Loss-of-Control Prediction

Quadcopter Loss-of-Control Prediction Using Recurrent Neural Networks

Master of Science Thesis

Anique Altena

Delft University of Technology

TU Delft

Loss-of-Control Prediction

Quadcopter Loss-of-Control Prediction Using Recurrent Neural Networks

by

Anique Altena

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended on Friday January 21st 2022 at 09:30.

Student Number:	4554345	
Project Duration:	March 2021 - January 2022	
Thesis Committee:	Prof. Dr. G.C.H.E. de Croon	Chairman
	Dr. Ir. C.C. de Visser	Daily Supervisor
	Dr. Ir. M.F.M. Hoogreef	External Member

TU Delft

Abstract

Unmanned aerial vehicles (UAVs), also known as drones, are increasingly used for a variety of professional and amateur applications. However, due to the extreme agility of these vehicles, they easily suffer from loss-of-control (LOC). On-board LOC prevention systems should be designed which require low memory and computing power, as these resources are rare on most drones. Data-driven techniques fit these requirements.

Recurrent neural networks (RNNs) are data-driven techniques that have characteristics that are desirable for the loss-of-control prediction problem. These models can detect patterns from sequential data automatically. Traditional RNNs can identify short-term time-dependent features in time-series and advanced RNN structures can recognise even longer time-dependent patterns. The aim of this research is to identify which RNN model is most suitable for LOC prediction and if it can generalise loss-of-control prediction of quadcopters for different scenarios, where the aerodynamic characteristics, operating conditions, quadcopter type or loss-of-control event are altered compared to the nominal scenario. As it is unknown beforehand how complex and time-dependent the failure features are that are present in the sensor data, four RNN architectures will be tested to identify which model performs best: long short-term memory (LSTM) network, bidirectional LSTM (BiLSTM) network, LSTM network preceded by a convolutional neural network (CNN-LSTM) and gated recurrent unit (GRU) network.

Real-life flight tests using a Tiny Whoop quadcopter were performed where LOC was initiated by demanding a too high yaw rate of $\pm$2000 deg/s. On-board (sensor) measurements were captured during these flights and no external tracking system or sensors were necessary to predict an upcoming LOC event. All models were capable of predicting loss-of-control and the commanded rotor values provided the clearest early warning signals for this, as these values showed saturation before the start of the LOC event. For optimal performance, the gyroscopic measurements or accelerations should be used as well.

Different results were found for the various generalisation scenarios. For both changing mass and propeller type, the models underestimated the time to loss-of-control, which was as expected. Therefore, it is possible to compensate for these deviations and the models can thus still be used for LOC prediction when these changes are present. On the other hand, for flying in wind conditions and using a different quadcopter, the results were inconclusive and additional research is necessary to draw conclusions on generalisation performance. The downside of the models is that they can only be used for loss-of-control events that happen by demanding a too high yaw rate and the time to loss-of-control should be similar to the ones from the training set.

Preface

This project started out of an interest in artificial intelligence applied to real-life flight data which contributes to sustainable aviation. Although the ambition was to create a fully working, closed-loop loss-of-control prediction system that actually could make aircraft fly more efficient, I soon realised that I must appreciate the small achievements accomplished on my way to fulfil this. I cannot solve this problem on my own in nine months and I hope that the findings of this project can serve as a starting point for any enthusiastic student, scientist or others who share the same ideal.

This research would not have been possible without the guidance of Coen de Visser. I would like to thank him for giving me the freedom and confidence to direct this thesis project the way I had in mind and for all the discussions and brainstorm meetings we have had. However, I am in particular thankful for his enthusiasm with which he kept repairing the drones I broke at least twice a week during the experimental phase of this research. Without his endless enthusiasm and effort, it would not have been possible to achieve the results we have.

After five and a half years, my time as a student at the TU Delft has come to an end. Throughout these years, I have made many new friends with whom I have gone through all the ups and downs that come along with the life of a student. I am grateful for all of them and hope to keep in touch for many more years. For now, I am excited that my time at the TU Delft has not finished yet and I am looking forward to start my career as a PhD student at the faculty of Aerospace Engineering.

Anique Altena
Delft, January 2022

Contents

List of Abbreviations

Abbreviation	Definition
AE	Auto-Encoder
AFTCS	Active Fault Tolerant Control System
ANN	Artificial Neural Network
API	Application Programming Interface
BiLSTM	Bidirectional Long Short-Term Memory
CAST	Commercial Aviation Safety Team
CD	Contrastive Divergence
CG	Centre of Gravity
CNN	Convolutional Neural Network
CSD	Critical Slowing Down
DL	Deep Learning
DNN	Deep Neural Network
DTW	Dynamic Time Warping
EA-FMS	Envelope-Aware Flight Management System
ESC	Electronic Speed Controller
EWS	Early Warning Signals
FAA	Federal Aviation Authority
FC	Flight Controller
FDD	Fault Detection and Diagnosis
FSAM	Flight Safety Assessment and Management
FTCS	Fault Tolerant Control System
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
IMU	Inertial Measurement Unit
JSAT	Joint Safety Analysis Team
kNN	k-Nearest Neighbour
LOC	Loss-of-Control
LSTM	Long Short-Term Memory
ML	Machine Learning
NASA	National Aeronautics and Space Administration
PFTCS	Passive Fault Tolerant Control System
PHM	Prognostics and Health Management
QLC	Quantitative Loss-of-Control Criteria
RBM	Restricted Boltzmann Machine
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RUL	Remaining Useful Life
SFE	Safe Flight Envelope
SFEP	Safe Flight Envelope Protection
SMC	Sliding Mode Controller
SPA	Sparse Auto-Encoder
SRF	Single Rotor Failure
SRUCKF	Square Root Unscented Complementary Kalman Filter
sUAV	Small Unmanned Aerial Vehicle
SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle

List of Figures

List of Tables

1

Introduction

Unmanned Aerial Vehicles (UAVs), better known as drones, have gained increased popularity over the past decade for a large variety of applications. The vehicles are used professionally for both civil and military purposes, such as in agriculture, cargo delivery, aerial observation, search and rescue operations and entertainment applications. Beside this, drones are nowadays also used for recreational purposes. According to the Federal Aviation Authority, in 2019 1.32 million small UAVs were registered for recreational purposes in the United States, which is only expected to grow over the next five year [15].

This increasing popularity of drones raises concerns about safe flight of these vehicles, especially for drones used for recreational purposes. Amateur pilots are not aware how powerful UAVs can be and can have trouble dealing with the extreme agility of such vehicles. They are not familiar with the dynamical possibilities and limitations of drones, which can lead to damage or even loss-of-control, leading to crashes if left unaddressed. In a 2017 study, 100 UAV reported mishaps were analysed, out of which 34 were categorised as loss-of-control, making it the largest category of UAV mishaps [6]. These numbers emphasise the importance of more detailed research on loss-of-control prediction and mitigation techniques.

Currently, no techniques exist that can predict loss-of-control on-board of drones. The amount of memory and computing power available in these vehicles is the limiting factor for conventional techniques which rely on flight envelope determination. The aim of this research is therefore to create a data-driven algorithm that can detect dangerous flight and predict the time until the start of a loss-of-control event. Real-life flight tests using a quadcopter UAV are performed to obtain data from failure runs which are used to create this algorithm and test for what different applications this algorithm can be used.

The remainder of this thesis is structured as follows. First, a motivation for this research is provided in Chapter 2, together with the research objective and question. Part I presents the preliminary study that was performed to obtain an understanding of the loss-of-control problem and different data-driven techniques. In Part II, the results of the research are presented by means of a scientific paper. Conclusions and recommendations are provided in Part III. Finally, this thesis ends with the appendices in Part IV, where additional results that were not reported in the paper are written down.

2

Thesis Project

This thesis project focuses on predicting loss-of-control (LOC) for unmanned aerial vehicles. This chapter motivates that this is a relevant research topic. Next to this, the research objective and question will be provided.

2.1. Research Motivation

Loss-of-control is the cause of the majority of the (fatal) accidents in aviation [7]. Due to this, LOC prevention has received major attention since the late 90s. There are many approaches that contribute to LOC prevention, from smart algorithms that can detect loss-of-control to improved pilot training to prevent human error induced loss-of-control. On the other hand, LOC prediction and mitigation are less investigated topics and solutions that are invented cannot tell the exact time until LOC occurs. Being able to predict LOC will happen within a certain time horizon will benefit both safe and sustainable aviation. Safety is improved, as such an algorithm allows for earlier intervention from the flight crew or the autopilot. Sustainable aviation benefits as aircraft can be designed with lower safety margins, as they can fly closer to the boundaries of the safe flight envelope. This is because a LOC prediction system provides early warning signals when flying too close to these boundaries, which allows for enough time to recover to a safe condition. Lower safety margins imply lighter aircraft, which need less fuel and are therefore more sustainable. It is thus worth creating algorithms that can predict the exact time to an upcoming LOC event.

Not only aircraft suffer from loss-of-control. For drones, this is also the main cause for crashes [6], which implies that an algorithm should be created that works for these vehicles as well. However, drones suffer from low computing power and available memory, which means that the proposed method should require only information that is available on-board. Data-driven techniques that use on-board sensor measurements serve as a solution. Another benefit of these techniques is that they can learn patterns from data automatically. This is useful as it is usually unknown what exact combination of vehicle behaviour and operating conditions causes a LOC event. One point of concern is that a data-driven model is trained on previous failure runs and it is therefore unknown to what extent it can generalise for scenarios that deviate from these runs in terms of aircraft characteristics, operational conditions and LOC events. This will be an essential part of the thesis project.

2.2. Research Question

To investigate if LOC prediction using a data-driven algorithm is possible and if such a model can generalise, simulations and real-life flight tests will be conducted using a quadcopter UAV. It is convenient to use a quadcopter as test case, because this is less costly and safer, as many failure runs should be done to validate the working of a LOC prediction algorithm. Beside this, quadcopters are the most commonly used multicopter UAVs, which means easy access and support for these vehicles is available. To research generalisation capabilities of the model, representative configurations and conditions that happen often during aircraft nominal flight will be chosen. These are changes in weight and centre of gravity position, changes in wind speed and changes in LOC scenarios. The weight of an aircraft

changes during flight due to fuel usage. This is also the main reason why the centre of gravity location alters during flight. Wind speed also changes continuously during flight. Lastly, the course of each aircraft accident is different, resulting in a large variety of LOC scenarios. If positive results are achieved for these varying conditions and scenarios on the quadcopter test case, the same methodology could work on aircraft in nominal flight facing the same variations.

With this in mind, the following research objective is proposed:

> The objective of this thesis project is to determine whether data-driven algorithms can generalise loss-of-control prediction of quadcopters with varying aerodynamic characteristics, operating in different conditions and for different LOC events. It is hypothesised that this can be done by training a neural network on nominal quadcopter configuration failure data and use this network to predict loss-of-control for quadcopters in different configurations and for different operational conditions and envelope violations.

