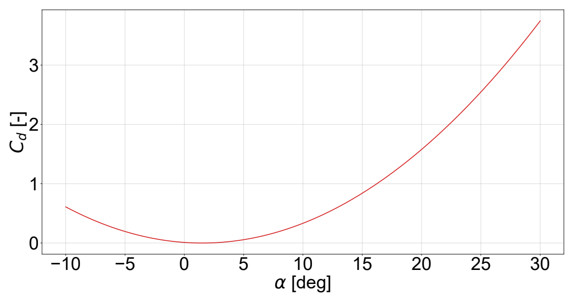


Fig. 16 Identified airfoil drag coefficient curve with respect to the angle of attack from the aerodynamic gray-box model [12] data. Illustration of Eq. (61)**.**

B. Bebop 2 mass and aerodynamic forces and moments

To observe the magnitude of the forces and moments caused by the change in mass, the front left Bebop 2 propeller is simulated as damaged with a loss of 20% of its length. It is rotating at 600 rad/s counterclockwise from a top-down view (ζ=-1) for 0.25 seconds. The attitude of the drone is such that the z-axis direction of the propeller and inertial coordinate frame coincide. Additionally, the drone is moving with a body linear velocity $\overrightarrow{V}^B = \begin{bmatrix} 3 & 0 & -1 \end{bmatrix}^T$ [m/s] and body angular velocity $\overrightarrow{\Omega} = \overrightarrow{0}$ rad/s. The Bebop 2 propeller has a total mass of 5.07 [g] and a mass per blade (without the central hub) of 1.11 [g].

Figures 17 and 18 show the forces and moments caused by the change of mass for the aforementioned scenario, respectively. As can be observed, the forces and moments in the x- and y- directions have a oscillatory behaviour due to the propeller rotations. Additionally, since the drone's propeller z-axis is aligned with its counterpart in the inertial frame, the gravity force coincides with the z-axis in the propeller coordinate frame, leading to a constant force in the z-direction and a zero moment about the z-axis.

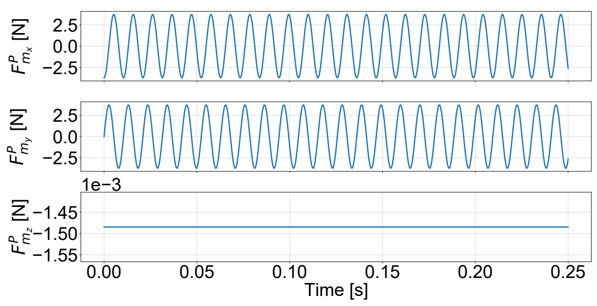


Fig. 17 BET-simulated evolution of forces caused due to mass change upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

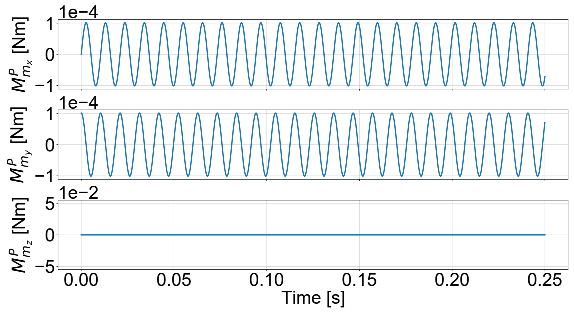


Fig. 18 BET-simulated evolution of moments caused due to mass change upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Next, the aerodynamic forces and moments are also analysed for the same 0.25 s Bebop 2 propeller scenario, leading to the results illustrated in Fig. 19 and Fig. 20. As can be observed, the forces and moments around the y-axis are centred around the 0 datum, whereas the wrenches in the x- and z-direction are biased. The wrench in the x-direction is negatively biased because the lift and drag forces are the highest when the damaged blade is advancing and not retreating. Since the propeller is rotating counterclockwise, the thrust produced when the blade is advancing creates a negative moment around the x-axis and the force creating the torque points towards the negative x-direction. The oscillatory motion in the z-direction is also due to the incoming air velocity from the vehicle linear and angular velocities, which causes the damaged blade incoming flow to be different when it is advancing than when it is retreating. In the case that the drone were hovering, then the aerodynamic wrench in this direction would be constant and will have a value approximately equal to the observed bias.

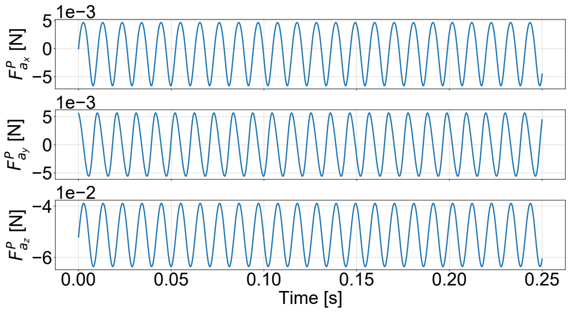


Fig. 19 BET-simulated evolution of aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

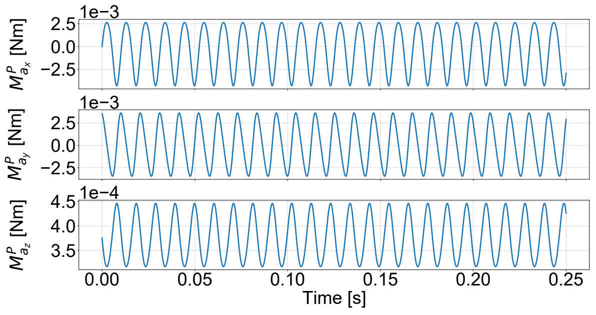


Fig. 20 BET-simulated evolution of aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Furthermore, Fig. 21 and Fig. 22 show the mass and aerodynamic caused forces and moments super-imposed. The mass change effects are predominant in the x- and y-components of the force, whereas the aerodynamic effects are predominant in the force z-direction and in all moment directions.

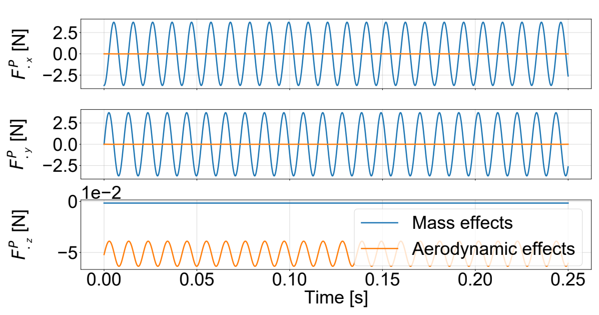


Fig. 21 BET-simulated evolution of mass and aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

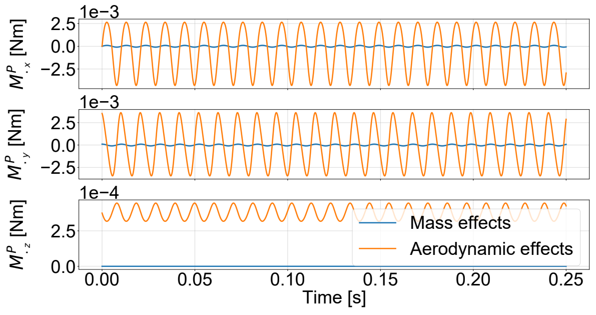


Fig. 22 BET-simulated evolution of mass and aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Finally, the mass and aerodynamic effects around the propeller's centre of rotation are combined as shown by Eq. (62) and Eq. (63); quantities that will be used in the validation phase. Combining both effects for the discussed 0.25 s simulation leads to Fig. 23 and Fig. 24. Previous literature is correct in focusing on the mass related centrifugal forces, since they are one to three orders of magnitude greater than the rest. However, they ignore the effects outside the x-y plane in the propeller coordinate frame, especially the force in the z-direction. Although subtle, the oscillations in the moment signals could help in the identification of the blade damage.

$$\Delta\overrightarrow{F}^P = \overrightarrow{F}^P_{m1} + \overrightarrow{F}^P_{m2} - \overrightarrow{F}^P_{a1} - \overrightarrow{F}^P_{a2} \tag{62}$$

$$\Delta\overrightarrow{M}^P = \overrightarrow{M}^P_{m} - \overrightarrow{M}^P_{a1} - \overrightarrow{M}^P_{a2} \tag{63}$$

Even though there is a different order of magnitude among the forces and the moments, all the signals are oscillatory. Figures 25 and 26 show the upper and lower limit of these wrench oscillations for different degrees of blade damage, namely from 0% (intact blade) to 100% (complete blade loss). All forces and moments have their upper and lower limits symmetric with respect to the 0 datum, except the force and moment in the z-direction. ΔF^P_z moves in the positive direction with increasing blade damage — the higher the blade damage, the larger the thrust loss (positive thrust points down in the propeller coordinate system). ΔM^P_z moves in the negative direction with increasing blade damage because the analysis is done on a counter-clockwise rotating propeller.

The higher the blade damage, the higher the oscillations. However, depending on whether the dominating effect is mass or aerodynamic, the behaviour of those limits is different. The gradient of the upper and lower limits of ΔF^P_x can be observed in Fig. 27; the gradient first increases and around 50% starts decaying. This is caused by the modelled double trapezoid Bebop 2 blade shape. As the blade is progressively damaged from the tip (0% damage) to the location

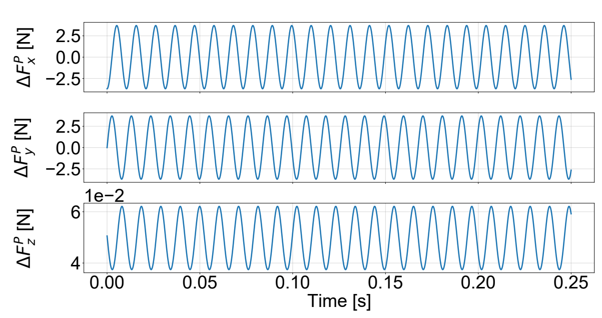


Fig. 23 BET-simulated evolution of forces upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

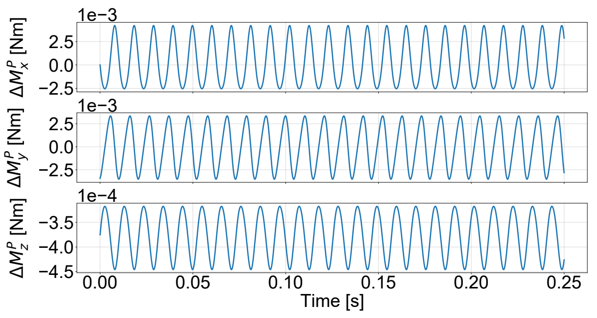


Fig. 24 BET-simulated evolution of moments upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

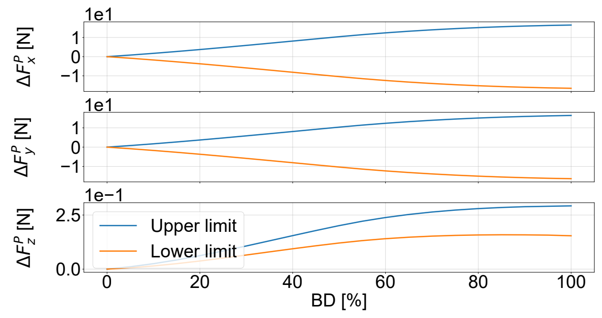


Fig. 25 Upper and lower limits of the forces' oscillations for different degrees of BET-simulated blade damage.

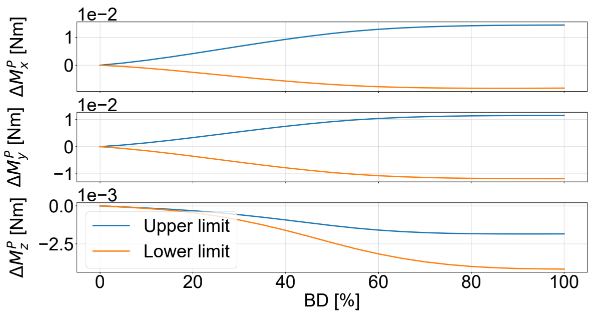


Fig. 26 Upper and lower limits of the moments' oscillations for different degrees of BET-simulated blade damage.

of the central chord (50% damage) the removed blade sections are progressively growing in size, causing constantly greater shifts of the centre of gravity and, hence, greater increments in the centrifugal force. When the damage reaches the maximum chord, the removed blade sections start to decrease in size, leading to smaller centre of gravity shifts and more slowly growing centrifugal forces.

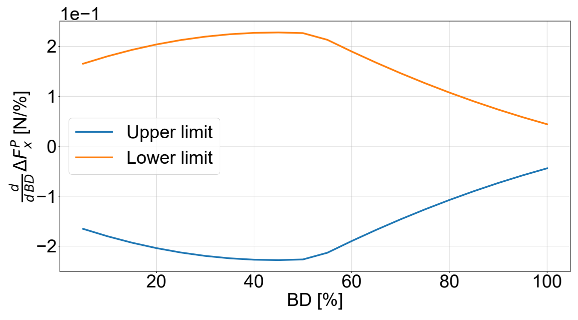


Fig. 27 Gradient of the upper and lower limits of the ΔF_x^P oscillations with respect to different degrees of BET-simulated blade damage.

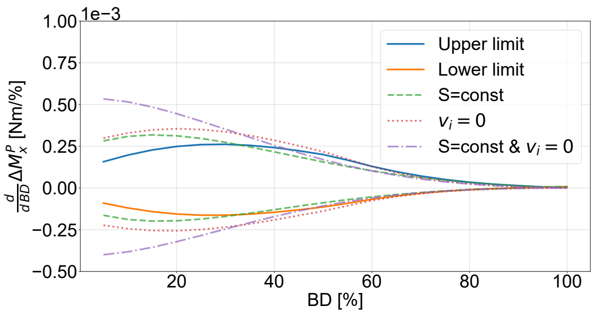


Fig. 28 Gradient of the upper and lower limits of the ΔM_x^P oscillations with respect to different degrees of BET-simulated blade damage. The non solid lines represent scenarios in which the blade section area is constant and/or there is no induced velocity.

Figure 28 shows the gradients of the upper and lower oscillation limits of ΔM_x^P, an aerodynamic dominated wrench component. As can be observed, the moment gradient increases in magnitude until approximately 30%. This initial increment is caused by the combined effect of the increasing blade section area when traversing the blade from the tip towards the central maximum chord and the induced velocity, which ultimately affects the angle of attack. This can be observed when the blade section area is made constant and the induced velocity is removed, then the oscillations are only decaying in amplitude. However, when at least one of these two factors is still active, there is an initial increment in the gradient.

When the 30% blade damaged is reached, the reducing distance from the blade section to the centre of rotation (r_{jk}) becomes the most influential factor in the further decay of the gradient. ΔM_x^P is proportional to r_{jk}^3 when considering that at high propeller rotational rates the airspeed is dominated by $V_{A_{jk_2}}$ (Eq. (16)), which is a function of r_{ijk}, and the airspeed is squared in the computation of ΔL_k and ΔD_k (Eq. (11) and Eq. (12)). The lift and drag components are then used for the computation of the ΔT_{jk}, which is multiplied again by r_{jk} to obtain the moment.

C. Importance of induced velocity

The importance of including the induced velocity in the aerodynamic calculations outlined in subsection IV.A becomes clear when comparing Fig. 29 and Fig. 30. When the linear induced velocity model is included, the amplitude of the oscillations and the datum around which they oscillate are decreased in magnitude. This is attributed to the reduction in the angle of attack created by the introduction of v_i, a consequence which can be observed in Fig. 31 and Fig. 32. These figures show box plots of the angles of attack seen by each of the 100 blade sections in the 16,000 scenarios used for identification of the lift and drag curves (subsection VI.A) with and without the linear induced velocity model, respectively.

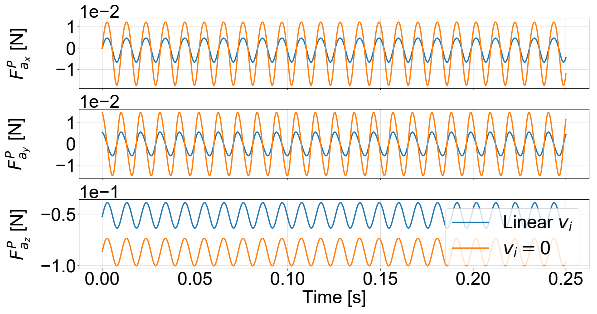


Fig. 29 BET-simulated evolution of aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 [s] rotating at ω_0 = 600 [rad/s] with and without linear inflow model.

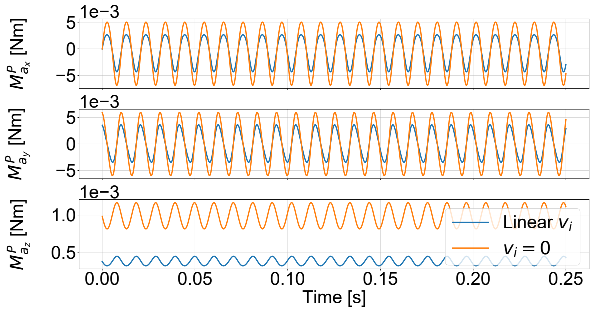


Fig. 30 BET-simulated evolution of aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 [s] rotating at ω_0 = 600 [rad/s] with and without linear inflow model.

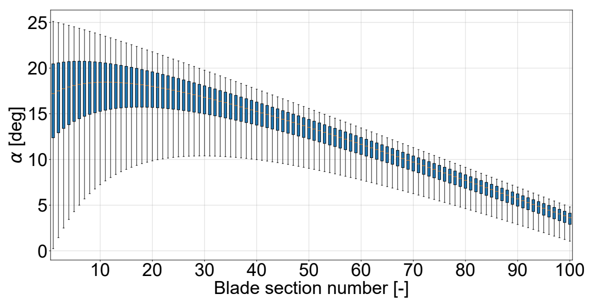


Fig. 31 Box plot with the angles of attack seen by each BET-simulated blade section during 16,000 data point optimisation without induced velocity model. The inputs that shape each data point are taken from uniform distributions with the following value ranges: $\vec{V}_x^B$ =[-3, 3] m/s, $\vec{V}_y^B$ =0 m/s, $\vec{V}_z^B$ =[-2.5, -0.5] m/s, $\vec{\Omega} = \vec{0}$ rad/s, ω =[300, 1256] rad/s.

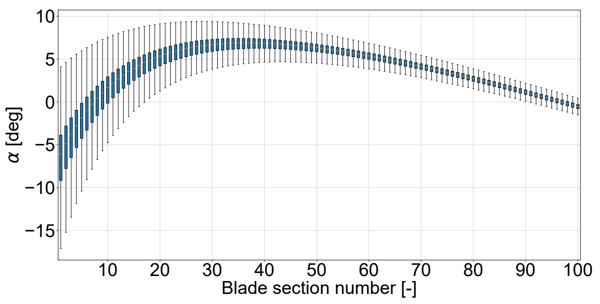
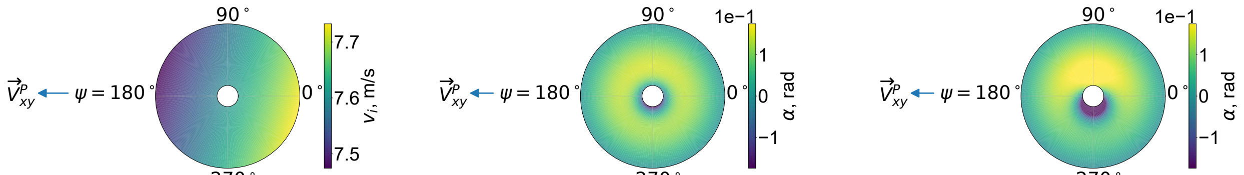


Fig. 32 Box plot with the angles of attack seen by each BET-simulated blade section during 16,000 data point optimisation with linear inflow velocity model. The inputs that shape each data point are taken from uniform distributions with the following value ranges: $\vec{V}_x^B$ =[-3, 3] m/s, $\vec{V}_y^B$ =0 m/s, $\vec{V}_z^B$ =[-2.5, -0.5] m/s, $\vec{\Omega} = \vec{0}$ rad/s, ω =[300, 1256] rad/s.

From the angle of attack box plots (Fig. 31 and Fig. 32), the following additional observations can be made:

1) Apart from the general reduction in the angle of attack, the line that could be drawn from the blade sections' angle of attack medians only becomes linear at a higher angle of attack when the induced velocity model is introduced. In contrast, when $v_i = 0$ the median line is linear for the largest part of the plot. Even though the v_i model is linear, it is important to remind the reader that this property applies in the x and y directions in the propeller reference frame, as can be observed in Fig. 12, not along the blade radial direction.
2) Even though the twist of the blade goes from 25° to 5° from the root to the tip, the line of medians of Fig. 31 has a lower value than the twist — especially close to the root — because the distribution of the linear body velocity in the z-direction is biased towards negative values (the drone is flying upwards). In subsection VI.A it was established that V_z^B has a value in the closed interval [-2,-0.5] m/s.
3) For both scenarios, the range of angles of attack is larger at the root due to the higher sensitivity to the vehicle's velocity; the blade section tangential velocity due to the propeller's rotation is lower at the root than at the tip.

The value of the induced velocity and its effect on the angle of attack can also be visualised in the propeller plane, as illustrated in Fig. 33a and Fig. 33b for the Bebop 2 propeller located on the front left of the vehicle. The propeller is moving towards the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise. The empty internal concentric circle represents the propeller hub, which is not an aerodynamic surface, resulting in an annulus heat map. Figure 33b reflects the same behaviour as in Fig. 32 but in 2D, namely that the angle of attack rapidly increases close to the root until about 35% of the blade before it starts decaying more slowly towards the tip. The low angle of attack values close to the root in the direction of ψ =270° correspond to the retreating blade sections whose airspeed caused by the propeller rotation ($\overrightarrow{V}^{BL}_{A_{jk_2}}$) acts in opposite direction to the airspeed caused by the linear and angular displacement of the vehicle ($\overrightarrow{V}^{BL}_{A_{jk_1}}$). When the rotational speed is lowered to 300 rad/s, the stalled retreating blade sections become more apparent, as shown in Fig. 33c.

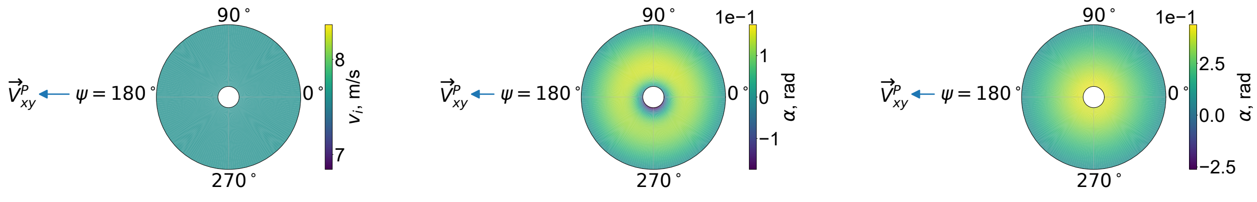


(a) ω **=1256 rad/s** **(b)** ω **=1256 rad/s** **(c)** ω **=300 rad/s**

Fig. 33 Heat map of the linear induced model velocity and angle of attack for the BET-simulated front left Bebop 2 propeller moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

Finally, Fig. 34a and Fig. 34b show the same induced velocity and angle of attack plots when a uniform induced flow is considered. As can be observed, the variations in both variables brought by the linear model corrections are very small, practically unnoticeable in the angle of attack when comparing the results with Fig. 33b. While the uniform inflow model creates an induced velocity of 7.6 m/s across the complete blade, the linear inflow model creates an induced velocity that varies from 7.48 to 7.72 m/s. Even though literature has proven empirically that the linear inflow model is more accurate than the uniform counterpart [23], the difference can be considered negligible for propellers of small radius, as it is the case for most commercial drones such as the Bebop 2; a dependency on the blade radius which can be deduced from Eq. (21). The effect of the linear model can be observed in the main rotor system of helicopters which have blades longer than 1.5 metres.

Hence, the additional computations required for the uniform model corrections could be ignored in simulation. However, it is essential to include the uniform induced model, as it has been shown that it can modify the vertical airspeed seen by the blade sections from 1 to 8.6 m/s at full propeller rotational speed (1256 rad/s) for the Bebop 2 example scenario (Fig. 34a). This effect can be visualised when comparing the angle of attack heat map when there is a uniform induced velocity, as it is the case in Fig. 34b, to the scenario when $v_i = 0$ shown in Fig. 34c. The latter figure is a 2D representation of Fig. 31. The importance of the uniform inflow model is especially noticeable at the retreating blade sections close to the root.



(a) Uniform induced model **(b) Uniform induced model** **(c) No induced model**

Fig. 34 Heat map of the induced model velocity and angle of attack for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s and moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

D. Assumptions and recommendations

For the development of the blade damage forces and moments model, as well as the lift and drag coefficient curves identification, the following assumptions were made:

1) The mass along the blade is homogeneous, meaning that the centroid equals the location of the centre of gravity.
2) The Bebop 2 blades are simplified as two trapezoids with parallel sides connected by the long parallel side.
3) The twist decreases linearly from the root to the tip.
4) The airfoil is constant throughout the blade.
5) The cross flow along the span of the blade is ignored.
6) Aeroelasticity effects are ignored.
7) The blade root and tip lift losses are ignored.
8) The induced velocity is computed with the simplified linear induced inflow. It is assumed to be a good approximation of the real induced velocity as demonstrated empirically in previous literature.
9) The nonlinear aerodynamic effects between (damaged) blades are not considered.
10) The nonlinear aerodynamic effects between propellers are not considered.
11) The nonlinear aerodynamic effects between the propellers and the body frame are not considered.
12) The data used for the identification of the lift and drag coefficient curves is obtained from the aerodynamic gray-box model [12] that provides the propeller thrust. Hence, the present work adopts the assumptions taken for the development of this model.
13) The blade is cut parallel to the edge of the propeller, perpendicular to its span, such that the remaining polygon is still a trapezoid. Hence, slanted or irregular cuts are not considered.

Further work in the simulation of propeller damage could be oriented towards the refinement of the model developed in this chapter in order to remove one or multiple of the aforementioned assumptions; contributing to its generalisation and application to different propeller types. For instance, the geometrical assumptions 1)-4) could be eliminated by creating a 3D model (digital twin) of the propeller using scanning technologies that probe the propeller through physical touch (with contact), such as Coordinate Measuring Machines [30], or scanning technologies that exploit acoustic, optical or magnetic approaches (without contact), such as laser scanning, structured light or photogrammetry [31, 32] (e.g. structure from motion). If translated to a CAD model, this would allow the computation of the twist, chord and volume of each blade section, the latter being used for the computation of the centre of gravity when the density of the material is known. Additionally, such model would contribute to the potential discovery of multiple airfoils present in the blade. If that would be the case, the parameter vector of Eq. (47) would be expanded with the polynomial coefficients used to identify the lift and drag coefficient curves of those additional airfoils.

Assumptions 5)-11) are related to the degree of aerodynamic complexity introduced in the model. In particular, assumption 6) points out that aeroelastic effects have been ignored. Most literature in this regard is oriented towards the modelling of helicopter aeroelastic and blade flapping behaviour [33]. Unfortunately, this knowledge is not directly applicable to drones given that helicopters have a horizontal hinge, also known as flapping hinge, which allows the blade to be displaced up and down to compensate for the rotor lift dissymmetry [23]. Instead, commercial drone rotors lack an articulated head, causing their material to bend and the rotor to tilt with the possibility of flapping [34]. As an alternative, the field of wind energy could be explored since Blade Element Momentum Theory approaches have been used as the aerodynamic component of wind turbine aeroelastic models [35]. However, given the circular dependency between the blade deformations (aeroelastic effects), the induced velocity, and the generated moments and forces, the authors consider such implementation to be challenging for real-time simulations.

Assumption 7) mentions that the blade root and tip losses were ignored. At those blade locations, the circulation must be equal to zero and at the tip there is an additional reduction of lift due to the appearance of tip vortices — airflow around the tip due to the pressure difference between the pressure side (high pressure) and the suction side (low pressure). In the field of wind energy [19], these effects are taken into account by multiplying the induced velocity with a correction factor that is a function of the distance to the centre of rotation (r). This factor would acquire a value of 1 in the centre of the blade and a value of 0 at the edges, allowing the induced velocity to fall to zero at the blade edges. Alternatively, previous literature [19, 23, 36] has also proposed the Prandtl tip-loss factor approximation (B=0.95-0.98) to compute the effective blade radius (R_e = BR) and account for the loss of blade lift. As a result, the outer portion of the blade ($R-R_e$) is considered to be incapable of carrying lift. Given that in helicopter aerodynamics the introduction of the tip loss factor can cause rotor thrust reductions between 6-10% [23], the study of its implementation in drone propellers is recommended for the further improvement of the BET thrust and torque predictions.

Regarding the induced velocity model used (assumption 8)), a comprehensive benchmark study of the different induced inflow models applied to drone propellers is missing in the current literature and it could be considered a line of further work. It is recommended that future studies investigate the suitability and accuracy of the inflow models of Mangler and Squire [37, 38] and Ormiston [39, 40]. The former associates the pressure field across the rotor disk to the inflow with the incompressible, linearised Euler equations. This method originally requires to solve for the rotor loading (Δp) using BET, an approach which is computationally expensive when compared to the linear induced inflow model that optimises the induced velocity before the first BET iteration. However, for the purpose of this research, the thrust obtained from the gray-box aerodynamic model can be used for the computation of the required rotor loading ($T = \pi R^2 \Delta p$), allowing the v_i identification before any BET computation.

Other interesting approaches to consider include linear inflow models, such as those from Payne [41] and Pitt & Peters [25], as well as the Pitt-Peters [42] and the Peters-He [43, 44] dynamic inflow models. The last two approaches have been consolidated and broadly used in the field of rotorcraft dynamics because they exploit unsteady actuator disc theory for hover and forward flight. Instead of ignoring wind-speed fluctuations by averaging the wind field (frozen wake model) or assuming that the instantaneous wind velocity corresponds to that of steady-flow conditions (equilibrium wake model), these dynamic inflow models accurately describe the wake behaviour by assuming the existence of a delay before the induced inflow reacts to modifications in the wind field (unsteady-flow) [19]. Additionally, they are both represented in state-space form, which could be implemented and solved in real-time simulations, and there exist augmentations to their original formulations which include wake distortion effects during manoeuvring flight [45]. Furthermore, even though vortex methods are much more accurate, their computational cost is too high for online blade damage simulations [23, 46].

For the introduction of non-linear inter-propeller, inter-blade or body-blade interactions (assumptions 9)-11)), the creation of a data-driven model that provides the highly nonlinear lift and drag contributions of each blade section, that are not encapsulated in the BET model, is recommended. Similar work that could serve as inspiration is carried out within the field of aerodynamics, discipline in which turbulence is modelled for Reynolds-Averaged Navier-Stokes (RANS) computations using artificial intelligence [47, 48] (data assimilation for CFD closure). In this approach, physics is exploited for simulating large scale flow behaviours, whereas machine learning, a mostly black-box approach, is used for modelling the highly nonlinear lower scale turbulence using experimental data. Within aerodynamics, this method is valued for its low computational cost when compared to higher fidelity but more expensive simulations, such as Direct Numerical Simulations.

Finally, the gray-box aerodynamic model [28] is a data-driven identification approach with physical and semi-physical parameters. Its parametric model structure, namely a piecewise polynomial, is variable since components have been added and removed according to a stepwise selection scheme depending on their contribution to the model accuracy. Beyond its structure, the main model assumption derives from the identification of its parameters with wind tunnel data obtained in quasi-steady flow conditions; there is no rate of change of velocity with time at a single point in the test section volume but the vehicle states, such as the angle of attack, constantly change due to its circular flight motion. It does not enter the unsteady-aerodynamic flow regime because effects caused by the changing circulation and wake on the aerodynamic surfaces are not considered. Future research that would aim to use the developed fault detection and identification framework "in the wild" under the presence of wind field changes, drastic manoeuvres, gusts and turbulences would require the revision of this assumption.

VII. Model validation

The model was validated by comparing the thrust and torque signals measured in an experimental set-up to those predicted by the BET model given the same input conditions ($\overrightarrow{V}^B$, $\overrightarrow{\Omega}$, ω). For that purpose, an experimental campaign was carried out in the Open Jet Facility wind tunnel at the Faculty of Aerospace Engineering at TU Delft.

A. Test set-up and data collection

The OJF is a wind tunnel with an octagonal open test section of 2.85 metres in width and height through which the air flows into a room with a width of 13 metres and a height of 8 metres. The maximum wind speed that can be reached is 35 m/s. For the measurement of thrust and torque, the Series 1580 test stand from Tyto robotics was used. This is a dynamometer for drone propulsion systems capable of measuring up to 5 kg of thrust and 2 Nm of torque, as well as voltage, current, power, motor rotational speed and vibration. Figure 35 and Fig. 36 show the test stand from the side and top, highlighting its most important components.

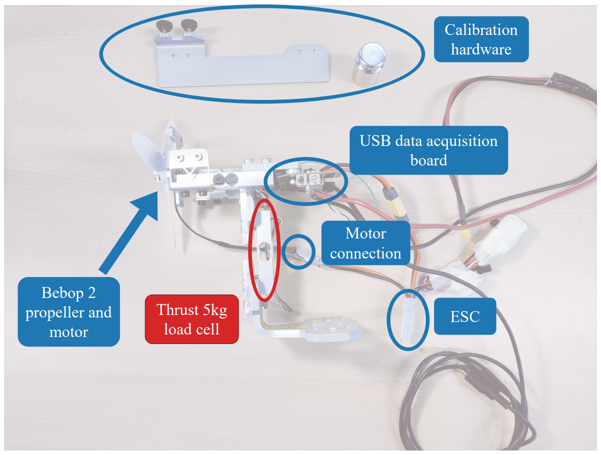


Fig. 35 Tyto stand: side view with calibration hardware.

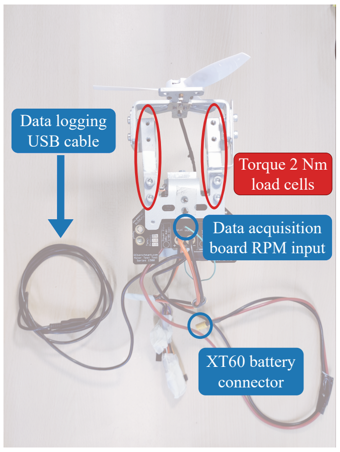


Fig. 36 Tyto stand: top view.

The local influence of the test platform on the freestream flow was minimised by means of a beam assembly, as can be seen in Fig. 37 and Fig. 38. This also enables the positioning of the test stand into the wind tunnel air flow, reducing the wind tunnel wall effects on the flow field.

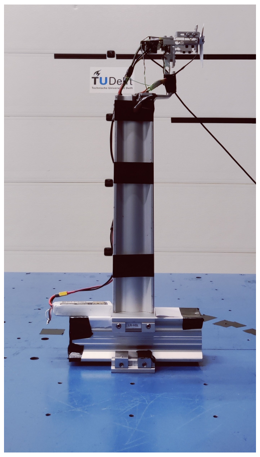


Fig. 37 Test set-up in the wind tunnel.

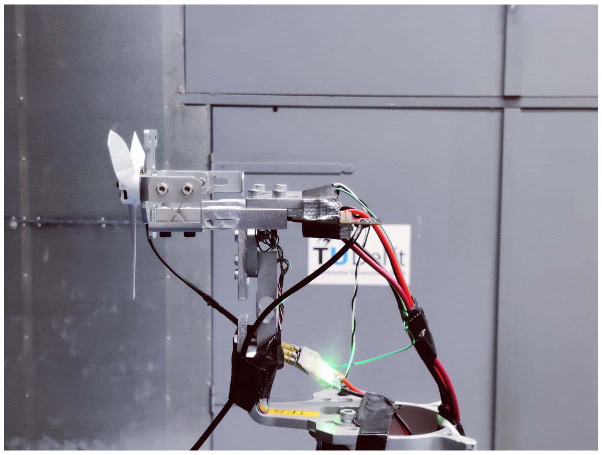


Fig. 38 Tyto test stand in the wind tunnel.

Table 2 shows the parameters that were modified between measurements and their value ranges. For the current experimental campaign, only a single blade was cut at a time with BD percentage of damage. Furthermore, as mentioned in subsection VI.A, the inputs for the BET model in order to create a prediction of thrust and torque are the linear velocity of the drone $\overrightarrow{V}^B$, its angular velocity $\overrightarrow{\Omega}$ and the propeller rotational speed ω. In the wind tunnel, $\overrightarrow{V}^B$ is simulated as the negative wind speed vector and, since the test stand is not rotated during each measurement, $\overrightarrow{\Omega}$ is considered equal to zero. The wind speed vector is decomposed in the wind speed vector magnitude V_∞ and the angle of the normal of the propeller plane with respect to the airflow, also known as the propeller incidence angle i_p (Fig. 39).

Table 2 Experimental campaign testing parameters and values.

Parameter	Unit	Values
BD	%	0, 10, 25
i_p	°	0, 15, 30, 45, 60, 75, 90
V_∞	m/s	0, 2, 4, 6, 9, 12
ω	rad/s	300, 500, 700, 900, 1100

The maximum blade damage tested was 25% due to excessive vibration loading induced on the load cell. The blade damage was created with a cut on the blade orthogonal to its span, as can be seen in Fig. 40 and Fig. 41 for 10% and 25%, respectively.

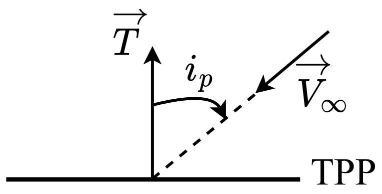


Fig. 39 Propeller incidence angle.

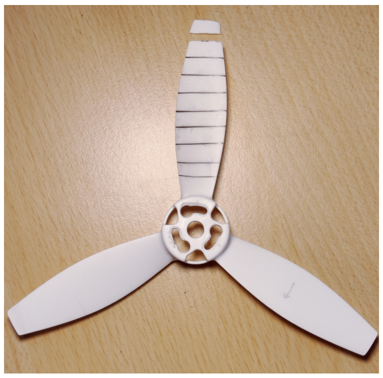


Fig. 40 Damaged propeller with BD=10%.

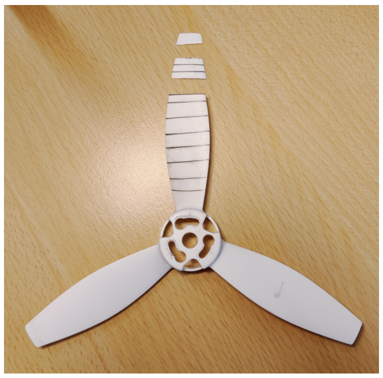


Fig. 41 Damaged propeller with BD=25%.

Test section awareness was provided by the OptiTrack multi-camera three-dimensional optical tracking system, which was also used to measure the incidence angle of the propeller.

Each combination in Table 2 is called an scenario and the order in which the parameters were modified was: $\omega \rightarrow V_\infty \rightarrow i_p \rightarrow BD$. For every scenario, thrust and torque data was gathered at 7Hz for 20 seconds. Given that there are 630 scenarios, the data gathering component of the experimental campaign lasted 3.5 hours.

A few challenges were encountered which affect the results and conclusions derived from the data gathered:

1) Even though the test stands measures accelerometer and propeller rotation values at around 100Hz, it measures the thrust and the torque at 6-7Hz. This is insufficient for signal reconstruction because, in the case of blade damage, the sinusoids observed in section VI have a frequency of 50Hz-175Hz, depending on the propeller rotational speed. The frequency of those oscillations have a much higher value than the Nyquist frequency of 3.5Hz derived from the test stand wrench sampling frequency of 7Hz.
   There exist larger and more complex test stands for the measurement of wrenches for larger propellers. However, their sensitivity or accuracy to small drone propellers might be insufficient, especially if small vibrations in the order of 10^{-4} are expected to be observed.
2) During the execution of the experiments it was encountered that the test stand resonated with the vertical beam and/or the platform at certain rotational frequencies. This resulted in the observation of peaks in the rpm and wrench measurement signals.

3) The 5kg thrust load cell was not able to withstand prolonged vibrations at 25% blade damage or survive a higher degree of blade damage for more than a few seconds.

B. Experimental data pre-processing

The data pre-processing phase of the model validation has 2 steps. First, the data is corrected by adding the effect of the wind on the test stand. For that purpose, the thrust and torque were measured without a propeller installed at different wind speeds and angles with respect to the incoming flow. The mean measured wrenches were then added to each data point depending on their V_∞ and i_p values.

Second, in the case of blade damage it was observed that the forces and moments were increasing with time even though the parameters in Table 2 were kept constant. This is considered an error of the sensor and the signals are detrended.

C. Experimental results

The results from the data gathered during the experimental campaign will be discussed. First, subsubsection VII.C.1 will present the results without blade damage followed by subsubsection VII.C.2 which will examine those in the presence of 10% and 25% blade damage.

1. Without blade damage

This section will delve into analysing the impact of the last three input parameters from Table 2 on the performance of the BET model when there is no blade damage (BD=0%). In contrast when there is blade damage, it is possible to compare BET's performance to that of the gray-box aerodynamic model.

First, the effect of the incidence angle on the results can be observed by fixing the wind and the propeller rotational speeds to constant values, whereas i_p is increased from 0° to 90°. This is done, for V_∞=2 m/s and ω=700 rad/s in Fig. 42 and Fig. 43, where the results are shown for thrust and torque, respectively.

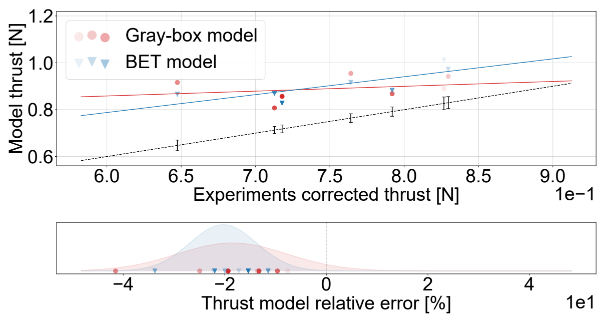


Fig. 42 Experimental and model thrust measurements and their relative error for: BD=0%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

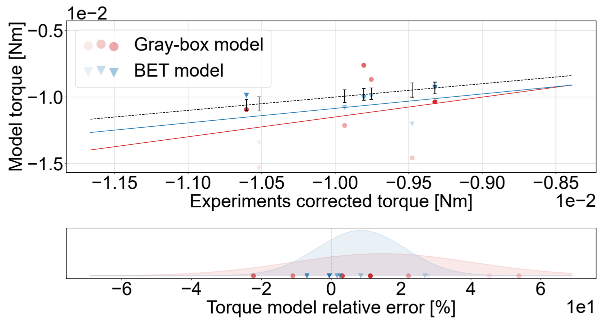


Fig. 43 Experimental and model torque measurements and their relative error for: BD=0%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

The upper window of each plot presents the values obtained in the experimental campaign (x-axis) with respect to the values obtained by each of the models (y-axis) given the same conditions in terms of $\overrightarrow{V}^B$, $\overrightarrow{\Omega}$ and ω. In an ideal scenario, both models' data points would fall on the dashed black line, meaning that both experimental and simulated results are equal. Unfortunately, that is mostly not the case and the solid red and blue lines represent the linear fit of each model's data. Besides that, the data points are plotted with different degrees of transparency. The degree of transparency varies linearly from the brightest or most opaque markers representing those data points measured at i_p=0° to the most transparent ones representing those data points measured at i_p=90°. Finally, the whiskers represent the range of values in which 95% of the experimental wrench data samples can be found ($\approx 2\sigma$). For the computation of these value ranges, the standard deviation of the forces and moments exerted by the wind on the test stand (subsection VII.B) were included. The lower window of each plot shows the wrench relative error of each model with respect to the experimental measurements, as well as a fitted Gaussian curve to those error data points.

As expected, the measured thrust decreases with decreasing i_p because the blade element angle of attack is decreased due to a higher wind speed perpendicular to the plane of rotation. Furthermore, it can be observed that the thrust is always positively biased and the torque is negatively biased for both models. This indicates that there exist some unmodelled physical effects that have not been taken into consideration, among which might be those outlined in subsection VI.D. Additionally, these plots show that the performance of the BET and gray-box models is very similar, which supports the hypothesis that the BET model has been well identified and that the errors are due to those in the gray-box model whose data was used for identification.

After having seen the effect of varying i_p, the next step would be the analysis of the influence of the propeller rotational speed. All the results for different values of ω can be synthesised and compressed in Fig. 44 by plotting the mean of the relative error for each propeller rotational speed with whiskers representing 1.96 times the standard deviation. Both the mean and the standard deviation were obtained from the Gaussian curves in the lower window of plots identical to Fig. 42 for different values of ω. Two conclusions can be derived for the BET and gray-box aerodynamic models comparison. First, the BET model is more accurate in torque but less in thrust. Second, the BET is more (over) confident of its predictions because of its smaller confidence intervals. The latter observation is also visible in Fig. 42 and Fig. 43 due to the taller and narrower Gaussian fitted curves for the BET model. Besides that, it is shown that model accuracy increases with ω for both models.

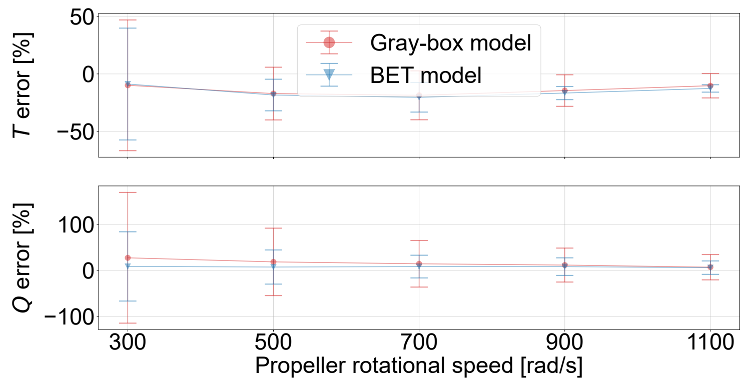


Fig. 44 BET and gray-box aerodynamic model thrust and torque relative error for: BD=0% and V_∞=2 m/s.

Finally, in order to observe the effect of the increasing wind speed, Fig. 44 is repeated for each wind speed listed in Table 2 within the same plot. This is shown in Fig. 45 and Fig. 46 for the gray-box aerodynamic and BET models, respectively. Note that the whiskers representing the confidence intervals have been removed for clarity. From these plots, four observations can be made:

1) The performance of both thrust and torque degrades with increasing wind speed for both models.
2) The relative thrust error of the BET model has a sudden increase when the wind speed is 6 m/s or higher when compared to the gray-box aerodynamic model.
3) The BET model performs better than the gray-box aerodynamic model in terms of torque except at 12 m/s.
4) The performance of the gray-box aerodynamic model at a wind speed of 12 m/s for thrust and higher than 4 m/s for torque is very low (sometimes with relative error values above 1000% for torque).

The reason behind the first three differences in performance between models originates from a design choice in subsection VI.A, namely that the BET model airfoil lift and drag coefficients were identified with wind speeds up to 3.6 m/s. Hence, the BET model has never seen data collected at wind speeds higher than 4 m/s. The last observation is unexpected as the gray-box aerodynamic model was identified with data gathered at wind speeds up to 14 m/s.

One general conclusion that can be derived from these observations is that the BET model architecture has a stronger physical foundation for torque than for thrust. Both were identified with data collected at wind speeds lower than 3.6 m/s and the torque is able to perform better at those speeds that the model had not seen before during identification, namely 4, 6, 9 m/s, when compared to the thrust. In most cases, it even performs better than the gray-box aerodynamic model that was used for the identification data generation. This highlights that the unmodelled aerodynamic effects have a stronger impact on the thrust than on the torque.

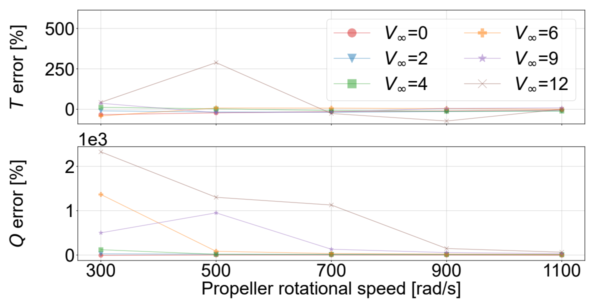


Fig. 45 Gray-box aerodynamic model thrust and torque relative error for BD=0%.

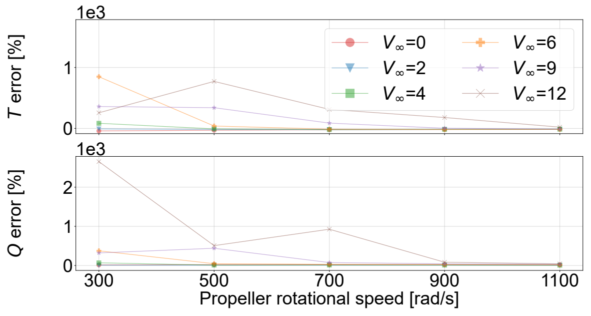


Fig. 46 BET model thrust and torque relative error for BD=0%.

2. With blade damage

In the presence of blade damage, two signal features need to be validated, namely the bias of the signal and the amplitude of the damage induced oscillations. As in subsubsection VII.C.1, the same sensitivity analysis will be performed in which the effect of the parameters in Table 2 will be analysed. In this case, the blade will have 10% and 25% damage. Furthermore, the BET model with a healthy propeller was observed to have errors which could be attributed to the gray-box aerodynamic model used for identification. Hence, the BET model's performance with blade damage will also be compared to that presented in subsubsection VII.C.1 without blade damage. The latter serves as baseline since it can be considered to be the expected default error introduced by the identification dataset, and not from the BET model's architecture (the object of validation of the present section). Finally, as it is the purpose of this research to fill a gap in the literature by developing a thrust and torque propeller model capable of producing reliable estimations in the presence of blade damage, there is no signal from the gray-box aerodynamic model that can be used for control.

To observe the impact on the results when varying i_p, Fig. 47 to 50 show the thrust and torque plots for both blade damages when V_∞=2 m/s and ω=700 rad/s. In contrast to the scenarios without blade damage, when BD=25% the thrust and the torque are not always positively and negatively biased, respectively. This outlying behaviour can only be observed in the thrust plots when ω=500 and 700 rad/s, and it is attributed to the resonance of the test stand with the vertical beam and the platform, as mentioned in subsection VII.A.

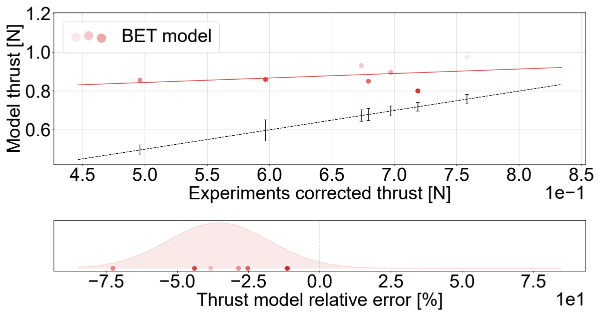


Fig. 47 Experimental and model thrust measurements and their relative error for: BD=10%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

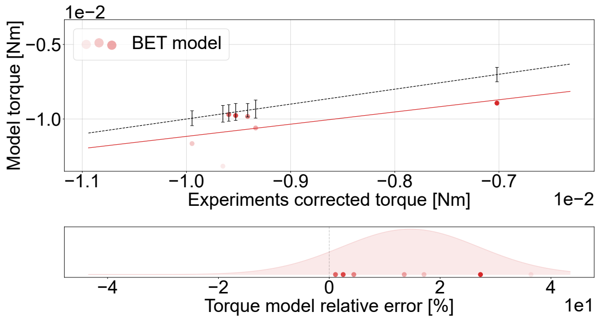


Fig. 48 Experimental and model torque measurements and their relative error for: BD=10%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

For the evaluation of the impact of ω on the results, Fig. 51 and Fig. 52 show the relative error for each value of ω for both degrees of blade damage. As can be seen, with BD=10%, the relative error behaves in a similar manner as when there is no damage: its mean and standard deviation decrease with ω. However, when BD=25%, that pattern is not observed and the mean even crosses the x-axis for the thrust between 500 and 700 rad/s. Again, this anomalous behaviour is attributed to the strong vibrations observed during the experimental campaign at those rotational speeds and blade damage values.

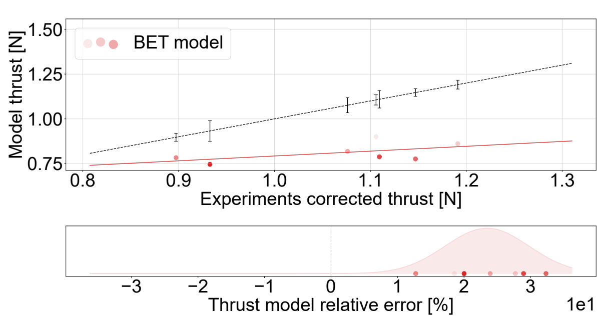


Fig. 49 Experimental and model thrust measurements and their relative error for: BD=25%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

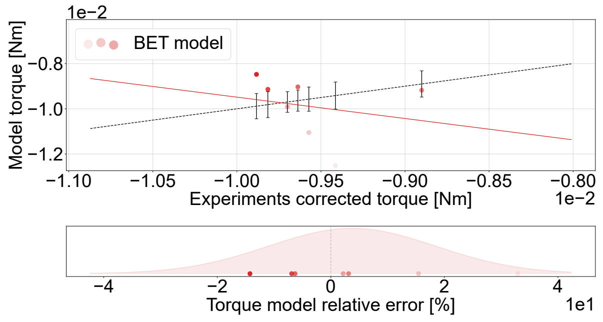


Fig. 50 Experimental and model torque measurements and their relative error for: BD=25%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

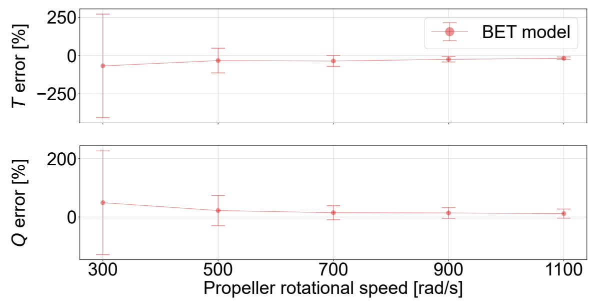


Fig. 51 BET model thrust and torque relative error for: BD=10% and V_∞=2 m/s.

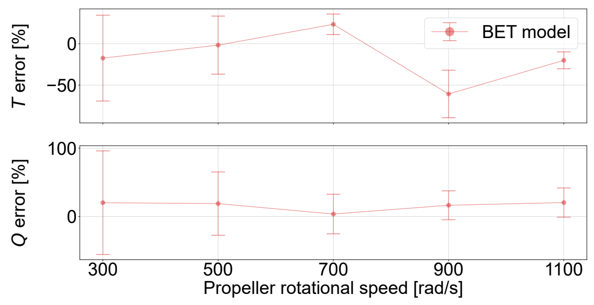


Fig. 52 BET model thrust and torque relative error for: BD=25% and V_∞=2 m/s.

In order to better assess the error originated from the BET model architecture in scenarios with blade damage, Fig. 53 shows the same curves for BD=10% and 25% after subtracting the error when BD=0% — an error which could be attributed to the identification data (Δerror). It shows worse performance for BD=10% at low ω values up to 700 rad/s than BD=25%. Additionally, two more observations can be made. First, the performance at BD=10% improves with ω whereas the performance at BD=25% does not show a clear pattern (of improvement). Second, the model architecture performance could have an error as high as 76.9% (thrust, BD=10%) and as low as 4.9% (torque, BD=25%) for low and high ω values, respectively. Table 3 shows the Δerror range for both blade damages for thrust and torque.

Table 3 Δerror ranges for thrust and torque for 10% and 25% blade damage at 2 m/s wind speeds.

	***BD*=10%**	***BD*=25%**
T Δerror [%]	[-76.9, -23.6]	[-68.5, 15.8]
Q Δerror [%]	[5.1, 39.8]	[-4.9, 14.5]

After having varied i_p and ω, the final parameter to be modified is the V_∞, as shown in Fig. 54 and Fig. 55 for 10% and 25% blade damage, respectively. Again, a sudden decline in performance is observed in both scenarios for wind speeds higher than 4 m/s, phenomenon attributed to the BET model identification data limited to 3.6 m/s.

Finally, the amplitude of the damage induced oscillations needs to be validated. Unfortunately, the sampling frequency is very low for reliable signal reconstruction. In an attempt to reconstruct the signals, the authors tried two approaches, namely an evolutionary algorithmic approach with Particle Swarm Optimisation and an statistical approach with the Lomb-Scargle periodogram [49–51]. The goal is to fit a sinusoid with the same frequency as the ω at which the propeller was oscillating to the experimental data. Both approaches were deemed unfit for the challenging task.

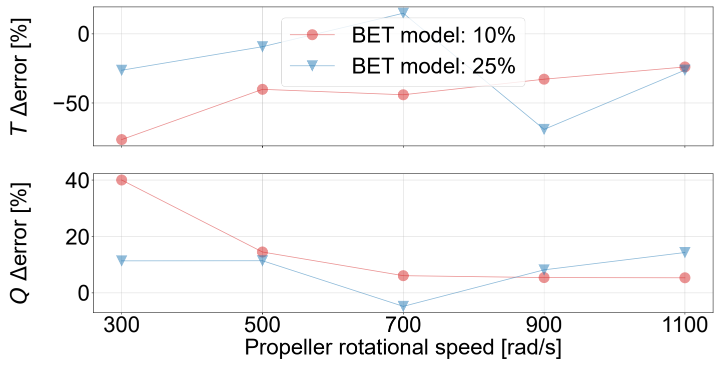


Fig. 53 BET model thrust and torque relative error for BD=[10, 25]% and V_∞=2 m/s, after subtracting the relative error when BD=0%.

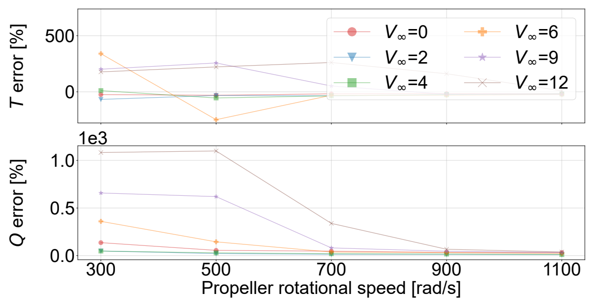


Fig. 54 BET model thrust and torque relative error for BD=10%.

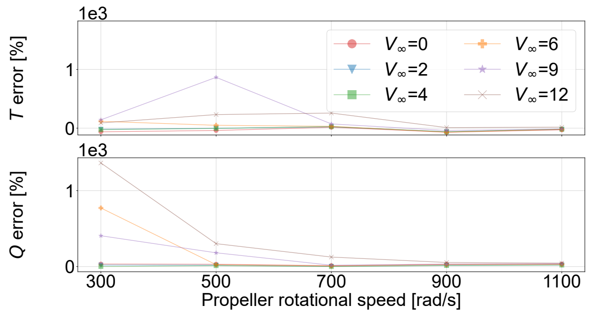


Fig. 55 BET model thrust and torque relative error for BD=25%.

VIII. Conclusion

A white-box model for blade damage simulation, which combines the effects caused by the shift of the centres of gravity and pressure, has been proposed. The mass effects were modelled by discretising the propeller in trapezoids whereas the aerodynamic effects were derived from first principles exploiting Blade Element Theory (BET). Additionally, a BET based methodology for the identification of the UAV propeller 2D aerodynamic properties, namely the airfoil lift and drag curves as a function of the angle of attack, was presented. Such information is currently unknown for most commercial off-the-shelf UAVs.

The presented model has three key advantages. First, it does not require additional costly experimental wind tunnel campaigns for the blade damage modelling. Second, it enables the simulation of any degree of blade damage instead of being limited to a discrete number of failure scenarios within a safe flight regime. Third, it is complementary to existing healthy UAV models and can be used as a plug-in to extend its range of operations to damaged cases. By reducing the component level at which the wrenches are computed, from the complete propeller to single blades and individual blade sections, it will be possible to alternate between models depending on the required level of detail.

The methodology was applied to the Bebop 2 drone, leveraging on the available gray-box aerodynamic model of the chosen platform [12] to build the BET model. From the results it was concluded that previous literature was correct in claiming that the centrifugal forces due to the shift in the centre of gravity were dominant. However, they ignore the loss of weight and the aerodynamic effects, which are not negligible especially at high degrees of blade damage and propeller rotational speeds. The main concern are the neglected forces which, depending on the drone's geometry, could lead to large moments around the UAV's CG. Those oscillations could help in the identification of blade damage and ignoring them could render fault-tolerant control approaches unsuccessful when deployed in the real world.

Furthermore, the induced velocity was shown to be an essential parameter in the model. However, due to the small propeller size of most UAVs, the corrections brought by the linear inflow model over the simpler uniform baseline are negligible and can be ignored in simulation.

To validate the proposed approach, its predicted forces and moments were compared to those obtained from wind tunnel experiments. They were conducted at the Open Jet Facility at Delft University of Technology with a Bebop 2 propeller mounted on a static test stand while four parameters were varied, namely the degree of blade damage (BD), the propeller incidence angle (i_p), the wind speed (V_∞) and the propeller rotational speed (ω).

In the scenarios without blade damage, it was possible to compare the BET model performance with that of the gray-box aerodynamic model. The performance of both is very similar, indicating that the BET model has been well identified and its validation errors are attributed to the identification data gathered from the gray-box aerodynamic model, instead of the BET model architecture. Besides that, the thrust is positively biased and the torque is negatively biased for both models. This remark points to the existence of unmodelled physical effects, among which might be those outlined in subsection VI.D. Additionally, the performance of both models decreases with lower values of ω and higher values of V_∞. This shows that both approaches struggle to correctly model blade sections under a negative angle of attack; phenomenon mostly emergent in those ω-V_∞ conditions for blade sections close to the propeller hub.

Despite their similarities, four differences can be found between both models. First, the BET model is slightly more accurate in torque but less in thrust. Second, it is more (over) confident of its predictions when compared to the gray-box aerodynamic model. Third, the BET model experiences a faster decline in thrust performance for wind speeds higher than 4 m/s. Fourth, the BET model performs better than the gray-box aerodynamic model in terms of torque, except at 12 m/s. The reason behind these divergences in performance originates from a design choice, namely that the BET model was identified with wind speeds up to 3.6 m/s. Hence, the BET model has never seen data collected at wind speeds higher than 4 m/s. Furthermore, an unexpected result is the low performance of the gray-box aerodynamic model at high speeds, as it was identified with wind tunnel data gathered at wind speeds up to 14 m/s.

Moreover, the BET model architecture has a stronger physical foundation for torque than for thrust. Both were identified with data collected at wind speeds lower than 3.6 m/s and the torque is able to perform better at those speeds that the model had not seen before during identification, namely 4, 6, 9 m/s, when compared to the thrust. In most cases, it even performs better than the gray-box aerodynamic model that was used for the identification data generation. This highlights that the unmodelled aerodynamic effects have a stronger impact on the thrust than on the torque.

Regarding those scenarios with blade damage, the bias and the amplitude of the damage induced wrench oscillations needs to be validated. When comparing the experimental and BET model signal bias, that of BD=10% behaves similarly to that of BD=0%. In contrast, when BD=25% the relative error does not decrease with ω as it would be expected. This outlying behaviour, especially noticeable when ω=500 and 700 rad/s, is attributed to the resonance of the test set-up.

For the validation of the oscillations' amplitude, two approaches were implemented for signal reconstruction, namely Particle Swarm Optimization (metaheuristic evolutionary optimization algorithm) and the Lomb-Scargle periodogram (statistical algorithm). Unfortunately, it could not be reliably assessed due to the inaccuracies/noise of the load cell and the difficulty in reconstruction attributed to the low sampling rate.

Besides the recommendations derived from the model assumptions and outlined in subsection VI.D, the authors can only recommend the in-house design of a test stand for the measurement of wrenches of partially damaged propellers. Such a stand would require a sampling frequency above 100 Hz for measurements carried out at the minimum rotational speed of 300 rad/s (or 350 Hz for ω of 1100 rad/s) and a dampening system which prevents resonance with the rest of the structure and the testing platform. Additionally, especially designed load cells have to be used capable of withstanding at least 6g of sustained vibrations.

In addition, the characterization of the complete test set-up in order to identify its dynamics could contribute to the removal of the resonance present in the measurements. Knowledge about the complete system behaviour would allow a deeper understanding of the measured signals and the separation of the set-up dynamics from the "pure" thrust and torque oscillations caused by the damaged propeller.

To conclude, the BET model has been well identified and it has a performance without blade damage similar to that of the gray-box aerodynamic model used for identification. In the presence of blade damage, its performance at high propeller rotational speeds is similar to that without blade damage, with differences in relative error oscillating between 5% and 24%. However, the errors at low propeller rotational speeds can be more than 3 times higher; oscillating between 15% and 75%. Besides that, the validation of the damage induced oscillations amplitude is not possible due to the challenges encountered in the experimental set-up. As a result, it is difficult to fully validate the BET model. The authors hope that the outlined lessons will serve as basis in the design of a future experimental campaign with more specialised hardware. The developed "plug-in" BET model with its future work aspires to become an indispensable cost-effective tool for researchers when designing and testing their work to build more resilient UAVs against blade damage in a wide range of fields, from fault detection and diagnosis to fault-tolerant control.

Appendix

A. Induced velocity computation: gradient-descent approach

As discussed in section IV, the computation of the uniform induced velocity can not be solved analytically and requires a numerical approach. However, this computation needs to happen in every time step of a simulation after the drone has suffered blade damage in one of its propellers; hence, the efficiency of this optimisation is of paramount importance. To this end, the goal of this chapter is to check the possibility of using a computationally efficient gradient-descent approach.

In order to define the optimisation problem objective function, the alternative definition of the airspeed at the rotor (V_R) of Eq. (64) will be used in conjunction with the Glauert formula presented in Eq. (19). In contrast with Eq. (20), the new definition of V_R translates the 3 components of the linear velocity of the propeller assembly ($\overrightarrow{V}^P$) into 2 components, namely its magnitude (V) and the angle of attack of the rotor disk relative to the oncoming flow (α_d). The latter is illustrated in Fig. 56.

$$V_R = \sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2} \tag{64}$$

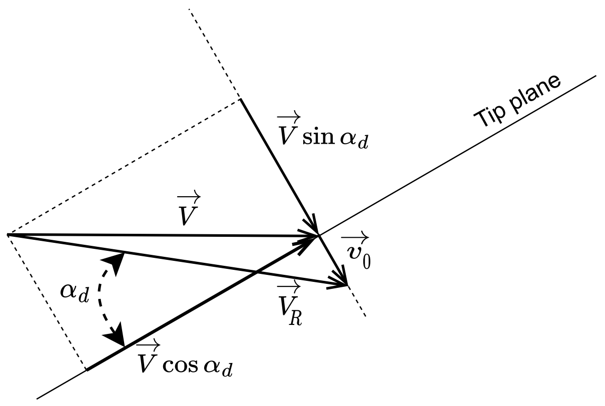


Fig. 56 Angle of attack of the rotor relative to the oncoming flow.

The optimisation problem objective function can be defined as described in Eq. (65), which is the same as finding the location where the function $f(v_0)$ intersects the x-axis.

$$\min_{v_0} \quad |f(v_0)| = \left|T - 2\rho\pi R^2 v_0\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2}\right| \tag{65}$$

Gradient-descent methods are used in optimization for finding the local minimum of a differentiable function by traversing the solution space in the opposite direction of the function gradient, also known as the direction of steepest descend. In the case of the present objective function, local minima will be found where the derivative of $f(v_0)$ with respect to v_0 is zero and where $f(v_0) = 0$. In the case that it can be proven that the function $f(v_0)$ is strictly monotonic, meaning that it only increases or decreases, then $f(v_0)$ will not have local minima and it will be zero at a single value of v_0. Then, there exists a single (global) minimum in the objective function and a gradient-descent approach could be used to find it. Given the definition of the Glauert formula (Eq. (19)), the uniform induced velocity can only have a positive value. Hence, it is only required to prove the strict monotonocity for v_0 values in the half-open interval $[0, \infty)$.

Equation (66) shows the derivative of $f(v_0)$ with respect to v_0 and Eq. (67) shows the uniform induced velocity values that make it zero. As can be seen, $f(v_0)$ has one or two optima when $9\sin^2\alpha_d - 8 \geq 0$. Since the uniform induced velocity can only be positive, the only interesting solution comes from negative α_d angles, ergo when the condition in Eq. (68) is met. When the angle of attack of the rotating disk is higher than $\arcsin -2\sqrt{2}/3$, the function is strictly monotone and gradient-descent would be able to find the global minimum.

$$\frac{df(v_0)}{dv_0} = -2\rho\pi R^2\left(\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2} + v_0\frac{V\sin\alpha_d + v_0}{\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2}}\right) \tag{66}$$

$$v_0 = \frac{V}{4}\left(-3\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right) \tag{67}$$

$$\sin\alpha_d \leq -\frac{2\sqrt{2}}{3} \tag{68}$$

During nominal flight, the drone will experience a positive α_d when in cruise. However, in the case of failure, when the induced velocity has to be computed, the drone could pitch or roll excessively causing the air flow to impact the propeller from below. Hence, it is important to consider the presence of the discovered local minima. Even though the function can contain local minima, they could be avoided by a proper selection of hyper-parameters and initialisation of the optimisation; tuning the gradient-descent to the particular (known) function.

Considering extreme values of v_0, it can be observed in Eq. (65) that the second term of $f(v_0)$ is dominant. As a result, $f(-\infty)$ leads to a positive value and $f(\infty)$ to a negative one, meaning that the function is decreasing in value independently of the value of α_d. In the case that there is a local minimum (Eq. (68) is fulfilled) and it takes place at a lower uniform induced velocity than when $f(v_0) = 0$, as illustrated in Fig. 57, the gradient-descent could be initialised with a high value of v_0 to guarantee that the optimisation will encounter the global optimum before the local minimum. Since the function is decreasing, this approach would not work if $f(v_0) < 0$ at $df(v_0)/dv_0 = 0$, as can be seen in Fig. 58. In order to check whether this latter scenario exists, Eq. (67) is inserted in $f(v_0)$, leading to Eq. (69).

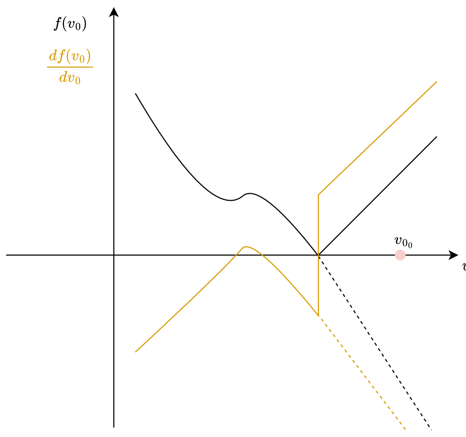


Fig. 57 Desired scenario: local minima takes place at lower induced velocity than global minima, so gradient descend will reach global minima first. The dotted line corresponds to $f(v_0)$, whereas the bold line to $|f(v_0)|$.

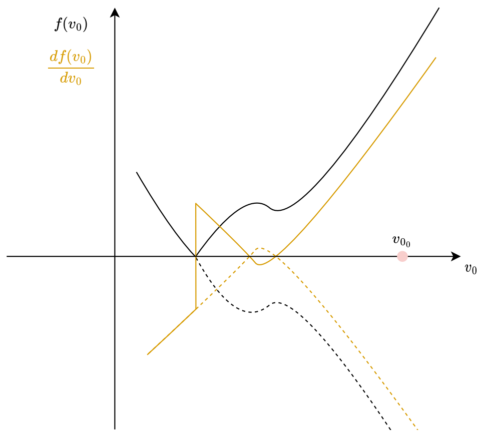


Fig. 58 Undesired scenario: local minima takes place at higher induced velocity than global minima, so gradient descend will reach local minima first. The dotted line corresponds to $f(v_0)$, whereas the bold line to $|f(v_0)|$.

$$T - \frac{\rho\pi R^2 V^2}{2}\left(-3\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right)\sqrt{\cos^2\alpha_d + \frac{1}{16}\left(\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right)^2} \leq 0 \tag{69}$$

Since the local minimum can only be found when $\sin\alpha_d \in [-1, -2\sqrt{2}/3]$ (Eq. (68) is met), the two limits of this range are inserted in Eq. (69), resulting in the two conditions presented in Eq. (70) and Eq. (71). Observing both conditions, the second one is automatically met when the first one is fulfilled. Hence only when Eq. (68) and Eq. (70) are met, there is a local minimum which takes place with a higher uniform induced velocity than when $f(v_0) = 0$. In that case, initialising the optimisation with a high value of v_0 would most likely not converge to the undesired local minimum.

$$\sin\alpha_d = -\frac{2\sqrt{2}}{3}, \quad T < \frac{\sqrt{3}}{3}\rho\pi R^2 V^2 \tag{70}$$

$$\sin\alpha_d = -1, \quad T < \frac{1}{2}\rho\pi R^2 V^2 \tag{71}$$

To check whether the blade damage simulation with the Bebop 2 model would encounter scenarios in which both conditions (Eq. (68) and Eq. (70)) are met, 100,000 scenarios are run with the following set of conditions: $\overrightarrow{V}_x^B \in [-3, 3]$, $\overrightarrow{V}_y^B = 0$, $\overrightarrow{V}_z^B \in [-3, 3]$, $\overrightarrow{\Omega} = \overrightarrow{0}$ and $\omega \in [300, 1256]$. When compared to those nominal conditions presented in subsection VI.A, here the body velocity in the z-direction can also acquire positive values and its absolute value is higher in order to account for the failure cases. Figure 59 shows 100,000 points representing all the scenarios in a plot of vehicle velocity over the thrust (V-T). As can be observed, there is not overlap between the set of points that fulfil the first condition in Eq. (68) (the pink points within the convex hull) and those that meet the second condition in Eq. (70) (the green points under the dashed line). As a result, for the simulations of the present research, only the desired scenario illustrated in Fig. 57 will be observed. Hence, if the gradient-descent is initialised with a high positive value of v_0, it will always encounter the global minimum first.

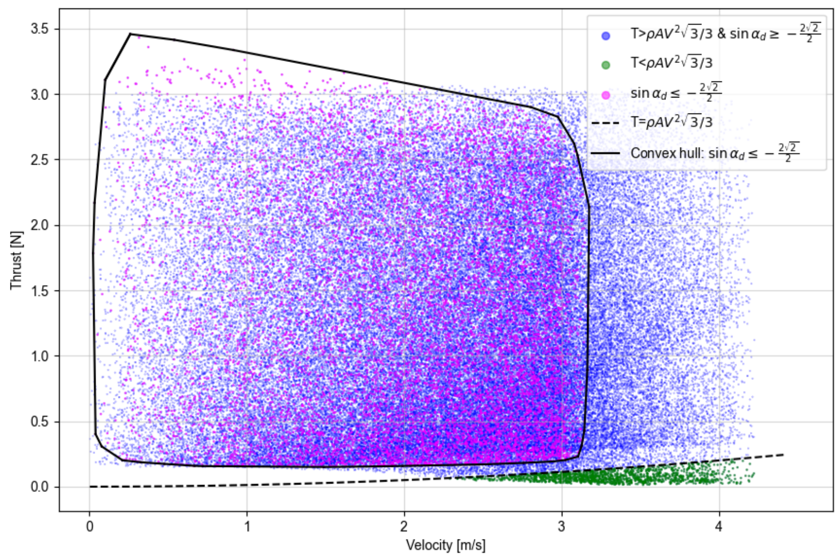


Fig. 59 V-T graph for 100,000 simulation scenarios of the Bebop 2 drone. The convex hull encapsulates all the pink points that meet (first) the condition in Eq. (68)**. The green points under the dashed line meet the (second) condition outlined in Eq.** (70)**. The magenta points are those scenarios in which neither of the conditions are met. From the figure, there is no overlap between both conditions sets.**

Now that it has been proven to be beneficial to initialise the gradient-descent with a high positive value of uniform induced velocity, the question is what the exact initialisation value should be. Observing Fig. 57, it is enough to initialise the gradient-descent with a uniform induced velocity value higher than the maximum v_0 that the local minimum could have. If initialised between the local and global minimum, the gradient-descent will move the solution towards the global optimum to the right along the v_0 axis. If initialised to the right of the global minimum, the gradient-descent will move the solution towards the global optimum to the left along the v_0 axis.

Given Eq. (67), the maximum v_0 that the local minimum could have is found when the output of the square root is positive and α_d has a value of -90°; then, the uniform induced flow equals the incoming flow velocity ($v_0 = V$). From the 100,000 scenarios presented in Fig. 59, the maximum velocity observed is 4.24 m/s. Therefore, if the gradient-descent optimisation is initialised with $v_{0_0} = 4.5$ m/s, the initial function evaluation will always be carried out to the right of the local optima.

Furthermore, the gradient-descent optimisation requires the selection of the learning rate (γ). This hyper-parameter needs to be carefully chosen in order to avoid overshooting the global optimum and landing in the local minimum. Given the update law of the gradient-descent provided by Eq. (72), the algorithm can overshoot the global optimum by a value equal to $\gamma\frac{d|f(v_0)|}{dv_0}$. Hence, the smaller this update step, the lower the probability that the optimisation overshoots the global optimum and lands in the local minimum.

$$v_{0_{i+1}} = v_{0_i} - \gamma \frac{d|f(v_{0_i})|}{dv_0} = v_{0_i} - \gamma \frac{df(v_{0_i})}{dv_0} \frac{f(v_{0_i})}{|f(v_{0_i})|} \tag{72}$$

For the current research, two values are considered for the learning rate, namely 0.5 and 0.1. Additionally, the value of the learning rate is decreased by a factor of 0.5 every time $\frac{d|f(v_0)|}{dv_0}$ changes sign. Using the same 100,000 simulation scenarios of the Bebop 2 drone presented in Fig. 59, the performance results are observed in Table 4. An optimisation is defined as successful when its output is fed to $f(v_0)$ and the result is less than 10^{-5}. As can be seen, the gradient-descent approach has a 100% success rate for both learning rates, in contrast with the slight worse performance of 98.56% for Nelder-Mead. Additionally, the gradient-descent optimisation shows an approximately 20% and 55% computational time reduction with respect to the Nelder-Mead alternative for the 0.1 and 0.5 learning rates, respectively. Hence, the chosen learning rate for the present research is 0.5.

Table 4 Performance results of Nelder-Mead and Gradient-Descent with learning rate values of 0.1 and 0.5.

	Success rate [%]	Time per scenario [ms]
Nelder-Mead	98.56	5.51
Gradient-descent (γ=0.1)	100	4.39
Gradient-descent (γ=0.5)	**100**	**2.45**

Finally, the optimisation concludes when one of the following 3 conditions is met:

1) The maximum number of iterations is reached (i_{max}=10,000).
2) The change in the solution is lower than a threshold ($m < 0.01$) for a certain number of iterations ($c_{max} = 20$).
3) The denominator of the gradient update ($|f(v_0)|$) is less than a very small value ($\epsilon = 10^{-10}$) because then the solution has been found.

B. Airfoil lift and drag coefficient validation

To validate the identified aerodynamic model it is verified whether the model residuals approximate zero mean white noise. This is done for the residuals from those data points used for the identification of the model, as well as for an additional validation dataset made of ($\overrightarrow{V}^B$, $\overrightarrow{\Omega}$, ω) input combinations not seen yet by the model. This validation dataset has 4,000 data points, which is 25% the size of the training dataset. Table 5 shows the mean of the residual curves, as well as their percentage relative to the average value of the thrust or torque. As can be observed, their values are low (below 1%), which indicates that the error of both, identification and validation datasets, could be considered to be zero mean. Besides that, the thrust error is approximately three times lower than the torque error for the NRMSE metrics, which means that the fitted C_l and C_d curves approximate the thrust data better than the torque.

Table 5 Error metrics results for the thrust and torque identification and validation datasets.

	Identification thrust	Identification torque	Validation thrust	Validation torque
Mean value	$-2.89 \cdot 10^{-3}$ [N]	$-1.40 \cdot 10^{-4}$ [Nm]	$-1.56 \cdot 10^{-3}$ [N]	$-1.22 \cdot 10^{-4}$ [Nm]
Mean percentage	-0.23 [%]	-0.97 [%]	-0.13 [%]	-0.86 [%]
NRMSE	$8.43 \cdot 10^{-2}$ [-]	0.26 [-]	$8.30 \cdot 10^{-2}$ [-]	0.25 [-]

Next, it is assessed whether the residual is uncorrelated. For that purpose, Fig. 60 and Fig. 61, as well as Fig. 62 and Fig. 63, show the normalised autocorrelation curves for the residuals with their 95% confidence bounds for the identification and the validation datasets, respectively. Almost all of the autocorrelations fall within the 95% confidence limits and there is no apparent pattern. A few lags fall slightly outside the bounds, but it is not enough to indicate non-randomness. Hence, the residuals can be considered to be white and uncorrelated.

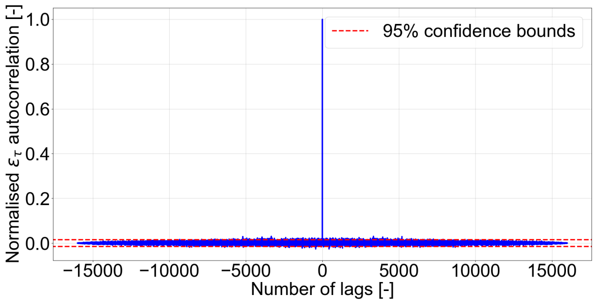


Fig. 60 Identification thrust error normalised autocorrelation with 95% confidence bounds.

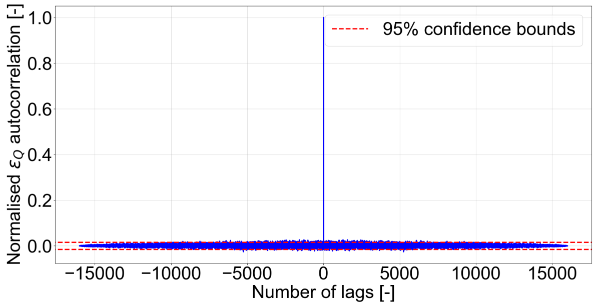


Fig. 61 Identification torque error normalised autocorrelation with 95% confidence bounds.

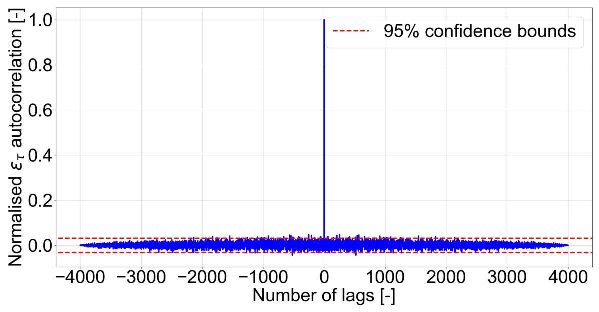


Fig. 62 Validation thrust error normalised autocorrelation with 95% confidence bounds.

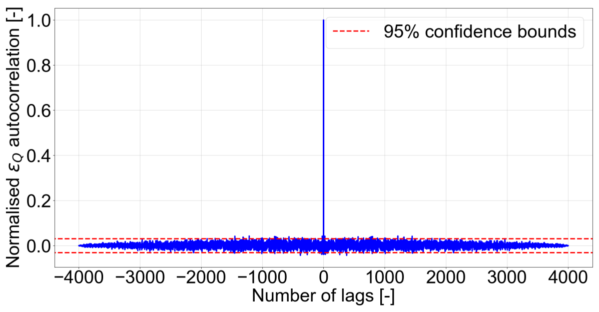


Fig. 63 Validation torque error normalised autocorrelation with 95% confidence bounds.

Acknowledgments

The authors thank Charlie Bogaerts and Burhanuddin Saify for their insights and help in the execution of the experimental wind tunnel campaign.

References

[1] Isermann, R., “Fault Diagnosis Systems An Introduction from Fault Detection to Fault Tolerance,” *Fault-Diagnosis Systems*, 2006. https://doi.org/10.1007/3-540-30368-5.

[2] Tidriri, K., Chatti, N., Verron, S., and Tiplica, T., “Bridging Data-Driven and Model-Based Approaches for Process Fault Diagnosis and Health Monitoring: A Review of Researches and Future Challenges,” *Annual Reviews in Control*, Vol. 42, No. C, 2016, pp. 63–81. https://doi.org/10.1016/j.arcontrol.2016.09.008.

[3] Heredia, G., and Ollero, A., “Detection of Sensor Faults in Small Helicopter UAVs Using Observer/Kalman Filter Identification,” *Mathematical Problems in Engineering*, Vol. 2011, 2011. https://doi.org/10.1155/2011/174618.

[4] Aboutalebi, P., Abbaspour, A., Forouzannezhad, P., and Sargolzaei, A., “A Novel Sensor Fault Detection in an Unmanned Quadrotor Based on Adaptive Neural Observer,” *Journal of Intelligent & Robotic Systems*, 2017. https://doi.org/10.1007/s10846-017-0690-7.

[5] Ducard, G., *Actuator Fault Detection in UAVs*, Springer Netherlands, Dordrecht, 2015, pp. 1071–1122. https://doi.org/10.1007/978-90-481-9707-1_43.

[6] Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M., “Autonomous Obstacle Avoidance and Manoeuvring on a Vision-Guided MAV using On-Board Processing,” *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 2472–2477. https://doi.org/10.1109/icra.2011.5980095.

[7] Sun, S., Baert, M., Schijndel, B., and De Visser, C., “Upset Recovery Control for Quadrotors Subjected to a Complete Rotor Failure from Large Initial Disturbances,” *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 4273–4279. https://doi.org/10.1109/icra40945.2020.9197239.

[8] Nguyen, D.-T., Saussié, D., and Saydy, L., "Fault-Tolerant Control of a Hexacopter UAV based on Self-Scheduled Control Allocation," *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 385–393. https://doi.org/10.1109/icuas.2018.8453440.

[9] Xue, Y., Zhen, Z., Yang, L., and Wen, L., "Adaptive Fault-Tolerant Control for Carrier-Based UAV with Actuator Failures," *Aerospace Science and Technology*, Vol. 107, 2020, p. 106227. https://doi.org/10.1016/j.ast.2020.106227.

[10] Wang, B., Shen, Y., and Zhang, Y., "Active Fault-Tolerant Control for a Quadrotor Helicopter Against Actuator Faults and Model Uncertainties," *Aerospace Science and Technology*, Vol. 99, 2020, p. 105745. https://doi.org/10.1016/j.ast.2020.105745.

[11] Jiang, Y., Zhiyao, Z., Haoxiang, L., and Quan, Q., "Fault Detection and Identification for Quadrotor Based on Airframe Vibration Signals: A Data-Driven Method," *2015 34th Chinese Control Conference (CCC)*, 2015, pp. 6356–6361. https://doi.org/10.1109/chicc.2015.7260639.

[12] Sun, S., and de Visser, C., "Aerodynamic Model Identification of a Quadrotor Subjected to Rotor Failures in the High-Speed Flight Regime," *IEEE Robotics and Automation Letters*, Vol. 4, No. 4, 2019, pp. 3868–3875. https://doi.org/10.1109/lra.2019.2928758.

[13] Avram, R. C., Zhang, X., and Khalili, M., "Quadrotor Actuator Fault Diagnosis with Real-Time Experimental Results," *Annual Conference of the PHM Society*, Vol. 8, 2016. https://doi.org/10.36001/phmconf.2016.v8i1.2504.

[14] Ghalamchi, B., Jia, Z., and Mueller, M. W., "Real-Time Vibration-Based Propeller Fault Diagnosis for Multicopters," *IEEE/ASME Transactions on Mechatronics*, Vol. 25, No. 1, 2020, pp. 395–405. https://doi.org/10.1109/tmech.2019.2947250.

[15] Newman, S., "Principles of Helicopter Aerodynamics," *The Aeronautical Journal*, Vol. 111, 2007, pp. 825–826. https://doi.org/10.1017/S0001924000087352.

[16] Selig, M., *Modeling Full-Envelope Aerodynamics of Small UAVs in Realtime*, Toronto, Ontario, Canada, 2010. https://doi.org/10.2514/6.2010-7635.

[17] Orsag, M., and Bogdan, S., *Influence of Forward and Descent Flight on Quadrotor Dynamics*, IntechOpen, London, United Kingdom, 2012, Chap. 7, pp. 141–156. https://doi.org/10.5772/37438.

[18] Khan, W., and Nahon, M., "Toward an Accurate Physics-Based UAV Thruster Model," *IEEE/ASME Transactions on Mechatronics*, Vol. 18, No. 4, 2013, pp. 1269–1279. https://doi.org/10.1109/tmech.2013.2264105.

[19] Burton, T., Sharpe, D., Henkins, N., and Bossanyi, E., *Wind Energy Handbook*, 2$^{\text{nd}}$ ed., Wiley, Chichester, West Sussex, 2011.

[20] Gill, R., and D'Andrea, R., "Propeller Thrust and Drag in Forward Flight," *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 2017, pp. 73–79. https://doi.org/10.1109/ccta.2017.8062443.

[21] Gill, R., and D'Andrea, R., "Computationally Efficient Force and Moment Models for Propellers in UAV Forward Flight Applications," *Drones*, Vol. 3, No. 4, 2019. https://doi.org/10.3390/drones3040077.

[22] Niemiec, R., and Gandhi, F., "Effects of Inflow Model on Simulated Aeromechanics of a Quadrotor Helicopter," *Proceedings of the 2016 72nd American Helicopter Society (AHS) International Annual Forum*, 2016.

[23] Leishman, J. G., *Principles of Helicopter Aerodynamics*, Cambridge University Press, Cambridge, 2006.

[24] Howlett, J. J., *UH-60A Black Hawk Engineering Simulation Program*, Vol. 1, NTIS, Springfield, Va., 1981.

[25] Pitt, D., and Peters, D., "Theoretical Prediction of Dynamic Inflow Derivatives," *Vertica*, Vol. 5, 1981.

[26] White, F., and Blake, B. B., "Improved Method Of Predicting Helicopter Control Response And Gust Sensitivity," *Proceedings of the 1979 35th Annual Forum of American Helicopter Society*, 1979.

[27] Drees, J., "A Theory of Airflow Through Rotors and its Application to some Helicopter Problems," *Journal of the Helicopter Association of Great Britain, 3*, Vol. 2, 1949, pp. 79–104.

[28] Sun, S., de Visser, C. C., and Chu, Q., "Quadrotor Gray-Box Model Identification from High-Speed Flight Data," *Journal of Aircraft*, Vol. 56, No. 2, 2019, pp. 645–661. https://doi.org/10.2514/1.c035135.

[29] Byrd, R. H., Hribar, M. E., and Nocedal, J., "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *SIAM Journal on Optimization*, Vol. 9, No. 4, 1999, pp. 877–900. https://doi.org/10.1137/s1052623497325107.

[30] Mostafa, A.-B., and Ebrahim, M., "3D Laser Scanners' Techniques Overview," *International Journal of Science and Research (IJSR)*, Vol. 4, 2015, pp. 5–611. https://doi.org/10.1007/s41062-021-00550-9.

[31] Perez-Cortes, J.-C., Perez, A., Saez-Barona, S., Guardiola, J.-L., Salvador Igual, I., and Sáez-Barona, S., "A System for In-Line 3D Inspection without Hidden Surfaces," *Sensors*, Vol. 18, 2018, p. 2993. https://doi.org/10.3390/s18092993.

[32] Bhatti, A. Q., Wahab, A., and Sindi, W., "An Overview of 3D Laser Scanning Techniques and Application on Digitization of Historical Structures," *Innovative Infrastructure Solutions*, Vol. 6, No. 4, 2021, p. 186. https://doi.org/10.1007/s41062-021-00550-9.

[33] Leza, D. V., "Development of a Blade Element Method for CFD Simulations of Helicopter Rotors using the Actuator Disk Approach," Master's thesis, Delft University of Technology, 2018.

[34] Omari, S., Hua, M.-D., Ducard, G., and Hamel, T., "Nonlinear Control of VTOL UAVs Incorporating Flapping Dynamics," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 2419–2425. https://doi.org/10.1109/iros.2013.6696696.

[35] Zhang, P., and Huang, S., "Review of Aeroelasticity for Wind Turbine: Current Status, Research Focus and Future Perspectives," *Frontiers in Energy*, Vol. 5, No. 4, 2011, p. 419–434. https://doi.org/10.1007/s11708-011-0166-6.

[36] Gessow, A., and Myers, G. C., *Aerodynamics of the Helicopter*, F. Ungar Pub. Co., New York, 1967.

[37] Mangler, K. W., and Squire, H. B., *The Induced Velocity Field of a Rotor*, Ministry of Supply, Aeronautical Research Council, London, 1950.

[38] Stepniewski, W. Z., *Rotary-Wing Aerodynamics*, Dover Publications, New York, 1984.

[39] Ormiston, R. A., *An Actuator Disk Theory for Rotor Wake Induced Velocities*, Advisory Group for Aerospace Research and Development (AGARD-CP-111), Aerodynamics of Rotary Wings, 1972. https://doi.org/10.1007/978-3-030-05455-7_2-2.

[40] Ormiston, R. A., "Induced Power of the Helicopter Rotor," *60th Annual Forum of the American Helicopter Society International*, 2004, pp. 33–53.

[41] Payne, P. R., "Helicopter Dynamics and Aerodynamics," Vol. 63, No. 585, 1959. https://doi.org/10.1017/S0368393100071728.

[42] Peters, D. A., and Haquang, N., "Dynamic Inflow for Practical Applications," *Journal of the American Helicopter Society*, Vol. 33, No. 4, 1988, pp. 64–68. https://doi.org/10.4050/jahs.33.64.

[43] Peters, D. A., Boyd, D. D., and He, C., "Finite-State Induced-Flow Model for Rotors in Hover and Forward Flight," *Journal of The American Helicopter Society*, Vol. 34, 1989, pp. 5–17. https://doi.org/10.4050/jahn.34.5.

[44] Peters, D. A., and He, C., "Correlation of Measured Induced Velocities with a Finite-State Wake Model," *Journal of The American Helicopter Society*, Vol. 36, 1991, pp. 59–70. https://doi.org/10.4050/jahs.36.59.

[45] Zhao, J., Prasad, J., and Peters, D., "Rotor Dynamic Wake Distortion Model for Helicopter Maneuvering Flight," *Journal of The American Helicopter Society*, Vol. 49, 2004, pp. 414–424. https://doi.org/10.4050/jahs.49.414.

[46] Leishman, J. G., Bhagwat, M. J., and Bagai, A., "Free-Vortex Filament Methods for the Analysis of Helicopter Rotor Wakes," *Journal of Aircraft*, Vol. 39, No. 5, 2002, pp. 759–775. https://doi.org/10.2514/2.3022.

[47] Singh, A. P., Medida, S., and Duraisamy, K., "Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils," *AIAA Journal*, Vol. 55, No. 7, 2017, pp. 2215–2227. https://doi.org/10.2514/1.j055595.

[48] Volpiani, P. S., Meyer, M., Franceschini, L., Dandois, J., Renac, F., Martin, E., Marquet, O., and Sipp, D., "Machine Learning-Augmented Turbulence Modeling for RANS Simulations of Massively Separated Flows," *Physical Review Fluids*, Vol. 6, 2021, p. 064607. https://doi.org/10.1103/physrevfluids.6.064607.

[49] Lomb, N. R., "Least-Squares Frequency Analysis of Unequally Spaced Data," *Astrophysics and Space Science*, Vol. 39, No. 2, 1976, pp. 447–462. https://doi.org/10.1007/bf00648343.

[50] Scargle, J. D., "Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data," *Astrophysical Journal*, Vol. 263, 1982, pp. 835–853. https://doi.org/10.1086/160554.

[51] Scargle, J. D., "Studies in Astronomical Time Series Analysis. III. Fourier Transforms, Autocorrelation Functions, and Cross-Correlation Functions of Unevenly Spaced Data," *Astrophysical Journal*, Vol. 343, 1989, p. 874. https://doi.org/10.1086/167757.

7 Scientific Paper 2: Unreal Success: Vision-Based UAV Fault Detection and Diagnosis Framework

Unreal Success: Vision-Based UAV Fault Detection and Diagnosis Framework

José Ignacio de Alvear Cárdenas* and Coen de Visser†

Online fault detection and diagnosis (FDD) enables Unmanned Aerial Vehicles (UAVs) to take informed decisions upon actuator failure during flight, adapting their control strategy or deploying emergency systems. Despite the camera being a ubiquitous sensor on-board of most commercial UAVs, it has not been used within FDD systems before, mainly due to the nonexistence of UAV multi-sensor datasets that include actuator failure scenarios. This paper presents a knowledge-based FDD framework based on a lightweight LSTM network and a single layer neural network classifier that fuses camera and Inertial Measurement Unit (IMU) information. Camera data is pre-processed by first computing its optical flow with RAFT-S, a state-of-the-art deep learning model, and then extracting features with the backbone of MobileNetv3-S. Short-Time Fourier Transform is applied on the IMU data for obtaining its time-frequency information. For training and assessing the proposed framework, *UUFOSim* was developed: an Unreal Engine-based simulator built on AirSim that allows the collection of high-fidelity photo-realistic camera and sensor information with the possibility of injecting actuator failures during flight. Data were collected in simulation for the Bebop 2 UAV with 16 failure cases. Results demonstrate the added value of the camera and the complementary nature of both sensors with failure detection and diagnosis accuracies of 99.98% and 98.86%, respectively.

I. Nomenclature

$b_{\text{cam}}, b_{\text{IMU}}$	=	Camera and IMU buffers
C_{UE4}	=	Occupancy grid cell size in ICF_{UE4} length units
$f_{\text{cam}}, f_{\text{IMU}}$	=	Camera and IMU sampling frequency, HZ
f_{FDD}	=	FDD execution frequency, Hz
f_{p}	=	AirSim physics engine thread calling frequency
$f_{\text{res}}, t_{\text{res}}$	=	STFT frequency and time resolutions
k_{UE4}	=	Conversion factor between UE4 and AirSim coordinate frames
n_x, n_y	=	Occupancy grid coordinates
n_{win}	=	STFT window size, samples
o	=	STFT window overlap size, samples
$\overrightarrow{P}_i$	=	Waypoint coordinate vector
p_{i_x}, p_{i_y}	=	Waypoint coordinates in the occupancy grid coordinate frame
$\bar{p}_{i_x}, \bar{p}_{i_y}, \bar{p}_{i_z}$	=	Waypoint coordinates in the AirSim drone coordinate frame
$\overrightarrow{X}$	=	Vehicle state vector
x, y, z	=	Position coordinates, m
x_r, y_r, z_r	=	Reference position coordinates, m
x_{D_0}, y_{D_0}	=	Drone initial spawn coordinates in ICF_{UE4}
$x_{\text{UE4}}, y_{\text{UE4}}$	=	Position coordinates in ICF_{UE4}
v	=	Measurement noise
ψ_r	=	Drone reference heading, rad
ω	=	Propeller rotational speed, rad/s
Ω	=	Vehicle angular velocity, rad/s

*Graduate Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology
†Associate Professor, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology

II. Introduction

With the advent of Smart Cities, Unmanned Air Vehicles (UAVs) have seen a surge in their number of applications, from package delivery [1, 2] to Urban Air Mobility (UAM) [3]. Most recently, as a response to the COVID-19 pandemic [4], the implementation of UAVs for medical purposes has been accelerated. Zipline, a drone start-up in California (USA), has been granted permission for transporting medical supplies in North Carolina and AVY, a start-up based in Amsterdam (The Netherlands), has received a grant from the European Commission for urgent medical transport between healthcare facilities. With Air Traffic Control programs under development for the management of drones, such as the U-Space in Europe [5], one of the main concerns of the future crowded urban airspace is safety [6].

Most of the research in this field has been focused on fault tolerant control [7], with companies such as Verity Studios successfully filing a patent in 2020 for a final product [8]. However, in order to improve the resilience of multi-rotor and hybrid drones to potential failures, work is also carried out in obstacle avoidance [9], upset recovery [10], system identification [11] or fault detection and diagnosis [12]; the latter consisting of the fault classification, as well as its location and magnitude identification. Literature in actuator fault detection and diagnosis (FDD) is very extensive but it deals with a single failure type at a time and has been limited to the manipulation of signals from the Inertial Measurement Unit (IMU), namely the accelerometers and gyroscopes, or additional external sensors such as microphones or optical flow sensors [13, 14], which add weight and complexity to the system. Cameras are nowadays ubiquitous in commercial UAVs and they have been ignored for this task, even though their information is already being processed for navigation, such as Simultaneous Localisation and Mapping (SLAM) [15], and state estimation in GPS denied urban regions, such as Visual Inertial Odometry [16]. Visual information is very rich and it could potentially identify multiple failure types at once, as well as increase the accuracy when fused with the IMU sensor.

FDD expands the envelope of the UAV's self-awareness and allows informed decisions when deploying emergency systems, such as a parachute, and switching between controllers or internal physics models to counteract a failure. Figure 1 shows a classification of FDD methods from literature which can be divided in 3 main groups: model-based, signal-based and knowledge-based. Historically, knowledge-based approaches have shown to be the most suitable for dealing with high-dimensional visual data, especially with the rise of deep learning model architectures. In contrast with model-based approaches, they do not make use of the physical properties of the system and do not build a mathematical model. However, they require greater amounts of historic data in order to create implicit models.

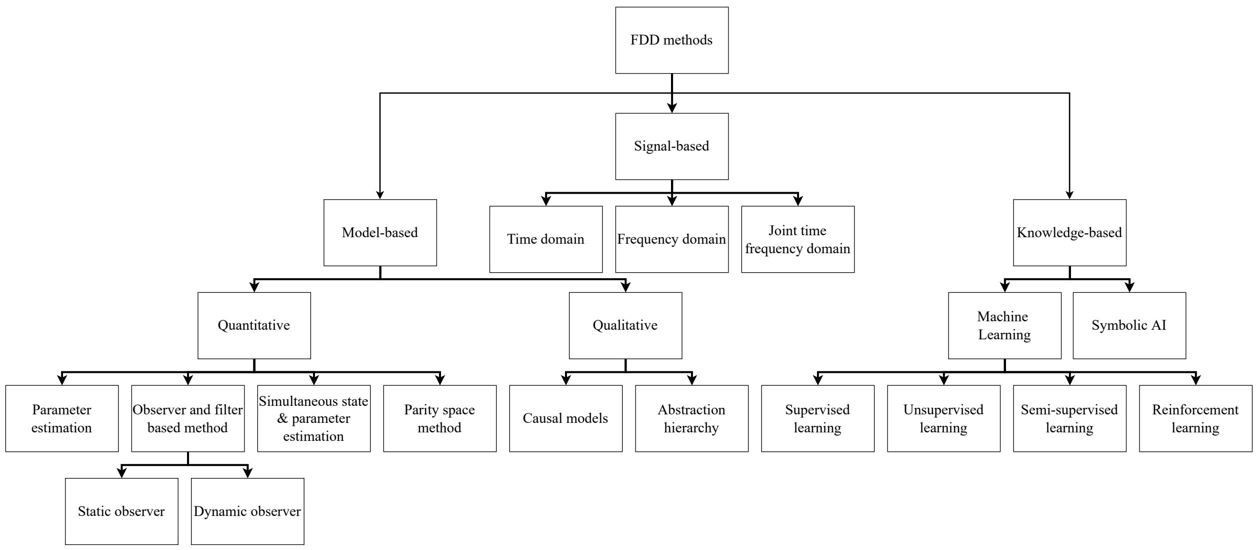


Fig. 1 Fault detection and diagnosis method taxonomy.

Machine learning methods are exploited in modern fault detection systems because they do not require a lot of computations, allowing their use online in real-time. Among them, Long Short-Term Memory (LSTM) networks are a supervised learning method that can be fed sequential data in order to extract temporal relationships to generate an output [17]. It generates predictions based on the current input and an internal state that stores information from an arbitrary number of previous inputs. It is usually combined with batch normalization (BN) to reduce the internal covariate shift and accelerate the convergence of the algorithm. Zhao et al. [18] exploit this combination to extract dynamic information from data for online FDD within chemical processes and they demonstrate its superiority over alternative knowledge-based FDD approaches.

For the development and performance assessment of vision-based FDD algorithms, it is required to have a dataset which includes IMU and camera output in nominal flight and in failure scenarios. Unfortunately, the current available datasets do not include IMU sensor information, such as the VisDrone dataset [19] or the Indoor Navigation UAV Dataset [20], and do not have any recorded scenarios with failures, such as the UZH-FPV Drone Racing Dataset [21] or the Zurich Urban Micro Aerial Vehicle Dataset [22].

Gathering large quantities of data for knowledge-based fault detection models with an UAV is very time consuming, dangerous and expensive; data would have to be annotated, multiple failure modes would have to be induced in the vehicle and the flight environment, as well as the UAV, would have to be adapted to minimise the potential risk. Besides that, in an experimental physical setting it is very difficult to collect data from various environments and conditions. A suitable alternative is the simulation of the vehicle in a realistic environment, the storage of the sensor synthetic data for model training and the transfer learning to the real world UAV. It has been observed that the addition of large quantities of synthetic data to a smaller real dataset would lead to a performance increment when compared to the scenario in which only real data is collected [23].

In previous literature, Gazebo has been the quadrotor simulation tool of choice by the research community leading to simulators such as RotorS [24] and Hector [25]. Despite its high-fidelity physics engine, the output quality of its visual cues is far from photo-realistic. In this regard, there has been an effort in developing high-fidelity simulators for computer vision tasks in the last 10 years. Blender's Game Engine was used for the development of simulators, such as MORSE [26] from 2011 to 2016. However, Unity and Unreal Engine have become the new state-of-the-art (SOTA). Besides their photo-realism, these engines have the benefit of providing an online asset marketplace for the generation of an infinite number of simulation environments.

Examples of photo-realistic simulators are FlightGoggles [27] and Flightmare [28] developed in Unity, and UnrealCV [29], Sim4CV [30] and AirSim [31] developed in Unreal Engine. AirSim was launched in 2017 by Microsoft as an open-source simulator built on Unreal Engine 4 (UE4) for AI research. It is a modular framework that fosters the simulation of autonomous drones and ground vehicles with realistic physics and visual cues. It also includes C++ and Python APIs that allow the researcher to interact programmatically with the vehicle for the extraction of state and sensor information, as well as for providing vehicle control inputs. In contrast with the other simulators, it has an adaptable framework for the introduction of new vehicle models and it is well documented. Thanks to its modularity, later works have been built on AirSim for specialised applications, such as the AirSim Drone Racing Lab [32]. Another promising simulator is Isaac Sim[*] developed by NVIDIA for the development and deployment of artificial learning applied to robotics in their Omniverse simulator environment. Unfortunately, its computational requirements are beyond the specifications of most commercially available workstations.

The main contribution of this paper is an LSTM-based online FDD framework that fuses camera and IMU data. To this end, the camera information is pre-processed by a SOTA optical flow model in order to extract the magnitude and direction of the vehicle's ego-motion. The IMU data is passed through a Short-Time Fourier Transform for feature extraction. To the authors' knowledge, it is the first time that both sensor sources are combined for UAV FDD. Furthermore, this paper also presents *UUFOSim* (Unreal UAV Failure injectiOn Simulator), a data gathering pipeline built on AirSim for the collection of synthetic flight data with actuator failures in a urban environment.

The potential of UUFOSim has been demonstrated for the Parrot Bebop® 2 UAV. Its aerodynamic model is available from literature [11, 33] and it has been complemented with the blade damage model from [34]. The FDD framework was run at 10 Hz and it had to distinguish between 17 states: 16 failure states, namely four levels of blade damage failure for each of the four propellers, and a healthy state. The results show the added value of the camera-IMU combination versus their isolated performances.

The remainder of this paper is organized as follows. Section III describes the data gathering pipeline within UUFOSim. Section IV covers the FDD framework and provides the details of the camera and IMU data pre-processing. Then, section V presents the results when both contributions are applied to the Bebop 2 platform. Finally, concluding remarks and recommendations for further work are provided in section VI.

[*]`https://developer.nvidia.com/isaac-sim`

III. UUFO Simulator

The simulator, that the authors have named Unreal UAV Failure injectiOn Simulator (UUFOSim), consists of flying a simulated drone or Undiagnosed Failing Object (UFO) in a urban environment avoiding obstacles between two random locations. UFOs fly at a uniformly sampled constant altitude and an actuator failure, within a set of modes, is injected at a random point along the trajectory. During the whole flight, including the manoeuvres after the failure, the camera and IMU data is stored to later shape the dataset for the training and testing of the FDD framework.

Figure 2 shows the three main blocks that shape the data gathering pipeline. Once the flight is concluded by achieving one of the terminating conditions, the environment and drone are reset to their original state and the cycle is repeated. The loop continues for as many flights as it is desired for building the dataset.

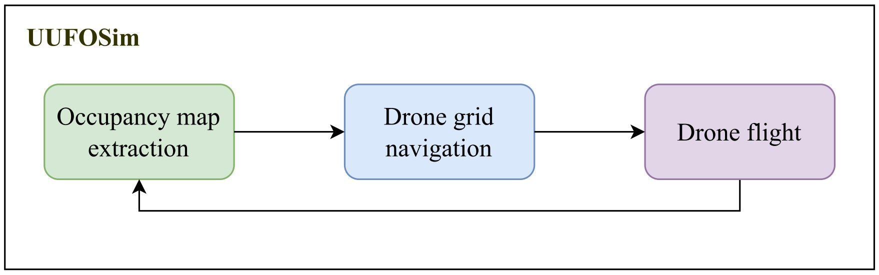


Fig. 2 Data gathering pipeline block diagram

In order to discuss in detail each of the presented blocks, this section will first present how the information from the environment is extracted offline and translated to an occupancy grid in subsection III.A. Once the environment state is known, the path that the drone should follow is computed in subsection III.B using common robot path planning algorithms. Finally, the drone flight including the sensor data collection and fault injection during flight are discussed in subsection III.C.

A. Environment and occupancy map

The environment is discretised into a matrix of inter-independent fix size cells which store whether they are free or occupied in the form of a boolean. This form of representation is called a grid occupancy map and has been exploited in the autonomous driving industry [35, 36], and more recently in the (flying) robotics sector [37, 38], to reduce the environment information to a tractable and efficient data structure.

Figure 3 describes all the steps taken to build a static 2D grid occupancy map in order to encapsulate all the information about the static obstacles found by the drone at its flying altitude prior to executing its flight.

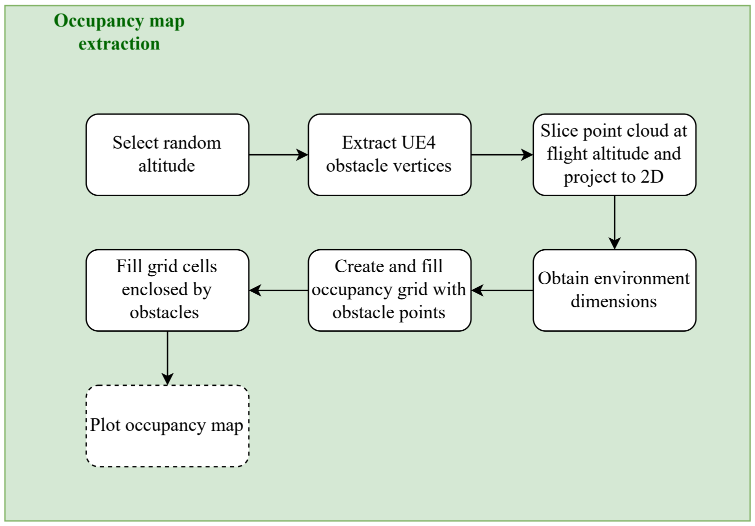

Occupancy map extraction
Select random altitude
Extract UE4 obstacle vertices
Slice point cloud at flight altitude and project to 2D
Obtain environment dimensions
Create and fill occupancy grid with obstacle points
Fill grid cells enclosed by obstacles
Plot occupancy map

Fig. 3 Occupancy map extraction block diagram.

First, the flight altitude is randomly selected within a range of possible altitudes. The flight altitude changes between iterations in order to prevent the overfitting of the FDD algorithm to object instances found at a certain height, like bushes, trees or windows, as well as the location of the horizon line in the drone captured images.

For the extraction of obstacle information, AirSim provides an API that returns UE4 assets' triangular static meshes (as can be seen in Fig. 4 for a sphere) as a Face-Vertex Mesh. This results in a 3D point cloud in which each point has a label for an object in the environment.

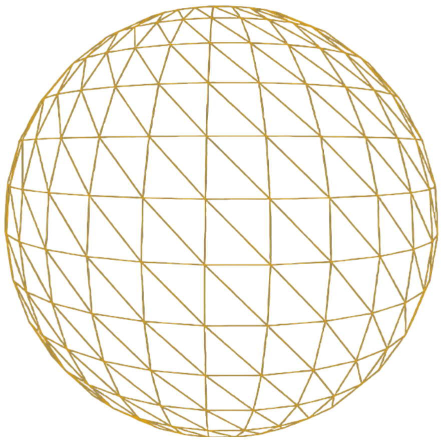


Fig. 4 Sphere mesh in Unreal Engine 4.

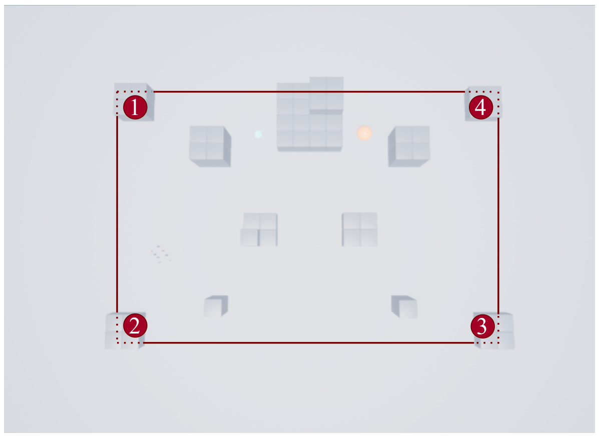


Fig. 5 Blocks environment limits for drone flight bounded by the 4 monoliths in the red rectangle corners.

Figure 5 shows the Blocks environment that will be used to showcase the steps carried out on the grid occupancy map. It consists of grey blocks, an orange sphere, a blue cone and multiple cylinders on the environment's left centre part. Since it is desired that the drone flies exclusively around the obstacles of the environment, its limits are defined by the furthest points in the 2D point cloud x and y directions. For the Blocks environment, the drone flights are confined to the red rectangle shaped by the four corner monoliths, resulting in a 3D point cloud of obstacles of 46,248 points.

Moreover, the drone maintains constant altitude during a flight. Thanks to this assumption, the point cloud can be reduced by slicing it and storing only the points found within a range around that altitude. The 3D points are then projected to the 2D x-y plane reducing them to a 2D point cloud.

Fig. 6 shows the 2D point cloud of the obstacles in the scene when the chosen drone flight altitude is seven metres with an altitude range of three metres. The points that are close to each other with the same colour are part of the same object. The current figure has 8,956 2D points because there are many overlapping projected vertices from multiple altitudes. This abundance of points can be seen when zooming to the orange blob in Fig. 7. Despite having reduced the number of obstacle points by a factor of five (and the information by almost a factor of eight, since the altitude information has been discarded), there are many points that do not provide any information; only the outer edge of each group of objects carry information that should be preserved for the representation of obstacles in the environment.

The solution to the problem of unnecessary points that should be discarded is the introduction of the occupancy map. The environment is discretised in cells and those with points within its boundaries are considered as occupied (black) whereas those with no points are empty (white). As a result, independently of how many points are within a cell, they are translated to a single data point. For the Blocks example, the 2D point cloud is transformed from the world coordinate frame to the grid coordinate frame, projecting all points into the 2D grid and filling all the cells occupied by obstacle points. This process can be observed applied to the Blocks environment in Fig. 8 and Fig. 9. From this point, only the grid information is passed along the FDD data gathering pipeline, decreasing the stored data and further computational load. Instead of the initial 8,956 2D points considered before the occupancy grid implementation, now a grid of 80 by 54 cells is used, resulting in a total of 4,320 cells. Given that now the data points (cells) are homogeneously distributed, the information about their location does not have to be stored as long as the x and y dimensions of the occupancy grid are known, meaning that the 2D information has been transformed to a 1D data stream. As a result, the number of data points has been reduced by a factor of two, whereas the amount of information stored has been further reduced by a factor of four. Table 1 summarises the evolution of the number of points and coordinates (information) through the aforementioned projection and occupancy grid stages.

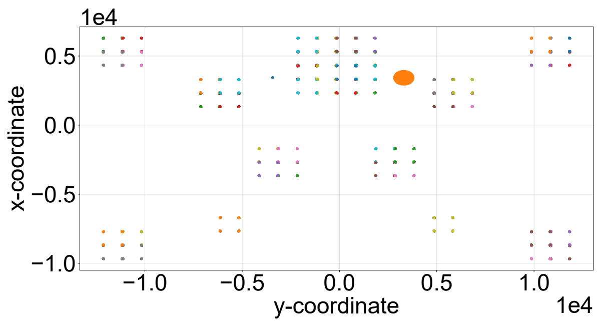


Fig. 6 2D projection of the Blocks environment object vertices within 4 and 10 metres altitude.

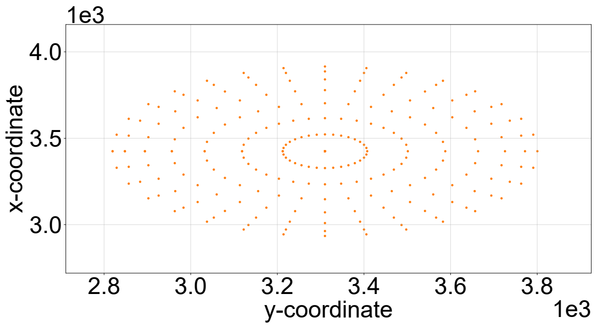


Fig. 7 Zoom-in of the 2D projected points of the sphere.

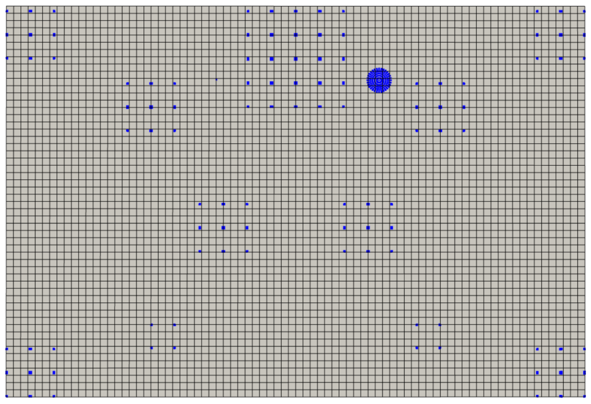


Fig. 8 2D points projected in empty occupancy grid.

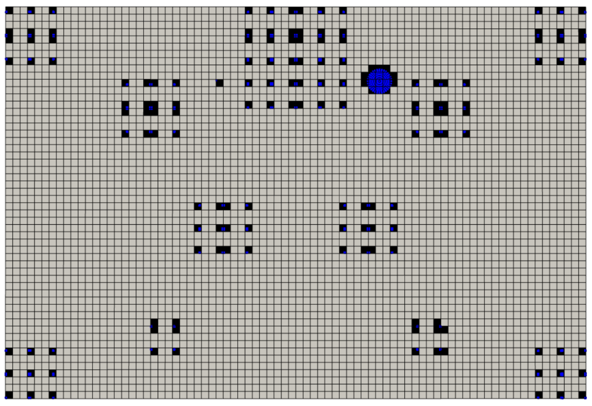


Fig. 9 Filled occupancy grid and 2D projected points.

Table 1 Evolution of the number of points and coordinates upon the occupancy map generation.

	Original	2D projection	Occupancy map
Points	46,248 (100%)	8,956 (19.36%)	4,320 (9.34%)
Coordinates	138,744 (100%)	17,912 (12.91%)	4,320 (3.11%)

As can be observed from Fig. 9, filling the grid cells occupied by obstacle vertices is not enough for creating a reliable occupancy map. There are grid cells that lie within objects, that should not be accessible but that are not marked as occupied since there is no vertex of the static mesh on that particular cell. This problem worsens the finer the mesh of the occupancy map. To solve this, the following algorithm was developed that exploits the Delaunay triangulation:

1) First, each of the objects in the environment is assigned the coordinates of the grid cells occupied by their remaining 2D points projected on the occupancy map. Those objects with less than three grid coordinates or whose coordinates shape a line along the x or y axis are discarded from this process since they can not enclose other cells.
2) For each of the remaining objects, the grid cells that define the outer edge of their described polygon are identified. For that purpose, Delaunay triangulation is used, which creates a triangular mesh of the object grid cells. It is looped over all the created triangles for each of the objects and, in the case that an edge is covered more than once, then it is an internal edge shared by two triangles. Therefore, it is discarded as a potential polygon outer edge.
3) Once the outer edges of the obstacle polygon are identified, is it looped over all grid coordinates and assessed whether they lie within the polygon boundaries. After going through all the empty grid coordinates for each of the (obstacle) polygons, all the grid cells located within obstacles have been identified.

To conclude, all the grid cells found within the obstacles are marked as occupied within the occupancy map, leading to Fig. 10 for the Blocks environment. This occupancy map is passed on to the path planning module.

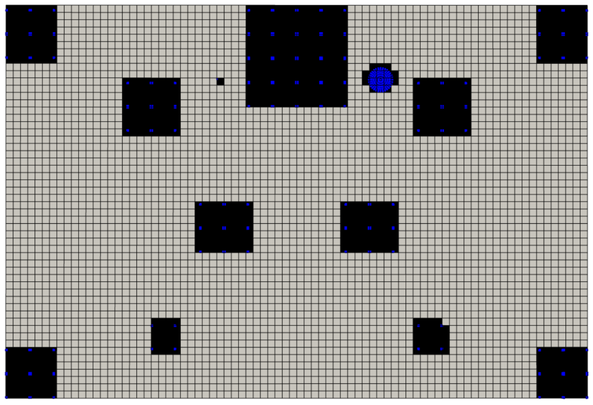


Fig. 10 **Filled occupancy grid considering obstacle inner cells identified with Delaunay triangulation.**

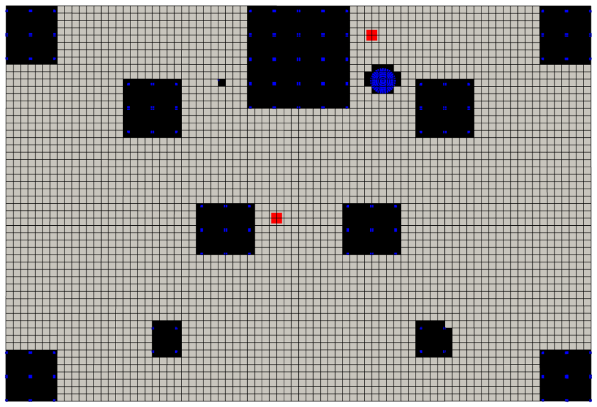
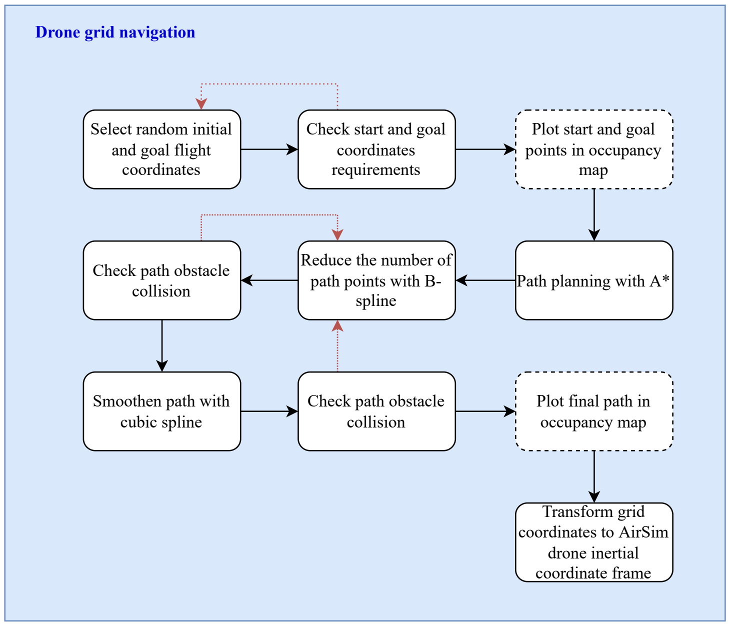


Fig. 11 **Occupancy grid with start and goal locations.**

B. Path planning

The steps to achieve a smooth drone flight path can be observed in Fig. 12.


Drone grid navigation
Select random initial and goal flight coordinates
Check start and goal coordinates requirements
Plot start and goal points in occupancy map
Path planning with A*
Reduce the number of path points with B-spline
Check path obstacle collision
Smoothen path with cubic spline
Check path obstacle collision
Plot final path in occupancy map
Transform grid coordinates to AirSim drone inertial coordinate frame
Fig. 12 **Drone grid navigation block diagram.**

1. Start and goal selection

First, a random initial and goal flight coordinates are generated, and it is verified whether those grid locations fulfil three design requirements:

1) The distance between the start and end location is greater than a minimum distance given as input by the user. This prevents extremely short paths which do not allow the injection of failures.
2) The distance between the start and the end location is smaller than a maximum distance given as input by the user. This prevents extremely long paths that would decrease the number of flights that would be executed in the allocated time for data collection and do no add much value to the training of the FDD framework.
3) The start and goal locations have to be located beyond a minimum distance from all identified obstacles in the occupancy map. This guarantees that there will not be any unexpected collision due to the drone dimensions.

The random selection and requirement check is repeated until the start and goal coordinates fulfil the established requirements. Once that is the case, they are included within the occupancy map. For the Blocks example, the start and goal location can be seen in Figure 11 as red dots.

2. Path planning algorithm selection

Only two types of classic robot path planning methods are considered, namely grid or discrete approaches and road-map methods. A visual classification of the algorithms considered in the present research can be observed in Fig. 13.

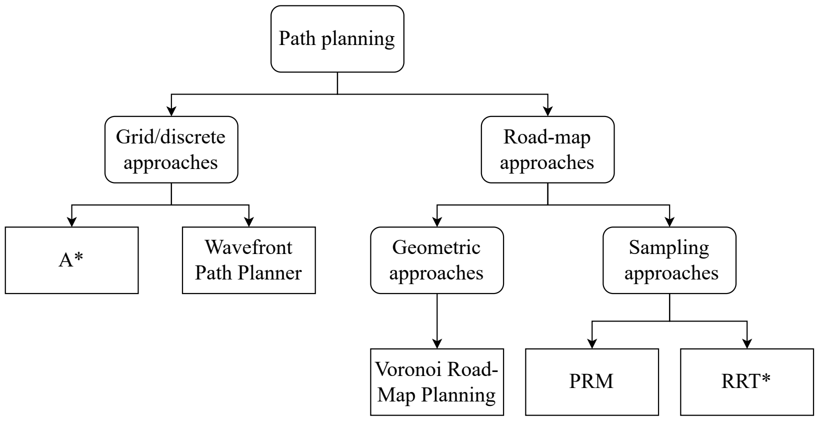


Fig. 13 Path planning algorithm classification

Both methods within the first group consist of two steps, namely a propagation or search step and a back propagation step. The main difference between both methods within the first group is that the Wavefront Path Planner applies its initial search (wave propagation) throughout the complete grid whereas A* uses a function to decide which cells are worth inspecting given the already discovered solution space. As a result, even though every iteration in the search step of the A* algorithm is more computationally expensive, less iterations need to be considered since only a portion of the grid is inspected.

The main difference between the sampling options is that in PRM the drone must pass through randomly sampled points in the solution space whereas in RRT* it only has to move in their direction; it does not require point-to-point convergence.

All the approaches when adapted can maintain a tunable safe distance from the obstacles in the environment, except for the Voronoi Road-Map planning that tries to maximise this distance. When applied to the Blocks environment, A* shows the lowest computation time when compared to the rest, followed by Voronoi Road-Map planning. Wavefront Path Planner and PRM show similar computation time, whereas RRT* is the worst performing, requiring a time 17 times higher than A*. From this analysis, A* algorithm is the algorithm of choice. Figure 14 shows the A* path for the Blocks example.

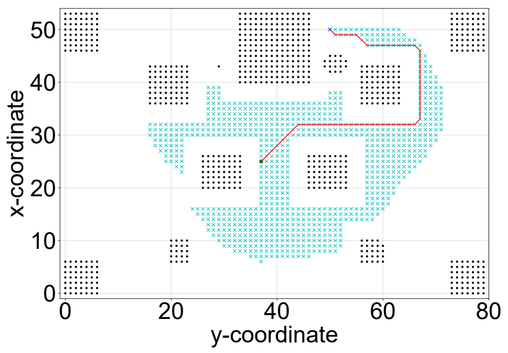


Fig. 14 A* Path Planner applied to the Blocks environment example.

3. B-spline path point number reduction

There are many points that shape the final flight path that do not provide valuable information. All the points along straight lines can be suppressed and reduced to two points before smoothing the trajectory. Additionally, some curves could be avoided if straight lines were taken. Whereas A* provided a flight path from the start to the goal using points separated by the size of an occupancy grid cell, the B-spline could be used to reduce the path to its most indispensable points.

B-splines [39] of order k are piece-wise polynomials that serve as the basis of spline functions and are capable of generating smooth trajectories connecting a provided set of data points. They are of degree k-1 and k-2 times continuously differentiable. For the present research, B-splines of degree 2 are used, which means that they are of order 3 and are 1 time continuously differentiable. Since the goal is to create an alternative using only the most indispensable points, the algorithm starts aiming at creating a path only with 5% of the points. If unsuccessful, this point reduction strategy is repeated increasing the percentage of kept points by 5% every time. If the percentage of kept points reaches 100%, then reduction of points is deemed not possible and the A* generated path is passed on to the next step in the path planning pipeline.

In the case that a reduced path is computed, it is checked for collisions with the obstacles in the environment. To that end, the vectors connecting each pair of points along the new path are discretised and it is checked whether the grid cells of which they are part of are occupied. Additionally, this process is repeated with two parallel vectors displaced one cell to the right and to the left of the original vector in order to maintain at least one cell distance from all the present obstacles. In the example presented in Fig. 15, even though the central vector connects the two path points through open space, an obstacle is detected because one of the points of the right parallel vector can be found within an occupied grid cell. The positive detection of collision with an obstacle has the same effect as an unsuccessful reduction of path points: the percentage of kept points is increased by 5% and a new B-spline is generated whose reduced flight path would be again checked for obstacles.

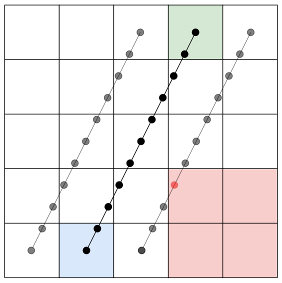


Fig. 15 B-spline reduced path obstacle detection. The blue and green cells are two path points, and the red cells are occupied by environment obstacles. The black lines are the discretised vectors for obstacle detection.

The benefits of B-spline path point number reduction can be visually appreciated in the occupancy grid. Figure 16 and Fig. 17 show the flight path before and after the B-spline path point reduction was applied. As can be seen, Fig. 16 is the same path as presented in Fig. 14 and it consists of 61 points. In contrast, Fig. 17 only required 9 points to carry out the same path.

4. Cubic spline path smoothing

Computation of the cubic spline with the A* flight path as input would have not led to any considerable smoothing given the fine path discretisation to single occupancy grid cells. The B-spline allowed the discovery of the pivot points on which the cubic smoothing spline can be built.

The smoothing is required in order to avoid sharp corners. The distance between each point in the final smooth flight path is again 1 occupancy grid cell size but the points are not constrained to the corner of each cell anymore, as it was the case for the A* and B-spline reduced flight paths. Finally, it is checked for collision with obstacles using the same approach as with the B-spline.

The red arrows in Fig. 18 show the final flight path and Fig. 19 visually confirms that the cubic spline passes through the pivot points (yellow circles) and the flight path points are not confined to the occupancy grid cells. Additionally, the improvement thanks to the presented approach (small red arrows), which combines B-splines and cubic splines, becomes evident when compared to the original A* flight path (small green arrows).

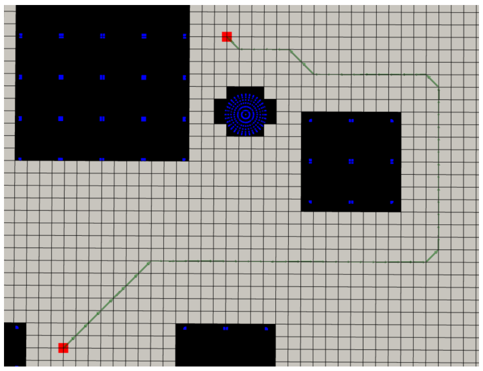


Fig. 16 Zoom-in of Blocks occupancy map with A* flight path represented by small green arrows.

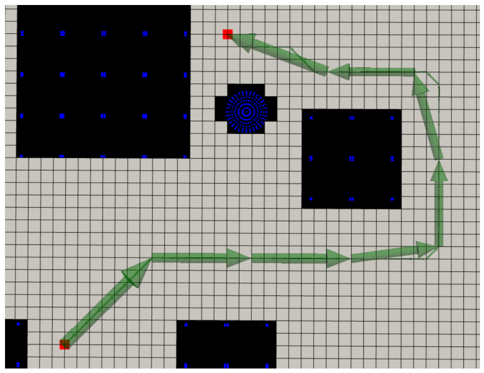


Fig. 17 Zoom-in of Blocks occupancy map with B-spline reduced flight path represented by large green arrows.

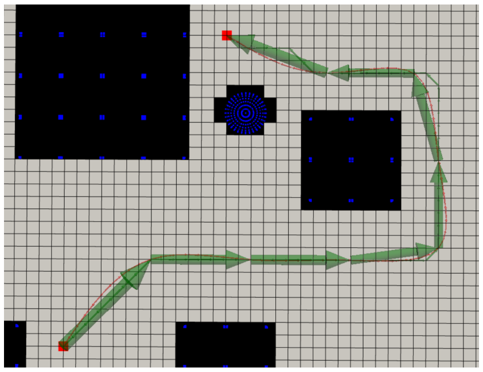


Fig. 18 Zoom-in of Blocks occupancy map with cubic spline smoothed flight path represented by small red arrows.

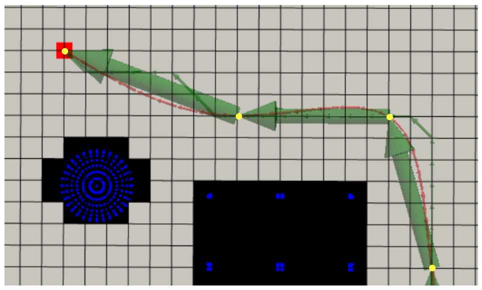


Fig. 19 Visual confirmation of the final path not being constrained to the occupancy grid and it passes through B-spline pivot points (yellow circles).

5. Flight path transformation to AirSim drone inertial coordinate frame

Finally, the grid coordinates of the smoothed spline flight path are transformed to the AirSim drone inertial coordinate frame. For that purpose, first the 4 inertial coordinate frames used for the data gathering pipeline are defined next:

1) **Unreal Engine 4 inertial coordinate frame (ICF$_{\mathbf{UE4}}$)**: it is the coordinate frame used to build the environment. Therefore, the centre of coordinates and the direction of its axes vary from map to map. It is a drone independent coordinate frame.
2) **Occupancy grid inertial coordinate frame (ICF$_{\mathbf{OG}}$)**: it has its origin at the bottom left of the occupancy map with the y-axis pointing to the right and the x-axis pointing to the top in the 2D grid. All the objects and points in the occupancy map have positive coordinates. The environment has been discretised with cells of predefined size, C$_{\text{UE4}}$. As can be seen in Eq. (1) and Eq. (2), in order to define the number of cells in the grid (n_x and n_y), the minimum and maximum X- and Y-coordinates among all the obstacles in UE4 environment ($x_{\text{UE4}_{min}}$, $x_{\text{UE4}_{max}}$, $y_{\text{UE4}_{min}}$, $y_{\text{UE4}_{max}}$) are required. It is a drone independent coordinate frame.

$$n_x = (x_{\text{UE4}_{max}} - x_{\text{UE4}_{min}})/C_{\text{UE4}} \tag{1}$$

$$n_y = (y_{\text{UE4}_{max}} - y_{\text{UE4}_{min}})/C_{\text{UE4}} \tag{2}$$

3) **AirSim inertial coordinate frame (ICF$_{\mathbf{AS}}$)**: it has its origin at the same location as ICF$_{\text{UE4}}$ with its axes pointing in the same directions. The only difference is the scale of its units: 1 unit in ICF$_{\text{AS}}$ is equivalent to 100 units in ICF$_{\text{UE4}}$. This factor is defined as k$_{\text{UE4}}$. It is a drone independent coordinate frame.

4) **AirSim drone inertial coordinate frame ($\mathbf{ICF_{ASD}}$)**: it is the same as ICF_{AS} with the only difference that its origin has been shifted to the location where the drone is spawned for the first time within the environment. The location of the drone within the controller is expressed using this inertial coordinate frame. The drone spawn coordinates in the ICF_{UE4} are given as x_{D_0} and y_{D_0}. It is a drone dependent coordinate frame.

In order to transform the flight path from the occupancy map to the AirSim drone inertial coordinate frame, the transformations in Fig. 20 were used, using the shown parameters.

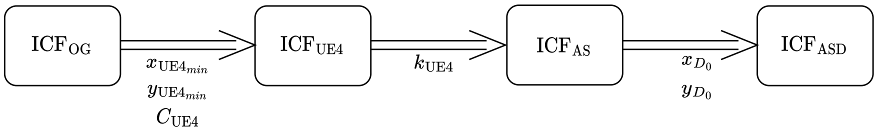


Fig. 20 Inertial coordinate frame transformations: from $\mathbf{ICF_{OG}}$ to $\mathbf{ICF_{ASD}}$

Given a trajectory T: $[(p_{1_x}, p_{1_y}), (p_{2_x}, p_{2_y}), ..., (p_{n_x}, p_{n_y})]$ of n points in $\mathbb{R}^2$ in ICF_{OG}, the points of the flight path can be transformed to ICF_{ASD} ($\bar{p}_{i_x}$ and $\bar{p}_{i_y}$) with Eq. (3) and Eq. (4).

$$\bar{p}_{i_x} = (p_{1_x} \cdot C_{\mathrm{UE4}} + x_{\mathrm{UE4}_{min}})/k_{\mathrm{UE4}} - x_{D_0} \quad i = 1, 2, ..., n \tag{3}$$

$$\bar{p}_{i_y} = (p_{1_y} \cdot C_{\mathrm{UE4}} + y_{\mathrm{UE4}_{min}})/k_{\mathrm{UE4}} - y_{D_0} \quad i = 1, 2, ..., n \tag{4}$$

C. Data collection

The next step is to fly the drone within the UE4 environment, potentially induce an actuator failure and gather all the vision-based and signal data for the FDD training. Figure 21 summarises all the steps taken during the final block of the data gathering pipeline. In the following sections, each of the blocks will be briefly discussed.

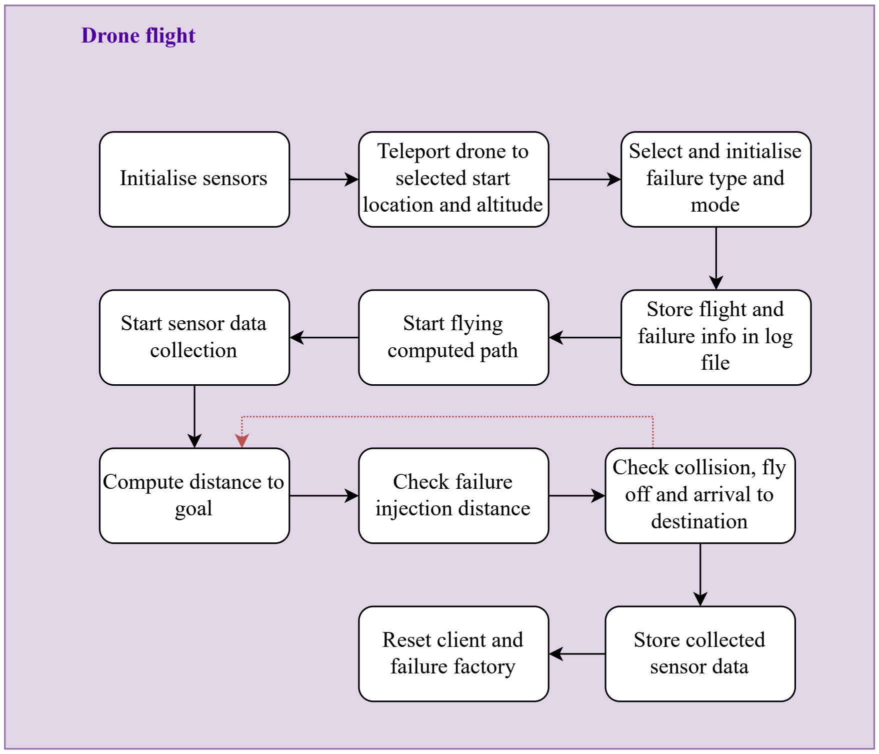

Drone flight
Initialise sensors
Teleport drone to selected start location and altitude
Select and initialise failure type and mode
Store flight and failure info in log file
Start flying computed path
Start sensor data collection
Compute distance to goal
Check failure injection distance
Check collision, fly off and arrival to destination
Store collected sensor data
Reset client and failure factory

Fig. 21 Drone flight block diagram

1. Sensor initialisation and drone teleportation

In the sensor initialisation stage, the data structures in charge of storing the IMU and camera data are created. A single drone can carry multiple cameras, each generating different information such as depth, segmentation or RGB.

Once the sensors have been initialised, the drone is teleported to the start location with the heading already pointing towards the first path point in the trajectory. The user can specify whether it is desired for the drone to take-off from the ground or it should be initialised directly at the chosen altitude. For the purpose of the current research, in which the desire is to analyse failures during the cruise phase, the drone is already teleported to the right altitude, as can be observed in Fig. 22 for the Blocks environment.

Fig. 22   Drone teleported to start location

2. Failure type & mode selection and initialisation

Four actuator failure types are considered: actuator saturation, when the propeller is locked rotating at its maximum rotational rate; actuator lock, when the actuator is locked as a percentage of its maximum rotational rate; propeller fly-off, when the propeller is detached completely from the motor; and propeller damage, when one or more blades are broken. The first three failures are simulated by ignoring the controller propeller rotational rate outputs in favour of predefined locked values. As an example, the propellers of the Bebop 2 drone can attain a maximum rotational rate of 1256 rad/s. When the actuator of a propulsion unit suffers a saturation failure, its rotational rate fed to the physics model is locked at 1256 rad/s. In the case of being a locked propeller at 50%, then the physics model is fed a locked value of 628 rad/s (1256/2). If the propeller has flown-off, then a locked value of 0 rad/s is used. In all these scenarios, the forces and moments change but no oscillatory behaviour is observed.

To simulate the fourth actuator failure type, the Blade Element Theory (BET) model presented in [34] was implemented as a plug-in to the nominal vehicle physics model. Blade damage requires a closer look at the propeller aerodynamics and centre of gravity shift, which create a loss of thrust and vibrations along the three body axis. These aerodynamic and mass effects can be observed in Fig. 23 and Fig. 24, which show the BET-simulated forces and moments for the Bebop 2 drone front left propeller after suffering 20% damage while rotating at 600 rad/s. In terms of model performance, in the presence of up to 25% blade damage and with the same UAV platform, de Alvear et al. [34] report a relative model error between 5% and 24% when simulating high propeller rotational speeds. At low propeller rotational speeds, the same relative error is reported to oscillate between 15% and 75%. Additionally, since the BET model airfoil lift and drag coefficients were identified for this particular UAV platform at low vehicle velocities, it is the most accurate when flying at speeds up to 3.6 m/s.

Each failure type has a different number of potential failure modes. For example, the propeller fly-off has one failure mode for each propeller, therefore there are four failure modes for this failure type. Additionally, there exist two hyper-parameters that can increase or limit the number of failure modes for each failure type, namely discrete vs continuous and abrupt vs linear. In the continuous case, the degree of failure of the damaged propeller and locked actuator failure types can be given any value in the open interval (0,1), whereas in the discrete case they can only obtain a value of the list <0.2, 0.4, 0.6, 0.8>. In the abrupt case, the failure takes place in a single time step once it is induced, whereas in the linear case it is linearly transitioned from the nominal to the desired fault state. As an example, if the propeller is rotating at 600 rad/s, the desired locking coefficient is 0.2, the maximum rotational speed is 1256 rad/s and the failure linear rate of change is -0.15, then the locking coefficient will linearly change from 0.478 (600/1256) to 0.2 in 1.85 seconds.

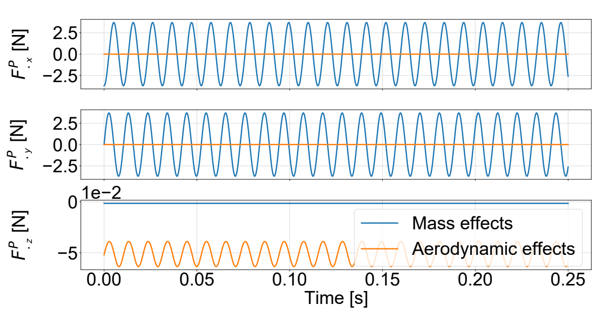


Fig. 23 BET-simulated evolution of mass and aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s [34].

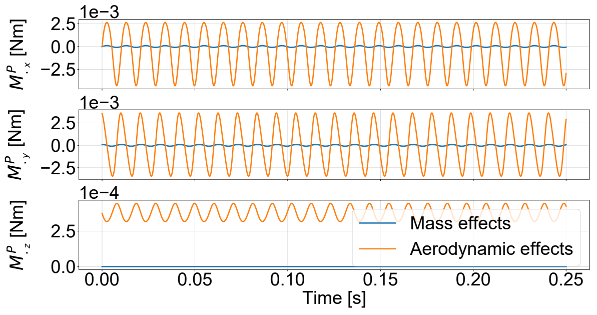


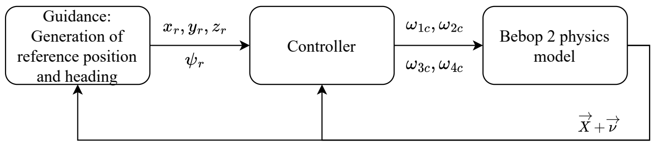
Fig. 24 BET-simulated evolution of mass and aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s [34].

In order to acquire a balanced data set which could be used for the FDD algorithm training, the user chooses the failure types and modes to include in the data set, and a pool of potential combinations is created from which it is uniformly sampled before every flight. As an example, if the user chooses discrete and abrupt actuator saturation, as well as discrete and abrupt actuator lock, then for each flight the algorithm will randomly sample with a uniform distribution from a pool of 21 alternatives, namely four actuator saturation options (one per propeller), 16 actuator lock options (four per propeller) and one healthy option (no damage).

Moreover, before each flight, a random distance along the planned trajectory is chosen for failure injection that is at least five metres from the start and goal locations in order to avoid capturing the transients present at the flight initialization and completion.

3. Drone flight: guidance, control and physics model

The drone is commanded to fly the computed path. For that purpose, the simulation loops over the three blocks shown in Fig. 25. First, with the trajectory path points defined in the path planning phase, the guidance block creates the reference position (x_r, y_r, z_r) and heading (ψ_r) for the controller given the current vehicle states ($\overrightarrow{X}$) polluted with measurement noise (ν). Second, the controller translates the desired position and heading into commanded actuator rotation velocities (ω_{ic}| i=1,2,3,4) taking into account the current states of the vehicle. Finally, the physics model block simulates the effect of those commands on the drone and provides the states at the next time step as measured by the vehicle sensors.



Guidance: Generation of reference position and heading
x_r, y_r, z_r
ψ_r
Controller
ω_{1c}, ω_{2c}
ω_{3c}, ω_{4c}
Bebop 2 physics model
$\vec{X}+\vec{\nu}$

Fig. 25 Drone flight guidance, controller and physics model pipeline.

Until know, the simulation pipeline is platform agnostic. However, a UAV has to be chosen for the introduction of the controller and the physics model, as well as for the demonstration of the FDD framework. For this paper, the Bebop 2 drone is the UAV of choice. In order to simulate as close as possible to reality its behaviour, the authors implemented within the C++ API of AirSim the Incremental Nonlinear Dynamic Inversion (INDI) controller [7] and the gray-box physics model [11, 33] developed at Delft University of Technology.

The used numerical integration scheme is the Beeman and Schofield explicit method [40] outlined in Eq. (5) and Eq. (6), where x is the position or attitude, v is the linear or angular velocity and a is the linear or angular acceleration. Additionally, Δt is the time step duration and the subscripts n+1, n and n-1 refer to the next, current and previous time steps. It is designed for second order ordinary differential equations, necessary to translate accelerations into positions. This approach substitutes the modified velocity Verlet algorithm [41] for the integration of the linear and angular acceleration, as well as the Euler method for the computation of position and orientation, used within AirSim by default.

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{6}\left(4a_n - a_{n-1}\right)\Delta t^2 \qquad (5) \qquad\qquad v_{n+1} = v_n + \frac{1}{6}\left(2a_{n+1} + 5a_n - a_{n-1}\right)\Delta t \qquad (6)$$

Contrary to the controller and physics model that originate from previous work at TU Delft, the authors exploited the guidance approach already implemented within AirSim and applied some modifications. At its core, this method provides at every time step an intermediate reference position on the line connecting the previous and next waypoints at a user-predefined "Look Ahead" (LA) distance from the current location projected on that line. Figure 26 illustrates an example scenario that will aid in the visual explanation of this reference position computation. Three trajectory waypoints are shown as $\vec{P}_1$, $\vec{P}_2$ and $\vec{P}_3$. Furthermore, the drone is initially positioned at $\vec{x}_{(n-1)}$, as represented by a red diamond. When the vehicle is at a waypoint or along the line connecting two consecutive waypoints, as it is the case here, the next reference position is at an LA distance from the current location along that line. The current reference position $\vec{x}_{r_{(n-1)}}$ is marked as a green hexagon in the figure. The vector that connects the reference location with the vehicle position projected along the waypoint connecting line is called the "Goal Vector" ($\overrightarrow{GV}$), and it is computed by subtracting the reference and actual positions, as shown in Eq. (7). In this case, the projected position $\vec{x}_{(n-1)_p}$ coincides with the actual drone location $\vec{x}_{(n-1)}$ and the departure waypoint $\vec{P}_1$, but that is not always the case.

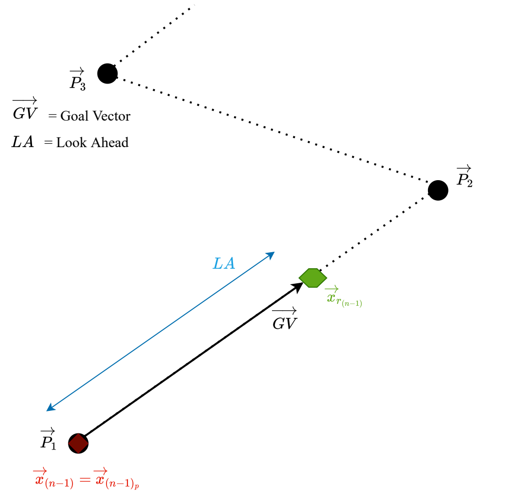


Fig. 26 Guidance block approach at time step n-1.

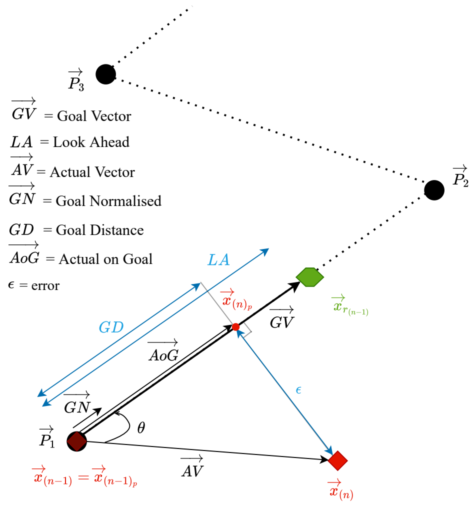


Fig. 27 Guidance block approach at time step n: geometry before reference position definition.

$$\overrightarrow{GV} = \vec{x}_{r_{(n-1)}} - \vec{x}_{(n-1)_p} \qquad (7)$$

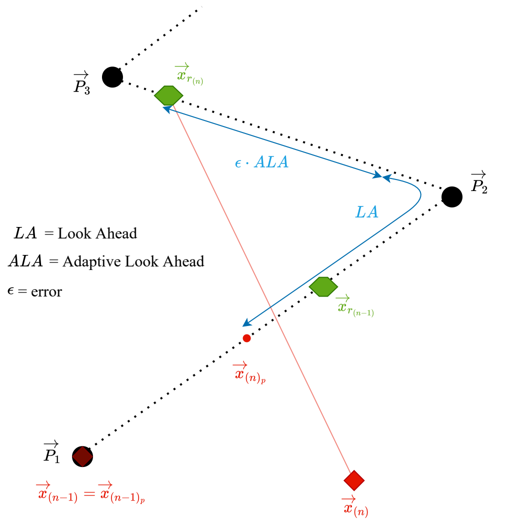
Next, Fig. 27 shows the next time step as the drone has flown away from the line connecting the waypoints and is now found at $\vec{x}_{(n)}$. As it can be seen, the vehicle has travelled the "Actual Vector" ($\overrightarrow{AV}$) and has covered a "Goal Distance" (GD) along the line connecting the previous and next waypoints. The former is the difference between the current and previous coordinates, as shown in Eq. (8). The latter can be computed using the dot product between $\overrightarrow{AV}$ and the normalised $\overrightarrow{GV}$ vector (Eq. (9)), also known as "Goal Normalised" or $\overrightarrow{GN}$, as shown in Eq. (10). The vector connecting the progress along the waypoint line, namely the difference between $\vec{x}_{(n)_p}$ and $\vec{x}_{(n-1)_p}$, is called "Actual on Goal" ($\overrightarrow{AoG}$) and it is computed according to Eq. (11).

$$\overrightarrow{AV} = \vec{x}_{(n)} - \vec{x}_{(n-1)} \qquad (8) \qquad\qquad \overrightarrow{GN} = \frac{\overrightarrow{GV}}{|\overrightarrow{GV}|} \qquad (9)$$

$$GD = \overrightarrow{AV} \cdot \overrightarrow{GN} = |\overrightarrow{AV}| \cos\theta \qquad (10) \qquad\qquad \overrightarrow{AoG} = GD \cdot \overrightarrow{GN} \qquad (11)$$

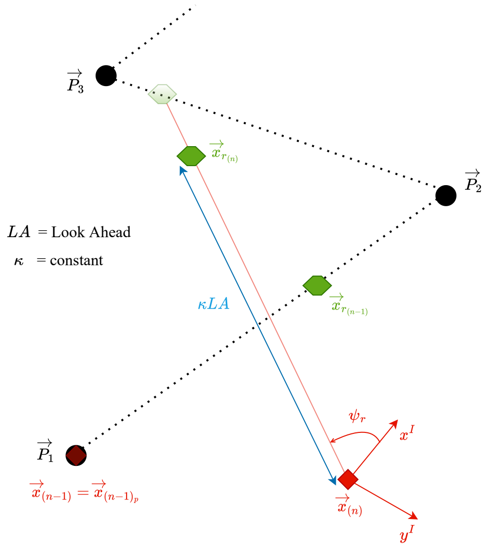
These vectors allow the computation of the closest distance between the waypoint line and the current vehicle position, also known as deviation error (ϵ), with Eq. (12). In contrast with the previous situation, now the new reference point $\overrightarrow{x}_{r_{(n)}}$ is located at a distance $LA + \epsilon \cdot ALA$ from $\overrightarrow{x}_{(n)_p}$ in order to accelerate the error correction. ALA is a user defined value called "Adaptive Look Ahead", which allows to tune the importance of the waypoint line vehicle deviation in the definition of the next reference location. This is shown in Fig. 28.

$$\epsilon = |\overrightarrow{AV} - \overrightarrow{AoG}| \tag{12}$$



$\overrightarrow{P}_3$
$\overrightarrow{x}_{r_{(n)}}$
$\epsilon \cdot ALA$
$\overrightarrow{P}_2$
LA
LA = Look Ahead
ALA = Adaptive Look Ahead
ϵ = error
$\overrightarrow{x}_{r_{(n-1)}}$
$\overrightarrow{x}_{(n)_p}$
$\overrightarrow{P}_1$
$\overrightarrow{x}_{(n-1)} = \overrightarrow{x}_{(n-1)_p}$
$\overrightarrow{x}_{(n)}$

Fig. 28 Guidance block approach at time step n: reference position definition.



$\overrightarrow{P}_3$
$\overrightarrow{x}_{r_{(n)}}$
$\overrightarrow{P}_2$
LA = Look Ahead
κ = constant
κLA
$\overrightarrow{x}_{r_{(n-1)}}$
ψ_r
x^I
$\overrightarrow{P}_1$
$\overrightarrow{x}_{(n-1)} = \overrightarrow{x}_{(n-1)_p}$
$\overrightarrow{x}_{(n)}$
y^I

Fig. 29 Guidance block approach at time step n: reference position correction.

As can be observed when comparing Fig. 26 and Fig. 28, the introduction of the adaptive look ahead can increase the error between the reference and actual position considerably. If the controller is not able to correct the deviation fast, it will lead to loss of control. In order to reduce the probability of emergence of this event, the authors have introduced a simple but effective modification in the AirSim guidance block. Once the reference position has been computed, it is checked whether the distance between the reference and current vehicle positions is higher than a constant (κ) times LA. If that is the case, then a new reference position is defined in the same direction as the old one but at a distance of κLA from the current position. The effect of this modification can be observed in Fig. 29. For the current research, it was found that a value of 1.5 for the constant was effective to reduce the emergence of the aforementioned undesired behaviour.

The second part of the guidance block is the computation of the yaw reference. It is desired that the drone points in the direction of the desired flight path, meaning that it should point towards the reference point found in the reference position definition, as shown in Fig. 29. Hence, the reference yaw is found with Eq. (13).

$$\psi_{r_{(n)}} = \arctan \frac{y_{r_{(n)}} - y_{(n)}}{x_{r_{(n)}} - x_{(n)}} \tag{13}$$

4. Sensor data collection, failure injection and flight termination

During the execution of the flight, sensor data is being collected. AirSim provides a Python API for obtaining information from the different sensors and the camera with simple functions that return the value registered by the sensor at the time of the function call. Unfortunately, the maximum frequency at which these functions can be called or record data is 26 Hz within the simple Blocks environment for a single drone, value far from the desired 500-1000 Hz for the IMU or 30-60 Hz frames per second for the camera. Additionally, this sampling rate changes during the flight, depending on the workload experienced by the different threads within the simulator in UE4 and the other computations carried out by the Python API. Within a single flight test, a difference of a factor of 2.3 has been recorded between the fastest and slowest sample rates.

In order to increase the sample rate of the IMU, its data is collected and stored within C++ before being sent back to the Python API. For that purpose, the IMU data collection has been coupled to the physics engine of the simulator; with every time step in the physics engine it is verified whether IMU information should be stored and, if that is the case, it is saved in a vector for later retrieval. Once the flight is concluded, the information regarding that flight is called from the Python API for storage and the C++ vector is cleaned for the next flight. The same process has been implemented for other signal-based sensors, namely the barometer, magnetometer and GPS. The user only has to choose the sensors to activate and their sampling rate.

Another main benefit of coupling the IMU data retrieval to the physics engine is that it is immune to simulation slow-downs due to time intervals with a high computational load (e.g. when rendering UE4 environment sections with a higher count of assets). If the simulator slows down, then the sensor data collection does it by the same amount thanks to its linkage to the physics engine. The main disadvantage of this approach is that the sample rate choices are limited to factors of the physics engine thread calling rate. With the nominal physics engine thread calling period of 0.003 seconds (f_p = 333.33 Hz), the sampling frequencies available for IMU data gathering are discrete and limited to values of f_p divided by integer values.

With respect to the video sampling rate, it is not possible to couple the image data storage to the physics engine because the AirSim image retrieval functions are part of another simulator thread. Therefore, in order to increase the number of frames stored per second (fps), the simulation clock is modified. When the clockspeed is modified, the physics engine sampling rate changes by the same factor. As a result, slowing down the simulation allows higher IMU sampling rates and a larger number of choices. For instance, using a clockspeed factor of 0.5 would allow an IMU sampling rate of up to 666.66 Hz. Therefore, the clockspeed factor is a hyper-parameter that has to be tuned by carrying a trade-off between the IMU and camera sampling rates.

At the same time as data is being gathered, the distance to the goal location is computed and it is assessed whether the drone has reached the distance along the planned trajectory at which the failure should be injected. Once that point is reached, the Python API calls the C++ method that introduces the chosen failure. In the case of actuator saturation, actuator lock or propeller fly-off, the damage coefficient is changed to the desired value. In the case of blade damage, the forces and moments that the lost blade sections would hypothetically generate are subtracted from those computed by the Bebop 2 gray-box aerodynamic model. Depending on whether the failure is abrupt or continuous, the mentioned parameters are changed at a single time step to the desired value or they are linearly changed.

Once the failure has been injected, the simulation pipeline starts to check whether any of following flight termination conditions has been reached: collision with the ground, collision with an obstacle, drone flies above a predefined altitude or timeout, meaning that a predefined number of seconds after the failure injection has been reached.

5. Flight & failure metadata logging and sensor data storage

Once the flight has been concluded, flight and failure metadata, such as the type, location and magnitude of the failure, are stored for the posterior labeling of the gathered data for the training and testing of FDD algorithms. The data recorded by the IMU at every time step of the flight are stored in a .csv file. All the camera frames are stored in the same directory as the IMU, each image with the name of the timestamp at which it was taken in order to preserve the temporal sequence information.

Finally, the client, sensors and failure factory are reset to their original values in order to repeat the complete data gathering pipeline shown in Fig. 2 for the next flight.

D. Debugging tool: signal scoping

In a similar fashion as when IMU data are gathered, any signal within the guidance, control and physics model can be stored in a vector within the C++ API during the flight in order to be plotted upon flight termination with a call from the Python API. The goal is to achieve a similar functionality as the signal scoping tool within Matlab to ease the debugging of these simulator components. The result of such implementation is a single user input in the Python API in which it must specify the signals it would like to plot. Additionally, it has the option of choosing which signals to plot together in a single figure for direct comparison.

The introduction of this tool accelerates the debugging process of the simulator and it will facilitate the smooth development and implementation of alternative forms of guidance and control, as well as different drone physics models, by future researchers. Figure 30 and Fig. 31 show two examples of scoped signals. The first shows the 3D trajectory that the drone followed when commanded to fly in a straight line in the x-direction. The second shows the commanded propeller rotation velocities for the same flight.

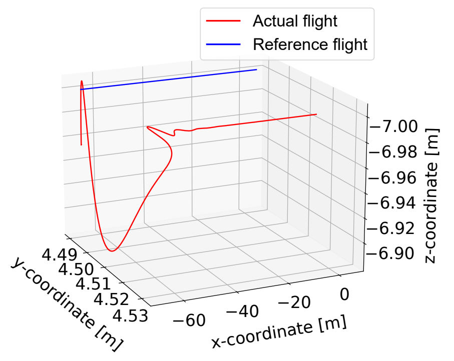


Fig. 30 3D actual and reference trajectories for single sample flight.

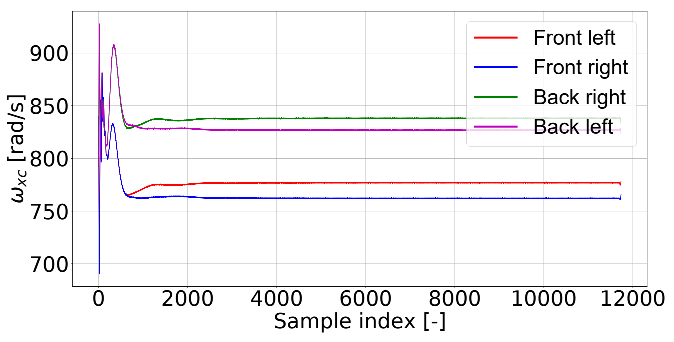


Fig. 31 Propeller commanded rotational velocity of each actuator in rad/s for single sample flight.

IV. Fault detection and diagnosis framework

The goal of this section is the development of an actuator FDD framework, which does not only alert the drone computer about the presence of an actuator failure, but it is able to point to the failed actuator and quantify the damage. For that purpose, the authors propose an architecture which fuses the information obtained from the Inertial Measurement Unit (IMU) and the camera on-board of the drone. Thanks to the simulator developed in section III, it is possible to use knowledge-based approaches previously impossible due to the lack of data.

The complete FDD architecture can be observed in Fig. 32. Early in the pipeline it consists of two separate paths for independently processing the camera and IMU information for the extraction of features. Then, both paths' features are concatenated and fed to a Long Short-Term Memory (LSTM) for data fusion, architecture with feedback connections which allows the ingestion of sequential data. Finally, the output is passed on to a dense Neural Network for classification with a number of neurons equal to the number of distinct classes. For example, in the case of failure detection there are only two classes, namely healthy or failure. Hence, the classifier would have only two neurons in the output layer.

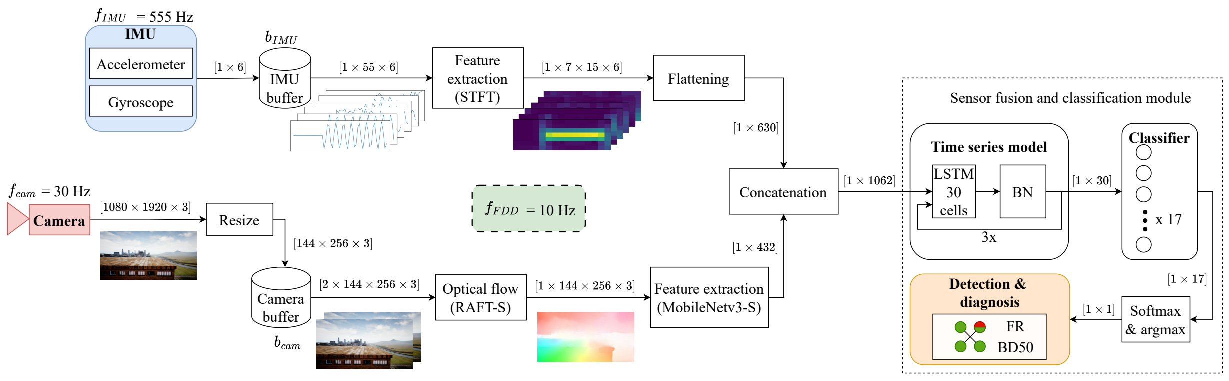


Fig. 32 The FDD pipeline consists of (i) an IMU time-frequency feature extractor in the form of a Short-Time Fourier Transform, (ii) the MobileNetV3-S as feature extractor from the camera optical flow computed with RAFT-S and (iii) a Long Short-Term Memory network followed by a single layer Neural Network as sensor fusion and classification module. The FDD framework is run at 10 Hz and the sampling rate of the IMU and camera are 555 Hz and 30 Hz, respectively.

One of the main challenges for any FDD architecture that aims at fusing multiple data sources with different sampling rates is the synchronisation of the information without discarding precious data. On-board of most drones, the IMU is able to produce samples at rates multiple times higher than the camera. A naive approach would be running the FDD at the same frequency as the camera and taking the last data point from the camera and the IMU at every time step; discarding all the IMU samples collected between camera shots. The developed FDD architecture can run at a commanded frequency different than both sensors on board, as long as it is equal or smaller than the slower sensor. This highlights the flexibility of the architecture, being able to adapt to different computation constraints.

Next, each of the architecture components will be explained in detail. Even though the framework could be applied to any aerial platform, the frequencies and tensor sizes flowing through the architecture correspond to the Bebop 2 drone used in the present research. First, subsection IV.A will dive into the different components required for the image feature extraction. Then, subsection IV.B will discuss how the IMU data is processed. Finally, subsection IV.C will show how the sensor features are translated to a detection and diagnosis prediction by treating the tasks as a classification problem.

A. Camera data processing

The inspiration for the introduction of the camera into the FDD pipeline stems from the observation that human beings are able to detect that they are falling thanks to their "natural time differentiated accelerometer" or vestibular system, an apparatus within the inner ear that provides information about changes in acceleration, as well as from their visual sensory system. When the vestibular system is saturated (e.g. rapidly rotating on an office chair) or the changes in acceleration are imperceptible (e.g. accumulating slow changes in aircraft attitude), the visual sensory system is still able to detect the subject's ego motion thanks to the relative movement of elements of the environment in its visual field. For instance, if a human subject sees a block moving to the right in a static environment, the subject understands that it is moving then to the left.

The two main factors affecting judgement of self-motion are the gradients and the pattern of optical flow which provide information about the relative velocity (amount) and direction of relative motion, respectively. Hence, the authors believe that knowledge about the magnitude and direction of the optical flow could enhance the diagnosis component of the FDD framework by implicitly quantifying the failure magnitude and identifying the failed actuator. For instance, if the front right clockwise rotating (from top view) propeller is lost, then it is expected that the drone will lose lift, tilt forward and rotate clockwise. In optical flow, this should translate to a vector field with an up-left direction. The stronger the gradient, the greater the failure magnitude.

There are two ways in which optical flow can be represented, namely sparse and dense optical flow [42]. The main difference is that the first computes the optical flow for a predetermined number of features of interest whereas the second computes it for the complete frame. Even though the sparse optical flow is less computationally expensive, it has two main problems. First, those features of interest may disappear or become hidden after a few frames, forcing the optical flow approach to select new features. Second, the algorithm may choose different features between frames as some become more salient than others throughout time. As a result, it is difficult to infer a potential actuator failure from a specific optical flow change pattern as it could be attributed to the tracking of different features over consecutive frames. Hence, dense optical flow was chosen.

In literature there are two main classes of dense optical flow approaches, namely traditional or classical energy-based and deep-learning based. In recent years, deep learning based approaches have been able to surpass the traditional counterparts in accuracy and lower inference times, allowing them to run in real time and becoming the de facto choice for computationally constrained devices and platforms [43, 44]. In most cases, the performance of optical flow approaches is compared using the Average End Point Error (AEPE) on the MPI-Sintel final dataset and the Fl-all in the KITTI2015 dataset [45]. The AEPE is the average Euclidean distance between the estimated and ground truth optical flow vectors, and the Fl-all is the percentage of flow outliers averaged over all pixels. The MPI-Sintel final dataset[†] [46] is a 564 frame animated movie synthetic dataset with realistic illuminations, reflections and rendering effects; whereas the KITTI2015 dataset[‡] [47, 48] is a 200 frame real-world dataset collected from a moving car.