Based on this research objective, the main research question is set to:

> Can a neural network generalise loss-of-control prediction of quadcopters with aerodynamic characteristics, operational conditions and LOC scenarios representative for aircraft in nominal condition?

The main question consists of several parts that should be investigated separately by means of subquestions. The first part focuses on identifying data-driven methods that can be used for LOC prediction. The second part focuses on neural networks, as it is expected that these are most suited for LOC prediction. Therefore, state-of-the-art neural network architectures are investigated and the most accurate architecture is identified. Furthermore, there are three more parts, all related to different conditions and scenarios that should be investigated as described in the main question. The subquestions are listed below:

1. How is loss-of-control defined for a nominally operating quadcopter?
2. Which data-driven approaches can be used for loss-of-control prediction?
   (a) Which data-driven techniques exist in literature?
   (b) Which techniques are most suitable for LOC prediction?
   (c) Which sensor data and/or operational conditions should be considered for prediction?
3. What neural network architecture performs best in predicting loss-of-control?
   (a) Which NN architectures exist in literature?
   (b) Can NN architectures be used for real-time (off-board) deployment?
   (c) Which architecture achieves highest prediction accuracy?
4. Can NN algorithms generalise the prediction of loss-of-control for quadcopters with different weight and CG location?
   (a) How much can the weight of the quadcopter vary before prediction accuracy is not sufficient anymore, where the CG location is attempted to remain the same?
   (b) How far can the CG location be altered before prediction accuracy is not sufficient anymore, where the quadcopter weight is attempted to remain the same?
   (c) Can both the weight and CG location be changed while still predicting LOC sufficiently?
5. Can NN algorithms generalise the prediction of loss-of-control for quadcopters flying in different wind conditions?
   (a) How much wind can be present before prediction accuracy is not sufficient anymore?
   (b) Does the wind direction influence the prediction accuracy?

6. Can NN algorithms generalise the prediction of loss-of-control for quadcopters subject to different loss-of-control scenarios?
    (a) Can a NN trained to predict LOC by demanding a too high yaw rate also predict LOC when demanding a too high pitch rate?
    (b) Can a NN trained to predict LOC by demanding a too high yaw rate also predict LOC when demanding a too high roll rate?

The first and second subquestions and the first part of the third subquestion will be researched through a literature review that is reported in Part I. The remainder of question 3 will be investigated using a quadcopter simulation environment and dedicated deep learning model creation and training platform. Results of this are presented in Part IV, Appendix A. Questions 4 to 6 will be inspected using the same quadcopter simulator and deep learning platform and after that will be validated with real-life flight tests. The most important results of these flight tests are presented in a scientific article in Part II.

Part I

Preliminary Thesis Report

3

Unmanned Aerial Vehicle Safety

In present day, Unmanned Aerial Vehicles (UAVs), more commonly referred to as drones, are used more often in civil and military applications, such as cargo delivery, search and rescue operations, aerial observation, precision architecture and other applications. The drone market is expected to keep growing in the near future, facilitated by dropping prices, high-level equipment such as built-in cameras and relatively easy manoeuvring [15]. The Federal Aviation Authority (FAA) expects that the recreational small UAV (sUAV) fleet in the United States will continue growing over the next few years, from 1.32 million in 2019 to 1.48 million in 2024 [15]. This increase poses safety threats, as users are usually not trained drone pilots and therefore unaware of unsafe situations. From this, ensuring safe flight of UAVs has recently become an emerging field of interest. Current safety enhancement methods must be explored, as well as future opportunities to improve safety. This chapter first discusses a currently often used tool for flight safety enhancement: fault tolerant control systems. After that, two other concepts that are closely related to safe flight are elaborated upon: safe flight envelope and loss-of-control.

3.1. Fault Tolerant Control System

Fault Tolerant Control Systems (FTCS) are one of the possible solutions for enhancing UAV safety. The aim of such a control system is to maintain stability and performance of the UAV in case any anomaly or failure occurs [42]. This system can be classified into two categories: Passive Fault Tolerant Control Systems (PFTCS) and Active Fault Tolerant Control Systems (AFTCS). Passive systems contain fixed controllers, which are robust against a predefined class of faults [42]. On the other hand, active controllers have a reconfigurable controller that can react to the failures present in the system, such that acceptable stability and performance are preserved under any faulty condition [42]. The latter is robust to a broader range of failures, but requires on-board, real-time Fault Detection and Diagnosis (FDD).

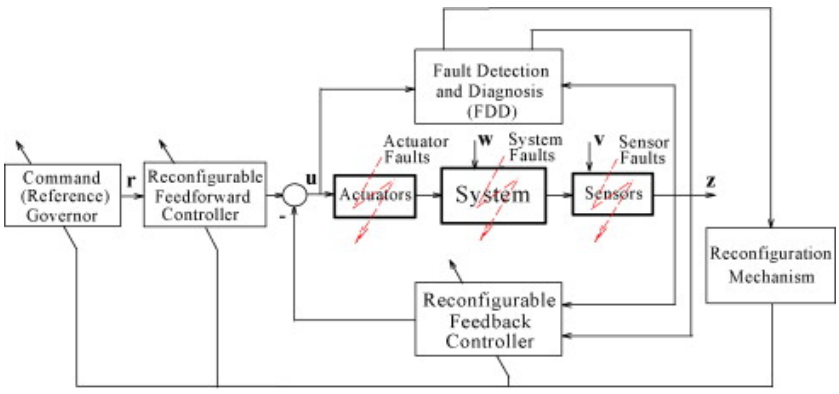


Figure 3.1: Standard structure of an AFTCS [42]

Figure 3.1 shows the structure of an AFTCS. The FDD scheme has a prominent place within this structure, as it drives the reconfiguration of the controllers. The prime goal of the FDD is fault detection, isolation and identification. Two possible ways to do this exist: model-based or data-driven. Hybrid options can be used as well, utilising strengths of both approaches.

Model-based approaches use a mathematical representation of the aircraft or UAV based on physical principles to estimate the expected nominal state of the system. If the true state deviates from the expected state beyond a predetermined threshold, the system is said to be faulty. The downside is that fault identification is not included. Most model-based approaches use Kalman Filter variants for state estimation. Gośliński et al. [19] derived a new type of Kalman filter, Square Root Unscented Complementary Kalman Filter (SRUCKF), for low computational attitude estimation of a quadcopter UAV, which can be used to observe sensor faults. In 2018, Hasan and Johansen [20] cascaded both a nonlinear Thau observer and a linearised Kalman Filter for actuator fault diagnosis. Other state estimation methods are used as well. In a recent study, Mallavalli and Fekih [26] proposed to use an adaptive fuzzy state observer, which can estimate states that are immeasurable by the system's sensors. An advanced Sliding Mode Controller (SMC) uses the outputs of the observer to guarantee stability of a quadcopter subject to multiple actuator faults and external disturbances.

Data-driven approaches utilise data from sensor measurements to detect and identify anomalies. A data-driven model can work in different ways. A first approach works in a similar fashion as the model-based approach. The model uses data to learn the nominal behaviour of the system and during operation it compares the true behaviour with the expected behaviour to detect off-nominal conditions. A second approach identifies features in measurement data that are associated with specific conditions, such as nominal conditions or specific fault conditions. If similar conditions are measured during flight, the model can classify the condition, which directly includes identification. A downside of the latter approach is that only specific faulty conditions can be identified. A large amount of data-driven techniques exist, which can be divided into many different categories. More information on this can be found in Chapter 4.

Although data-driven approaches have their advantages compared to model-based approaches, there are also limitations that must be faced. These approaches require a large training data set containing many failure runs to be able to learn failure features. Another downside is that the generalisation capabilities of these methods are not yet widely investigated. The data used for testing the accuracy of a model is usually similar to the data used for training the model. It is unknown to what extent data-driven models can correctly detect faults outside the training and test domain.

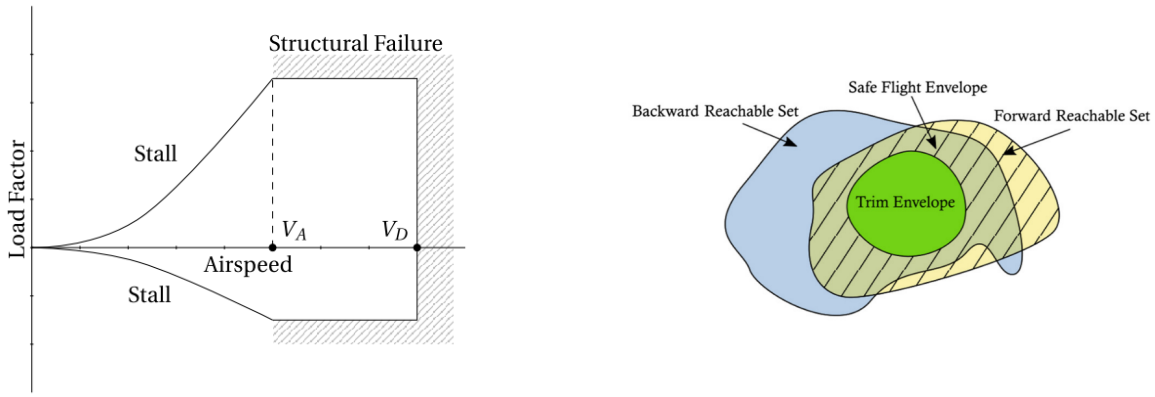
Despite these limitations, many studies exist where data-driven approaches are successfully applied to UAVs for fault detection and diagnosis. In a 2020 paper, Bronz et al. [9] proposed a classical machine learning technique, Support Vector Machine (SVM), for real-time multiclass actuator fault detection in a small fixed-wing UAV. Sadhu et al. [33] used an advanced deep learning approach, combining both a convolutional neural network (CNN) and bidirectional long short-term memory (BiLSTM) network to detect propeller breakdown of a quadcopter. An auto-encoder (AE) is used after fault detection to classify the fault type. To determine the level of degradation of a quadcopter propeller, Manukyan et al. [28] used a more classical data-driven approach. As core technique, they use a k-nearest neighbours (kNN) algorithm that classifies samples based on similarity or a distance measure. As distance measure, the Dynamic Time Warping (DTW) technique is applied, which can deal with time-series of unequal length.

Fault detection and diagnosis is thus a broadly researched topic for UAV safety enhancement. However, an important note should be made here. Aircraft and UAV incidents or accidents do not only occur due to faults present within the system. Another possible cause is flight envelope violation. To prevent this from happening, a more sophisticated FDD system is required, which can also predict the proximity of the UAV to the boundaries of the envelope and take timely actions.

The remainder of this chapter will elaborate upon this aspect of UAV safety enhancement. First, more detail will be given about the flight envelope. After that, one of the possible consequences of flight envelope violation, loss-of-control, will be introduced, which is the key topic of this research.