Within this deep-learning based approaches there are three architectures that, according to the authors, stand out from the literature for their high accuracy and low inference time, while providing their code and trained model weights. With their trade-off metrics shown in Table 2, they are:

- CNNs for Optical Flow using Pyramid, Warping, and Cost Volume (PWC-NET) [49]. It was published in June 2018, one of the fastest methods in literature and the fastest from the selection; it is considered a milestone algorithm in the field [45].
- Recurrent All-Pairs Field Transforms for Optical Flow (RAFT) [50]. It was published in November 2020 and it shows the highest performance of the three considered approaches in the MPI-Sintel dataset with the highest reported inference time [50].
- Displacement-Invariant Matching Cost Learning for Accurate Optical Flow Estimation (DICL-Flow) [51]. It was published in December 2020 and it shows a reported runtime and performance between the PWC-NET and RAFT approaches.

[†]http://sintel.is.tue.mpg.de/

[‡]https://www.cvlibs.net/datasets/kitti/

Table 2 Performance and inference time comparison of dense optical flow approaches.

Method	Time	K-15 train		K-15 test	S-train (EPE)		S-test (EPE)	
	(s)	EPE	Fl-all	Fl-all	Clean	Final	Clean	Final
PWC-NET[§]	0.03	10.35	33.67%	-	2.55	3.93	-	-
RAFT[¶]	0.2	5.04	17.4%	-	1.43	2.71	-	-
DICL-Flow[‖]	0.08	8.70	23.6%	-	1.94	3.77	-	-
RAFT-S	-	7.51	26.9%	-	2.21	3.35	-	-
Farneback [52]	1	10.50	-	53.09%	-	8.9	-	-

All of these deep-learning based approaches were trained with data from the FlyingChairs [53] and FlyingThings3D [54] datasets. Next to them, Table 2 contains two more entries: a classical approach for comparison, namely Gunnar Farneback's algorithm [55] developed in 2003, and a small pre-trained RAFT model (RAFT-S) implemented within the Torchvision library. In contrast with the original RAFT model, it contains five times less parameters but it maintains superior performance in the MPI-Sintel final train dataset when compared to the PWC-NET and DICL-Flow models.

Unfortunately, it is not clear whether the reported inference times in literature were obtained from systems with similar compute specifications. To compare these approaches, they were executed on three datasets collected with UUFOSim at different image resolutions, resulting in the inference times shown in Table 3. Each time value is the average that each algorithm took to predict the optical flow for 250 frames on a laptop with a 6 core Intel Core i7-9750H CPU, 16 GB of RAM DDR4 and an NVIDIA Quadro P2000 with 5 GB of GDDR5 memory. As can be seen, even though DICL-Flow is the intermediate option from literature, it presents the worst inference time for all resolutions.

Table 3 Inference time of dense optical flow approaches on the collected UE4 dataset at different resolutions.

Methods	256×144	512×288	1024×576
	(s)	(s)	(s)
PWC-NET	0.073	0.143	0.423
RAFT	0.17	0.17	0.36
DICL-Flow	0.274	0.296	0.617
RAFT-S	0.06	0.10	0.35
Farneback	0.008	0.042	0.177

Figure 33 allows for a visual comparison of the approaches' optical flow quality with a frame from the 1024×576 dataset. As can be seen in Fig. 33b, even though PWC-NET has the lowest run time among the deep learning approaches, its optical flow prediction is very noisy without any recognisable features, indicating a poor cross-dataset generalization. Furthermore, from Fig. 33f it can be seen that Farneback does not perceive slight movements. Most of the pixels are black, leading to the loss of potential features (pixels) that could serve as rich sources of information further down the FDD pipeline. Besides that, a strong flickering behaviour has been observed in Farneback's optical flow over multiple frames, which hints to unreliable predictions.

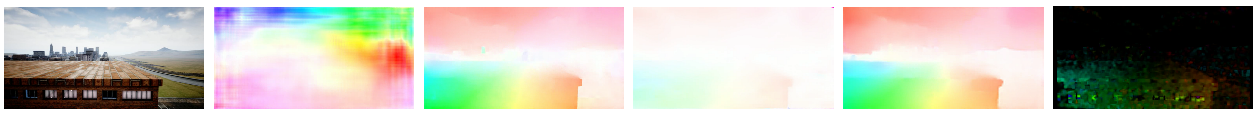

(a) Original **(b) PWC-NET** **(c) RAFT** **(d) DICL-Flow** **(e) RAFT-S** **(f) Farneback**

Fig. 33 Dense optical flow visual quality comparison.

[§]https://github.com/philferriere/tfoptflow
[¶]https://github.com/princeton-vl/RAFT
[‖]https://github.com/jytime/DICL-Flow

Given the high inference time of DICL-Flow with the collected dataset and the low visual quality of PWC-NET and Farneback, the two remaining options for optical flow computation are RAFT and RAFT-S. As both show similar visual quality and RAFT-S has a run time three times lower than its larger version for the lowest resolution, RAFT-S is chosen as the optical flow module of the FDD pipeline.

Returning the attention to Fig. 32, the bottom information path shows the camera data processing. In the case of the Bebop 2 drone, the camera captures images at 1080p, meaning frames of 1080 pixels in height and 1920 in width with three RGB channels [1080×1920×3], and they are resized to a tensor of dimensions [144×256×3] before being stored in the camera buffer (b_{cam}). Then, at every time step at which the FDD framework is executed, the b_{cam} contains $f_{\text{cam}}/f_{\text{FDD}} + 1$ samples, and the first and last entry of the buffer are passed on to the optical flow model. Here, f_{cam} stands for the fps rate at which the drone collects image data and f_{FDD} is the frequency at which the FDD pipeline is executed on board of the drone. Next, the buffer is emptied except for the last stored image, which remains in memory for the next FDD time step. This ensures the temporal coherence of the optical flow over multiple FDD calls.

Once RAFT-S computes the optical flow, the output tensor is fed to a feature extractor. For this part of the pipeline, the authors opted for transfer learning instead of developing their own. The model of choice was the backbone of MobileNetV3-Small [56] with frozen weights pre-trained on the ImageNet dataset [57] because it has the lowest inference time among all keras pre-trained models** at the time of writing. A depth multiplier (alpha) of 0.75 was set in order to proportionally decrease the number of filters in each layer, achieving a reduction in the number of parameters from 2.9 to 2.4 million (3 ms of inference time). Finally, the last layer of MobileNetV3-Small is set to be a global average pooling layer which collapses the width and height of the output tensor to a single feature, resulting in a 1D tensor of 432 features.

B. IMU data processing

From the IMU, the FDD algorithm receives six 1D data streams, namely the linear acceleration and the angular velocity in the x, y and z directions. Two key signal features that contribute to the detection and classification of these failures are the evolution of their bias through time and the amplitude of their oscillations; the latter especially in the case of blade damage, as highlighted in [34]. Information about both features can be encapsulated in their Short-Time Fourier Transform (STFT), creating compact time-frequency maps or spectrograms and removing potential sensor noise. To illustrate this, Fig. 34 shows the accelerometer signal in the x-direction and its spectogram for a random flight within the dataset which experienced a blade damage failure of 0.8, 6.83 seconds after the start (as highlighted by the red dashed vertical line). As can be seen, failure can easily be detected by the sudden appearance of signal content at high frequencies, in this case between 173 and 186 Hz.

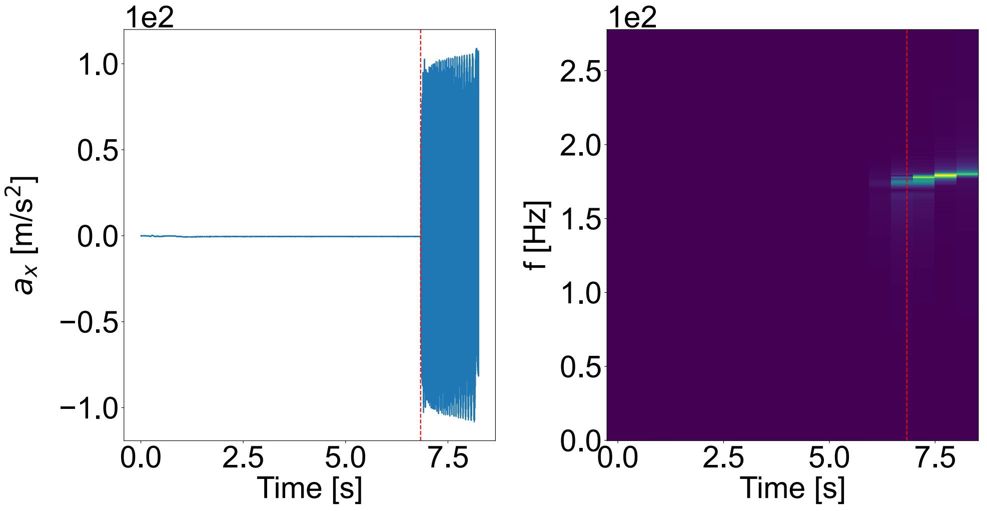


Fig. 34 UAV x-axis acceleration and its spectrogram. The dashed vertical line denotes the time of failure.

**https://keras.io/api/applications/

From the IMU information path shown in the upper half of Fig. 32, the incoming data from the accelerometer and the gyroscope is stored in a buffer (b_{IMU}). Once the FDD module is called, the buffer is emptied and its data is used for computing the STFT. This form of frequency analysis is a windowed approach which divides the time signal into small equally sized segments and applies an independent Fourier transform to each one of them. Hence, there is a trade-off between the time and frequency resolutions; the wider the window the higher the frequency resolution at the expense of the time resolution. Since the STFT is applied to small sample sizes of n_{seg}=$\lfloor f_{\text{IMU}}/f_{\text{FDD}}\rfloor$ at a time, a window size of n_{win}=$\lfloor n_{\text{seg}}/4\rfloor$ is chosen with $o = \lfloor 3/4 n_{\text{win}}\rfloor$ samples of overlap between windows, i.e. a stride of s=$\lceil 1/4 n_{\text{win}}\rceil$. The sample vector is padded such that the time resolution or the number of steps in which the time axis of the spectrogram is divided is t_{re}=$\lceil n_s/(n_{\text{win}}$-$o)\rceil$+1. As can be seen, as n_{win} increases, t_{res} decreases. The opposite is observed in the frequency resolution f_{res}=$\lfloor n_{\text{win}}/2\rfloor + 1$.

For the present research, f_{IMU} of the collected dataset and f_{FDD} approximately equal 555 Hz and 10 Hz, respectively. Hence, 55 samples are fed to the STFT at every FDD time step, which outputs a tensor of dimensions [7×15×6]. This means a frequency resolution of seven and a temporal resolution of 15. Figure 35 and Fig. 36 show the IMU signals and their STFTs for the same flight as in Fig. 34, using a time segment of 0.1 s (f_{FDD}=10 Hz) starting at 6.78 s in order to include the transition from a healthy to a failure state. Again, it can still be clearly observed when the blade damage has taken place for failure detection. Finally, the STFT output tensor is flattened to a single dimensional tensor of 630 features that will be fused with those coming from the camera data processing path of the pipeline.

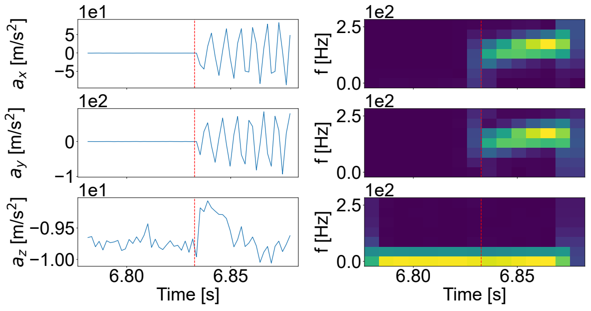


Fig. 35 Sample flight accelerometer signals and spectrograms for a 0.1 s time interval starting at 6.78 s.

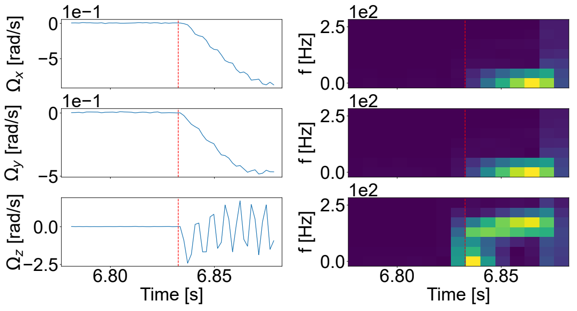


Fig. 36 Sample flight gyroscope signals and spectrograms for a 0.1 s time interval starting at 6.78 s.

C. Sensor fusion and classification module

As can be seen in Fig. 32, the features from the camera and the IMU are concatenated into a single vector of 1062 features and fed to a sequence-to-sequence LSTM model, which allows the FDD framework to take decisions based on current and previous data at every time step. LSTM cells have an internal state that stores information from an arbitrary number of previous inputs which, in conjunction with the current input, is used to extract sequential relationships to generate an output. For the present research, the time series model consists of a simple stack of three LSTM layers of 30 cells, each followed by a Batch Normalization (BN) layer; transformation that maintains the mean and standard deviation of its input batch close to zero and one, respectively. At every FDD time step, an input vector of 1062 features is fed into the network which outputs a tensor of 30 features.

The last stage of the FDD pipeline is the classifier that will simultaneously perform the tasks of failure detection, failure magnitude quantification and failed propeller identification. The problem is simplified by considering each potential drone state, namely each failure mode and the healthy state, as a class. As an example, if abrupt actuator saturation and abrupt propeller fly-off are considered as the only modes of failure, then the classification layer would have to discern among nine classes, namely two failure classes per propeller and one for the healthy state. To perform this classification task, a single layer dense neural network layer (NN) is used with the number of neurons equal to the number of classes, followed by the softmax activation function in order to generate a multinomial probability distribution; the model outputs the probability it believes the input belongs to each class. The goal is that the highest probability is attributed to the correct failure or healthy drone state at each time during the flight.

Both the LSTM model and the classifier are the only two trainable components of the FDD pipeline, as the RAFT-S and MobileNetV3-S weights are frozen. For their training, the sparse categorical cross-entropy loss function and adam optimizer are used, both extensively exploited in literature for multi-class classification.

V. Results

A. UUFOSim dataset

The right clockspeed is a function of the image resolution; the larger the image, the lower the camera sampling rate at the same clockspeed. Therefore, a trade-off needs to be performed between sampling rate accuracy and simulation speed for the chosen image resolution of 256×144 (width×height). Figures 37 and 38 show 20 simulations at different clockspeeds and their camera and IMU sampling rates. It can be observed that a clockspeed of 0.6 has a large spread of camera sampling rates between flights and the clockspeeds of 0.4 and 0.5 have IMU sampling rates far below the desired 512 Hz. Since the remaining clockspeeds show similar performance, 0.3 was selected for being the fastest.

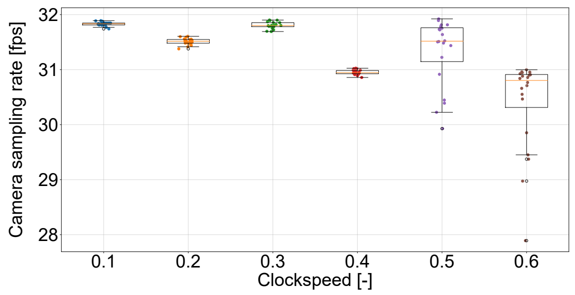


Fig. 37 Boxplot of the camera sampling rate for different clockspeeds with an image resolution of 256×144 pixels (width×height).

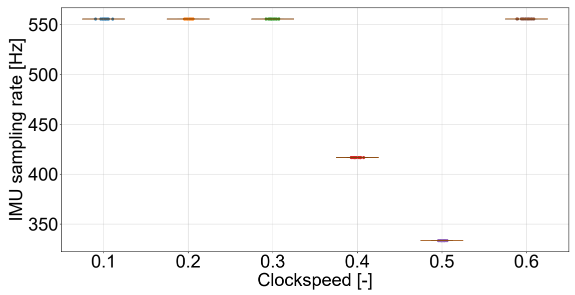


Fig. 38 Boxplot of the IMU sampling rate for different clockspeeds with an image resolution of 256×144 pixels (width×height).

The IMU sampling rate samples are almost constant at the same clockspeed because the data gathering of this sensor has been coupled with the simulator's physics model, as dicussed in subsection III.C. In contrast, the camera sample rates are much more dispersed, especially the higher the clockspeed. For the same simulation time and slower clockspeed, the simulation checks the thread that receives the calls from the Python API more frequently. Hence, the frequency at which camera images can be called is higher, reducing the impact of simulation slow downs and, hence, the camera sampling rate dispersion.

5,000 flights were flown with a clockspeed of 0.3 and image resolution of 256×144. To verify that the camera and IMU sampling rate predictions estimated with 20 flights were accurate, the same box plot was created with the flown 5,000 flights. The results are shown in Fig. 39 and Fig. 40: the camera runs at 31.81 fps and the IMU has a sampling rate of 555.59 Hz.

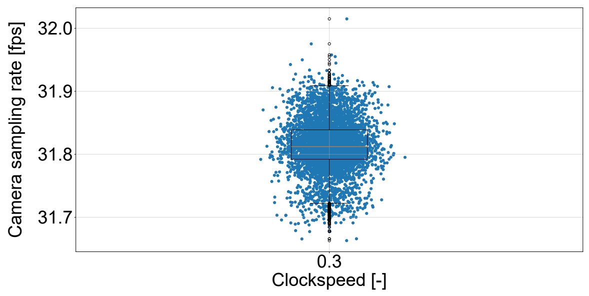


Fig. 39 Boxplot of the camera sampling rate for 5,000 flights with a clockspeed of 0.3 and an image resolution of 256×144 pixels (width×height).

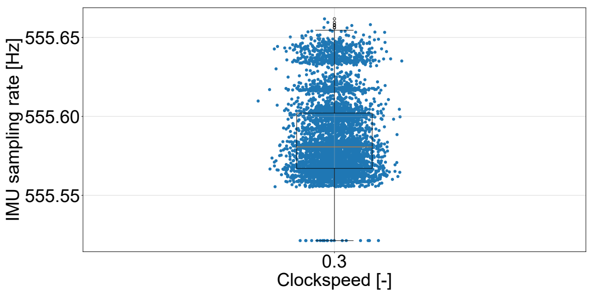


Fig. 40 Boxplot of the IMU sampling rate for 5,000 flights with a clockspeed of 0.3 and an image resolution of 256×144 pixels (width×height).

The simulation pipeline discussed in section III was run in a Windows OS PC with a 20 core Intel Xeon W-2255 CPU, 32 GB of RAM DDR4 and an NVIDIA RTX A4000 GPU with 16 GB of GDDR6 memory. The 5,000-flight dataset was collected in 61.67 hours and has a memory footprint of 239 GB. Only blade damage failures of 20%, 40%, 60% and 80% were simulated since those are the failure modes that will be used to train and test the FDD pipeline. A sample of frames from a single flight separated by 35 frames from each other can be observed in Fig. 41.

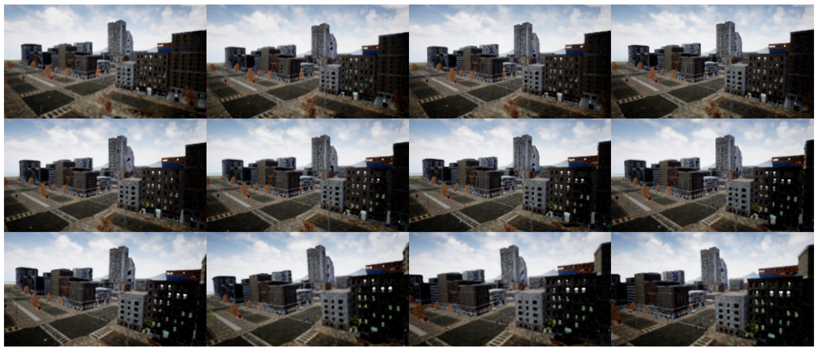


Fig. 41 Camera captured frames during single flight read from top to bottom and from left to right (only shown one every 35 frames).

B. Fault detection and diagnosis framework

To demonstrate the potential of the proposed FDD framework, only four modes of discrete failure were considered per propeller, namely 20%, 40%, 60% and 80% single abrupt blade damage. As a result, there are 17 classes among which the FDD pipeline should discern. For this purpose, the dataset was split into 70% training, 20% validation and 10% testing. Each flight of this dataset has a variable duration between 6 and 16.9 seconds with an average length of 11.6 seconds. The first second of every flight is ignored in order to avoid the acceleration transient after the flight has started. From the remaining flight time, single 5.5-second data snippets are used per flight in order to batch train and evaluate the pipeline with equal length data sequences without padding. Flights of length shorter than 6.5 seconds only constitute 0.72% of the total dataset and were eliminated. Besides that, flights that were not properly recorded in UE4 — the drone does not take off or the sensor data is not recorded at the correct rate — were also removed. At the end, the training dataset consisted of 3,468 5.5-second flights.

Table 4 shows the results for the pipeline presented in section IV in terms of inference time and test accuracy. The runtime was obtained from the same compute setup that was used to generate Table 3. Furthermore, three different types of test accuracy are considered, namely general, detection and diagnosis. The first refers to the accuracy outputted by the model. The second is obtained by lumping the failure classes 2 to 17 into a single class and computing the resulting accuracy. This means that a prediction of a data point whose ground truth is a failure class is deemed correct as long as any class from 2 to 17 is chosen, independently of whether the right class is predicted. The third is estimated by ignoring the data points whose ground truth is class 1 (the healthy state) and recomputing the accuracy of correctly classifying the failure among the remaining classes.

Table 4 FDD accuracy and inference time results. With a total of 17 classes, four discrete and abrupt failure modes were simulated for the Bebop 2 UAV per propeller, namely 20%, 40%, 60% and 80% single blade damage.

Data processing	Data fusion model	Inference time	General accuracy	Detection accuracy	Diagnosis accuracy
		(ms)	(%)	(%)	(%)
IMU	LSTM (l3-c30)+BN	**88.20**	80.70	**99.98**	50.52
CAM	LSTM (l3-c30)+BN	240.01	95.93	98.53	89.94
CAM+IMU	LSTM (l3-c30)+BN	250.47	**99.55**	**99.98**	**98.86**
	Dense (l3-c128)+BN	241.77	93.56	**99.98**	83.49

Additionally, the same metrics of modified versions of the pipeline are also presented in order to demonstrate the added value of each of its components. The "Data processing" column stands for the active branches of the network, where CAM and IMU are networks with only the camera or the IMU paths active. LSTM (l3-c30)+BN is the data fusion architecture explained in subsection IV.C, whereas Dense(l3-c128)+BN is an alternative approach where the temporal relationships of the data are ignored by substituting the LSTM network with a three-layer dense NN with 128 neurons per layer.

Even though the IMU-only network feeds the sequential model with 46% more features than the camera-only network, as can be seen in Fig. 32, the latter shows an overwhelming superiority in the diagnosis of the failures with a 39.42% difference in accuracy. The reason behind that difference can be observed in Fig. 42; its confusion matrix of the predicted and true failure modes. The IMU-only network systematically confuses the front right (FR) and front left propellers (FL), as well as the back right (BR) and back left (BL). However, despite being unable to identify the failed propeller, it is able to infer the correct degree of damage. This is shown by the parallel diagonals three cells apart.

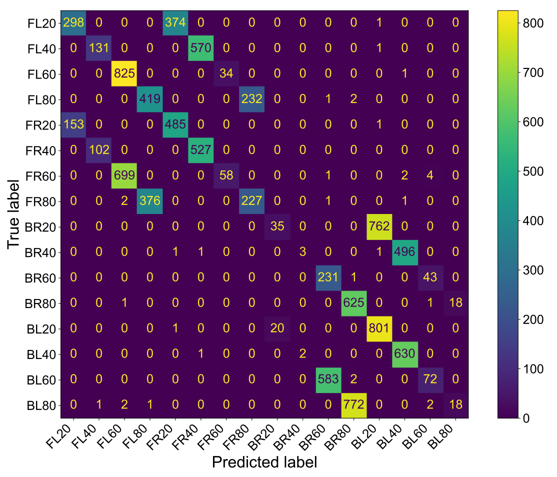


Fig. 42 IMU-only LSTM model confusion matrix of the failure modes.

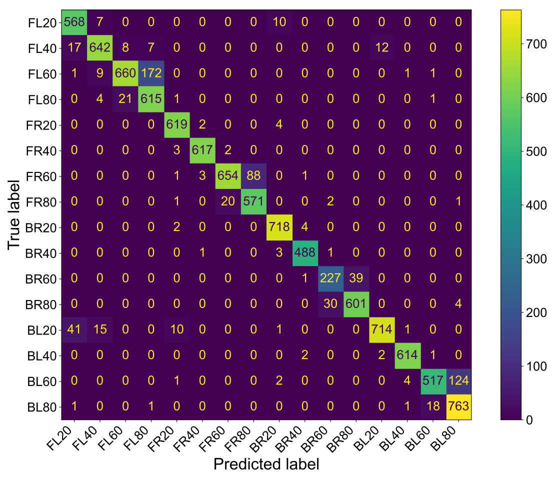


Fig. 43 Camera-only LSTM model confusion matrix of the failure modes.

In contrast, Fig. 43 shows that the camera-only network is able to correctly identify the failed actuator but fails to always accurately quantify the damage. Most of the incorrectly labelled predictions are one degree of damage higher or lower than the true label, but within the same actuator.

Both observations demonstrate the complementary nature of the camera and IMU sensors, which combined lead to the highest measured diagnosis accuracy of 98.86%. Figure 44 shows the IMU+CAM network confusion matrix with the main diagonal filled with -1's in order to visually highlight error patterns. From the multiple coloured parallel lines to the main diagonal, it can be inferred that the largest source of error originates from failing to correctly identify the damaged actuator. However, it is not constrained to the front and back propeller combinations, as it was the case for the IMU-only model.

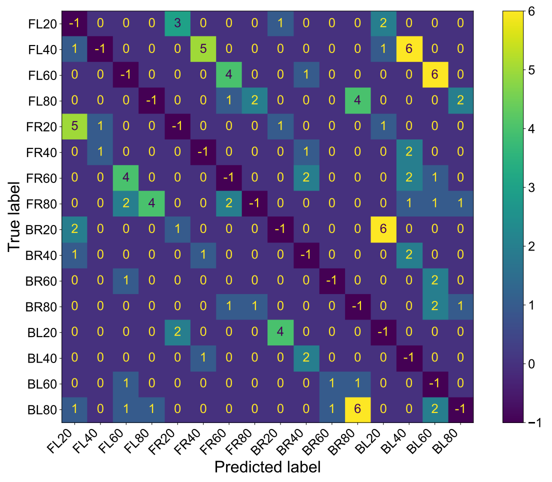


Fig. 44 IMU+CAM LSTM model confusion matrix of the failure modes with -1's in main diagonal.

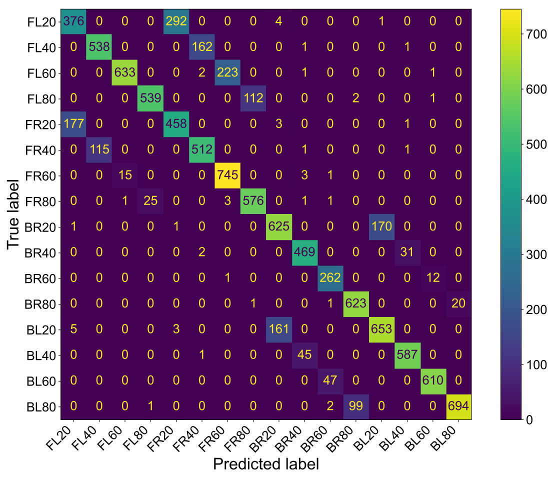


Fig. 45 IMU+CAM Dense model confusion matrix of the failure modes.

Furthermore, the difference in diagnosis accuracy between the CAM+IMU LSTM and Dense models highlights the importance of including the data temporal relationships in the FDD framework. However, it can also be seen that this information does not play a role when detecting the presence of a failure.

From the confusion matrix of the CAM+IMU Dense model shown in Fig. 45, the misinterpretation among the failures in the front and back propeller groups can again be seen. From this, it can be deduced that it is not the optical flow but its change that allows their decoupling. If the optical flow and the LSTM can each be considered a first derivative in time, then it is the second derivative of the camera's visual information that carries the differentiation factor between left and right actuators.

Finally, despite the success of the combined sensor approach, it has an inference time 2.84 times higher than the IMU-only approach: 8.03 ms (3.20%) for STFT, 72.30 ms (28.77%) for RAFT-S, 90.53 ms (36.03%) for MobileNetV3-S, and 80.41 ms (32.00%) for the LSTM+BN and classifier model. Further work has to be done in reducing the compute required by the camera path of the model by, for instance, developing tailored optical flow and feature extraction models. Additionally, an ablation study has to be performed on the hyper-parameters of the LSTM network.

VI. Conclusion

This paper proposes a novel UAV actuator FDD framework that fuses for the first time camera and IMU data online with an LSTM network. The framework pre-processes the camera information by first computing its optical flow with the RAFT-S model and then extracting features with the backbone of the MobileNetV3-S model. Both are off the shelf pre-trained efficient SOTA deep neural networks. STFT is applied on the IMU signals in order to obtain time-frequency features in the form of flattened spectrograms.

Additionally, a high-fidelity photo-realistic UAV simulator built in Unreal Engine 4 on AirSim, called UUFOSim, was presented. It is the first simulator that allows the collection of multi-sensor UAV flight data with mid-flight actuator failures injected programmatically. To the authors knowledge, UUFOSim generated the first synthetic dataset in literature for the training and testing of UAV actuator FDD approaches. Such data is of superior quality when compared to alternative simulation environments, e.g. Gazebo, allowing the development of applications with a reduced reality gap.

To demonstrate the potential of the FDD framework, UUFOSim was used to generate a dataset of 5,000 flights flown in a urban environment by a Bebop 2 platform with four options of blade damage per propeller injected during flight. The drone platform was simulated using a gray-box aerodynamic model [11] complemented with a Blade Element Theory blade damage model [34]; both obtained from literature.

The IMU-only model has shown to fail to perform damage actuator identification by systematically confusing the left and right propellers, whereas the camera-only model errors are from failure magnitude quantification. When combined, they fill the gap left by each other's weaknesses. Results show the complementary nature of the IMU and camera for FDD, achieving an accuracy of 99.98% for detection and 98.86% for diagnosis on the test dataset.

The need for a model which considers the temporal relationships in sequential data was demonstrated by substituting the LSTM layers with dense NN that do not share information about previous inputs. This modified FDD model led to a decrease in diagnosis accuracy by 15.37 percentage points without any gain in inference time.

Despite the high accuracy of the proposed vision-based FDD framework, it has an inference time of 250 ms, 2.84 times higher than the IMU-only alternative model. This observation calls for further work on the reduction of the computations required to process the camera data by developing tailored optical flow and feature extraction models for the task; these components currently account for more than 64% of the inference time. Optical flow ground truth images can be retrieved from UUFOSim for the training and testing of an in-house optical flow model. MobileNetV3-S could be further reduced in size by progressively removing the last layers and unfreezing its weights for fine-tuning. Alternatively, it should be investigated whether the current camera pipeline could be substituted by a sparse optical flow approach (e.g. Lucas-Kanade [58]) followed by two histograms, one for the magnitude and another for the direction of the sparse optical flow vectors. The number of the bucket with the highest count for each histogram would be fed to the sensor fusion. Since this work has shown that the main contribution of the camera is the identification of the failed actuator, it may be the case that only the vector direction histogram would be necessary. Moreover, an ablation study should be performed on the hyper-parameters of the LSTM network, which could lead to a reduction in layers and/or cells. The authors also expect the rise of compute power available by the time UAVs and UAM concepts are introduced in urban environments.

Future work includes the study of a probabilistic classifier, such as a Bayesian NN, in order to provide a degree of confidence besides a prediction, as well as improving the explainability of the black-box model. The potential of other architectures that ingest sequential (image) data, such as Convolutional LSTMs and lightweight attention-based machine learning approaches, should also be considered. Another alley of investigation is the substitution of MobileNetV3-S by an image Fourier Transform as a more efficient feature extractor. Furthermore, atmospheric turbulence models should be implemented within the simulator in order to assess the robustness of the FDD approach to external disturbances; they could induce a similar initial UAV motion as an actuator failure. Additionally, a hybrid dataset could be built which combines large quantities of synthetic UUFOSim data with a smaller real world dataset in order to reduce the reality gap. Data from multiple drones could be collected in order to make the FDD framework platform agnostic. Finally, the proposed framework should be implemented on a real Bebop 2 platform to validate the results.

To conclude, the proposed framework demonstrates the potential of including the UAV on-board camera for online failure detection and diagnosis. The authors hope that UUFOSim will help the research community to build benchmarks that will assist in the tracking of the future progress of UAV FDD, as well as other tasks that aim at making future drones more resilient to failures.

References

[1] Aurambout, J.-P., Gkoumas, K., and Ciuffo, B., "Last Mile Delivery by Drones: An Estimation of Viable Market Potential and Access to Citizens Across European Cities," *European Transport Research Review*, Vol. 11, 2019. https://doi.org/10.1186/s12544-019-0368-2.

[2] Choudhury, S., Solovey, K., Kochenderfer, M. J., and Pavone, M., "Efficient Large-Scale Multi-Drone Delivery using Transit Networks," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4543–4550. https://doi.org/10.1613/jair.1.12450.

[3] Thipphavong, D. P., Apaza, R., Barmore, B., Battiste, V., Burian, B., Dao, Q., Feary, M., Go, S., Goodrich, K. H., Homola, J., Idris, H. R., Kopardekar, P. H., Lachter, J. B., Neogi, N. A., Ng, H. K., Oseguera-Lohr, R. M., Patterson, M. D., and Verma, S. A., "Urban Air Mobility Airspace Integration Concepts and Considerations," *2018 Aviation Technology, Integration, and Operations Conference*, Atlanta, GA, 2018. https://doi.org/10.2514/6.2018-3676.

[4] Khan, H., Kushwah, K. K., Singh, S., Urkude, H., Maurya, M. R., and Sadasivuni, K. K., "Smart Technologies Driven Approaches to Tackle COVID-19 Pandemic: A Review," *3 Biotech*, Vol. 11, No. 2, 2021. https://doi.org/10.1007/s13205-020-02581-y.

[5] Lappas, V., Zoumponos, G., Kostopoulos, V., Shin, H., Tsourdos, A., Tantarini, M., Shmoko, D., Munoz, J., Amoratis, N., Maragkakis, A., Machairas, T., and Trifas, A., "EuroDRONE, a European UTM Testbed for U-Space," *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 1766–1774. https://doi.org/10.1109/icuas48674.2020.9214020.

[6] Mohammed, F., Idries, A., Mohamed, N., Al-Jaroodi, J., and Jawhar, I., "UAVs for Smart Cities: Opportunities and Challenges," *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 267–273. https://doi.org/10.1109/icuas.2014.6842265.

[7] Sun, S., Wang, X., Chu, Q., and De Visser, C., "Incremental Nonlinear Fault-Tolerant Control of a Quadrotor With Complete Loss of Two Opposing Rotors," *IEEE Transactions on Robotics*, Vol. PP, 2020, pp. 1–15. https://doi.org/10.1109/tro.2020.3010626.

[8] Mueller, M. W., Lupashin, S., D'andrea, R., and Waibel, M., "Controlled Flight of a Multicopter Experiencing a Failure Affecting an Effector," , 08 2020. URL https://patents.google.com/patent/EP3007973A1.

[9] Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M., "Autonomous Obstacle Avoidance and Manoeuvring on a Vision-Guided MAV Using On-Board Processing," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 2472–2477. https://doi.org/10.1109/icra.2011.5980095.

[10] Sun, S., Baert, M., Schijndel, B., and De Visser, C., "Upset Recovery Control for Quadrotors Subjected to a Complete Rotor Failure from Large Initial Disturbances," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 4273–4279. https://doi.org/10.1109/icra40945.2020.9197239.

[11] Sun, S., and de Visser, C., "Aerodynamic Model Identification of a Quadrotor Subjected to Rotor Failures in the High-Speed Flight Regime," *IEEE Robotics and Automation Letters*, Vol. 4, No. 4, 2019, pp. 3868–3875. https://doi.org/10.1109/lra.2019.2928758.

[12] Jiang, Y., Zhiyao, Z., Haoxiang, L., and Quan, Q., "Fault Detection and Identification for Quadrotor Based on Airframe Vibration Signals: A Data-Driven Method," *2015 34th Chinese Control Conference (CCC)*, 2015, pp. 6356–6361. https://doi.org/10.1109/chicc.2015.7260639.

[13] Chen, Z., Chen, W., Liu, X., and Song, C., "Fault-Tolerant Optical Flow Sensor/SINS Integrated Navigation Scheme for MAV in a GPS-Denied Environment," *J. Sensors*, Vol. 2018, 2018, pp. 1–17. https://doi.org/10.1155/2018/9678505.

[14] Iannace, G., Ciaburro, G., and Trematerra, A., "Fault Diagnosis for UAV Blades Using Artificial Neural Network," *Robotics*, Vol. 8, 2019, p. 59. https://doi.org/10.3390/robotics8030059.

[15] García, S., López, M. E., Barea, R., Bergasa, L. M., Gómez, A., and Molinos, E. J., "Indoor SLAM for Micro Aerial Vehicles Control Using Monocular Camera and Sensor Fusion," *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2016, pp. 205–210. https://doi.org/10.1109/icarsc.2016.46.

[16] Scaramuzza, D., and Zhang, Z., "Visual-Inertial Odometry of Aerial Robots," *Encyclopedia of Robotics*, 2020.

[17] Chen, K., "Recurrent Neural Networks for Fault Detection : An Exploratory Study on a Dataset about Air Compressor Failures of Heavy Duty Trucks," Master's thesis, Halmstad University, School of Information Technology, 2018.

[18] Zhao, H., Sun, S., and Jin, B., "Sequential Fault Diagnosis Based on LSTM Neural Network," *IEEE Access*, Vol. 6, 2018, pp. 12929–12939. https://doi.org/10.1109/access.2018.2794765.

[19] Zhu, P., Wen, L., Du, D., Bian, X., Fan, H., Hu, Q., and Ling, H., "Detection and Tracking Meet Drones Challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021, pp. 1–1. https://doi.org/10.1109/tpami.2021.3119563.

[20] Kouris, A., and Bouganis, C., "Learning to Fly by MySelf: A Self-Supervised CNN-based Approach for Autonomous Navigation," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5216–5223. https://doi.org/10.1109/iros.2018.8594204.

[21] Delmerico, J., Cieslewski, T., Rebecq, H., Faessler, M., and Scaramuzza, D., "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset," *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019, pp. 6713–6719. https://doi.org/10.1109/icra.2019.8793887.

[22] Majdik, A., Till, C., and Scaramuzza, D., "The Zurich Urban Micro Aerial Vehicle Dataset," *The International Journal of Robotics Research*, Vol. 36, 2017, p. 027836491770223. https://doi.org/10.1177/0278364917702237.

[23] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V., "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 2018, pp. 4243–4250. https://doi.org/10.1109/icra.2018.8460875.

[24] Furrer, F., Burri, M., Achtelik, M., and Siegwart, R., *RotorS – A Modular Gazebo MAV Simulator Framework*, Springer International Publishing, 2016, Chap. 7, pp. 595–625. https://doi.org/10.1007/978-3-319-26054-9_23.

[25] Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., and von Stryk, O., "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," *RoboCup 2013: Robot World Cup XVII*, edited by S. Behnke, M. Veloso, A. Visser, and R. Xiong, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 624–631. https://doi.org/10.1007/978-3-662-44468-9_58.

[26] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S., "Modular Open Robots Simulation Engine: MORSE," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 46 – 51. https://doi.org/10.1109/icra.2011.5980252.

[27] Guerra, W., Tal, E., Murali, V., Ryou, G., and Karaman, S., "FlightGoggles: Photorealistic Sensor Simulation for Perception-Driven Robotics Using Photogrammetry and Virtual Reality," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6941–6948. https://doi.org/10.1109/iros40897.2019.8968116.

[28] Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D., "Flightmare: A Flexible Quadrotor Simulator," *Conference on Robot Learning*, PMLR, 2020, pp. 1147–1157. https://doi.org/10.5167/uzh-193792.

[29] Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Soo Kim, T., Wang, Y., and Yuille, A., "UnrealCV: Virtual Worlds for Computer Vision," *ACM Multimedia Open Source Software Competition*, 2017, p. 1221–1224. https://doi.org/10.1145/3123266.3129396.

[30] Müller, M., Casser, V., Lahoud, J., Smith, N., and Ghanem, B., "Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications," *International Journal of Computer Vision*, Vol. 126, No. 9, 2018, p. 902–919. https://doi.org/10.1007/s11263-018-1073-7.

[31] Shah, S., Dey, D., Lovett, C., and Kapoor, A., "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *Field and Service Robotics*, Springer International Publishing, Zürich, Switzerland, 2018, pp. 621–635. https://doi.org/10.1007/978-3-319-67361-5_40.

[32] Madaan, R., Gyde, N., Vemprala, S., Brown, M., Nagami, K., Taubner, T., Cristofalo, E., Scaramuzza, D., Schwager, M., and Kapoor, A., "AirSim Drone Racing Lab," *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, Proceedings of Machine Learning Research, Vol. 123, PMLR, Vancouver, Canada, 2020, pp. 177–191.

[33] Sun, S., de Visser, C. C., and Chu, Q., "Quadrotor Gray-Box Model Identification from High-Speed Flight Data," *Journal of Aircraft*, Vol. 56, No. 2, 2019, pp. 645–661. https://doi.org/10.2514/1.c035135.

[34] de Alvear Cárdenas, J. I., and de Visser, C., "Blade Element Theory Model for UAV Blade Damage Simulation," , 12 2022. Unpublished.

[35] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixão, T. M., Mutz, F., de Paula Veronese, L., Oliveira-Santos, T., and De Souza, A. F., "Self-driving cars: A survey," *Expert Systems with Applications*, Vol. 165, 2021, p. 113816. https://doi.org/10.1016/j.eswa.2020.113816.

[36] Godoy, J., Jiménez, V., Artuñedo, A., and Villagra, J., "A Grid-Based Framework for Collective Perception in Autonomous Vehicles," *Sensors (Basel, Switzerland)*, Vol. 21, No. 3, 2021. https://doi.org/10.3390/s21030744.

[37] Carloni, R., Lippiello, V., D'Auria, M., Fumagalli, M., Mersha, A., Stramigioli, S., and Siciliano, B., "Robot Vision: Obstacle-Avoidance Techniques for Unmanned Aerial Vehicles," *Robotics & Automation Magazine, IEEE*, Vol. 20, 2013, pp. 22–31. https://doi.org/10.1109/mra.2013.2283632.

[38] Krämer, M. S., and Kuhnert, K.-D., "Multi-Sensor Fusion for UAV Collision Avoidance," *Proceedings of the 2018 2nd International Conference on Mechatronics Systems and Control Engineering*, Association for Computing Machinery, New York, NY, USA, 2018, p. 5–12. https://doi.org/10.1145/3185066.3185081.

[39] Wang, K., "B-Splines Joint Trajectory Planning," *Computers in Industry*, Vol. 10, No. 2, 1988, pp. 113–122. https://doi.org/https://doi.org/10.1016/0166-3615(88)90016-4.

[40] Schofield, P., "Computer Simulation Studies of the Liquid State," *Computer Physics Communications*, Vol. 5, No. 1, 1973, pp. 17–23. https://doi.org/10.1016/0010-4655(73)90004-0.

[41] Swope, W. C., Andersen, H. C., Berens, P. H., and Wilson, K. R., "A Computer Simulation Method for the Calculation of Equilibrium Constants for the Formation of Physical Clusters of Molecules: Application to Small Water Clusters," *The Journal of Chemical Physics*, Vol. 76, No. 1, 1982, pp. 637–649. https://doi.org/10.1063/1.442716.

[42] Fortun, D., Bouthemy, P., and Kervrann, C., "Optical Flow Modeling and Computation: A Survey," *Computer Vision and Image Understanding*, Vol. 134, 2015, pp. 1–21. https://doi.org/10.1016/j.cviu.2015.02.008.

[43] Hur, J., and Roth, S., "Optical Flow Estimation in the Deep Learning Age," *Modelling Human Motion: From Human Perception to Robot Design*, Springer International Publishing, Cham, 2020, pp. 119–140. https://doi.org/10.1007/978-3-030-46732-6_7.

[44] Shah, S. T. H., and Xuezhi, X., "Traditional and Modern Strategies for Optical Flow: An Investigation," *SN Applied Sciences*, Vol. 3, No. 3, 2021, p. 289. https://doi.org/10.1007/s42452-021-04227-x.

[45] Zhai, M., Xiang, X., Lv, N., and Kong, X., "Optical Flow and Scene Flow Estimation: A Survey," *Pattern Recognition*, Vol. 114, 2021, p. 107861. https://doi.org/10.1016/j.patcog.2021.107861.

[46] Butler, D., Wulff, J., Stanley, G., and Black, M. J., "A Naturalistic Open Source Movie for Optical Flow Evaluation," *ECCV*, 2012, p. 611–625. https://doi.org/10.1007/978-3-642-33783-3_44.

[47] Menze, M., Heipke, C., and Geiger, A., "Joint 3D Estimation of Vehicles and Scene Flow," *Proc. of the ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015, pp. 427–434. https://doi.org/10.5194/isprsannals-ii-3-w5-427-2015.

[48] Menze, M., Heipke, C., and Geiger, A., "Object Scene Flow," *ISPRS Journal of Photogrammetry and Remote Sensing*, Vol. 140, 2018, pp. 60–76. https://doi.org/10.1016/j.isprsjprs.2017.09.013.

[49] Sun, D., Yang, X., Liu, M.-Y., and Kautz, J., "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume," *CVPR*, 2018, pp. 8934–8943. https://doi.org/10.1109/CVPR.2018.00931.

[50] Teed, Z., and Deng, J., "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow," *Computer Vision – ECCV 2020*, edited by A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Springer International Publishing, Cham, 2020, pp. 402–419. https://doi.org/10.1007/978-3-030-58536-5_24.

[51] Wang, J., Zhong, Y., Dai, Y., Zhang, K., Ji, P., and Li, H., "Displacement-Invariant Matching Cost Learning for Accurate Optical Flow Estimation," *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Curran Associates Inc., Red Hook, NY, USA, 2020, p. 15220–15231. https://doi.org/10.5555/3495724.3497000.

[52] Kroeger, T., Timofte, R., Dai, D., and Van Gool, L., "Fast Optical Flow Using Dense Inverse Search," *Computer Vision – ECCV 2016*, edited by B. Leibe, J. Matas, N. Sebe, and M. Welling, Springer International Publishing, Cham, 2016, pp. 471–488. https://doi.org/10.1007/978-3-319-46493-0_29.

[53] Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., v.d. Smagt, P., Cremers, D., and Brox, T., "FlowNet: Learning Optical Flow with Convolutional Networks," *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2758–2766. https://doi.org/10.1109/iccv.2015.316.

[54] Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., and Brox, T., "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4040–4048. https://doi.org/10.1109/cvpr.2016.438.

[55] Farnebäck, G., "Two-Frame Motion Estimation Based on Polynomial Expansion," *Image Analysis*, edited by J. Bigun and T. Gustavsson, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 363–370. https://doi.org/10.1007/3-540-45103-x_50.

[56] Howard, A., Pang, R., Adam, H., Le, Q., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y., "Searching for MobileNetV3," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324. https://doi.org/10.1109/iccv.2019.00140.

[57] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L., "ImageNet: A Large-Scale Hierarchical Image Database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. https://doi.org/10.1109/cvpr.2009.5206848.

[58] Lucas, B. D., and Kanade, T., "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1981, p. 674–679. https://doi.org/10.5555/1623264.1623280.

IV

Thesis report

8 UUFOSim: Unreal UAV Failure injectiOn Simulator

For the development and performance assessment of the fault detection and diagnosis algorithm, it is required to have a dataset which includes IMU and camera output in nominal flight and in failure scenarios. Unfortunately, the current available datasets do not include IMU sensor information, such as the VisDrone dataset [221, 222] or the Indoor Navigation UAV Dataset [223], and do not have any recorded scenarios with failures, such as the UZH-FPV Drone Racing Dataset [224] or the Zurich Urban Micro Aerial Vehicle Dataset [225].

As a result, the first milestone of this master research project is the development of a dataset which fills the gaps left by literature. However, collecting a dataset from real drone flights is immediately faced by three main challenges. First, recording data from failed drones during flight is very time consuming when considering the preparation, safety and failure injection setup, circumstances that would lead to a reduced size dataset. Second, it would be very expensive since the failures would lead to the partial or complete drone maintenance with every flight. Third, the flight arena available at most research labs, such as the CyberZoo at Delft University of Technology, has a limited availability and environment feature richness, with constant factors such as illumination. This last challenge could limit the learning and generalisation capability of any vision and knowledge-based fault detection and diagnosis approach. Therefore, a more efficient, cheaper and safer alternative that would lead to a larger and richer dataset would be the collection of data within a simulated environment. As discussed in the literature, AirSim has been selected to be the simulator of choice for this project mainly due to its adaptability, available Python and C++ APIs, as well as its extensive documentation and support.

The simulator, that the author has named Unreal UAV Failure injectiOn Simulator (UUFOSim), consists of flying a simulated drone or Undiagnosed Failing Object (UFO) in a urban environment avoiding obstacles between two random locations. UFOs fly at a uniformly sampled constant altitude and an actuator failure, within a set of modes, is injected at a random point along the trajectory. During the whole flight, including the manoeuvres after the failure, the camera and IMU data is stored to later shape the dataset for the training and testing of the FDD framework.

Figure 8.1 shows the three main blocks that shape the data gathering pipeline, namely the occupancy map extraction, the drone grid navigation and the drone flight. Once the flight is concluded, including the data storage, the environment and drone are reset to their original state and the cycle is repeated. The loop continues for as many flights as it is desired for building the dataset.

Section 8.1, section 8.2 and section 8.3 discuss each of the presented blocks, from the offline extraction of environment information in the form of an occupancy grid, to the fault injection during flight and data storage. Next, section 8.4 performs a trade-off between data sampling rate and data collection speed as a function of the simulator clockspeed. This is followed by a brief description of a

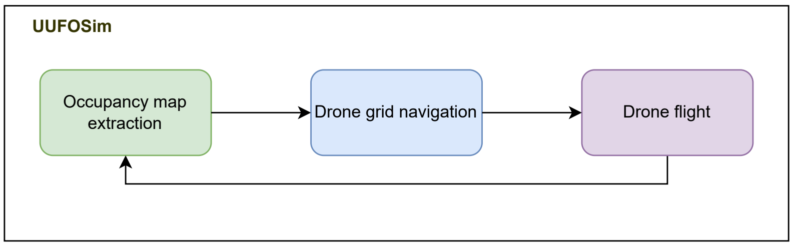


Figure 8.1: Data gathering pipeline block diagram

signal scoping tool developed for the debugging of the simulator in section 8.5. Finally, section 8.6 will present the collected dataset for assessing the FDD framework.

8.1. Environment and occupancy map

As mentioned before, the environment is simulated and can be modified at will but,

- *how can the information of the environment be translated to a data structure that can be easily manipulated within C++ or Python?*

Additionally, most of the environments found in Unreal Engine 4 are quite large with a lot of objects and details. If all that information would be loaded every time that the path planning module has to be executed, every flight or iteration would be computationally expensive and inefficient. Therefore,

- *what is the most adequate data structure in order to compress the environment information such that the time to collect data, equivalent to an iteration, is minimised?*

The answer to both questions can be found in the autonomous driving industry [226, 227], and more recently in the (flying) robotics sector [228, 229], in which the environment around the vehicle or agent is discretised and represented by a grid occupancy map. Under this representation, the environment is discretised into a matrix of inter-independent fix size cells (voxels) which store information about whether they are free or occupied with a 0 and a 1, respectively.

For the purpose of this project, a static 2D grid occupancy map is built in order to encapsulate all the information about the static obstacles found by the drone at its flying altitude prior to executing its flight. Figure 8.2 describes all the steps taken in order to build this environment representation that will ease the computation of the flight path that the flying vehicle should follow.

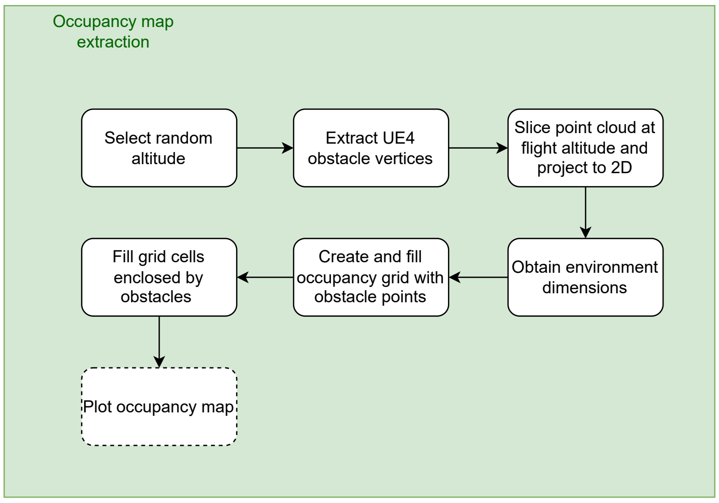

Occupancy map extraction
Select random altitude
Extract UE4 obstacle vertices
Slice point cloud at flight altitude and project to 2D
Obtain environment dimensions
Create and fill occupancy grid with obstacle points
Fill grid cells enclosed by obstacles
Plot occupancy map

Figure 8.2: Occupancy map extraction block diagram.

First, the altitude at which the drone of the current iteration will fly is randomly selected within a range of possible altitudes for the chosen environment. The flying altitude changes from iteration to iteration in order to prevent the overfitting or overreliance of the FDD algorithm to object instances found at a certain height, like bushes, trees or windows, as well as the presence and location of the horizon line in the images taken by the drone.

For the extraction of obstacle information, we exploit the fact that Unreal Engine 4 uses static meshes; geometries shaped by polygons that can be rendered efficiently by the graphics card and that are used to create world geometry, as can be seen from the creation environment of this game engine in Figure 8.3. AirSim provides an API for the extraction of all the (triangular) meshes present in the environment as a Face-Vertex Mesh. As a result, the position of all the vertices, as well as the triplets of vertex indices that shape each of the triangular faces, are given as output within a single data structure for each object present in the scene. At this point in the occupancy map extraction, we have a 3D point cloud in which each point has a label for an object in the environment. Since the extraction of the vertices that shape all the objects of the environment is an expensive operation, once executed for the first time with an environment, the points are stored in a ".p" file such that they can be easily extracted in future drone flights.

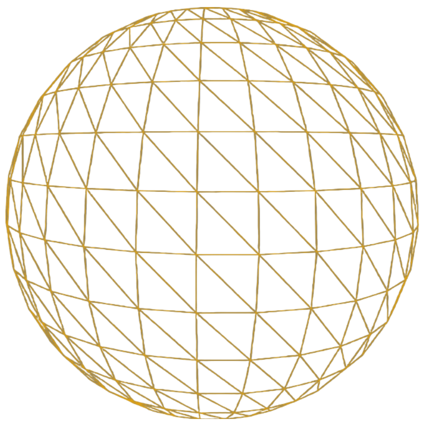


Figure 8.3: Sphere mesh in Unreal Engine 4.

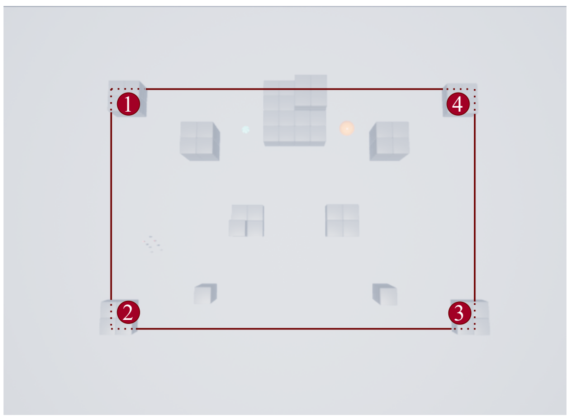


Figure 8.4: "Blocks" environment limits for drone flight bounded by the 4 monoliths in the red rectangle corners.

For the purpose of gathering data for the development and validation of a FDD framework, it has been chosen to fly the drones at a constant altitude during an iteration. Thanks to this assumption, the point cloud can be reduced by slicing it and storing only the points found within a height range around the drone flight altitude $[z - \Delta z, z + \Delta z]$, being z the chosen altitude and Δz the user chosen altitude range for object vertex filtering. To conclude this step, the z-coordinate of all the points is made zero, projecting all the 3D points to the 2D x-y plane. The output is a reduced 2D point cloud.

In Figure 8.4 the reader can observe the "Blocks" environment, the default AirSim environment for testing, adopted due to its simplicity. As can be seen, it consists of grey blocks, an orange sphere (the same one as Figure 8.3), a blue cone and multiple cylinders on the centre left part of the environment. Even though the environments within Unreal Engine 4 are located within a 3D space (canvas) that could extend for kilometres in every direction, it is desired that the drones fly exclusively around the obstacles or assets of the environment and do not venture to fly in empty space. Therefore, the dimensions of the environment and the occupancy map are defined by the furthest points in the 2D point cloud in the x and y direction. In the case of the default "Blocks" AirSim environment, the drone flights are confined to the red rectangle defined by the four monoliths located at the corners, resulting in a 3D point cloud of obstacles of 46,248 points..

As explained, once the 3D point cloud of obstacles is retrieved, it is sliced and projected to the 2D plane. Figure 8.5 shows the 2D point cloud of the obstacles in the scene when the drone flight altitude chosen is seven metres with an altitude range (Δz) of three metres. The points that are close to each other with the same colour are part of the same object. Even though there seems to be about 100 left to the naked eye and a large orange blob, the current figure has 8,956 2D points because there are many overlapping projected vertices from multiple altitudes. This abundance of points can be seen when zooming to the orange blob in Figure 8.6, which happens to be the sphere of the Blocks environment deformed due to the difference in axes scales. As a result, despite having reduced the number of obstacle points by a factor of five (and the information by a factor of almost eight, since the altitude information has been discarded), there are many points that do not provide any information when a clear flight path has to be computed for the drone; only the outer edge of each object or groups of objects carry information that should be preserved for the representation of obstacles in the environment. As an example, all the inner circles of points shown in the sphere could be discarded in further computations.

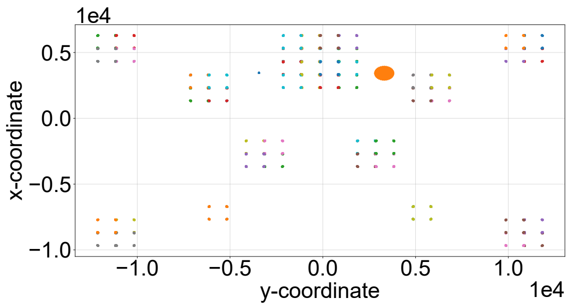


Figure 8.5: 2D projection of the Blocks environment object vertices located within 4 and 10 metres altitude.

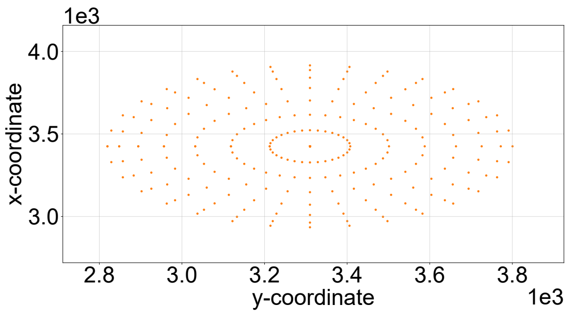


Figure 8.6: Zoom-in of the 2D projected points of the sphere

A solution to the problem of unnecessary points that should be discarded is the introduction of the aforementioned occupancy map. The environment is discretised in cells and those with points within its boundaries are considered as occupied (black) whereas those with no points are empty (white). As a result, independently of how many points are within a cell, they are translated to a single data point.

With the environment dimensions and the cell size provided by the user, it is possible to define the grid that shapes the occupancy map. Next, the 2D point cloud is transformed from the world coordinate frame to the grid coordinate frame, projecting all points into the 2D grid and filling all the cells occupied by obstacle points. This process can be observed in Figure 8.7 and Figure 8.8, where the blue points defining the extracted obstacle vertices of the AirSim ”Blocks” environment are first projected to the 2D grid and then the occupied cells are turned black. From this point onwards, the blue points are discarded and only the grid information is passed along the FDD data gathering pipeline, decreasing the stored data and further computational load. Instead of the initial 8,956 2D points considered before the occupancy grid implementation, now a grid of 80 by 54 cells is used, resulting in a total of 4,320 cells. Given that now the data points (cells) are homogeneously distributed, the information about their location does not have to be stored as long as the x and y dimensions of the occupancy grid are known, meaning that the 2D information has been transformed to a 1D data stream. As a result, the number of data points has been reduced by a factor of two, whereas the amount of information stored has been reduced by a factor of four. Table 8.1 summarises the evolution of the number of points and coordinates (information) through the aforementioned projection and occupancy grid stages.

Table 8.1: Evolution of the number of points and coordinates upon the occupancy map generation.

	Original	2D projection	Occupancy map
Points	46,248 (100%)	8,956 (19.36%)	4,320 (9.34%)
Coordinates	138,744 (100%)	17,912 (12.91%)	4,320 (3.11%)

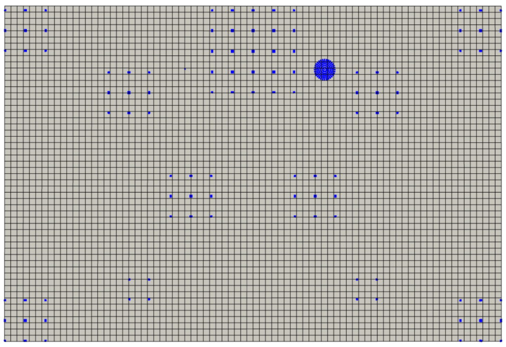


Figure 8.7: 2D points projected in empty occupancy grid.

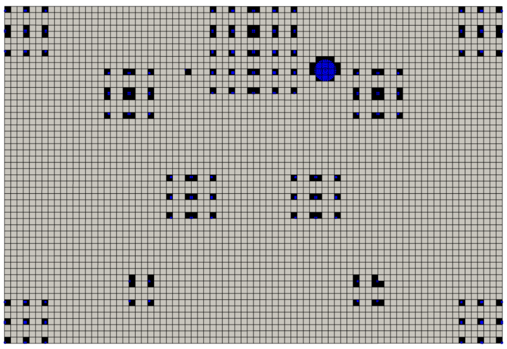


Figure 8.8: Filled occupancy grid with 2D projected points

As can be observed from Figure 8.8, filling the grid cells occupied by obstacle vertices is not enough for creating a reliable occupancy map. There are grid cells that lie within objects, that should not be accessible but that are not marked as occupied since there is no vertex of the static mesh on that particular cell. This problem worsens the finer the mesh of the occupancy map. To solve this, an algorithm is developed that exploits the Delaunay triangulation.

1. First, each of the objects in the environment is assigned the coordinates of the grid cells occupied by their remaining 2D points projected on the occupancy map. Those objects with less than three grid coordinates or whose coordinates shape a line along the x or y axis are discarded from this process since they can not enclose other cells.

2. For each of the remaining objects, the grid cells that define the outer edge of their described polygon (concave hull of a set of points) are identified. For that purpose, Delaunay triangulation is used, which creates a triangular mesh of the object grid cells. It is looped over all the created triangles for each of the obstacles and, in the case that an edge is covered more than once, then it is an internal edge shared by two triangles; it must lie within the polygon whose outer edge is tried to be discovered. Therefore, it is discarded as a potential polygon outer edge.

3. Once the outer edges of the obstacle polygon are identified, is it looped over all grid coordinates and assessed whether they lie within the polygon boundaries. This is done thanks to the Python ”matplotlib.path” module that allows the creation of a polygon based on the counter-clockwise ordered points that shape its outer edge and a method that checks whether given grid coordinates can be found inside the defined polygon. After going through all the empty grid coordinates for each of the (obstacle) polygons, all the grid cells located within obstacles have been identified.

To conclude, all the grid cells found within the environment obstacles are marked as occupied in the occupancy map, leading to Figure 8.9 for the ”Blocks” AirSim environment.

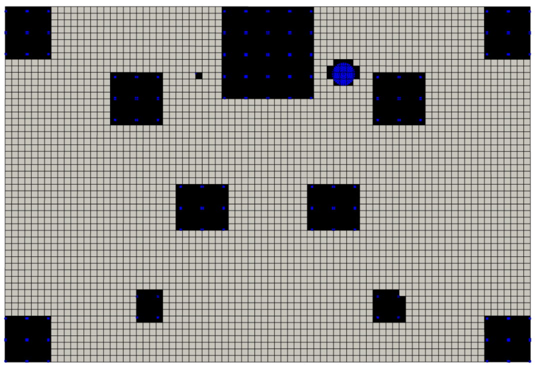


Figure 8.9: Filled occupancy grid considering obstacle inner cells identified with Delaunay triangulation.

8.2. Path planning

As can be observed in Figure 8.1, once the occupancy map has been extracted, it is fed to the drone navigation module which should be able to determine the path that the drone should follow within the Unreal Engine 4 simulated environment. The steps to achieve a smooth drone flight path can be observed in Figure 8.10.

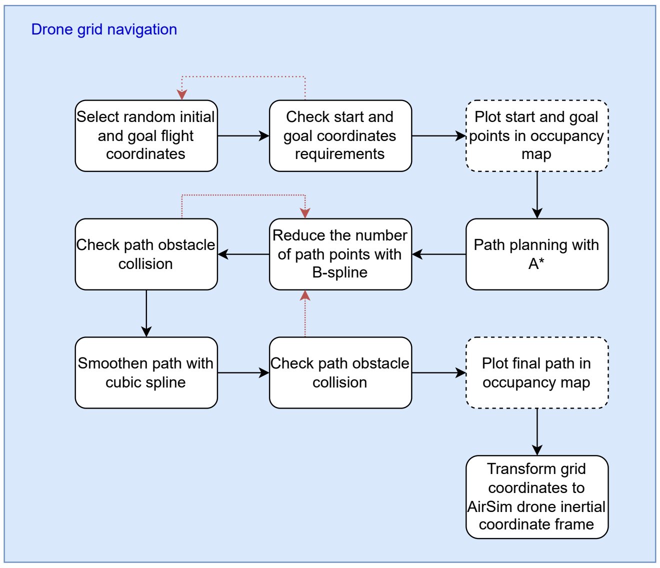

Drone grid navigation
Select random initial and goal flight coordinates
Check start and goal coordinates requirements
Plot start and goal points in occupancy map
Path planning with A*
Reduce the number of path points with B-spline
Check path obstacle collision
Smoothen path with cubic spline
Check path obstacle collision
Plot final path in occupancy map
Transform grid coordinates to AirSim drone inertial coordinate frame

Figure 8.10: Drone grid navigation block diagram.

8.2.1. Start and goal selection

First, a random initial and goal flight coordinates are generated using as upper limits the environment dimensions discovered in the occupancy map extraction. Additionally, it is verified whether those grid locations fulfil three design requirements:

1. The distance between the start and end location is greater than a minimum distance given as input by the user. This prevents extremely short paths which do not allow the injection of failures.
2. The distance between the start and the end location is smaller than a maximum distance given as input by the user. This prevents extremely long paths that would decrease the number of flights that would be executed in the allocated time for data collection and do no add much value to the training of the FDD framework.
3. The start and goal locations have to be located beyond a minimum distance from all identified obstacles in the occupancy map. This guarantees that there will not be any unexpected collision due to the drone dimensions.

As can be seen in Figure 8.10, the random selection and requirement check is repeated until the start and goal coordinates fulfil the established requirements. Once that is the case, they are included within the occupancy map. The occupancy grid coordinate system starts at the bottom left; therefore, the start and goal locations in Figure 8.11 are given as (25, 37) and (50, 50), respectively.

8.2.2. Path planning algorithm selection

Now that the environment is understood and the start and goal locations have been identified, it is necessary to plan the path that the drone should follow. For that purpose, only two types of classic robot path planning methods are considered, namely grid or discrete approaches and road-map methods. A visual classification of the algorithms considered in the present research can be observed in Figure 8.12.

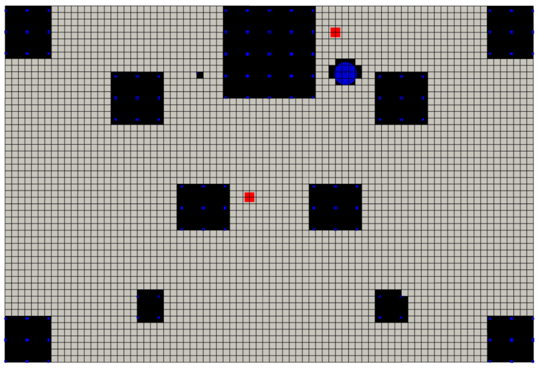


Figure 8.11: Occupancy grid with start and goal locations.

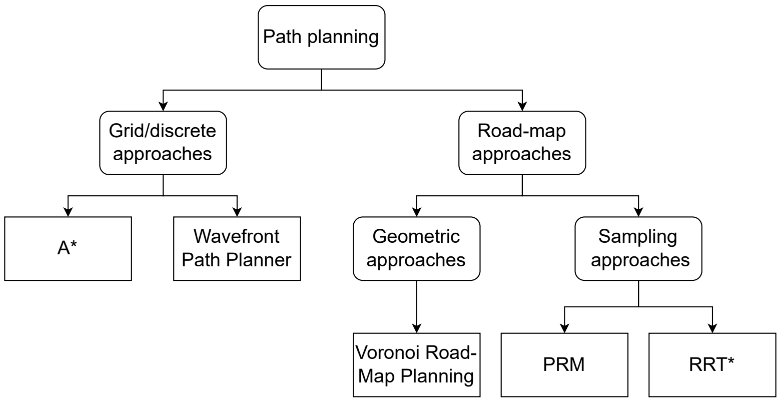


Figure 8.12: Path planning algorithm classification

Within the first group, the Wavefront Path Planner and the A* algorithms are considered. Both methods consist of two steps: a propagation or search step and a back propagation step. Their main difference is that the former applies its initial search (wave propagation) throughout the complete grid whereas the latter uses a function to decide which cells are worth inspecting given the already discovered solution space. As a result, even though every iteration in the search step of the A* algorithm is more computationally expensive, less iterations need to be considered since only a portion of the grid is inspected.

Within the road-map approaches, two other groups can be distinguished, namely geometric and sampling approaches. Voronoi Road-Map Planning would be part of the first class whereas Rapidly-Exploring Random Trees* (RRT*) and Probabilistic road-map Planning (PRM) would be part of the second. The main difference between the sampling options, as it will become clear in their detailed explanation, is that in PRM the drone must pass through randomly sampled points in the solution space whereas in RRT* it only has to move in the direction of those points; it does not require point-to-point convergence.

Literature was consulted for the selection of the aforementioned 5 candidates. First, the authors in [230] show in their benchmark the superior time performance of the A* and Wavefront path planners when compared to RRT* and sPRM, a variant of PRM. However, the authors in [231] argue that the superiority of A* over RRT is connected to the complexity of the environment. While the time performance of A* deteriorates with increasing complexity, the same metric remains stable for RRT. Hence, the author of this research wants to compare the implementations of these four approaches when applied to the developed occupancy grids. Additionally, since it is desired to maintain a safe distance from obstacles in order to minimise the chances of collision and observe the undisturbed failed behaviour of the drone, Voronoi Road-Map planning [232] was also taken into consideration since it aims at maximising the distance from the environment obstacles.

Despite the existence of a richer and more extensive literature around path planning algorithms [233–236], with classical alternatives such as the Potential Field algorithm [237] or modern evolutionary or heuristic-based approaches such as Particle Swarm Optimisation [238], only the mentioned strategies are considered due to their simplicity, efficiency and perfect fit for an occupancy map environment. Heuristic approaches [239] are more efficient than classical approaches in complex dynamic environments with multiple obstacles, properties that can not be attributed to the environments considered in the present research. Additionally, their implementation is more difficult, they require ample tuning and long computation times. In contrast with classical methods, nature-inspired approaches can achieve optimal solutions. However, the goal is not to compute the shortest or fastest path between two points, but to compute as fast as possible any path connecting two provided points, since the algorithm will be part of a data gathering pipeline.

The main metric considered for the selection of the most suitable algorithm for data gathering is the computation time. Also considered but to a lesser extent is the possibility of maintaining a safe distance from all obstacles. Unfortunately, there does not exist a comprehensive benchmark that evaluates the computation time required by the most common methods. Besides that, the results vary greatly between application, testing experiments [240] and actual implementation. Therefore, each of the considered algorithms will be explained and their output when applied to the Blocks environment will be shown. At the end, a short trade-off will be carried out in order to choose the most suitable path planning algorithm.

Voronoi Road-Map Planning

The Voronoi Road-Map Planning is a graph search geometric method, meaning that the solution space is first discretised in the form of a graph using geometrical (instead of sampling) approaches before performing a global search for a minimum cost path. This approach counts with two main steps. First, a road-map is built in the open space of the environment. A road-map is a union of curves such that any start and end point in the occupancy map which is not covered by an obstacle can be connected by a path. The Voronoi Planning algorithm has the property that those curves shaping the road-map are equidistant from the obstacles present in the solution space, maximizing the safe flight distance to any obstacle. Second, the start and goal locations, as well as the points composing the computed road-map, are fed to the Dijkstra search algorithm in order to find the shortest path. The shortest paths from the start and goal coordinates to the road-map lead to the access point to the road-map q', from the start to the road, and the departure point q", from the road to the goal, respectively.

In its essence, this 2D planning approach consists of filtering the available cells of the occupancy grid such that only those that are the furthest from the obstacles (including the start and goal locations) are considered as nodes part of a graph. Then the shortest path between 2 nodes in this graph is computed. Figure 8.13 show the Voronoi Road-Map in action, taking 1.52 seconds in computation time. The blue round points reflect a discretised version of the road-map, the green crosses on top of some blue points reflect the paths that the Dijkstra algorithm studied as potential parts of the solution, the red line is the final flight path and the yellow points reflect the entry (q') and exit points (q") of the drone to the Voronoi road-map. As can be seen, there is a shorter path to the goal if the drone would have taken a shortcut using the road-map routes around the orange ball. As an extension to the vanilla Voronoi Road-Map planning algorithm, the sections of the road-map closer than a minimum distance to the obstacles in the environment are ignored. In the shown example, an obstacle distance of 3 grid cells is used, reason behind the rejection of the alternative narrow alleys.

Wavefront Path Planner

The Wavefront Path Planner can be considered as a breadth-first search of a graph consisting of all the cells in the occupancy grid. Again, this method consists of two main steps. First, a ”wave” is propagated from the goal to all the cells in the occupancy grid, which are also considered as nodes that store the minimum distance to the goal cell. The nodes located at the wavefront are the source nodes, which is the case of the goal in the first iteration. As can be seen in Figure 8.14, the goal node (blue cell)

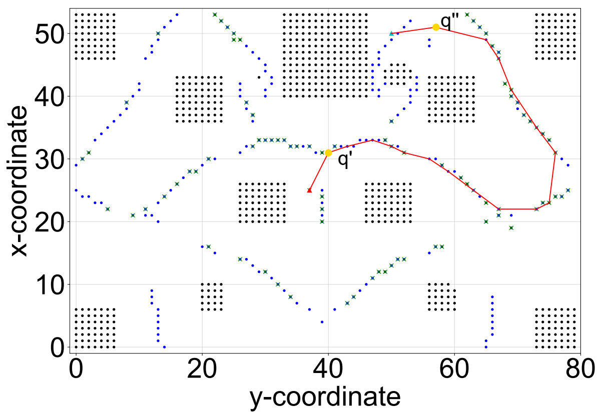


Figure 8.13: Voronoi Road-Map Planning in Blocks environment. Computation time = 1.52 [s]

starts with an initial value of 2, whereas the rest start with a value of zero and the obstacles start with an infinite value. Each of the cells in contact with the source node (the goal) is visited and it acquires the value stored by the source node plus its distance to the source node, as can be seen in Figure 8.15. In the current example, the distance from a cell to all its surrounding ones is 1. With every iteration, the ”wavefront” is in contact with new non-visited cells and this process of distance computation to the goal is repeated. However, instead of the goal cell, now those located at the wavefront are used as source nodes. In the case that the algorithm visits a node adjacent to a source node which already stores a distance value, the visited node will store the minimum value between its stored and the newly computed distance. The wavefront propagation step concludes once all the nodes have become source at least once, as can be seen in Figure 8.16.

0	0	0	0	0
0	0	inf	0	0
0	inf	inf	0	0
0	0	0	0	0
2 S	0	0	0	0

Figure 8.14: Wavefront Path Planner example: starting configuration

0	0	0	0	0
0	0	inf	0	0
0	inf	inf	0	0
3 S	3 S	0	0	0
2	3 S	0	0	0

Figure 8.15: Wavefront Path Planner example: wave step first iteration

6	6	6	7	7
5	5	inf	6	6
4	inf	inf	5	6
3	3	4	5	6
2	3	4	5	6

Figure 8.16: Wavefront Path Planner example: wave step final iteration

With an occupancy grid with all the cells aware of their distance to the goal, the second step consists of finding the shortest path. For that purpose, commencing from the start, an iterative back propagating process is followed in which the node with the lowest stored distance from the adjacent nodes is chosen. As can be seen in Figure 8.17, first the cell with a value of 6 is chosen, since it is the adjacent cell to the start with the lowest distance value. Then, the cell with 5 is chosen, after that the cell with 4, and so on till the goal is reached. Applying this path planning algorithm to the Blocks example, leads to the result shown in Figure 8.18, where the gradient map reflects the distance to the source; cyan means a low distance whereas pink means far a away from the goal. In this case, instead of a distance of one between all the adjacent cells, an euclidean template was used in which a step in the diagonal direction would be a distance of $\sqrt{2}$. Besides that, the cells within a certain predefined distance from the obstacles stored an infinite (”inf”) value to the goal. In this way, they are not considered during the back propagation and guarantee a safe flight distance. It has the same visual effect as inflating the identified environment obstacles in all directions, in this case, by 3 cells, as can be seen from Figure 8.18. It took 3.51 seconds of computation time to generate the proposed path.

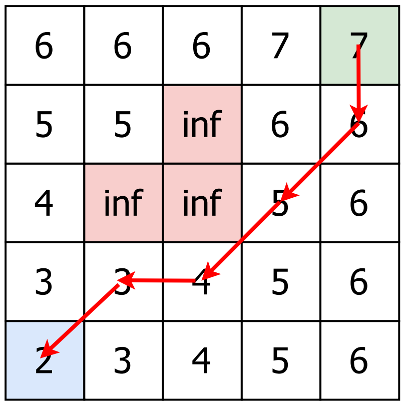


Figure 8.17: Wavefront Path Planner example: back propagation

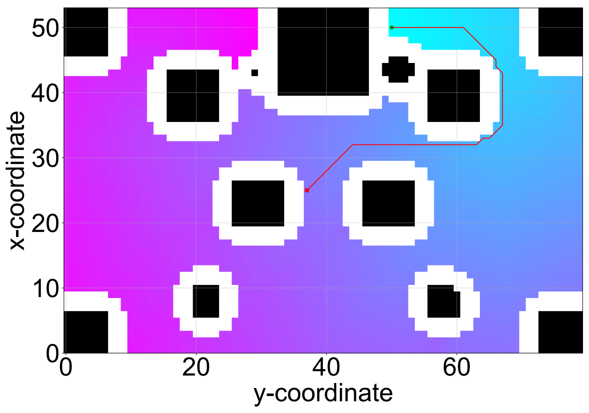


Figure 8.18: Wavefront Path Planner applied to the Blocks environment example. Computation time = 3.51 [s]

Rapidly-Exploring Random Trees
Rapidly-Exploring Random Trees (RRT) is an efficient algorithm for searching non-convex high-dimensional spaces based on building a tree like structure from random samples in the search space. In the case of path planning, 3 main steps can be distinguished. First, a random cell in the occupancy grid is proposed as the new direction in which the tree should be grown q_{rand}. Second, the closest point in the current tree to q_{rand} is identified as q_{near}. Third, the tree is grown from q_{near} in the direction of q_{rand} by a predefined incremental distance or growth factor Δq. In the case that an obstacle is found in the new location, the tree is not grown and the process is repeated. The iterative process is concluded once the tree contains a node in the goal region. By its nature, RRT tends to expand quickly towards unsearched regions.

RRT* is an optimized version of RRT [241] in which each vertex stores the relative distance (cost) it has travelled relative to its parent vertex. In contrast with RRT that accepts the new created vertex from q_{near} in the direction of q_{rand} as long as it does not land on an obstacle, RRT* first checks whether there are vertices in the vicinity of q_{near}, within a predefined radius, that have a lower cost. If that is the case, q_{near} is replaced by this lower cost alternative. Additionally, once a new node is added to the tree, it is checked for its neighbouring tree vertices whether a connection to this new node would lead to a lower cost. If that is the case, the tree is rewired, resulting in smoother and shorter paths than RRT. Even though the output of this optimal RRT option is better, the required computation is higher, in particular due to the tree rewiring and the obstacle avoidance. RRT* is stopped if the maximum number of iterations is reached or once the edge of one of the tree branches is within a predefined distance to the goal and the final branch does not collide with an obstacle.

The result of this planning algorithm in the Blocks example can be observed in Figure 8.19, consuming 5.23 seconds of computation time. In order to guarantee a minimum distance from the obstacles, every new branch and node candidate must maintain a minimum predefined distance from the obstacles in order to be accepted into the tree. In the Blocks example, that is a distance of 3 grid cells. The green lines reflect the tree branches and the red lines are the final flight path. As can be seen, RRT* tends to expand quickly towards the open space regions (exploration) instead of exploiting the already discovered areas. Its main disadvantage are the additional number of hyper-parameters that must be tuned for each specific environment; in particular, the maximum number of iterations expanding the tree and the growth factor.

Probabilistic road-map Planning
The Probabilistic road-map Planning (PRM) [242] is very similar to the Voronoi counterpart in that a road-map is built and the Dijkstra algorithm is used for building the shortest path between a start and goal point. The main difference is that it uses a sampling instead of a geometrical approach when exploring the solution space; the road-map is not equidistant from the obstacles in the environment. As the previous methods, it consists of 2 steps: a construction and a query phase. In the construction

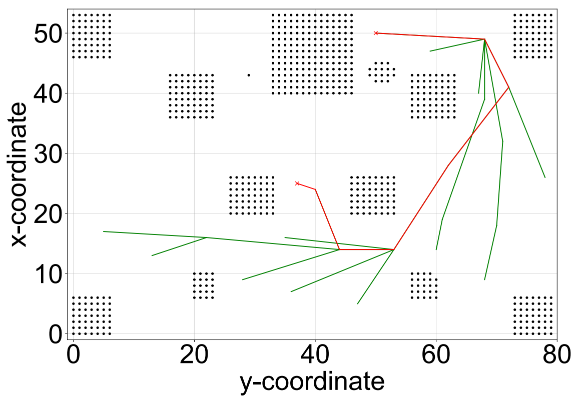


Figure 8.19: Rapidly-Exploring Random Trees* applied to the Blocks environment example. Computation time = 5.23 [s]

space, the unoccupied solution space is randomly sampled and a graph or road-map is built by connecting each node with its k-nearest neighbours. For each connection it is checked that no obstacle is collided. Then, in the query phase, the generated graph is fed to the Dijkstra search algorithm for constructing the final shortest path.

Figure 8.20 shows the result when applied to the Blocks environment, a solution that took 3.53 seconds to be computed. As with the Voronoi method, the blue circles represent the randomly sampled nodes during the construction phase, the green crosses are the points considered by the Dijkstra algorithm for the shortest path during the query phase and the red line describes the final chosen flight path. As can be seen, due to the stochastic nature of the sample points, the final path is not smooth nor the shortest possible. In order to maintain a safe distance from the obstacles in the environment, no points are sampled within a predefined distance to the black points. In the current example that is a distance of 3 grid cells.

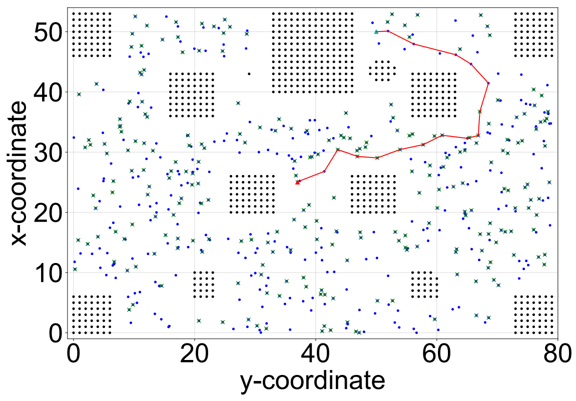


Figure 8.20: Probabilistic road-map Planning applied to the Blocks environment. Computation time = 3.53 [s]

A* algorithm

Finally, the A* algorithm for path planning is an informed approach in which the known location of the start and goal locations is exploited within a evaluation function in order to efficiently find the shortest path. The occupancy grid is considered as a graph in which each cell is a node. The evaluation function ($f(n)$) is the result of the sum of two other functions, namely the operating cost function ($k(n)$) and the heuristic function ($h(n)$), as can be seen in Equation 8.1. When the former is evaluated for a node, it provides information about the distance that it has been covered from the start to reach it. When the latter is evaluated for a node, it provides an estimation of the distance that will have to be traversed to go from that node to the goal. For the present research, the heuristic function used is the straight distance from the node to the goal. This can be seen in Equation 8.2, where $[n_x, n_y]$ are any nodes coordinates and $[g_x, g_y]$ are the coordinates of the goal.

$$f(n) = k(n) + h(n) \tag{8.1}$$

$$h(n) = \sqrt{(g_x - n_x)^2 + (g_y - n_y)^2} \tag{8.2}$$

The nodes store their distance to the start and their parent node, meaning the node closer to the start to which they are connected. Nodes can be classified in 3 groups. First, the visited nodes are those that have been inspected, store their distance to the start cell, store which is their parent node and whose adjacent nodes are known. Second, the explored nodes are those adjacent to visited nodes, whose distance to the start node have been computed and store which is their parent node but whose adjacent nodes are not known. Third, the unexplored nodes are those that are not adjacent to visited nodes and, therefore, their distance to the start, parent node and adjacent nodes are not known.

Figure 8.21 shows how information will be presented in the next example, which is the same as the one discussed with the Wavefront Path Planner. Each node in the occupancy grid contains its value of the evaluation function in the centre, the node id in the top left and a letter in the top right indicating whether it is a visited (v), explored (e) or unexplored node (u). Additionally, under the evaluation function, the value of the operating cost function, the heuristic function and the parent node are stored. As can be seen in Figure 8.22, when the algorithm is initialised, only the start node contains all the information. The rest are unexplored nodes which only contain their id number and their distance to the goal or $k(n)$.

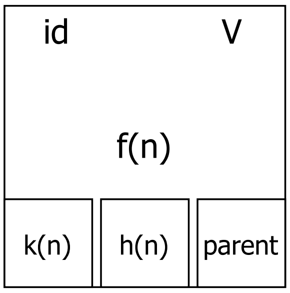


Figure 8.21: A* Path Planner example: single cell

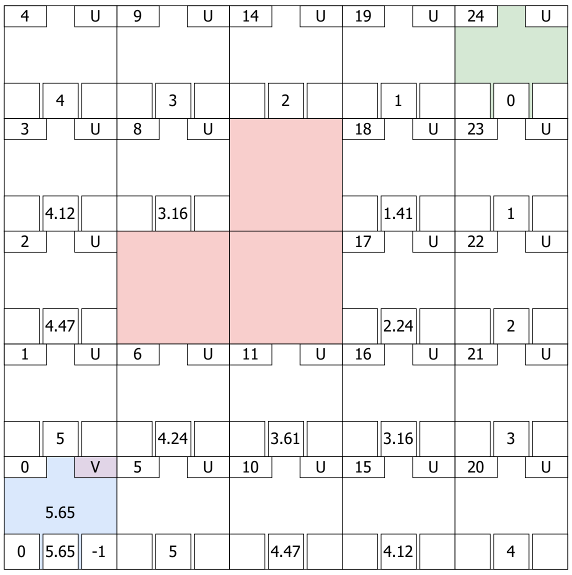


Figure 8.22: A* Path Planner example: starting configuration

As in most previous methods, two main steps can be considered: the search and the back propagation phases. During the search phase, two lists are used to store information about the nodes, namely the open (O) and closed lists (C). The open list stores the explored nodes and the closed list stores the visited nodes. Commencing from the start cell as the first visited node, the evaluation function of its adjacent nodes that are not in the closed list is computed, these nodes become explored nodes and they are added to the open list, as can be seen in Figure 8.23. Then, the node in the open list with the lowest evaluation function value is chosen as the next visited node. This process is repeated until the next visited node is the goal, as can be seen in Figure 8.24, Figure 8.25 and Figure 8.26.

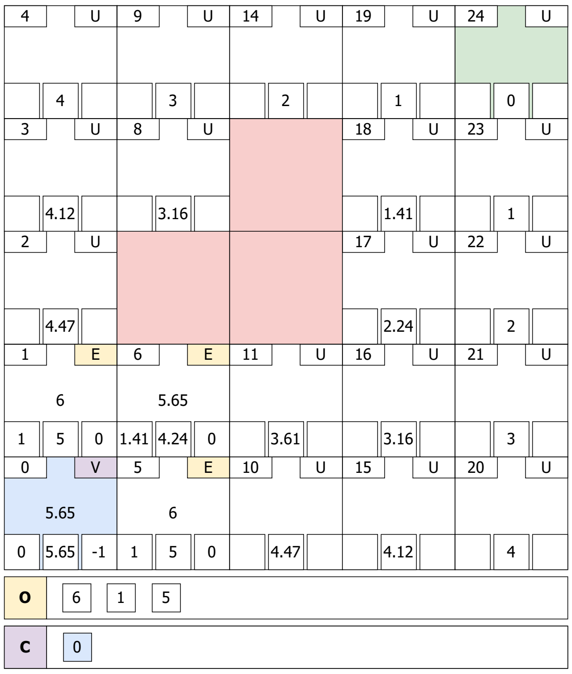


Figure 8.23: A* Path Planner example: first iteration in search step

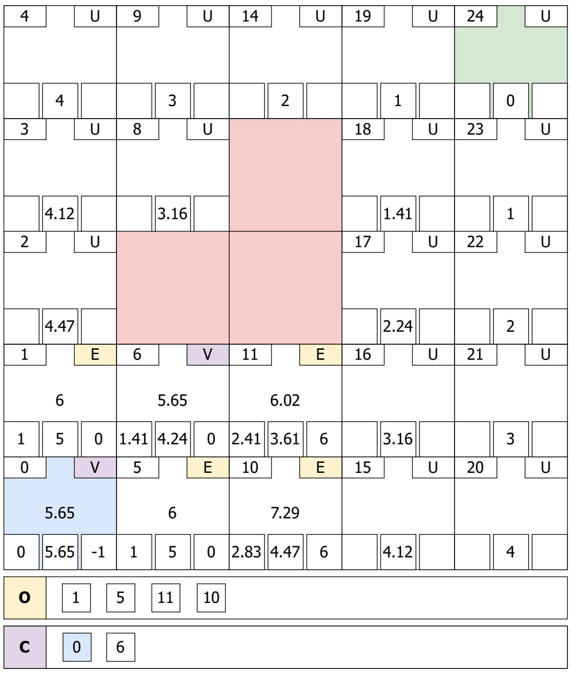


Figure 8.24: A* Path Planner example: second iteration in search step

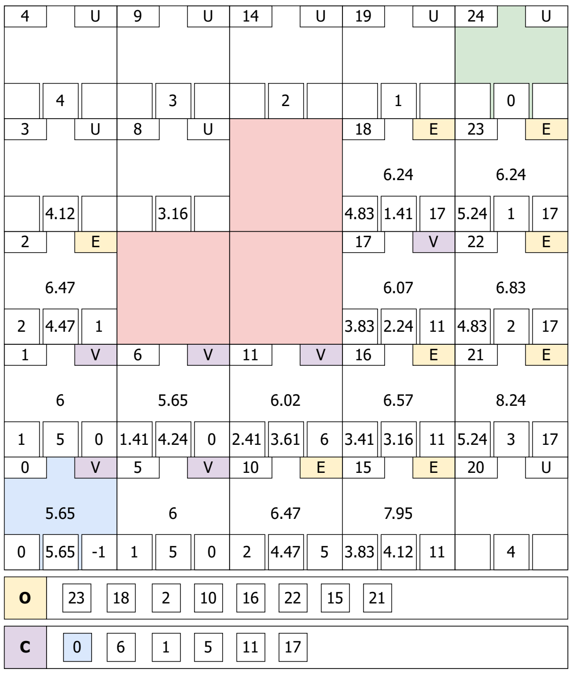


Figure 8.25: A* Path Planner example: penultimate iteration in search step

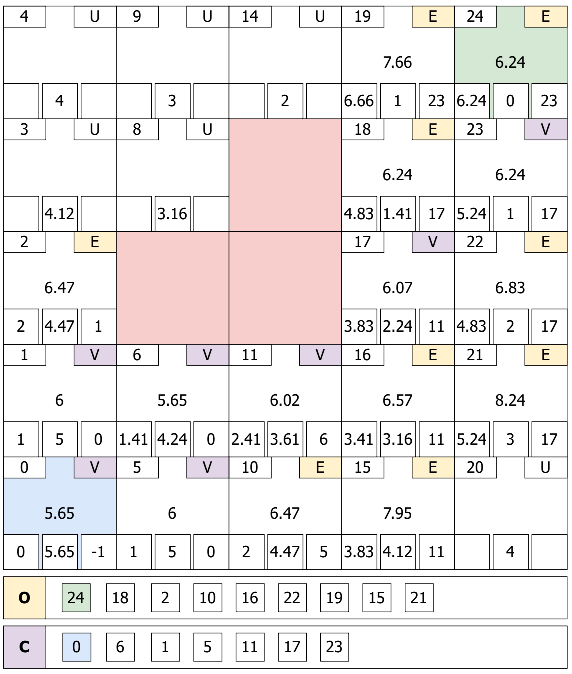


Figure 8.26: A* Path Planner example: last iteration in search step

Finally, back propagation is carried out from the goal following the parent nodes stored by the cells. The goal node points to cell 23, which points to cell 17, and this one to 1, and so on till the start cell is reached, as shown in Figure 8.27. Figure 8.28 shows the A* algorithm applied to the Blocks environment, which required 0.3 seconds of computation. The green dot is the start cell, the blue cross is the goal cell, the cyan crosses are the visited and explored nodes during the search step and the red line is the final back propagated flight path. As can be seen, the cells within 3 units from the obstacles were not considered during the search step in order to guarantee a safe flight.

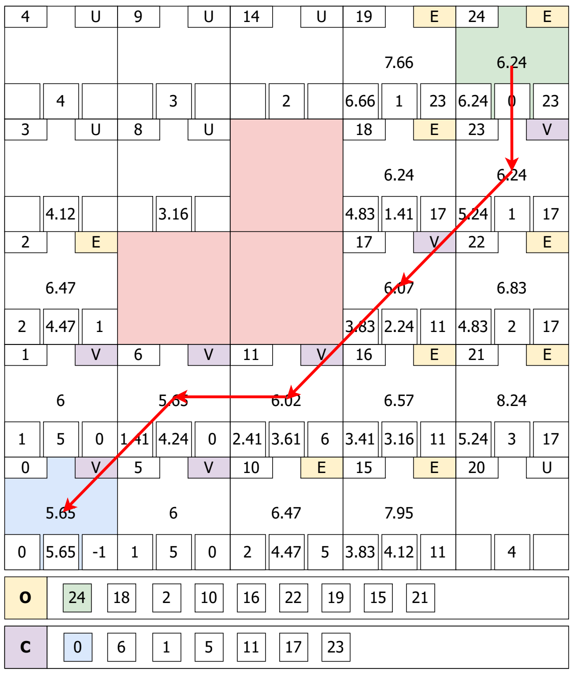


Figure 8.27: A* Path Planner example: back propagation step

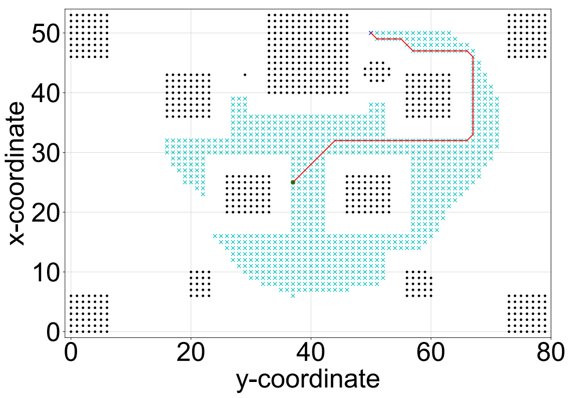


Figure 8.28: A* Path Planner applied to the Blocks environment example. Computation time = 0.3 [s]

Path planning algorithm trade-off

Table 8.2 shows a summary of the two criteria used to judge the path planning algorithms when applied to the Blocks environment and the same start and goal locations as shown in Figure 8.11. As can be seen, all the approaches when adapted can maintain a safe distance from the obstacles in the environment and this safe distance can be tuned for all methods, except for the Voronoi Road-Map planning that tries to maximise it. Regarding the computation time, A* shows superior performance when compared to the rest, followed by Voronoi Road-Map planning. Wavefront Path Planner and PRM show similar computation time, whereas RRT* is the worst performing requiring a time 17 times higher than A*.

Algorithms	Voronoi	Wavefront	RRT*	PRM	**A***
Computation time [s]	3.51	1.52	5.23	3.53	**0.3**
Safe distance	✓	✓	✓	✓	**✓**

Table 8.2: Path planning algorithms trade-off table given single data point.

However, it must be noted that the afore-mentioned results reflect the computation time for the particular scenario presented in Figure 8.11, a single data point. In order to establish a fair comparison, 100 scenarios were created by randomly sampling start and goal locations within the Blocks environment, as well as the flight altitude used to slice the 3D point cloud obtained from AirSim. Since A* showed superior performance for the tested data point, the results will be presented relative to this algorithm. For that purpose, the metric m(x) in Equation 8.3 is computed for each algorithm for each of the 100 iterations. Here, t_x is referred to the computation time that the 'x' algorithm has taken for that scenario, being 'x' the wavefront, Voronoi, PRM or RRT* path planners; whereas t_{A*} is the time that it has taken for the A* algorithm. A positive value means that the algorithm in question has taken longer to compute its path than A*.

$$m(x) = (t_x - t_{A*})/t_{A*} \tag{8.3}$$

Figure 8.29 shows the metric value for each of the algorithms for the 100 iterations in the form of a histogram, including their mean and standard deviations. As forecasted by Table 8.2, all of them perform worse than A* in terms of computation time. The same behaviour is observed: the Voronoi path planner is the next best alternative, PRM and the Wavefront path planner are very similar, and RRT* is the worst performing. Additionally, RRT* got the largest standard deviation, which shows that its performance varies a lot from scenario to scenario. There are even data points with a value larger than 200, which means that RRT* required more than 200 times the time that it took A* to compute its path given the same environment, start and goal locations. Given the presented results, **A*** has been chosen as the algorithm for path planning in the data gathering pipeline.

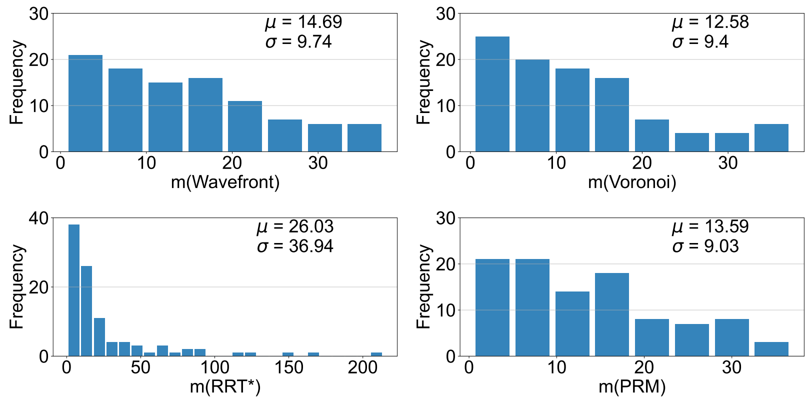


Figure 8.29: Relative computation time for each path planning algorithm with respect to A* (m(x)) for 100 different scenarios.

8.2.3. B-spline path point number reduction

As can be seen in Figure 8.28, there are many points that shape the final flight path that do not provide valuable information. All the points along straight lines can be suppressed and reduced to two points before being fed to the controller. Additionally, some curves could be avoided if straight lines would be taken. An example of this preventable behaviour can be observed in the top right corner of the flight path created with A* in Figure 8.28: the drone would fly next to the block and ball obstacles boundaries till the goal is reached instead of flying in a straight line from the moment it takes the corner around the block to the goal. Whereas A* provided a flight path from the start to the goal using points separated by occupancy grid cell size, B-spline could be used to reduce the path to its most indispensable points.

B-splines [243] of order k are piece-wise polynomials that serve as the basis of spline functions and are capable of generating smooth trajectories connecting a provided set of data points, also known as knots. They are of degree k-1 and k-2 times continuously differentiable. For the present research, B-splines of degree 2 are used, which means that they are of order 3 and are 1 time continuously differentiable. It has been made possible to define the desired number of points present in the filtered path as a percentage of the number of points of the A* path. Since the goal is to create an alternative using only the most indispensable points, the algorithm starts aiming at creating a path only with 5% of the points. If unsuccessful, this point reduction strategy is repeated increasing the percentage of kept points by 5% every time. If the percentage of kept points reaches 100%, then reduction of points is deemed not possible and the A* generated path is passed on to the next step in the path planning pipeline.

In the case that a reduced path was computed, it is checked for collisions with the obstacles in the environment. For that purpose, the vectors connecting each pair of points along the new path are discretised and it is verified whether the grid cells of which they are part of are occupied. Additionally, this process is repeated with two parallel vectors displaced one cell to the right and to the left of the original vector in order to maintain at least one cell distance from all the present obstacles. In the example presented in Figure 8.30, even though the central vector connects the two path points through open space, an obstacle is detected because one of the points of the right parallel vector can be found within an occupied grid cell. The positive detection of collision with an obstacle has the same effect as an unsuccessful reduction of path points: the percentage of kept points is increased by 5% and a new B-spline is generated whose reduced flight path would be later checked for obstacles.

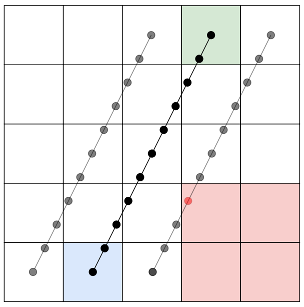


Figure 8.30: B-spline reduced path obstacle detection. The blue and green cells are two path points, and the red cells are occupied by environment obstacles. The black lines are the discretised vectors for obstacle detection.

The benefits of B-spline path point number reduction can be visually appreciated in the occupancy grid. Figure 8.31 and Figure 8.32 show the flight path before and after the B-spline path point reduction was applied. As can be seen, Figure 8.31 is the same path as presented in Figure 8.28 and it consists of 61 points. In contrast, Figure 8.32 only required 9 points to carry out the same path.

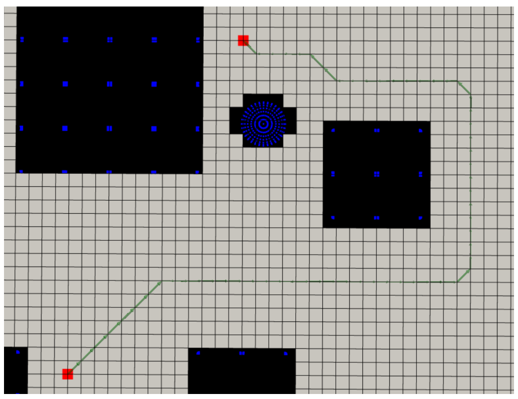


Figure 8.31: Zoom-in of Blocks occupancy map with A* flight path represented by small green arrows.

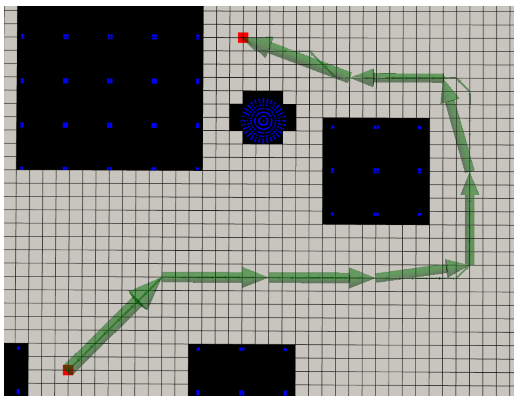


Figure 8.32: Zoom-in of Blocks occupancy map with B-spline reduced flight path represented by large green arrows.

8.2.4. Cubic spline path smoothing

Once a path connecting the start and goal locations has been found with A* and it has been pruned with the B-spline, leading to the essential key points along the path, a smooth trajectory can be computed with a cubic spline that passes through those remaining flight path points. Computation of the cubic spline with the A* flight path as input would have not led to any considerable smoothing given the fine path discretisation to single occupancy grid cells, as can be seen in Figure 8.31. The B-spline allowed the discovery of the pivot points on which the cubic spline can be built.

The smoothing is required in order to avoid sharp corners and ease the workload of the controller later in the pipeline. The distance between each point in the final smooth flight path is again 1 occupancy grid cell size but the points are not constrained to the corner of each cell anymore, as it was the case for the A* and B-spline reduced flight paths. Finally, before accepting the flight path generated by the cubic spline, it is checked for collision with obstacles using the same approach as with the B-spline. In the case of collision, a new B-spline reduced flight path is computed, increasing the kept points by 5%, before estimating a new cubic spline.

The red arrows in Figure 8.33 show the final flight path and Figure 8.34 visually confirms that the cubic spline passes through the pivot points (yellow circles) and the flight path points are not confined to the occupancy grid cells. Additionally, the improvement thanks to the presented approach (small red arrows), which combines B-splines and cubic splines, becomes evident when compared to the original A* flight path (small green arrows).

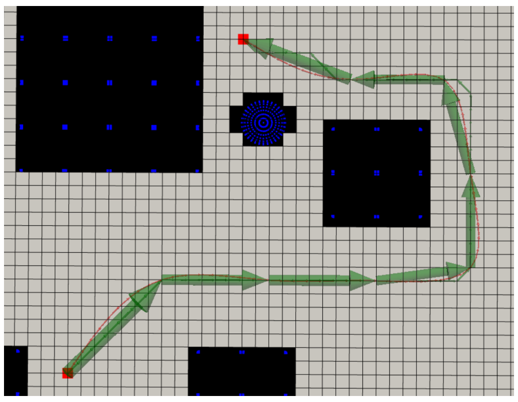


Figure 8.33: Zoom-in of Blocks occupancy map with cubic spline smoothed flight path represented by small red arrows.

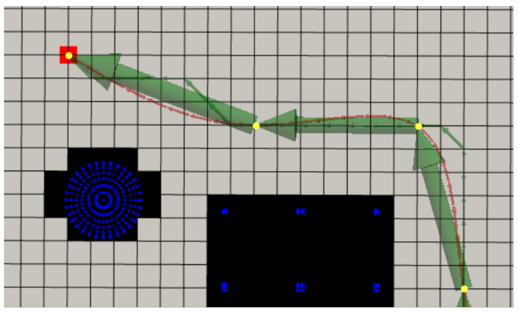


Figure 8.34: Visual confirmation of the final path not being constrained to the occupancy grid and it passes through B-spline pivot points (yellow circles).

8.2.5. Flight path transformation to AirSim drone inertial coordinate frame

Finally, the grid coordinates of the smoothed spline flight path are transformed to the AirSim drone inertial coordinate frame. For that purpose, first the 4 inertial coordinate frames used for the data gathering pipeline are defined next:

1. **Unreal Engine 4 inertial coordinate frame ($\mathbf{ICF_{UE4}}$)**: it is the coordinate frame used to build the environment. Therefore, the centre of coordinates and the direction of its axes vary from map to map. It is a drone independent coordinate frame.

2. **Occupancy grid inertial coordinate frame ($\mathbf{ICF_{OG}}$)**: it has its origin at the bottom left of the occupancy map with the y-axis pointing along the columns (to the right) and the x-axis pointing along the rows (to the top) in the 2D grid; the same directions as those of the $\mathrm{ICF_{UE4}}$. Therefore, all the objects and points in the occupancy map have positive coordinates. The environment has been discretised with cells of predefined size, $\mathrm{C_{UE4}}$. As can be seen in Equation 8.4 and Equation 8.5, in order to define the number of cells in the grid (n_x and n_y), the minimum and maximum X- and Y-coordinates among all the obstacles in UE4 environment ($x_{\mathrm{UE4}_{min}}$, $x_{\mathrm{UE4}_{max}}$, $y_{\mathrm{UE4}_{min}}$, $y_{\mathrm{UE4}_{max}}$) are required. It is a drone independent coordinate frame.

$$n_x = (x_{\mathrm{UE4}_{max}} - x_{\mathrm{UE4}_{min}})/C_{\mathrm{UE4}} \tag{8.4}$$

$$n_y = (y_{\mathrm{UE4}_{max}} - y_{\mathrm{UE4}_{min}})/C_{\mathrm{UE4}} \tag{8.5}$$

3. **AirSim inertial coordinate frame ($\mathbf{ICF_{AS}}$)**: it has its origin at the same location as $\mathrm{ICF_{UE4}}$ with its axes pointing in the same directions. The only difference is the scale of its units: 1 unit in $\mathrm{ICF_{AS}}$ is equivalent to 100 units in $\mathrm{ICF_{UE4}}$. This factor is defined as $\mathrm{k_{UE4}}$. It is a drone independent coordinate frame.

4. **AirSim drone inertial coordinate frame ($\mathbf{ICF_{ASD}}$)**: it is the same as $\mathrm{ICF_{AS}}$ with the only difference that its origin has been shifted to the location where the drone is spawned for the first time within the environment. The location of the drone within the controller is expressed using this inertial coordinate frame. The drone spawn coordinates in the $\mathrm{ICF_{UE4}}$ are given as x_{D_0} and y_{D_0}. It is a drone dependent coordinate frame.

In order to transform the flight path from the occupancy map to the AirSim drone inertial coordinate frame, the transformations in Figure 8.35 were used, using the shown parameters.

$$\begin{tikzcd}
\boxed{\text{ICF}_{\text{OG}}} \arrow[r, Rightarrow, "{x_{\text{UE4}_{min}},\ y_{\text{UE4}_{min}},\ C_{\text{UE4}}}"'] & \boxed{\text{ICF}_{\text{UE4}}} \arrow[r, Rightarrow, "k_{\text{UE4}}"'] & \boxed{\text{ICF}_{\text{AS}}} \arrow[r, Rightarrow, "{x_{D_0},\ y_{D_0}}"'] & \boxed{\text{ICF}_{\text{ASD}}}
\end{tikzcd}$$

Figure 8.35: Inertial coordinate frame transformations: from ICF_{OG} to ICF_{ASD}

Given a trajectory t:[(p_{1_x}, p_{1_y}), (p_{2_x}, p_{2_y}), ..., (p_{n_x}, p_{n_y})] of n points in $\mathbb{R}^2$ in the occupancy grid inertial coordinate frame, the points of the flight path can be transformed to the AirSim drone coordinate frame ($\bar{p}_{i_x}$ and $\bar{p}_{i_y}$) with Equation 8.6 and Equation 8.7.

$$\bar{p}_{i_x} = (p_{1_x} \cdot C_{\text{UE4}} + x_{\text{UE4}_{min}})/k_{\text{UE4}} - x_{D_0} \quad i = 1, 2, ..., n \tag{8.6}$$

$$\bar{p}_{i_y} = (p_{1_y} \cdot C_{\text{UE4}} + y_{\text{UE4}_{min}})/k_{\text{UE4}} - y_{D_0} \quad i = 1, 2, ..., n \tag{8.7}$$

8.3. Data collection

Once the final flight path has been computed, the next step is to fly the drone within the Unreal Engine 4 environment, as can be observed in Figure 8.1, potentially inducing an actuator or sensor failure and gathering all the vision-based and signal data for the FDD training. Figure 8.36 summarises all the steps taken during the final block of the data gathering pipeline. In the following sections, each of the blocks will be briefly discussed.

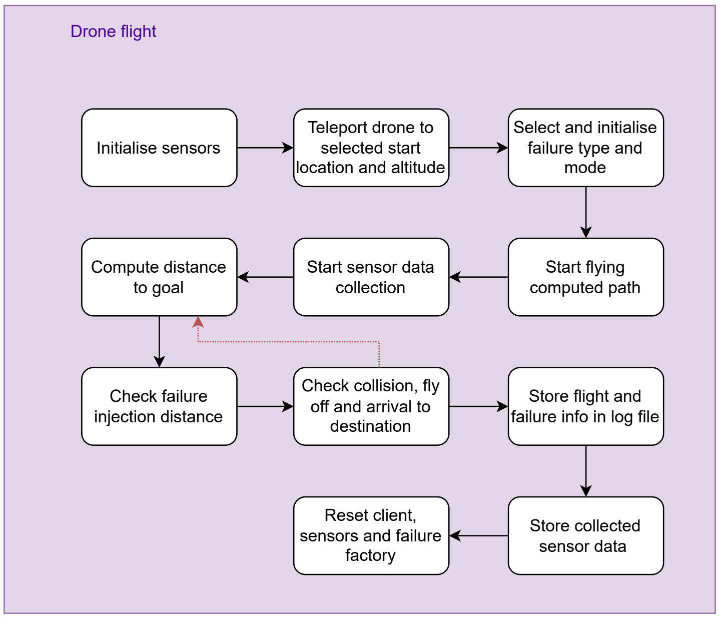

Drone flight
Initialise sensors
Teleport drone to selected start location and altitude
Select and initialise failure type and mode
Start flying computed path
Start sensor data collection
Compute distance to goal
Check failure injection distance
Check collision, fly off and arrival to destination
Store flight and failure info in log file
Store collected sensor data
Reset client, sensors and failure factory

Figure 8.36: Drone flight block diagram

8.3.1. Sensor initialisation and drone teleportation

During the sensor initialisation stage, the data structures in charge of storing the IMU and camera data are created. A single drone can carry multiple cameras, each generating different information such as depth, segmentation or RGB. Besides that, it can be specified whether a float or integer encoding of the pixel is desired, as well as whether the images should or not be compressed.

Once the sensors have been initialised, the drone is teleported to the start location with the heading already pointing towards the first path point in the trajectory. The user can specify whether it is desired for the drone to take-off from the ground or it should be initialised directly at the chosen altitude. This could be deemed useful when analysing failures during the take-off manoeuvre. However, for the purpose of the current research in which the desire is to analyse failures during the cruise phase, the drone is already teleported to the right altitude, as can be observed in Figure 8.37 for the presented Blocks example. The next step is the selection and initialisation of the failure type and mode, as well as the flight and failure info logging, before flying the computed path and collecting all the sensor data.

Figure 8.37: Drone teleported to start location

8.3.2. Failure type & mode selection and initialisation

For the present research, there are four actuator failure types considered with their respective failure modes. The first three failures are simulated by locking the propeller rotational rates to fix values. The fourth type requires a closer look at the propeller aerodynamics and centre of gravity shift, which is explained in detail in chapter 9. The different failures are explained next:

1. **Actuator saturation**: the actuator is saturated and the propeller is locked rotating at its maximum rotational rate. For the Bebop 2 drone, the propellers can attain a maximum rotational rate (ω_{max}) of 1256 rad/s. As a result, when the actuator of a propulsion unit suffers a saturation failure, it is meant that its controller desired rotational rate fed to the physics model is fixed at its maximum value, namely 1256 rad/s for the Bebop 2. The controller output is ignored in favour of the saturated value.

2. **Actuator lock**: the actuator is locked rotating at a percentage of its maximum rotational rate (locking coefficient, k_{lock}). As a result, when the actuator of a propulsion unit suffers a locking failure, it is meant that its controller desired rotational rate is fixed at a percentage of its maximum value. For example, if the front right propeller is locked at 50% of ω_{max}, then the physics model is fed a value of $0.5 \cdot \omega_{max}$ for that propulsion unit, namely 628 rad/s for the Bebop 2. The controller output is ignored in favour of the locked value.

3. **Propeller fly-off**: the propeller flies off and, as a result, complete thrust is lost from this propulsion unit. The damage coefficient of the affected propulsion unit has a value of 0.

4. **Propeller damage**: the propeller has been damaged and, as a result, only a percentage of the desired thrust is attained. Additionally, forces and moments are introduced in the propeller due a shift in the centre of pressure and the centre of gravity from the centre of rotation. These new dynamics cause vibrations along the three axis, phenomenon which is explained and shown in detail in section 9.3.

Each failure type has a different number of potential failure modes. For instance, the propeller fly-off has one failure mode for each propeller, therefore there are four failure modes for this failure type. Additionally, there exist two hyper-parameters that can increase or limit the number of failure modes for each failure type, namely:

1. **Discrete or continuous**: n the continuous case, the degree of failure of the damaged propeller and locked actuator failure types can be given any value in the open interval (0,1), whereas in the discrete case they can only obtain a value of the list <0.2, 0.4, 0.6, 0.8>.

2. **Abrupt or linear**: when abrupt is chosen, the failure takes place in a single time step once it is induced. For instance, if the 50% abrupt damage propeller failure is chosen to take place after 3 seconds in flight, the damage coefficient has a value of 1.0 until the moment when the simulation time is 3.003 (3 seconds plus the nominal physics engine sample time of 0.003), when it acquires a value of 0.5.

   If the linear option is chosen, then a coefficient rate of change with respect to time ($\dot{k}_{fail}$) is randomly selected. It is used to transition from the nominal to the desired coefficient. In the case of the propeller fly-off, it is used to linearly transition from the nominal damage coefficient of 1.0 to 0.0. In the case of the saturated and the locked actuators, it is used to linearly transition from the current percentage of rpms with respect to ω_{max} to the desired locking coefficient. As an example, if the propeller is rotating at 600 rad/s, the desired locking coefficient is 0.2 and $\dot{k}_{fail}$=-0.15 s^{-1}, then the locking coefficient will linearly change from the moment of failure from 0.478 (600/1256) to 0.2 by -0.15 per second.

In order to acquire a balanced data set which could be used for the FDD algorithm training, the user chooses the failure types and modes to include in the data set, and a pool of potential combinations is created from which it is uniformly sampled before every flight. As an example, if the user chooses discrete and abrupt actuator saturation, as well as discrete and abrupt actuator lock, then for each flight the algorithm will randomly sample with a uniform distribution from a pool of 21 alternatives, namely four actuator saturation options (one per propeller), 16 actuator lock options (four per propeller) and one healthy option (no damage).

The aforementioned failure type and mode selection is done by the user prior to the data gathering in order to create a pool of failures from which to choose at run time. Additionally, before each flight, the distance at which the failure is injected is chosen. This is done by randomly selecting a distance along the planned trajectory that is at least five metres from the start and goal locations in order to avoid capturing the transients present at the flight initialization and completion.

8.3.3. Drone flight: guidance, control and physics model

Now that the failure has been selected, the drone is commanded to fly the computed path. For that purpose, the simulation loops over the three blocks shown in Figure 8.38. First, the guidance block creates the reference position (x_r, y_r, z_r) and heading (ψ_r) for the controller given the current vehicle states ($\vec{X}$) polluted with measurement noise (ν) and the trajectory path points defined in the path planning phase (section 8.2). Second, the controller translates the desired position and heading into commanded actuator rotation velocities (ω_{ic}| i=1,2,3,4) taking into account the current states of the vehicle. Finally, the physics model block simulates the effect of those commands on the drone and provides the states at the next time step as measured by the vehicle sensors. In order to simulate as close as possible to reality the behaviour of the Bebop 2 drone, the author implemented within the C++ API of AirSim the Incremental Nonlinear Dynamic Inversion (INDI) controller [41] and the gray-box physics model [244] developed at Delft University of Technology. Each of the three blocks will be further discussed in the current section.

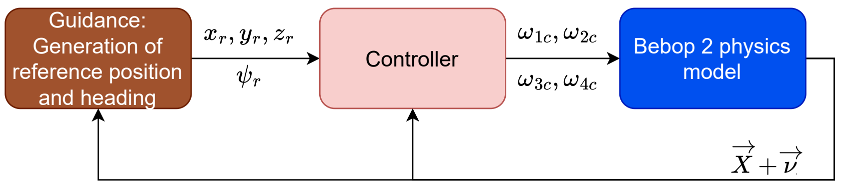

Guidance: Generation of reference position and heading
x_r, y_r, z_r
ψ_r
Controller
ω_{1c}, ω_{2c}
ω_{3c}, ω_{4c}
Bebop 2 physics model
$\vec{X} + \vec{\nu}$

Figure 8.38: Drone flight guidance, controller and physics model pipeline.

The drone flight simulation starts with the guidance block which translates the path points that define the trajectory into intermediate reference locations and headings (reference signals), as shown in Figure 8.39. At every time step, in contrast with feeding directly the next trajectory path point to the controller, the guidance block takes into consideration the current vehicle states in order to avoid passing reference signals very distant from the current ones. The latter could create a large signal error that could drive the controller unstable through time.

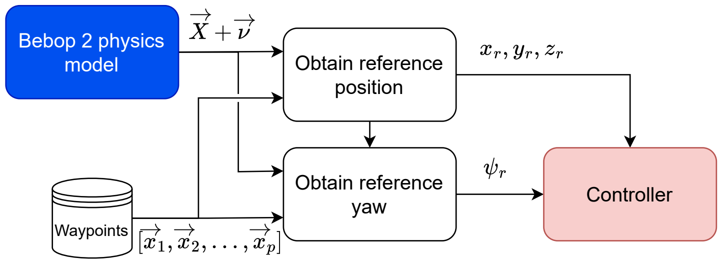


Figure 8.39: Guidance block pipeline.

Contrary to the controller and physics model that originate from previous work at TU Delft, the author exploited the guidance approach already implemented within AirSim and applied some modifications. At its core, this method provides at every time step an intermediate reference position on the line connecting the previous and next waypoints at a user-predefined "Look Ahead" (LA) distance from the current location projected on that line. Figure 8.40 illustrates an example scenario that will aid in the visual explanation of this reference position computation. Three trajectory waypoints computed in section 8.2 are shown as $\vec{P}_1$, $\vec{P}_2$ and $\vec{P}_3$. Furthermore, the drone is initially positioned at $\vec{x}_{(n-1)}$, as represented by a red diamond. When the vehicle is at a waypoint or along the line connecting two consecutive waypoints, as it is the case here, the next reference position is at an LA distance from the current location along that line. The vector that connects the reference location with the vehicle position projected along the waypoint connecting line is called the "Goal Vector" ($\overrightarrow{GV}$), and it is computed by subtracting the reference and actual positions, as shown in Equation 8.8. In this case, the projected position $\vec{x}_{(n-1)_p}$ coincides with the actual drone location $\vec{x}_{(n-1)}$ and the departure waypoint $\vec{P}_1$, but that is not always the case. The current reference position $\vec{x}_{r_{(n-1)}}$ is marked as a green hexagon in the figure.

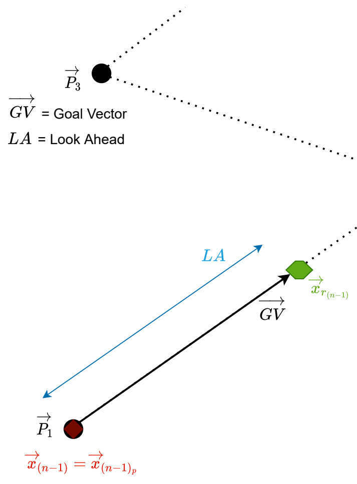


Figure 8.40: Guidance block approach at time step n-1.

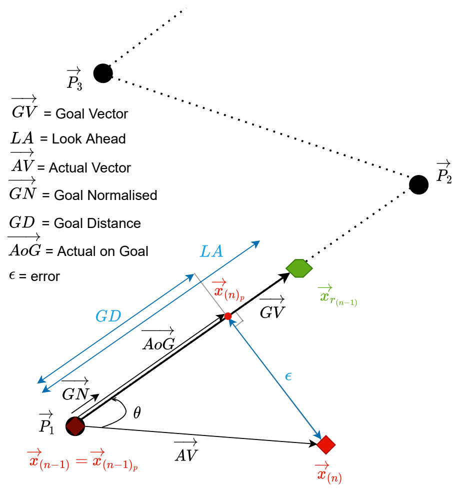


Figure 8.41: Guidance block approach at time step n: geometry before reference position definition.

$$\overrightarrow{GV} = \vec{x}_{r_{(n-1)}} - \vec{x}_{(n-1)_p} \tag{8.8}$$

Next, Figure 8.41 shows the next time step as the drone has flown away from the line connecting the waypoints and is now found at $\vec{x}_{(n)}$. As it can be seen, the vehicle has travelled the "Actual Vector" ($\overrightarrow{AV}$) and has covered a "Goal Distance" (GD) along the line connecting the previous and next waypoints. The former is the difference between the current and previous coordinates, as shown in Equation 8.9. The latter can be computed using the dot product between $\overrightarrow{AV}$ and the normalised $\overrightarrow{GV}$ vector (Equation 8.10), also known as "Goal Normalised" or $\overrightarrow{GN}$, as shown in Equation 8.11. The vector connecting the progress along the waypoint line, namely the difference between $\vec{x}_{(n)_p}$ and $\vec{x}_{(n-1)_p}$, is called "Actual on Goal" ($\overrightarrow{AoG}$) and it is computed according to Equation 8.12.

$$\overrightarrow{AV} = \vec{x}_{(n)} - \vec{x}_{(n-1)} \quad (8.9) \qquad \overrightarrow{GN} = \hat{\overrightarrow{GV}} = \frac{\overrightarrow{GV}}{|\overrightarrow{GV}|} \quad (8.10)$$

$$GD = \overrightarrow{AV} \cdot \overrightarrow{GN} = |\overrightarrow{AV}| \cos\theta \quad (8.11) \qquad \overrightarrow{AoG} = GD \cdot \overrightarrow{GN} \quad (8.12)$$

These vectors allow the computation of the closest distance between the waypoint line and the current vehicle position, also known as deviation error (ϵ), with Equation 8.13. In contrast with the previous situation, now the new reference point $\vec{x}_{r_{(n)}}$ is located at a distance $LA + \epsilon \cdot ALA$ from $\vec{x}_{(n)_p}$ in order to accelerate the error correction. ALA is a user defined value called "Adaptive Look Ahead", which allows to tune the importance of the waypoint line vehicle deviation in the definition of the next reference location. This is shown in Figure 8.42.

$$\epsilon = |\overrightarrow{AV} - \overrightarrow{AoG}| \quad (8.13)$$

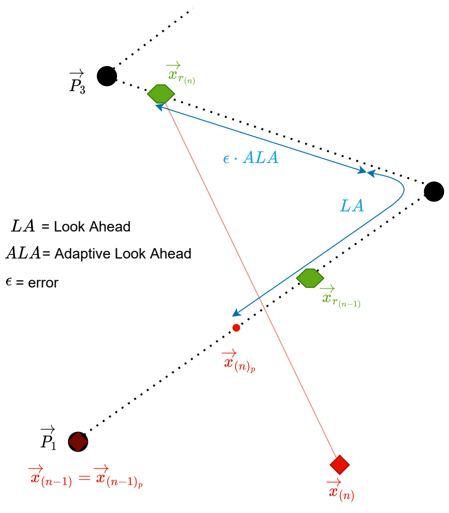


Figure 8.42: Guidance block approach at time step n: reference position definition.

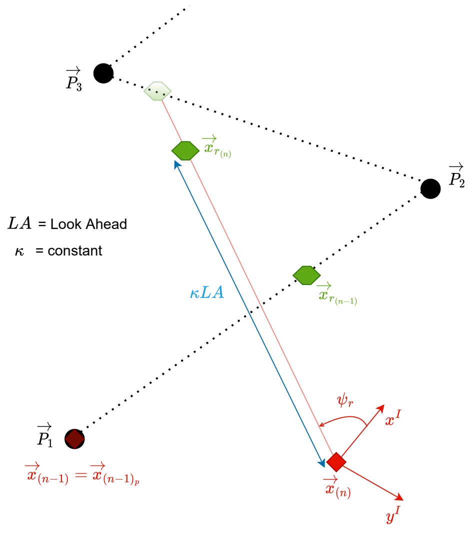


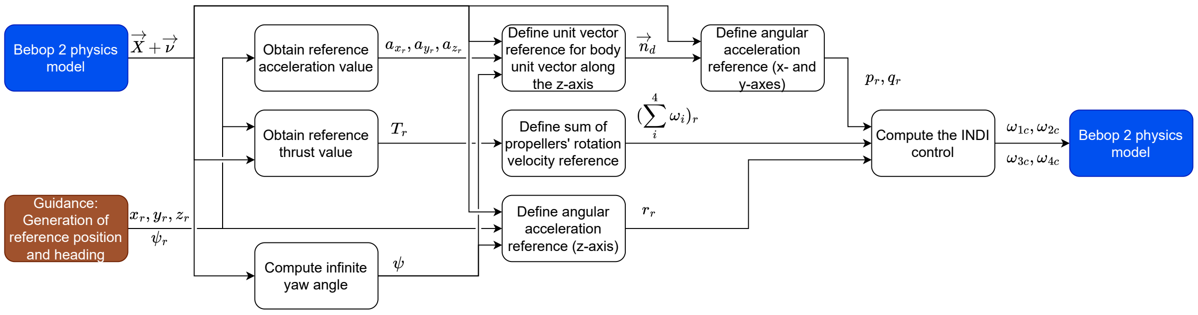
Figure 8.43: Guidance block approach at time step n: reference position correction.

As can be observed when comparing Figure 8.40 and Figure 8.42, the introduction of the adaptive look ahead can increase the error between the reference and actual position considerably. If the controller is not able to correct the deviation fast, it will lead to loss of control. In order to reduce the probability of emergence of this event in simulation, the author has introduced a simple but effective modification in the AirSim guidance block. Once the reference position has been computed, it is checked whether the distance between the reference and current vehicle positions is higher than a constant (κ) times LA. If that is the case, then a new reference position is defined in the same direction as the old one but at a distance of κLA from the current position. The effect of this modification can be observed in Figure 8.43. For the current research, it was found that a value of 1.5 for the constant was effective to reduce the emergence of the aforementioned undesired behaviour.

The second part of the guidance block is the computation of the yaw reference. It is desired that the drone points in the direction of the desired flight path, meaning that it should point towards the reference point found in the reference position definition, as shown in Figure 8.43. Hence, the reference yaw is found with Equation 8.14.

$$\psi_{r(n)} = \arctan \frac{y_{r(n)} - y_{(n)}}{x_{r(n)} - x_{(n)}} \tag{8.14}$$

Regarding the INDI controller, Figure 8.44 shows a schematic of the information flow. The computation of the infinite yaw angle refers to a function that converts it from the closed interval [-π, π] rad to the open interval (-∞, ∞) rad. This is necessary for the INDI controller to track the yaw reference signal without discontinuities. For instance, if the current and reference yaw angles are -178° and 176°, respectively, this function allows the drone to rotate 6° in the negative yaw direction instead of 354° in the positive direction, avoiding a potential loss of control. For the derivation and equations corresponding to the rest of the blocks the reader is referred to [245] and [41].



Bebop 2 physics model
$\vec{X}+\vec{\nu}$
Obtain reference acceleration value
$a_{x_r}, a_{y_r}, a_{z_r}$
Define unit vector reference for body unit vector along the z-axis
$\vec{n}_d$
Define angular acceleration reference (x- and y-axes)
p_r, q_r
Obtain reference thrust value
T_r
Define sum of propellers' rotation velocity reference
$(\sum_i^4 \omega_i)_r$
Compute the INDI control
ω_{1c}, ω_{2c}
ω_{3c}, ω_{4c}
Bebop 2 physics model
Guidance: Generation of reference position and heading
x_r, y_r, z_r
ψ_r
Define angular acceleration reference (z-axis)
r_r
Compute infinite yaw angle
ψ

Figure 8.44: INDI controller information flow schematic.

Finally, the last block in Figure 8.38 is the gray-box aerodynamic model developed at TU Delft, whose schematic is shown in Figure 8.45. It is a piece-wise polynomial whose parameters have been identified with wind tunnel data obtained in quasi-steady flow conditions. As can be observed, it receives as input the commanded actuator rotation velocities from the controller and computes the forces and moments experienced by the vehicle. The integration of those forces and moments leads to the angular and linear accelerations and velocities, as well as the attitude and position of the vehicle; states which are fed again to the controller after measurement noise has been added. The numerical integration scheme followed is the Beeman and Schofield explicit method [246] outlined in Equation 8.15 and Equation 8.16, where x is the position or attitude, v is the linear or angular velocity and a is the linear or angular acceleration. Additionally, Δt is the time step duration and the subscripts n+1, n and n-1 refer to the next, current and previous time steps. It is designed for second order ordinary differential equations, necessary to translate accelerations into positions. This approach substitutes the modified velocity Verlet algorithm [247] for the integration of the linear and angular acceleration, as well as a simple Euler method for the computation of position and orientation, used within AirSim by default.

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{6}\left(4a_n - a_{n-1}\right)\Delta t^2 \tag{8.15}$$

$$v_{n+1} = v_n + \frac{1}{6}\left(2a_{n+1} + 5a_n - a_{n-1}\right)\Delta t \tag{8.16}$$

The explanation of most blocks can be retrieved from [245] and [244]. The actuators are assumed to be all the same, to have first order dynamics, namely a low pass filter with a time constant of τ=1/30, and available actuator feedback. The filter dynamics are shown in Equation 8.17. For the computation of the low pass filter internal state integral, the 2-step Adams-Bashfort explicit integration scheme was used. This method uses the filter internal state derivative from the current and previous time step in order to compute the filter's output, as can be seen in Equation 8.18.

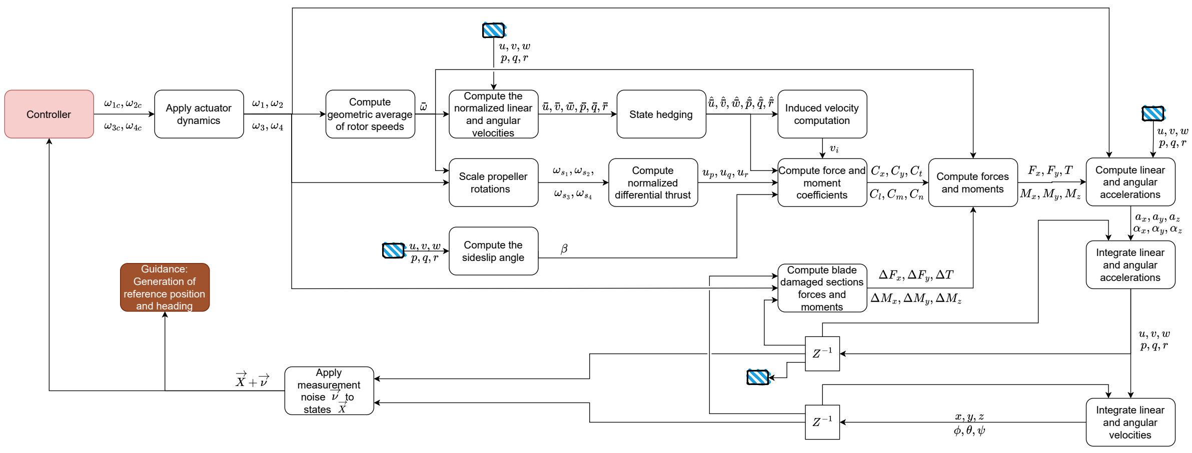

Controller
ω_{1c}, ω_{2c}
ω_{3c}, ω_{4c}
Apply actuator dynamics
ω_1, ω_2
ω_3, ω_4
Compute geometric average of rotor speeds
$\bar{\omega}$
u, v, w p, q, r
Compute the normalized linear and angular velocities
$\bar{u}, \bar{v}, \bar{w}, \bar{p}, \bar{q}, \bar{r}$
State hedging
$\hat{\bar{u}}, \hat{\bar{v}}, \hat{\bar{w}}, \hat{\bar{p}}, \hat{\bar{q}}, \hat{\bar{r}}$
Induced velocity computation
v_i
Scale propeller rotations
$\omega_{s_1}, \omega_{s_2}, \omega_{s_3}, \omega_{s_4}$
Compute normalized differential thrust
u_p, u_q, u_r
Compute force and moment coefficients
C_x, C_y, C_t
C_l, C_m, C_n
Compute forces and moments
F_x, F_y, T
M_x, M_y, M_z
Compute linear and angular accelerations
a_x, a_y, a_z $\alpha_x, \alpha_y, \alpha_z$
Integrate linear and angular accelerations
Compute the sideslip angle
β
Guidance: Generation of reference position and heading
Compute blade damaged sections forces and moments
$\Delta F_x, \Delta F_y, \Delta T$
$\Delta M_x, \Delta M_y, \Delta M_z$
Z^{-1}
$\vec{X} + \vec{\nu}$
Apply measurement noise $\vec{\nu}$ to states $\vec{X}$
x, y, z ϕ, θ, ψ
Integrate linear and angular velocities

Figure 8.45: Bebop 2 drone gray-box aerodynamic physics model schematic.

$$\dot{x}_n = \frac{1}{\tau}(u_n - x_n) \qquad (8.17)$$
$$y_n = x_n$$

$$x_{n+1} = x_n + \frac{3}{2}\dot{x}_n\Delta t - \frac{1}{2}\dot{x}_{n-1}\Delta t \qquad (8.18)$$

Furthermore, the computation of the geometric average of the rotor speeds can be done with Equation 8.19. The linear and angular velocities are normalised by the factors shown in Equation 8.20 and Equation 8.21, respectively. R is the radius described by the propeller and b is the distance of the propeller to the body x-axis. Additionally, the propeller rotations are scaled by multiplying them by $1/\bar{\omega}$ and the normalised differential thrusts for each moment axis are computed with Equation 8.22, Equation 8.23 and Equation 8.24.

$$\bar{\omega} = \sqrt{\frac{\sum_i^4 \omega_i^2}{4}} \qquad (8.19)$$

$$K_{pqr} = \frac{b}{\bar{\omega} \cdot R} \qquad (8.20) \qquad\qquad K_{uvw} = \frac{1}{\bar{\omega} \cdot R} \qquad (8.21)$$

$$u_p = (\omega_{s_1}^2 + \omega_{s_4}^2) - (\omega_{s_2}^2 + \omega_{s_3}^2) \qquad (8.22) \qquad\qquad u_q = (\omega_{s_1}^2 + \omega_{s_2}^2) - (\omega_{s_3}^2 + \omega_{s_4}^2) \qquad (8.23)$$

$$u_r = (\omega_{s_1}^2 + \omega_{s_3}^2) - (\omega_{s_2}^2 + \omega_{s_4}^2) \qquad (8.24)$$

State hedging is a technique which scales the states in order to avoid reaching unachievable values beyond those presented in Table 8.3 for the Bebop 2 drone. In the case of the linear and angular velocities in the z-direction, their values are saturated such that they remain within their ranges. As an example, if the velocity in the z-direction (w) has a value of 0.1, then its value after state hedging would be 0.05. Due to the coupling in the x- and y-directions, Equation 8.25 to Equation 8.28 show the hedged (scaled) value for the linear and angular velocities in those directions. sgn() stands for the sign function.

$$V_{uv} = u^2 + v^2 - u_{max}^2 \qquad\qquad V_{pq} = -|p| \cdot \frac{q_{max}}{p_{max}} + q_{max} - |q|$$

$$\hat{\bar{u}} = \begin{cases} \text{sgn}(u) \cdot \sqrt{\dfrac{u_{max}^2}{v^2/u^2 + 1}}, & \text{if } V_{uv} > 0 \;\&\; |u| > 0.001 \\ 0, & \text{if } V_{uv} > 0 \end{cases} \qquad (8.25)$$

$$\hat{\bar{v}} = \begin{cases} \dfrac{v \cdot \hat{\bar{u}}}{u}, & \text{if } V_{uv} > 0 \;\&\; |u| > 0.001 \\ u_{max}, & \text{if } V_{uv} > 0 \end{cases} \qquad (8.26)$$

Table 8.3: State hedging linear and angular velocities range limits

Parameter	min	max
u	-0.5	0.5
v	-0.5	0.5
w	-0.2	0.05
p	-0.008	0.008
q	-0.02	0.02
r	-0.015	0.015

$$\hat{\hat{p}} = \begin{cases} \dfrac{\text{sgn}(q) \cdot q_{max}}{\text{sgn}(q \cdot p)\frac{q_{max}}{p_{max}} + \frac{q}{p}}, & \text{if } V_{pq} < 0 \ \& \ |p| > 0.001 \\ 0, & \text{if } V_{pq} < 0 \end{cases} \quad (8.27)$$

$$\hat{\hat{q}} = \begin{cases} \dfrac{q \cdot \hat{\hat{p}}}{p}, & \text{if } V_{pq} < 0 \ \& \ |p| > 0.001 \\ q_{max}, & \text{if } V_{pq} < 0 \end{cases} \quad (8.28)$$

Moreover, the sideslip angle is computed with Equation 8.29 and the induced velocity, as well as the force and moment coefficients, are computed with the polynomials whose coefficients were identified in the wind tunnel experiments of [244].

$$\beta = \arctan \frac{v}{u} \quad (8.29)$$

The forces and moments are computed with their coefficients using the templates in Equation 8.30 and Equation 8.31. Besides that, the computation of their corrections due to the potential presence of blade damage will be discussed later in chapter 9 (Equation 9.62 and Equation 9.63). Moving on to the last stages of the physics model, the computation of the linear and angular accelerations is given by Equation 8.32 and Equation 8.33. $\vec{R}_{BI}$ stands for the rotational matrix from the inertial to the body coordinate frames, $\vec{g}$ stands for the gravity vector, m is the mass of the vehicle, $\vec{I}_v$ corresponds to the vehicle inertia matrix and $\vec{M}_{I_p}$ represents the moments due to the gyroscopic effects and rotor spin-up torque (Equation 8.34). Finally, after the integration steps, before returning the states to the controller and guidance blocks, measurement (zero-mean Gaussian) noise $\vec{v}$ is added. The variance values that define the noise probability density function of each state can be seen in Table 8.4.

$$F_. = 4\pi\rho C_.\bar{\omega}^2 R^4 + \Delta F_. \quad (8.30)$$

$$M_. = 4\pi\rho C_.\bar{\omega}^2 R^4 b + \Delta M_. \quad (8.31)$$

$$\dot{\vec{V}} = \vec{F}/m + \vec{R}_{BI}\vec{g} - \vec{\Omega} \times \vec{V} \quad (8.32)$$

$$\dot{\vec{\Omega}} = \left(\vec{M} + \vec{M}_{I_p} - \vec{\Omega} \times \vec{I}_v\vec{\Omega}\right)/\vec{I}_v \quad (8.33)$$

$$M_{I_p} = \begin{bmatrix} qI_p\left(-\omega_1 + \omega_2 - \omega_3 + \omega_4\right) \\ pI_p\left(\omega_1 - \omega_2 + \omega_3 - \omega_4\right) \\ I_p\left(-\dot{\omega}_1 + \dot{\omega}_2 - \dot{\omega}_3 + \dot{\omega}_4\right) \end{bmatrix} \quad (8.34)$$

8.3.4. Sensor data collection, failure injection and flight termination

During the execution of the flight, data is being collected. AirSim provides a Python API for obtaining information from the different sensors and the camera, with simple functions of the style ”get*Sensor*Data()” that return the value registered by the sensor (single data point) at the time of the function call, ”simGetImages()” that returns multiple images at the time of the function call and ”startRecording()” which stores all the camera data according to user predefined settings until ”stopRecording()” is called. Unfortunately, the maximum frequency at which these functions can be called or record data is 26 Hz within the Blocks environment for a single drone, value far from the desired 500-1000 Hz for the IMU or 30-60 Hz frames per second for the camera. As an example, the Bebop 2 drone has a video sampling

Table 8.4: Zero-mean Gaussian noise state variances

State	σ^2	**State**	σ^2
x	$7{\cdot}10^{-6}$	ϕ	$7{\cdot}10^{-5}$
y	$7{\cdot}10^{-6}$	θ	$7{\cdot}10^{-5}$
z	$7{\cdot}10^{-6}$	ψ	$7{\cdot}10^{-5}$
u	$1.4{\cdot}10^{-4}$	p	$6.019{\cdot}10^{-1}$
v	$1.4{\cdot}10^{-4}$	q	$1.75{\cdot}10^{-2}$
w	$7{\cdot}10^{-4}$	r	$6.027{\cdot}10^{-2}$

rate of 30 Hz[1] and it uses the MPU 6050 IMU [248] which has a maximum gyro and accelerometer sampling rates of 10 kHz and 1 kHz[2], respectively. Additionally, this sampling rate changes during the flight, depending on the workload experienced by the different threads within the simulator in Unreal Engine 4 and the other computations carried out by the Python API. For example, within a single flight test, a difference of a factor of 2.3 has been recorded between the fastest and slowest sample rates.

In order to increase the sample rate of the IMU, its data will be collected and stored within C++ before being sent back to the Python API. For that purpose, the IMU data collection has been coupled to the physics engine of the simulator; with every time step in the physics engine it is verified whether IMU information should be stored and, if that is the case, it is saved in a vector for later retrieval. This is done by verifying that the desired time between samples has passed from the last time a sample was saved. As an example, if the desired IMU sample rate is $f_{s_{IMU}}$=100 Hz and the last sample was saved at simulation time of 1 second, then no sample will be saved until the simulation time reaches for the first time a value higher than 1.01 seconds. If that moment is at simulation time 1.013, then the next sample will be saved at a time later than 1.023. Once the flight is concluded, the information regarding that flight is called from the Python API for storage and the vector is cleaned for the next flight. The same process has already been implemented for other signal-based sensors, namely the barometer, magnetometer and GPS. The user only has to choose the sensors to activate and their sampling rate.

Another main benefit of coupling the IMU data retrieval to the physics engine is that it is immune to simulation slow-downs due to time intervals with a high computational load (e.g. when rendering Unreal Engine environment sections with a higher count of assets). If the simulator slows down, then the sensor data collection does it by the same amount thanks to its linkage to the physics engine. The main disadvantage of this approach is that the sample rate choices are limited to factors of the physics engine thread calling rate. With the nominal physics engine thread calling period of 0.003 seconds (f_p = 333.33 Hz), the sampling frequencies available for IMU data gathering are discrete and limited to values of f_p divided by integer values, such as 333.3, 166.7, 111.1, 83.3, 66.7, etc.

With respect to the video sampling rate, it is not possible to couple the image data storage to the physics engine because the AirSim image retrieval functions are part of another simulator thread. When the physics engine is run, the simulator mutex is locked and can not be accessed by the image recording thread. Therefore, if the user would like to retrieve an image within the physics engine iteration, a deadlock is reached: the image thread is waiting for the lock to be released by the physics engine while the latter waits for the image retrieval function to finish in order to release the lock. Therefore, in order to increase the number of frames stored per second (fps), the simulation clock is modified. Instead of running a real-time simulation, in which a second in simulation corresponds to a physical second, the simulation can be slowed down by a factor. For instance, if the clockspeed is set to 0.2, meaning 1 second in simulation corresponds to 5 physical seconds, the camera sampling rate equals 60 Hz.

When the clockspeed is modified, the physics engine sampling rate is modified by the same factor. As a result, slowing down the simulation allows higher IMU sampling rates and higher number

[1]`https://www.drones.nl/drones/parrot-bebop-2-drone/specs`

[2]`https://eggelectricunicycle.bitbucket.io/MicroWorks_30B4_board--Datasheets_30B4--MPU6050_freq.html`

of choices. For instance, using a clockspeed factor of 0.5 would allow an IMU sampling rate of up to 666.66 Hz, with the possibility of choosing 666.7, 333.3, 222.2, 166.7, 133.3, etc. Therefore, the clock-speed factor is a hyper-parameter that has to be tuned by carrying a trade-off between the IMU and camera sampling rates. Such trade-off will be discussed later in this document, namely in section 8.4.

At the same time as data is being gathered, the distance to the goal location is computed and it is assessed whether the drone has reached the distance along the planned trajectory at which the failure should be injected. Once that point is reached, the Python API calls the C++ function that introduces the chosen failure. In the case of actuator saturation or saturation lock, the locking coefficient k_{lock} is changed to the desired value. In the case of the propeller fly-off, the commanded rotational velocity of the lost propeller is overwritten with a value of zero ($\omega_x = 0$). In the case of blade damage, as it will be explained in chapter 9, the forces and moments that the lost blade sections would hypothetically generate are subtracted from those computed by the Bebop 2 gray-box aerodynamic model in subsection 8.3.3. Depending on whether the failure is abrupt or continuous, the mentioned parameters are changed at a single time step to the desired value or they are changed progressively with $\dot{k}_{fail}$.

Once the failure has been injected, the simulation pipeline starts to check whether any of following flight termination conditions has been reached:

1. Collision with the ground
2. Collision with an obstacle
3. Drone flies above a predefined altitude
4. Timeout, meaning a predefined number of seconds after the failure injection has been reached

The first two flight termination conditions can be discovered thanks to the ”simGetCollisionInfo()” function available within the Python API. It detects whether the vehicle has collided against any asset and, if that is the case, it provides its name. If the name of a ground asset is returned, then the first termination condition is activated; otherwise, the second. Also, when the drone descends below an altitude of 0.75 meters, then the first termination condition has been reached. For the present work, the third and fourth activation conditions are activated when the drone flies beyond two times the desired cruise altitude and when 2 seconds have passed since the failure injection, respectively.

8.3.5. Flight & failure metadata logging and sensor data storage

Once the flight has been concluded, some flight and failure metadata are stored for debugging purposes, as well as for the classification of the gathered data for the training of FDD algorithms. The data that are collected for each flight are summarised and briefly explained in Table 8.5. The only parameters that could lead to confusion are *Failure_mode* and *Failure_mode_local*. When the user selects failure types and modes, the pool of options is created. In the previous example in which the user chose discrete and abrupt actuator saturation, as well as discrete and abrupt actuator lock, the number of potential options was 21. In this particular scenario, *Failure_mode* is a number between 1 and 21 that reflects the chosen mode among all the options. If the chosen option is e.g. 10, then *Failure_type* is ”actuator_locked”, *Failure_mode* is 10 and *Failure_mode_local* is 5 since the first five refer to the actuator saturation and healthy actuator options.

After logging the flight and failure information, all the sensor data is stored in *Sensor_folder*. The data recorded by the IMU at every time step of the flight are stored in a .csv file organised according to the headers shown in Table 8.6. All the camera frames are stored in the same directory as the IMU, each image with the name of the timestamp at which it was taken in order to preserve the temporal sequence information. Furthermore, some additional metadata involving the camera, containing the information shown in Table 8.7, is stored in the same directory as the collected frames.

Finally, after logging valuable metadata from the flight and storing the sensor information, the client, sensors and failure factory are reset to their original values in order to repeat the complete data gathering pipeline shown in Figure 8.1 for the next flight.

Table 8.5: Flight and failure logged metadata

Parameter	Explanation
Iteration	Flight number or flight iteration
Sensor_folder	Name of the folder where the sensor data for that flight has been stored
Start_timestamp	Simulator timestamp at which the flight and sensor data gathering started
End_timestamp	Simulator timestamp at which the flight and sensor data gathering ended
ClockSpeed	Clockspeed at which the simulator runs
Failure	Boolean for the presence of a failure: 1 means that a failure was injected
Failure_type	Failure type that was injected from the user selected failure types
Failure_mode	Failure mode chosen from all those available within the user selected failures types
Failure_mode_local	Failure mode chosen from those available within the chosen Failure_type
Time_modality	Boolean for the activation of abrupt/linear failure mode: 1 means discrete failure mode
Continuity	Boolean for the activation of the discrete/continuous failure mode: TRUE means continuous failure mode
Time_linear_slope	$\dot{k}_{fail}$ in the case of linear failure type, otherwise a value of -1
Magnitude_start	Propeller rotational speed percentage with respect to ω_{max} at failure injection
Failure_magnitude	Magnitude of the chosen failure (e.g. 0.25 locking coefficient)
Start_propeller_angle	Propeller rotation angle at which the failure was injected (propeller damage failure type parameter)
Blade	Blade index that was damaged (propeller damage failure type parameter)
Failure_timestamp	Timestamp at which the failure was injected
Distance	Distance from the goal location at which the failure was injected
Percent_trip	Percentage of the planned flight that was completed before the failure injection
Collision_type	Type of collision after failure: no collision and reached goal location (0), collision with obstacle (1), collision with ground (2), flown away exceeding predefined altitude limit (3), 2 seconds have passed since failure (4) and 40/*clockspeed* seconds have passed since flight initialisation (5)
Camera_fps	Frames per second at which camera data was gathered
IMU_frequency	Inertial measurement unit sampling frequency

Table 8.6: IMU data storage headers and explanation

Parameter	Explanation
timestamp	Timestamp at which the IMU measurement was taken
orientations_w	Drone orientation in quaternions: rotation value
orientation_x *orientation_y* *orientation_z*	Drone orientation in quaternions: x, y and z coordinates
angular_velocity_x *angular_velocity_y* *angular_velocity_z*	Drone angular velocity around its x-, y- and z-axes
linear_acceleration_x *linear_acceleration_y* *linear_acceleration_z*	Drone linear acceleration along the x-, y- and z-axes

Table 8.7: Camera logged metadata

Parameter	Explanation
camera_name	Number of the camera
camera_position.x_val *camera_position.y_val* *camera_position.z_val*	Initial camera x, y and z coordinates
camera_orientation.w_val	Initial camera orientation rotation value
camera_orientation.x_val *camera_orientation.y_val* *camera_orientation.z_val*	Initial camera orientation x, y and z coordinates
pixels_as_float	Boolean for saving the pixel information: 1 means that it will be saved in float format
compress	Boolean for saving a png compressed image: 1 means that it will be compressed
width	Width of the captured frame
height	Height of the captured frame
image_type	Type of image captured: Scene (0), DepthPlanar (1), DepthPerspective(2), etc.

8.4. Clockspeed selection

The right clockspeed is a function of the image resolution; the larger the image, the lower the camera sampling rate at the same clockspeed. Therefore, a trade-off needs to be performed between sampling rate accuracy and simulation speed for the chosen image resolution of 256×144 (width×height). Figure 8.46 and Figure 8.47 show 20 simulations at different clockspeeds and their camera and IMU sampling rates. It can be observed that a clockspeed of 0.6 has a large spread of camera sampling rates between flights and the clockspeeds of 0.4 and 0.5 have IMU sampling rates far below the desired 512 Hz. Since the remaining clockspeeds show similar performance, 0.3 was selected for being the fastest.

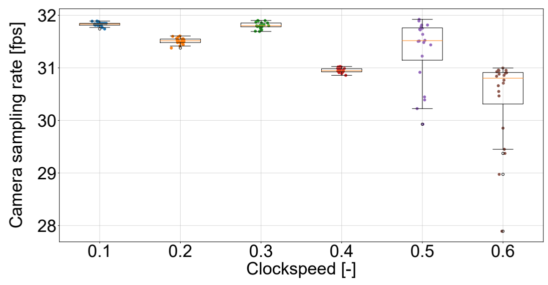


Figure 8.46: Boxplot of the camera sampling rate for different clockspeeds with an image resolution of 256×144 pixels (width×height).

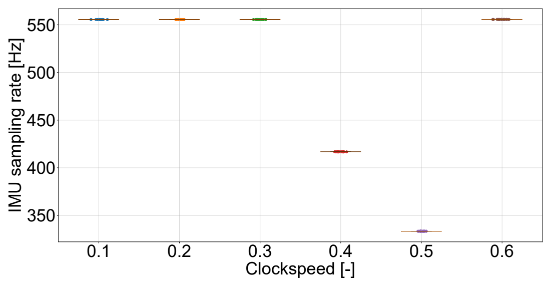


Figure 8.47: Boxplot of the IMU sampling rate for different clockspeeds with an image resolution of 256×144 pixels (width×height).

The IMU sampling rate samples are almost constant at the same clockspeed because the data gathering of this sensor has been coupled with the simulator's physics model, as dicussed in section 8.3. In contrast, the camera sample rates are much more dispersed, especially the higher the clockspeed. For the same simulation time and slower clockspeed, the simulation checks the thread that receives the calls from the Python API more frequently. Hence, the frequency at which camera images can be called is higher, reducing the impact of simulation slow downs and, hence, the camera sampling rate dispersion.

5,000 flights were flown with a clockspeed of 0.3 and image resolution of 256×144. To verify that the camera and IMU sampling rate predictions estimated with 20 flights were accurate, the same box plot was created with the flown 5,000 flights. The results are shown in Figure 8.48 and Figure 8.49: the camera runs at 31.81 fps and the IMU has a sampling rate of 555.59 Hz.

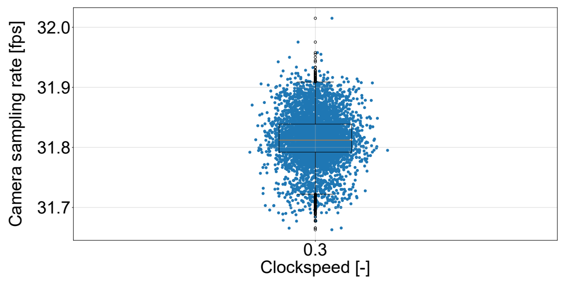


Figure 8.48: Boxplot of the camera sampling rate for 5,000 flights with a clockspeed of 0.3 and an image resolution of 256×144 pixels (width×height).

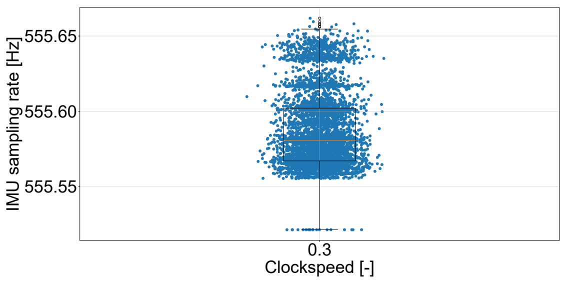


Figure 8.49: Boxplot of the IMU sampling rate for 5,000 flights with a clockspeed of 0.3 and an image resolution of 256×144 pixels (width×height).

8.5. Debugging tool: signal scoping

In a similar fashion as when IMU data are gathered (subsection 8.3.4), any signal within the guidance, control and physics model can be stored in a vector within the C++ API during the flight in order to be plotted upon flight termination with a call from the Python API. The goal is to achieve a similar functionality as the signal scoping tool within Matlab to ease the debugging of these simulator components. The result of such implementation is a single user input in the Python API in which it must specify the signals it would like to plot. Additionally, it has the option of choosing which signals to plot together in a single figure for direct comparison. Once the flight is concluded, they are plotted to the screen for analysis and saved in the computer for later inspection.

The introduction of this tool has accelerated the debugging process of the simulator and it will facilitate the smooth development and implementation of alternative forms of guidance and control, as well as different drone physics models, by future users. Finally, Figure 8.50 and Figure 8.51 show two examples of scoped signals. The first shows the 3D trajectory that the drone followed when commanded to fly in a straight line in the x-direction. The second shows the commanded propeller rotation velocities for the same flight.

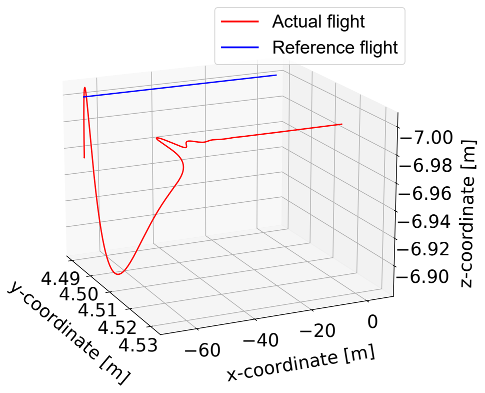


Figure 8.50: 3D actual and reference trajectories for single sample flight.

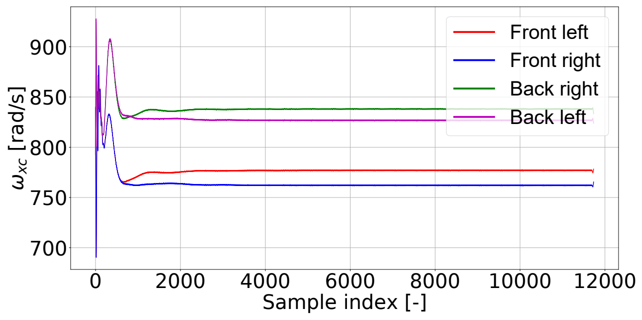


Figure 8.51: Propeller rotational velocity of each actuator in rad/s for sample flight.

8.6. Dataset

The simulation pipeline discussed in Figure 8.1 was run in a Windows OS PC with a 20 core Intel Xeon W-2255 CPU, 32 GB of RAM DDR4 and an NVIDIA RTX A4000 GPU with 16 GB of GDDR6 memory. The 5,000-flight dataset was collected in 61.67 hours and has a memory footprint of 239 GB. Only blade damage failures of 20%, 40%, 60% and 80% were simulated since those are the failure modes that will be used to train and test the FDD pipeline. A sample of frames from a single flight separated by 35 frames from each other can be observed in Figure 8.52.

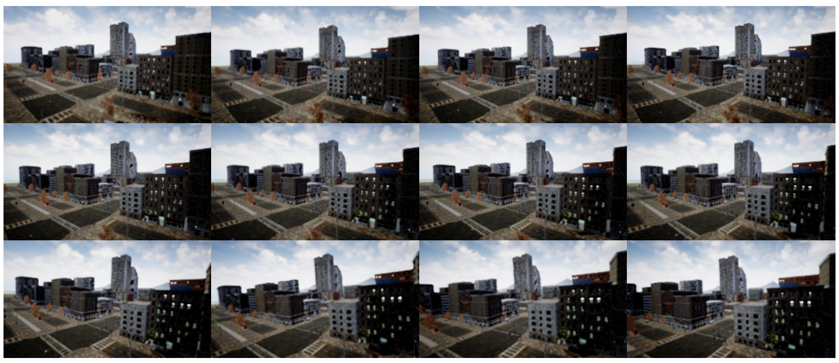


Figure 8.52: Camera captured frames during single flight read from top to bottom and from left to right (only shown one every 35 frames).

9 Propeller damage

In contrast with the other actuator failure modes outlined in subsection 8.3.2, the propeller damage is more difficult to simulate. Whereas the alternative actuator failure modes are symmetrical failures in which the rotational velocity of the propeller is fixed at a certain value or the complete propeller has flown off, propeller damage caused by the chipping or breaking of a blade leads to asymmetrical forces and moments acting on the system that go beyond the change in thrust.

Previous literature in the field of fault diagnosis have exploited simplifications of the simulation of blade damage. The authors in [249] consider actuator faults, such as structural damage to the propellers or degradation of the rotors, as partial loss of effectiveness, which causes a partial loss of thrust generated by the damaged rotor. This is simulated by multiplying the commanded rotor angular velocity by a factor lower than 1 in order to obtain the ”true” rotor angular velocity. The main drawback of this approach is that all the vibrations in the system due to the unbalance of forces and moments are ignored.

Another approach is proposed by the authors in [250], which introduce sinusoids in the force signals to simulate the vibrations caused by the propeller unbalance. The sinusoids only consist of the decomposition of the centrifugal force in the x and y components caused by the displacement of the propeller centre of gravity due to blade damage. Unfortunately, this approach does not consider the vibrations in the moment signals, as well as the vibrations caused by the changed aerodynamics.

This section aims at carrying out a more in depth analysis in the problem of blade damage simulation, such that its effects can be included in the Unreal Engine 4 simulation upon drone failure. The goal of the present chapter is to obtain the forces and moments that have to be added to the healthy counterparts computed by the original physics model in order to account for the damaged propeller. The flow of these computations is visually illustrated in Figure B.1. For that purpose, section 9.1 will present the created forces and moments due to the change of the propeller mass and its centre of gravity shift. Furthermore, section 9.2 will show how the change in forces and moments is computed due to the partial loss of blade aerodynamic surface. This section exploits the mathematical process of Blade Element Theory and it constitutes the main contribution of this research to propeller damage simulation. Then, section 9.3 will present the final force and moment signals when the afore-mentioned effects are combined, as well as the assumptions taken in the presented approach and some recommendations for future work. Finally, the results will be followed by an experimental campaign in section 9.4 that aimed at the validation of the BET model.

9.1. Mass related force and moment changes

Due to the change in propeller mass and its corresponding shift in its centre of gravity, the following moments and forces have to be taken into consideration for a single propeller:

1. The gravity force of the propeller is decreased due to a decrease in its mass. The gravitational

force in the inertial z-direction has to be decomposed in the 3 components of the propeller frame. Output: $\vec{F}^P_{m1} = \left[F^P_{m1_x} \quad F^P_{m1_y} \quad F^P_{m1_z}\right]^T$.

2. Due to the shift in the centre of gravity, there is a moment created with the gravity force vector and the arm from the centre of the propeller to the new centre of gravity location. This moment is changing direction with the propeller rotation and it is decomposed in the x,y and z directions. Output: $\vec{M}^P_m = \left[M^P_{m_x} \quad M^P_{m_y} \quad M^P_{m_z}\right]^T$.

3. A centrifugal force appears due to the shift of the centre of gravity out of the centre of rotation of the propeller. This centrifugal force is changing direction with the propeller rotation and it is decomposed in the x and y directions. Output: $\vec{F}^P_{m2} = \left[F^P_{m2_x} \quad F^P_{m2_y} \quad 0\right]^T$.

They are all of measured in the propeller's reference frame, which is the equivalent to the body reference frame translated to the centre of the propeller hub. As can be observed, the change in mass causes a change in the 3 components of the force and moments signals. Next, each of the aforementioned contributions will be analysed.

The first forces to be obtained are those caused by the loss of mass. Since the goal is to compute the forces and moments that have to be added to those resulting from the physics model when there is no failure, the force required to be added is in the opposite direction of the gravity vector, as can be seen in Equation 9.1. Here, m_{loss} is the lost mass and $\vec{R}_{PI}$ is the transformation matrix from the inertial to the propeller coordinate frame, which can be seen in Equation 9.2. Depending on the drone attitude, the gravity vector can have a value in its 3 dimensional components in the propeller coordinate frame.

$$\vec{F}^P_{m1} = \vec{R}_{PI} \begin{bmatrix} 0 \\ 0 \\ -g m_{\text{loss}} \end{bmatrix} \tag{9.1}$$

$$\vec{R}_{PI} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{9.2}$$

Second, the shift in the centre of gravity (CG) causes the appearance of moments around the centre of rotation of the propeller. In order to compute these moments, the arm from the propeller central hub to the new CG location must be computed. For that purpose, the blade has been modelled as a group of trapezoids. As can be observed in Figure 9.1, in the case of the Bebop 2 propeller, its blade can be split up in two trapezoids connected at their base which is situated at the location of the largest blade chord. c_r, c_t and c_c are the chords lengths at the root, tip and the location of longest chord, respectively.

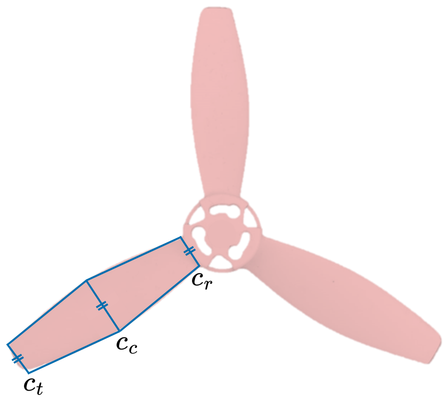


Figure 9.1: Bebop 2 propeller top view and trapezoid simplification

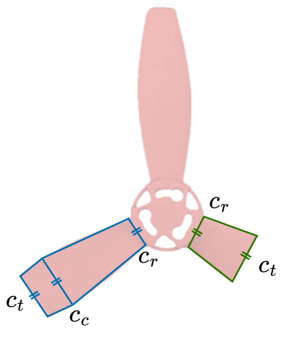


Figure 9.2: Damaged Bebop 2 propeller top view and trapezoid simplification

The CG of each blade is computed separately depending on whether it has damage or not. In the case that there is a damage, the tip chord will move along the span of the blade toward the central hub. In the case that damage causes partial blade loss closer to the central hub than the location of c_c, then there would be only one trapezoid in the blade planform and c_c would disappear, as can be seen in Figure 9.2.

For the computation of the centre of gravity, it is assumed that the density of the blades is constant, so that the CG will coincide with the centroid of the blade. Hence, the centroid of each trapezoid is computed with Equation 9.3 and they are weighted together with their respective areas (which should be proportional to the mass given our assumption) with Equation 9.4. n_t stands for the number of trapezoids within a blade and Figure 9.3 illustrates the trapezoid geometrical variables.

$$\bar{y}_{c_{\text{trapezoid}_i}} = \frac{h_i}{3}\frac{2c_{i+1}+c_i}{c_{i+1}+c_i} \tag{9.3}$$

$$\bar{y}_{\text{CG}_{BL}} = \bar{y}_{c_{BL}} = \frac{\sum_i^{n_t} \bar{y}_{c_{\text{trapezoid}_i}}(c_i+c_{i+1})h_i/2}{\sum_i^{n_t}(c_i+c_{i+1})h_i/2} \tag{9.4}$$

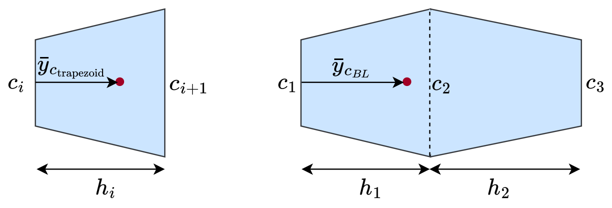


Figure 9.3: Blade and trapezoid geometry and centroid

Once the centroid of each of the blades are found, they are weighted again with their respective areas in order to find the centroid of the complete propeller. With the location of the CG computed as in Equation 9.5 and the gravitational force calculated in Equation 9.1, the moments caused by the gravity force are computed in Equation 9.6.

$$\vec{r}^P_{CG} = \begin{bmatrix} x^P_{CG} \\ y^P_{CG} \\ 0 \end{bmatrix} \tag{9.5}$$

$$\vec{M}^P_m = \vec{r}^P_{CG} \times \vec{F}^P_{m1} \tag{9.6}$$

Third, the shift of the centre of gravity causes the appearance of a centrifugal force due to the rotation of the propeller. This force does not exist in a healthy propeller because the arm between the CG and the centre of rotation is zero. The magnitude of the centrifugal force is computed with Equation 9.7, where m_P is the propeller mass, ω is the rotational velocity and r_{CG} is the distance between the centres of rotation and gravity, as computed in Equation 9.8.

$$F_{m2} = m_P \omega^2 r_{CG} \tag{9.7}$$

$$r_{CG} = \sqrt{x^2_{CG} + y^2_{CG}} \tag{9.8}$$

The centrifugal force is later decomposed in the x and y components, leading to the vector shown in Equation 9.9. θ_{CG} is the angle that $\vec{r}_{CG}$ creates with the propeller coordinate frame and is computed with Equation 9.10. This centrifugal force is illustrated in Figure 9.4.

$$\vec{F}^P_{m2} = \begin{bmatrix} F_{m2}\cos\theta_{CG} \\ F_{m2}\sin\theta_{CG} \\ 0 \end{bmatrix} \tag{9.9}$$

$$\theta_{CG} = \arctan\frac{y^P_{CG}}{x^P_{CG}} \tag{9.10}$$

Figure B.2 visually illustrates the discussed computation of the damaged propeller mass related forces and moments with a flowchart. The next step is the computation of those corresponding to the change in aerodynamics.