3.2. Flight Envelope

During flight, aircraft and UAVs must limit themselves in terms of operating conditions, states and manoeuvres combinations. Certain combinations lead to damage or even loss-of-control. The boundaries in which these systems can operate safely define the flight envelope. Traditionally, the flight envelope is defined as ”the area of altitude and airspeed where an airplane is constrained to operate” [16]. An example of such a flight envelope is show in Figure 3.2a. However, this definition is limited to normal operation, not including dynamic behaviour of the aircraft or environmental conditions. An extended definition is known as the Safe Flight Envelope (SFE), which is defined as ”the part of the state space for which safe operation of the aircraft and safety of its cargo can be guaranteed and externally posed constraints will not be violated” [36]. Manoeuvring within the limits of the safe flight envelope is one of the key criteria to ensure safe flight. The technique used to remain within these boundaries is Safe Flight Envelope Protection (SFEP).



(a) Traditional flight envelope

(b) SFE as intersection of forward and backward reachable set

Figure 3.2: Two possible ways to visualise the flight envelope [29]

3.2.1. Safe Flight Envelope Protection

To ensure safety in-flight, it is necessary to manoeuvre within the limits of the safe flight envelope. Safe flight envelope protection is the convenient name for ensuring that an (unmanned) aerial vehicle operates within these boundaries. This task can be divided into two parts. First, the SFE must be determined. Second, it must be guaranteed that the vehicle operates within the limits of the SFE. The second part has been researched extensively for aircraft and can be incorporated in different ways. A first approach is to directly incorporate a protection mechanism in the control design, such that the pilot cannot manoeuvre the aircraft outside its safe flight conditions. A second approach gives a timely warning to the pilot that the aircraft is in the proximity of the boundaries of the flight envelope [36]. Similar approaches can be applied to UAVs.

The challenge of SFEP is the first part: identifying the safe flight envelope, especially determining the limits of the dynamic behaviour of an aircraft or UAV. This is because the envelope is not only dependent on the dynamics of the aircraft, but also on operating conditions it is flying in. Each combination of both has slightly different boundaries, which should be known exactly for successful SFEP. In the past, significant research has been executed to determine this dynamic flight envelope of aircraft [36]. It started with regular wind tunnel testing and CFD analysis, from which the results were validated by expensive flight tests. Due to increasing computing power, analytical methods using aircraft models gained more interest. Another important tool is bifurcation analysis, which allows for computing of an entire set of steady state solutions. More recently, database-driven approach towards SFE determination have been explored. Zhang et al. [41] proposed a closed loop envelope protection system, where aircraft damage identification is used for online flight envelope retrieval from a large database. Lastly, a promising technique for flight envelope determination of UAVs is based upon reachability analysis.

The reachable set for an aircraft or UAV are all states that can be reached within a certain time window

given an initial condition. A forward and backward reachable set can be determined, where the safe flight envelope is regarded as the intersection between both [34]. This is visualised in Figure 3.2b. Reachable sets can be determined by solving Equation 3.1 over a predefined time window. A larger time window implies a larger reachable set.

$$\dot{\boldsymbol{x}}(t) = f(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \quad \boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{u} \in U \tag{3.1}$$

The forward reachable set is determined by solving Equation 3.1 forward in time. The result is a set of all states $\boldsymbol{x} \in \mathbb{R}^n$ for which there exists an input $\boldsymbol{u} \in U$ such that $\boldsymbol{x}$ can be reached from an initial condition $\boldsymbol{x}(0)$. On the other hand, the backward reachable set is calculated through solving Equation 3.1 backwards. It spans the set $\boldsymbol{x} \in \mathbb{R}^n$ for which an input $\boldsymbol{u} \in U$ exists such that the initial condition $\boldsymbol{x}(0)$ can be reached from these states. The intersection of both defines the safe flight envelope. Within this envelope trajectories from an initial condition to a state and back exist for all states.

The reachable set theory has been used in the past for determination of the SFE for different aircraft types. Van Oort [36] used this theory to identify the longitudinal SFE for an F-16 aircraft for different aircraft configurations and flight conditions. In 2015, Lombaerts et al. [25] proposed an online SFE determination approach for impaired aircraft using the robust reachable set, which does take uncertainties in the aircraft model into account. More recently, this theory has been applied to quadcopter SFE determination. Sun and de Visser [34] used Monte-Carlo trajectory simulations to determine the forward and backward reachable set of a quadcopter during high-speed flight.

Determining the SFE of a UAV in real-time is not a straightforward process. It demands continuous high computing power, which is often not available on UAVs. Beside this, current flight envelope estimation techniques depend heavily on the time window used for calculations, which raises questions about the usability of the envelopes. Therefore, other possibilities towards boundary exceeding prevention should be explored. One option is to predict when loss-of-control (LOC) will occur, such that timely actions can be taken to prevent this from happening. LOC is one of the possible consequences of safe flight envelope violation, meaning that when an upcoming LOC is identified, the UAV is flying close to the envelope boundaries. LOC prediction and prevention is therefore a technique that can supplement safe flight envelope protection in case the latter fails.

3.3. Loss-of-Control

Exceeding the boundaries of the safe flight envelope can have disastrous consequences, such as loss-of-control. LOC is responsible for the majority of (fatal) accidents in aviation. In December 2020, Boeing published the annual Statistical Summary of Commercial Jet Airplane Accidents in which accidents in commercial aviation from 1959 until 2019 were analysed [7]. Between 2010 and 2019, 22% of all fatal accidents resulted from loss-of-control, which makes it the largest occurrence category for fatal accidents.

Not only commercial aircraft are subject to loss-of-control; UAVs can also suffer from LOC. In a 2017 study, Belcastro et al. [6] analysed 100 reported small UAV (sUAV) (lighter than 25 kg) mishaps between 2010 and 2015, where 34 were caused by loss-of-control, making it the largest shareholder in sUAV mishap causes. These numbers confirm the need for extensive research in loss-of-control events in (unmanned) aerial vehicles, by looking into its primary causal and contributing factors, as well as prevention, mitigation and recovery methods. This section first provides an overview of the current status of loss-of-control research. To allow for an unbiased experiment for quadcopter loss-of-control prediction, a quantifiable LOC definition should be identified. This section therefore ends with possible definitions and identification of the most suitable one.

3.3.1. LOC research

Loss-of-control gained a lot of interest at the beginning of this millennium. In 2000, the Commercial Aviation Safety Team (CAST) founded the Joint Safety Analysis Team (JSAT). Their aim was to study data on loss-of-control events and recommend interventions that could lead to a reduction in fatal accident rate of 80% by 2007 [32]. JSAT defined LOC as a ”significant, unintended departure of the aircraft from controlled flight, the operational flight envelope, or usual flight attitudes, including ground events”,

where significant refers to events resulting in an incident or accident [32]. This definition has two main drawbacks.

A first drawback is that there are no quantifiable guidelines, which makes it complicated to consistently identify events as loss-of-control. To provide these guidelines, Boeing Company and NASA Langley Research Center cooperated to develop measurable criteria, known as Quantitative Loss-of-Control Criteria (QLC) [38]. To do so, they described LOC "as motion that is:

- outside the normal operating flight envelopes
- not predictably altered by pilot control inputs
- characterized by nonlinear effects, such as kinematic/inertial coupling, disproportionately large responses to small state variable changes, or oscillatory/divergent behavior
- likely to result in high angular rates and displacements
- characterized by the inability to maintain heading, altitude, and wings-level flight"

Flight dynamic parameters, such as angle of attack (α), sideslip angle (β), bank (ϕ) and pitch angle (θ) and pitch (q) and roll rate (p), that can be related to this LOC definition were identified and translated to five flight envelopes: Adverse Aerodynamic envelope, Unusual Attitude envelope, Structural Integrity envelope, Dynamic Pitch Control envelope and Dynamic Roll Control envelope. When comparing these envelopes to flight test data, it can be concluded that "the excursion of three envelopes is a clear indication of LOC" [38]. In 2014, Chongvisal et al. [12] used this definition for aircraft LOC prediction and prevention. For LOC prediction, the aircraft state is constantly monitored with respect to these five flight envelopes. For LOC prevention, information from this monitoring will be translated to a control law limiting the pilot commands to maintain states that are within the boundaries of the envelope.

A second drawback of the loss-of-control definition stated by JSAT is that according to this definition, LOC events will result in incidents or accidents. However, as noticed by Bromfield and Landry [8], this definition excludes events where airplane control was recovered. During such events, the aircraft or UAV is in an upset condition. An upset condition is defined as "any uncommanded or inadvertent event with an abnormal aircraft attitude, rate of change of aircraft attitude, acceleration, airspeed, or flight trajectory", where abnormal should be defined relative to the flight phase and type of aircraft [23]. Bromfield and Landry identified several recovery factors based upon non-fatal LOC events that should all be satisfied to recover from loss-of-control resulting form an upset condition. UAVs can also recover from upset conditions, as was proven by Baert [1], who created a recovery algorithm for a quadcopter with single rotor failure (SRF).

Other major contributions to LOC research were done by the National Aeronautics and Space Administration (NASA). Since 1999, NASA has conducting detailed research to address both aircraft and UAV loss-of-control and by creating and testing technologies for LOC prevention, detection, mitigation and recovery [4]. In 2010, Belcastro and Jacobson [3] proposed a holistic approach for aircraft LOC prevention called 'Aircraft Integrated Resilient Safety Assurance & Failsafe Enhancement (AIRSAFE)', which includes advanced modelling and simulation of vehicle dynamics in off-nominal conditions, continuous vehicle health management, flight safety assurance technologies, crew interface management and verification and validation techniques for evaluation of all these technologies. Furthermore, Belcastro et al. [5] identified the key characteristics of LOC, primary causes and causal and contributing factors, based upon analysis of aircraft LOC accident analysis. This resulted in a list of potential future LOC risks and follow-up activities for LOC prevention and recovery.

A research group from the University of Michigan, directed by Prof. Ella Atkins, investigated LOC prevention methods according to the AIRSAFE approach. Atkins proposed an Envelope-Aware Flight Management System (EA-FMS), which combines flight safety assessment and management (FSAM), envelope estimation, adaptive flight planning, system identification and adaptive control in one system to enable prevention and recovery of LOC events [13]. The system is able to warn the pilot and override commands just in time to recover from a LOC scenario. The system however depends on offline simulation of nominal and anomaly cases and therefore on an accurate model of the desired aircraft.

Most research mentioned previously has focused on LOC prevention. Few literary references discuss active, on-board LOC prediction methods. As mentioned earlier, Chongvisal [12] proposed a prediction method that monitors the state of the aircraft with respect to the five flight envelopes. More recently, van der Pluijm [29] proposed to use a data-driven approach towards LOC prediction. In this context, LOC was identified by a critical state in the measurement data, which could be visualised in the measurement data by a sudden change of almost all variables. Van der Pluijm proposed Critical Slowing Down (CSD) to identify Early Warning Signals (EWS) of a critical transition in complex systems. The method was tested on a quadcopter subject to single rotor failure. However, as noted by the author, CSD requires slow and monotonic change to observe EWS properly. Quadcopters and other aerial vehicles are usually subject to sudden changes close to the flight envelope boundaries, which leads to questions on the suitability of CSD for LOC prediction.

3.3.2. Quadcopter LOC Definition

Previous LOC research status review shows that an unambiguous, quantifiable LOC definition is not straightforward to determine. This is however important for predicting loss-of-control for a UAV. An unambiguous definition of the point in time where LOC occurs allows for consistent labelling among all failure runs. Based upon analysis in this and previous section, several possible definitions can be deduced:

1. Loss-of-control is quantified by the position of the state vector relative to the safe envelope, which is the intersection of the forward and backward reachable sets. If this vector is outside this envelope, the system is in LOC condition for the time horizon used to compute these sets. The moment in time where the boundaries of the SFE are exceeded, LOC begins. Although this definition is the only true theoretical definition, it has some practical issues. The reachability analysis is sensitive to operating conditions and the time horizon necessary to compute the sets is case sensitive. In addition, computing the SFE requires a high fidelity model which is currently not trivial for all aircraft. Therefore other, more practical definitions must be used.

2. Loss-of-control is quantified by monitoring its position relative to the five envelopes as described by the QLC, as proposed by Chongvisal et al. [12] in 2014. However, as noted by Sun and de Visser [34], this approach is not suitable for a quadcopter due to its different behaviour compared to a fixed-wing aircraft. As a quadcopter will be the test case for this research, this must be considered. Apart from that, similar issues arise as mentioned for previous definition. Flight envelopes are case depended, making this definition unsuitable.

3. Loss-of-control is quantified by detection of an upset condition, to ensure events where control was regained are included as well. The beginning of the upset condition can be used as LOC indication. When looking at the definition of upset condition proposed by Lambregts et al. [23] stated earlier, it can be concluded that upset conditions are context dependent. Therefore, the downside of using this LOC quantification is that a second definition ambiguity arises, as this definition is not clearly quantifiable as well. Therefore, this definition is not suitable either.

4. Loss-of-control is quantified by a critical state, depending on what type of LOC is assessed, as proposed by van der Pluijm [29]. For example, for pitch rate limits, this is denoted as $q_{critical}$, which is the pitch rate where it is expected that LOC occurs. This moment in time is identified in the sensor data by a sharp change of almost all variables. LOC is initiated by the state prior to this critical state. It is however not known precisely when LOC occurs, meaning that a possible range should be defined (e.g. between $q_{LOC,1} = q_{critical} - 1$ and $q_{LOC,2} = q_{critical} - 0.5$). The downside is that this is less accurate than some of the previous definitions, however it is not flight condition dependent as opposed to previous definitions. Only the sensor data should be observed, making it the most accessible definition. Therefore, this will be the definition used for the remainder of this research.

3.4. Conclusion

Extensive research has been conducted for loss-of-control prevention, detection, mitigation and recovery. Only little research focuses on LOC prediction. This is however an important tool in aircraft and UAV safety enhancement. From data related to aircraft and UAV incidents and accidents it can be

concluded that loss-of-control is still the cause for the majority of these events, despite all effort put in LOC prevention. An on-board, real-time LOC prediction system that can send timely warnings to the pilot or overrule pilot commands to remain within the SFE boundaries will result in less LOC events. Not only from a safety perspective is LOC prediction useful, also when looking from a sustainability point of view LOC prediction can be utilised. When a reliable LOC prediction system can be installed on-board and deployed in real-time, aircraft and UAV can fly closer to the boundaries of the envelope. This reduces safety margins that must be applied during design, making the system lighter, thus more sustainable. Due to its potential consequences, it is worth investigating new techniques suitable for LOC prediction.

4

Data-Driven Prediction Techniques

Although loss-of-control prediction of UAVs is becoming more urgent due to the increasing availability of these vehicles for a larger audience, it has not been widely researched yet. Due to this lack of information on LOC prediction, different areas in which prediction using time-series generated by multiple sensors is required should be explored. To this end, this chapter first discusses the field of prognostics and health management. After this, an introduction to data preprocessing techniques is provided. This chapter ends with a section on deep learning methods and a conclusion providing the most promising LOC prediction techniques.

4.1. Prognostics and Health Management

One of these fields is Prognostics and Health Management (PHM), which is occupied with monitoring the health condition of complex systems by utilising large amounts of data coming from sensors [30]. A prognostic framework detects and classifies anomalies or faults, and analyses the degradation of the system with this anomaly present [2]. However, the main concern of PHM is predicting the Remaining Useful Life (RUL). The goal of RUL prediction is to estimate the time left before failure of a system component or complete system. RUL prediction is a regression problem, which is the same type of problem as loss-of-control prediction, meaning that similar methods can be used for both. PHM is thus an interesting framework to utilise for enhancing quadcopter safety, as both anomaly detection and LOC prediction are relevant.

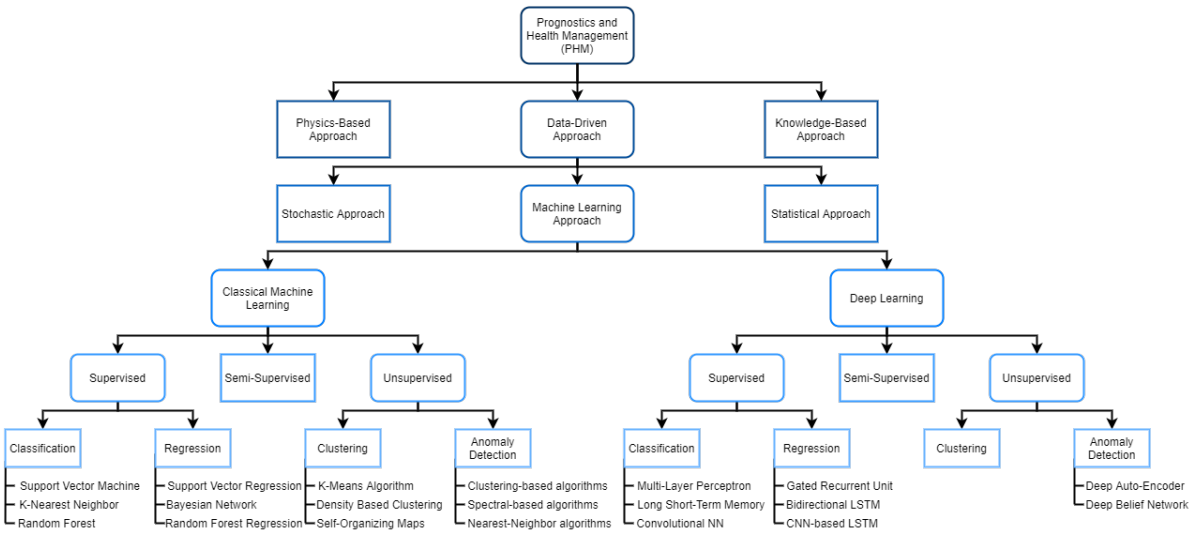


Figure 4.1: Categorical breakdown of PHM methods [30] [40] [43]

Prognostic methods can be categorised into three domains: model-based, knowledge-based and data-driven approaches [2]. The model-based approach requires a mathematical model describing the dynamics of a system. For simple systems, this approach usually achieves high prediction accuracy. However, for more complex and highly dynamic systems like aircraft and quadcopters detailed models that describe the complete motion perfectly are not available. This makes a model-based approach unsuitable for LOC prediction. A knowledge-based approach uses the experiences from past failures to estimate when new failures will occur by comparing the current condition with previous failure events. This is again an approach that can achieve high accuracy, however it is limited to failure events experienced before, meaning that it is not a robust method. Data-driven approaches utilise data coming from sensors within the system and automatically learn features from this, which determine the current condition of the system. The advantage compared to the other two approaches is that it is not necessary to understand the exact dynamics of the system and knowledge from previous runs does not have to be translated into a prediction algorithm manually. Data-driven approaches can be divided into stochastic algorithms, statistical algorithms and machine learning methods [2]. The first two approaches use traditional methods like Bayesian-based approaches, Hidden Markov Models and regression. This research will focus on the last type of data-driven approaches: machine learning methods.

Originally, machine learning (ML) was defined in 1959 by Arthur Samuel as ”the field of study that gives computers the ability to learn without being explicitly programmed”. The key point is its ability to learn patterns from data automatically, without explicit guidelines on how to do this. Machine learning techniques can be categorised in several ways based on their characteristics. A first approach to do this is based on classical machine learning and deep learning (DL). Classical machine learning techniques are already used for decades and include techniques such as Support Vector Machines (SVMs), nearest neighbour, decision trees, naive Bayes classifier and k-means clustering. It is closely related to the stochastic and statistical data-driven approaches. Deep learning on the other hand uses Artificial Neural Networks (ANN) consisting of more than one hidden layer. This method has gained increased popularity since the introduction of backpropagation for neural network training by David Rumelhart, Geoffrey Hinton and Ronald Williams in 1986 [31]. Since then, the performance of these approaches improved massively, and nowadays deep learning methods achieve superior performances compared to classical machine learning methods for PHM applications [10] [43].

Another way to categorise machine learning techniques is supervised or unsupervised approaches. Supervised learning algorithms learn to represent a function that maps inputs to outputs based on example input-output pairs (x, y) [14]. The expected output related to an input is called a label, which can be e.g. a specific fault type, health indicator or RUL estimation. Supervised methods are used for classification and regression problems. Unsupervised methods, on the other hand, do not have access to labels. An unsupervised model therefore learns features based upon similarity. It can group similar data points, which is clustering, or it can compare input data to nominal, known data to identify off-nominal conditions, which is anomaly detection.

Lastly, machine learning techniques can be distinguished between generative and discriminative models. Generative models learn the joint probability distribution, whereas discriminative models learn the conditional probability density function, which can be interpreted as the probability of output *y* given input *x*. An application of this is classification, where input *x* is identified as belonging to class *y*. Generative models attempt to generate new instances based upon the underlying data distribution [30]. In PHM applications, generative models learn to reconstruct normal system behaviour. This expected normal state is compared with the true state to identify faulty conditions, which is called anomaly detection. Discriminative models can have different purposes, such as directly identifying a condition as healthy or faulty (binary classification), identifying the type of fault (multiclass classification) or predicting the RUL (regression). A categorical breakdown of the PHM approaches including example techniques found in literature is visualised in Figure 4.1.