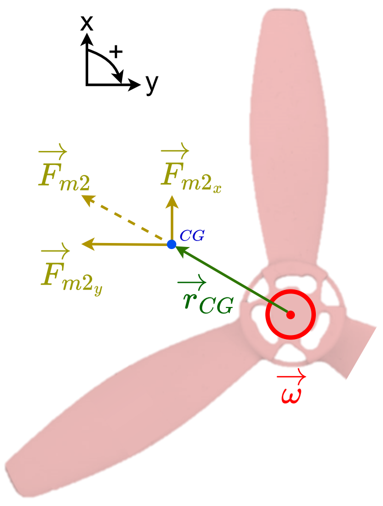

x
+
y
$\overrightarrow{F}_{m2}$
$\overrightarrow{F}_{m2_x}$
CG
$\overrightarrow{F}_{m2_y}$
$\overrightarrow{r}_{CG}$
$\overrightarrow{\omega}$

Figure 9.4: Illustration of centrifugal force on a damaged propeller

9.2. Aerodynamics related force and moment changes

After the analysis of the disturbance in forces and moments due to the change in the propeller mass, the next point of analysis is the change due to the modified aerodynamics. Compared to the healthy state, the following forces and moments can be found for a single propeller:

1. Loss of thrust due to the loss of blade sections. Since the forces are measured in the propeller reference frame, a loss of thrust causes only a change in the force vector z-direction. Output: $\vec{F}^P_{a1} = \begin{bmatrix} 0 & 0 & F^P_{a1_z} \end{bmatrix}^T$.

2. Loss of the torque moment and change of forces in the x-y rotor plane due to a reduction in the blade drag and lift. The force that generates the torque from the missing blade sections can be decomposed in the x-y direction; force components that oscillate due to the propeller rotation. Output: $\vec{F}^P_{a2} = \begin{bmatrix} F^P_{a2_x} & F^P_{a2_y} & 0 \end{bmatrix}^T$ and $\vec{M}^P_{a1} = \begin{bmatrix} 0 & 0 & M^P_{a1_z} \end{bmatrix}^T$.

3. Change in moment due to the change in the location of the centre of pressure of the propeller. Since the force of this moment is parallel to the z-axis, the moment will be decomposed the x-y plane. This moment vector constantly changes direction due to the rotation of the propeller. Output: $\vec{M}^P_{a2} = \begin{bmatrix} M^P_{a2_x} & M^P_{a2_y} & 0 \end{bmatrix}^T$.

Again, it can be seen that the aerodynamic changes of the blade create forces and moments along the 3 propeller axes. For their computation, the Blade Element Theory mathematical process is exploited. This method discretises the blade along its span in sections of equal length, determines their individual generated moments and forces, and adds all of them in order to obtain those generated by the complete propeller. When a blade is damaged, the forces and moments that would had been generated by the missing blade sections would be subtracted from those computed by the physics model in the healthy state.

For the present research, the Blade Element Theory (BET) will be first explained in subsection 9.2.1. From the explanation it will become evident the need to obtain the C_l and C_d curves with respect to the angle of attack of the Bebop 2 blades. Hence, subsection 9.2.2 will present the identification of this curves. This process is carried out using as data the thrust and torque obtained from a gray-box model developed in previous research at Delft University of Technology with wind tunnel experiments [244]. Finally, subsection 9.2.4 will show how each of the forces and moments listed at the start of the present section (section 9.2) are computed given the model developed using BET.

9.2.1. Blade Element Theory

The goal of the BET method is the computation of the thrust and torque generated by the complete blade through the sum of the contributions of all its sections. For that purpose, the lift and drag equations, which can be seen in Equation 9.11 and Equation 9.12, are applied to each of the blade sections k of span length equal to dr. For the rest of the paper, the subscript i stands for the propeller, j for the blade and k for the blade section.

$$\Delta L_k(r_k, \psi_k) = \frac{1}{2} C_{l_k}(\alpha_k(r_k, \psi_k)) \rho V_{A_k}^2(r_k, \psi_k) c_k(r_k) dr \tag{9.11}$$

$$\Delta D_k(r_k, \psi_k) = \frac{1}{2} C_{d_k}(\alpha_k(r_k, \psi_k)) \rho V_{A_k}^2(r_k, \psi_k) c_k(r_k) dr \tag{9.12}$$

ρ is the air density which depends on the altitude at which the drone flies with respect to the sea level. For the present research, it is always assumed that it is flown at sea level conditions, meaning that ρ equals 1.225. Furthermore, c_k is the blade section average chord. Because the blade is abstracted as two trapezoids, c_k is equal to the average of the chords at the start and end of the blade section. V_{A_k} is the airspeed seen by the blade section perpendicular to its span and C_{l_k} and C_{d_k} are the lift and drag coefficients of the 2D blade airfoil, respectively. As can be seen, these parameters are a function of the angle of attack (α_k), the distance from the blade section centroid to the centre of rotation (r_k) and the blade section azimuth angle (ψ_k). The last one is an angle measured on the propeller plane and it is defined to have a value of zero degrees (ψ=0) in the direction of the drag and it increases its value in the direction of rotation. The r and ψ definitions can be visualised in Figure 9.5 and Figure 9.6. When a variable is a function of r and ψ, it will be represented by $(\cdot)$ for readability purposes.

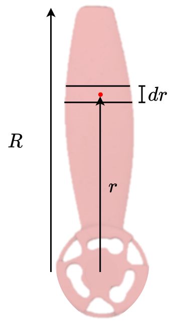


Figure 9.5: Blade geometrical parameters

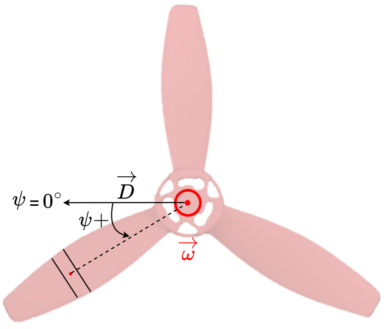


Figure 9.6: Azimuth angle visualization

The airspeed at each blade section (V_{A_k}), which, as it will be shown later, is essential for the computation of the angle of attack, has to be computed taking into account three main components:

1. The combined linear and angular velocities of the drone.
2. The rotational velocity of the propeller, taking into account the distance of the blade section to the centre of rotation.
3. The induced velocity, which is the additional velocity in the propeller z-axis direction through the rotor disk caused by the propeller's air suction for the generation of lift. This movement of air with the induced velocity is called induced flow or downwash.

First, Equation 9.13 is used to compute the linear velocity of the propeller assembly ($\vec{V}^P$) from the body linear ($\vec{V}^B$) and angular velocities ($\vec{\Omega}$). The $\vec{d}$ matrix (Equation 9.14) is used to convert the rotational rates of the vehicle to linear velocities, exploiting the known drone geometry shown in Figure 9.7 [28]. Each row of the d matrix ($\vec{d}_i$) corresponds to each of the drone propellers. The first row corresponds to the left-front propeller and the following rows to the other propellers moving clockwise from a top-down view of the drone.

$$\vec{V}_i^P = \vec{\Omega} \times \vec{d}_i^T + \vec{V}^B \tag{9.13}$$

$$\vec{d} = \begin{bmatrix} l & -b & 0 \\ l & b & 0 \\ -l & b & 0 \\ -l & -b & 0 \end{bmatrix} \tag{9.14}$$

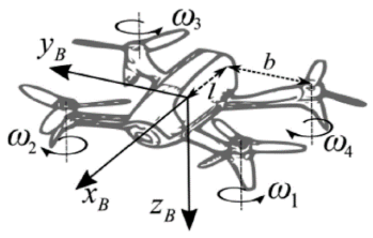


Figure 9.7: Drone geometry [28]

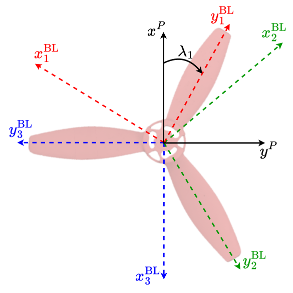


Figure 9.8: Blade coordinate frame.

Then, the propeller linear velocity is translated to the blade coordinate frame (BL), which rotates with the respective blade, as can be seen in Figure 9.8. The angle of the blade with the propeller x-axis is λ_j and it is used in Equation 9.15 for the coordinate frame transformation. As can be observed, a minus sign precedes the transformation matrix because the airspeed vector is opposite to the displacement direction. It is assumed that this value of airspeed, which depends on the vehicle linear and angular velocities, does not depend on the position along the blade.

$$\vec{V}_{A_{ijk_1}}^{BL} = \vec{V}_{A_{ij}}^{BL} = -\begin{bmatrix} \sin\lambda_j & -\cos\lambda_j & 0 \\ \cos\lambda_j & \sin\lambda_j & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{V}_i^P \tag{9.15}$$

Second, the component of the velocity due to the rotation of the propeller is the product of the distance of the blade section centroid to the centre of rotation (r_{ijk}) and the rotational velocity of the propeller (ω_i), as can be seen in Equation 9.16. The main benefit of the chosen blade coordinate frame is that the component of the velocity due to the propeller rotation only exists along the x-axis. Finally, ζ_i is a variable which acquires a value of 1 if the i^{th} propeller is rotating clockwise and -1 if it is rotating counter-clockwise.

$$\vec{V}_{A_{ijk_2}}^{BL}(r_{ijk}) = \begin{bmatrix} \zeta_i \omega_i r_{ijk} \\ 0 \\ 0 \end{bmatrix} \tag{9.16}$$

Third, the computation of the induced velocity. There exist multiple approaches in literature for computing this velocity field across the rotor disk, most of the them based on estimates and empirical tests. Some previous research [251] assumes ideal propeller geometry and they consider a constant uniform induced velocity along the propeller, which is mostly not the case in forward flight. The approach followed for the present research is the same one used in [252] and that is thoroughly explained in [29], which combines an initial uniform inflow estimation for the complete propeller with local (blade section) linear inflow model corrections.

For the computation of the uniform induced velocity (${v_0}^1$), the Glauert formula presented in Equation 9.19 is derived from the combination of the mass flow and the propeller thrust equations, shown in Equation 9.17 and Equation 9.18, respectively. According to the principles of momentum and energy conservation, the far wake velocity equals the airspeed before the rotor plus two times the induced velocity [29], leading to a change in velocity across the rotor of $\Delta V = 2v_0$.

[1]Since it will be constantly referred to the same single propeller, the subscript i to denote a specific propeller is dropped for the rest of this work for readability purposes.

$$\dot{m} = \rho \pi R^2 V_R \tag{9.17}$$ $$T = \dot{m}\Delta V = \dot{m}(V_{A_i} + 2v_0 - V_{A_i}) = 2\dot{m}v_0 \tag{9.18}$$

$$v_0 = \frac{T}{2\rho\pi R^2 V_R} \tag{9.19}$$

Given that the airspeed at the rotor (V_R) equals the propeller airspeed plus the induced velocity along the z-axis direction, it can be expressed as shown in Equation 9.20 using the translational velocity found in Equation 9.13. Again, there is a minus before the velocity component in the z-direction because the airspeed of the propeller is a vector in the opposite direction to its translational velocity. The final Glauert equation does not have a closed form, so the induced velocity can be computed using an iterative optimisation technique, such as Nelder-Mead. For the UE4 simulation, since the induced velocity needs to be computed in every time step after the blade damage, a tailored gradient-descent (1D first order derivative) algorithm was implemented, an approach discussed in Appendix A.

$$V_R = \sqrt{V_x^P V_x^P + V_y^P V_y^P + (-V_z^P + v_0)^2} \tag{9.20}$$

Once the uniform inflow velocity is computed, it can be used as the basis for local corrections for the computation of the linear inflow model. There are multiple estimation models for the computation of the linear inflow, such as those proposed by Howlett [253], Pitt & Peters [254] and White & Blake [255]. However, for the present research, the model developed by Drees [256] will be used since it is one of the best representations when compared to empirical data [29]. Equation 9.21 models the induced velocity using the uniform inflow as basis and modifying it with the k_x and k_y weighting factors, which are computed in Equation 9.22 and Equation 9.23, respectively.

$$v_i(\cdot) = v_0(1 + k_x r\cos\psi + k_y r\sin\psi) \tag{9.21}$$

$$k_x = \frac{4}{3}\frac{(1 - \cos \mathrm{X} - 1.8\mu_x^2)}{\sin \mathrm{X}} \tag{9.22}$$ $$k_y = -2\mu_x \tag{9.23}$$

X is the wake skew angle or the angle that the wake creates with respect to the z-axis of the propeller. It is illustrated in Figure 9.9 and it is computed using the propeller airspeed and the uniform induced velocity, as can be seen in Equation 9.24. Furthermore, μ_x is the tip speed ratio or advanced ratio and it is defined as the airspeed projected on the x-y plane in the propeller coordinate frame (the velocity parallel to the plane of the rotor) normalised with the blade length and propeller rotational velocity (Equation 9.25). The resulting inflow can be observed in Figure 9.10, which shows how the induced velocity changes per blade section with respect to its distance to the centre of rotation and the azimuth angle.

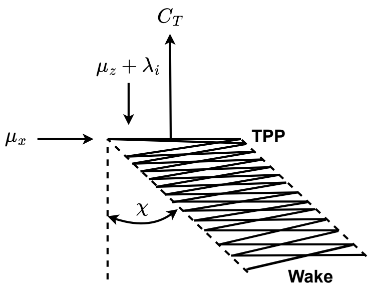


Figure 9.9: Illustration of the wake skew angle [29]

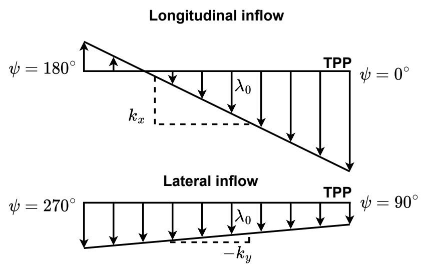


Figure 9.10: Linear inflow model [29]

$$\tan \mathrm{X} = \frac{\sqrt{V_x^P V_x^P + V_y^P V_y^P}}{v_0 - V_z^P} \tag{9.24}$$ $$\mu_x = \frac{\sqrt{V_x^P V_x^P + V_y^P V_y^P}}{\omega_i R} \tag{9.25}$$

Having computed the 3 components of the blade section airspeed for the lift and drag equations, they are summed in Equation 9.26.

$$\vec{V}^{BL}_{A_{jk}}(\cdot) = \vec{V}^{BL}_{A_{jk_1}} + \vec{V}^{BL}_{A_{jk_2}}(r_{jk}) + \begin{bmatrix} 0 & 0 & v_{i_{jk}}(\cdot) \end{bmatrix}^T \tag{9.26}$$

Given the airspeed, the angle of attack seen by the blade can be obtained from Equation 9.27. Here, θ_{jk} is the average twist of the blade section and it is a linear function of the distance from the centre of rotation (Equation 9.28, where θ_{tw} is the blade twist rate per radius of the rotor and θ_0 is the blade twist at the blade root). The closer to the root, the higher the twist in order to compensate for the lower tangential velocity from the propeller rotation. When the tangential velocity decreases, the velocity along the propeller z-axis has a higher impact on the definition of the angle of attack. In the case of forward flight, that causes a reduction of the angle of attack at the root.

$$\alpha_{jk}(\cdot) = \theta_{jk}(r_{jk}) - \arctan \frac{V^{BL}_{A_{jk_z}}(\cdot)}{V^{BL}_{A_{jk_x}}(\cdot)} \tag{9.27}$$

$$\theta_{jk}(r_{jk}) = \theta_0 - r_{jk}\theta_{tw} \tag{9.28}$$

Finally, from the blade section lift and drag contributions determined in Equation 9.11 and Equation 9.12, it is possible to determine the generated thrust (ΔT) and torque (ΔQ) by each blade section with Equation 9.29 and Equation 9.30, respectively[2]. Here, ϕ_{jk} is the blade section flow angle and it is defined as the difference between the local twist angle θ_{jk} and the local angle of attack α_{jk}, as can be seen in Equation 9.31.

$$\Delta T = \Delta L \cos\phi - \Delta D \sin\phi \tag{9.29}$$

$$\Delta Q = r\Delta L \sin\phi + r\Delta D \cos\phi \tag{9.30}$$

$$\phi_{jk}(\cdot) = \theta_{jk}(r_{jk}) - \alpha_{jk}(\cdot) \tag{9.31}$$

For the present research, Blade Element Theory can be used to compute the forces and moments of the missing blade sections upon blade damage. Unfortunately, the lift and drag coefficients required for the computation of the lift (Equation 9.11) and drag (Equation 9.12) forces are unknown for the Bebop 2 propellers. Section 9.2.2 will show how these coefficients are identified.

9.2.2. Airfoil lift and drag coefficients identification

Unfortunately, for most commercial drones, an aerodynamic model of the propeller's blade lift and drag does not exist. Previous literature [257] has taken a Hybrid Blade Element Momentum Theory (BEMT) approach for the computation of the induced velocity and aerodynamic coefficients. This approach is based on equating the thrust derived from BET (blade geometry) and the thrust derived from Momentum Theory (interacting flow characteristics). A two step method is described in which first, the induced velocity is estimated and then, the aerodynamic coefficients. However, this method has two main drawbacks. First, it does not explain how the induced velocity is computed without having previously computed the aerodynamic parameters, since they are required for the computation of the lift and the drag of each blade section in the computation of the thrust from BET. Second, the described optimisation method takes between 2 to 10 hours with only 20 data points. Such long computation times renders the simulation of blade damage scenarios in which the induced velocity must be computed for every time step, unfeasible. This section will thoroughly explain an alternative optimisation approach for identifying the lift and drag coefficient curves with much shorter computation times.

The department of Control & Simulation at Delft University and Technology developed in 2019 a gray-box aerodynamic model of the Bebop 2 drone through wind tunnel experiments [244]. With this model it is possible to retrieve the thrust and torque of each propeller giving as input the vehicle body linear and angular velocities, as well as the propeller rotational velocity and information about the drone geometry (R, and l and b from Equation 9.14).

Given the same input values, the thrust and torque computed with this gray-box model should be the same as those obtained from the sum of the blade sections' moment and forces contributions following Blade Element Theory (BET). Hence, the gray-box model and BET outputs are combined in

[2]Since every term of these equations is meant for a particular blade section, the subscripts jk have been left out here to enhance readability. For the same reason, the variables each of the terms are a function of have also been removed, since they are all a function of (ψ_{jk}, r_{jk}).

order to solve a constrained minimisation problem that finds the lift and drag coefficient functions. For that purpose, the C_l and C_d coefficients are modelled as polynomials of m and n degree as a function of the angle of attack, as can be seen in Equation 9.32 and Equation 9.33, respectively.

$$C_{l_{jk}} = x_0 + x_1\alpha_{jk} + x_2\alpha_{jk}^2 + \cdots + x_m\alpha_{jk}^m \quad (9.32) \qquad C_{d_{jk}} = y_0 + y_1\alpha_{jk} + y_2\alpha_{jk}^2 + \cdots + y_n\alpha_{jk}^n \quad (9.33)$$

In the case of the BET thrust and torque, they can be computed with Equation 9.34 and Equation 9.35 using these lift and drag coefficient polynomials. Again, the ζ parameter represents the direction of rotation. If from a top-down view the propeller is rotating clockwise, then ζ will be positive, otherwise it will be negative.

$$\begin{aligned} T &= \left(\sum_j^{n_b}\sum_k^{n_{bs}} \frac{1}{2}\rho C_{l_{jk}} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} dr\right) - \left(\sum_j^{n_b}\sum_k^{n_{bs}} \frac{1}{2}\rho C_{d_{jk}} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} dr\right) \\ &= \frac{1}{2}\rho dr\left[\left(\sum_o^m x_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \cos\phi_{jk}\right) - \left(\sum_o^n y_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \sin\phi_{jk}\right)\right] \end{aligned} \tag{9.34}$$

$$Q = -\zeta\frac{1}{2}\rho dr\left[\left(\sum_o^m x_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk}\right) + \left(\sum_o^n y_o \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk}\right)\right] \tag{9.35}$$

With the previous definitions of the thrust and the torque, it is possible to create a system of the $\vec{A}\vec{x} = \vec{b}$ form. Each pair of rows of the A matrix and the b vector corresponds to the thrust and torque of a data point from the BET and gray-box aerodynamic models, respectively. A data point refers to a set of conditions ($\vec{V}^B$, $\vec{\Omega}$, ω) that are provided as input to both models. Equation 9.36 to Equation 9.42 show the different components of the system with q data points. The underscore form $(.)_x$ is referring to the $(.)$ information of data point x.

$$l_1^o = \sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} \quad (9.36) \qquad l_2^o = -\sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} \quad (9.37)$$

$$l_3^o = -\zeta\sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \sin\phi_{jk} \quad (9.38) \qquad l_4^o = -\zeta\sum_j^{n_b}\sum_k^{n_{bs}} \alpha_{jk}^o r_{jk} c_{jk} V_{A_{jk}}^2 \cos\phi_{jk} \quad (9.39)$$

$$\vec{A} = \frac{1}{2}\rho dr \begin{bmatrix} (l_1^0)_1 & (l_1^1)_1 & \cdots & (l_1^m)_1 & (l_2^0)_1 & (l_2^1)_1 & \cdots & (l_2^n)_1 \\ (l_3^0)_1 & (l_3^1)_1 & \cdots & (l_3^m)_1 & (l_4^0)_1 & (l_4^1)_1 & \cdots & (l_4^n)_1 \\ (l_1^0)_2 & (l_1^1)_2 & \cdots & (l_1^m)_2 & (l_2^0)_2 & (l_2^1)_2 & \cdots & (l_2^n)_2 \\ (l_3^0)_2 & (l_3^1)_2 & \cdots & (l_3^m)_2 & (l_4^0)_2 & (l_4^1)_2 & \cdots & (l_4^n)_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ (l_1^0)_q & (l_1^1)_q & \cdots & (l_1^m)_q & (l_2^0)_q & (l_2^1)_q & \cdots & (l_2^n)_q \\ (l_3^0)_q & (l_3^1)_q & \cdots & (l_3^m)_q & (l_4^0)_q & (l_4^1)_q & \cdots & (l_4^n)_q \end{bmatrix} \tag{9.40}$$

$$\vec{x} = \begin{bmatrix} x_0 & x_1 & \cdots & x_m & y_0 & y_1 & \cdots & y_n \end{bmatrix}^T \tag{9.41}$$

$$\vec{b} = \begin{bmatrix} (T)_1 & (Q)_1 & (T)_2 & (Q)_2 & \cdots & (T)_q & (Q)_q \end{bmatrix}^T \tag{9.42}$$

The gray-box aerodynamic model does not make a distinction between different propeller azimuth angles. Therefore, the output of the BET model, namely the $\vec{A}$ matrix, has to be averaged over a rotation of the propeller, integrating over the azimuth angle [257]. Equation 9.43 shows how this is done for l_1^0; the same procedure can be applied to l_2^0, l_3^0 and l_4^0. In practice, instead of integrating, a specific discrete number of rotation angles are chosen, and their contributions to the $\vec{A}$ matrix are computed and averaged.

$$l_1^o = \frac{1}{2\pi}\int_{\psi=0}^{2\pi}\sum_{j}^{n_b}\sum_{k}^{n_{bs}}\alpha_{jk}^o(\cdot)c_{jk}(r_{jk})V_{A_{jk}}^2(\cdot)\cos\phi_{jk}(\cdot)\,d\psi \tag{9.43}$$

The shown system of equations could be solved with simple Least Squares methods, such as Ordinary Least Squares, Weighted Least Squares or Generalized Least Squares. However, the implementation of these methods does not guarantee that the lift and drag curves with respect to the angle of attack have their characteristic recognisable shape, namely that the lift curve increases until the critical angle of attack when it starts stalling and that the drag curve has a parabolic shape with increasing drag as the angle of attack increases. These methods will try to fit the data points in order to minimize the error between the gray-box aerodynamic and BET models, even though the generated curves have no physical sense. For instance, in the present research it has not been considered the aerodynamic interference between blades, between propellers, and between propellers and the vehicle body. These assumptions are expected to create some errors. The aforementioned approximation methods will aim at reducing those nonlinear errors by fitting complex curves which are very different than the lift and drag coefficient curves that the identification aims to find.

Hence, a constrained optimisation problem is posed in order to include airfoil aerodynamic knowledge in the solver. Given that there is a large difference in scale between the thrust and torque values, in the order of 10^2, the Normalised Root Mean Squared Error (NRMSE) is used as objective function (Equation 9.44 and Equation 9.45), where the standard deviation of the aerodynamic gray-model output (σ_b; the standard deviation of the observations) is used as normalisation factor. Instead of computing the error for the thrust and torque equations together and computing the standard deviation of the complete $\vec{b}$ vector, their errors and their respective observations' standard deviations were computed separately. The figure that the optimisation function aims at minimising is the averaged thrust and torque NRMSE, as can be seen in Equation 9.46. Hence, the objective function for the computation of the aerodynamic parameters $\vec{x}^*$ is defined in Equation 9.47.

$$\vec{\epsilon} = \vec{b} - \vec{A}\vec{x} \tag{9.44}$$

$$\text{NRMSE} = \frac{\sqrt{\frac{\vec{\epsilon}^T\cdot\vec{\epsilon}}{N}}}{\sigma_b} \tag{9.45}$$

$$\text{NRMSE}_{\text{total}} = \frac{\text{NRMSE}_T + \text{NRMSE}_Q}{2} \tag{9.46}$$

$$\vec{x}^* = \arg\min_{\vec{x}} \text{NRMSE}_{\text{total}} \tag{9.47}$$

Furthermore, the following lenient constraints were used to achieve the recognisable shape of the lift and drag coefficient curves:

1. The maximum lift coefficient can not be higher than 5 within the angle of attack range of -30 to 30 degrees: $C_l(\alpha) < 5, \quad \forall\alpha \in [-30°, 30°]$.
2. The lift coefficient curve should have a negative slope within the angle of attack range of 25 to 30 degrees: $dC_l(\alpha)/d\alpha < 0, \quad \forall\alpha \in [25°, 30°]$.
3. The lift coefficient curve should have a positive slope within the angle of attack range of 0 to 7 degrees: $dC_l(\alpha)/d\alpha > 0, \quad \forall\alpha \in [0°, 7°]$.
4. The lift coefficient curve should intersect the angle of attack axis within the angle of attack range of -10 to 10 degrees: $\min C_l(\alpha) < 0, \quad \forall\alpha \in [-10°, 10°]$.
5. The drag coefficient curve can not be negative within the angle of attack range of -30 to 30 degrees: $C_d(\alpha) > 0, \quad \forall\alpha \in [-30°, 30°]$.

For the declaration of these constraints, matrix $\vec{C}(\vec{\alpha}) \in R^{s\times(mn)}$ is created with the range of angles of attack mentioned in each constraint definition. The matrix has the same number of rows as integer angles within the constraint range, namely $s = \alpha_{\text{max}} - \alpha_{\text{min}} + 1$ for $\alpha \in [\alpha_{\text{min}}, \alpha_{\text{max}}]$; each row corresponds to an angle of attack. The number of columns equals the length of the parameter vector $\vec{x}$. Its input is an angle of attack vector which is a function of the α_{min} and α_{max}, as can be deduced from its definition in Equation 9.48 and Equation 9.49.

$$\vec{\alpha}(\alpha_{\text{min}}, \alpha_{\text{max}}) = \begin{bmatrix}\alpha_0 & \alpha_1 & \cdots & \alpha_\eta\end{bmatrix}^T \quad (9.48) \qquad \alpha_\eta = \alpha_{\text{min}} + \eta, \quad \eta = 0, \ldots, s-1 \quad (9.49)$$

Since a constraint regarding the lift coefficient curve does not require information about the drag coefficient parameters, the last n columns will be full of zeros, as can be seen in Equation 9.50 for constraints 1 and 4. In the case of the drag constraint (constraint 5), it would be the first m columns that would be full of zeros, as shown in Equation 9.51. Constraints 2 and 3 impose a limit in the gradient of the curve, so the derivative of $\vec{C}_{C_l}$ with respect to the angle of attack is taken in Equation 9.52.

$$\vec{C}_{C_l}(\vec{\alpha}) = \left[\begin{array}{ccccc|c} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n & \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^n & \vec{0}_{s\times m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \\ 1 & \alpha_{s-1} & \alpha_{s-1}^2 & \cdots & \alpha_{s-1}^n & \end{array}\right] \quad (9.50) \qquad \vec{C}_{C_d}(\vec{\alpha}) = \left[\begin{array}{c|ccccc} & 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n \\ \vec{0}_{s\times n} & 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^n \\ & \vdots & \vdots & \vdots & \cdots & \vdots \\ & 1 & \alpha_{s-1} & \alpha_{s-1}^2 & \cdots & \alpha_{s-1}^n \end{array}\right] \quad (9.51)$$

$$\frac{d\vec{C}_{C_l}(\vec{\alpha})}{d\alpha} = \left[\begin{array}{ccccc|c} 0 & 1 & 2\alpha_0 & \cdots & m\alpha_0^{m-1} & \\ 0 & 1 & 2\alpha_1 & \cdots & m\alpha_1^{m-1} & \vec{0}_{s\times m} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \\ 0 & 1 & 2\alpha_{s-1} & \cdots & m\alpha_{s-1}^{m-1} & \end{array}\right] \quad (9.52)$$

The $\vec{C}$ matrix or its derivative is multiplied with the parameter vector $\vec{x}$ and the maximum or minimum value from the output is taken for the definition of the inequality constraints. As a result, the following constrained optimisation problem is posed:

$$\begin{aligned} \min_{\vec{x}} \quad & \text{NRMSE}_{\text{total}} \\ \text{s.t.} \quad & \max\left(\vec{C}_{C_l}(\vec{\alpha})\vec{x}\right) - 5 < 0, \ (\alpha_{\text{min}}, \alpha_{\text{max}}) = (-30, 30) \\ & \max\left(\frac{d\vec{C}_{C_l}(\vec{\alpha})}{d\alpha}\vec{x}\right) < 0, \ (\alpha_{\text{min}}, \alpha_{\text{max}}) = (25, 30) \\ & \min\left(\frac{d\vec{C}_{C_l}(\vec{\alpha})}{d\alpha}\vec{x}\right) > 0, \ (\alpha_{\text{min}}, \alpha_{\text{max}}) = (0, 7) \\ & \min\left(\vec{C}_{C_l}(\vec{\alpha})\vec{x}\right) < 0, \ (\alpha_{\text{min}}, \alpha_{\text{max}}) = (-10, 10) \\ & \min\left(\vec{C}_{C_d}(\vec{\alpha})\vec{x}\right) > 0, \ (\alpha_{\text{min}}, \alpha_{\text{max}}) = (-30, 30) \end{aligned} \quad (9.53)$$

Figure B.3 visually illustrates the lift and drag coefficient identification with a flowchart. Now that the BET model and lift and drag coefficients identification method have been discussed, the next step will be the identification of the C_l/α and C_d/α curves for the Bebop 2 drone in subsection 9.2.3. This will allow the computation of the changed aerodynamic moments and forces caused by blade damage using the developed BET model in subsection 9.2.4.

9.2.3. Bebop 2 airfoil lift and drag coefficients identification

The identification method for the lift and drag coefficients presented in subsection 9.2.2 was implemented for the Bebop 2 drone, whose vehicle and propeller geometry is summarised in Table 9.1. Here, n_b stands for the number of blades per propeller. Besides that, the geometrical parameters of the propeller can be visualised in Figure 9.11.

In subsection 9.2.2 it was mentioned that the aerodynamic gray-box and BET models require as input for each data point a set of conditions ($\vec{V}^B$, $\vec{\Omega}$, ω) beyond the drone geometry. The range of those input conditions is explained next:

Table 9.1: Geometrical properties of the Bebop 2 drone and propeller.

Drone geometry		Propeller geometry							
b	l	c_r	c_c	c_t	h_1 & h_2	R	n_b	θ_0	θ_{tw}
(mm)	(mm)	(mm)	(mm)	(mm)	(mm)	(mm)	(-)	(°)	(°/mm)
115	87.5	13	20	8	32	75	3	27	0.29

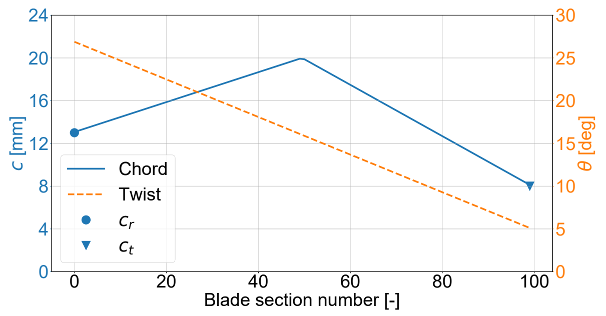


Figure 9.11: Blade sections' chord and twist values from the root (left) to the tip (right).

1. Appendix C discusses different drone linear velocity ($\vec{V}^B$) input generation schemes and arguments the choice outlined next. The drone linear velocity is constrained in the x-z plane (V_y^B=0). Its value in the z-direction (V_z^B) is sampled from a uniform distribution over the closed interval [-2, -0.5] m/s for every data point, avoiding positive velocities in the z-axis that could cause nonlinear behaviour that was not accounted for by the aerodynamic gray-box model. An example would be Vortex Ring State (VRS), which is the aerodynamic condition that causes a severe loss of lift when the vehicle descends vertically over its own wake.

   Furthermore, the value of the drone linear velocity in the x-direction (V_x^B) is also sampled from a uniform distribution, in this case over the closed interval [-3, 3] m/s for every data point. As a result, for the identification of the lift and drag curves only scenarios in which the drone is ascending, flying forward or backwards are considered.

2. The angular velocity of the drone is always zero ($\vec{\Omega} = \vec{0}$).

3. The rotation velocity of the propeller (ω) is sampled from a uniform distribution over the closed interval [300, 1256] rad/s for every data point.

Furthermore, the drag and lift curves were approximated with second degree polynomials (m and n) and the integral with respect to the azimuth angle presented in Equation 9.43 was approximated with 10 discrete equally spaced azimuth angles starting at 0°, namely [0°, 36°, 72°, ..., 288°, 324°]. The constrained optimisation method is posed using Python's Scipy package with the trust-region interior point method ("trust-constr") [258] solver. Additionally, the number of blade sections (n_{bs}) and the number of data points used for the identification (q) were determined from the hyper-parameter (q-n_{bs}) selection presented in Appendix D, study which lead to 100 blade sections and 16,000 data points.

Using the aforementioned chosen hyper-parameters, the data gathering and optimisation took 10,707 seconds (2 hours and 58 minutes) in a consumer laptop with an Intel Core i7-9750H CPU running Python 3.7, including 41 solver iterations. It was possible to identify the coefficients of the polynomials that define the lift and drag curves with respect to the angle of attack, which are shown in Equation 9.54 and Equation 9.55. Figure 9.12 and Figure 9.13 show the value of the lift and drag coefficients as a function of the angle of attack.

$$C_l = 0.24 + 5.15\alpha - 12.25\alpha^2 \qquad (9.54) \qquad C_d = 0.0092 - 0.79\alpha + 15.13\alpha^2 \qquad (9.55)$$

As can be observed from the lift and drag coefficient plots, only 1 of the constraints of the minimisation problem outlined at the end of subsection 9.2.2 is limiting in the solution, namely that the drag

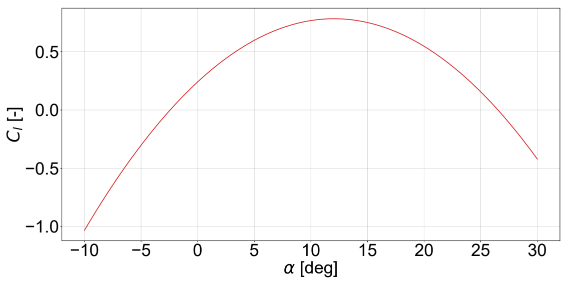


Figure 9.12: Identified airfoil lift coefficient curve with respect to the angle of attack from the aerodynamic gray-box model [28] data. Illustration of Eq. (9.54).

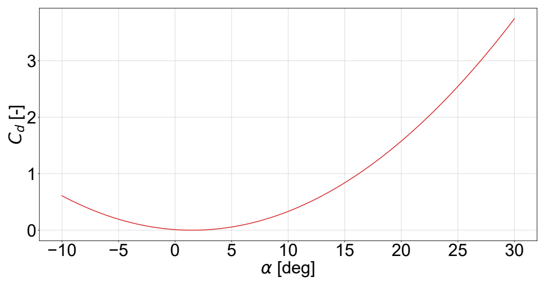


Figure 9.13: Identified airfoil drag coefficient curve with respect to the angle of attack from the aerodynamic gray-box model [28] data. Illustration of Eq. (9.55).

coefficient cannot be negative (the 5th constraint). The first constraint limited the lift coefficient to a value lower than 5 [-], which in this case is not larger than 0.8 [-]. The second and third constraints required the lift coefficient slope to be positive between 0° and 7°, whereas it should be negative when higher than 25°. It can be seen that the lift coefficient curve attains its maximum at around 13°, away from the aforementioned angle of attack ranges. The fourth constraint required the lift coefficient curve to cross the x-axis between -10° and 10°, and it crossed it at around -3°.

To validate the identified aerodynamic model it is verified whether the model residuals approximate zero mean white noise. This is done for the residuals from those data points used for the identification of the model, as well as for an additional validation dataset made of $(\vec{V}^B, \vec{\Omega}, \omega)$ input combinations not seen yet by the model. This validation dataset has 4,000 data points, which is 25% the size of the training dataset. Figure 9.14 and Figure 9.15 show the residuals of the identification dataset, whereas Figure 9.16 and Figure 9.17 of the validation dataset for the thrust and torque, respectively. Table 9.2 shows the mean of the curves, as well as their percentage relative to the average value of the thrust or torque. As can be observed, their values are low (below 1%), which indicates that the error of both, identification and validation datasets could be considered to be zero mean. Besides that, the thrust error is approximately three times lower than the torque error for the NRMSE metrics, which means that the fitted C_l and C_d curves approximate the thrust data better than the torque.

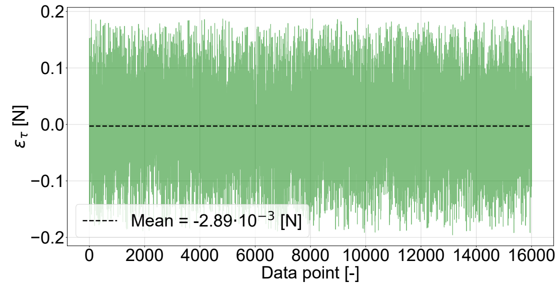


Figure 9.14: Identification data thrust error with a NRMSE$_\tau$ = $8.43 \cdot 10^{-2}$ [-].

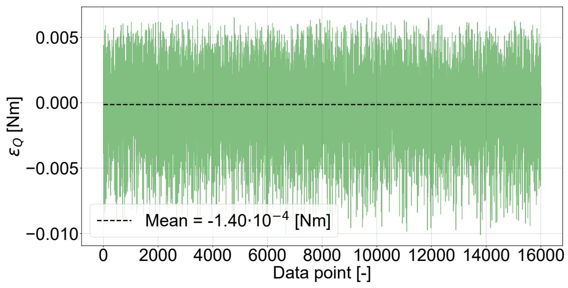


Figure 9.15: Identification data torque error with a NRMSE$_\tau$ = 0.26 [-].

Table 9.2: Error metrics results for the thrust and torque identification and validation datasets.

	Identification thrust	Identification torque	Validation thrust	Validation torque
Mean value	-2.89·10^{-3} [N]	-1.40·10^{-4} [Nm]	-1.56·10^{-3} [N]	-1.22·10^{-4} [Nm]
Mean percentage	-0.23 [%]	-0.97 [%]	-0.13 [%]	-0.86 [%]
NRMSE	8.43·10^{-2} [-]	0.26 [-]	8.30·10^{-2} [-]	0.25 [-]

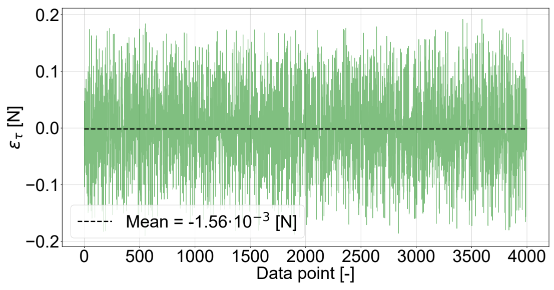


Figure 9.16: Validation data thrust error with a NRMSE$_\tau$ = $8.30 \cdot 10^{-2}$ [-].

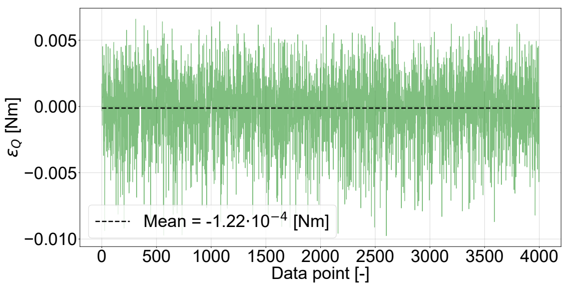


Figure 9.17: Validation data torque error with a NRMSE$_\tau$ = 0.25 [-].

Next, it is assessed whether the residual is uncorrelated. For that purpose, Figure 9.18 and Figure 9.19, as well as Figure 9.20 and Figure 9.21, show the normalised autocorrelation curves for the residuals with their 95% confidence bounds for the identification and the validation datasets, respectively. Almost all of the autocorrelations fall within the 95% confidence limits and there is no apparent pattern. A few lags fall slightly outside the bounds, but it is not enough to indicate non-randomness. Hence, the residuals can be considered to be white and uncorrelated.

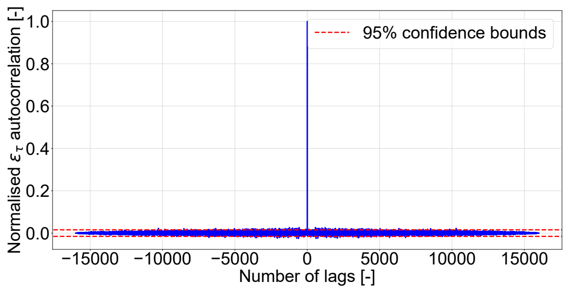


Figure 9.18: Identification thrust error normalised autocorrelation with 95% confidence bounds.

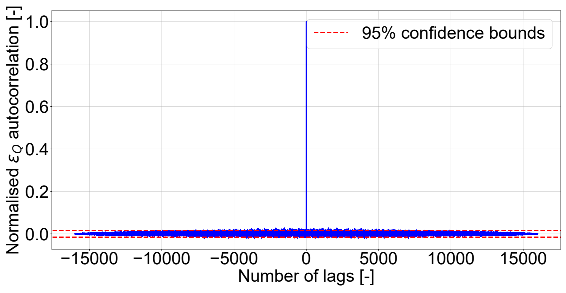


Figure 9.19: Identification torque error normalised autocorrelation with 95% confidence bounds.

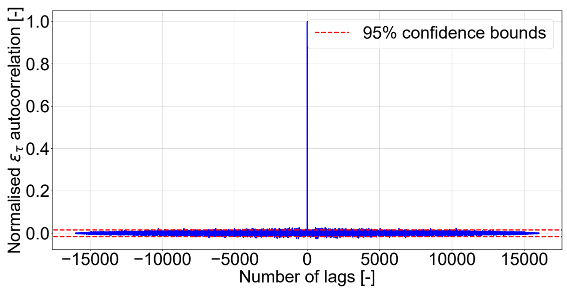


Figure 9.20: Validation thrust error normalised autocorrelation with 95% confidence bounds.

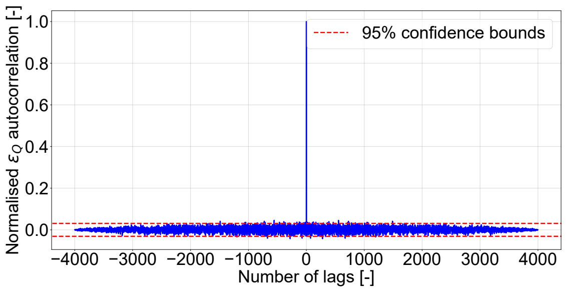


Figure 9.21: Validation torque error normalised autocorrelation with 95% confidence bounds.

With the identified lift and drag coefficient curves, the next step is the computation of the forces and moments caused due to the aerodynamic effects upon blade damage, which will be carried out in the next section.

9.2.4. Aerodynamic forces and moments computation

Once the percentage of blade damage to be simulated is known, the corresponding ”lost” blade sections are identified. For instance, if a 30% blade damage is considered in a blade composed of 100 sections, the 30 sections closest to the blade tip are the ones lost. Then, their forces and moments are added to later be subtracted from those of the nominal operating conditions obtained from the healthy blade model output. With the detailed BET explanation from subsection 9.2.1, the discussion of the computation of the aerodynamic forces and moments boils down to referencing to the right equations within the aforementioned section.

Equation 9.29 can be used for the computation of the thrust of a single blade section. The added thrust value of all the lost blade sections leads to $F^P_{a1_z}$, as given by Equation 9.56. ξ_{jk} is a boolean which has a value of 1 when the blade section is damaged and 0 when it is in its nominal state. Then, the moments about the propeller x- and y-axes are computed by decomposing the moment generated by the blade section thrust around the centre of rotation with Equation 9.57 and Equation 9.58.

$$F^P_{a1_z} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} \xi_{jk} \Delta T_{jk} \tag{9.56}$$

$$M^P_{a2_x} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} -\xi_{jk} \Delta T_{jk} r_{jk} \sin \lambda_j \tag{9.57}$$

$$M^P_{a2_y} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} \xi_{jk} \Delta T_{jk} r_{jk} \cos \lambda_j \tag{9.58}$$

Furthermore, Equation 9.30 can be used for the computation of the torque of each blade section and their integral leads to the moment about the z-axis, as shown in Equation 9.59. The blade section force in the rotor plane can be obtained by dividing the torque by the magnitude of the moment arm. Then, this force is decomposed in the x- and y-directions, as shown in Equation 9.60 and Equation 9.61, in order to obtain the last aerodynamic forces.

$$M^P_{a1_z} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} \xi_{jk} \Delta Q_{jk} \tag{9.59}$$

$$F^P_{a2_x} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} \xi_{jk} \frac{\Delta Q_{jk}}{r_{jk}} \cos\left(\lambda_j - \zeta \frac{\pi}{2}\right) \tag{9.60}$$

$$F^P_{a2_y} = \sum_{j=1}^{n_b} \sum_{k=1}^{n_{bs}} \xi_{jk} \frac{\Delta Q_{jk}}{r_{jk}} \sin\left(\lambda_j - \zeta \frac{\pi}{2}\right) \tag{9.61}$$

A flowchart visually illustrating the computation of these forces and moments can be seen in Figure B.4. Their effects in a simulated 0.25 [s] failure scenario, as well as when combined with the mass change effects, will be shown in the next section.

9.3. Results, assumptions and recommendations

With the outlined methodology in section 9.1 and section 9.2 now it is possible to combine the mass and aerodynamic related effects in order to compute the forces and moments that need to be added to the nominal physics model in order to account for the blade damage. Hence, subsection 9.3.1 discusses how the theory presented is translated to a damaged propeller simulation. Additionally, subsection 9.3.2 demonstrates the importance of including the linear induced velocity in the computations given the drastic modifications it brings to the angle of attack and the wrench results. Finally, subsection 9.3.3 discusses some assumptions that were taken in this chapter, as well as some recommendations for further work in the simulation of blade damage.

9.3.1. Bebop 2 mass and aerodynamic forces and moments

To observe the magnitude of the forces and moments caused by the change in mass, the front left Bebop 2 propeller is simulated as damaged with a loss of 20% of its length. It is rotating at 600 rad/s counterclockwise from a top-down view (ζ=-1) for 0.25 seconds. The attitude of the drone is such that the z-axis of the propeller and inertial coordinate frame coincide. Additionally, the drone is moving with a body linear velocity $\vec{V}^B = \begin{bmatrix}3 & 0 & -1\end{bmatrix}^T$ [m/s] and body angular velocity $\vec{\Omega} = \vec{0}$ rad/s. The Bebop 2 propeller has a total mass of 5.07 [g] and a mass per blade (without the central hub) of 1.11 [g].

Figure 9.22 and Figure 9.23 show the forces and moments caused by the change of mass for the aforementioned scenario, respectively. As can be observed, the forces and moments in the x- and y-directions have a oscillatory behaviour due to the propeller rotations. Additionally, since the drone propeller´s z-axis direction is aligned with its counterpart in the inertial frame, the gravity force coincides with the z-axis in the propeller coordinate frame, leading to a constant force in the z-direction and a zero moment about the z-axis.

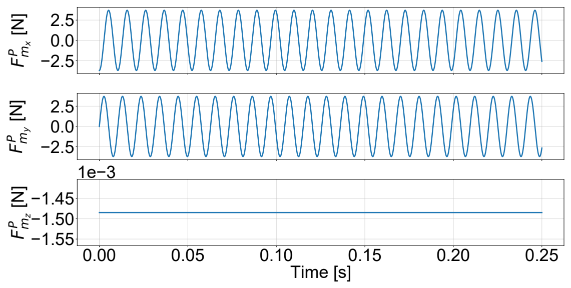


Figure 9.22: BET-simulated evolution of forces caused due to mass change upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

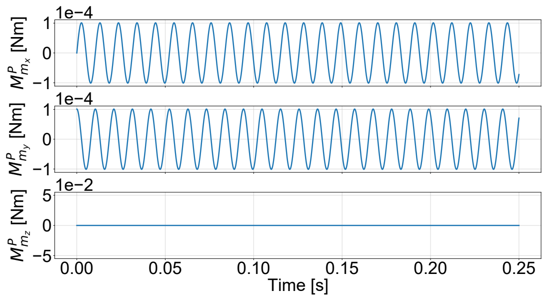


Figure 9.23: BET-simulated evolution of moments caused due to mass change upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Next, the aerodynamic forces and moments are also analysed for the same 0.25 s Bebop 2 propeller scenario, leading to the results illustrated in Figure 9.24 and Figure 9.25. As can be observed, the forces and moments around the y-axis are centred around the 0 datum, whereas the wrenches in the x- and z-direction are biased. The wrench in the x-direction is negatively biased because the lift and drag forces are the highest when the damaged blade is advancing and not retreating. Since the propeller is rotating counterclockwise, the thrust produced when the blade is advancing creates a negative moment around the x-axis and the force creating the torque points towards the negative x-direction. The oscillatory motion in the z-direction is is also due to the incoming air velocity from the vehicle linear and angular velocities, which causes the damaged blade incoming flow to be different when it is advancing than when it is retreating. In the case that the drone would be hovering, then the aerodynamic wrench in this direction would be constant and will have a value approximately equal to the observed bias, as can be seen in Figure 9.26 and Figure 9.27.

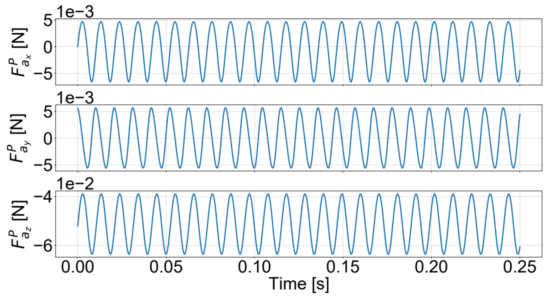


Figure 9.24: BET-simulated evolution of aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

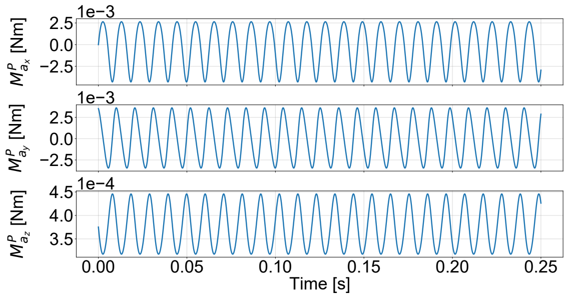


Figure 9.25: BET-simulated evolution of aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Furthermore, Figure 9.28 and Figure 9.29 show the mass and aerodynamic caused forces and moments super-imposed. The mass change effects are predominant in the x- and y-components of the force, whereas the aerodynamic effects are predominant in the force z-direction and in all moment directions.

Finally, the mass and aerodynamic effects around the propeller's centre of rotation are combined as shown by Equation 9.62 and Equation 9.63; quantities that will be used in the validation phase in the wind tunnel, as presented in section 9.4. Combining both effects for the discussed 0.25 s simulation

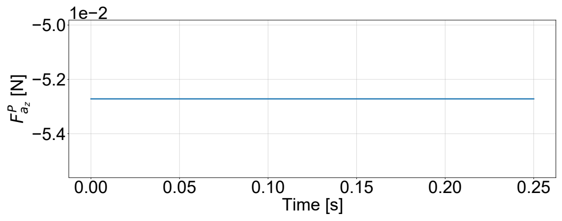


Figure 9.26: BET-simulated evolution of aerodynamic force in the z-direction during hover upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

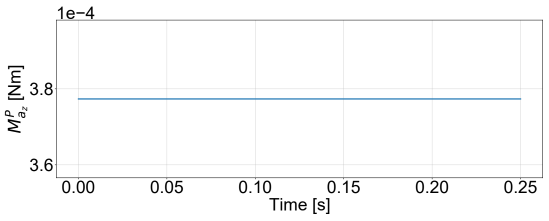


Figure 9.27: BET-simulated evolution of aerodynamic moment in the z-direction during hover upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

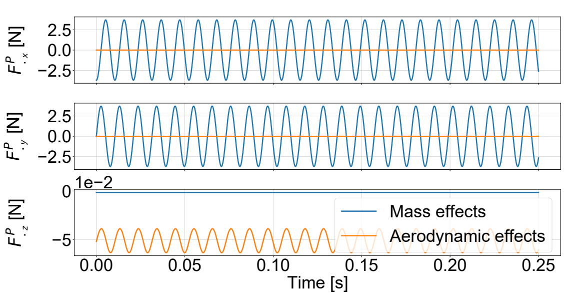


Figure 9.28: BET-simulated evolution of mass and aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

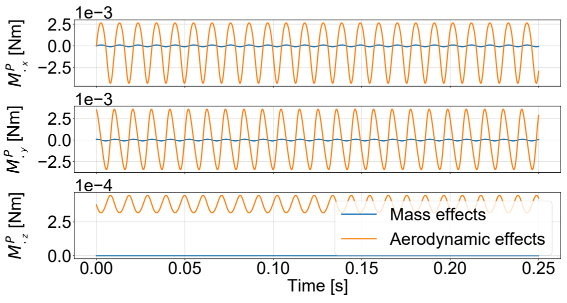


Figure 9.29: BET-simulated evolution of mass and aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

leads to Figure 9.30 and Figure 9.31. Previous literature is correct in focusing on the mass related centrifugal forces, since they are one to three order of magnitude greater than the rest. However, they ignore the effects outside the x-y plane in the propeller coordinate frame, especially the force in the z-direction which would create additional moments around the drone's CG. Although subtle, the oscillations in the moment signals could help in the identification of the blade damage, namely the detection of the affected propeller and the failure magnitude.

$$\Delta\vec{F}^P = \vec{F}^P_{m1} + \vec{F}^P_{m2} - \vec{F}^P_{a1} - \vec{F}^P_{a2} \tag{9.62}$$

$$\Delta\vec{M}^P = \vec{M}^P_{m} - \vec{M}^P_{a1} - \vec{M}^P_{a2} \tag{9.63}$$

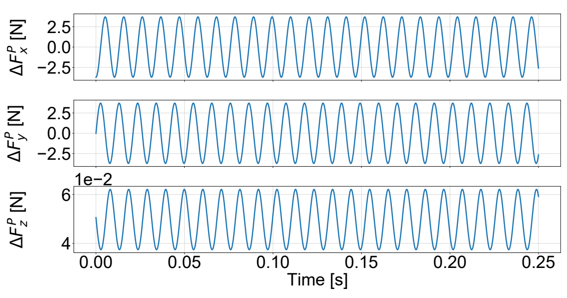


Figure 9.30: BET-simulated evolution of forces upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

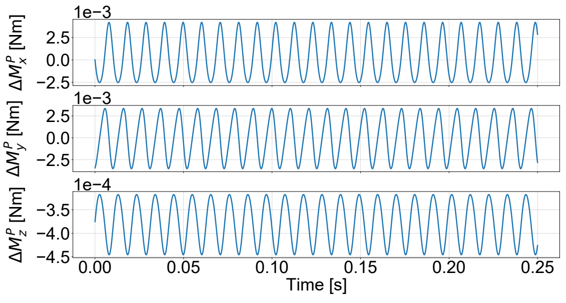


Figure 9.31: BET-simulated evolution of moments upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s.

Even though there is a different order of magnitude among the forces and the moments, all the signals are oscillatory. Figure 9.32 and Figure 9.33 show the upper and lower limit of these wrench oscillations for different degrees of blade damage, namely from 0% (intact blade) to 100% (complete blade loss). All forces and moments have their upper and lower limits symmetric with respect to the 0 datum, except the force and moment in the z-direction. ΔF^P_z moves in the positive direction with increasing blade damage — the higher the blade damage, the larger the thrust loss (positive thrust points down in the propeller coordinate system). ΔM^P_z moves in the negative direction with increasing blade damage — with a negative rotating propeller, the aerodynamic moments are positive before being subtracted for obtaining the total value.

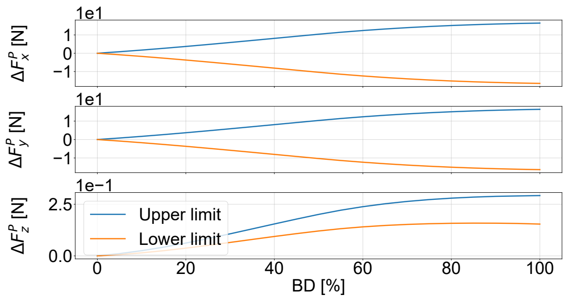


Figure 9.32: Upper and lower limits of the forces' oscillations for different degrees of BET-simulated blade damage.

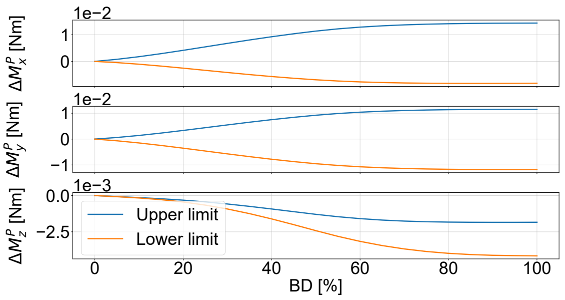


Figure 9.33: Upper and lower limits of the moments' oscillations for different degrees of BET-simulated blade damage.

The higher the blade damage, the higher the oscillations. However, depending on whether the dominating effect is mass or aerodynamic, the behaviour of those limits is different. The gradient of the upper and lower limits of ΔF_x^P can be observed in Figure 9.34; the gradient first increases and around 50% starts decaying. This is caused by the modelled double trapezoid Bebop 2 blade shape which can be observed in Figure 9.1 and whose geometrical parameters have been listed in Table 9.1. As the blade is progressively damaged from the tip (0% damage) to the location of the central chord (50% damage) the removed blade sections are progressively growing in size, causing constantly greater shifts of the centre of gravity and, hence, greater increments in the centrifugal force. When the damage reaches the maximum chord, the removed blade sections start to decrease in size, leading to smaller centre of gravity shifts and more slowly growing centrifugal forces.

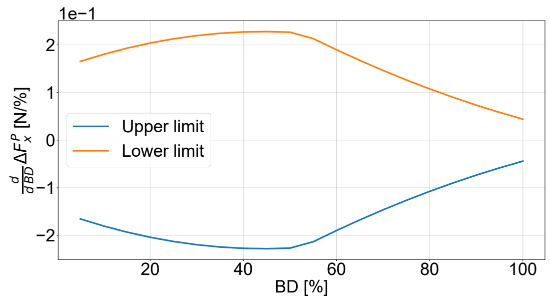


Figure 9.34: Gradient of the upper and lower limits of the ΔF_x^P oscillations with respect to different degrees of BET-simulated blade damage.

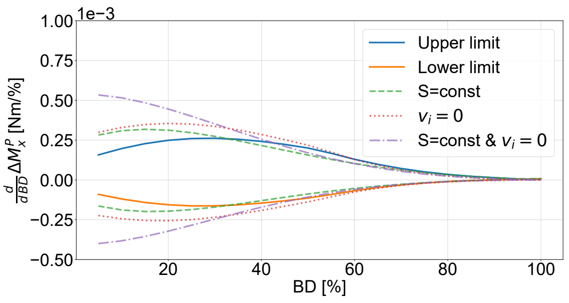


Figure 9.35: Gradient of the upper and lower limits of the ΔM_x^P oscillations with respect to different degrees of BET-simulated blade damage. The non solid lines represent scenarios in which the blade section area is constant and/or there is no induced velocity.

Figure 9.35 shows the gradients of the upper and lower oscillation limits of ΔM_x^P, an aerodynamic dominated wrench component. With a blade divided in 100 blade sections, these gradients approximately correspond to the contribution of each blade section to the moment. For example, the gradient at 20% blade damage corresponds to the contribution by the 19th blade section. As can be observed, the moment gradient increases in magnitude until approximately 30%. This initial increment is caused by the combined effect of the increasing blade section area when traversing the blade from the tip towards the central maximum chord and the induced velocity, which ultimately affects the angle of attack. When the blade section area is constant and the induced velocity is removed, the oscillations are only decaying in amplitude. However, when at least one of these two factors is still active, there is an increment in the gradient, as it can be seen in Figure 9.35.

When the 30% blade damaged is reached, the reducing distance from the blade section to the centre of rotation (r_{jk}) becomes the most influential factor in the further decay of the gradient. ΔM_x^P is proportional to r_{jk}^3 when considering that at high propeller rotational rates the airspeed is dominated by $V_{A_{jk_2}}$ (Equation 9.16), which is a function of r_{jk}, and the airspeed is squared in the computation of

ΔL_k and ΔD_k (Equation 9.11 and Equation 9.12). The lift and drag components are then used for the computation of the ΔT_{jk}, which is multiplied by the r_{jk} again to obtain the moment.

9.3.2. Importance of induced velocity

The importance of including the induced velocity in the aerodynamic calculations outlined in subsection 9.2.1 becomes clear when comparing Figure 9.24 and Figure 9.25 with Figure 9.36 and Figure 9.37. When the linear induced velocity model is included, the amplitude of the oscillations and the datum around which they oscillate are decreased in magnitude. This is attributed to the reduction in the angle of attack created by the introduction of v_i, a consequence which can be observed in Figure 9.38 and Figure 9.39. These figures show box plots of the angles of attack seen by each of the blade section from the blade root to the tip in the optimisation with 16,000 data points discussed in subsection 9.2.3 with and without the linear induced velocity model, respectively.

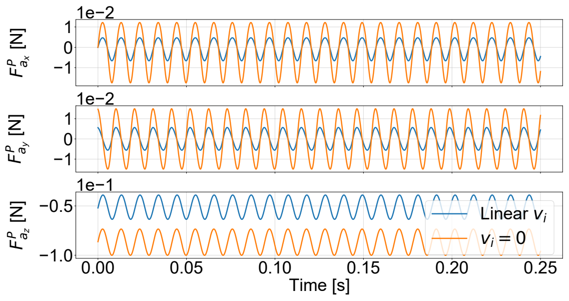


Figure 9.36: BET-simulated evolution of aerodynamic forces generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s with and without linear inflow model.

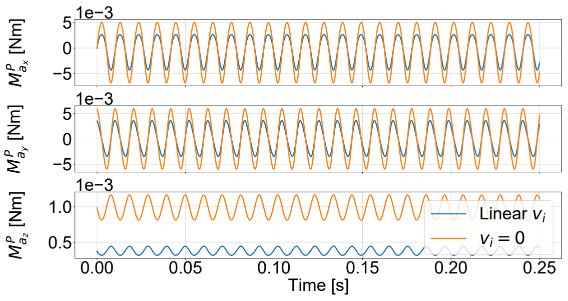


Figure 9.37: BET-simulated evolution of aerodynamic moments generated by lost blade sections upon 20% Bebop 2 blade damage for 0.25 s rotating at ω_0 = 600 rad/s with and without linear inflow model.

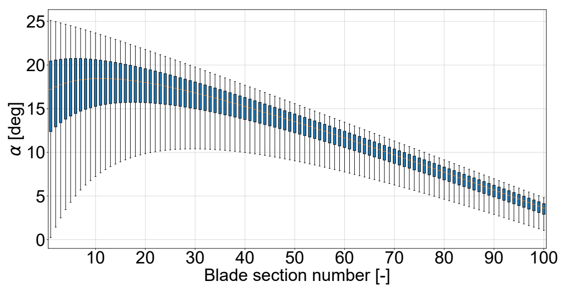


Figure 9.38: Box plot with the angles of attack seen by each BET-simulated blade section during 16,000 data point optimisation without induced velocity model. The inputs that shape each data point are taken from uniform distributions with the following value ranges: $\vec{V}^B_x$ =[-3,3] m/s, $\vec{V}^B_y$ =0 m/s, $\vec{V}^B_z$ =[-2.5,-0.5] m/s, $\vec{\Omega} = \vec{0}$ rad/s, ω =[300,1256] rad/s.

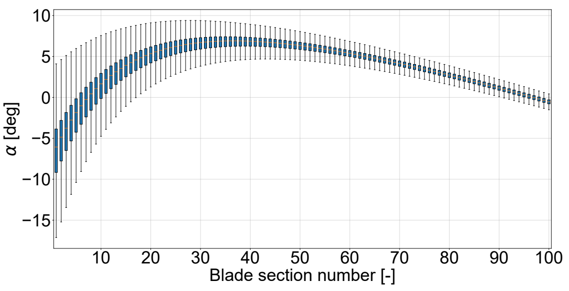


Figure 9.39: Box plot with the angles of attack seen by each BET-simulated blade section during 16,000 data point optimisation with linear inflow velocity model. The inputs that shape each data point are taken from uniform distributions with the following value ranges: $\vec{V}^B_x$ =[-3,3] m/s, $\vec{V}^B_y$ =0 m/s, $\vec{V}^B_z$ =[-2.5,-0.5] m/s, $\vec{\Omega} = \vec{0}$ rad/s, ω =[300,1256] rad/s.

From the angle of attack box plots (Figure 9.38 and Figure 9.39), the following additional observations can be made:

1. Apart from the general reduction in the angle of attack, the line that could be drawn from the blade sections' angle of attack medians becomes linear at a higher angle of attack when the induced velocity model is introduced. In contrast, when $v_i = 0$ the median line is linear for the largest part of the plot. Even though the v_i model is linear, it is important to remind the reader that this property applies in the x and y directions in the propeller reference frame, as can be observed in Figure 9.10, not along the blade radial direction.
2. Even though the twist of the blade goes from 25° to 5° from the root to the tip, the line of medians of Figure 9.38 has a lower value than the twist — especially close to the root — because the

distribution of the linear body velocity in the z-direction is biased towards negative values (the drone is flying upwards). In subsection 9.2.3 it was established that V_z^B has a value in the closed interval [-2,-0.5] m/s.

3. For both scenarios, the range of angles of attack is larger at the root due to the higher sensitivity to the vehicle's velocity; the blade section tangential velocity due to the propeller's rotation is lower at the root than at the tip.

The value of the induced velocity and its effect on the angle of attack can also be visualised in the propeller plane, as illustrated in Figure 9.40 and Figure 9.41 for the Bebop 2 propeller located on the front left of the vehicle. The propeller is moving towards the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise. The empty internal concentric circle represents the propeller hub, which is not an aerodynamic surface, resulting in an annulus heat map. Figure 9.41 reflects the same behaviour as in Figure 9.39 but in 2D, namely that the angle of attack rapidly increases close to the root until about 35% of the blade before it starts decaying more slowly towards the tip. The low angle of attack values close to the root in the direction of ψ =270° correspond to the retreating blade sections whose airspeed caused by the propeller rotation ($\vec{V}^{BL}_{A_{jk_2}}$) acts in opposite direction to the airspeed caused by the linear and angular displacement of the vehicle ($\vec{V}^{BL}_{A_{jk_1}}$). When the rotational speed is lowered to 300 rad/s, the stalled retreating blade sections become more apparent, as shown in Figure 9.42.

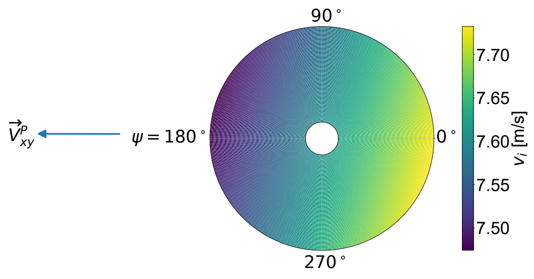


Figure 9.40: Heat map of the linear induced model velocity for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

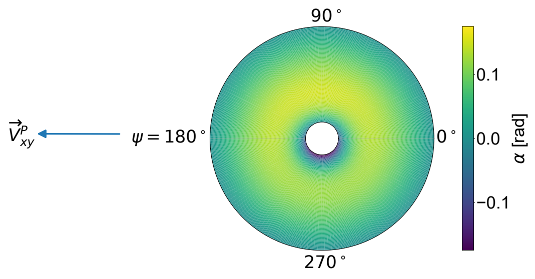


Figure 9.41: Heat map of the angle of attack for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s, including linear inflow model, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

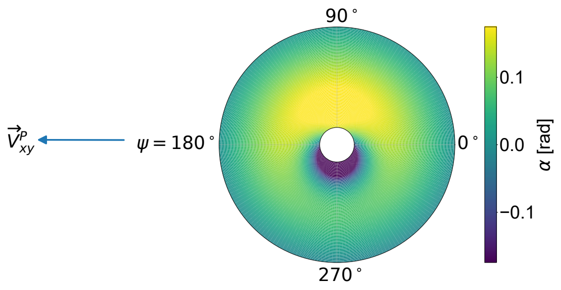


Figure 9.42: Heat map of the angle of attack for the BET-simulated front left Bebop 2 propeller rotating at 300 rad/s, including linear inflow model, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

Finally, Figure 9.43 and Figure 9.44 show the same induced velocity and angle of attack plots when a uniform induced flow is considered. As can be observed, the variations in both variables brought by the uniform model corrections are very small, practically unnoticeable in the angle of attack when comparing the results with Figure 9.41. While the uniform inflow model creates an induced velocity of 7.6 m/s across the complete blade, the linear inflow model creates an induced velocity that varies from 7.48 to 7.72 m/s. Even though literature has proven empirically that the linear inflow model is more accurate than the uniform counterpart [29], the difference can be considered negligible for propellers of small radius, as it is the case for most commercial drones such as the Bebop 2; a dependency on the blade radius which can be deduced from Equation 9.21. The effect of the linear model can be observed in the main rotor system of helicopters which have blade lengths higher than 1.5 metres.

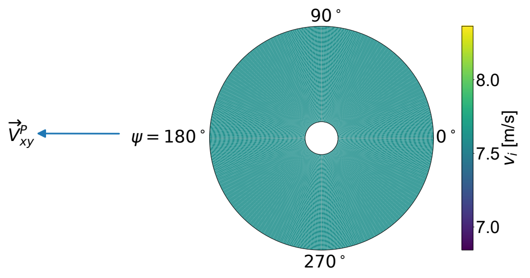


Figure 9.43: Heat map of the uniform induced model velocity for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

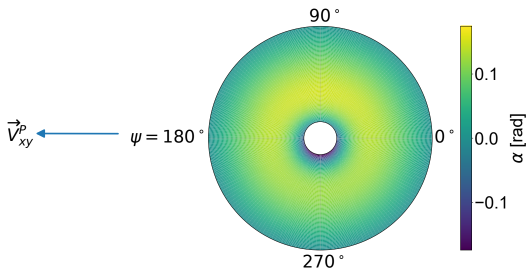


Figure 9.44: Heat map of the angle of attack for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s, including uniform inflow model, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

Hence, the additional computations required for the uniform model corrections could be ignored in the Unreal Engine 4 data collection simulation. However, it is essential to include the uniform induced model, as it has been shown that it can modify the vertical airspeed seen by the blade sections from 1 to 8.6 m/s at full propeller rotational speed (1256 rad/s) for the Bebop 2 example scenario (Figure 9.43). This effect can be visualised when comparing the angle of attack heat map when there is a uniform induced velocity, as it is the case in Figure 9.44, to the scenario when v_i=0 shown in Figure 9.45. The latter figure is a 2D representation of Figure 9.38. The importance of the uniform inflow model is especially noticeable at the retreating blade sections close to the root.

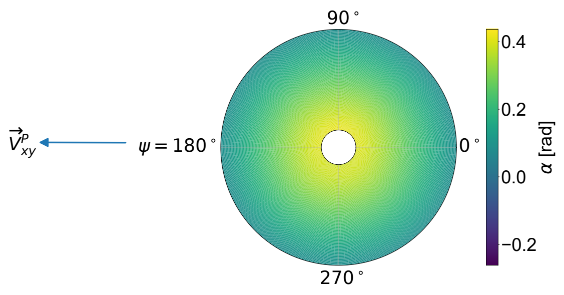


Figure 9.45: Heat map of the angle of attack for the BET-simulated front left Bebop 2 propeller rotating at 1256 rad/s, moving to the left with 3 m/s, out of the plane with 1 m/s and it is rotating counter-clockwise.

9.3.3. Assumptions and recommendations

For the development of the blade damage forces and moments model, as well as the lift and drag coefficient curves identification, the following assumptions were made:

1. The mass along the blade is homogeneous, meaning that the centre of gravity is at the centroid.
2. The Bebop 2 blades are simplified as two trapezoids connected by the long parallel side.
3. The twist decreases linearly from the root to the tip.
4. The airfoil is constant throughout the blade.
5. The cross flow along the span of the blade is ignored.
6. Aeroelasticity effects are ignored.

7. The blade root and tip lift losses are ignored.
8. The induced velocity is computed with the simplified linear induced inflow. It is assumed to be a good approximation of the real induced velocity as demonstrated empirically in previous literature.
9. The nonlinear aerodynamic effects between (damaged) blades are not considered.
10. The nonlinear aerodynamic effects between propellers are not considered.
11. The nonlinear aerodynamic effects between the propellers and the body frame are not considered.
12. The data used for the identification of the lift and drag coefficient curves is obtained from the aerodynamic gray-box model that provides the propeller thrust. Hence, the present work adopts the assumptions taken for the development of this model.
13. The blade is cut parallel to the edge of the propeller, perpendicular to its span, such that the remaining polygon is still a trapezoid. Hence, slanted or irregular cuts are not considered.

Further work in the simulation of propeller damage could be oriented towards the refinement of the model developed in this chapter in order to remove one or multiple of the aforementioned assumptions; contributing to its generalisation and application to different propeller types. For instance, the geometrical assumptions 1-4 could be eliminated by creating a 3D model (digital twin) of the propeller using scanning technologies that probe the propeller through physical touch (with contact), such as Coordinate Measuring Machines (CMM) [259], or scanning technologies that exploit acoustic, optical or magnetic approaches (without contact), such as laser scanning, structured light or photogrammetry [260, 261] (e.g. structure from motion). If translated to a CAD model, this would allow the computation of the twist, chord and volume of each blade section, the latter being used for the computation of the centre of gravity when the density of the material is known. Additionally, such model would contribute to the potential discovery of multiple airfoils present in the blade. If that would be the case, the parameter vector of Equation 9.41 would be expanded with the polynomial coefficients used to identify the lift and drag coefficient curves of those additional airfoils.

Assumptions 5-11 are related to the degree of aerodynamic complexity introduced in the model. In particular, assumption 6 points out that aeroelastic effects have been ignored. Most literature in this regard is oriented towards the modelling of helicopter aeroelastic and blade flapping behaviour [262]. Unfortunately, this knowledge is not directly applicable to drones given that helicopters have a flapping hinge, which allows the blade to be displaced up and down to compensate for the rotor lift dissymmetry [29]. Instead, commercial drone rotors lack an articulated head, causing their material to bend and the rotor to tilt with the possibility of flapping [263]. As an alternative, the field of wind energy could be explored since Blade Element Momentum Theory approaches have been used as the aerodynamic component of wind turbine aeroelastic models [264]. However, given the circular dependency between the blade deformations (aeroelastic effects), the induced velocity, and the generated moments and forces, the authors consider such implementation to be challenging for real-time simulations.

Assumption 7 mentions that the blade root and tip losses were ignored. At those blade locations, the circulation must be equal to zero and at the tip there is an additional reduction of lift due to the appearance of tip vortices — airflow around the tip due to the pressure difference between the pressure side (high pressure) and the suction side (low pressure). In the field of wind energy [265], these effects are taken into account by multiplying the induced velocity with a correction factor that is a function of the distance to the centre of rotation (r). This factor would acquire a value of 1 in the centre of the blade and a value of 0 at the edges, allowing the induced velocity to fall to zero at the blade edges. Alternatively, previous literature [29, 265, 266] has also proposed the Prandtl tip-loss factor approximation (B=0.95-0.98) to compute the effective blade radius (R_e = BR) and account for the loss of blade lift. As a result, the outer portion of the blade (R-R_e) is considered to be incapable of carrying lift. Given that in helicopter aerodynamics the introduction of the tip loss factor can cause rotor thrust reductions between 6-10% [29], the study of its implementation in drone propellers is recommended for the further improvement of the BET thrust and torque predictions.

Regarding the induced velocity model used (assumption 8), a comprehensive benchmark study of the different induced inflow models applied to drone propellers is missing in the current literature and it could be considered a line of further work. It is recommended that future studies investigate the suitability and accuracy of the inflow models of Mangler and Squire [267, 268] and Ormiston [269, 270]. The former associates the pressure field across the rotor disk to the inflow with the incompressible, linearised Euler equations. This method originally requires to solve for the rotor loading (Δp) using BET, an approach which is computationally expensive when compared to the linear induced inflow model that optimises the induced velocity before the first BET iteration. However, for the purpose of this research, the thrust obtained from the gray-box aerodynamic model can be used for the computation of the required rotor loading ($T = \pi R^2 \Delta p$), allowing the v_i identification before any BET computation.

Other interesting approaches to consider include linear inflow models, such as those from Payne [271] and Pitt & Peters [254], as well as the Pitt-Peters [272] and the Peters-He [273, 274] dynamic inflow models. The last two approaches have been consolidated and broadly used in the field of rotorcraft dynamics because they exploit unsteady actuator disc theory for hover and forward flight. Instead of ignoring wind-speed fluctuations by averaging the wind field (frozen wake model) or assuming that the instantaneous wind velocity corresponds to that of steady-flow conditions (equilibrium wake model), these dynamic inflow models accurately describe the wake behaviour by assuming the existence of a delay before the induced inflow reacts to modifications in the wind field (unsteady-flow) [265]. Additionally, they are both represented in state-space form, which could be implemented and solved in real-time simulations, and there exist augmentations to their original formulations which include wake distortion effects during manoeuvring flight [275]. Furthermore, even though vortex methods are much more accurate, their computational cost is too high for online blade damage simulations [29, 276].

For the introduction of non-linear inter-propeller, inter-blade or body-blade interactions (assumptions 9-11), the creation of a data-driven model that provides the highly nonlinear lift and drag contributions of each blade section, that are not encapsulated in the BET model, is recommended. Similar work that could serve as inspiration is carried out within the field of aerodynamics, discipline in which turbulence is modelled for Reynolds-Averaged Navier-Stokes (RANS) computations using artificial intelligence [277, 278] (data assimilation for CFD closure). In this approach, physics is exploited for simulating large scale flow behaviours, whereas machine learning, a mostly black-box approach, is used for modelling the highly nonlinear lower scale turbulence using experimental data. Within aerodynamics, this method is valued for its low computational cost when compared to higher fidelity but more expensive simulations, such as Direct Numerical Simulations.

The gray-box aerodynamic model [244] is a data-driven identification approach with physical and semi-physical parameters. Its parametric model structure, namely a piecewise polynomial, is variable since components have been added and removed according to a stepwise selection scheme depending on their contribution to the model accuracy. Beyond its structure, the main model assumption derives from the identification of its parameters with wind tunnel data obtained in quasi-steady flow conditions; there is no rate of change of velocity with time at a single point in the test section volume but the vehicle states, such as the angle of attack, constantly change due to its circular flight motion. It does not enter the unsteady-aerodynamic flow regime because effects caused by the changing circulation and wake on the aerodynamic surfaces are not considered. Future research that would aim to use the developed fault detection and identification framework "in the wild" under the presence of wind field changes, drastic manoeuvres, gusts and turbulences would require the revision of this assumption.

Finally, the author of this research would not only like to acknowledge the potential of the work discussed in this chapter for the computation of the emerging wrenches upon propeller damage, but also for the identification of the blade aerodynamic properties — features mostly unknown in commercial drones. The present research and mentioned further work reduce the component level at which the wrenches are computed, from the complete propeller, to single blades and individual blade sections. Such reduction in scale will allow simulations capable of alternating between multiple physics models depending on the level of detail required. For example, simulation with the gray-box aerodynamic model could be considered the standard, only complemented by the BET modelled $\Delta\vec{F}^P$ and $\Delta\vec{M}^P$ upon blade damage. More accurate physics models contribute to the creation of more realistic simulations that will foster the potential discovery of emerging subtle data features capable of improving the current UAV on-board failure detection and diagnosis effectiveness.

9.4. Model validation

An experimental campaign was carried out in the Open Jet Facility (OJF) wind tunnel at the Faculty of Aerospace Engineering at TU Delft. The reasoning behind the test, as well as its setup and encountered challenges are explained in subsection 9.4.1 and subsection 9.4.2. Then, subsection 9.4.3 and subsection 9.4.4 discuss the data pre-processing steps and the results, respectively. Finally, subsection 9.4.5 presents the conclusions from the test campaign related to the validation of the BET model.

9.4.1. Experimental campaign rationale

Two experimental approaches could have been followed for the validation of the BET developed model:

1. Validation of the identified airfoil lift and drag curves. For that purpose, the thrust and torque of a propeller, whose aerodynamic curves are known, are measured at different ($\vec{V}^B$, $\vec{\Omega}$, ω) conditions. Those conditions, as well as the measured wrenches, are fed to the BET model and the identified aerodynamic curves are compared to the theoretical ones for validation.

2. Validation of the measured thrust and torque. For that purpose, the thrust and torque of a Bebop 2 propeller are measured at different ($\vec{V}^B$, $\vec{\Omega}$, ω) conditions. The measured wrenches are compared to those obtained by the BET model, which has been fit the gray-box aerodynamic model, under the same conditions.

The second approach was selected for the identification of the BET model for two reasons. Firstly, the validation of the identified airfoil lift and drag curves requires having a drone propeller whose aerodynamic curves are known. Unfortunately, those aerodynamic curves are mostly unknown for commercial drones. A potential solution would be a 3D scan of a propeller, the isolation of its airfoil and the identification of its lift and drag curves. These are steps which add additional work that is not required for the second validation approach.

Secondly, once the propeller is available, a lot of data points would be required in order to identify the BET model to fit the data of this new propeller. In Appendix D it is shown that approximately 16,000 data points would be required for a good fit. If every data point would mean 20 seconds of test time in order to eliminate the transients, that translates to approximately 89 hours of required data gathering. This is equivalent to more than a full week (7 days) full time of testing at the wind tunnel, which would be unfeasible. The wind tunnel is required in order to create different wind speed conditions. In contrast, the second validation approach uses the gray-box aerodynamic model for data gathering in simulation, which is orders of magnitude faster and does not require the use of expensive testing facilities.

To sum up, both approaches exploit a similar set-up with a test stand measuring the thrust and torque of a propeller in the wind tunnel at different conditions. However, the second approach is faster and more economic since it does not require a 3D scan of the propeller or long experimental campaigns.

9.4.2. Test set-up, data collection and challenges

The experimental campaign was carried out from the 1st till 7th August 2022 at the OJF at Delft University of Technology. It is a wind tunnel with an octagonal open test section of 2.85 metres in width and height through which the air flows into a room with a width of 13 metres and a height of 8 metres, as can be observed in Figure 9.46. The maximum wind speed that can be reached is 35 m/s.