The complete data-driven PHM cycle including a classical and deep learning path is visualised in Figure 4.2. Two approaches can be taken in order to predict remaining useful life [27]. The first approach starts with manual feature extraction and selection, followed by detection and classification of an anomaly, after which a degradation model that captures the evolution of this anomaly is applied recursively

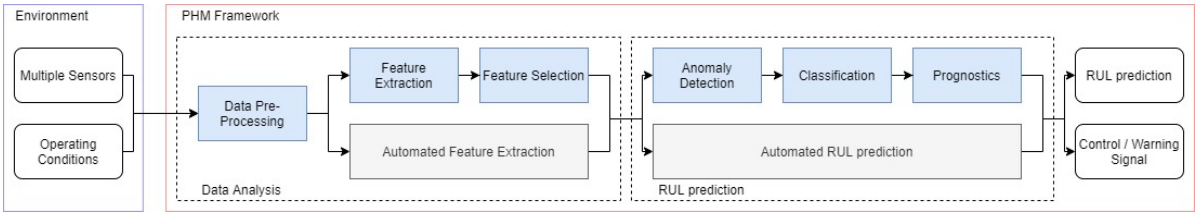


Figure 4.2: Modules of classical data-driven and deep learning Prognostics and Health Management cycle

to propagate this in time. Once it surpasses a predefined threshold, failure is said to occur. The time between measurements and failure is defined as the RUL. Both phases use classical machine learning methods. A second approach directly estimates RUL from the measurement data, where both feature extraction and RUL prediction are done automatically using deep learning algorithms.

The remainder of this chapter will evaluate the data-driven PHM cycle in more detail. First, data analysis will be discussed, which includes data preprocessing techniques to improve RUL prediction accuracy. Second, state-of-the-art deep learning prediction methods will be explored.

4.2. Data Analysis

Conventional data-driven health monitoring approaches, like classical machine learning, require extensive, manual data preprocessing. To ensure optimal performance of the prediction model, informative features should be extracted from the measurements. There are several ways to do this. Usually, only data coming from sensors showing a clear degradation trend are used. Next, features are manually extracted. Time domain features include statistical features such as the mean and variance. Frequency domain features can be extracted after Fourier analysis, where a time signal is converted into a frequency signal. Similar features that were extracted in the time-domain can be extracted in frequency domain. Combined time-frequency features are also used, which can be extracted after e.g. wavelet transform [43]. Lastly, as classical data-driven techniques cannot deal with large-scale data, data compression techniques are applied, such as Principle Component Analysis. The main idea behind this technique is to find correlations between data coming from several sensors and only continue with data showing the strongest trend. These preprocessing steps require expert knowledge as well as extensive human labour, which is undesirable.

Deep learning techniques, on the other hand, require less data preprocessing in comparison to classical machine learning methods. These techniques can deal with large-scale data and are able to extract features automatically, decreasing the amount of human work involved. However, preprocessing techniques can still boost performance of the automatic feature extraction. Often used preprocessing techniques are selection of specific sensors and time-domain feature extraction. Furthermore, data is normalised and a sliding window technique is used.

Sensor selection

Not all sensors show a clear positive or negative trend over time which point towards degradation or potential loss-of-control. To reduce dimensionality and with that decrease training and inference time, only those sensors that show a clear trend should be chosen. Sensors are selected through visual inspection.

Normalisation

Data coming from various sensors monitor system health while being subject to many operating conditions. The measurements taken from these sensors work on different scales. When feeding the data directly into a DL network, excessive weights and biases will appear, which slows down training or even results in failed training due to exploding gradients. To prevent this from happening, data should be normalised. Most commonly used normalisation methods are the z-score and min-max normalisation, given by Equation 4.1 and Equation 4.2 respectively [37].

$$x_i' = \frac{x_i - \mu_i}{\sigma_i} \tag{4.1}$$

$$x_i' = \frac{x_i - \min x_i}{\max x_i - \min x_i} \tag{4.2}$$

In these equations, x_i denotes the raw data from sensor *i*, μ_i denotes the mean of the raw sensor data of sensor *i* and σ_i denotes the standard deviation of sensor *i*. x_i' represents the normalised data.

Sliding Time Window

The sliding time window technique is commonly applied to time-series data to achieve data segmentation. The core idea behind this is to create more data samples, which has positive effects on RUL prediction performance. A visual explanation on how to apply this technique on measurement data is shown in Figure 4.3.

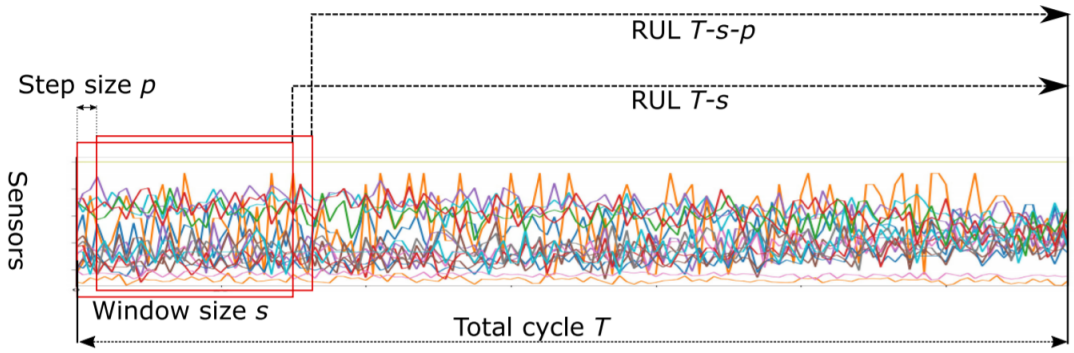


Figure 4.3: A window with size *s* slides over the measurement data to create small samples of data [10]

The total time from beginning of the experiment to failure (or loss-of-control) is T. A window with size set to s is sliding over the measurement data of all n sensors, with a step size of p, leading to I number of samples. Each sample will thus have a size of $s \times n$. The time to failure, here the RUL, is defined as the time from the last sample present in the chunk until failure, $T - s - p \cdot (i - 1)$, with i the sample number, $i \in I$. The window size s has a large influence on the performance of the model, and should therefore be tuned together with other hyperparameters.

4.3. Deep Learning Methods

Over the past decade, deep learning as a tool for health monitoring of complex systems has gained increasing interest. In a 2020 review on deep learning for PHM applications, Rezaeianjouybari and Shang analysed 137 unique studies that apply a deep learning approach for health monitoring published between 2013 and September 2019 [30], proving the increased popularity. One of the reasons for this is that a tremendous amount of deep learning models exist. This is because they have many characteristics, such as network type, amount of layers and nodes, activation functions of the nodes, objective or loss functions and optimisation algorithms [39], which can all take on many forms and shapes. However, some architectures, which is the combination of all these characteristics, are more suitable for a specific problem than others. Network type has the largest influence on this, as this defines the mathematical operations that are being executed on the data and in what order. Five base deep learning architectures can be identified: Restricted Boltzmann Machines (RBM), Auto-Encoders (AE), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Generative Adversarial Networks (GAN). An overview of their variants and characteristics is given in Table 4.1.

Table 4.1: Characteristics of the standard deep learning architectures and their most important variants [30] [40] [43]

Network Type	Variants	Goal	(Un)supervised
Auto-Encoder		Generative	Unsupervised
	Sparse Auto-Encoder		
	Denoising Auto-Encoder		
Restricted Boltzmann Machine		Generative	Unsupervised
	Deep Belief Network		
	Deep Boltzmann Machine		Supervised
Convolutional Neural Network		Discriminative	Supervised
Recurrent Neural Network		Discriminative	Supervised
	Long Short-Term Memory		
	Gated Recurrent Unit		
Generative Adversarial Network		Generative	Unsupervised

An auto-encoder network consists of an encoder part and a decoder part. The goal of this network is to construct new representations of the input dataset, meaning that it is a generative type of neural network. A standard auto-encoder consists of three layers, and the model is optimised such that the reconstruction error is minimal. Auto-encoders thus do not require labels, making it an unsupervised technique. A drawback of the standard auto-encoder is that it tends to learn the identity transformation, failing to create new representations. To prevent this, a sparsity constraint can be added to the cost function, leading to the Sparse Auto-Encoder (SPA). Another common AE variant is the Denoising Auto-Encoder (DAE). As opposed to the standard auto-encoder, this network corrupts the input data by adding noise and its purpose is to reconstruct the clean input data. The denoising auto-encoder is therefore robust for small perturbations in the input data. The hidden layer of the auto-encoder represents features of the input data. To learn more complex features, multiple auto-encoders can be stacked together, leading to a stacked auto-encoder. The layers are added and trained one-by-one, allowing for layer-wise training. As this network can learn features in an unsupervised fashion, the hidden layer's weights and biases are often used as pretrained parameters of deep neural networks, which ensures better convergence in comparison to random initialisation. The standard and stacked auto-encoder are drawn on the first row in Figure 4.4.

Restricted Boltzmann Machines are two-layer neural networks that form a bipartite graph. This is a graph consisting of two groups of nodes, visible and hidden nodes, which are fully interconnected, however nodes within a group are not connected. All connections are undirected, meaning that information can flow both directions. The purpose of the RBM is again signal reconstruction, which means that it is a generative model. An RBM is trained using the Contrastive Divergence (CD) algorithm and the purpose is to optimise the log likelihood. By stacking multiple RBMs, a Deep Belief Network (DBN) is created, where all layers are directed, except the top layer, which makes them closely related to a conventional artificial neural network. This network parameters can be trained per layer using the CD algorithm, which can again be used as pretrained initialisation for Deep Neural Networks (DNN). A second RBM variant is the Deep Boltzmann Machine (DBM), where additional undirected layers are added to the standard RBM. Due to the undirected connections, the layers must be trained jointly in a supervised fashion, which is computationally more expensive. A standard RBM, DBM and DBN are shown on the second row in Figure 4.4.

Convolutional Neural Networks have two layer types alternating each other: convolutional layers and pooling layers. Convolutional layers apply the convolution operation to subsets of the input data. By sliding a so called filter or kernel over the complete input set (either 1D or 2D), local features are identified. The pooling layer is a dimensionality reduction layer, which picks the most significant features outputted by the convolutional layer. This layer applies non-linear functions like *max*. By stacking multiple convolutional and pooling layers behind each other, more abstract features can be extracted. The CNN is followed by some fully connected layers and a final layer responsible for regression or classi-

fication, making it a discriminative technique. CNN are typically used for image processing, like object detection problems. Figure 4.4 shows a CNN with one convolutional and pooling layer on the third row.

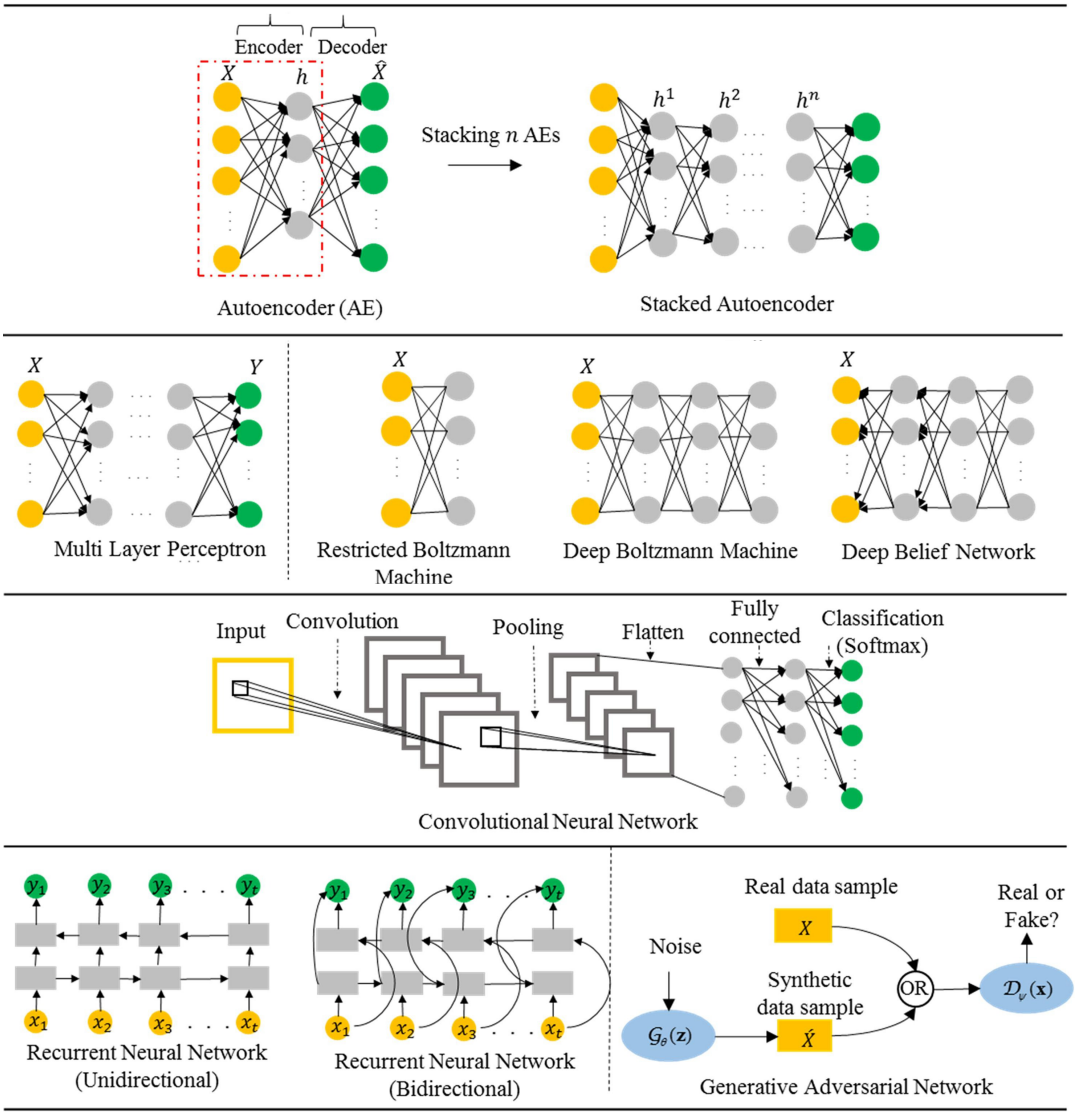


Figure 4.4: Base deep learning architectures for PHM purposes [30]

Opposed to conventional deep neural networks, Recurrent Neural Networks contain recurrent nodes, which allow for a memory. Recurrent nodes have a feedback loop, meaning that at time *t*, this node receives the input of the network together with its own output of time *t - 1*. This makes RNN architectures most suitable for analysing sequential data, like language processing. An RNN can also be bidirectional, where data processed in both a forward and backward manner. When long-term dependencies need to be captured, the network may suffer from the vanishing or exploding gradient problem. To overcome this problem, recurrent nodes are replaced by Long-Short Term Memory (LSTM) cells or Gated Recurrent Unit (GRU) cells. Both cells have a similar working principle. Gates are introduced that allow for long term features to be maintained for longer period of time. RNN are discriminative models. A standard RNN and bidirectional RNN are shown on the left on the lowest row in Figure 4.4. The square blocks can be conventional recurrent cells, as well as LSTM or GRU cells.

Finally, Generative Adversarial Networks are a relative new type of network, introduced only in 2014 by Goodfellow [18]. This is a generative model consisting of two networks: a generator and a discriminator. The generator creates fake input data based on the original dataset. The discriminator receives

both real and fake data and attempts to identify the real data. Although the sample generation is very realistic, these type of models suffer from mode collapse, which means that the reconstructed data show limited variety, meaning that the model can not generalise well. More research is required before it can be widely applied in PHM applications. The working principle of the GAN is shown on the right on the lowest row of Figure 4.4.

Advantages and disadvantages of previously mentioned network types are summarised in Table 4.2. The problem of predicting an upcoming loss-of-control is essentially a regression problem utilising time-series data. Based upon this and previous discussion of network types, it can be concluded that a recurrent structure is the most promising approach. This is the most suited network for time-series analysis. However, the basic recurrent neural network suffers from vanishing or exploding gradient issues. This means that the weights of nodes go to zero or plus or minus infinity respectively, resulting in bad performance. LSTM and GRU cells solve this problem. Therefore, the working principles of these variants should be investigated in more detail.

Table 4.2: Advantages and disadvantages for different deep learning architectures

Network Type	Advantages	Disadvantages
Auto-Encoder	- Unsupervised - No faulty data necessary - Layer-by-layer pretraining can serve as initialization for DNN	- Only anomaly detection
Restricted Boltzmann Machine	- Unsupervised (except DBM) - No faulty data necessary - Layer-by-layer pretraining can serve as initialization for DNN	- Only anomaly detection
Convolutional Neural Network	- Can learn more complex features than AE and RBM - Most suitable for image processing	- Supervised - (Manual) hyperparameter fine-tuning required for optimal result
Recurrent Neural Network	- Can learn more complex features than AE and RBM - Most suitable for sequential and time-series data	- Supervised - (Manual) hyperparameter fine-tuning required for optimal result
Generative Adversarial Network	- Unsupervised - Realistic input reconstructions	- Only anomaly detection - Mode collapse

4.3.1. Long Short-Term Memory

Long Short-Term Memory networks are widely used within the field of prognostics and health management. They are used for both classification and regression, the latter being RUL prediction. Chen et al. [10] created an attention-based LSTM RUL prediction network, which is able to identify more relevant features and time steps. An additional feature fusion includes both automatic extracted features and handcrafted features, to boost performance. Next to this, LSTM networks come in various shapes and combinations. They are stacked or used in parallel with regular DNNs or CNNs for additional feature extraction. In 2019, Li et al. [24] proposed parallel analysis of raw sensor data by an LSTM and CNN network, after which a second LSTM network merges features extracted by both, leading to a RUL estimation. Bidirectional LSTM (BiLSTM) structures are used to account for future states as well. In

a paper written by Huang et al. [21], two BiLSTM architectures are stacked, where the first one is responsible for feature extraction on normalised raw sensor data and the second one for higher-level feature extraction of these learned features as well as feature extraction from normalised operational conditions. A final part with fully connected layers results in a RUL estimation.

Performance of LSTM architectures compared to classical machine learning methods, auto-encoders and restricted Boltzmann machines is similar or higher even without hyperparameter fine-tuning, as proven by [43]. In an experiment for tool wear prediction, BiLSTM outperformed classical ML methods, a standard auto-encoder, a DBN, an LSTM and a CNN network. Only a denoising auto-encoder performed better. However, considering the complexity of an (Bi)LSTM, hyperparameter tuning will boost performance of these methods, whereas this is not the case for AE or RBM variants. This confirms that LSTM architectures are most suitable for time-series analysis.

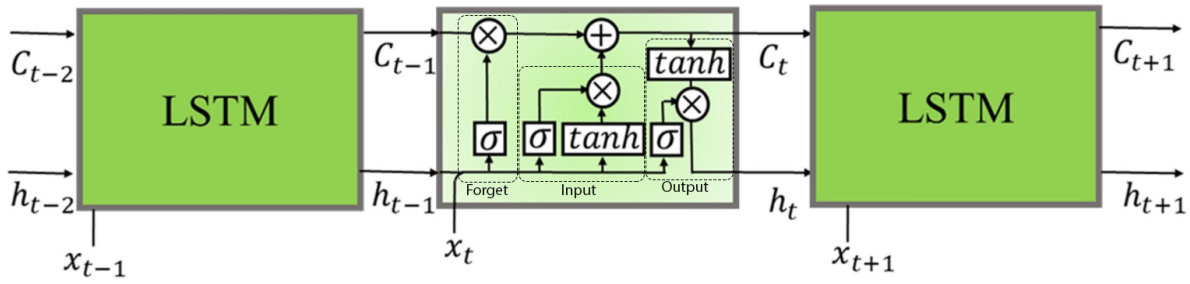


Figure 4.5: Sketch of internal flow of an LSTM cell [30]

Although various LSTM cell architectures exist, one of the most popular types was proposed by Gers, Schmidhuber and Cummins in 2000 [17]. Three gates, input gate, forget gate and output gate, regulate the flow into and out of the cell. This is shown in Figure 4.5. At each time step *t*, the hidden state $\mathbf{h}_t$ is updated using the input data at this time step *t*, $\mathbf{x}_t$, the hidden state of the previous time step *t-1*, $\mathbf{h}_{t-1}$, and network parameters Θ_{LSTM}. The updating equations are as follows:

$$\begin{aligned}\overrightarrow{\mathbf{h}}_t &= f\left(\overrightarrow{\mathbf{x}}_t, \overrightarrow{\mathbf{h}}_{t-1}; \rightarrow\Theta_{\text{LSTM}}\right) \\ &= \begin{cases} \overrightarrow{\mathbf{z}}_t = \tanh\left(\overrightarrow{\mathbf{W}}_z \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_z \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_z\right) \\ \overrightarrow{\mathbf{i}}_t = \sigma\left(\overrightarrow{\mathbf{W}}_i \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_i \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_i\right) \\ \overrightarrow{\mathbf{f}}_t = \sigma\left(\overrightarrow{\mathbf{W}}_f \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_f \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_f\right) \\ \overrightarrow{\mathbf{c}}_t = \overrightarrow{\mathbf{z}}_t \odot \overrightarrow{\mathbf{i}}_t + \overrightarrow{\mathbf{c}}_{t-1} \odot \overrightarrow{\mathbf{f}}_t \\ \overrightarrow{\mathbf{o}}_t = \sigma\left(\overrightarrow{\mathbf{W}}_o \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_o \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_o\right) \\ \overrightarrow{\mathbf{h}}_t = \tanh\left(\overrightarrow{\mathbf{c}}_t\right) \odot \overrightarrow{\mathbf{o}}_t \end{cases}\end{aligned} \tag{4.3}$$

$\overrightarrow{\mathbf{W}}_Z$, $\overrightarrow{\mathbf{W}}_i$, $\overrightarrow{\mathbf{W}}_f$ and $\overrightarrow{\mathbf{W}}_o$ are input weights and therefore related to $\mathbf{x}_t$. $\overrightarrow{\mathbf{R}}_Z$, $\overrightarrow{\mathbf{R}}_i$, $\overrightarrow{\mathbf{R}}_f$ and $\overrightarrow{\mathbf{R}}_o$ are recurrent weights and therefore related to $\mathbf{h}_{t-1}$. Furthermore, $\overrightarrow{\mathbf{b}}_Z$, $\overrightarrow{\mathbf{b}}_i$, $\overrightarrow{\mathbf{b}}_f$ and $\overrightarrow{\mathbf{b}}_o$ are biases. σ is the logistic sigmoid activation function and tanh is the hyperbolic tangent activation function. Lastly, $\odot$ denotes element wise multiplication. The weights and biases are learned during model training.