For the measurement of thrust and torque, the Series 1580 test stand from Tyto robotics was used. This is a dynamometer for drone propulsion systems capable of measuring up to 5 kg of thrust and 2 Nm of torque, as well as voltage, current, power, motor rotational speed and vibration. Figure 9.47 and Figure 9.48 show the test stand from the side and top highlighting its most important components.

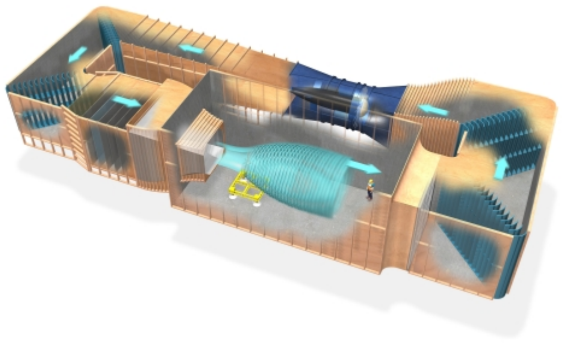


Figure 9.46: Schematic of the Open Jet Facility [30]

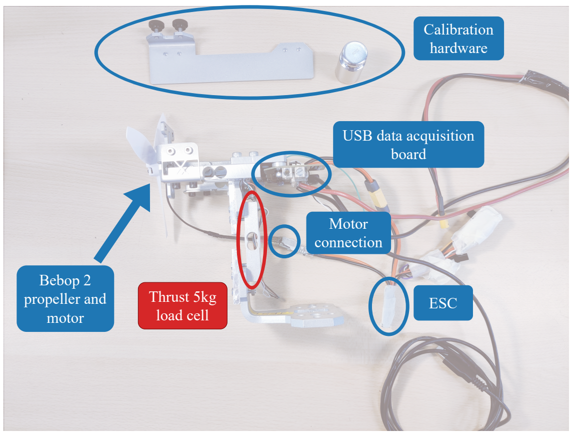


Figure 9.47: Tyto stand: side view with calibration hardware

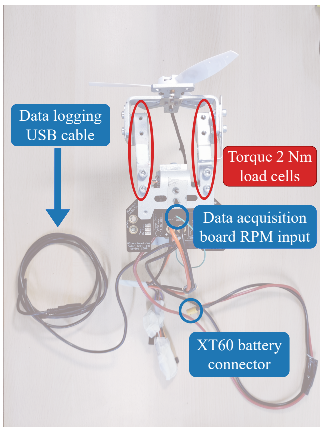


Figure 9.48: Tyto stand: top view

The local influence of the test platform on the freestream flow was minimised by means of a beam assembly, as can be seen in Figure 9.49 and Figure 9.50. This also enables the positioning of the test stand into the wind tunnel air flow, reducing the wind tunnel wall effects on the flow field.

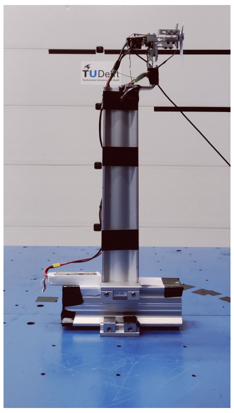


Figure 9.49: Test set-up in the wind tunnel.

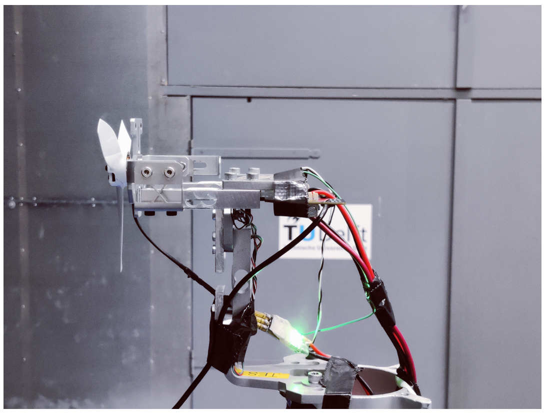


Figure 9.50: Tyto test stand in the wind tunnel.

Table 9.3 shows the parameters that were modified between measurements and their value ranges. For the current experimental campaign, only a single blade was cut at a time with BD percentage of damage. Furthermore, as mentioned in subsection 9.2.3, the inputs for the BET model in order to create a prediction of thrust and torque are the linear velocity of the drone $\vec{V}^B$, its angular velocity $\vec{\Omega}$ and the propeller rotational speed ω. In the wind tunnel, $\vec{V}^B$ is simulated as the negative wind speed vector ($\vec{V}_w^B$) and, since the test stand is not rotated during each measurement, $\vec{\Omega}$ is considered equal to zero. $\vec{V}_w^B$ is decomposed in the wind speed vector magnitude V_∞ and the angle of the normal of the propeller plane with respect to the airflow, also known as the propeller incidence angle i_p (Fig. 9.51).

Table 9.3: Experimental campaign testing parameters and values.

Parameter	Unit	Values
BD	%	0, 10, 25
i_p	°	0, 15, 30, 45, 60, 75, 90
V_∞	m/s	0, 2, 4, 6, 9, 12
ω	rad/s	300, 500, 700, 900, 1100

The maximum blade damage tested was 25% due to excessive vibration loading induced on the load cell. The blade damage was created with a cut on the blade orthogonal to its span, as can be seen in Figure 9.52 and Figure 9.53 for 10% and 25%, respectively.

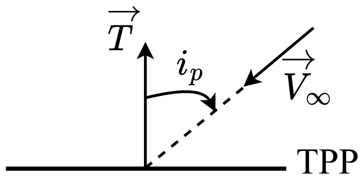


Figure 9.51: Propeller incidence angle.

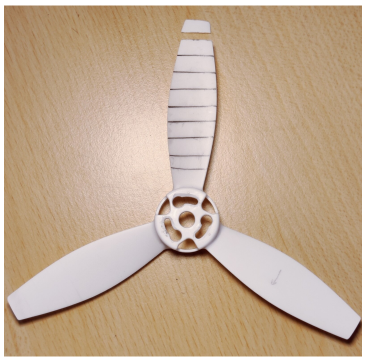


Figure 9.52: Damaged propeller with BD=10%.

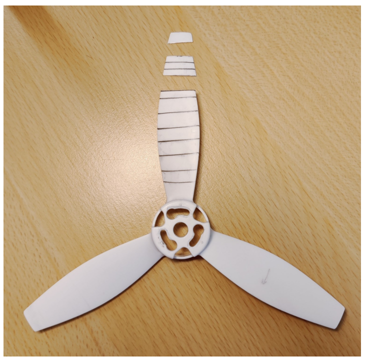


Figure 9.53: Damaged propeller with BD=25%.

The propeller rotational speed was controlled from the laptop to which the data logging cable of the test stand was connected. The test stand has an UI with which it is possible to control the propeller rotational speed and read rpm measurements in real time. The wind speed was controlled and monitored from the OJF control room. Test section awareness was provided by the OptiTrack multi-camera three-dimensional optical tracking system, which was also used to measure the incidence angle of the propeller. The resulting test stand positions were marked on the platform as can be seen in Figure 9.54.

Each combination in Table 9.3 is called an scenario and the order in which the parameters were modified was: $\omega \rightarrow V_\infty \rightarrow i_p \rightarrow BD$. This means that, first, the data was gathered maintaining the blade damage, propeller incidence angle and wind speed constant whereas the propeller rotational speed was increased from 300 rad/s to 1100 rad/s during a single measurement. As a result, every measurement contained 5 scenarios, one for each rotational speed. Once the time for an scenario was concluded, the rotational speed of the propeller was increased to the next value of the discrete list provided in Table 9.3. Once the maximum ω value is reached, then V_∞ is increased by one step and the process is repeated.

For every scenario, thrust and torque data was gathered at 7Hz for 20 seconds. Given that there are 630 scenarios, the data gathering component of the experimental campaign lasted 3.5 hours; much shorter than the full week required for the first validation approach discussed in subsection 9.4.1.

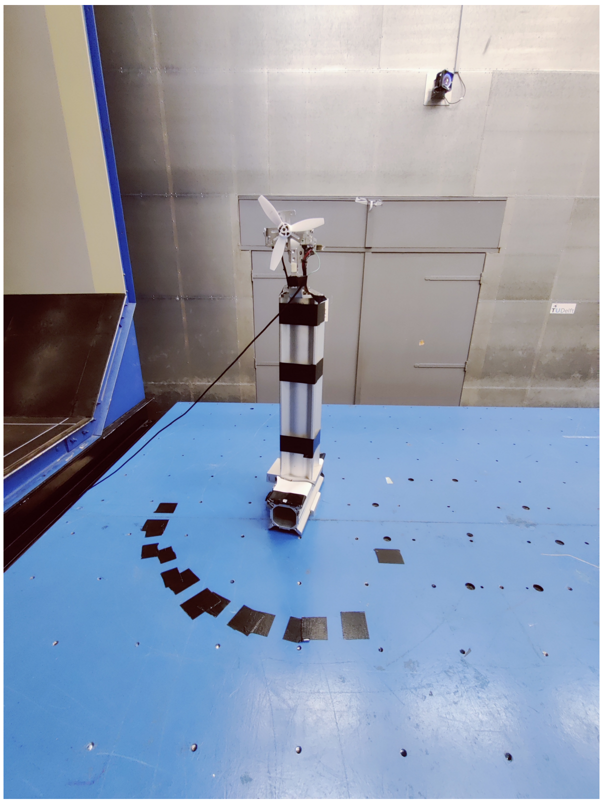


Figure 9.54: Test stand positions marked on the platform with tape using OptiTrack system.

Despite a well-prepared experimental campaign and test set-up design, a few challenges were encountered which may affect the results and conclusions derived from the data gathered:

1. Even though the test stands measures accelerometer and propeller rotation values at around 100Hz, it measures the thrust and the torque at 6-7Hz. This is insufficient for signal reconstruction because, in the case of blade damage, the sinusoids observed in section 9.3 have a frequency of 50Hz-175Hz, depending on the propeller rotational speed. The frequency of those oscillations have a much higher value than the Nyquist frequency of 3.5Hz derived from the test stand wrench sampling frequency of 7Hz.

   Unfortunately, this was the only drone propeller test stand available within the department of Control & Simulation and MAVLab at the faculty of Aerospace Engineering at TU Delft known by the author. There exist larger and more complex test stands for the measurement of wrenches for larger propellers within other departments. However, since they need to be able to measure larger thrust values, their sensitivity or accuracy to those of small drone propellers, such as the Bebop 2 drone, might be insufficient, especially if small vibrations in the order of 10^{-4} are expected to be observed.

2. During the execution of the experiments it was encountered that the test stand resonated with the vertical beam and/or the platform at certain rotational frequencies. This resulted in the observation of peaks in the rpm and wrench measurement signals, as it will be later shown in the results.

   As recommendation for future work, a potential method for removing or minimising the impact of the resonance present in the measurements requires the characterization of the complete set-up, meaning the Tyto test stand on the vertical beam on top of the platform, in order to identify its dynamics. For that purpose, the accelerometer within the Tyto stand could record the data at 100 Hz. Then, this information would be used for simulating the set-up dynamics given a forcing function, namely the oscillating thrust force, as well as the interactions between the set-up and the damaged propeller induced vibrations. Knowledge about the complete system behaviour would allow a deeper understanding of the measured signals (explainability) and the separation of the set-up dynamics from the "pure" thrust and torque oscillations caused by the damaged propeller.

3. The 5kg thrust load cell was not able to withstand prolonged vibrations at 25% blade damage or survive a higher degree of blade damage for more than a few seconds. During the testing of BD=25% the original test stand thrust load cell was damaged and a new one was bought with urgency from another seller. As a result, metal adaptors were designed and manufactured in order to integrate the new load cell for the remaining measurements at 25% BD.

   The author considered using 10kg load cells for thrust in order to withstand higher vibrations. However, such a load cell would have a higher noise which would make the identification of the blade damage oscillations more difficult. Hence, the 5kg load cell was used and 25% blade damage was considered to be the limit for the available test stand.

   Unfortunately, after the experimental campaign, the author was informed that the newly bought 5kg load cell had a rated output accuracy (R.O.) of 0.03%. This means that it has an accuracy of $\pm$0.0015kg or $\pm$0.0147 N, an error larger than the amplitude of the damaged induced oscillations. As an example, Figure 9.30 shows an amplitude of 0.01 N when the propeller is oscillating at 600 rad/s with 25% damage. Much smaller oscillations are found with smaller degrees of blade damage and propeller rotation velocities.

An improved version of the current test stand would be the Series 1585 from Tyto Robotics which has a sampling rate of 80Hz. However, to solve the aforementioned challenges the author can only recommend the in-house design of a test stand for the measurement of wrenches of partially damaged propellers. Such a stand would require a sampling frequency above 100Hz for measurements carried out at the minimum rotational speed of 300 rad/s (or 350 Hz for ω of 1100 rad/s) and a dampening system which prevents resonance with the rest of the structure and the testing platform. Additionally, especially designed load cells have to be used capable of withstanding at least 6g of sustained vibrations.

9.4.3. Experimental data pre-processing

The data pre-processing phase of the model validation has 5 steps:

1. Experiment logbook comments implementation
2. Filename modification
3. RPM isolation
4. Effect of wind on stand correction
5. Signal detrending

During the execution of the experiments, a logbook is maintained with information about all the abnormal events and mistakes that took place. As an example, the thrust load cell started to behave anomalously till failure during the 25% blade damage testing. As a result, multiple scenarios had to be repeated and the old files had to be removed. Such file removal did not take place during the time at the wind tunnel and had to be carried out as the initial step of the data pre-processing step.

Once the logbook comments were implemented, the next step was the modification of all the filenames. Originally, the files have the name of the date and time when they were created, such as "*Log_2022-08-07_124724.csv*". However, such name does not provide information about the recorded scenario. As a result, they were modified to the following format: *bX_aY_wZ.csv*, where X is the blade damage percentage, Y is the angle of the propeller plane with respect to the flow and Z is the wind speed. The X, Y and Z values were retrieved from the comments in the header of the original .csv files.

The next pre-processing step is the RPM isolation. As mentioned in subsection 9.4.2, each measurement contained 5 scenarios, those corresponding to the different propeller rotational speeds while the rest of the parameters were maintained constant. However, it is desired to have each scenario in a different .csv file for later data analysis instead of the exported .csv file per measurement.

To decide which time steps to use in order to split the measurement file for the different ω, the information about the electrical signal input sent to the ESC for rpm control can be used. Similarly to servo motors, the ESC is controlled by Pulse Width Modulation (PWM), namely a discrete signal that is either 0 or 1. The longer the signal has a value of 1 compared to a value of 0, the faster the motor spins. If the signal has a value of 0 and a value of 1 for the same time duration, the motor does not spin. Fortunately, next to the wrench and propeller rotation data stored at each time step, the test stand stores the time it spends in microseconds with a value of 1 per oscillation, also known as ESC value. An example plot of that ESC value can be seen in Figure 9.55. It clearly shows as flat plateaus the time steps at which the commanded ω was constant, as well as those transitional time steps at which there is a linear increment. Given that it is known that there should be 5 plateaus, the constant rpm time steps are separated programmatically by finding the 5 most repeated ESC values and their corresponding time steps. The results can be observed in the plot of ω with respect to time shown in Figure 9.56. The data corresponding to each rpm constant value is separated in different .csv files with the following format: *bX_aY_wZ_rR.csv*, where R is the value of ω for that particular file.

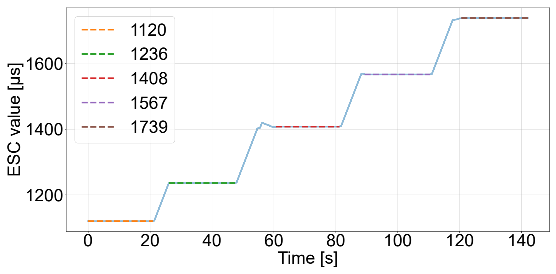


Figure 9.55: Smooth electrical input signal to the ESC for rpm control with identified plateaus.

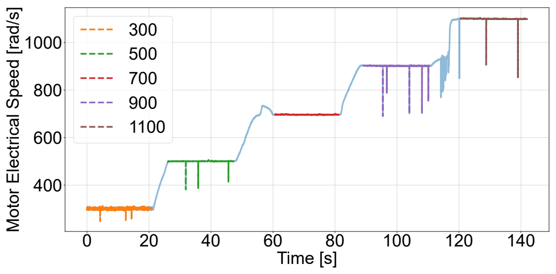


Figure 9.56: Motor electrical speed with respect to time with the identified commanded rpm values.

Performing the same operation directly on the ω signal would have been more challenging due to its noise, its outliers and the effects of the challenges faced during the experimental campaign and presented in subsection 9.4.2, such as the test stand resonance. Additionally, it can be observed in Figure 9.57 that, despite a constant commanded rpm shown in Figure 9.55, the rpms signal at 1,100 rad/s is not constant but decreases linearly; observation which would further difficult the rpm separation directly on the ω signal. The implementation of a smoothing filter would have introduced lag and an inaccurate rpm separation.

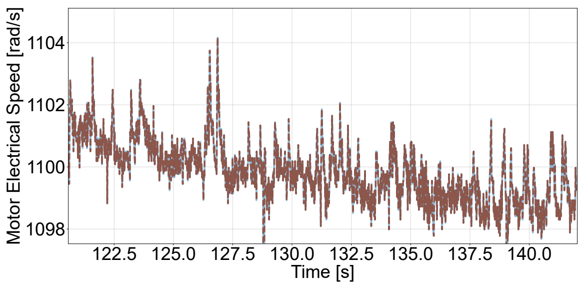


Figure 9.57: Motor electrical speed with respect to time zoom-in at 1,100 rad/s.

Now that each scenario has been isolated in each own file, the next pre-processing step is the correction of the data by adding the effect of the wind on the test stand. For that purpose, before finalising the experimental campaign, the thrust and torque were measured without a propeller installed at different wind speeds and angles with respect to the incoming flow. The mean measured wrenches were then added to each file depending on their V_∞ and i_p values.

Finally, in the case of blade damage, it was observed that the forces and moments were increasing with time even though the parameters in Table 9.3 were kept constant. This is considered an error of the sensor and the signals are detrended as the final pre-processing step. As an example, Figure 9.58 shows the thrust measurements and their detrended values for a scenario with 10% blade damage, 0° of the propeller incidence angle, 2 m/s wind speed and 700 rad/s. The dash lines represent the linear fit of each data set.

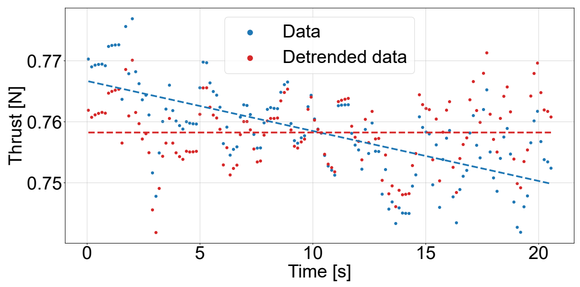


Figure 9.58: Thrust data sample and its detrended counterpart with their fitted linear curves.

9.4.4. Experimental results

The current section will discuss the results from the data gathered during the experimental campaign. Section 9.4.4.1 will present the results without blade damage followed by subsubsection 9.4.4.2 which will present those in the presence of 10% and 25% blade damage.

9.4.4.1. Without blade damage

Now that the data has been compartmentalized into single scenario files and pre-processed, it can be analysed. As it is shown in Table 9.3, there are four input parameters that define an scenario, namely the blade damage, the propeller incidence angle, the wind speed and the propeller rotational speed. This section will delve into analysing the impact of the last three on the performance of the BET model when there is no blade damage (BD=0%). Additionally, in contrast when there is blade damage, it is possible to compare BET's performance to that of the gray-box aerodynamic model.

First, the effect of the angle of the propeller plane with respect to the incoming flow on the results can be observed by fixing the wind speed and the propeller rotational speed to constant values, whereas i_p is increased from 0° to 90°. This is done, for V_∞=2 m/s and ω=700 rad/s in Figure 9.59 and Figure 9.60, where the results are shown for thrust and torque, respectively. A wind speed of 2 m/s has been chosen because the value of this parameter oscillates in the AirSim simulation between 0 and 3.6 m/s, and a value of 700 rad/s for the propeller rotational speed because it is the median of the range in Table 9.3. As a result, these plots show the performance for the average scenario to be encountered in simulation.

The upper window of each plot presents the values obtained in the experimental campaign (x-axis) with respect to the values obtained by each of the models (y-axis) given the same conditions in terms of $\vec{V}^B$, $\vec{\Omega}$ and ω. In an ideal scenario, both models' data points would fall on the dash black line, which would mean that both experimental and simulated results are equal. Unfortunately, that is mostly not the case and the solid red and blue lines represent the linear fit of each model's data. Besides that, the data points are plotted with different degrees of transparency. The degree of transparency varies linearly from the brightest or most opaque markers representing those data points measured at i_p=0° to the most transparent ones representing those data points measured at i_p=90°. Finally, the whiskers represent the range of values in which 95% of the experimental wrench data samples can be found ($\approx 2\sigma$). For the computation of these value ranges, the standard deviation of the forces and moments

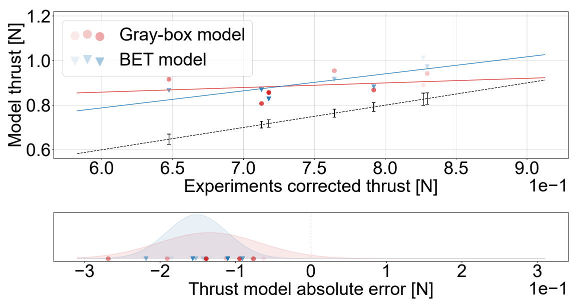


Figure 9.59: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

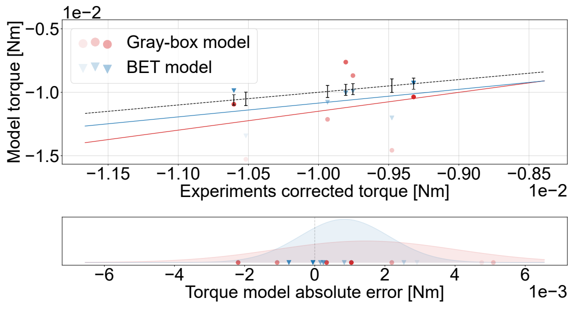


Figure 9.60: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

exerted by the wind on the test stand (subsection 9.4.3) were included. The lower window of each plot shows the wrench absolute error of each model with respect to the experimental measurements, as well as a fitted Gaussian curve to those error data points.

The plots corresponding to the different propeller rotational speeds with 0% blade damage and 2 m/s of wind speed can be found in section E.1. As expected, in general, the measured thrust decreases with increasing i_p (the less transparent, the lower the thrust) because the blade element angle of attack is decreased due to a higher wind speed perpendicular to the plane of rotation. Furthermore, it can be observed that the thrust is always positively biased and the torque is negatively biased for both models. This indicates that there exist some unmodelled physical effects that have not been taken into consideration, among which might be those outlined in subsection 9.3.3. Additionally, these plots show that the performance of the BET and gray-box models is very similar, which supports the hypothesis that the BET model has been well identified (the verification phase in subsection 9.2.3) and that the errors are due to those in the gray-box model whose data was used for identification.

After having seen the effect of varying i_p, the next step would be the analysis of the influence of the propeller rotational speed. All the results for different values of ω shown in section E.1 can be synthesised and compressed in Figure 9.61 by plotting the mean of the absolute error with whiskers representing 1.96 times the standard deviation ($\approx 2\sigma$; also known as the confidence interval or the range where 95% of the data points can be found). Both the mean and the standard deviation were obtained from the Gaussian curves in the lower window of all those plots. From these plots, two conclusions can be derived for the BET model compared to the gray-box aerodynamic model. First, the BET model is more accurate in torque but less in thrust. Second, the BET is more (over) confident of its predictions because of its smaller confidence intervals. The latter observation is also visible in the plots shown in section E.1 due to the taller and narrower Gaussian fitted curves for the BET model.

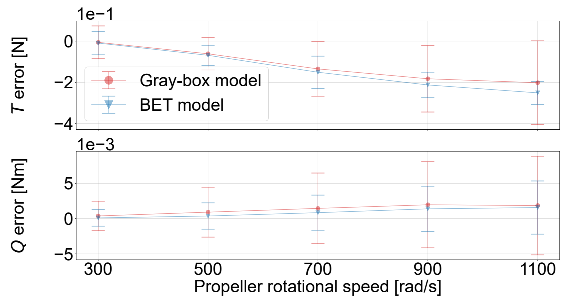


Figure 9.61: BET and gray-box aerodynamic model thrust and torque absolute error for: BD=0% and V_∞=2 m/s.

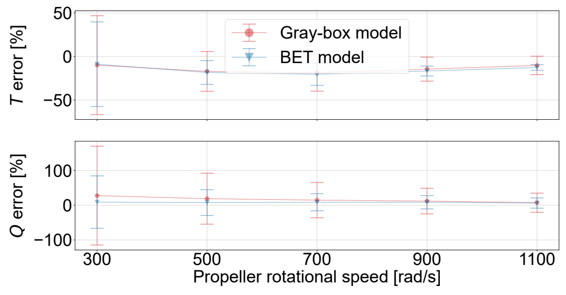


Figure 9.62: BET and gray-box aerodynamic model thrust and torque relative error for: BD=0% and V_∞=2 m/s.

Figure 9.61 shows that both the mean and the standard deviation of the absolute error grows with increasing ω. However, when the relative error is plotted instead, as shown in Figure 9.62, the opposite is observed. This shows that model accuracy increases with ω.

Finally, in order to observe the effect of the increasing wind speed, Figure 9.62 is repeated for each wind speed listed in Table 9.3 within the same plot. This is shown in Figure 9.63 and Figure 9.64 for the gray-box aerodynamic and BET models, respectively. Note that the whiskers representing the confidence intervals have been removed for clarity. From these plots, four observations can be made:

1. The performance of both thrust and torque degrades with increasing wind speed for both models.
2. The relative thrust error of the BET model has a sudden increase when the wind speed is 6 m/s or higher when compared to the gray-box aerodynamic model.
3. The BET model performs better than the gray-box aerodynamic model in terms of torque except at 12 m/s.
4. The performance of the gray-box aerodynamic model at a wind speed of 12 m/s for thrust and higher than 4 m/s for torque is very low (sometimes with relative error values above 1000% for torque).

The reason behind the first three differences in performance between models originates from a design choice in subsection 9.2.3, namely that the BET model was identified with wind speeds up to 3.6 m/s; the highest speeds operated in the AirSim simulator. Hence, the BET model has never seen data collected at wind speeds higher than 4 m/s. The last observation is unexpected as the gray-box aerodynamic model was identified with data gathered at wind speeds up to 14 m/s.

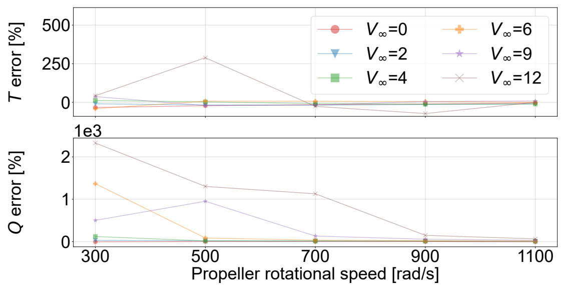


Figure 9.63: Gray-box aerodynamic model thrust and torque relative error for BD=0%.

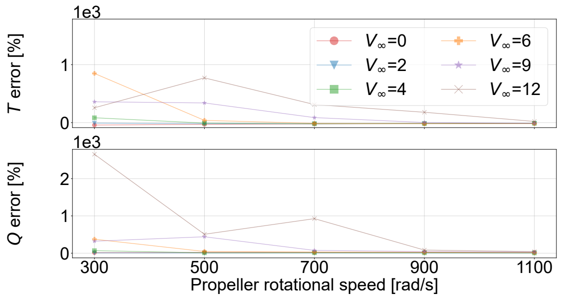


Figure 9.64: BET model thrust and torque relative error for BD=0%.

One general conclusion that can be derived from these observations is that the BET model architecture has a stronger physical foundation for torque than for thrust. Both were identified with data collected at wind speeds lower than 3.6 m/s and the torque is able to perform better at those speeds that the model had not seen before during identification, namely 4, 6, 9 m/s, when compared to the thrust. In most cases, it even performs better than the gray-box aerodynamic model that was used for the identification data generation. This highlights that the unmodelled aerodynamic effects have a stronger impact on the thrust than on the torque.

9.4.4.2. With blade damage

In the presence of blade damage, two signal features need to be validated, namely the bias of the signal and the amplitude of the damage induced oscillations. As in subsubsection 9.4.4.1, the same sensitivity analysis will be performed in which the effect of the parameters in Table 9.3 will be analysed. In this case, the blade will have 10% and 25% damage. Furthermore, the BET model with a healthy propeller was observed to have errors which could be attributed to the gray-box aerodynamic model used for identification. Hence, the BET model's performance with blade damage will also be compared to that presented in subsubsection 9.4.4.1 without blade damage. The latter serves as baseline since it can be considered to be the expected default error introduced by the identification dataset, and not from the BET model's architecture (namely the object of validation of the present section). Finally, as it is the purpose of this research to fill a gap in the literature by developing a thrust and torque propeller model capable of producing reliable estimations in the presence of blade damage, there is no signal from the gray-box aerodynamic model that can be used for control; in contrast to what was done in subsubsection 9.4.4.1.

To observe the impact on the results when varying i_p, Figure 9.65-9.68 show the thrust and torque plots for both blade damages when V_∞=2 m/s and ω=700 rad/s. As in section E.1, the plots recorded at the other propeller rotational speeds can be found in section E.2 and section E.3 for 10% and 25% blade damage, respectively. In contrast to the scenarios without blade damage, when BD=25% the thrust and the torque are not always positively and negatively biased, respectively. This outlying behaviour can only be observed in the thrust plots when ω=500 and 700 rad/s, and it is attributed to the resonance of the test stand with the vertical beam and the platform — the presence of such vibrations was noted in the experiment logbook close to those rotational frequencies.

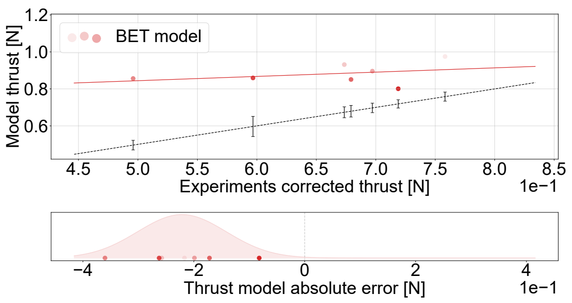


Figure 9.65: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

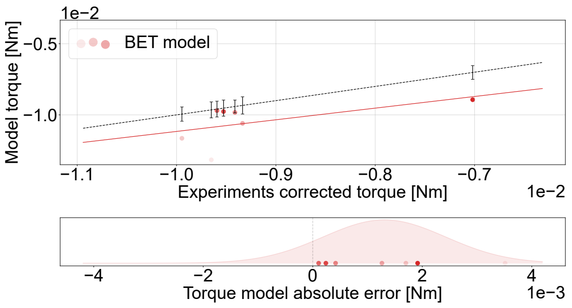


Figure 9.66: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

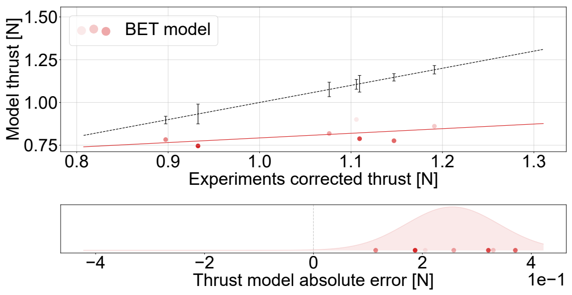


Figure 9.67: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental thrust would match.

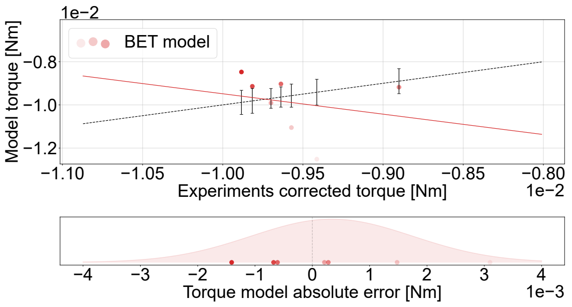


Figure 9.68: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s. The black dashed line represents the ideal scenario in which the model and experimental torque would match.

For the evaluation of the impact of ω on the results, these plots are compressed in Figure 9.69 and Figure 9.70 showing the relative error for each value of ω for both degrees of blade damage. As can be seen, with BD=10%, the relative error behaves in a similar manner as when there is no damage: its mean and standard deviation decrease with ω. However, when BD=25%, that pattern is not observed and the mean even crosses the x-axis for the thrust between 500 and 700 rad/s. Again, this anomalous behaviour is attributed to the strong vibrations observed during the experimental campaign at those rotational speeds and blade damage values.

As mentioned earlier, these results can be compared directly to those without blade damage in order to discern between two potential sources of error: the identification dataset and the model architecture. The reduction in performance beyond the error found without blade damage is associated to the BET

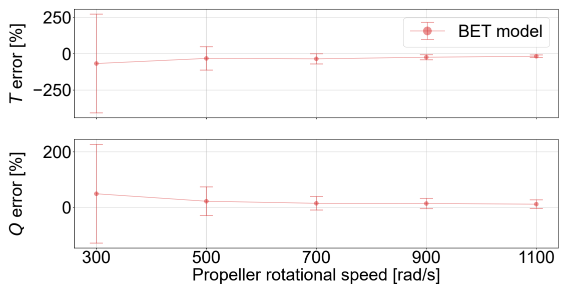


Figure 9.69: BET model thrust and torque relative error for: BD=10% and V_{∞}=2 m/s.

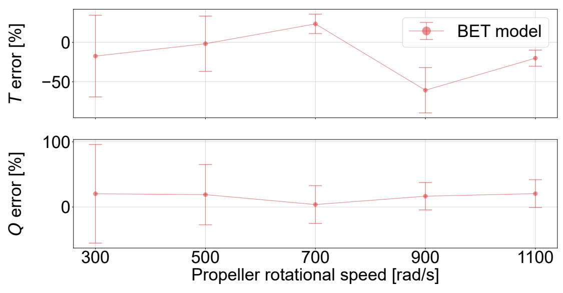


Figure 9.70: BET model thrust and torque relative error for: BD=25% and V_{∞}=2 m/s.

model architecture, which is the object of validation of this section. To facilitate this assessment, Figure 9.71 shows the relative error signals with respect to ω for BD=[0, 10, 25]%. It can be seen that the performance is very similar with and without blade damage, especially at higher propeller rotational speeds. Surprisingly, the 10% blade damage signal has a higher standard deviation than 25% BD and performs worse at low ω values.

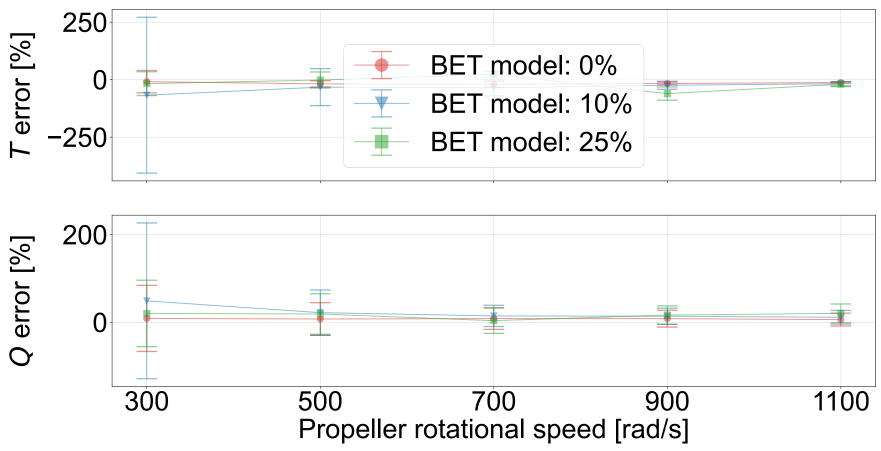


Figure 9.71: BET model thrust and torque relative error for: BD=[0, 10, 25]% and V_{∞}=2 m/s.

In order to better assess the error originated from the BET model architecture in scenarios where there is blade damage, Figure 9.72 shows the same curves for BD=10% and 25% as in Figure 9.71 after having subtracting the error when BD=0% — an error which could be attributed to the identification data. This is known in the current work as Δerror. For clarity, the confidence intervals were not included. This plot shows more clearly the worse performance of BD=10% at low ω values up to 700 rad/s compared to BD=25%. Additionally, two more observations can be made. First, the performance at BD=10% improves with ω whereas the performance at BD=25% does not show a clear pattern (of improvement). Second, the model architecture performance could have an error as high as 76.9% (thrust, BD=10%) and as low as 4.9% (torque, BD=25%) for low and high ω values, respectively. Table 9.4 shows the range of Δerror for both blade damages for thrust and torque.

Table 9.4: Δerror ranges for thrust and torque for 10% and 25% blade damage at 2 m/s wind speeds.

	***BD*=10%**	***BD*=25%**
T Δerror [%]	[-76.9, -23.6]	[-68.5, 15.8]
Q Δerror [%]	[5.1, 39.8]	[-4.9, 14.5]

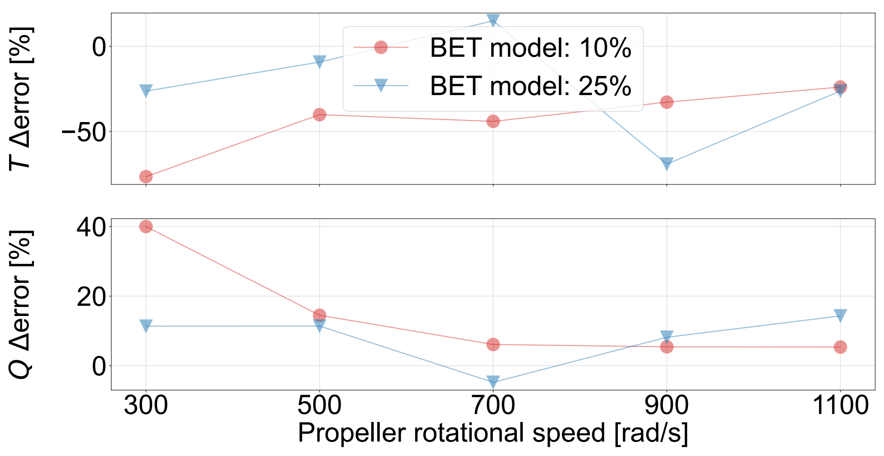


Figure 9.72: BET model thrust and torque relative error for BD=[10, 25]% and V_∞=2 m/s, after subtracting the relative error when BD=0%.

After having varied i_p and ω, the final parameter to be modified is the V_∞, as shown in Figure 9.73 and Figure 9.74 for 10% and 25% blade damage, respectively. Again, a sudden decline in performance is observed in both scenarios for wind speeds higher than 4 m/s, phenomenon attributed to the BET model identification data limited to 3.6 m/s. Besides that, the relative error at those wind speeds is higher at 10% blade damage than at 25%.

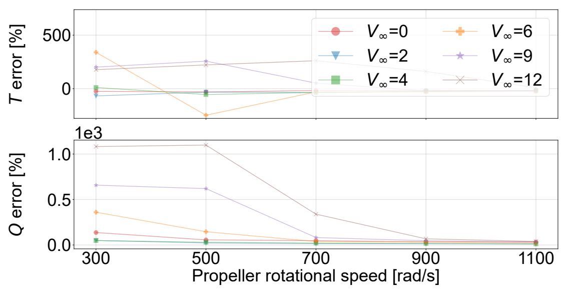


Figure 9.73: BET model thrust and torque relative error for BD=10%.

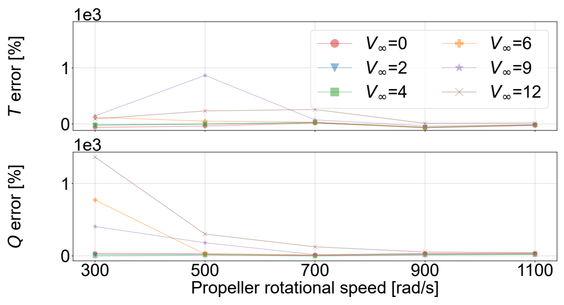


Figure 9.74: BET model thrust and torque relative error for BD=25%.

Finally, the amplitude of the damage induced oscillations needs to be validated. Unfortunately, the sampling frequency is very low for reliable signal reconstruction, as discussed in subsection 9.4.2. In an attempt to reconstruct the signals, the author tried two approaches, namely an evolutionary algorithmic approach with Particle Swarm Optimisation (PSO) and an statistical approach with the Lomb-Scargle periodogram. The goal is to fit a sinusoid with the same frequency as the ω at which the propeller was oscillating to the experimental data.

In the case of the PSO, the algorithm had to identify two variables, namely the sinusoid phase and its amplitude. The phase and the amplitude were constraint to the close intervals [0, 2π] and [0, 0.5], respectively. The maximum attainable amplitude value of 0.5 was chosen from experience when working with the BET model; the observed oscillations had always an amplitude below 0.5. It was initialised with 5,000 particles with a maximum number of 20 iterations as stopping condition. The cost function was defined as the Root Mean Squared Error (RMSE) between the experimental data points and the value of the identified sinusoid at the sampled times. Figure 9.75 shows the amplitude relative error results between the reconstructed experimental and model signals for BD=10% and V_∞=2 m/s for different propeller rotational speeds. The result evidences the failed reconstruction attempt with relative error values in the order of 10^6 for the torque. The reason behind such high errors is that PSO identified the torque signal amplitude to be very close to zero, which leads to very high values when dividing by the experimental amplitude for relative error computation. Consequently, the amplitude of the damaged propeller wrench oscillations can not be validated when reconstructing the experimental signal with PSO.

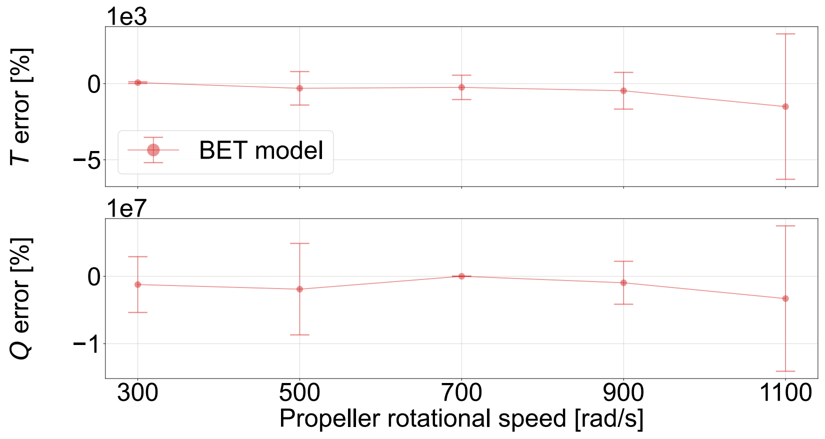


Figure 9.75: BET model thrust and torque oscillation amplitude relative error for BD=10% and V_∞=2 m/s using the Particle Swarm Optimisation for experimental signal reconstruction.

The alternative approach to PSO explored for signal reconstruction is the Lomb-Scargle periodogram [279–281]; statistical algorithm for detecting and characterizing periodic signals in unevenly sampled data. It is widely used in astronomy [282–284] with some special cases in other fields, such as bioinformatics [285]. The Lomb-Scargle periodogram fits a sinusoidal model to the data at the propeller rotational frequency, and the amplitude of that sinusoid is compared with the signals obtained from the BET model.

Figure 9.76 shows again the amplitude relative error for a propeller at 10% blade damage with 2 m/s wind speed. In this case, the Lomb-Scargle periodogram was used for experimental signal reconstruction. It can immediately been observed that the statistical approach has a superior performance with respect to the evolutionary counterpart when estimating the amplitude of the torque oscillations. Instead of resulting in a relative error in the order of 10^6 due to zero amplitude signal reconstructions, the Lomb-Scargle periodogram maintains a mean relative error below 100%. Furthermore, even though it maintains a similar mean relative error for thrust as the PSO, its standard deviation is lower, especially at higher values of ω. Hence, the statistical approach is deemed the best option for damaged propeller experimental signal reconstruction. Similar to section E.2 and section E.3, section E.4 and section E.5 contain the damage induced oscillation amplitude results corresponding to each propeller rotational speed when the experimental signal has been reconstructed with the Lomb-Scargle periodogram for 10% and 25% blade damage, respectively. Appendix F presents a brief discussion about the impact of measurement noise on the performance of this statistical approach for signal reconstruction.

As in the scenarios when there is no blade damage, from Figure 9.76 and Figure 9.77 it can be observed that the BET model performs better for torque than for thrust for both degrees of BD. This further supports the general conclusion stated at the end of subsubsection 9.4.4.1, that the BET model architecture has a stronger physical foundation for torque than for thrust. Moreover, no trend is observed that shows an improvement in performance with increasing propeller rotational speed.

Lastly, Figure 9.78 and Figure 9.79 compare the results for different wind speeds. The data points at zero wind speed are not shown since there are no oscillations in the thrust and torque signals. Again, the relative error grows with increasing wind speed, especially for 9 and 12 m/s. Additionally, the performance is observed to decrease for thrust at 500 and 700 rad/s with 25% blade damage. This is attributed to the resonance behaviour with the test set-up found at these frequencies. Finally, when compared to Figure 9.73 and Figure 9.74, the BET model presents a worse performance when esti-

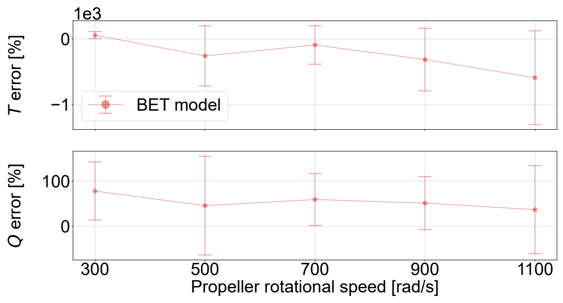


Figure 9.76: BET model thrust and torque oscillation amplitude relative error for BD=10% and V_∞=2 m/s using the Lomb-Scargle periodogram for experimental signal reconstruction.

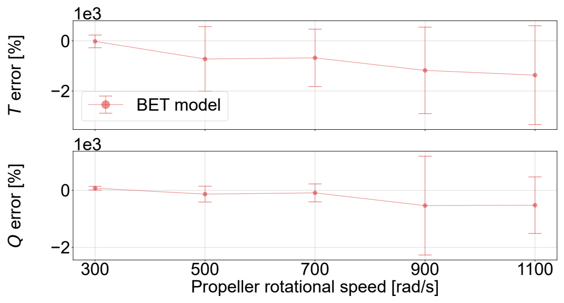


Figure 9.77: BET model thrust and torque oscillation amplitude relative error for BD=25% and V_∞=2 m/s using the Lomb-Scargle periodogram for experimental signal reconstruction.

mating the amplitude of the thrust and torque signal oscillations as opposed to their mean.

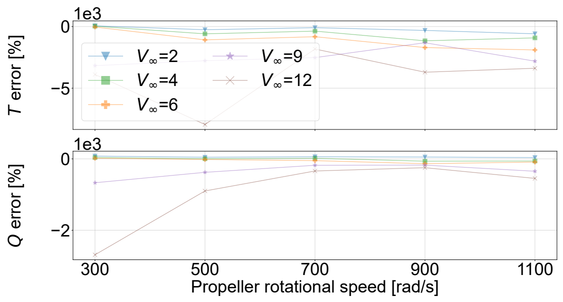


Figure 9.78: BET model thrust and torque oscillation amplitude relative error for BD=10% using the Lomb-Scargle periodogram for experimental signal reconstruction.

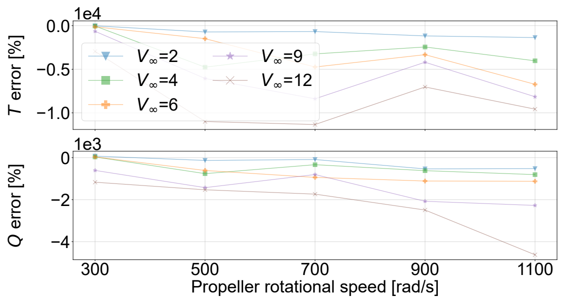


Figure 9.79: BET model thrust and torque oscillation amplitude relative error for BD=25% using the Lomb-Scargle periodogram for experimental signal reconstruction.

9.4.5. Validation conclusions

In the scenarios without blade damage, it is possible to compare the BET model performance with that of the gray-box aerodynamic model. It was observed that the performance of both is very similar, which indicates that the BET model has been well identified and its validation errors are attributed to the identification data gathered from the gray-box aerodynamic model, instead of the BET model architecture. Besides that, the thrust is positively biased and the torque is negatively biased for both models. This remark points to the existence of unmodelled physical effects, among which might be those outlined in subsection 9.3.3. Additionally, the performance of both models decreases with lower values of ω and higher values of V_∞. This shows that both approaches struggle to correctly model blade sections under a negative angle of attack; phenomenon mostly emergent in those ω-V_∞ conditions for blade sections close to the propeller hub.

Despite their similarities, four differences can be found between both models. First, the BET model is slightly more accurate in torque but less in thrust. Second, it is more (over) confident of its predictions when compared to the gray-box aerodynamic model. Third, the BET model experiences a faster decline in thrust performance for wind speeds higher than 4 m/s. Fourth, the BET model performs better than the gray-box aerodynamic model in terms of torque except at 12 m/s. The reason behind these divergences in performance originates from a design choice in subsection 9.2.3, namely that the BET model was identified with wind speeds up to 3.6 m/s; the highest speeds operated in the AirSim simulator. Hence, the BET model has never seen data collected at wind speeds higher than 4 m/s. Furthermore, an unexpected result is the low performance of the gray-box aerodynamic model at high speeds, as it was identified with wind tunnel data gathered at wind speeds up to 14 m/s.

One final conclusion that can be derived from these observations without blade damage is that the BET model architecture has a stronger physical foundation for torque than for thrust. Both were identified with data collected at wind speeds lower than 3.6 m/s and the torque is able to perform better at those speeds that the model had not seen before during identification, namely 4, 6, 9 m/s, when compared to the thrust. In most cases, it even performs better than the gray-box aerodynamic model that was used for the identification data generation. This highlights that the unmodelled aerodynamic effects have a stronger impact on the thrust than on the torque.

Regarding those scenarios with blade damage, it is necessary to validate the signal bias, as well as the damage induced oscillations. When comparing the experimental and BET model signal bias, that of BD=10% behaves similarly to that of BD=0%. On the contrary, when BD=25% the thrust and the torque are not always positively and negatively biased, respectively. Additionally, its relative error does not decrease with ω as it would be expected. This outlying behaviour, especially noticeable when ω=500 and 700 rad/s, is attributed to the resonance of the test stand with the vertical beam and the platform. Moreover, a sudden decline in performance is again observed in both scenarios for wind speeds higher than 4 m/s, phenomenon attributed to the BET model identification data limited to 3.6 m/s. Besides that, the relative error at those wind speeds is unexpectedly found to be higher at 10% blade damage than at 25%.

Finally, for the validation of the oscillations' amplitude, two approaches were implemented, namely Particle Swarm Optimization (metaheuristic evolutionary optimization algorithm) and the Lomb-Scargle periodogram (statistical algorithm). Unfortunately, the first option was not always able to fit a sinusoid to the data, especially for the torque. Hence, the Lomb-Scargle was chosen for signal reconstruction. Again, the BET model performs better for torque than for thrust, which supports the aforementioned conclusion that the BET model architecture has a stronger physical foundation for torque than for thrust. Additionally, the performance degrades with wind speed and it does not improve with propeller rotational speed. Instead, it shows a notorious increase in thrust relative error at 500 and 700 rad/s with 25% blade damage, which is attributed to the resonance observed at those frequencies. Despite relative errors in the order of 1,000% for some scenarios, the validation of the amplitude of the oscillations can not be reliably assessed due to the inaccuracies/noise of the load cell and the difficulty in reconstruction attributed to the low sampling rate.

To conclude, the BET model has been well identified and it has a performance without blade damage similar to that of the gray-box model. In the presence of blade damage, its performance at high propeller rotational speeds is similar to that without blade damage, with differences in relative error oscillating between 5% and 24%. However, the errors at low propeller rotational speeds can be more than 3 times higher; oscillating between 15% and 75%. Besides that, the validation of the damage induced oscillations amplitude is not possible due to the challenges encountered. As a result, it is difficult to fully validate the BET model. The author hopes that the lessons outlined in the present validation will serve as a basis in the design of a future experimental campaign with more specialised hardware.

10 Fault detection and diagnosis

The simulator developed in chapter 8 grants the engineer the possibility of creating an infinite number of different simulations programmatically with or without actuator failures. Additionally, chapter 9 can serve as recipe to develop higher fidelity blade damage simulations than currently available in literature. Thanks to its modularity, the physics, controller and failure modules can be easily swapped for those of other aerial platform. In contrast with readily available Matlab simulators, their integration with the presented pipeline in Unreal Engine 4 allows the collection of photo-realistic visual data. Such data is of superior quality when compared to alternative simulation environments, such as Gazebo, allowing the development of applications with a reduced reality gap.

The last milestone of this master research project is the development of an actuator Fault Detection and Diagnosis (FDD) framework, which does not only alert the drone computer about the presence of an actuator failure, but it is able to point to the failed actuator and quantify the damage. For that purpose, the author proposes an architecture which fuses the information obtained from the Inertial Measurement Unit (IMU) and the camera on-board of the drone. Thanks to the collected medium size dataset with the in-house developed simulator of in-flight drone actuator failures (section 8.6), it is possible to use knowledge-based approaches previously impossible due to the lack of data.

The chapter is organised in two main sections. First, section 10.1 will discuss the model architecture, diving into the separate processing of the camera and IMU data, as well as the fusion of both sources of information. Then, section 10.2 will present the results, including a short ablation study in order to understand the effect of each model component. Finally, section 10.3 will outline the main conclusions and some recommendations for further work.

10.1. FDD model architecture

The complete architecture of the model can be observed in Figure 10.1. Early in the pipeline it consists of two separate paths for independently processing the camera and IMU information for the extraction of features. Then, both paths' features are concatenated and fed to a Long Short-Term Memory (LSTM) for data fusion, architecture with feedback connections which allows the ingestion of sequential data. Finally, the output is passed on to a dense Neural Network for classification with a number of neurons equal to the number of distinct classes. For example, in the case of failure detection there are only two classes, namely healthy or failure. Hence, the classifier only has two neurons in the output layer.

One of the main challenges for any FDD architecture that aims at fusing multiple data sources with different sampling rates is the synchronisation of the information without discarding precious data. On-board of most drones, the IMU is able to produce samples at rates multiple times higher than the camera. A naive approach would be a running the FDD at the same frequency as the camera and taking the last data point from the camera and the IMU at every time step; discarding all the IMU samples collected between camera shots. As it will be demonstrated later in this section, the developed FDD architecture can run at a commanded frequency different than both sensors on board, as long as it is

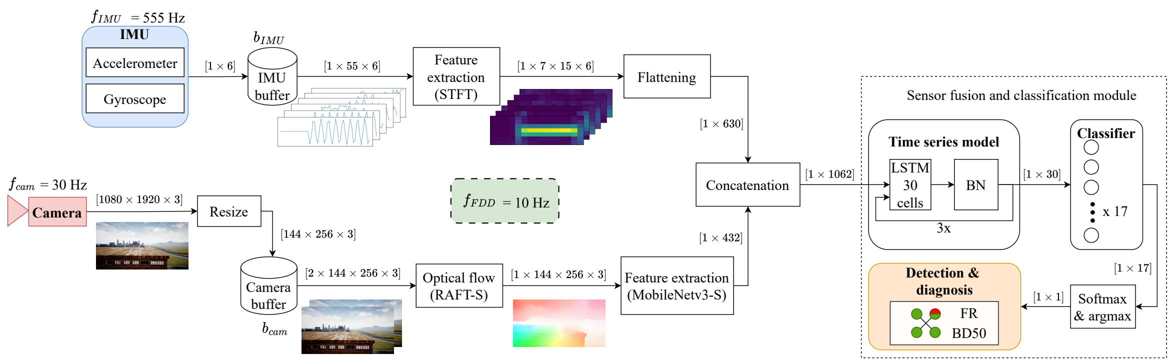


Figure 10.1: The FDD pipeline consists of (i) an IMU time-frequency feature extractor in the form of a Short-Time Fourier Transform, (ii) the MobileNetV3-S as feature extractor from the camera optical flow computed with RAFT-S and (iii) a Long Short-Term Memory network followed by a single layer Neural Network as sensor fusion and classification module. The FDD framework is run at 10 Hz and the sampling rate of the IMU and camera are 555 Hz and 30 Hz, respectively.

equal or smaller than the slower sensor. This highlights the flexibility of the architecture, being able to adapt to different computation constraints.

Next, each of the architecture components will be explained in detail. Even though the framework could be applied to any aerial platform, the values correspond to the Bebop 2 drone used in the present research. First, subsection 10.1.1 will dive into the different components required for the image feature extraction. Then, subsection 10.1.2 will discuss how the IMU data is processed. Finally, subsection 10.1.3 will show how the sensor features are translated to a detection and diagnosis prediction by treating the tasks as a classification problem.

10.1.1. Camera data processing

The inspiration for the introduction of the camera into the FDD pipeline stems from the observation that human beings are able to detect that they are falling thanks to their ”natural time differentiated accelerometer” or vestibular system, an apparatus within the inner ear that provides information about changes in acceleration, as well as from their visual sensory system. When the vestibular system is saturated (e.g. rapidly rotating on an office chair) or the changes in acceleration are imperceptible (e.g. accumulating slow changes in aircraft attitude), the visual sensory system is still able to detect the subject’s ego motion thanks to the relative movement of elements of the environment in its visual field. For instance, if a human subject sees a block moving to the right in a static environment, the subject understands that it is moving then to the left.

The two main factors affecting judgement of self-motion are the gradients and the pattern of optical flow which provide information about the relative velocity (amount) and direction of relative motion, respectively. Hence, the author believes that knowledge about the magnitude and direction of the optical flow could enhance the diagnosis component of the FDD framework by implicitly quantifying the failure magnitude and identifying the failed actuator. For instance, if the front right clockwise rotating (from top view) propeller is lost, then it is expected that the drone will lose lift, tilt forward and rotate clockwise. In optical flow, this should translate to a vector field with an up-left direction. The stronger the gradient, the greater the failure magnitude.

As mentioned in the literature study, there are two ways in which optical flow can be represented, namely sparse and dense optical flow [21]. The main difference is that the first computes the optical flow for a predetermined number of features of interest whereas the second computes it for the complete frame. Even though the sparse optical flow is less computationally expensive, it has two main problems. First, those features of interest may disappear or become hidden after a few frames, forcing the optical flow approach to select new features. Second, the algorithm may choose different features between frames as some become more salient than others throughout time. As a result, it is difficult to infer a potential actuator failure from a specific optical flow change pattern as it could be attributed to the tracking of different frame features. As an example, a feature far in the distance may be chosen at time step t with a small optical flow gradient pointing to the left, whereas at time step t+1 is is replaced by another closer feature with a large optical flow gradient pointing to the right. Despite the agent motion being minimal, the optical flow may wrongly infer the opposite. Hence, dense optical flow was chosen for the present research.

In literature there are two main classes of dense optical flow approaches, namely traditional or classical energy-based and deep-learning based. In recent years, deep learning based approaches have been able to surpass the traditional counterparts in accuracy and lower inference times, allowing them to run in real time and becoming the de facto choice for computationally constrained devices and platforms [286, 287]. In most cases, the performance of optical flow approaches is compared using the Average End Point Error (AEPE) on the MPI-Sintel final dataset and the Fl-all in the KITTI2015 dataset [288]. The AEPE is the average Euclidean distance between the estimated and ground truth optical flow vectors, and the Fl-all is the percentage of flow outliers averaged over all pixels. The MPI-Sintel final dataset[1] [289] is a 564 frame animated movie synthetic dataset with realistic illuminations, reflections and rendering effects; whereas the KITTI2015 dataset[2] [290, 291] is a 200 frame real-world dataset collected from a moving car.

Within this deep-learning based approaches there are three architectures that, according to the author, stand out from the literature for their high accuracy and low inference time, while providing their code and trained model weights. With their trade-off metrics shown in Table 10.1, they are:

- CNNs for Optical Flow using Pyramid, Warping, and Cost Volume (PWC-NET) [292]. It was published in June 2018, one of the fastest methods in literature and the fastest from the selection; it is considered a milestone algorithm in the field [288].
- Recurrent All-Pairs Field Transforms for Optical Flow (RAFT) [293]. It was published in November 2020 and it shows the highest performance of the three considered approaches in the MPI-Sintel dataset with the highest reported inference time [293].
- Displacement-Invariant Matching Cost Learning for Accurate Optical Flow Estimation (DICL-Flow) [294]. It was published in December 2020 and it shows a reported runtime and performance between the PWC-NET and RAFT approaches.

Table 10.1: Performance and inference time comparison of dense optical flow approaches.

Method	Time	K-15 train		K-15 test	S-train (EPE)		S-test (EPE)	
	(s)	EPE	Fl-all	Fl-all	Clean	Final	Clean	Final
PWC-NET[3]	0.03	10.35	33.67%	-	2.55	3.93	-	-
RAFT[4]	0.2	5.04	17.4%	-	1.43	2.71	-	-
DICL-Flow[5]	0.08	8.70	23.6%	-	1.94	3.77	-	-
RAFT-S	-	7.51	26.9%	-	2.21	3.35	-	-
Farneback [295]	1	10.50	-	53.09%	-	8.9	-	-

All of these deep-learning based approaches were trained with data from the FlyingChairs [296] and FlyingThings3D [297] datasets. Next to them, Table 10.1 contains two more entries: a classical approach for comparison, namely Gunnar Farneback's algorithm [298] developed in 2003, and a small pretrained RAFT model (RAFT-S) implemented within the Torchvision model library. In contrast with the original RAFT model, it contains five times less parameters but is able to maintain superior performance in the MPI-Sintel final train dataset when compared to the PWC-NET and DICL-Flow models.

[1]http://sintel.is.tue.mpg.de/
[2]https://www.cvlibs.net/datasets/kitti/
[3]https://github.com/philferriere/tfoptflow
[4]https://github.com/princeton-vl/RAFT
[5]https://github.com/jytime/DICL-Flow

Additionally, the inference time of all the implemented approaches with the datasets collected in this research at different resolutions (section 8.6) are shown in Table 10.2. Each time value is the average time that each algorithm took to predict the optical flow for 250 frames on a laptop with a 6 core Intel Core i7-9750H CPU, 16 GB of RAM DDR4 and an NVIDIA Quadro P2000 with 5 GB of GDDR5 memory. As can be seen, even though DICL-Flow appears to be the intermediate option from literature, it presents the worst inference time for all resolutions.

Table 10.2: Inference time comparison of dense optical flow approaches on the collected UE4 dataset at different resolutions.

Methods	**256×144**	**512×288**	**1024×576**
	(s)	(s)	(s)
PWC-NET	0.073	0.143	0.423
RAFT	0.17	0.17	0.36
DICL-Flow	0.274	0.296	0.617
RAFT-S	0.06	0.10	0.35
Farneback	0.008	0.042	0.177

Figure 10.2 allows for a visual comparison of the approaches' optical flow quality with a frame from the 1024×576 dataset. As can be seen in Figure 10.2b, even though PWC-NET has the lowest run time among the deep learning options, its optical flow prediction is very noisy without any recognisable features, indicating a poor cross-dataset generalization. Furthermore, from Figure 10.2f it can be seen that Farneback does not perceive slight movements. Most of the pixels are black, leading to the loss of potential features (pixels) that could serve as rich sources of information further down the FDD pipeline. Besides that, a strong flickering behaviour has been observed in Farneback's optical flow over multiple frames, which hints to unreliable predictions.

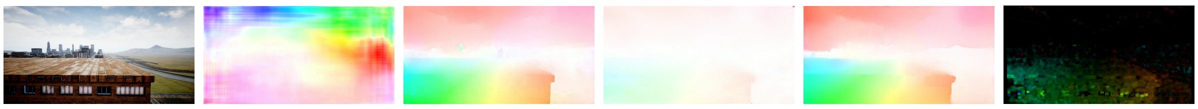

(a) Original (b) PWC-NET (c) RAFT (d) DICL-Flow (e) RAFT-S (f) Farneback

Figure 10.2: Dense optical flow visual quality comparison.

Given the high inference time of DICL-Flow with the collected dataset and the low visual quality of PWC-NET and Farneback, the two remaining options for optical flow computation are RAFT and RAFT-S. As both show similar visual quality and RAFT-S has a run time three times lower than its larger version for the lowest resolution, RAFT-S is chosen as the optical flow module of the FDD pipeline.

Returning the attention to Figure 10.1, the bottom information path shows the camera data processing. In the case of the Bebop 2 drone, the camera captures images at 1080p, meaning frames of 1080 pixels in height and 1920 in width with three RGB channels [1080×1920×3], and they are resized to a tensor of dimensions [144×256×3] before being stored in the camera buffer (b_{cam}). Then, at every time step at which the FDD framework is executed, the b_{cam} contains $f_{\text{cam}}/f_{\text{FDD}}$ + 1 samples, and the first and last entry of the buffer are passed on to the optical flow model. Here, f_{cam} stands for the fps rate at which the drone collects image data and f_{FDD} is the frequency at which the FDD pipeline is executed on board of the drone. Next, the buffer is emptied except for the last stored image, which remains in memory for the next FDD time step. This ensures the temporal coherence of the optical flow over multiple FDD calls.

Once RAFT-S computes the optical flow, the output tensor is fed to a feature extractor. For this part of the pipeline, the author opted for transfer learning instead of developing their own. The model of choice was the backbone of MobileNetV3-Small [299] with frozen weights pre-trained on the ImageNet dataset [300] because it has the lowest inference time among all keras pre-trained models[6] at the time of writing. A depth multiplier (alpha) of 0.75 was set in order to proportionally decrease the number of filters in each layer, achieving a reduction in the number of parameters from 2.9 to 2.4 million (3 ms of inference time). Finally, the last layer of MobileNetV3-Small is set to be a global average pooling layer which collapses the width and height of the output tensor to a single feature, resulting in a 1D tensor of 432 features.

10.1.2. IMU data processing

From the IMU, the FDD algorithm receives six 1D data streams, namely the linear acceleration and the angular velocity in the x, y and z directions. Two key signal features that contribute to the detection and classification of these failures are the evolution of their bias through time and the amplitude of their oscillations; the latter especially in the case of blade damage, as highlighted in chapter 9. Information about both features can be encapsulated in their Short-Time Fourier Transform (STFT), creating compact time-frequency maps or spectrograms and removing potential sensor noise. To illustrate this, Figure 10.3 shows the accelerometer signal in the x-direction and its spectogram for a random flight within the dataset which experienced a blade damage failure of 0.8, 6.83 seconds after the start (as highlighted by the red dashed vertical line). As can be seen, failure can easily be detected by the sudden appearance of signal content at high frequencies, in this case between 173 and 186 Hz.

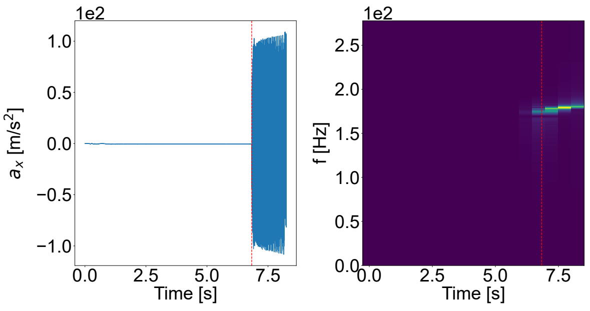


Figure 10.3: Drone acceleration along the x-axis and its spectrogram. The dashed vertical line denotes the time of failure.

From the IMU information path shown in the upper half of Figure 10.1, the incoming data from the accelerometer and the gyroscope is stored in a buffer (b_{IMU}). Once the FDD module is called, the buffer is emptied and its data is used for computing the STFT. This form of frequency analysis is a windowed approach which divides the time signal into small equally sized segments and applies an independent Fourier transform to each one of them. Hence, there is a trade-off between the time and frequency resolutions; the wider the window the higher the frequency resolution at the expense of the time resolution. Since the STFT is applied to small sample sizes of n_{seg}=$\lfloor f_{\text{IMU}}/f_{\text{FDD}}\rfloor$ at a time, a window size of n_{win}=$\lfloor n_{\text{seg}}/4\rfloor$ is chosen with $o = \lfloor 3/4 n_{\text{win}}\rfloor$ samples of overlap between windows, i.e. a stride of s=$\lceil 1/4 n_{\text{win}}\rceil$. The sample vector is padded such that the time resolution or the number of steps in which the time axis of the spectrogram is divided is t_{res}=$\lceil n_{\text{seg}}/(n_{\text{win}}\text{-}o)\rceil$+1. As can be seen, as n_{win} increases, t_{res} decreases. The opposite is observed in the frequency resolution f_{res}=$\lfloor n_{\text{win}}/2\rfloor$ + 1.

For the present research, f_{IMU} of the collected dataset and f_{FDD} approximately equal 555 Hz and 10 Hz, respectively. Hence, 55 samples are fed to the STFT at every FDD time step, which outputs a tensor of dimensions [7×15×6]. This means a frequency resolution of seven and a temporal resolution of 15. Figure 10.4 and Figure 10.5 show the IMU signals and their STFTs for the same flight as in Figure 10.3, using a time segment of 0.1 s (f_{FDD}=10 Hz) starting at 6.78 s in order to include the transition from a healthy to a failure state. Again, it can still be clearly observed when the blade damage has taken place for failure detection. Finally, the STFT output tensor is flattened to a single dimensional tensor of 630 features that will be fused with those coming from the camera processing path of the pipeline.

[6]https://keras.io/api/applications/

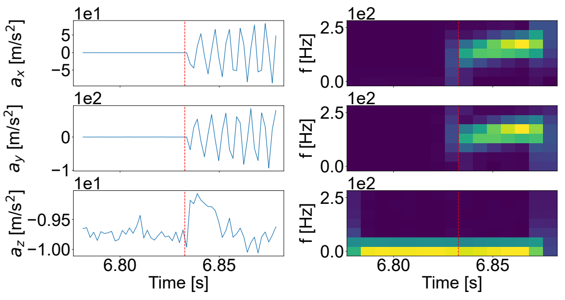


Figure 10.4: Sample flight accelerometer signals and their spectrograms for a 0.1 s time interval starting at 6.78 s.

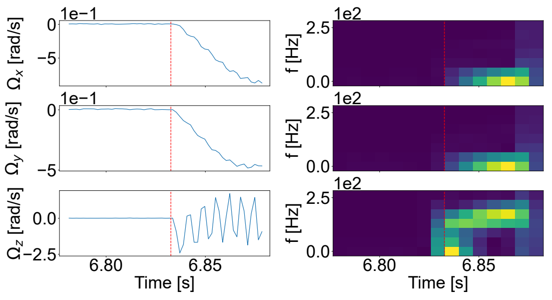


Figure 10.5: Sample flight gyroscope signals and their spectrograms for a 0.1 s time interval starting at 6.78 s.

10.1.3. Sensor fusion and classification module

As can be seen in Figure 10.1, the features from the camera and the IMU are concatenated into a single vector of [1062] features and fed to a sequence-to-sequence LSTM model, which allows the FDD framework to take decisions based on current and previous data at every time step. LSTM cells have an internal state that stores information from an arbitrary number of previous inputs which, in conjunction with the current input, is used to extract sequential relationships to generate an output. For the present research, the time series model consists of a simple stack of three LSTM layers of 30 cells, each followed by a Batch Normalization (BN) layer; transformation that maintains the mean and standard deviation of its input batch close to 0 and 1, respectively. At every FDD time step, an input vector of 1062 features is fed into the network which outputs a tensor of 30 features.

The last stage of the FDD pipeline is the classifier that will simultaneously perform the tasks of failure detection, failure magnitude quantification and failed propeller identification. The problem is simplified by considering each potential drone state, namely each failure mode and the healthy state, as a class. As an example, if abrupt actuator saturation and abrupt propeller fly-off are considered as the only modes of failure, then the classification layer would have to discern among nine classes, namely two failure classes per propeller and one for the healthy state. To perform this classification task, a single layer dense neural network layer is used with the number of neurons equal to the number of classes, followed by the softmax activation function in order to generate a multinomial probability distribution; the model outputs with what probability it believes the input belongs to each class. The goal is that the highest probability is attributed to the correct failure or healthy drone state at each time during the flight.

Both the LSTM model and the classifier are the only two trainable components of the FDD pipeline, as the RAFT-S and MobileNetV3-S weights are frozen. For their training, the sparse categorical cross-entropy loss function and adam optimizer are used, both extensively exploited in literature for multi-class classification.

10.2. Results

To demonstrate the potential of the proposed FDD framework, only four modes of failure were considered per propeller, namely 20%, 40%, 60% and 80% blade damage. As a result, there are 17 classes among which the FDD pipeline should discern. For this purpose, the dataset was split into 70% training, 20% validation and 10% testing. Each flight of this dataset has a variable duration between 6 and 16.9 seconds with an average length of 11.6 seconds. The first second of every flight is ignored in order to avoid the acceleration transient after the flight has started. From the remaining flight time, single 5.5-second data snippets are used per flight in order to batch train and evaluate the pipeline with equal length data sequences without padding. Flights of length shorter than 6.5 seconds only constitute 0.72% of the total dataset and were eliminated. Besides that, flights that were not properly recorded in UE4 — the drone does not take off or the sensor data is not recorded at the correct rate — were also removed. At the end, the training dataset consisted of 3,468 5.5-second flights.

Table 10.3 shows the results for the pipeline presented in section 10.1 in terms of inference time and test accuracy. The runtime was obtained from the same compute setup that was used to generate Table 10.2. Furthermore, three different types of test accuracy are considered, namely general, detection and diagnosis. The first refers to the accuracy outputted by the model. The second is obtained by lumping the failure classes 2 to 17 into a single class and computing the resulting accuracy. This means that a prediction of a data point whose ground truth is a failure class is deemed correct as long as any class from 2 to 17 is chosen, independently of whether the right class is predicted. The third is estimated by ignoring the data points whose ground truth is class 1 (the healthy state) and recomputing the accuracy of correctly classifying the failure among the remaining classes.

Table 10.3: FDD accuracy and inference time results. With a total of 17 classes, four discrete and abrupt failure modes were simulated for the Bebop 2 UAV per propeller, namely 20%, 40%, 60% and 80% single blade damage.

Data processing	**Data fusion model**	**Inference time** (ms)	**General accuracy** (%)	**Detection accuracy** (%)	**Diagnosis accuracy** (%)
IMU	LSTM (I3-c30)+BN	**88.20**	80.70	**99.98**	50.52
CAM	LSTM (I3-c30)+BN	240.01	95.93	98.53	89.94
CAM+IMU	LSTM (I3-c30)+BN	250.47	**99.55**	**99.98**	**98.86**
	Dense (I3-c128)+BN	241.77	93.56	**99.98**	83.49

Additionally, the same metrics of modified versions of the pipeline are also presented in order to demonstrate the added value of each of its components. The ”Data processing” column stands for the active branches of the network, where CAM and IMU are networks with only the camera or the IMU paths active. LSTM (I3-c30)+BN is the data fusion architecture explained in subsection 10.1.3, whereas Dense(I3-c128)+BN is an alternative approach where the temporal relationships of the data are ignored by substituting the LSTM network with a three-layer dense NN with 128 neurons per layer.

Even though the IMU-only network feeds the sequential model with 46% more features than the camera-only network, as can be seen in Figure 10.1, the latter shows an overwhelming superiority in the diagnosis of the failures with a 39.42% difference in accuracy. The reason behind that difference can be observed in Figure 10.6; its confusion matrix of the predicted and true failure modes. The IMU-only network systematically confuses the front right (FR) and front left propellers (FL), as well as the back right (BR) and back left (BL). However, despite being unable to identify the failed propeller, it is able to infer the correct degree of damage. This is shown by the parallel diagonals three cells apart.

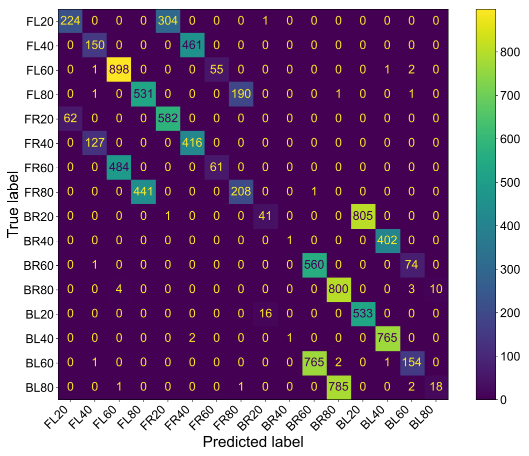


Figure 10.6: IMU-only LSTM model confusion matrix of the failure modes.

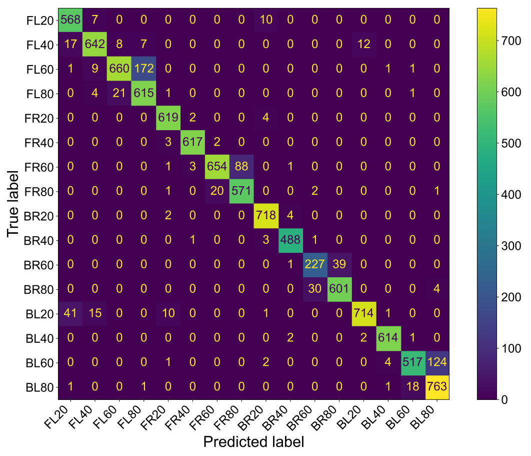


Figure 10.7: Camera-only LSTM model confusion matrix of the failure modes.

In contrast, Figure 10.7 shows that the camera-only network is able to correctly identify the failed actuator but fails to always accurately quantify the damage. Most of the incorrectly labelled predictions are one degree of damage higher or lower than the true label, but within the same actuator.

Both observations demonstrate the complementary nature of the camera and IMU sensors, which combined lead to the highest measured diagnosis accuracy of 98.86%. Figure 10.8 shows the IMU+CAM network confusion matrix with the main diagonal filled with -1's in order to visually highlight error patterns. From the multiple coloured parallel lines to the main diagonal, it can be inferred that the largest source of error originates from failing to correctly identify the damaged actuator. However, it is not constrained to the front and back propeller combinations, as it was the case for the IMU-only model.

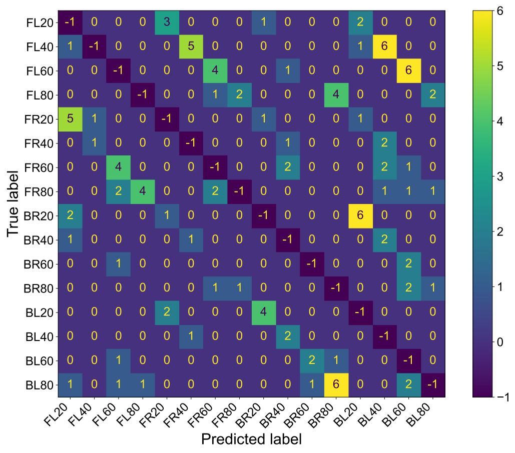


Figure 10.8: IMU+CAM LSTM model confusion matrix of the failure modes with -1's in the main diagonal.

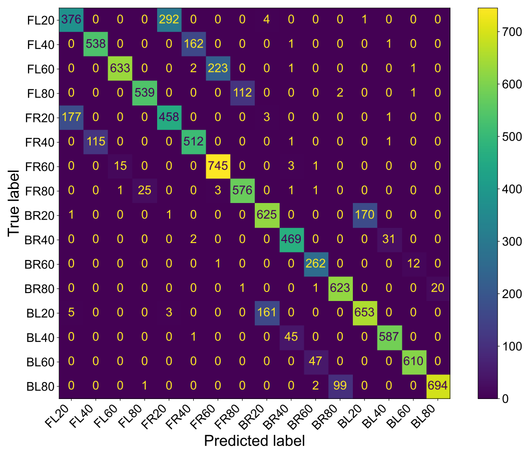


Figure 10.9: IMU+CAM Dense model confusion matrix of the failure modes.

Furthermore, the difference in diagnosis accuracy between the CAM+IMU LSTM and Dense models highlights the importance of including the data temporal relationships in the FDD framework. However, it can also be seen that this information does not play a role when detecting the presence of a failure.

From the confusion matrix of the CAM+IMU Dense model shown in Figure 10.9, the misinterpretation among the failures in the front and back propeller groups can again be seen. From this, it can be deduced that it is not the optical flow but its change that allows their decoupling. If the optical flow and the LSTM can each be considered a first derivative in time, then it is the second derivative of the camera's visual information that carries the differentiation factor between left and right actuators.

Finally, despite the success of the combined sensor approach, it has an inference time 2.84 times higher than the IMU-only approach: 8.03 ms (3.20%) for STFT, 72.30 ms (28.77%) for RAFT-S, 90.53 ms (36.03%) for MobileNetV3-S, and 80.41 ms (32.00%) for the LSTM+BN and classifier model. Further work has to be done in reducing the compute required by the camera path of the model by, for instance, developing tailored optical flow and feature extraction models. Additionally, an ablation study has to be performed on the hyper-parameters of the LSTM network.

10.3. Conclusions and recommendations

This research proposes a novel UAV actuator FDD framework that fuses for the first time camera and IMU data online with an LSTM network. The framework pre-processes the camera information by first computing its optical flow with the RAFT-S model and then extracting features with the backbone of the MobileNetV3-S model. Both are off the shelf pre-trained efficient SOTA deep neural networks. STFT is applied on the IMU signals in order to obtain time-frequency features in the form of flattened spectrograms.

To demonstrate the potential of the FDD framework, UUFOSim, developed in chapter 8, was used to generate a dataset of 5,000 flights flown in a urban environment by a Bebop 2 platform with four options of blade damage per propeller injected during flight. The drone platform was simulated using a gray-box aerodynamic model [28] from literature complemented with the Blade Element Theory blade damage model developed in chapter 9.

The IMU-only model has shown to fail to perform damage actuator identification by systematically confusing the left and right propellers, whereas the camera-only model errors are from failure magnitude quantification. When combined, they fill the gap left by each other's weaknesses. Results show the complementary nature of the IMU and camera for FDD, achieving an accuracy of 99.98% for detection and 98.86% for diagnosis on the test dataset.

The need for a model which considers the temporal relationships in sequential data was demonstrated by substituting the LSTM layers with dense neural networks that do not share information about previous inputs. This modified FDD model led to a decrease in accuracy by 15.37 percentage points without any gain in inference time.

Despite the high accuracy of the proposed vision-based FDD framework, it has an inference time of 250 ms, 2.84 times higher than the IMU-only alternative model. This observation calls for further work on the reduction of the computations required to process the camera data by developing tailored optical flow and feature extraction models for the task. Optical flow ground truth images can be retrieved from UUFOSim for the training and testing of an in-house optical flow model. MobileNetV3-S could be further reduced in size by progressively removing the last layers and unfreezing its weights for fine-tuning. Alternatively, it should be investigated whether the current camera pipeline could be substituted by a sparse optical flow approach (e.g. Lucas-Kanade [184]) followed by two histograms, one for the magnitude and another for the direction of the sparse optical flow vectors. The number of the bucket with the highest count for each histogram would be fed to the sensor fusion. Since this work has shown that the main contribution of the camera is the identification of the failed actuator, it may be the case that only the vector direction histogram would be necessary. Moreover, an ablation study should be performed on the hyper-parameters of the LSTM network, which could lead to a reduction in layers and/or cells. The authors also expect the rise of compute power available by the time UAVs and UAM concepts are introduced in urban environments.

Future work includes the study of a probabilistic classifier, such as a Bayesian NN, in order to provide a degree of confidence besides a prediction, as well as improving the explainability of the black-box model. The potential of other architectures that ingest sequential (image) data, such as Convolutional LSTMs and lightweight attention-based machine learning approaches should also be considered. Another alley of investigation is the substitution of MobileNetV3-S by an image Fourier Transform as a more efficient feature extractor. Furthermore, atmospheric turbulence models should be implemented within the simulator in order to assess the robustness of the FDD approach to external disturbances; they could induce a similar initial UAV motion as actuator failure. Additionally, a hybrid dataset could be built which combines large quantities of synthetic UUFOSim data with a smaller real world dataset in order to reduce the reality gap. Data from multiple drones could be collected in order to make the FDD framework platform agnostic. Finally, the proposed framework should be implemented on a real Bebop 2 platform to validate the results.

To conclude, the proposed framework demonstrates the potential of including the UAV on-board camera for online failure detection and diagnosis. The author hopes that it will open the doors to the development of new approaches that exploit the potential of this sensor for making future drones more resilient to failures.

V

Thesis conclusions and recommendations

11 Conclusions

The future introduction of UAV and UAM concepts in cities is strongly coupled to the advances in safety of these systems. To achieve this, actuator Fault Detection and Diagnosis (FDD) is a subfield of control engineering which improves the vehicle's self-awareness by detecting, identifying and quantifying on-board failures. It empowers the system to take informed decisions when counteracting a failure; critical situation in which it must decide when and whether to deploy safety emergency systems, such as a parachute, switching between controllers or loading different internal physics models. FDD literature is extensive but it has been constrained to the analysis of signals from the Inertial Measurement Unit (IMU) or external sensors that add weight and complexity to the system. Even though cameras are nowadays ubiquitous in commercial UAVs and their information is already processed for navigation and state estimation, they have been ignored for this task. The main bottleneck for the implementation of knowledge-based FDD approaches that are fed on rich visual information is the nonexistence of a dataset that captures UAV sensor data with mid-flight injected failures. To fill the identified research gaps, a three-step research project was conducted with the aim of assessing whether vision-based information enhances the FDD performance. Its stages can be visualised in Figure 11.1.