In case of a Bidirectional LSTM network, an additional layer is added where data is passed through from the newest state $\mathbf{x}_t$ to the oldest state $\mathbf{x}_1$, as opposed to a standard LSTM, where data is passed through from oldest state $\mathbf{x}_1$ to the newest state $\mathbf{x}_t$. This is visualised on the lowest row on the left in Figure 4.4. This backward pass has its own set of equations, similar to the forward pass equations:

$$
\begin{aligned}
\overleftarrow{\mathbf{h}}_t =& f\left(\overleftarrow{\mathbf{x}}_t, \overleftarrow{\mathbf{h}}_{t+1}; \overleftarrow{\Theta}_{\text{LSTM}}\right) \\
=& \begin{cases}
\overleftarrow{\mathbf{z}}_t = \tanh\left(\overleftarrow{\mathbf{W}}_z \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_z \overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{b}}_z\right) \\
\overleftarrow{\mathbf{i}}_t = \sigma\left(\overleftarrow{\mathbf{W}}_i \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_i \overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{b}}_i\right) \\
\overleftarrow{\mathbf{f}}_t = \sigma\left(\overleftarrow{\mathbf{W}}_f \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_f \overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{b}}_f\right) \\
\overleftarrow{\mathbf{c}}_t = \overleftarrow{\mathbf{z}}_t \odot \overleftarrow{\mathbf{i}}_t + \overleftarrow{\mathbf{c}}_{t-1} \odot \overleftarrow{\mathbf{f}}_t \\
\overleftarrow{\mathbf{o}}_t = \sigma\left(\overleftarrow{\mathbf{W}}_o \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_o \overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{b}}_o\right) \\
\overleftarrow{\mathbf{h}}_t = \tanh\left(\overleftarrow{\mathbf{c}}_t\right) \odot \overleftarrow{\mathbf{o}}_t
\end{cases}
\end{aligned}
\tag{4.4}
$$

When looking at Equation 4.3 and Equation 4.4 it can be observed that they are almost identical, except for the $\rightarrow$ and $\leftarrow$, which represent the forward pass and backward pass respectively. The complete output at time step t of a BiLSTM network is the element-wise summation of the outputs of both processes, which is represented by Equation 4.5.

$$
\mathbf{h}_t = \overrightarrow{\mathbf{h}}_t \oplus \overleftarrow{\mathbf{h}}_t \tag{4.5}
$$

4.3.2. Gated Recurrent Unit

The Gated Recurrent Unit is a relatively new kind of RNN type. It was introduced in 2014 by Cho et al. [11], whereas LSTM was already introduced in 2000. Due to this, less PHM applications using GRU architectures exist. In 2018, Zhao et al. [44] proposed a local feature-based GRU network, where first features are extracted from data manually, after which these features are fed into a bidirectional GRU network. A final fully connected regression part is added to obtain a RUL prediction. As for the LSTM methods, GRU can also be combined with other deep learning structures. Lai et al. [22] placed a CNN and GRU network in series for to detect both short-term local features and long-term pattern for time-series trends.

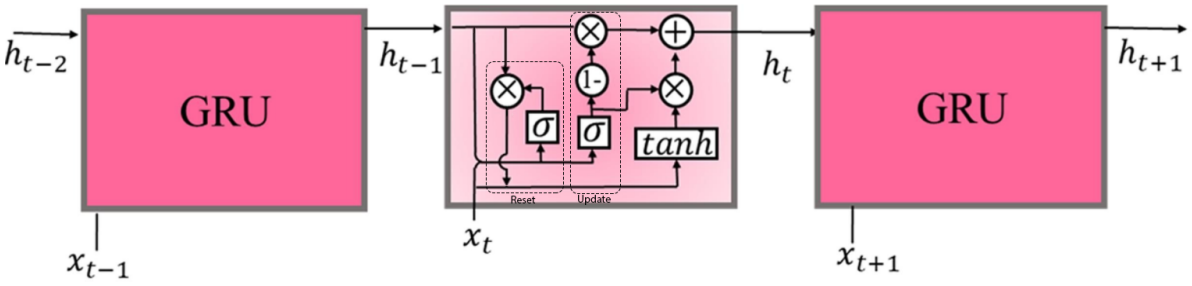


Figure 4.6: Sketch of internal flow of a GRU cell [30]

As opposed to the LSTM cell, the GRU cell only contains two gates, making training computationally less demanding and therefore faster. The input and output gate are combined into one update gate and the forget gate is renamed to reset gate. The flow through a GRU cell is shown in Figure 4.6. The flow through the GRU cell can be defined in a similar fashion as the flow through an LSTM cell. At time step *t*, the hidden state $\mathbf{h}_t$ is updated using the input data at time step *t*, $\mathbf{x}_t$, the hidden state at the previous time step *t-1*, $\mathbf{h}_{t-1}$ and the network parameters, Θ_{GRU}. The GRU can again be used in a bidirectional manner. The updating equation for the forward process is given by Equation 4.6, and the updating equation for the backward process is given by Equation 4.7. The output of the bidirectional GRU is the same as for the bidirectional LSTM, given by Equation 4.5.

$$\begin{aligned}\overrightarrow{\mathbf{h}}_t =& f\left(\overrightarrow{\mathbf{x}}_t, \overrightarrow{\mathbf{h}}_{t-1}; \overrightarrow{\Theta}_{\text{GRU}}\right)\\ =& \begin{cases} \overrightarrow{\mathbf{z}}_t = \sigma\left(\overrightarrow{\mathbf{W}}_z \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_z \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_z\right) \\ \overrightarrow{\mathbf{r}}_t = \sigma\left(\overrightarrow{\mathbf{W}}_r \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_r \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}_r\right) \\ \overrightarrow{\tilde{\mathbf{h}}}_t = \tanh\left(\overrightarrow{\mathbf{W}}_h \overrightarrow{\mathbf{x}}_t + \overrightarrow{\mathbf{R}}_h \left(\overrightarrow{\mathbf{r}}_t \odot \overrightarrow{\mathbf{h}}_{t-1}\right) + \overrightarrow{\mathbf{b}}_h\right) \\ \overrightarrow{\mathbf{h}}_t = \left(1 - \overrightarrow{\mathbf{z}}_t\right) \odot \overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{z}}_t \odot \overrightarrow{\tilde{\mathbf{h}}}_t \end{cases}\end{aligned} \tag{4.6}$$

$$\begin{aligned}\overleftarrow{\mathbf{h}}_t =& f\left(\overleftarrow{\mathbf{x}}_t, \overleftarrow{\mathbf{h}}_{t-1}; \overleftarrow{\Theta}_{\text{GRU}}\right)\\ =& \begin{cases} \overleftarrow{\mathbf{z}}_t = \sigma\left(\overleftarrow{\mathbf{W}}_z \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_z \overleftarrow{\mathbf{h}}_{t-1} + \overleftarrow{\mathbf{b}}_z\right) \\ \overleftarrow{\mathbf{r}}_t = \sigma\left(\overleftarrow{\mathbf{W}}_r \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_r \overleftarrow{\mathbf{h}}_{t-1} + \overleftarrow{\mathbf{b}}_r\right) \\ \overleftarrow{\tilde{\mathbf{h}}}_t = \tanh\left(\overleftarrow{\mathbf{W}}_h \overleftarrow{\mathbf{x}}_t + \overleftarrow{\mathbf{R}}_h \left(\overleftarrow{\mathbf{r}}_t \odot \overleftarrow{\mathbf{h}}_{t-1}\right) + \overleftarrow{\mathbf{b}}_h\right) \\ \overleftarrow{\mathbf{h}}_t = \left(1 - \overleftarrow{\mathbf{z}}_t\right) \odot \overleftarrow{\mathbf{h}}_{t-1} + \overleftarrow{\mathbf{z}}_t \odot \overleftarrow{\tilde{\mathbf{h}}}_t \end{cases}\end{aligned} \tag{4.7}$$

4.4. Conclusion

Deep learning methods perform well in RUL prediction applications. Due to similarities between a RUL prediction problem and LOC prediction problem it is expected that similar results can be achieved for the latter using the same techniques. Important similarities are that both issues aim to predict when a complex system operating in dynamic conditions fails and both problems have only limited and noisy sensor data available to train a data-driven model. The model that is most accurate in predicting RUL is a recurrent neural network. More specifically, long short-term memory and gated recurrent unit methods perform best, due to their capability of identifying long-term dependencies in time-series. The LSTM network has more gates to process information, which makes this model more accurate than GRU, especially if longer sequences of data must be analysed. However, the GRU cell can be optimised easier, due to which training time is reduced.

In addition to this, variants of these methods, such as bidirectional LSTM or GRU and combinations with other deep learning architectures have proven to be even more accurate than standard architectures. These architectures perform more data processing, due to which more low-level failure features can be identified. If more failure features can be assessed, prediction accuracy improves. Therefore, the following four architectures have been identified as most promising to investigate their capabilities in LOC prediction:

- LSTM
- CNN-LSTM
- BiLSTM
- GRU

It is expected that the more complex architectures, CNN-LSTM and BiLSTM, are more accurate as they can extract more complex features. However, more fine-tuning of hyperparameters is required before this is achieved. The standard GRU and LSTM have a similar lay-out, but the GRU is less complex due to the use of two gates instead of three. Therefore, the training time for this model will be less, however accuracy can be lower than for the LSTM structure.

5 Methodology

This chapter provides an overview of the project methodology. The research procedure is elaborated upon, where the different phases of the project and the desired aim of each phase are described in detail. Furthermore, the experimental set-up is described, where details about the simulation environment and flight tests that will be conducted are provided. Lastly, possible risks that could appear during the research or the experiment are identified and mitigated.

5.1. Research Procedure

The research consists of several phases, where the main goal of each phase is to answer parts of the research question. The different phases and their key aspects are visualised in Figure 5.1.