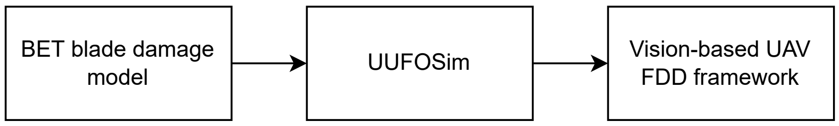


Figure 11.1: Research project stages

First, a white-box blade damage model based on Blade Element Theory was developed which combines the effects caused by the shift of the centres of gravity and pressure. Based on the vehicle's linear and angular velocities, as well as the propeller rotational speed, it predicts the forces and moments caused by the lost blade elements. In contrast with alternative approaches found in literature, the proposed method does not require expensive experimental campaigns for blade damage modelling, enables the simulation of any blade damage degree and it can be used as a plug-in to the healthy vehicle model, extending its range of operation to damaged cases. As a side contribution, 2D airfoil aerodynamic properties can be identified with the presented BET model — information currently unavailable for most off-the-shelf UAVs.

To validate the proposed methodology, a dedicated wind tunnel experimental campaign was performed in the Open Jet Facility at TU Delft, where a Bebop 2 propeller was mounted to a static test stand. Its forces and moments were compared to those predicted by the BET model at various conditions of wind velocity magnitude and direction, degree of blade damage and propeller rotational speed. The 2D aerodynamic properties of the Bebop propeller were identified using data from the gray-box aerodynamic model [244], previously identified by the department for the same platform.

Results without blade damage show a similar performance to that of the gray-box model, which decreases with lower propeller rotational rate (ω) and higher wind speed (V_∞). This shows that both approaches struggle to correctly model blade sections under a negative angle of attack; phenomenon mostly emergent in those ω-V_∞ conditions for blade sections close to the propeller hub.

Furthermore, predicted thrust and torque signals by both models show a bias with respect to the validation data, which points to the existence of unmodelled physical effects. Additionally, the gray-box model shows low performance at high wind speeds. This behaviour is unexpected as the model is reported to be identified with wind tunnel data gathered at speeds up to 14 m/s; value higher than the maximum 12 m/s used in the present research.

Moreover, the BET model architecture has a stronger physical foundation for torque than for thrust. Despite being trained with data collected at low wind speed, the torque is able to perform better at those speeds that the model had not seen before during training when compared to the thrust.

In the presence of blade damage, it is necessary to validate the signal bias, as well as the amplitude of the damage induced oscillations. When validating the bias, BET's performance at high propeller rotational speeds was similar to that without blade damage, with differences in relative error oscillating between 5% and 24%. However, the errors at low propeller rotational speeds were more than 3 times higher; oscillating between 15% and 75%. Unfortunately, the oscillations' amplitude could not be validated due to limitations in the experimental campaign, such as the sensor noise of the load cell, the low sampling rate of wrench signals by the test stand and the observed resonance in the test set-up.

Second, a high-fidelity photo-realistic UAV simulator built in Unreal Engine 4 (UE4) on AirSim, called UUFOSim (Unreal UAV Failure injectiOn Simulator), was proposed. It is based on the collection of synthetic sensor data from drone flights in a urban UE4 environment. For that purpose, the UUFOSim iterates over three stages. First, it generates an occupancy map of the environment. Second, it proposes an A* navigation strategy which avoids the obstacles of the environment. Third, it executes the flight, collects data and injects a failure. It is the first simulator that allows the collection of multi-sensor UAV flight data with mid-flight actuator injected failures programmatically.

UUFOSim was used to collect a dataset of 5,000 flights of the Bebop 2 drone with different degrees of blade damage. To this end, the aforementioned gray-box aerodynamic model, complemented with the developed BET blade damage model, simulated the platform physics. To the author's knowledge, UUFOSim generated the first synthetic dataset in literature for the training and testing of UAV actuator FDD approaches. Such data is of superior quality when compared to alternative simulation environments, such as Gazebo, allowing the development of applications with a reduced reality gap.

Third, to achieve the main goal of this research, a novel UAV actuator FDD framework that fuses for the first time camera and IMU data online with an LSTM network was proposed. The framework pre-processes the camera information by first computing its optical flow with the RAFT-S model and then extracting features with the backbone of the MobileNetV3-S model. Both are off the shelf pre-trained efficient state-of-the-art deep neural networks. IMU signals were pre-processed by applying a Short-Time Fourier Transform to obtain time-frequency features in the form of flattened spectrograms.

The dataset collected with UUFOSim was used for training and testing the FDD framework. The IMU-only variant of the model has shown to fail to perform damage actuator identification by systematically confusing the left and right propellers, whereas the camera-only variant errors are from failure magnitude quantification. When combined, they fill the gap left by each other's weaknesses. These observations show the complementary nature of the IMU and camera for FDD, which together are able to achieve an accuracy of 99.98% for detection and 98.86% for diagnosis on the test dataset. In contrast, the IMU-only and camera-only models achieve a diagnosis accuracy of 50.52% and 89.94%, respectively. Additionally, the need for a model which considers the temporal relationships in sequential data was demonstrated by substituting the LSTM layers with dense neural networks that do not share information about previous inputs. This modified FDD model led to a decrease in diagnosis accuracy by 15.37 percentage points without any gain in inference time.

Despite the high accuracy of the proposed vision-based FDD framework, it has an inference time of 250 ms, 2.84 times higher than the IMU-only alternative model. This observation calls for further work on the reduction of the computations required to process the camera data by developing tailored optical flow and feature extraction models for the task; these components currently account for more than 64% of the inference time. The author also expects the rise of compute power available by the time UAVs and UAM concepts are introduced in urban environments.

To conclude, the results from the three-step research project demonstrate that FDD approaches can be benefited from vision-based information, highlighting the complementary nature of the camera and IMU sensors. Whereas the IMU excels in damage quantification, the camera is superior in damage identification. Besides that, this work also contributes to the UAV research community by providing two new tools: the BET blade damage model and UUFOSim. The developed ”plug-in” BET model with its future work aspires to become an indispensable cost-effective tool for researchers when designing and testing their work to build more resilient UAVs against blade damage in a wide range of fields, from fault detection and diagnosis to fault-tolerant control. Also, the author hopes that UUFOSim will help the research community to build benchmarks that will assist in the tracking of the future progress of UAV FDD.

12 Recommendations for future work

First, multiple assumptions were highlighted when developing the BET blade damage model which could be considered as points of improvement; refinement of the model would contribute to its generalisation and application to different propeller types. Here, only the recommendations derived from geometrical and aerodynamic assumptions will be repeated. The reader is encouraged to review subsection 9.4.5 for a comprehensive list of assumptions and recommendations.

The geometrical assumptions, such as the discretisation of the propeller in trapezoids, the linearly decreasing twist and the constant airfoil throughout the blade, could be eliminated by creating a 3D model (digital twin) of the propeller using scanning technologies that probe the propeller through physical touch (with contact), such as Coordinate Measuring Machines [259], or scanning technologies that exploit acoustic, optical or magnetic approaches (without contact), such as laser scanning, structured light or photogrammetry [260, 261] (e.g. structure from motion). If translated to a CAD model, this would allow the computation of the twist, chord and volume of each blade section, the latter being used for the computation of the centre of gravity when the density of the material is known. Additionally, such model would contribute to the potential discovery of multiple airfoils present in the blade.

The aerodynamic assumptions that should be revisited first are the neglect of the aeroelasticity effects and the blade root and tip losses. Drone rotors lack an articulated head, causing their material to bend and the rotor to tilt with the possibility of flapping [263]. For that purpose, the field of wind energy could be explored since Blade Element Momentum Theory approaches have been used as the aerodynamic component of aeroelastic models [264]. Regarding the blade root and tip losses, the circulation must be equal to zero at those locations, and at the tip there is an additional reduction of lift due to the appearance of tip vortices. In the field of wind energy [265], these effects are taken into account by multiplying the induced velocity with a correction factor that is a function of the distance to the centre of rotation (r). This factor would acquire a value of 1 in the centre of the blade and a value of 0 at the edges, allowing the induced velocity to fall to zero at the blade edges. Alternatively, previous literature [29, 265, 266] has also proposed the Prandtl tip-loss factor approximation (B=0.95-0.98) to compute the effective blade radius (R_e = BR) and account for the loss of blade lift. As a result, the outer portion of the blade (R-R_e) is considered to be incapable of carrying lift. Given that in helicopter aerodynamics the introduction of the tip loss factor can cause rotor thrust reductions between 6-10% [29], the study of its implementation in drone propellers is recommended for the further improvement of the BET thrust and torque predictions.

Furthermore, due to the challenges encountered during the BET model validation, the author can only recommend the in-house design of a test stand for the measurement of wrenches of partially damaged propellers. Such a stand would require a sampling frequency above 100 Hz for measurements carried out at the minimum rotational speed of 300 rad/s (or 350 Hz for ω of 1100 rad/s) and a dampening system which prevents resonance with the rest of the structure and the testing platform. Especially designed load cells have to be used capable of withstanding at least 6g of sustained vibrations. In addition, the characterization of the complete test set-up in order to identify its dynamics could contribute to the removal of the resonance present in the measurements. Knowledge about the complete system behaviour would allow a deeper understanding of the measured signals and the separation of the set-up dynamics from the ”pure” thrust and torque oscillations caused by the damaged propeller. The author hopes that the outlined recommendations will serve as basis in the design of a future experimental campaign with more specialised hardware.

As final recommendation for future work applied to the BET model, the 2D aerodynamic properties of the propeller could be identified with real drone data instead of an existing aerodynamic model. This would prevent that the errors and assumptions in the aerodynamic model propagate to the BET model, easing the validation analysis.

Second, the rate at which camera data was sampled from UUFOSim show a great spread due to its dependence on the AirSim image retrieval functions. The author recommends the in-house development of an image capture module which allows its coupling with the physics engine. This would lead to a similar behaviour as observed with the IMU. Further work with UUFOSim would be its implementation in a computer cluster in order to deploy multiple UFOs (UAVs) simultaneously with multi-threading and decrease the time required for data gathering. Additionally, a benchmark dataset and web platform could be built in order to track the progress of future FDD approaches and foster research in this field, similar to how it is done with the Sintel and KITTI datasets.

Third, from the high inference time required by the proposed FDD framework, the computations applied on camera information should be reduced. Optical flow ground truth images can be retrieved from UUFOSim for the training and testing of an in-house optical flow model. MobileNetV3-S could be further reduced in size by progressively removing the last layers and unfreezing its weights for fine-tuning. Alternatively, it should be investigated whether the current camera pipeline could be substituted by a sparse optical flow approach (e.g. Lucas-Kanade [184]) followed by two histograms, one for the magnitude and another for the direction of the sparse optical flow vectors. The number of the bucket with the highest count for each histogram would be fed to the sensor fusion. Since this work has shown that the main contribution of the camera is the identification of the failed actuator, it may be the case that only the vector direction histogram would be necessary. Moreover, an ablation study should be performed on the hyper-parameters of the LSTM network, which could lead to a reduction in layers and/or cells.

Other future work applied to the FDD framework includes the study of a probabilistic classifier, such as a Bayesian NN, in order to provide a degree of confidence besides a prediction, as well as improving the explainability of the black-box model. The potential of other architectures that ingest sequential (image) data, such as Convolutional LSTMs and lightweight attention-based machine learning approaches, should also be considered. Another alley of investigation is the substitution of MobileNetV3-S by an image Fourier Transform as a more efficient feature extractor. Furthermore, atmospheric turbulence models should be implemented within the simulator in order to assess the robustness of the FDD approach to external disturbances; they could induce a similar initial UAV motion as actuator failure. Additionally, a hybrid dataset could be built which combines large quantities of synthetic UUFOSim data with a smaller real world dataset in order to reduce the reality gap. Data from multiple drones could be collected in order to make the FDD framework platform agnostic. Finally, the proposed framework should be implemented on a real Bebop 2 platform to validate the results.

VI

Appendices

A

Induced velocity computation: gradient-descent approach

As discussed in subsection 9.2.1, the computation of the uniform induced velocity can not be solved analytically and requires a numerical approach. However, this computation needs to happen in every time step of the UE4 simulation after the drone has suffered blade damage in one of its propellers; hence, the efficiency of this optimisation is of paramount importance for the gathering of large quantities of simulation data. To this end, the goal of this chapter is to check the possibility of using a gradient-descent approach with a low computational load.

In order to define the optimisation problem objective function, the alternative definition of the airspeed at the rotor (V_R) of Equation A.1 will be used in conjunction with the Glauert formula presented in Equation 9.19. In contrast with Equation 9.20, the new definition of V_R translates the 3 components of the linear velocity of the propeller assembly ($\vec{V}^P$) into 2 components, namely its magnitude (V) and the angle of attack of the rotor disk relative to the oncoming flow (α_d). The latter is illustrated in Figure A.1.

$$V_R = \sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2} \tag{A.1}$$

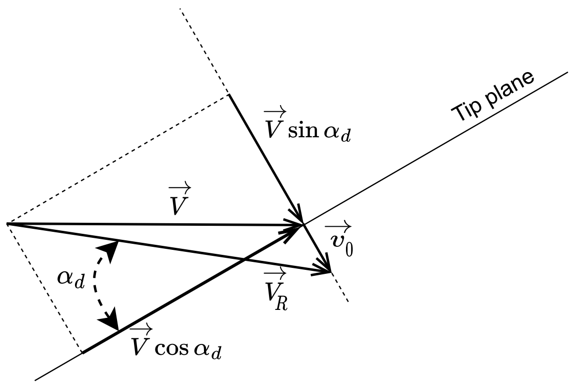


Figure A.1: Angle of attack of the rotor relative to the oncoming flow.

The optimisation problem objective function can be defined as described in Equation A.2, which is the same as finding the location where the function $f(v_0)$ intersects the x-axis.

$$\min_{v_0} \quad |f(v_0)| = \left|T - 2\rho\pi R^2 v_0\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2}\right| \tag{A.2}$$

Gradient-descent methods are used in optimization for finding the local minimum of a differentiable function by traversing the solution space in the opposite direction of the function gradient, also known as the direction of steepest descend. In the case of the present objective function, local minima will be found where the derivative of $f(v_0)$ with respect to v_0 is zero and where $f(v_0) = 0$. In the case that it can be proven that the function $f(v_0)$ is strictly monotonic, meaning that it only increases or decreases, then $f(v_0)$ will not have local minima and it will be zero at a single value of v_0. Then, there exists a single (global) minimum in the objective function and a gradient-descent approach could be used to find it. Given the definition of the Glauert formula (Equation 9.19), the uniform induced velocity can only have a positive value. Hence, it is only required to prove the strict monotonocity for v_0 values in the half-open interval [0, ∞).

Equation A.3 shows the derivative of $f(v_0)$ with respect to v_0 and Equation A.4 shows the uniform induced velocity values that make it zero. As can be seen, $f(v_0)$ has one or two optima when $9\sin^2\alpha_d - 8 \geq 0$. Since the uniform induced velocity can only be positive, the only interesting solution comes from negative α_d angles, ergo when the condition in Equation A.5 is met. When the angle of attack of the rotating disk is higher than $\arcsin -2\sqrt{2}/3$, the function is strictly monotone and gradient-descent would be able to find the global minimum.

$$\frac{df(v_0)}{dv_0} = -2\rho\pi R^2\left(\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2} + v_0\frac{V\sin\alpha_d + v_0}{\sqrt{(V\cos\alpha_d)^2 + (V\sin\alpha_d + v_0)^2}}\right) \quad \text{(A.3)}$$

$$v_0 = \frac{V}{4}\left(-3\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right) \quad \text{(A.4)}$$

$$\sin\alpha_d \leq -\frac{2\sqrt{2}}{3} \quad \text{(A.5)}$$

During nominal flight, the drone will experience a positive α_d when in cruise. However, in the case of failure, when the induced velocity has to be computed, the drone could pitch or roll excessively causing the air flow to impact the propeller from below. Hence, it is important to consider the presence of the discovered local minima. Even though the function can contain local minima, they could be avoided by a proper selection of hyper-parameters and initialisation of the optimisation; tuning the gradient-descent to the particular (known) function.

Considering extreme values of v_0, it can be observed in Equation A.2 that the second term of $f(v_0)$ is dominant. As a result, $f(-\infty)$ leads to a positive value and $f(\infty)$ to a negative one; the function is decreasing in value independently of α_d. In the case that there is a local minimum (Equation A.5 is fulfilled) and it takes place at a lower uniform induced velocity than when $f(v_0) = 0$, as illustrated in Figure A.2, the gradient-descent could be initialised with a high value of v_0 to guarantee that the optimisation will encounter the global optimum before the local minimum. Since the function is decreasing, this approach would not work if $f(v_0) < 0$ at $df(v_0)/dv_0 = 0$, as can be seen in Figure A.3. In order to check whether this latter scenario exists, Equation A.4 is inserted in $f(v_0)$, leading to Equation A.6.

$$T - \frac{\rho\pi R^2V^2}{2}\left(-3\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right)\sqrt{\cos^2\alpha_d + \frac{1}{16}\left(\sin\alpha_d \pm \sqrt{9\sin^2\alpha_d - 8}\right)^2} \leq 0 \quad \text{(A.6)}$$

Since the local minimum can only be found when $\sin\alpha_d \in [-1, -2\sqrt{2}/3]$ (Equation A.5 is met), the two limits of this range are inserted in Equation A.6, resulting in the two conditions presented in Equation A.7 and Equation A.8. Observing both conditions, the second one is automatically met when the first one is fulfilled. Hence only when Equation A.5 and Equation A.7 are met, there is a local minimum which takes place with a higher uniform induced velocity than when $f(v_0) = 0$. In that case, initialising the optimisation with a high value of v_0 would most likely not converge to the undesired local minimum.

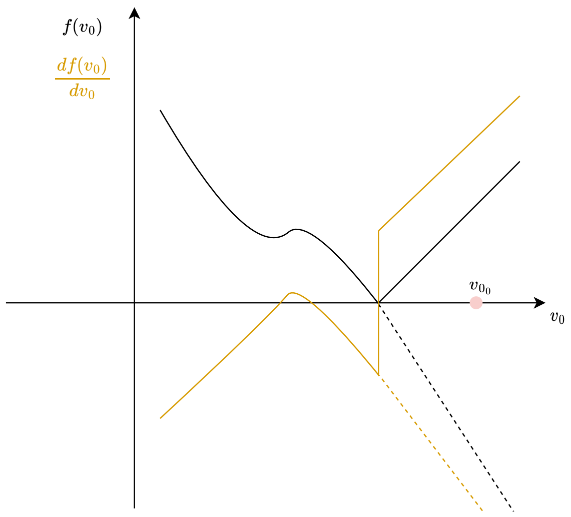


Figure A.2: **Desired scenario**: local minima takes place at lower induced velocity than global minima, so gradient descend will reach global minima first. The dotted line corresponds to $f(v_0)$, whereas the bold line to $|f(v_0)|$.

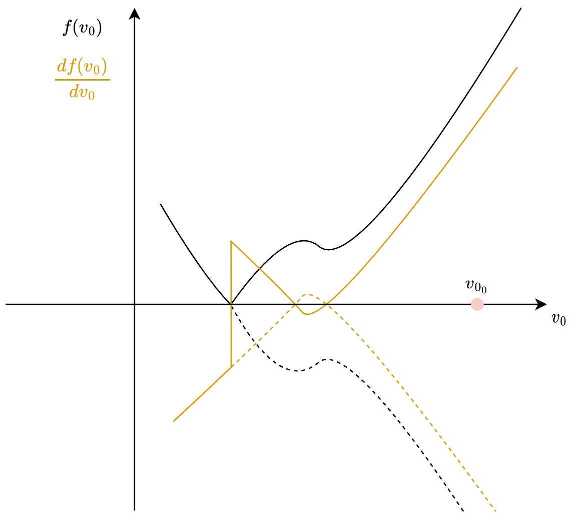


Figure A.3: **Undesired scenario**: local minima takes place at higher induced velocity than global minima, so gradient descend will reach local minima first. The dotted line corresponds to $f(v_0)$, whereas the bold line to $|f(v_0)|$.

$$\sin \alpha_d = -\frac{2\sqrt{2}}{3}, \quad T < \frac{\sqrt{3}}{3}\rho\pi R^2 V^2 \tag{A.7}$$

$$\sin \alpha_d = -1, \quad T < \frac{1}{2}\rho\pi R^2 V^2 \tag{A.8}$$

To check whether the blade damage simulation with the Bebop 2 model would encounter scenarios in which both conditions (Equation A.5 and Equation A.7) are met, 100,000 scenarios are run with the following set of conditions: $\vec{V}_x^B \in [-3,3]$, $\vec{V}_y^B = 0$, $\vec{V}_z^B \in [-3,3]$, $\vec{\Omega} = \vec{0}$ and $\omega \in [300,1256]$. When compared to those nominal conditions presented in subsection 9.2.3, here the body velocity in the z-direction can also acquire positive values and its absolute value is higher in order to account for the failure cases.Figure A.4 shows 100,000 points representing all the scenarios in a plot of vehicle velocity over the thrust (V-T). As can be observed, there is not overlap between the set of points that fulfil the first condition in Equation A.5 (the pink points within the convex hull) and those that meet the second condition in Equation A.7 (the green points under the dashed line). As a result, for the simulations of the present research, only the desired scenario illustrated in Figure A.2 will be observed. Hence, if the gradient-descent is initialised with a high positive value of v_0, it will always encounter the global minimum first.

Now that it has been proven to be beneficial to initialise the gradient-descent with a high positive value of uniform induced velocity, the question is what should be the exact initialisation value. Observing Figure A.2, it is enough to initialise the gradient-descent with a uniform induced velocity value higher than the maximum v_0 that the local minimum could have. If initialised between the local and global minimum, the gradient-descent will move the solution towards the global optimum to the right along the v_0 axis. If initialised to the right of the global minimum, the gradient-descent will move the solution towards the global optimum to the left along the v_0 axis.

Given Equation A.4, the maximum v_0 that the local minimum could have is found when the output of the square root is positive and α_d has a value of -90°; then, the uniform induced flow equals the incoming flow velocity ($v_0 = V$). From the 100,000 scenarios presented in Figure A.4, the maximum velocity observed is 4.24 m/s. Therefore, if the gradient-descent optimisation is initialised with v_{0_0}=4.5 m/s, the initial function evaluation will always be carried out to the right of the local optima.

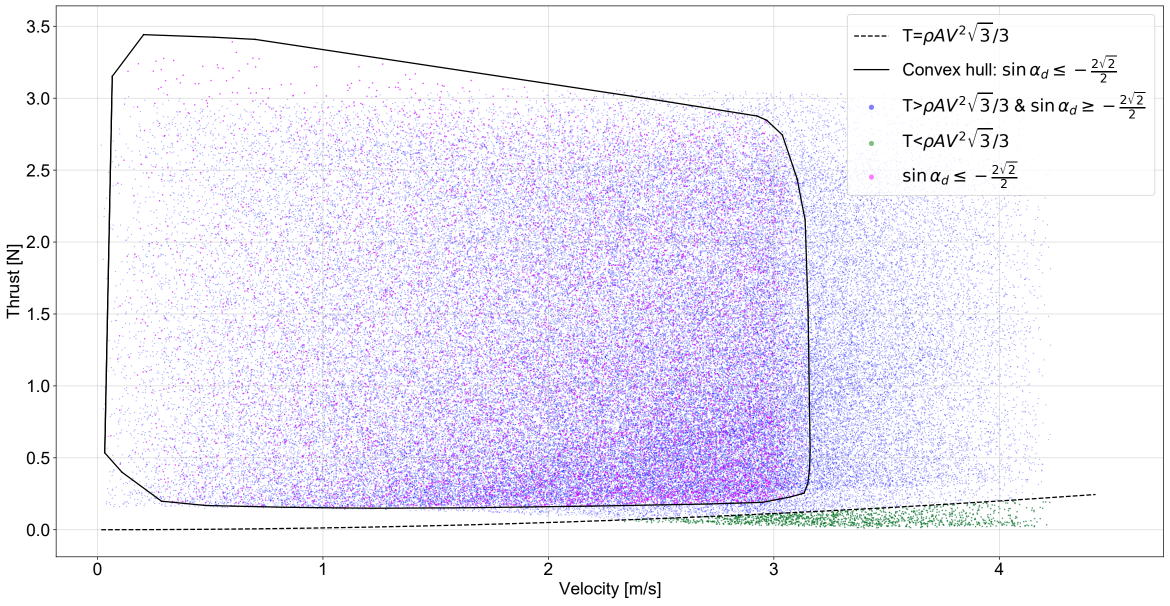


Figure A.4: V-T graph for 100,000 simulation scenarios of the Bebop 2 drone. The convex hull encapsulates all the pink points that meet (first) the condition in Equation A.5. The green points under the dashed line meet the (second) condition outlined in Equation A.7. The magenta points are those scenarios in which neither of the conditions are met. From the figure, there is no overlap between both conditions sets.

Furthermore, the gradient-descent optimisation requires the selection of the learning rate (γ). This hyper-parameter needs to be carefully chosen in order to avoid overshooting the global optimum and landing in the local minimum. Given the update law of the gradient-descent provided by Equation A.9, the algorithm can overshoot the global optimum by a value equal to $\gamma\frac{d|f(v_0)|}{dv_0}$. Hence, the smaller this update step, the lower the probability that the optimisation overshoots the global optimum and lands in the local minimum.

$$v_{0_{i+1}} = v_{0_i} - \gamma\frac{d|f(v_{0_i})|}{dv_0} = v_{0_i} - \gamma\frac{df(v_{0_i})}{dv_0}\frac{f(v_{0_i})}{|f(v_{0_i})|} \tag{A.9}$$

For the current research, two values are considered for the learning rate, namely 0.5 and 0.1. Additionally, the value of the learning rate is decreased by a factor of 0.5 every time $\frac{d|f(v_0)|}{dv_0}$ changes sign. Using the same 100,000 simulation scenarios of the Bebop 2 drone presented in Figure A.4, the performance results are observed in Table A.1. An optimisation is defined as successful when its output is fed to $f(v_0)$ and the result is less than 10^{-5}. As can be seen, the gradient-descent approach has a 100% success rate for both learning rates, in contrast with the slight worse performance of 98.56% for Nelder-Mead. Additionally, the gradient-descent optimisation shows an approximately 20% and 55% computational time reduction with respect to the Nelder-Mead alternative for the 0.1 and 0.5 learning rates, respectively. Hence, the chosen learning rate for the present research is 0.5.

Table A.1: Performance results of Nelder-Mead and Gradient-Descent with learning rate values of 0.1 and 0.5.

	Success rate [%]	Time per scenario [ms]
Nelder-Mead	98.56	5.51
Gradient-descent (γ=0.1)	100	4.39
Gradient-descent (γ=0.5)	**100**	**2.45**

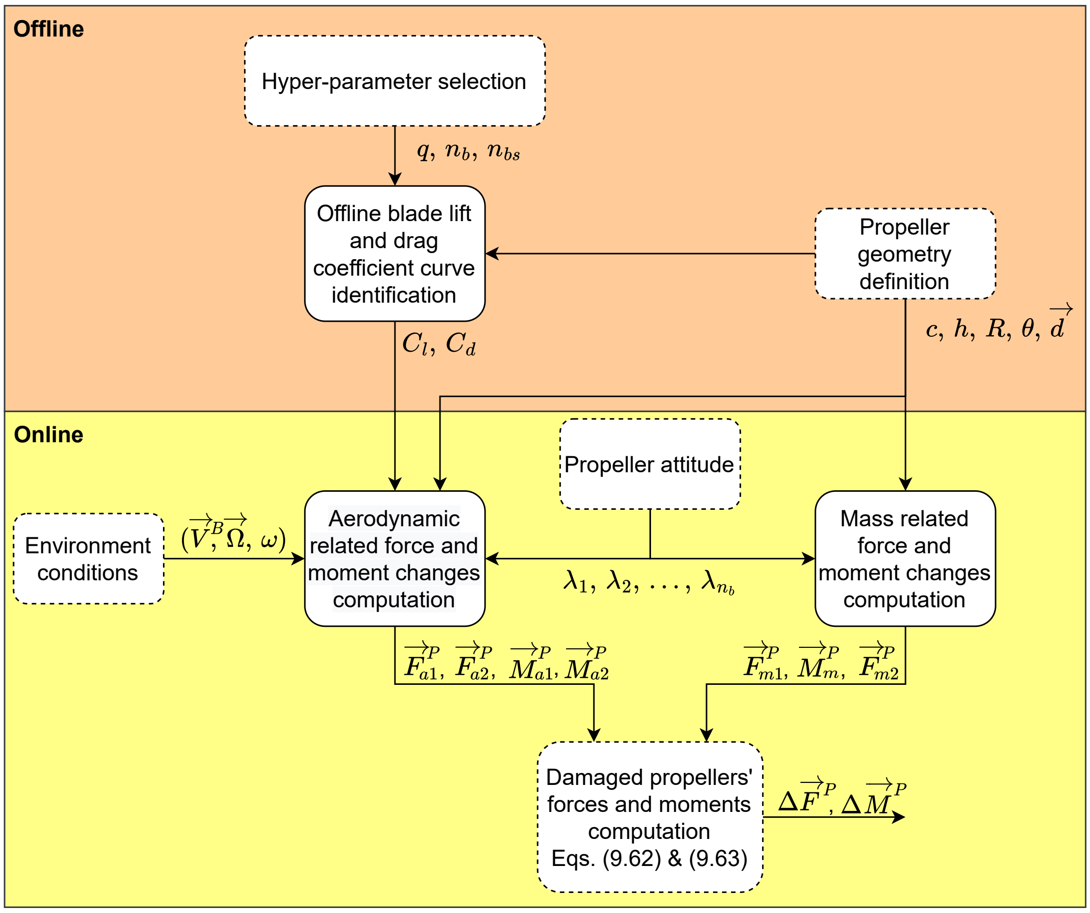
Finally, algorithm 1 shows the complete pseudo-code used for the simple implementation of the presented gradient-descent variant. No gradient-descent optimisation modifications, such as Momentum, RMSprop or Adam, were considered since they would increase the chance of overshooting the global minimum; especially due to the sharp discontinuity in the derivative $\frac{df(v_0)}{dv_0}$ observed in Figure A.2. As can be seen, the optimisation concludes when either of the following 3 conditions is met:

1. The maximum number of iterations is reached (i_{max}=10,000).
2. The change in the solution is lower than a threshold ($m < 0.01$) for a certain number of iterations ($c_{max} = 20$).
3. The denominator of the gradient update ($|f(v_0)|$) is less than a very small value ($\epsilon = 10^{-10}$) because then the solution has been found.

Algorithm 1: Gradient-descent variant

```
Input  : The derivative function $\frac{d|f(v_0)|}{dv_0}$, the initialisation $v_{0_0}$ and the function $|f(v_0)|$.
Output: The uniform induced velocity $(v_0^*)$.
// Initialise the parameter $x_i$, the previous gradient $y_{i-1}$ and the counter $c$
$x_i \leftarrow v_{0_0}$;
$y_{i-1} \leftarrow \frac{d|f(x_i)|}{dv_0}$;
$c \leftarrow 0$;
while $i < i_{max}$ do
    // Check that the optimum has not already been found
    if $|f(x_i)| < \epsilon$ then
        return $x_i$;
    end if

    // Compute gradient at current parameter value and apply gradient update
    $y_i \leftarrow \frac{d|f(x_i)|}{dv_0}$;
    $x_{i+1} \leftarrow x_i - \gamma y_i$;

    // Check whether the global minimum has been overshot and reduce learning
       rate
    if $y_i \neq 0$ then
        if $y_{i-1}/y_i < 0$ then
            $\gamma \leftarrow \gamma/2$;
        end if
    end if

    // Check whether the parameter change is smaller than threshold and whether
       counter has reached maximum
    if $|x_{i+1} - x_i| < m$ then
        $c \leftarrow c + 1$;
        if $c > c_{max}$ then
            return $x_{i+1}$;
        end if
    else
        $c \leftarrow 0$;
    end if

    // Update the parameter and its gradient for the next iteration
    $x_i \leftarrow x_{i+1}$;
    $y_{i-1} \leftarrow y_i$;
end while
return $x_i$;
```

B
Propeller damage flowcharts



Offline
Hyper-parameter selection
$q,\ n_b,\ n_{bs}$
Offline blade lift and drag coefficient curve identification
Propeller geometry definition
$c,\ h,\ R,\ \theta,\ \vec{d}$
$C_l,\ C_d$
Online
Propeller attitude
Environment conditions
$(\vec{V}^B, \vec{\Omega},\ \omega)$
Aerodynamic related force and moment changes computation
Mass related force and moment changes computation
$\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_{n_b}$
$\vec{F}^P_{a1},\ \vec{F}^P_{a2},\ \vec{M}^P_{a1}, \vec{M}^P_{a2}$
$\vec{F}^P_{m1},\ \vec{M}^P_{m},\ \vec{F}^P_{m2}$
Damaged propellers' forces and moments computation Eqs. (9.62) & (9.63)
$\Delta\vec{F}^P,\ \Delta\vec{M}^P$

Figure B.1: Flowchart of the damaged propeller offline and online computations. The lift and drag coefficient curves identification takes place offline, whereas the computation of forces and moments due to propeller damage are performed online. The blocks with a solid edge line are further expanded in the next figures.

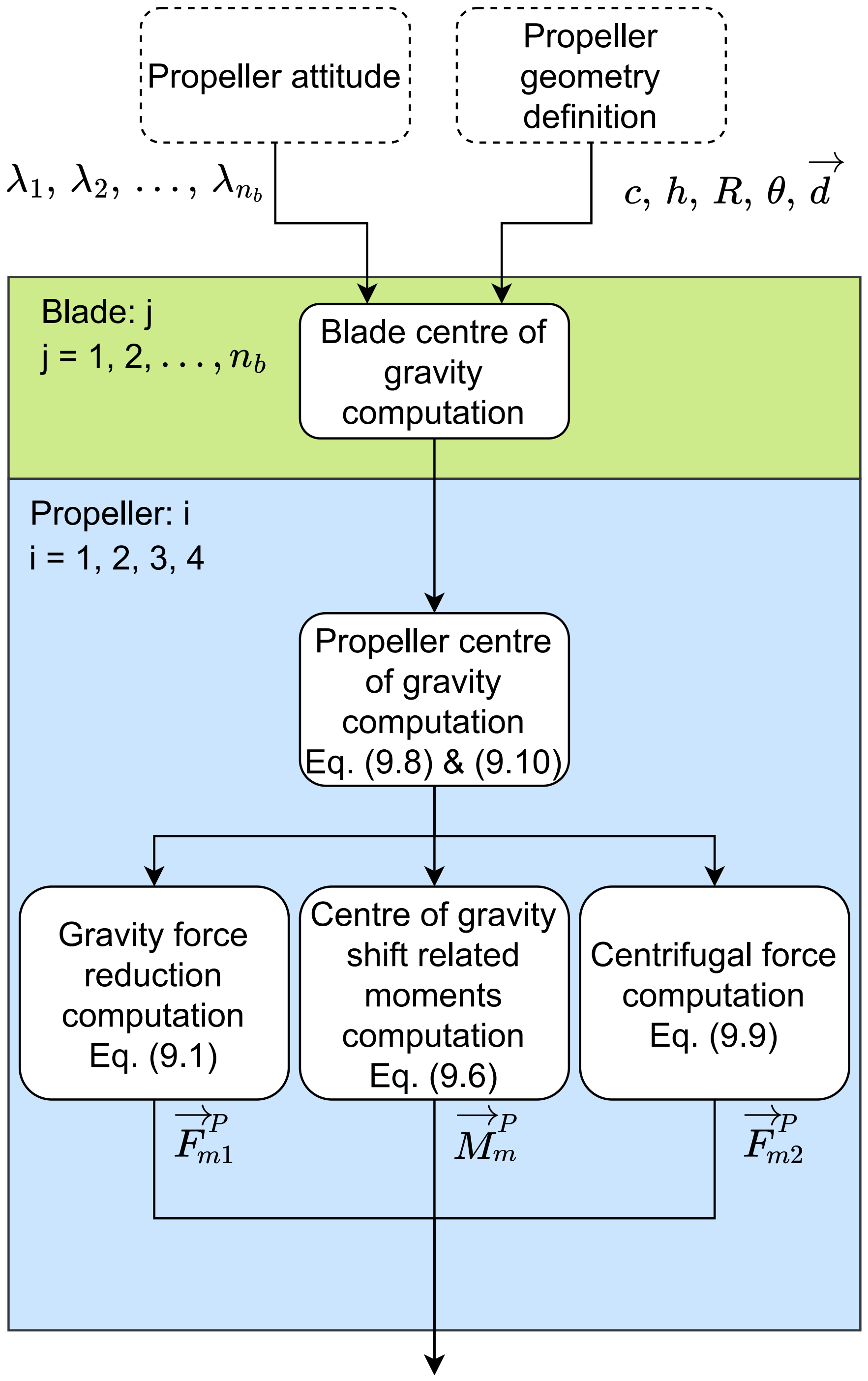
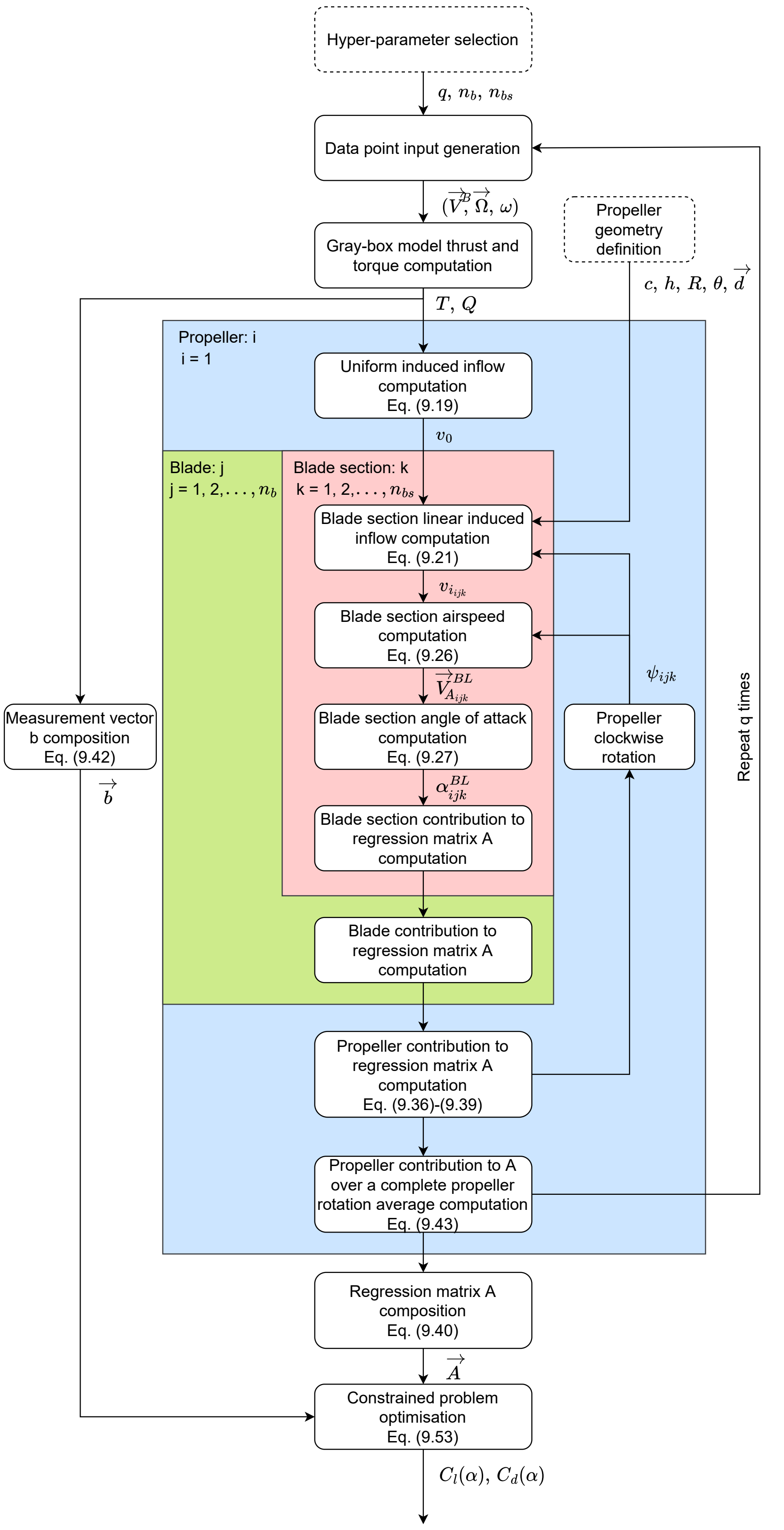

Propeller attitude
Propeller geometry definition
$\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_{n_b}$
$c,\ h,\ R,\ \theta,\ \vec{d}$
Blade: j
j = 1, 2, . . . , n_b
Blade centre of gravity computation
Propeller: i
i = 1, 2, 3, 4
Propeller centre of gravity computation Eq. (9.8) & (9.10)
Gravity force reduction computation Eq. (9.1)
Centre of gravity shift related moments computation Eq. (9.6)
Centrifugal force computation Eq. (9.9)
$\vec{F}^P_{m1}$
$\vec{M}^P_m$
$\vec{F}^P_{m2}$

Figure B.2: Flowchart of the computation of the damaged propeller mass related forces and moments at one time step during simulation.



Hyper-parameter selection
q, n_b, n_{bs}
Data point input generation
$(\vec{V}^B, \vec{\Omega}, \omega)$
Propeller geometry definition
Gray-box model thrust and torque computation
c, h, R, θ, $\vec{d}$
T, Q
Propeller: i
i = 1
Uniform induced inflow computation
Eq. (9.19)
v_0
Blade: j
j = 1, 2, . . . , n_b
Blade section: k
k = 1, 2, . . . , n_{bs}
Blade section linear induced inflow computation
Eq. (9.21)
$v_{i_{ijk}}$
Blade section airspeed computation
Eq. (9.26)
$\vec{V}^{BL}_{A_{ijk}}$
ψ_{ijk}
Repeat q times
Measurement vector b composition
Eq. (9.42)
Blade section angle of attack computation
Eq. (9.27)
Propeller clockwise rotation
$\vec{b}$
α^{BL}_{ijk}
Blade section contribution to regression matrix A computation
Blade contribution to regression matrix A computation
Propeller contribution to regression matrix A computation
Eq. (9.36)-(9.39)
Propeller contribution to A over a complete propeller rotation average computation
Eq. (9.43)
Regression matrix A composition
Eq. (9.40)
$\vec{A}$
Constrained problem optimisation
Eq. (9.53)
$C_l(\alpha)$, $C_d(\alpha)$

Figure B.3: Flowchart of the offline lift and drag coefficient curves identification.

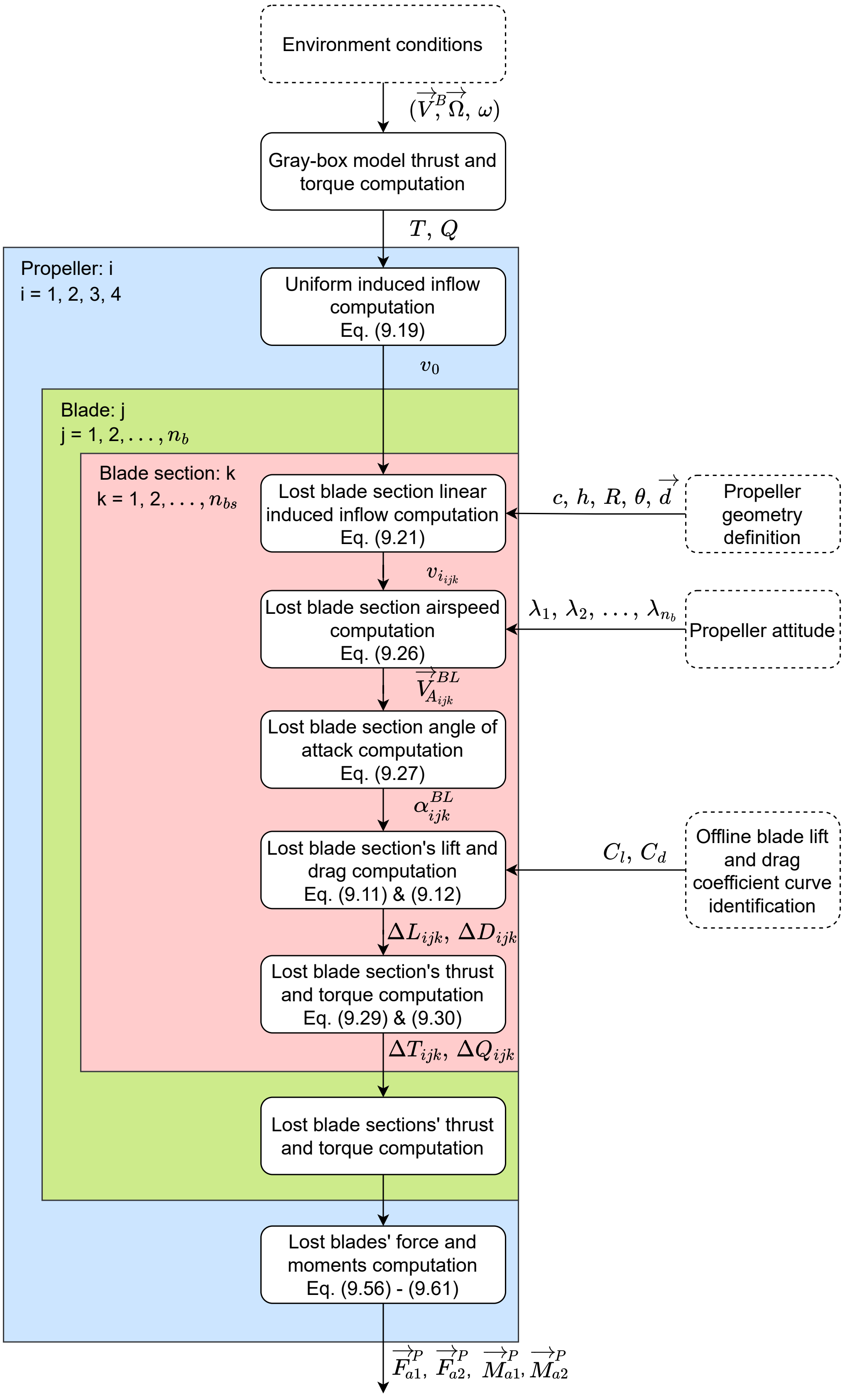

Environment conditions
$(\vec{V}^B, \vec{\Omega}, \omega)$
Gray-box model thrust and torque computation
T, Q
Propeller: i
i = 1, 2, 3, 4
Uniform induced inflow computation Eq. (9.19)
v_0
Blade: j
j = 1, 2, $\ldots, n_b$
Blade section: k
k = 1, 2, $\ldots, n_{bs}$
Lost blade section linear induced inflow computation Eq. (9.21)
$c, h, R, \theta, \vec{d}$
Propeller geometry definition
$v_{i_{ijk}}$
Lost blade section airspeed computation Eq. (9.26)
$\lambda_1, \lambda_2, \ldots, \lambda_{n_b}$
Propeller attitude
$\vec{V}^{BL}_{A_{ijk}}$
Lost blade section angle of attack computation Eq. (9.27)
α^{BL}_{ijk}
Lost blade section's lift and drag computation Eq. (9.11) & (9.12)
C_l, C_d
Offline blade lift and drag coefficient curve identification
$\Delta L_{ijk}, \Delta D_{ijk}$
Lost blade section's thrust and torque computation Eq. (9.29) & (9.30)
$\Delta T_{ijk}, \Delta Q_{ijk}$
Lost blade sections' thrust and torque computation
Lost blades' force and moments computation Eq. (9.56) - (9.61)
$\vec{F}^P_{a1}, \vec{F}^P_{a2}, \vec{M}^P_{a1}, \vec{M}^P_{a2}$

Figure B.4: Flowchart of the computation of the damaged propeller aerodynamic related forces and moments at one time step during simulation.

C

UAV linear velocity sampling strategy for simulation

When choosing the range of values for the linear drone velocity in the body coordinate frame, there are 3 variables that are inter-dependent, namely the magnitude of the linear drone velocity, the velocity component in the x-direction (V_x^B) and the velocity component in the z-direction (V_z^B). The velocity in the y-direction is always zero for identification purposes since the same information can be gathered by rotating the propeller with Equation 9.43. For the creation of input combinations ($\vec{V}^B$, $\vec{\Omega}$, ω), 3 different schemes were identified for the generation of linear drone velocity values and they are summarised in Table C.1.

Table C.1: Linear body velocity input generation schemes. *FV* refers to Fix Value, *FR* refers to Fixed Range, *VR* refers to Variable Range and *CV* means Computed Value. In all table entries, except those with *CV*, the corresponding value or range is included in the table. The scheme in bold reflects the option chosen for the present research.

	$\|\vec{V}^B\|$ [m/s]	V_z^B [m/s]	V_x^B [m/s]	Computation order
Scheme 1	FV (4)	FR ([-2.5,-0.5])	CV	$\|\vec{V}^B\| \rightarrow V_z^B \rightarrow V_x^B$
Scheme 2	VR ([$\max\left(V_z^B, 2\right)$, 4])	FR ([-2.5,-0.5])	CV	$V_z^B \rightarrow \|\vec{V}^B\| \rightarrow V_x^B$
Scheme 3	CV	FR ([-2.5,-0.5])	FR ([-3,3])	$V_z^B \rightarrow V_x^B \rightarrow \|\vec{V}^B\|$

The first scheme consists on fixing the magnitude of $|\vec{V}^B|$ to 4 m/s, selecting a value of V_z^B in the range [-2,-0.5] m/s and computing V_x^B from the relation that the sum of the squares of V_z^B and V_x^B should be equal to the square of $|\vec{V}^B|$. In the second option, the value of $|\vec{V}^B|$ was selected from the varying range $\max\left(V_z^B, 2\right)$ m/s, meaning that it was dependent on V_z^B. In the last scheme, V_x^B and V_z^B are computed independently from their own fixed ranges and the magnitude of the linear body velocity is computed with the aforementioned relation of squares. When looking at the computation order column of Table C.1, it can be observed that the main difference between schemes is the step at which $|\vec{V}^B|$ is computed.

Even though the first scheme might be the most intuitive choice — it is known from chapter 8 that the drone can not fly in simulation faster than 4 m/s — the third scheme is chosen. The reasoning behind it can be observed when comparing the thrust and torque autocorrelation plots of the 3 schemes. Figure C.1, Figure C.3 and Figure C.5 show those corresponding to the thrust, whereas Figure C.2, Figure C.4 and Figure C.6 those corresponding to the torque. Table C.2 presents the number and percentage of lags of the thrust and torque autocorrelation functions that lay outside of the 95% confidence interval. Additionally, the Durbin Watson statistic (DW) [301] is shown for the thrust and the torque. This test is used to determine whether autocorrelation is present in the identified model by analysing the autocorrelation of the first lag with Equation C.1; the lag which tends to have the highest autocorrelation value. If DW has a value close to 2, it indicates that no autocorrelation is present. The closer it approaches 0, the more evidence of positive serial correlation and the closer it approaches 4, the more evidence of negative serial correlation. When the DW is within the acceptable range [1.5-2.5], it can be stated that the residuals have relative independence according to this statistical test. Even though all schemes pass the second metric, the DW test, it is clear that the third scheme outperforms the alternatives in the first metric; the number of lags laying outside the 95% confidence bounds is considerably lower. Hence, scheme 3 was chosen for this work. Other statistical tests for analysing the presence of serial correlation considered for the present research, which lead to similar results as the DW test, are the Ljung-Box test [302] and the Breusch-Godfrey test [303, 304].

Table C.2: Number of lags that lay outside of the 95% confidence bounds for the thrust and torque for the 3 linear body velocity input generation schemes.

	Thrust # points	Thrust DW test	Torque # points	Torque DW test
Scheme 1	23,211 (72.53%)	1.92	6721 (21.00%)	1.98
Scheme 2	3303 (10.32%)	1.97	1261 (3.94%)	1.99
Scheme 3	603 (1.88%)	1.98	623 (1.94%)	2.01

$$DW = \frac{\sum_{t=1}^{N} (\epsilon_t - \epsilon_{t-1})^2}{\sum_{t=0}^{N} \epsilon_t^2} \tag{C.1}$$

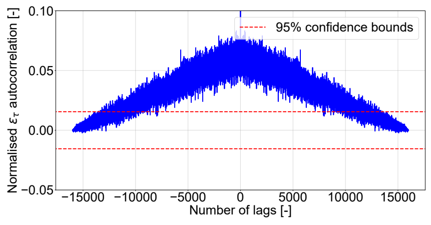


Figure C.1: Zoom-in of scheme 1 normalised thrust autocorrelation with 95% confidence bounds.

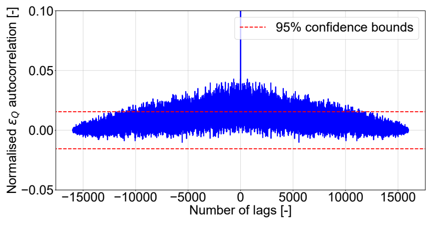


Figure C.2: Zoom-in of scheme 1 normalised torque autocorrelation with 95% confidence bounds.

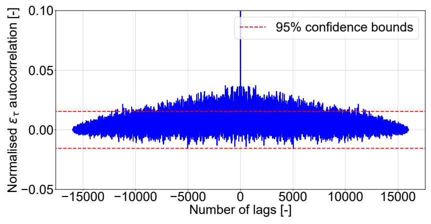


Figure C.3: Zoom-in of scheme 2 normalised thrust autocorrelation with 95% confidence bounds.

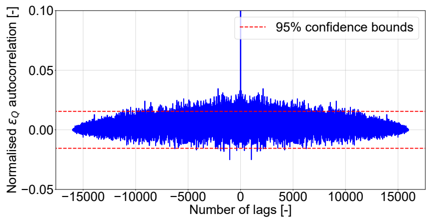


Figure C.4: Zoom-in of scheme 2 normalised torque autocorrelation with 95% confidence bounds.

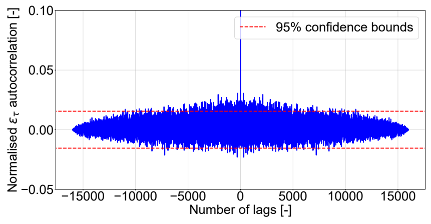


Figure C.5: Zoom-in of scheme 3 normalised thrust autocorrelation with 95% confidence bounds. Zoom-in of Figure 9.18.

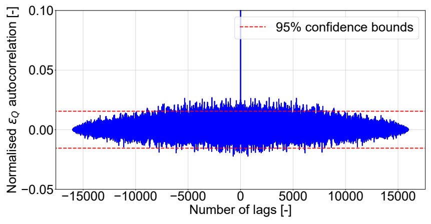


Figure C.6: Zoom-in of scheme 3 normalised torque autocorrelation with 95% confidence bounds. Zoom-in of Figure 9.19.

D
BET hyper-parameter selection

The finer the blade discretisation and the number of data points used, the closer the BET approximates reality and the better the input space is explored. At low n_{bs} and q values, the optimised polynomial coefficients of the $C_l - \alpha$ and $C_d - \alpha$ curves are very sensitive to changes in n_{bs} and q. As the value of these hyperparameters is increased, convergence of the lift and drag coefficient curves is observed. However, a higher discretisation of the blade and input space leads to higher computation times, as well as slow blade damage simulations when implemented in UE4 due to a higher number of finer blade sections. The sensitivity analysis aims at finding the lowest n_{bs} and q values at which this convergence is observed in order to minimise the computational time.

For that purpose, Figure D.1 and Figure D.2 shows a heat map of the value of the lift and drag polynomial coefficients as a function of data points and blade sections. The number of data points range from 1,000 to 100,000 in steps of 1,000, whereas the number of blade sections range from 50 to 800 in steps of 50. As can be observed, it is clear that the values greatly change when the number of samples (q) is low, but it is very difficult to assess from these heat maps the convergence of the optimisation due to the different magnitude of the identified coefficients and the range of the colour bars.

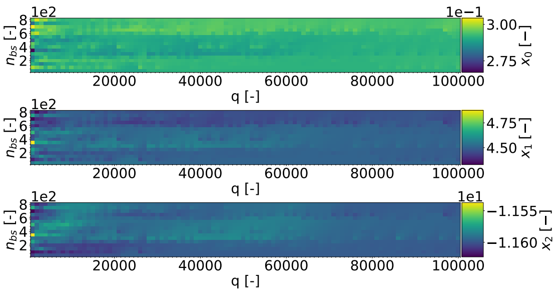


Figure D.1: Heat map with the value of the polynomial coefficients used for the lift coefficient identification (Equation 9.32).

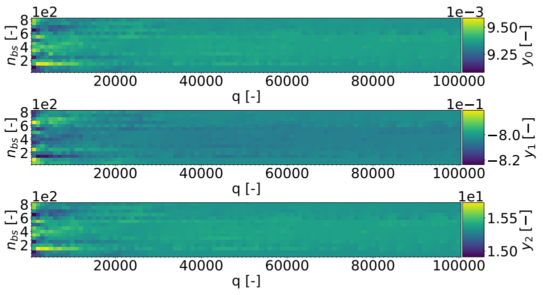


Figure D.2: Heat map with the value of the polynomial coefficients used for the drag coefficient identification (Equation 9.33).

Optimisation convergence along the q axis can also be understood as the approximation to zero of the change in value of the identified coefficients from one q value to the next. Hence, the change in the identified coefficients values when increasing the number of data points is computed. Additionally, these values are normalised with the corresponding lift/drag coefficient identified with the largest number of data points (in this case $a_{\max}$ =100,000), since it is assumed to be the most accurate value. Equation D.1 shows an example of this calculation for the first polynomial coefficient of the lift coefficient presented in Equation 9.32.

$$D_q x_{0_{\{q_a;n_{bs_b}\}}} = \frac{x_{0_{\{q_a;n_{bs_b}\}}} - x_{0_{\{q_{(a-1)};n_{bs_b}\}}}}{x_{0_{\{q_{a_{\max}};n_{bs_b}\}}}} \cdot 100, \qquad \begin{aligned} q_a &= 1000 + 1000a, & a &= 1, 2, ..., a_{\max} \\ n_{bs_b} &= 50 + 50b, & b &= 0, 1, ..., b_{\max} \end{aligned} \tag{D.1}$$

Figure D.3 and Figure D.4 shows the heat maps of the lift and drag coefficients' gradients, respectively. From the heat maps it can be seen that the general trend is that the change in the coefficients decreases with the increasing number of data points. However, there exist blade sections (rows in the heat maps) where this is not the case, especially when $n_{bs} \in [250, 400]$.

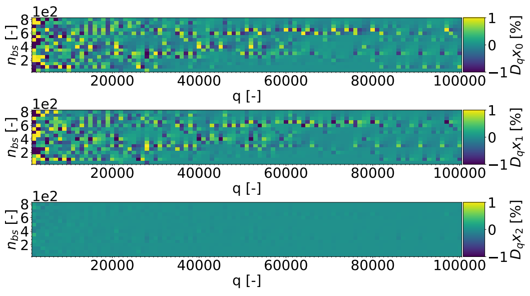


Figure D.3: Heat map with the value of the change of the polynomial coefficients used for the lift coefficient identification along the number of data points axis.

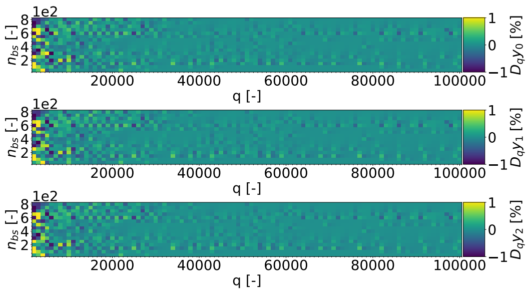


Figure D.4: Heat map with the value of the change of the polynomial coefficients used for the drag coefficient identification along the number of data points axis.

Figure D.5 shows the change in the first lift polynomial coefficient for 300 blade sections as a function of the number of data points; a linear representation of the sixth row of the first lift heat map. As can be seen, the graph is oscillating around the x axis and the mean of all the points is 2.83·10^{-4}; the optimisation is oscillating between two close optima but it can be considered to have virtually converged. Hence, in order to dampen this oscillatory behaviour from the heat maps, the moving average (MA) is computed with a subset size (s) of 10 points, as shown in Equation D.2 for the first lift polynomial coefficient. Figure D.6 and Figure D.7 present the final updated heat maps.

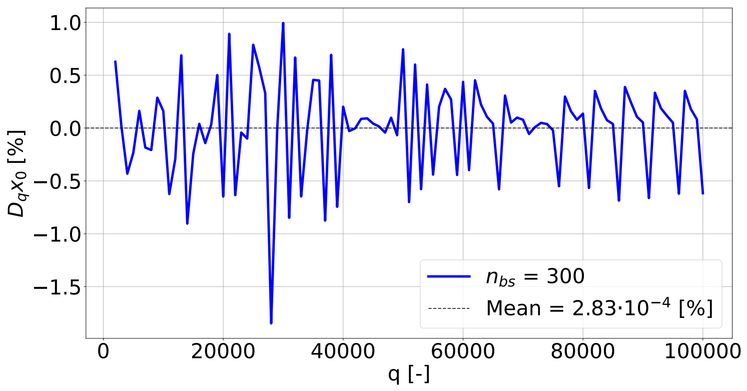


Figure D.5: First lift polynomial coefficient change with respect to the number data points ($D_q x_{0_{\{q_a;300\}}}$).

$$MA(D_q x_{0_{\{q_a;n_{bs_b}\}}}, s) = \frac{\sum_{k=0}^{\min\{s,a\}-1} D_q x_{0_{\{q_{(a-k)};n_{bs_b}\}}}}{\min\{s,a\}} \tag{D.2}$$

In order to be able to decide which is the right number of data points, a single metric as a function of q needs to be defined. For that purpose, all the heat maps are first collapsed into a single one by taking for each $q - n_{bs}$ combination the maximum absolute value among all heat maps, as given by

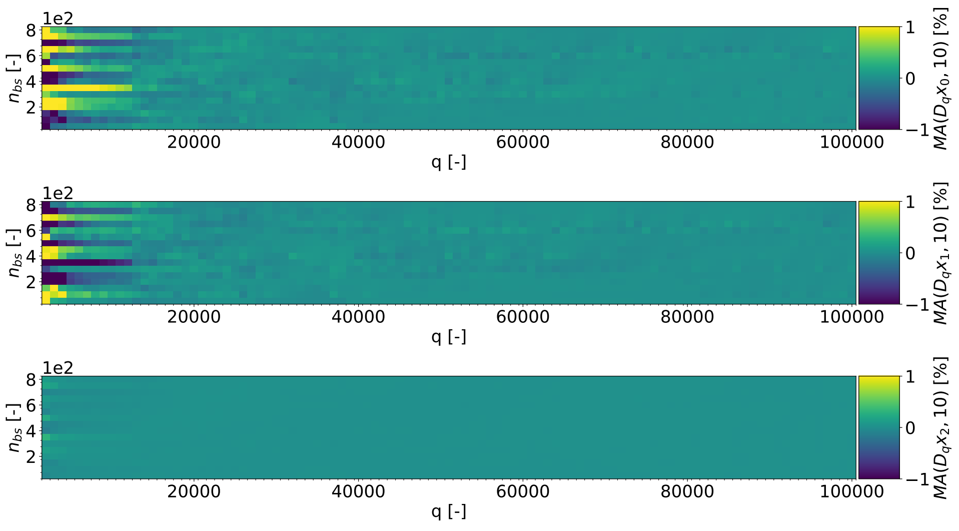


Figure D.6: Moving average (Equation D.2) applied along the q axis of Figure D.3; the polynomial coefficients of the lift coefficient identification.

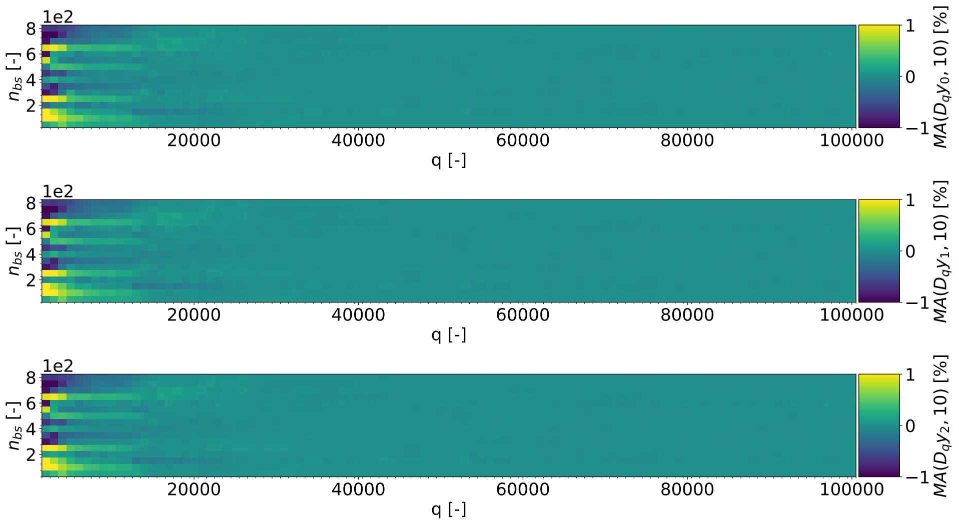


Figure D.7: Moving average (Equation D.2) applied along the q axis of Figure D.4; the polynomial coefficients of the drag coefficient identification.

Equation D.3 and as can be observed in Figure D.8. The maximum absolute value is taken instead of the average since it is desired that all the polynomial coefficients meet a certain metric that will be later defined. If the average would be chosen among the lift and drag polynomial coefficients, the very unstable behaviour of one coefficient could be dampened and hidden by the converging behaviour of the rest. Furthermore, the final heat map is collapsed by taking the average among all the blade section numbers, leading to the 1D function ($h_q(q_a)$) shown in Equation D.4 and illustrated in Figure D.9.

$$g_q(q_a, n_{bs_b}) = \max\left\{\left|MA(D_q x_{j_{\{q_a;n_{bs_b}\}}}, 10)\right| : j = 0,1,...,m;\ \left|MA(D_q y_{k_{\{q_a;n_{bs_b}\}}}, 10)\right| : k = 0,1,...,n\right\} \tag{D.3}$$

$$h_q(q_a) = \frac{\sum_{b=0}^{b_{\text{max}}} g(q_a, n_{bs_b})}{b_{\text{max}} + 1} \tag{D.4}$$

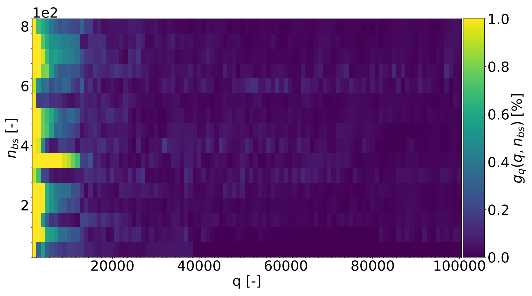


Figure D.8: Collapsed lift and drag polynomial coefficients' heat maps according to Equation D.3

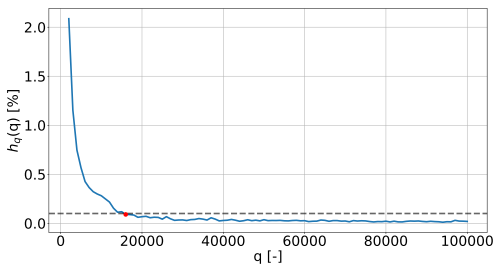


Figure D.9: Collapsed heat map along the n_{bs} axis according to Equation D.4.

Given $h_q(q_a)$, the ideal number of data points (q_{ii}) used for the lift and drag coefficient identification is such that $\max\{h_q(q_a) : a = ii, ii+1, ..., a_{\max}\} < 0.1$. The threshold of 0.1 was chosen heuristically. The horizontal dashed line in Figure D.9 represents this threshold. As can be seen, the ideal q_a given the previous condition is 16,000.

Repeating the same process along the n_{bs} axis in order to define the ideal number of blade sections, leads to Figure D.10. As can be observed, the polynomial coefficients do not converge with respect to the number of blade sections and the $h_q(q_a)$ value remains above 0.1% but below 1%. This means that with every 50 new blade sections, the maximum change in any of the polynomial coefficients is less than 1%.

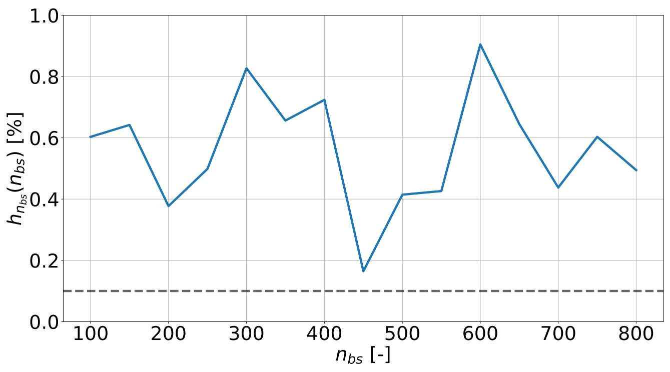


Figure D.10: Collapsed heat map along the q axis in a similar procedure as the outlined to generate Figure D.9.

The number of blade sections is important for two aspects: the blade discretisation should be high enough to allow multiple degrees of blade damage failure, and the blade discretisation should be low enough in order to allow for low computation times upon blade damage failure during simulation in UE4; allowing fast data gathering. For the present research, 100 blade sections were chosen since it allows to divide the blade in integer percentages and it is low in the range of n_{bs} values considered in the current section. The final selected number of data points and blade sections are summarised in Table D.1.

Table D.1: Selected BET hyper-parameters ($q - n_{bs}$).

	Number of data points (q)	Number of blade sections (n_{bs})
Selected hyper-parameter value	16,000	100

E
BET model validation results

This appendix shows an extended version of the results presented in subsection 9.4.4. section E.1 shows the thrust and torque results for different propeller speeds when there is no blade damage and the wind speed is constant at 2 m/s. When there is propeller damage, two signal characteristics need to be validated, namely the signal mean and the amplitude of its oscillations. First, section E.2 and section E.3 show the same plots as section E.1 for the signal mean of 10% and 25% blade damage, respectively. Finally, section E.4 and section E.5 repeat the same process comparing the model generated and the reconstructed experimental signal amplitudes.

E.1. Thrust and torque validation results for BD=0% and V_∞=2 m/s

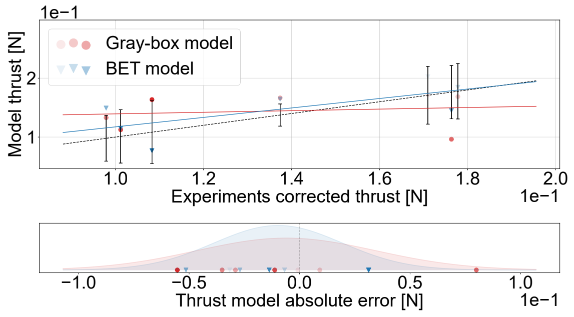


Figure E.1: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=300 rad/s.

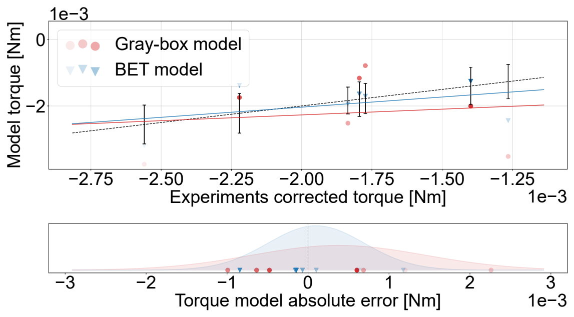


Figure E.2: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=300 rad/s.

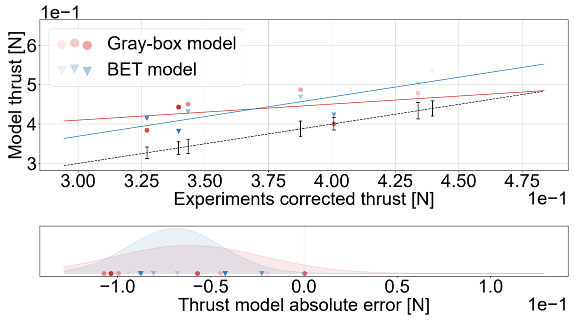


Figure E.3: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=500 rad/s.

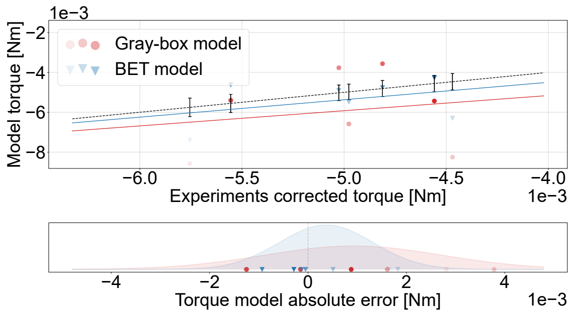


Figure E.4: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=500 rad/s.

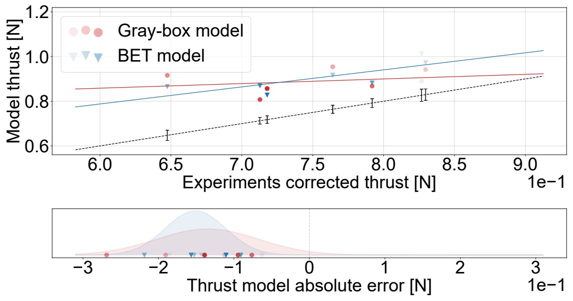


Figure E.5: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=700 rad/s.

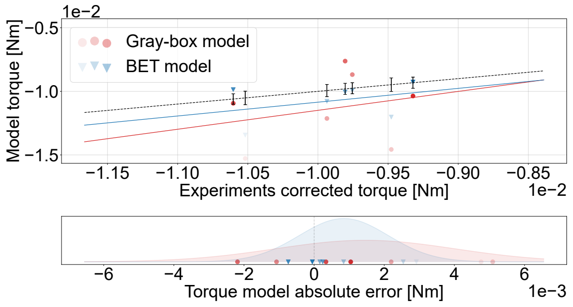


Figure E.6: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=700 rad/s.

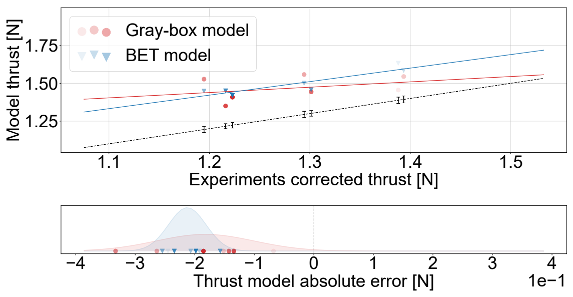


Figure E.7: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=900 rad/s.

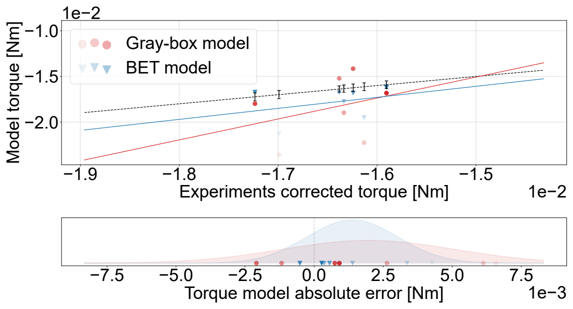


Figure E.8: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=900 rad/s.

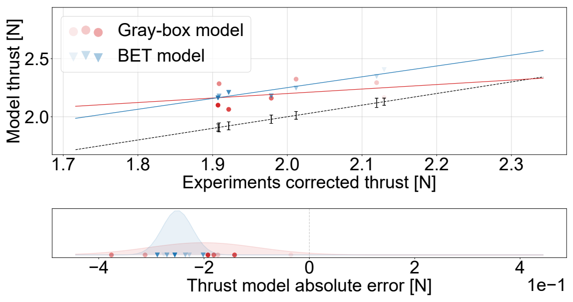


Figure E.9: Experimental and model thrust measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=1100 rad/s.

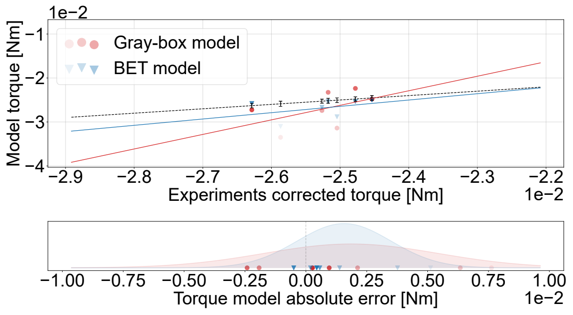


Figure E.10: Experimental and model torque measurements and their absolute error for: BD=0%, V_∞=2 m/s and ω=1100 rad/s.

E.2. Thrust and torque validation results for BD=10% and V_∞=2 m/s

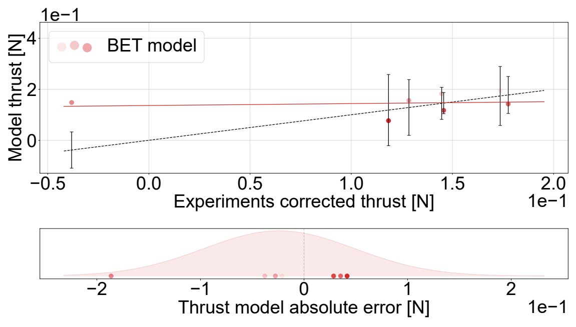


Figure E.11: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=300 rad/s.

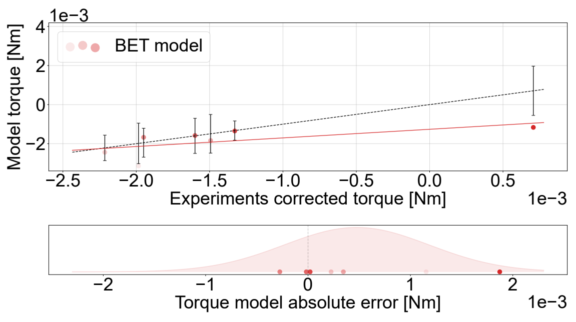


Figure E.12: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=300 rad/s.

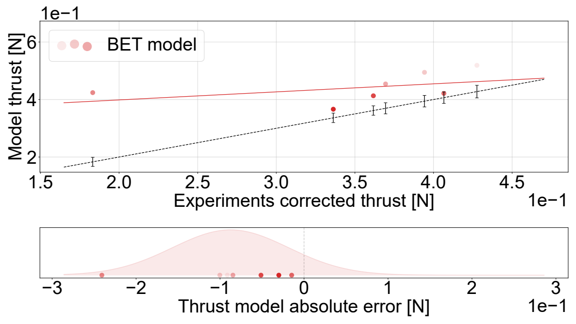


Figure E.13: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=500 rad/s.

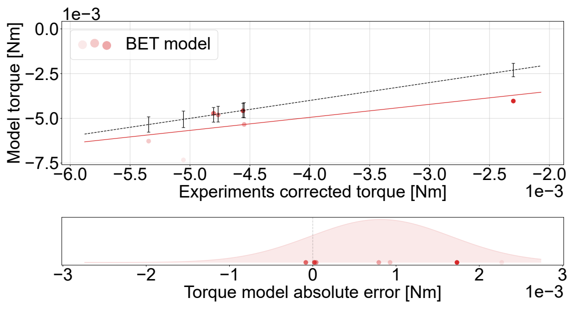


Figure E.14: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=500 rad/s.

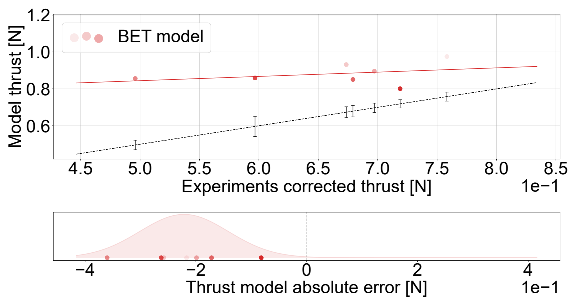


Figure E.15: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s.

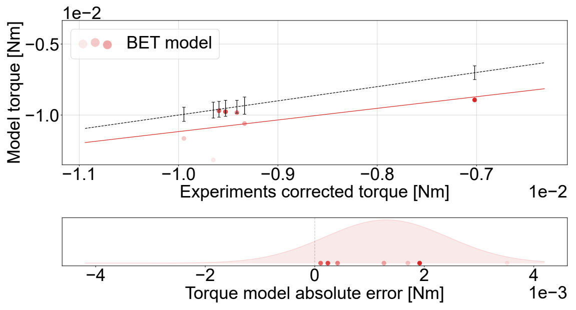


Figure E.16: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s.

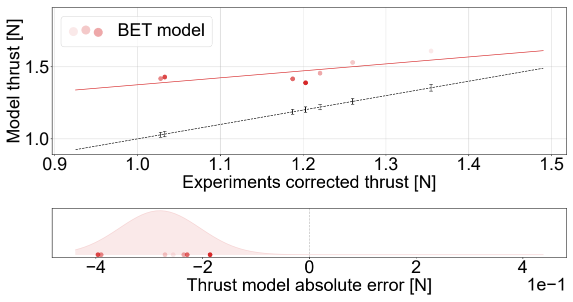


Figure E.17: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=900 rad/s.

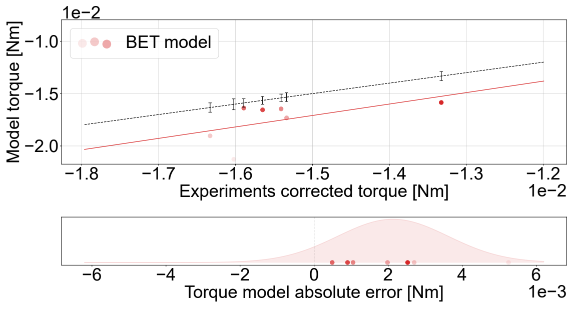


Figure E.18: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=900 rad/s.

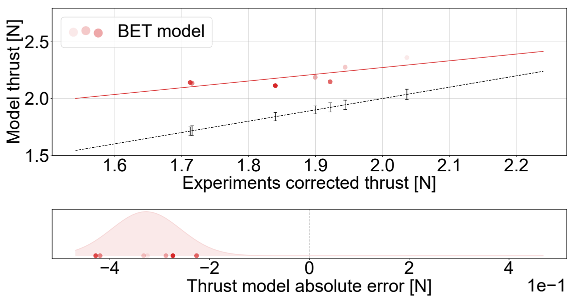


Figure E.19: Experimental and model thrust measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=1100 rad/s.

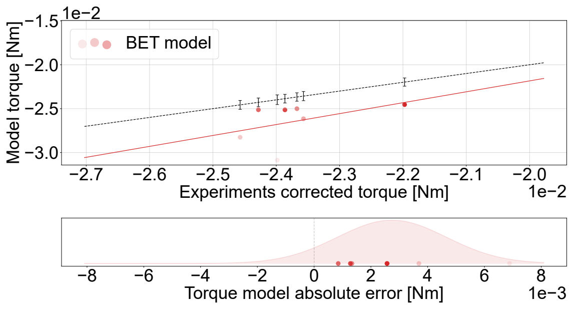


Figure E.20: Experimental and model torque measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=1100 rad/s.

E.3. Thrust and torque validation results for BD=25% and V_∞=2 m/s

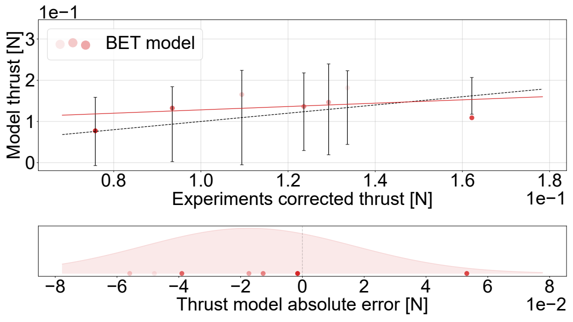


Figure E.21: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=300 rad/s.

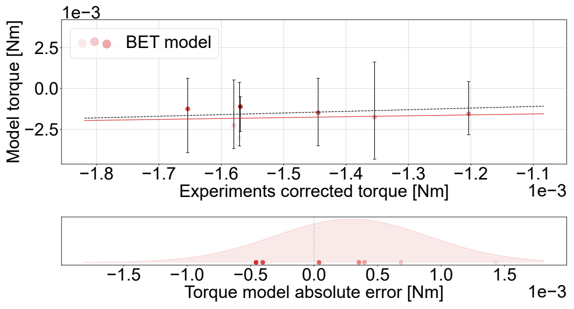


Figure E.22: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=300 rad/s.

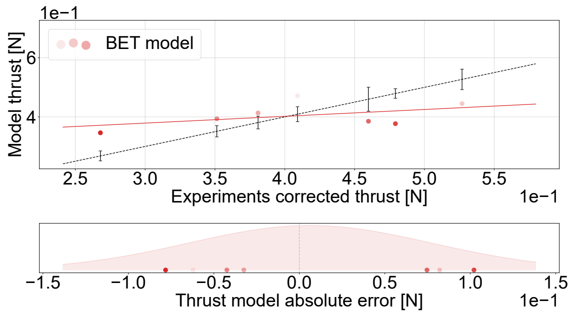


Figure E.23: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=500 rad/s.

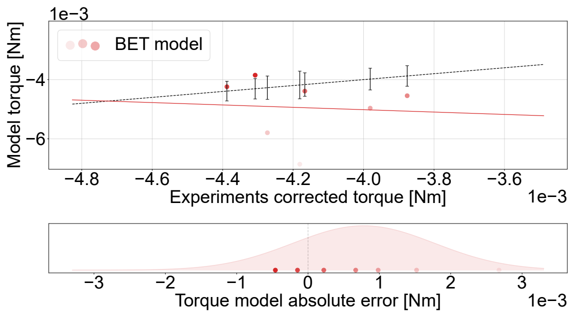


Figure E.24: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=500 rad/s.

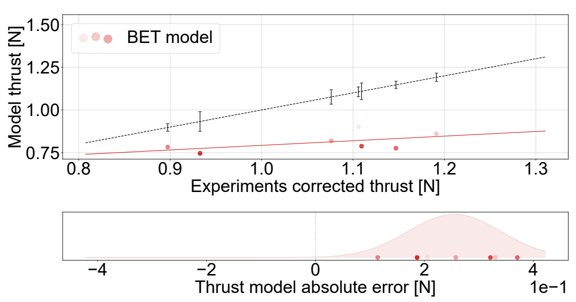


Figure E.25: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s.

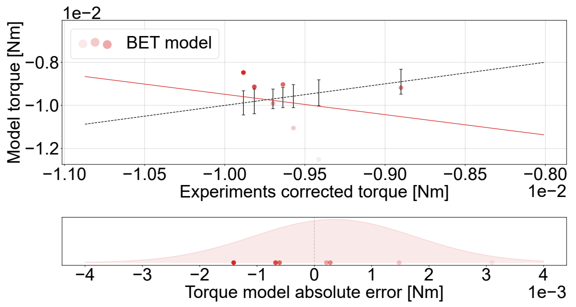


Figure E.26: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s.

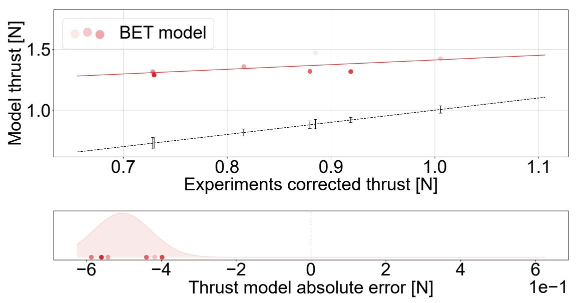


Figure E.27: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=900 rad/s.

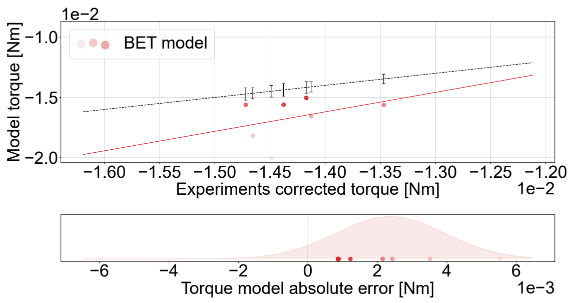


Figure E.28: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=900 rad/s.

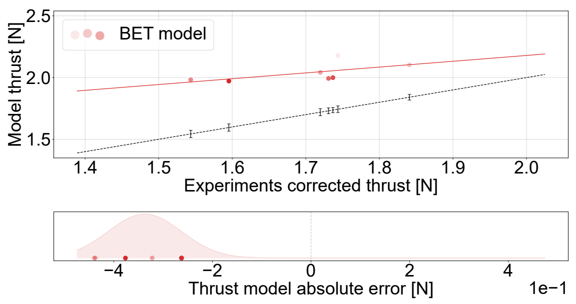


Figure E.29: Experimental and model thrust measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=1100 rad/s.

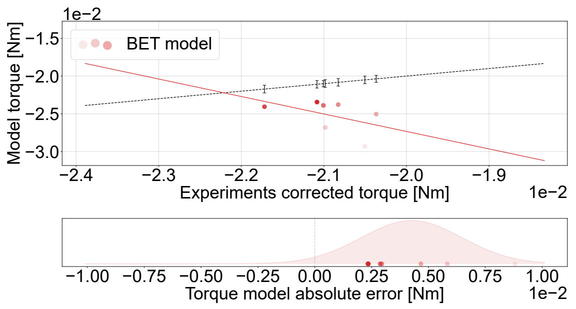


Figure E.30: Experimental and model torque measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=1100 rad/s.

E.4. Thrust and torque damage induced oscillation amplitude validation results for BD=10% and V_∞=2 m/s

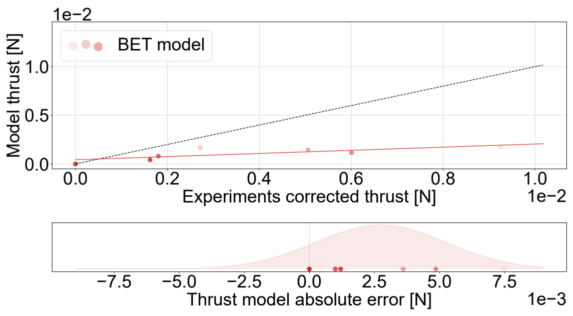


Figure E.31: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=300 rad/s.

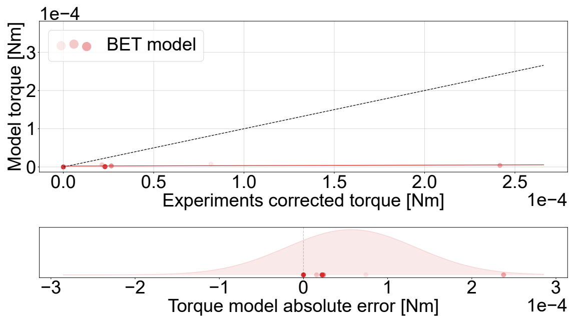


Figure E.32: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=300 rad/s.

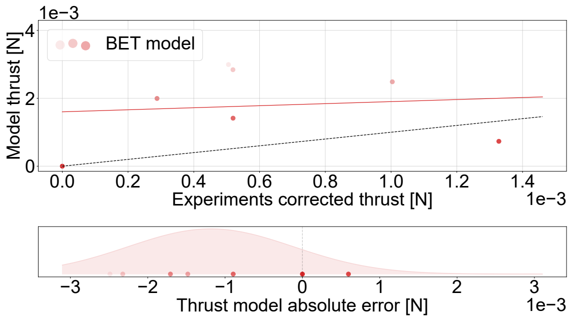


Figure E.33: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=500 rad/s.

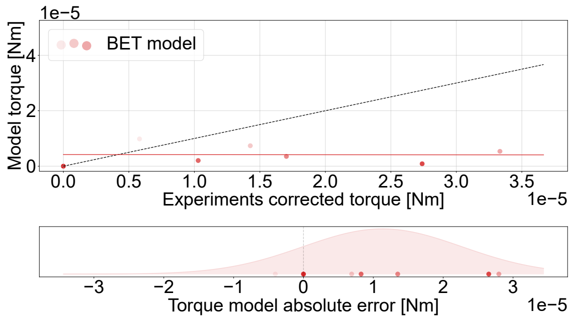


Figure E.34: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=500 rad/s.

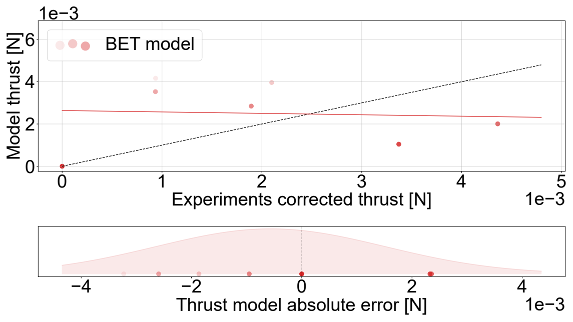


Figure E.35: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s.

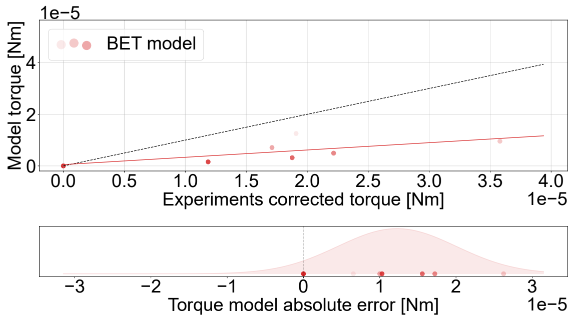


Figure E.36: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=700 rad/s.

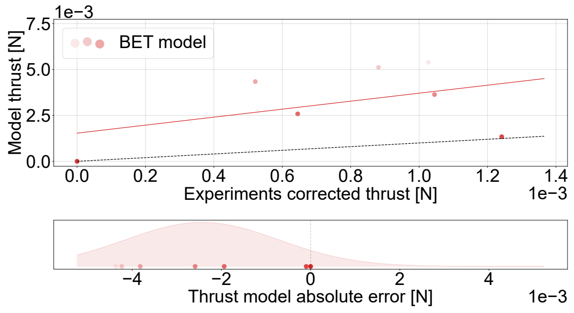


Figure E.37: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=900 rad/s.

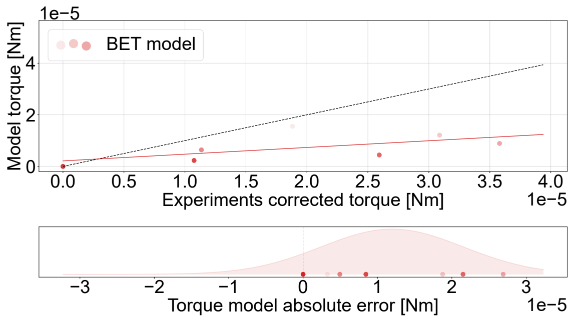


Figure E.38: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=900 rad/s.

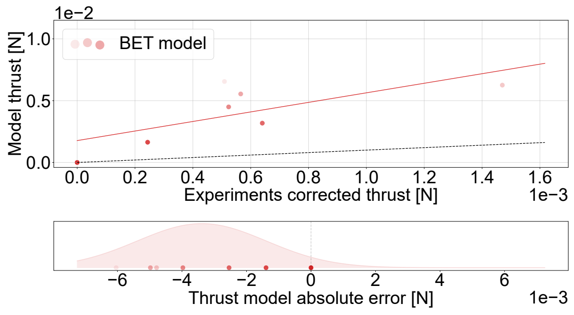


Figure E.39: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=1100 rad/s.

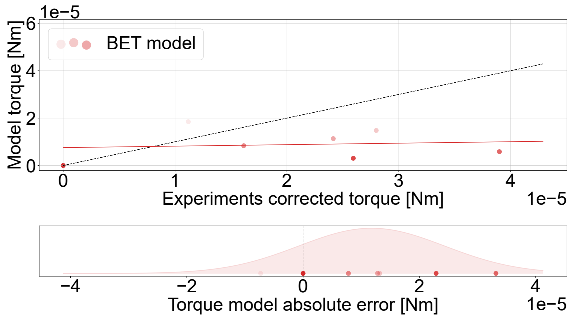


Figure E.40: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=10%, V_∞=2 m/s and ω=1100 rad/s.

E.5. Thrust and torque damage induced oscillation amplitude validation results for BD=25% and V_∞=2 m/s

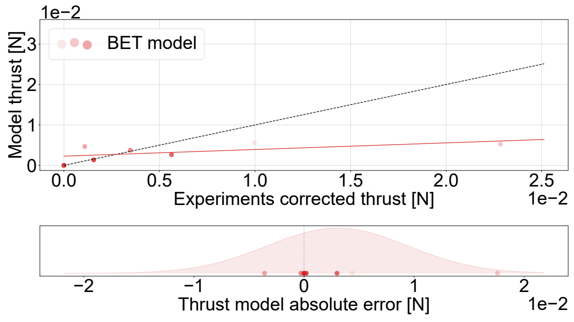


Figure E.41: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=300 rad/s.

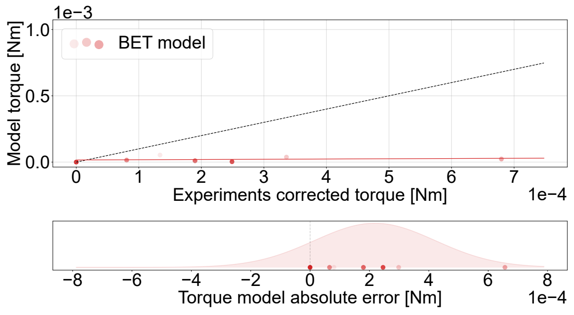


Figure E.42: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=300 rad/s.

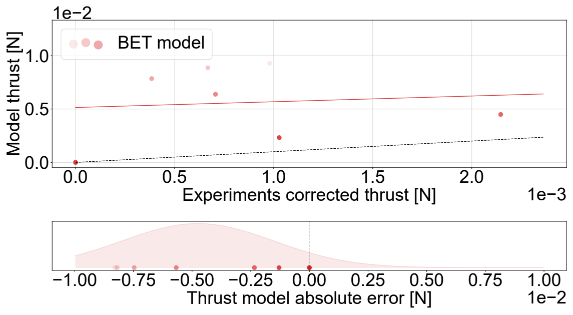


Figure E.43: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=500 rad/s.

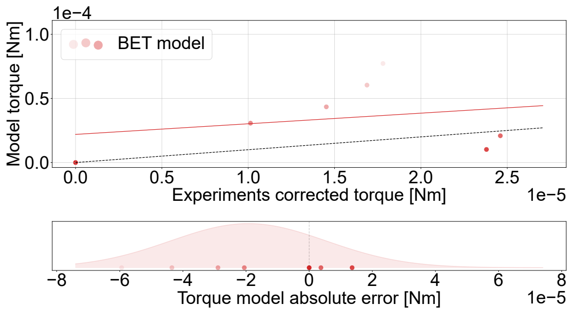


Figure E.44: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=500 rad/s.

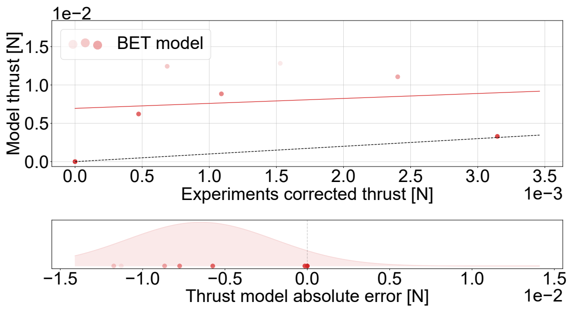


Figure E.45: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s.

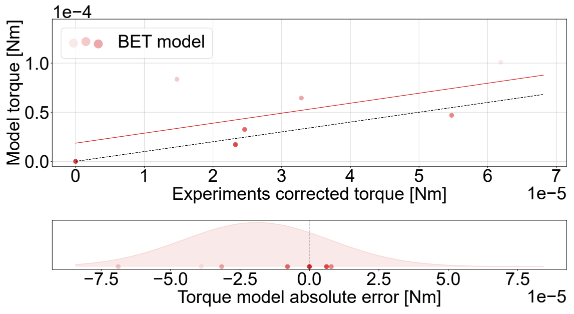


Figure E.46: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=700 rad/s.

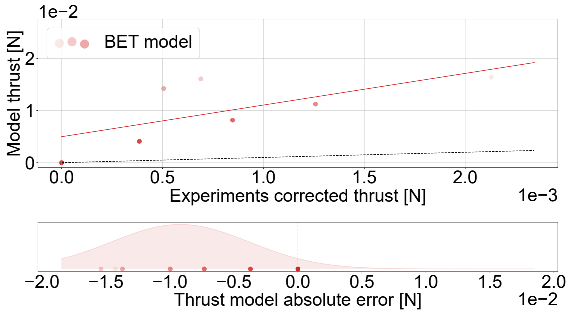


Figure E.47: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=900 rad/s.

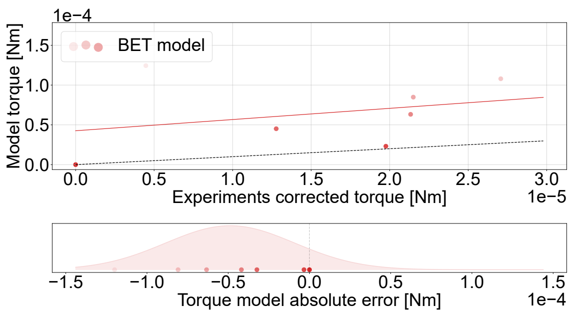


Figure E.48: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=900 rad/s.

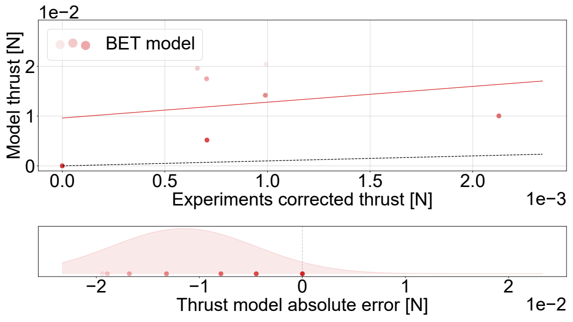


Figure E.49: Experimental and model damage induced thrust oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=1100 rad/s.

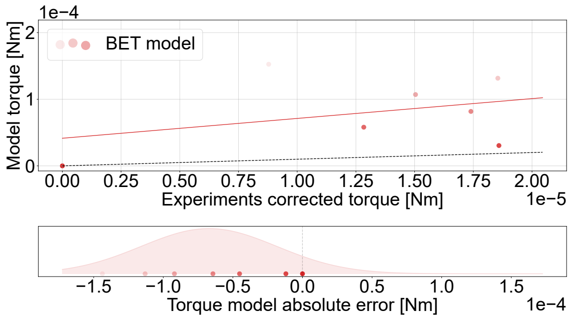


Figure E.50: Experimental and model damage induced torque oscillations amplitude measurements and their absolute error for: BD=25%, V_∞=2 m/s and ω=1100 rad/s.

F

Measurement noise impact on the Lomb-Scargle periodogram signal reconstruction

In subsubsection 9.4.4.2 it was shown that the Lomb-Scargle periodogram shows a superior performance for signal reconstruction when compared to PSO. However, subsection 9.4.2 mentioned the presence of multiple sources of error that could have polluted the experimental measurements, hindering the reliable identification of the damage induced thrust and torque signal oscillations' amplitude. Hence, this section aims at analysing the impact of noise in the measurements for signal reconstruction.

For that purpose, the scenario presented in Figure 9.30 will be simulated as an example. The goal is the identification of a sinusoid with a frequency of 95.5 Hz, 0.01 N of amplitude, a vertical displacement of 0.05 N and a phase of 180°. The wave spans 20 seconds and it is sampled every 0.1497 seconds (6.68 Hz) with an error in the sampling time modelled by a normal distribution centred at zero and a standard deviation equal to 0.0057 seconds, the actual sampling time error observed in the experimental data. In order to assess the impact of noise in the measurements, the sinusoidal signal is polluted with zero mean Gaussian noise with an example standard deviation of 0.015. Figure F.1 shows the first 0.5 seconds of the clean sinusoid, the noisy signal and the samples taken from the latter for signal reconstruction using the Lomb-Scargle periodogram. Figure F.2 presents the reconstructed signal with this statistical approach. As can be observed, the mean and the amplitude of the reconstructed signal has some errors. For the purpose of the present research, only the amplitude error is exploited from the signal reconstruction. In this case the amplitude relative error is 16.51%.

From this simple example, it is already visible the deviation of the reconstructed signal from its original counterpart due to the presence of measurement noise. In order to observe its impact depending on the noise magnitude, Figure F.3 shows the reconstructed amplitude for different values of the noise standard deviation. For each value of the standard deviation, 100 signals were reconstructed with the same Gaussian distribution of measurement noise. Hence, each value of the x-axis has 100 data points which have been reduced to a mean and a standard deviation in the form of a confidence interval.

Beyond the growing bias, the most important aspect is the growing confidence interval. When the noise has a standard deviation equal to four times the expected amplitude (0.04), the reconstructed one can be already twice as large (0.02), meaning a relative error of 100%. As mentioned in subsection 9.4.2, the load cell has an accuracy of $\pm$0.0147 N. This could be approximately modelled as a zero mean Gaussian noise with a standard deviation of 0.0147/2=0.00735 N. Looking at Figure F.3, a sinusoid white noise standard deviation of 0.00735 N corresponds to a reconstructed amplitude with a mean at 0.01 N and a confidence interval of two times 0.00201 N. This means that the Lomb-Scargle periodogram could introduce a maximum reconstruction relative error of 20.1% when only considering the load cell sensor noise for this particular example.

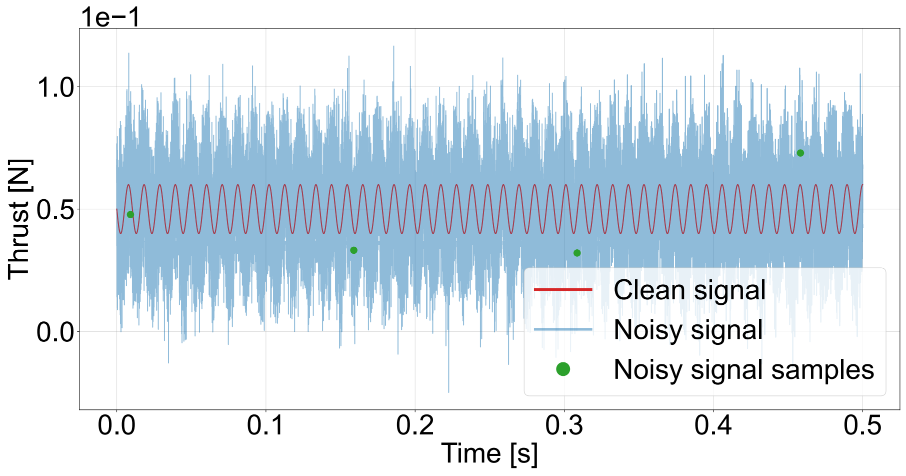


Figure F.1: Clean signal of a sinusoid with frequency of 95.5 Hz and amplitude of 0.01 N to be reconstructed with the samples taken from its noisy version.

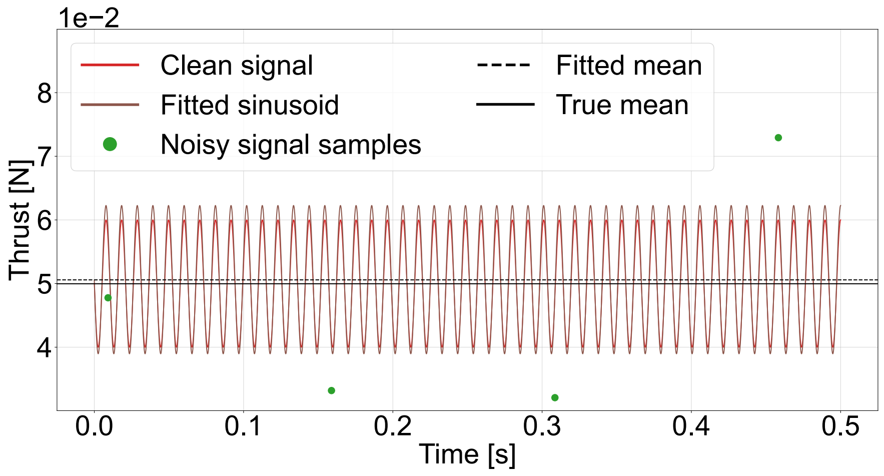


Figure F.2: Clean and reconstructed sinusoids with frequency of 95.5 Hz and amplitude of 0.01 N with their mean values represented by the bold and dashed lines, respectively.

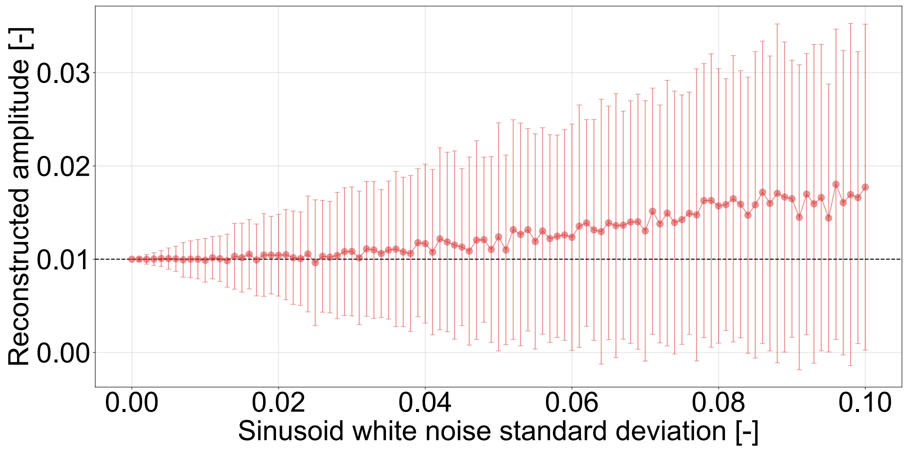


Figure F.3: Reconstructed amplitude of a sinusoid with frequency of 95.5 Hz and amplitude of 0.01 N as a function of the measurement noise Gaussian standard deviation.

The aforementioned relative error will be higher for lower propeller rotational speeds, lower propeller damage and lower angles of the propeller plane with respect to the flow because the oscillation amplitudes to be identified will be smaller, whereas the load cell noise will remain constant. A clear example is the scenario given by BD=10%, $\angle\vec{V}^B$=75%, $\vec{V}^B$=2 m/s and ω=300 rad/s (47.75 Hz), whose amplitude is defined by the BET model in Figure E.31 as 10^{-3}. Figure F.4 shows the reconstructed amplitude for different levels of measurement noise for this particular scenario. In this case, the load cell noise with a standard deviation of 0.00735 N corresponds to a reconstructed amplitude with a mean at 0.001547 N and a confidence interval of two times 0.001503 N. This means that the Lomb-Scargle periodogram could introduce a maximum reconstruction relative error of 204.5%.

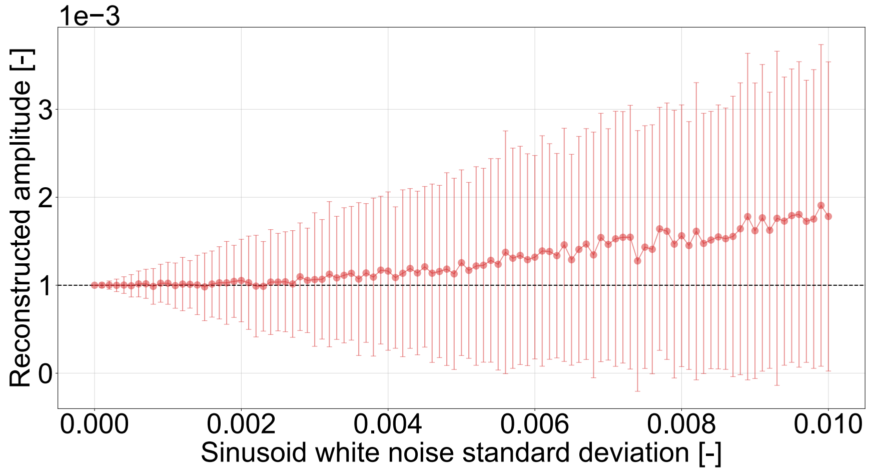


Figure F.4: Reconstructed amplitude of a sinusoid with frequency of 47.75 Hz and amplitude of 10^{-3} N as a function of the measurement noise Gaussian standard deviation.

This shows that, even though the Lomb-Scargle periodogram is superior to the Particle Swarm Optimisation, it still can introduce relative errors in the reconstructed amplitude higher than 200%, calling into question the validity of the damage induced thrust and torque oscillations' amplitude validation.

Bibliography

[1] Inseok Hwang, S. Kim, Y. Kim, and C. E. Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18(3):636–653, 05 2010. doi: 10.1109/tcst.2009.2026285.

[2] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229 – 252, 2008. doi: 10.1016/j.arcontrol.2008.03.008.

[3] Xuewu Dai and Zhiwei Gao. From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis. *IEEE Transactions on Industrial Informatics*, 9(4):2226–2238, 11 2013. doi: 10.1109/tii.2013.2243743.

[4] Mohammed Imran and Sarah A. Alsuhaibani. Chapter 7 - A neuro-fuzzy inference model for diabetic retinopathy classification. In *Intelligent Data Analysis for Biomedical Applications*, Intelligent Data-Centric Systems, pages 147 – 172. Academic Press, 2019. doi: 10.1016/b978-0-12-815553-0.00007-0.

[5] Saed Hussain, Maizura Mokhtar, and Joe M. Howe. Sensor failure detection, identification, and accommodation using fully connected cascade neural network. *IEEE Transactions on Industrial Electronics*, 62(3):1683–1692, March 2015. doi: 10.1109/tie.2014.2361600.

[6] Payam Aboutalebi, Alireza Abbaspour, Parisa Forouzannezhad, and Arman Sargolzaei. A novel sensor fault detection in an unmanned quadrotor based on adaptive neural observer. *Journal of Intelligent & Robotic Systems*, 10 2017. doi: 10.1007/s10846-017-0690-7.

[7] Renato Tinos, Marco H. Terra, and Marcel Bergerman. Fault detection and isolation in cooperative manipulators via artificial neural networks. In *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA'01)*, pages 492–497, México City, México, Sep. 2001. doi: 10.1109/cca.2001.973914.

[8] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. Chapter 10 - Convolutional neural networks. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 173 – 191. Academic Press, 2020. doi: 10.1016/b978-0-12-815739-8.00010-9.

[9] Dingfei Guo, Maiying Zhong, Hongquan Ji, Yang Liu, and Rui Yang. A hybrid feature model and deep learning based fault diagnosis for unmanned aerial vehicle sensors. *Neurocomputing*, 319: 155 – 163, 2018. doi: 10.1016/j.neucom.2018.08.046.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Jorge I. Mireles González. Deep recurrent neural networks for fault detection and classification. Master's thesis, University of Waterloo, Canada, 2018.

[12] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330, International Convention Centre, Sydney, Australia, 08 2017. PMLR. doi: 10.5555/3305381.3305518.

[13] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks — A tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022. doi: 10.1109/mci.2022.3155327.

[14] Weike Sun, Antonio R.C. Paiva, Peng Xu, Anantha Sundaram, and Richard D. Braatz. Fault detection and identification using Bayesian recurrent neural networks. *Computers & Chemical Engineering*, 141:106991, 2020. doi: 10.1016/j.compchemeng.2020.106991.

[15] Paul M. Frank and Birgit Köppen-Seliger. Fuzzy logic and neural network applications to fault diagnosis. *International Journal of Approximate Reasoning*, 16(1):67 – 88, 1997. doi: 10.1016/S0888-613x(96)00116-8.

[16] Mihiar Ayoubi. Fuzzy systems design based on a hybrid neural structure and application to the fault diagnosis of technical processes. *Control Engineering Practice*, 4(1):35 – 42, 1996. doi: 10.1016/0967-0661(95)00204-8.

[17] Pedro Santos, Luisa Villa Montoya, Anibal Reñones, Andrés Bustillo, and Jesús Maudes. An SVM-based solution for fault detection in wind turbines. *Sensors*, 15:5627–48, 03 2015. doi: 10.3390/s150305627.

[18] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *Neural Information Processing*, pages 373–382, Guangzhou, 2017. Springer International Publishing. doi: 10.1007/978-3-319-70096-0_39.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. doi: 10.5555/3295222.3295349.

[20] Rana Abdallah, Raed Kouta, Charles Sarraf, Jaafar Gaber, and Maxime Wack. Fault tree analysis for the communication of a fleet formation flight of UAVs. In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 202–206, Milan, 2017. doi: 10.1109/icsrs.2017.8272821.

[21] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 134:1–21, 2015. doi: 10.1016/j.cviu.2015.02.008.

[22] Davide Scaramuzza and Zichao Zhang. Visual-inertial odometry of aerial robots. *Encyclopedia of Robotics*, 2020. doi: 10.1007/978-3-642-41610-1_71-1.

[23] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Geometry-based next frame prediction from monocular video. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1700–1707, Los Angeles, CA, 2017. doi: 10.1109/ivs.2017.7995953.

[24] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5667–5675, 2018. doi: 10.1109/cvpr.2018.00594.

[25] John Skinner, S. Garg, N. Sünderhauf, P. Corke, B. Upcroft, and M. Milford. High-fidelity simulation for evaluating robotic vision performance. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2737–2744, 2016. doi: 10.1109/iros.2016.7759425.

[26] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635, Zürich, Switzerland, 2018. Springer International Publishing. doi: 10.1007/978-3-319-67361-5_40.

[27] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, Yizhou Wang, and Alan Yuille. UnrealCV: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, page 1221–1224, 2017. doi: 10.1145/3123266.3129396.

[28] Sihao Sun and Coen de Visser. Aerodynamic model identification of a quadrotor subjected to rotor failures in the high-speed flight regime. *IEEE Robotics and Automation Letters*, 4(4):3868–3875, 2019. doi: 10.1109/lra.2019.2928758.

[29] J. Gordon Leishman. *Principles of helicopter aerodynamics*. Cambridge University Press, Cambridge, 2006. ISBN 0521858607.

[30] Faculty of Aerospace Engineering at Delft University of Technology. Open jet facility. URL https://www.tudelft.nl/lr/organisatie/afdelingen/flow-physics-and-technology/facilities/low-speed-wind-tunnels/open-jet-facility. Accessed: 2022-09-24.

[31] Jean-Philippe Aurambout, Konstantinos Gkoumas, and Biagio Ciuffo. Last mile delivery by drones: an estimation of viable market potential and access to citizens across European cities. *European Transport Research Review*, 11, 12 2019. doi: 10.1186/s12544-019-0368-2.

[32] Shushman Choudhury, Kiril Solovey, Mykel J. Kochenderfer, and Marco Pavone. Efficient large-scale multi-drone delivery using transit networks. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4543–4550, 2020. doi: 10.1613/jair.1.12450.

[33] David P. Thipphavong, Rafael Apaza, Bryan Barmore, Vernol Battiste, Barbara Burian, Quang Dao, Michael Feary, Susie Go, Kenneth H. Goodrich, Jeffrey Homola, Husni R. Idris, Parimal H. Kopardekar, Joel B. Lachter, Natasha A. Neogi, Hok Kwan Ng, Rosa M. Oseguera-Lohr, Michael D. Patterson, and Savita A. Verma. Urban air mobility airspace integration concepts and considerations. In *2018 Aviation Technology, Integration, and Operations Conference*, Atlanta, GA, 06 2018. doi: 10.2514/6.2018-3676.

[34] UNICEF Supply Division. How drones can be used to combat COVID-19. Technical report, UNICEF, 2020. URL https://www.unicef.org. [Accessed 10 January 2021].

[35] Hameed Khan, Kamal K. Kushwah, Saurabh Singh, Harshika Urkude, Muni Raj Maurya, and Kishor Kumar Sadasivuni. Smart technologies driven approaches to tackle COVID-19 pandemic: a review. *3 Biotech*, 11(2), 2021. doi: 10.1007/s13205-020-02581-y.

[36] Ira Boudway. Zipline medical drones begin flying in the United States, 2020. URL https://www.bloomberg.com. [Accessed 10 January 2021].

[37] Patrique Zaman. AVY receives EIC accelerator and flies above Europe, 2020. URL https://www.avy.eu. [Accessed 10 January 2021].

[38] Patrique Zaman. AVY joins AGS airports led consortium to develop UK's first medical delivery drone network, 2021. URL https://www.avy.eu. [Accessed 27 January 2021].

[39] V. Lappas, G. Zoumponos, V. Kostopoulos, H. Shin, A. Tsourdos, M. Tantarini, D. Shmoko, J. Munoz, N. Amoratis, A. Maragkakis, T. Machairas, and A. Trifas. EuroDRONE, a European UTM testbed for U-Space. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1766–1774, 2020. doi: 10.1109/icuas48674.2020.9214020.

[40] Farhan Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar. UAVs for smart cities: Opportunities and challenges. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 267–273, 2014. doi: 10.1109/icuas.2014.6842265.

[41] Sihao Sun, Xuerui Wang, Qiping Chu, and Coen De Visser. Incremental nonlinear fault-tolerant control of a quadrotor with complete loss of two opposing rotors. *IEEE Transactions on Robotics*, PP:1–15, 08 2020. doi: 10.1109/tro.2020.3010626.

[42] Mark W. Mueller, Sergei Lupashin, Raffaello D'andrea, and Markus Waibel. Controlled flight of a multicopter experiencing a failure affecting an effector, 08 2020. URL https://patents.google.com/patent/EP3007973A1.

[43] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and manoeuvring on a vision-guided MAV using on-board processing. In *2011 IEEE International Conference on Robotics and Automation*, pages 2472–2477, Shanghai, 2011. doi: 10.1109/icra.2011.5980095.

[44] Sihao Sun, Matthias Baert, Bram Schijndel, and Coen De Visser. Upset recovery control for quadrotors subjected to a complete rotor failure from large initial disturbances. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4273–4279, Paris, France, 05 2020. doi: 10.1109/icra40945.2020.9197239.

[45] Yan Jiang, Zhao Zhiyao, Liu Haoxiang, and Quan Quan. Fault detection and identification for quadrotor based on airframe vibration signals: A data-driven method. *2015 34th Chinese Control Conference (CCC)*, pages 6356–6361, 2015. doi: 10.1109/chicc.2015.7260639.

[46] Sergio García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor SLAM for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210, 2016. doi: 10.1109/icarsc.2016.46.

[47] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40), 2020. doi: 10.1126/scirobotics.aaz9712.

[48] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019. doi: 10.1016/j.neunet.2018.12.002.

[49] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250, Brisbane, 2018. doi: 10.1109/icra.2018.8460875.

[50] Rolf Isermann. Fault diagnosis systems an introduction from fault detection to fault tolerance. *Fault-Diagnosis Systems*, 01 2006. doi: 10.1007/3-540-30368-5.

[51] Rolf Isermann. Supervision, fault-detection and fault-diagnosis methods — An introduction. *Control Engineering Practice*, 5(5):639 – 652, 1997. doi: 10.1016/s0967-0661(97)00046-4.

[52] Rolf Isermann. Model-based fault-detection and diagnosis – status and applications. *Annual Reviews in Control*, 29(1):71 – 85, 2005. doi: 10.1016/j.arcontrol.2004.12.002.

[53] Yinsheng Chen, Jingli Yang, Yonghui Xu, Shouda Jiang, Xiadong Liu, and Qi Wang. Status self-validation of sensor arrays using gray forecasting model and bootstrap method. *IEEE Transactions on Instrumentation and Measurement*, 65(7):1626–1640, 2016. doi: 10.1109/tim.2016.2540942.

[54] Khaoula Tidriri, Nizar Chatti, Sylvain Verron, and Teodor Tiplica. Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges. *Annual Reviews in Control*, 42(C):63–81, 2016. doi: 10.1016/j.arcontrol.2016.09.008.

[55] Guillermo Heredia and Anibal Ollero. Detection of sensor faults in small helicopter UAVs using observer/Kalman filter identification. *Mathematical Problems in Engineering*, 2011, 09 2011. doi: 10.1155/2011/174618.

[56] Shen Yin, Bing Xiao, Steven X. Ding, and Donghua Zhou. A review on recent development of spacecraft attitude fault tolerant control system. *IEEE Transactions on Industrial Electronics*, 63 (5):3311–3320, 2016. doi: 10.1109/tie.2016.2530789.

[57] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293 – 311, 2003. doi: 10.1016/s0098-1354(02)00160-6.

[58] Venkat Venkatasubramanian, Raghunathan Rengaswamy, and Surya N Kavuri. A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies. *Computers & Chemical Engineering*, 27(3):313 – 326, 2003. doi: 10.1016/s0098-1354(02)00161-8.

[59] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Surya N. Kavuri, and Kewen Yin. A review of process fault detection and diagnosis: Part III: Process history based methods. *Computers & Chemical Engineering*, 27(3):327 – 346, 2003. doi: 10.1016/s0098-1354(02)00162-x.

[60] Daoliang Li, Ying Wang, Jinxing Wang, Cong Wang, and Yanqing Duan. Recent advances in sensor fault diagnosis: A review. *Sensors and Actuators A: Physical*, 309:111990, 2020. doi: 10.1016/j.sna.2020.111990.

[61] Ahmed Shokry and Antonio Espuña. The ordinary kriging in multivariate dynamic modelling and multistep-ahead prediction. In *28th European Symposium on Computer Aided Process Engineering*, volume 43 of *Computer Aided Chemical Engineering*, pages 265 – 270. Elsevier, 2018. doi: 10.1016/b978-0-444-64235-6.50047-4.

[62] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[63] David Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. doi: 10.1038/323533a0.

[64] Seongmin Heo and Jay H. Lee. Fault detection and classification using artificial neural networks. *10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018*, 51(18):470 – 475, 2018. doi: 10.1016/j.ifacol.2018.09.380.

[65] Gino Iannace, Giuseppe Ciaburro, and Amelia Trematerra. Fault diagnosis for UAV blades using artificial neural network. *Robotics*, 8:59, 07 2019. doi: 10.3390/robotics8030059.

[66] Bodgan M. Wilamowski. Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine*, 3(4):56–63, 12 2009. doi: 10.1109/mie.2009.934790.

[67] David Hunter, Hao Yu, Michael S. Pukish, III, Janusz Kolbusz, and Bodgan M. Wilamowski. Selection of proper neural network sizes and architectures—A comparative study. *IEEE Transactions on Industrial Informatics*, 8(2):228–240, May 2012. doi: 10.1109/tii.2012.2187914.

[68] Bogdan M. Wilamowski and H. Yu. Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803, 11 2010. doi: 10.1109/tnn.2010.2073482.

[69] Lifeng Wang, Yichong He, Zhixiang Zhang, and Congkui He. Trajectory tracking of quadrotor aerial robot using improved dynamic inversion method. *Intelligent Control and Automation*, 04:343–348, 01 2013. doi: 10.4236/ica.2013.44040.

[70] Hamed Ghazavi Khorasgani, Mohammad B. Menhaj, Heidar A. Talebi, and Firooz Bakhtiari-Nejad. Neural-network-based sensor fault detection & isolation for nonlinear hybrid systems. *IFAC Proceedings Volumes*, 45(20):1029 – 1034, 2012. doi: 10.3182/20120829-3-mx-2028.00203.

[71] Heidar A. Talebi and Rajni V. Patel. An intelligent fault detection and recovery scheme for reaction wheel actuator of satellite attitude control systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 3282–3287, 10 2006. doi: 10.1109/cacsd-cca-isic.2006.4777164.

[72] Mou Chen, Peng Shi, and Cheng-Chew Lim. Adaptive neural fault-tolerant control of a 3-DOF model helicopter system. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46 (2):260–270, 2016. doi: 10.1109/tsmc.2015.2426140.

[73] Ihab Samy, Ian Postlethwaite, and Da-Wei Gu. Survey and application of sensor fault detection and isolation schemes. *Control Engineering Practice*, 19(7):658 – 674, 2011. doi: 10.1016/j.conengprac.2011.03.002.

[74] Marco H. Terra and Renato Tinos. Fault detection and isolation in robotic systems via artificial neural networks. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 2, pages 1605–1610 vol.2, Tampa, FL, 12 1998. doi: 10.1109/cdc.1998.758522.

[75] Turker Ince, Serkan Kiranyaz, Levent Eren, Murat Askar, and Moncef Gabbouj. Real-time motor fault detection by 1-D convolutional neural networks. *IEEE Transactions on Industrial Electronics*, 63(11):7067–7075, 11 2016. doi: 10.1109/tie.2016.2582729.

[76] Olivier Janssens, Viktor Slavkovikj, Bram Vervisch, Kurt Stockman, Mia Loccufier, Steven Verstockt, Rik Van de Walle, and Sofie Van Hoecke. Convolutional neural network based fault detection for rotating machinery. *Journal of Sound and Vibration*, 377:331 – 345, 2016. doi: 10.1016/j.jsv.2016.05.027.

[77] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1D convolutional neural networks and applications: A Survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021. doi: 10.1016/j.ymssp.2020.107398.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.

[79] Kunru Chen. Recurrent neural networks for fault detection : An exploratory study on a dataset about air compressor failures of heavy duty trucks. Master's thesis, Halmstad University, School of Information Technology, 2018.

[80] H. A. Talebi, K. Khorasani, and S. Tafazoli. A recurrent neural-network-based sensor and actuator fault detection and isolation for nonlinear systems with application to the satellite's attitude control subsystem. *IEEE Transactions on Neural Networks*, 20(1):45–60, 01 2009. doi: 10.1109/tnn.2008.2004373.

[81] Haitao Zhao, Shaoyuan Sun, and Bo Jin. Sequential fault diagnosis based on LSTM neural network. *IEEE Access*, 6:12929–12939, 2018. doi: 10.1109/access.2018.2794765.

[82] Divish Rengasamy, Hervé P. Morvan, and Grazziela P. Figueredo. Deep learning approaches to aircraft maintenance, repair and overhaul: A review. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 150–156, 2018. doi: 10.1109/itsc.2018.8569502.

[83] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[84] John Mitros and Brian Mac Namee. On the validity of Bayesian neural networks for uncertainty estimation. In *Artificial Intelligence and Cognitive Science*, 2019.

[85] Jeremy Nixon, Mike Dusenberry, Ghassen Jerfel, Timothy Nguyen, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. *ArXiv*, abs/1904.01685, 2019.

[86] Ananya Kumar, Percy Liang, and Tengyu Ma. Verified uncertainty calibration. In *NeurIPS*, 2019. doi: 10.5555/3454287.3454627.

[87] Sebastien Haneuse. Part I: The Bayesian paradigm. Harvard University Lecture, 2015.

[88] Don van Ravenzwaaij, Peter Cassey, and Scott Brown. A simple introduction to Markov chain Monte–Carlo sampling. *Psychonomic Bulletin & Review*, 25, 03 2016. doi: 10.3758/s13423-016-1015-8.

[89] Adrian F. M. Smith and Gareth O. Roberts. Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, 55(1):3–23, 1993. doi: 10.1111/j.2517-6161.1993.tb01466.x.

[90] Cajo ter Braak. A Markov chain Monte Carlo version of the genetic algorithm differential evolution: Easy Bayesian computing for real parameter spaces. *Statistics and Computing*, 16:239–249, 09 2006. doi: 10.1007/s11222-006-8769-1.

[91] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. doi: 10.1214/aoms/1177729694.

[92] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015. doi: 10.5555/2969442.2969527.

[93] Zhi-Hua Zhou. *Ensemble Learning*, pages 270–273. Springer US, Boston, MA, 2009. doi: 10.1007/978-0-387-73003-5_293.

[94] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. doi: 10.5555/3045118.3045290.

[95] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011. doi: 10.5555/2986459.2986721.

[96] Manfred Opper. *A Bayesian Approach to On-Line Learning*, page 363–378. Cambridge University Press, USA, 1999. doi: 10.5555/304710.304756.

[97] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *ArXiv*, abs/1506.02158, 2015.

[98] Yaniv Ovadia, E. Fertig, J. Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *NeurIPS*, 2019. doi: 10.5555/3454287.3455541.

[99] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *ICML*, 2020. doi: 10.5555/3524938.3525442.

[100] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 41–50, 2019. doi: 10.1109/cvpr.2019.00013.

[101] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31:105–112, 03 2009. doi: 10.1016/j.strusafe.2008.06.020.

[102] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *ICML*, Stockholm, 2018.

[103] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian active learning with image data. In *ICML*, 2017. doi: 10.5555/3305381.3305504.

[104] Toan Tran, Thanh-Toan Do, I. Reid, and G. Carneiro. Bayesian generative active deep learning. In *ICML*, 2019.

[105] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *CoRR*, abs/1704.02798, 2017.

[106] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *Proceedings of The 33rd International Conference on Machine Learning*, 06 2015. doi: 10.5555/3045390.3045502.

[107] Nick Pawlowski, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *ArXiv*, abs/1711.01297, 2017.

[108] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2218–2227, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. doi: 10.5555/3305890.3305910.

[109] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *ICML*, Lille, 2015. doi: 10.5555/3045118.3045316.

[110] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1027–1035, Red Hook, NY, USA, 2016. Curran Associates Inc. doi: 10.5555/3157096.3157211.

[111] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965. doi: 10.1016/S0019-9958(65)90241-X.

[112] Lotfi A. Zadeh. Fuzzy algorithms. *Information and Control*, 12(2):94 – 102, 1968. ISSN 0019-9958. doi: 10.1016/s0019-9958(68)90211-8.

[113] Petr Cintula, Christian G. Fermüller, and Carles Noguera. Fuzzy Logic. In *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2017 edition, 2017.

[114] Chuen-Chien Lee. Fuzzy logic in control systems: fuzzy logic controller. I. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–418, March 1990. doi: 10.1109/21.52551.

[115] Chuen-Chien Lee. Fuzzy logic in control systems: fuzzy logic controller. II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):419–435, March 1990. doi: 10.1109/21.52552.

[116] Robert Babuška. Fuzzy modeling - a control engineering perspective. In *IEEE International Conference on Fuzzy Systems*, volume 4, pages 1897 – 1902 vol.4, Barcelona, 04 1995. doi: 10.1109/fuzzy.1995.409939.

[117] Robert Babuška and Henk B. Verbruggen. An overview of fuzzy modeling for control. *Control Engineering Practice*, 4(11):1593 – 1606, 1996. doi: 10.1016/0967-0661(96)00175-x.

[118] Robert Babuška. *Fuzzy modeling and identification*. PhD dissertation, Delft University of Technology, Delft, The Netherlands, 1996.

[119] Rolf Isermann. On fuzzy logic applications for automatic control, supervision, and fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 28(2): 221–235, March 1998. doi: 10.1109/3468.661149.

[120] Sauter, Mary, Sirou, and Thieltgen. Fault diagnosis in systems using fuzzy logic. In *1994 Proceedings of IEEE International Conference on Control and Applications*, pages 883–888 vol.2, Glasgow, UK, Aug 1994. doi: 10.1109/cca.1994.381205.

[121] Rita A. Ribeiro. Fuzzy space monitoring and fault detection applications. *Journal of Decision Systems*, 15(2-3):267–286, 2006. doi: 10.3166/jds.15.267-286.

[122] Ebrahim H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1 – 13, 1975. doi: 10.1016/s0020-7373(75)80002-2.

[123] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, 01 1985. doi: 10.1109/tsmc.1985.6313399.

[124] Javad J. Jassbi, Paulo J. A. Serra, Rita A. Ribeiro, and Alessandro Donati. A comparison of Mandani and Sugeno inference systems for a space fault detection application. In *2006 World Automation Congress*, pages 1–8, Budapest, July 2006. doi: 10.1109/wac.2006.376033.

[125] Linlin Li, Steven X. Ding, Ying Yang, and Yong Zhang. Robust fuzzy observer-based fault detection for nonlinear systems with disturbances. *Neurocomputing*, 174:767 – 772, 2016. doi: 10.1016/j.neucom.2015.09.102.

[126] Kenneth Chi hang Lo, Eric H. K. Fung, and Yiu-Kwong Wong. Intelligent automatic fault detection for actuator failures in aircraft. *IEEE Transactions on Industrial Informatics*, 5(1):50–55, Feb 2009. doi: 10.1109/tii.2008.2008642.

[127] Hong-Chan Chin. Fault section diagnosis of power system using fuzzy logic. *IEEE Transactions on Power Systems*, 18(1):245–250, Feb 2003. doi: 10.1109/tpwrs.2002.807095.

[128] Adrián Rodríguez Ramos, Carlos Domínguez Acosta, P. J. Torres, Eileen I. Serrano Mercado, Gerson Beauchamp-Báez, L. A. Rifón, and O. Llanes-Santiago. An approach to multiple fault diagnosis using fuzzy logic. *Journal of Intelligent Manufacturing*, 30:429–439, 2019.

[129] Chaochao Chen, Bin Zhang, George Vachtsevanos, and Marcos Orchard. Machine condition prediction based on adaptive neuro–fuzzy and high-order particle filtering. *IEEE Transactions on Industrial Electronics*, 58(9):4353–4364, Sep. 2011. doi: 10.1109/tie.2010.2098369.

[130] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. doi: 10.1145/130385.130401.

[131] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. doi: 10.1007/978-1-4757-3264-1.

[132] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55(1):169 – 186, 2003. doi: 10.1016/s0925-2312(03)00431-4.

[133] Prashant M. Pawar and Sung Nam Jung. Support vector machine based online composite helicopter rotor blade damage detection system. *Journal of Intelligent Material Systems and Structures*, 19(10):1217–1228, 2008. doi: 10.1177/1045389X07084713.

[134] Sheng-Fa Yuan and Fu-Lei Chu. Support vector machines-based fault diagnosis for turbo-pump rotor. *Mechanical Systems and Signal Processing*, 20(4):939 – 952, 2006. doi: 10.1016/j.ymssp.2005.09.006.

[135] Jaroslaw Kurek and Stanislaw Osowski. Support vector machine for fault diagnosis of the broken rotor bars of squirrel-cage induction motor. *Neural Computing and Applications*, 19:557–564, 2009. doi: 10.1007/s00521-009-0316-5.

[136] Bernhard Scholkopf and Alexander J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

[137] Elgiz Baskaya, Murat Bronz, and Daniel Delahaye. Fault detection diagnosis for small UAVs via machine learning. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pages 1–6, St. Petersburg, Sep. 2017. doi: 10.1109/dasc.2017.8102037.

[138] Kicheol Jeong and Seibum Choi. Model-based sensor fault diagnosis of vehicle suspensions with a support vector machine. *International Journal of Automotive Technology*, 20:961–970, 2019. doi: 10.1007/s12239-019-0090-z.

[139] Gao Yun-hong, Zhao Ding, and Li Yi-bo. Small UAV sensor fault detection and signal reconstruction. In *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, pages 3055–3058, Shenyang, 12 2013. doi: 10.1109/mec.2013.6885550.

[140] Fang Deng, Su Guo, Rui Zhou, and Jie Chen. Sensor multifault diagnosis with improved support vector machines. *IEEE Transactions on Automation Science and Engineering*, 14(2):1053–1063, April 2017. doi: 10.1109/tase.2015.2487523.

[141] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647.

[142] Pangun Park, Piergiuseppe Di Marco, Hyejeon Shin, and Junseong Bang. Fault detection and diagnosis using combined autoencoder and long short-term memory network. *Sensors*, 19, 2019. doi: 10.3390/s19214612.

[143] Li Jiang, Zhihuan Song, Zhiqiang Ge, and Junghui Chen. Robust self-supervised model and its application for fault detection. *Industrial & Engineering Chemistry Research*, 56(26):7503–7515, 2017. doi: 10.1021/acs.iecr.7b00949.

[144] Geoffrey E. Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-35289-8_32.

[145] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, August 2002. doi: 10.1162/089976602760128018.

[146] Xin Huang, Xiaodong Zhang, Yiwei Xiong, Hongcheng Liu, and Yingjie Zhang. A novel intelligent fault diagnosis approach for early cracks of turbine blades via improved deep belief network using three-dimensional blade tip clearance. *IEEE Access*, 9:13039–13051, 2021. doi: 10.1109/access.2021.3052217.

[147] Pandit Dash, Bighnaraj Naik, Janmenjoy Nayak, and Vimal Shanmuganthan. Deep belief network-based probabilistic generative model for detection of robotic manipulator failure execution. *Soft Computing*, 01 2021. doi: 10.1007/s00500-021-05572-0.

[148] Saibo Xing, Yaguo Lei, Shuhui Wang, and Feng Jia. Distribution-invariant deep belief network for intelligent fault diagnosis of machines under new working conditions. *IEEE Transactions on Industrial Electronics*, 68(3):2617–2625, 2021. doi: 10.1109/tie.2020.2972461.

[149] Xue-Mei Chen, Chun-Xue Wu, Yan Wu, Nai-xue Xiong, Ren Han, Bo-Bo Ju, and Sheng Zhang. Design and analysis for early warning of rotor UAV based on data-driven DBN. *Electronics*, 8 (11), 2019. doi: 10.3390/electronics8111350.

[150] Jake Lever, Martin Krzywinski, and Naomi Altman. Principal component analysis. *Nature Methods*, 14(7):641–642, 07 2017. doi: 10.1038/nmeth.4346.

[151] Zhong-Gai Zhao and Fei Liu. On-line nonlinear process monitoring using kernel principal component analysis and neural network. In *Advances in Neural Networks*, pages 945–950, Chengdu, China, 2006. Springer Berlin Heidelberg.

[152] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3), June 2011. doi: 10.1145/1970392.1970395.

[153] Ivan Portnoy, Kevin Melendez, Horacio Pinzon, and Marco Sanjuan. An improved weighted recursive PCA algorithm for adaptive fault detection. *Control Engineering Practice*, 50:69–83, 2016. doi: 10.1016/j.conengprac.2016.02.010.

[154] Xueqin Liu, Uwe Kruger, Tim Littler, Lei Xie, and Shuqing Wang. Moving window kernel PCA for adaptive monitoring of nonlinear processes. *Chemometrics and Intelligent Laboratory Systems*, 96(2):132–143, 2009. doi: 10.1016/j.chemolab.2009.01.002.

[155] Chun-Mei Feng, Ying-Lian Gao, Jin-Xing Liu, Chun-Hou Zheng, Sheng-Jun Li, and Dong Wang. A simple review of sparse principal components analysis. In *Intelligent Computing Theories and Application*, pages 374–383, Lanzhou, 2016. Springer International Publishing. doi: 10.1007/978-3-319-42294-7_33.

[156] Lei Xie, Xiao-Zhong Lin, and Jiusun Zeng. Shrinking principal component analysis for enhanced process monitoring and fault isolation. *Industrial & Engineering Chemistry Research*, 52:17475–17486, 2013. doi: 10.1021/ie401030t.

[157] Xiaofang Wang, Xuehan Xiong, Maxim Neumann, AJ Piergiovanni, Michael S. Ryoo, Anelia Angelova, Kris M. Kitani, and Wei Hua. Attentionnas: Spatiotemporal attention cell search for video classification. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 449–465, Cham, 2020. Springer International Publishing. doi: 10.1007/978-3-030-58598-3_27.

[158] Jong-Min Lee, ChangKyoo Yoo, Sang Wook Choi, Peter A. Vanrolleghem, and In-Beum Lee. Nonlinear process monitoring using kernel principal component analysis. *Chemical Engineering Science*, 59(1):223–234, 2004. doi: 10.1016/j.ces.2003.09.012.

[159] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, 01 2015.

[160] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, Ethiopia, 2020.

[161] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768, 2020.

[162] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. In *NIPS*, pages 2211–2221, Long Beach, CA, 2017. doi: 10.5555/3294771.3294982.

[163] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4055–4064, Stockholmsmässan, Stockholm Sweden, 07 2018. PMLR.

[164] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations (ICLR)*, Ethiopia, 2020.

[165] Irwan Bello, Barret Zoph, Quoc Le, Ashish Vaswani, and Jonathon Shlens. Attention augmented convolutional networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3285–3294, Seoul, 2019. doi: 10.1109/iccv.2019.00338.

[166] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with Transformers. In *Computer Vision – ECCV 2020*, pages 213–229, Glasgow, 2020. Springer International Publishing. doi: 10.1007/978-3-030-58452-8_13.

[167] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. In *NeurIPS*, Vancouver, Canada, 2019.

[168] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Comput. Surv.*, 54(10s), sep 2022. doi: 10.1145/3505244.

[169] Tian Wang, Meina Qiao, Mengyi Zhang, Yi Yang, and Hichem Snoussi. Data-driven prognostic method based on self-supervised learning approaches for fault detection. *Journal of Intelligent Manufacturing*, 31, 10 2020. doi: 10.1007/s10845-018-1431-x.

[170] Seyed M. Mousavi, William L. Ellsworth, Weiqiang Zhu, Lindsay Y. Chuang, and Gregory C. Beroza. Earthquake transformer—an attentive deep-learning model for simultaneous earthquake detection and phase picking. *Nature Communications*, 11, 2020. doi: 10.1038/s41467-020-17591-w.

[171] Woon S. Lee, Doris L. Grosh, Frank A. Tillman, and Chang H. Lie. Fault tree analysis, methods, and applications: A review. *IEEE Transactions on Reliability*, R-34(3):194–203, 1985. doi: 10.1109/tr.1985.5222114.

[172] Georgios Kladis, John Economou, Antonios Tsourdos, Brian White, and Kevin Knowles. Fault diagnosis with matrix analysis for electrically actuated unmanned aerial vehicles. *Proceedings of The Institution of Mechanical Engineers Part G-journal of Aerospace Engineering*, 223:543–563, 08 2009. doi: 10.1243/09544100jaero422.

[173] Shi-Ning Ju, Cheng-Liang Chen, and Chuei-Tin Chang. Constructing fault trees for advanced process control systems application to cascade control loops. *IEEE Transactions on Reliability*, 53(1):43–60, 2004. doi: 10.1109/tr.2004.823849.

[174] Chang Jun Lee, Gibaek Lee, Chonghun Han, , and En Sup Yoon. A hybrid model for fault diagnosis using model based approaches and support vector machine. *Journal of Chemical Engineering of Japan*, 39(10):1085–1095, 2006. doi: 10.1252/jcej.39.1085.

[175] Gibaek Lee. Multiple-fault diagnosis based on system decomposition and dynamic PLS. *Industrial & Engineering Chemistry Research*, 42, 10 2003. doi: 10.1021/ie030084v.

[176] Wang Xiaolei, Yu Zhengning, Niu Xuemin, Lu Xianfeng, Yang Hao, and Liu Zhongjiawen. Combination multiple faults diagnosis method applied to the aero-engine based on improved signed directed graph. In *2019 4th International Conference on Measurement, Information and Control (ICMIC)*, pages 1–10, Harbin, China, 2019. doi: 10.1109/icmic48233.2019.9068567.

[177] Shu-Hsien Liao. Expert system methodologies and applications—a decade review from 1995 to 2004. *Expert Systems with Applications*, 28(1):93–103, 2005. doi: 10.1016/j.eswa.2004.08.003.

[178] Yuan Bo and Yang Jun. The application of a new intelligence expert system in the complex damage faults diagnosis of UAV's control surface. In *The Proceedings of the Multiconference on "Computational Engineering in Systems Applications"*, volume 2, pages 2062–2068, Beijing, 2006. doi: 10.1109/cesa.2006.4281978.

[179] Jie Zhang and Peter D. Roberts. Process fault diagnosis with diagnostic rules based on structural decomposition. *Journal of Process Control*, 1(5):259–269, 1991. ISSN 0959-1524. doi: 10.1016/0959-1524(91)85017-d.

[180] Shendy M. El-Shal and Alan S. Morris. A fuzzy expert system for fault detection in statistical process control of industrial processes. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(2):281–289, 2000. doi: 10.1109/5326.868449.

[181] Markus Lappe. *Optic flow*, pages 3035–3039. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-540-29678-2_4245.

[182] Hugh C. Longuet-Higgins and Kvetoslav Prazdny. The interpretation of a moving retinal image. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 208:385 – 397, 1980. doi: 10.1098/rspb.1980.0057.

[183] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1): 185–203, 1981. doi: 10.1016/0004-3702(81)90024-2.

[184] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. doi: 10.5555/1623264.1623280.

[185] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, Manchester, 1988.

[186] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision – ECCV 2006*, pages 430–443, Graz, Austria, 2006. Springer Berlin Heidelberg. doi: 10.1007/11744023_34.

[187] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, Seattle, WA, 1994. doi: 10.1109/cvpr.1994.323794.

[188] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi: 10.1007/3-540-45103-X_50.

[189] Bruno Herissé, Tarek Hamel, Robert Mahony, and François Russotto. Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow. *IEEE Transactions on Robotics*, 28(1):77–89, 2012. doi: 10.1109/tro.2011.2163435.

[190] Guido C.H.E. de Croon, Hann W. Ho, Christophe De Wagter, Erik-Jan van Kampen, Bart Remes, and Qiping P. Chu. Optic-flow based slope estimation for autonomous landing. *International Journal of Micro Air Vehicles*, 5(4):287–297, 2013. doi: 10.1260/1756-8293.5.4.287.

[191] Hann W. Ho, Guido C. H. E. de Croon, Erik-Jan van Kampen, Qiping Chu, and Max Mulder. Adaptive gain control strategy for constant optical flow divergence landing. *IEEE Transactions on Robotics*, 34(2):508–516, 2018. doi: 10.1109/tro.2018.2817418.

[192] Guido Croon. Monocular distance estimation with optical flow manoeuvres and efference copies: A stability-based strategy. *Bioinspiration & Biomimetics*, 11, 01 2016. doi: 10.1088/1748-3190/11/1/016004.

[193] Bas Hordijk, Kirk Scheper, and Guido Croon. Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, 35:69–90, 01 2018. doi: 10.1002/rob.21764.

[194] Farid Kendoul, Isabelle Fantoni, and Kenzo Nonami. Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57(6):591–602, 2009. doi: 10.1016/j.robot.2009.02.001.

[195] Francesco Valenti, Domenico Giaquinto, Luigi Musto, Andrea Zinelli, Massimo Bertozzi, and Alberto Broggi. Enabling computer vision-based autonomous navigation for unmanned aerial vehicles in cluttered GPS-denied environments. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3886–3891, Maui, HI, 2018. doi: 10.1109/itsc.2018.8569695.

[196] Luis Arreola, Andres Montes de Oca, Alejandro Flores, Jose Sanchez, and Gerardo Flores. Improvement in the UAV position estimation with low-cost GPS, INS and vision-based system: Application to a quadrotor UAV. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1248–1254, Dallas, TX, 2018. doi: 10.1109/icuas.2018.8453349.

[197] Vikrant More, Hitendra Kumar, Sarthak Kaingade, Pradeep Gaidhani, and Nitin Gupta. Visual odometry using optic flow for unmanned aerial vehicles. In *2015 International Conference on Cognitive Computing and Information Processing(CCIP)*, pages 1–6, Noid, India, 2015. doi: 10.1109/ccip.2015.7100731.

[198] Kent Rosser, Tran Nguyen, Philip Moss, and Javaan Chahl. Low complexity visual UAV track navigation using long□wavelength infrared. *Journal of Field Robotics*, 01 2021. doi: 10.1002/rob.22015.

[199] Stefan Hrabar, Gaurav S. Sukhatme, Peter I. Corke, Kane Usher, and Jonathan Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a UAV. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3309–3316, Edmonton, 2005. doi: 10.1109/iros.2005.1544998.

[200] Pinghai Gao, Daibing Zhang, Qiang Fang, and Shaogang Jin. Obstacle avoidance for micro quadrotor based on optical flow. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 4033–4037, Chongqing, 2017. doi: 10.1109/ccdc.2017.7979206.

[201] Volker Grabe, Heinrich H. Bülthoff, and Paolo R. Giordano. On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow. In *2012 IEEE International Conference on Robotics and Automation*, pages 491–497, Saint Paul, MN, 2012. doi: 10.1109/icra.2012.6225328.

[202] Shijie Zhang, Xiangtian Zhao, and Botian Zhou. Robust vision-based control of a rotorcraft UAV for uncooperative target tracking. *Sensors*, 20:3474, 06 2020. doi: 10.3390/s20123474.

[203] Yanhua Shao, Wenfeng Li, Hongyu Chu, Zhiyuan Chang, Xiaoqiang Zhang, and Huayi Zhan. A multitask cascading CNN with multiscale infrared optical flow feature fusion-based abnormal crowd behavior monitoring UAV. *Sensors*, 20(19), 2020. doi: 10.3390/s20195550.

[204] Zhongyuan Chen, Wanchun Chen, Xiaoming Liu, and Chuang Song. Fault-tolerant optical flow sensor/SINS integrated navigation scheme for MAV in a GPS-denied environment. *J. Sensors*, 2018:1–17, 2018. doi: 10.1155/2018/9678505.

[205] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *Int. J. Rob. Res.*, 34(3): 314–334, March 2015. doi: 10.1177/0278364914554813.

[206] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018. doi: 10.1109/tro.2018.2853729.

[207] Jeffrey Delmerico and Davide Scaramuzza. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2502–2509, Brisbane, 2018. doi: 10.1109/icra.2018.8460664.

[208] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, Hong Kong, 2014. doi: 10.1109/icra.2014.6906584.

[209] Guoquan Huang. Visual-inertial navigation: A concise review. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9572–9582, Montreal, Canada, 2019. doi: 10.1109/icra.2019.8793604.

[210] Vincent Casser, S. Pirk, R. Mahjourian, and A. Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *AAAI*, Hawaii, 2019. doi: 10.1609/aaai.v33i01.33018001.

[211] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8976–8985, 2019. doi: 10.1109/iccv.2019.00907.

[212] Javier Selva Castelló. A comprehensive survey on deep future frame video prediction. Master's thesis, Universitat de Barcelona, 2018.

[213] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. AirSim drone racing lab. In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, volume 123 of *Proceedings of Machine Learning Research*, pages 177–191, Vancouver, Canada, 12 2020. PMLR.

[214] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4CV: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, 03 2018. doi: 10.1007/s11263-018-1073-7.

[215] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pages 1147–1157. PMLR, 2020. doi: 10.5167/uzh-193792.

[216] Winter Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman. Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6941–6948, 2019. doi: 10.1109/iros40897.2019.8968116.

[217] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: MORSE. In *2011 IEEE International Conference on Robotics and Automation*, pages 46 – 51, Shanghai, 06 2011. doi: 10.1109/icra.2011.5980252.

[218] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *RotorS – A Modular Gazebo MAV Simulator Framework*, chapter 7, pages 595–625. Springer International Publishing, 01 2016. doi: 10.1007/978-3-319-26054-9_23.

[219] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. In Sven Behnke, Manuela Veloso, Arnoud Visser, and Rong Xiong, editors, *RoboCup 2013: Robot World Cup XVII*, pages 624–631, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi: 10.1007/978-3-662-44468-9_58.

[220] Ori Ganoni and Ramakrishnan Mukundan. A framework for visually realistic multi-robot simulation in natural environment. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2017.

[221] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Detection and tracking meet drones challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/tpami.2021.3119563.

[222] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Qinghua Hu, and Haibin Ling. Vision meets drones: Past, present and future. *arXiv preprint arXiv:2001.06303*, 2020.

[223] A. Kouris and C.S. Bouganis. Learning to fly by myself: A self-supervised CNN-based approach for autonomous navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5216–5223, Oct 2018. doi: 10.1109/iros.2018.8594204.

[224] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 6713–6719, 2019. doi: 10.1109/icra.2019.8793887.

[225] Andras Majdik, Charles Till, and Davide Scaramuzza. The Zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*, 36:027836491770223, 04 2017. doi: 10.1177/0278364917702237.

[226] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixão, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021. doi: 10.1016/j.eswa.2020.113816.

[227] Jorge Godoy, V. Jiménez, Antonio Artuñedo, and J. Villagra. A grid-based framework for collective perception in autonomous vehicles. *Sensors (Basel, Switzerland)*, 21(3), 2021. doi: 10.3390/s21030744.

[228] Raffaella Carloni, Vincenzo Lippiello, Massimo D'Auria, Matteo Fumagalli, A.Y. Mersha, Stefano Stramigioli, and Bruno Siciliano. Robot vision: Obstacle-avoidance techniques for unmanned aerial vehicles. *Robotics & Automation Magazine, IEEE*, 20:22–31, 12 2013. doi: 10.1109/mra.2013.2283632.

[229] Marc Steven Krämer and Klaus-Dieter Kuhnert. Multi-sensor fusion for UAV collision avoidance. In *Proceedings of the 2018 2nd International Conference on Mechatronics Systems and Control Engineering*, ICMSCE 2018, page 5–12, New York, NY, USA, 2018. Association for Computing Machinery. doi: 10.1145/3185066.3185081.

[230] A. Toma, H. Hsueh, H. Jaafar, R. Murai, P. J. Kelly, and S. Saeedi. Pathbench: A benchmarking platform for classical and learned path planning algorithms. In *2021 18th Conference on Robots and Vision (CRV)*, pages 79–86, Los Alamitos, CA, USA, May 2021. IEEE Computer Society. doi: 10.1109/crv52889.2021.00019.

[231] Christian Zammit and Erik-Jan Van Kampen. Comparison of A* and RRT in real–time 3D path planning of UAVs. *Unmanned Systems*, 10(02):129–146, 2022. doi: 10.2514/6.2020-0861.

[232] Adham Atyabi and David Powers. Review of classical and heuristic-based navigation and path planning approaches. *International Journal of Advancements in Computing Technology (IJACT)*, 5(14), 2013.

[233] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, USA, 2006. doi: 10.1017/cbo9780511546877.

[234] Howie Choset. Robotic motion planning lectures, September 2010.

[235] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016. doi: 10.1109/tiv.2016.2578706.

[236] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, and A. Jagadeesh. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019. doi: 10.1016/j.dt.2019.04.011.

[237] F. Arambula Cosío and M.A. Padilla Castañeda. Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, 40(9):1141–1156, 2004. doi: https://doi.org/10.1016/j.mcm.2004.05.001.

[238] Xin Chen and Yangmin Li. Smooth path planning of a mobile robot using stochastic particle swarm optimization. In *2006 International Conference on Mechatronics and Automation*, pages 1722–1727, 2006. doi: 10.1109/icma.2006.257474.

[239] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016. doi: 10.1016/j.robot.2016.08.001.

[240] Mohd Nadhir Ab Wahab, Samia Nefti-Meziani, and Adham Atyabi. A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annual Reviews in Control*, 50: 233–252, 2020. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2020.10.001.

[241] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

[242] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

[243] Kesheng Wang. B-splines joint trajectory planning. *Computers in Industry*, 10(2):113–122, 1988. ISSN 0166-3615. doi: https://doi.org/10.1016/0166-3615(88)90016-4.

[244] Sihao Sun, Coen C. de Visser, and Qiping Chu. Quadrotor gray-box model identification from high-speed flight data. *Journal of Aircraft*, 56(2):645–661, 2019. doi: 10.2514/1.c035135.

[245] Ewoud J. J. Smeur, Qiping Chu, and Guido C. H. E. de Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461, 2016. doi: 10.2514/1.G001490.

[246] P. Schofield. Computer simulation studies of the liquid state. *Computer Physics Communications*, 5(1):17–23, 1973. doi: 10.1016/0010-4655(73)90004-0.

[247] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 1982. doi: 10.1063/1.442716.

[248] Shuo Li, Michaël M.O.I. Ozo, Christophe De Wagter, and Guido C.H.E. de Croon. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *Robotics and Autonomous Systems*, 133:103621, 2020. doi: https://doi.org/10.1016/j.robot.2020.103621.

[249] Remus C. Avram, Xiaodong Zhang, and Mohsen Khalili. Quadrotor actuator fault diagnosis with real-time experimental results. In *Annual Conference of the PHM Society*, volume 8, 10 2016. doi: 10.36001/phmconf.2016.v8i1.2504.

[250] Behnam Ghalamchi, Zheng Jia, and Mark Wilfried Mueller. Real-time vibration-based propeller fault diagnosis for multicopters. *IEEE/ASME Transactions on Mechatronics*, 25(1):395–405, 2020. doi: 10.1109/tmech.2019.2947250.

[251] Rajan Gill and Raffaello D'Andrea. Computationally efficient force and moment models for propellers in UAV forward flight applications. *Drones*, 3(4), 2019. doi: 10.3390/drones3040077.

[252] Robert Niemiec and Feny Gandhi. Effects of inflow model on simulated aeromechanics of a quadrotor helicopter. In *Proceedings of the 2016 72nd American Helicopter Society (AHS) International Annual Forum*, 05 2016.

[253] J. J. Howlett. *UH-60A Black Hawk engineering simulation program*, volume 1. NTIS, Springfield, Va., 1981.

[254] Dale Pitt and David Peters. Theoretical prediction of dynamic inflow derivatives. *Vertica*, 5, 01 1981.

[255] Fred White and Bruce B. Blake. Improved method of predicting helicopter control response and gust sensitivity. In *Proceedings of the 1979 35th Annual Forum of American Helicopter Society*, 5 1979.

[256] J.M. Drees. A theory of airflow through rotors and its application to some helicopter problems. *Journal of the Helicopter Association of Great Britain, 3*, 2:79–104, 07 1949.

[257] Rajan Gill and Raffaello D'Andrea. Propeller thrust and drag in forward flight. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 73–79, 2017. doi: 10.1109/CCTA.2017.8062443.

[258] Richard H. Byrd, Mary E. Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999. doi: 10.1137/S1052623497325107.

[259] Abdel-Bary Mostafa and Mostafa Ebrahim. 3D laser scanners' techniques overview. *International Journal of Science and Research (IJSR)*, 4:5–611, 10 2015. doi: 10.1007/s41062-021-00550-9.

[260] Juan-Carlos Perez-Cortes, Alberto Perez, Sergio Saez-Barona, Jose-Luis Guardiola, Ismael Salvador Igual, and Sergio Sáez-Barona. A system for in-line 3D inspection without hidden surfaces. *Sensors*, 18:2993, 09 2018. doi: 10.3390/s18092993.

[261] Abdul Qadir Bhatti, Abdul Wahab, and Wadea Sindi. An overview of 3D laser scanning techniques and application on digitization of historical structures. *Innovative Infrastructure Solutions*, 6(4):186, 7 2021. doi: 10.1007/s41062-021-00550-9.

[262] Daniel Viguera Leza. *Development of a blade element method for CFD simulations of helicopter rotors using the actuator disk approach*. Master's thesis, Delft University of Technology, 2018.

[263] Sammy Omari, Minh-Duc Hua, Guillaume Ducard, and Tarek Hamel. Nonlinear control of VTOL UAVs incorporating flapping dynamics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2419–2425, 2013. doi: 10.1109/iros.2013.6696696.

[264] Pinting Zhang and Shuhong Huang. Review of aeroelasticity for wind turbine: Current status, research focus and future perspectives. *Frontiers in Energy*, 5(4):419–434, 2011. doi: 10.1007/s11708-011-0166-6.

[265] Tony Burton, David Sharpe, Nick Jenkins, and Ervin Bossanyi. *Wind energy handbook*, pages 511 – 558. John Wiley & Sons, Ltd, 04 2002. doi: 10.1002/0470846062.ch9.

[266] Alfred Gessow and Garry C. Myers. *Aerodynamics of the helicopter*. F. Ungar Pub. Co., New York, 1967. ISBN 0-8044-4275-4.

[267] K. W. Mangler and H. B. Squire. *The induced velocity field of a rotor*. Ministry of Supply, Aeronautical Research Council, London, 1950.

[268] W. Z. Stepniewski. Rotary-wing aerodynamics, 1984.

[269] Robert A. Ormiston. *An actuator disk theory for rotor wake induced velocities*. Advisory Group for Aerospace Research and Development (AGARD-CP-111), Aerodynamics of Rotary Wings, 1972. doi: 10.1007/978-3-030-05455-7_2-2.

[270] Robert A. Ormiston. Induced power of the helicopter rotor. In *60th Annual Forum of the American Helicopter Society International*, pages 33–53, 7–10 Jun 2004.

[271] Peter Rowland Payne. *Helicopter dynamics and aerodynamics*, volume 63. Pitman & Sons, 1959. doi: 10.1017/S0368393100071728.

[272] D. A. Peters and N. Haquang. Dynamic inflow for practical applications. *Journal of the American Helicopter Society*, 33(4):64–68, 1988. doi: 10.4050/jahs.33.64.

[273] David A. Peters, David Doug Boyd, and Cheng He. Finite-state induced-flow model for rotors in hover and forward flight. *Journal of The American Helicopter Society*, 34:5–17, 1989. doi: 10.4050/jahs.34.5.

[274] David A. Peters and Cheng He. Correlation of measured induced velocities with a finite-state wake model. *Journal of The American Helicopter Society*, 36:59–70, 1991. doi: 10.4050/jahs.36.59.

[275] JG Zhao, J.V.R. Prasad, and David Peters. Rotor dynamic wake distortion model for helicopter maneuvering flight. *Journal of The American Helicopter Society*, 49:414–424, 10 2004. doi: 10.4050/jahs.49.414.

[276] J. Gordon Leishman, Mahendra J. Bhagwat, and Ashish Bagai. Free-vortex filament methods for the analysis of helicopter rotor wakes. *Journal of Aircraft*, 39(5):759–775, 2002. doi: 10.2514/2.3022.

[277] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, 55(7):2215–2227, 2017. doi: 10.2514/1.j055595.

[278] Pedro Stefanin Volpiani, Morten Meyer, Lucas Franceschini, Julien Dandois, Florent Renac, Emeric Martin, Olivier Marquet, and Denis Sipp. Machine learning-augmented turbulence modeling for rans simulations of massively separated flows. *Physical Review Fluids*, 6:064607, 6 2021. doi: 10.1103/physrevfluids.6.064607.

[279] N. R. Lomb. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, 39(2):447–462, 2 1976. doi: 10.1007/bf00648343.

[280] Jeffrey D. Scargle. Studies in astronomical time series analysis. II. Statistical aspects of spectral analysis of unevenly spaced data. *Astrophysical Journal*, 263:835–853, 12 1982. doi: 10.1086/160554.

[281] Jeffrey D. Scargle. Studies in astronomical time series analysis. III. Fourier transforms, autocorrelation functions, and cross-correlation functions of unevenly spaced data. *Astrophysical Journal*, 343:874, 8 1989. doi: 10.1086/167757.

[282] R. V. Baluev. Assessing the statistical significance of periodogram peaks. *Monthly Notices of the Royal Astronomical Society*, 385(3):1279–1285, March 2008. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2008.12689.x.

[283] Zechmeister, M. and Kürster, M. The generalised Lomb-Scargle periodogram - a new formalism for the floating-mean and keplerian periodograms. *A&A*, 496(2):577–584, 2009. doi: 10.1051/0004-6361:200811296.

[284] G. Larry Bretthorst. Frequency estimation and generalized Lomb-Scargle periodograms. In *Statistical Challenges in Astronomy*, pages 309–329, New York, NY, 2003. Springer New York. ISBN 978-0-387-21529-7.

[285] Earl F. Glynn, Jie Chen, and Arcady R. Mushegian. Detecting periodic patterns in unevenly spaced gene expression time series using Lomb–Scargle periodograms. *Bioinformatics*, 22(3): 310–316, November 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti789.

[286] Junhwa Hur and Stefan Roth. Optical flow estimation in the deep learning age. In *Modelling Human Motion: From Human Perception to Robot Design*, pages 119–140. Springer International Publishing, Cham, 2020. doi: 10.1007/978-3-030-46732-6_7.

[287] Syed Tafseer Haider Shah and Xiang Xuezhi. Traditional and modern strategies for optical flow: an investigation. *SN Applied Sciences*, 3(3):289, 2 2021. doi: 10.1007/s42452-021-04227-x.

[288] Mingliang Zhai, Xuezhi Xiang, Ning Lv, and Xiangdong Kong. Optical flow and scene flow estimation: A survey. *Pattern Recognition*, 114:107861, 2021. doi: 10.1016/j.patcog.2021.107861.

[289] D. Butler, J. Wulff, G. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012. doi: 10.1007/978-3-642-33783-3_44.

[290] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3D estimation of vehicles and scene flow. In *Proc. of the ISPRS Workshop on Image Sequence Analysis (ISA)*, pages 427–434, 2015. doi: 10.5194/isprsannals-ii-3-w5-427-2015.

[291] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:60–76, 2018. doi: 10.1016/j.isprsjprs.2017.09.013.

[292] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, pages 8934–8943, 06 2018. doi: 10.1109/cvpr.2018.00931.

[293] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 402–419, Cham, 2020. Springer International Publishing. doi: 10.1007/978-3-030-58536-5_24.

[294] Jianyuan Wang, Yiran Zhong, Yuchao Dai, Kaihao Zhang, Pan Ji, and Hongdong Li. Displacement-invariant matching cost learning for accurate optical flow estimation. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, page 15220–15231, Red Hook, NY, USA, 2020. Curran Associates Inc. doi: 10.5555/3495724.3497000.

[295] Till Kroeger, Radu Timofte, Dengxin Dai, and Luc Van Gool. Fast optical flow using dense inverse search. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 471–488, Cham, 2016. Springer International Publishing. doi: 10.1007/978-3-319-46493-0_29.

[296] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015. doi: 10.1109/iccv.2015.316.

[297] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 06 2016. doi: 10.1109/cvpr.2016.438.

[298] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi: 10.1007/3-540-45103-x_50.

[299] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 10 2019. doi: 10.1109/iccv.2019.00140.

[300] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/cvpr.2009.5206848.

[301] J. Durbin and G. S. Watson. Testing for serial correlation in least squares regression: I. *Biometrika*, 37(3/4):409–428, 1950. doi: 10.2307/2332391.

[302] Greta Ljung and G. Box. On a measure of lack of fit in time series models. *Biometrika*, 65, 08 1978. doi: 10.1093/biomet/65.2.297.

[303] T. S. Breusch. Testing for autocorrelation in dynamic linear models. *Australian Economic Papers*, 17(31):334–355, 1978. doi: 10.1111/j.1467-8454.1978.tb00635.x.

[304] L. G. Godfrey. Testing against general autoregressive and moving average error models when the regressors include lagged dependent variables. *Econometrica*, 46(6):1293–1301, 1978. doi: 10.2307/1913829.