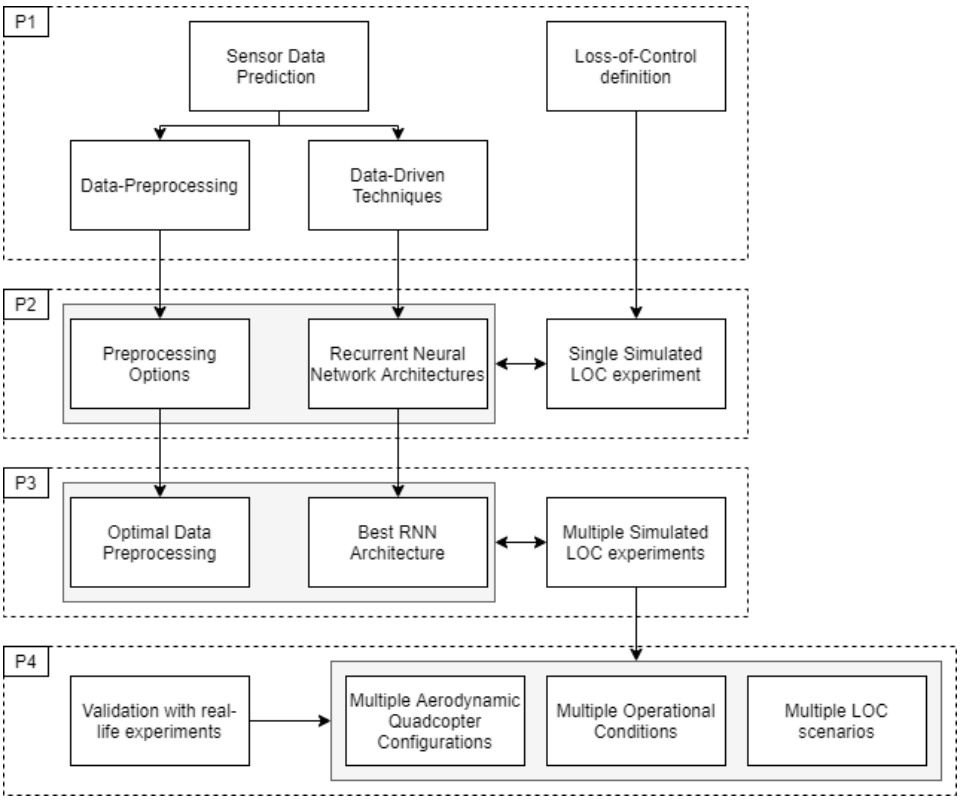


Figure 5.1: Proposed research plan

Four phases can be distinguished, which all have the aim to answer parts of the research questions. The four phases are as follows:

- P1 (Literature Review): Literature study is conducted after which current state of loss-of-control prediction is defined. Data-driven methods that can be used for prediction on sensor data are assessed and compared to identify the most promising approaches. The aim is to answer the first and second research subquestion and part of the third research subquestion. As shown in Figure 5.1, the desired output of this phase are several promising deep learning architectures that can be tested in the second phase.
- P2 (Model Selection): Simulation of loss-of-control for a quadcopter is done using an existing quadcopter simulator. Run-to-failure data is generated for a nominal quadcopter configuration in single operating condition settings and one LOC scenario. Several RNN architectures are trained to predict LOC using the generated data. This data is preprocessed using different techniques to determine the optimal preprocessing procedure, which leads to the best LOC prediction accuracy. Furthermore, the most accurate architecture is identified. During this phase, the remainder of research subquestion three will be answered.
- P3 (Simulation test case): Using the simulator, run-to-failure data is generated for various quadcopter configurations, operational conditions and LOC scenarios. Still using the model trained on the baseline experiment as used in phase two, the accuracy of the model's predictions is assessed, to determine how well the model can generalise LOC prediction. The aim of this research phase is to answer research subquestions four, five and six.
- P4 (Quadcopter test case): Real-life experiments using the Parrot Bebop 2 quadcopter are conducted to validate the results emerging from the simulation test cases in phase three. This phase will contribute to answering research subquestions four, five and six and will be a key phase for answering the main research question.

5.2. Tools

The main purpose of the thesis research is to investigate if deep learning methods can be used for prediction of loss-of-control. This research is only meant as a proof-of-concept, meaning that LOC prediction does not have to take place on-board, within a closed control loop. This provides the opportunity to use several different specialised tools. For model creation TensorFlow will be used, whereas run-to-failure simulations of the quadcopter will be conducted in a dedicated Bebop simulator.

5.2.1. TensorFlow

TensorFlow is one of most widely used free, open source platforms for machine learning applications. It offers many built-in functions and libraries which can be used to easily create, train and deploy machine learning solutions. For this, it uses Keras, which is an intuitive machine learning API (Application Programming Interface), written in Python. Python is often used in the scientific community due to its huge variety of (mathematical) toolboxes, libraries and extensions, making coding in Python straightforward and easy without requiring an extensive background in software engineering. These advantages of Python can be utilised when using TensorFlow.

TensorFlow and Keras include many functions and properties that can be used to build neural networks layer per layer, initialise weights and biases in different ways, apply various cost functions and optimisation algorithms and tune different hyperparameters. Apart from this, various built-in functions exist to preprocesses data and to assess the accuracy of a network. Due to these advantages, TensorFlow will be used to create the models used for LOC prediction.

5.2.2. Bebop Simulator

The behaviour of the Parrot Bebop 2 will be simulated in a Bebop simulator created by the Control & Simulation department of the Aerospace Engineering Faculty of the Delft University of Technology. The simulator is built in Simulink, which is a MATLAB based programming environment to graphically create and test models of dynamical systems. The simulator replicates the behaviour of the quadcopter for inputs working on the system and operating conditions in which it flies, and simulates the sensor outputs. The sensor outputs will be saved and used for training of a neural network in TensorFlow.

The advantages of using a quadcopter simulator compared to a real-life quadcopter is that a large amount of runs can be done, including failure runs, which is necessary as a neural network requires many failure runs in the training set to optimally learn failure features. Above this, quadcopter parameters, operating conditions and initial conditions can be set exactly as desired. This results in many perfect data sets which is ideal for network training. The downside is that this does not resemble real-life perfectly, as real experiments do not allow for complete control over all parameters and conditions and show flaws in the measurement data.

5.3. Experimental Set-Up

To validate the results from phase three, real-life experiments will be conducted. It is required to describe these in detail and state assumptions that are made during execution, such that they can be used for this purpose.

Assumptions

Most important for predicting LOC is a quantifiable definition of the moment in time where the quadcopter enters a LOC condition which enables consistent labelling among all failure runs. As there is no one clear definition, an assumption for this should be made. From literature review it became clear that there are several possibilities, described in Chapter 3. To determine the instant in time where LOC occurs, the method as proposed by van der Pluijm [29] will be used. He proposed to identify a critical state where it is expected that LOC is present. The exact moment in time where LOC occurs is the moment prior to this critical state. This is not as exact as other proposed methods, but it is independent on flight conditions and computationally less demanding.

A second assumption that must be made is when the LOC prediction is sufficient, as stated in some of the research subquestions. This will depend on the performance achieved by the model on the baseline experiment. To assess performance, different metrics are used. For prediction problems, often used metrics are the root mean square error and the scoring function.

- **Root Mean Square Error**: one of the most widely used metric is the Root Mean Square Error (RMSE), which can be calculated using Equation 5.1.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2} \tag{5.1}$$

  Here, $\hat{Y}_i$ is the estimated output value and Y_i is the true output value, where the output value for the given problem is the time to LOC. n is the total number of data samples.

- **Scoring Function**: often used in combination with the RMSE. The scoring function penalises late predictions and can be calculated according to Equation 5.2.

$$S = \begin{cases} \sum_{i=1}^{n}\left(e^{-\frac{h_i}{13}} - 1\right), & h_i < 0 \\ \sum_{i=1}^{n}\left(e^{\frac{h_i}{10}} - 1\right), & h_i \geq 0 \end{cases} \tag{5.2}$$

  In this equation, $h_i = \hat{Y}_i - Y_i$, where $\hat{Y}_i$, Y_i and n have the same definition as for the RMSE. The difference between RMSE and the scoring function is shown in Figure 5.2.

Before flight tests with different configurations, operating conditions and LOC scenarios will be conducted, an exact definition of sufficient will be determined.

Logistics

The experiments will be executed in the Cyberzoo of the Control & Simulation department of the faculty of Aerospace Engineering of the Delft University of Technology. In this arena, safe UAV flight tests can be conducted as it is surrounded by a net. However, due to limited space, the Cyberzoo has some limitations. The flight tests are designed such that they can be executed in this arena given these limitations, which influences e.g. the LOC scenario. A fan can be used to introduce wind. The Parrot Bebop

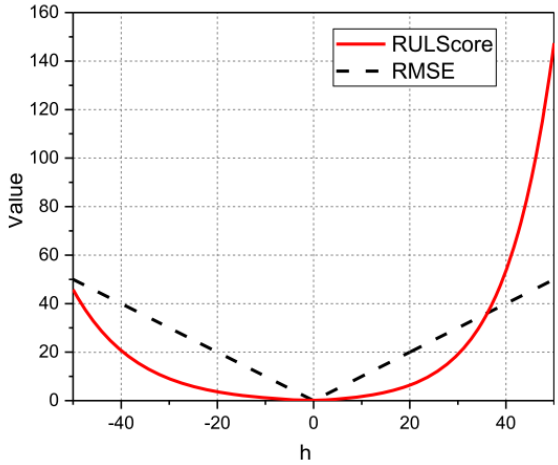


Figure 5.2: Differences between RMSE and Scoring Function [37]

2 quadcopter will be used during the flight tests. An image of the quadcopter, including the definition of the body frame and numbering of the rotors, can be found in Figure 5.3.

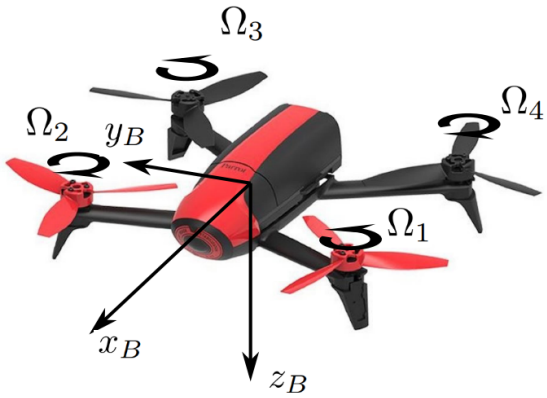


Figure 5.3: Definition of the body frame of the tested quadcopter, the Parrot Bebop 2 [34]

The manoeuvre that the drone must perform is a flip in pitch, while pitch rate is increased over time. Eventually, actuators will saturate, which results in loss-of-control. The challenge during this manoeuvre is the loss of height during the flip. Although this problem cannot be overcome, varying thrust during the manoeuvre will slow down the falling process. When the drone is oriented upwards, thrust is increased, whereas during a downward orientation, thrust is decreased. In order to perform this complicated manoeuvre consistently, the drone will be controlled automatically by an autopilot programmed in the open-source autopilot software PX4, which is written in C++.

Experiments

In the fourth phase, the results emerging from the simulations should be validated using a real-life test case. Different experiments will be conducted, which will all attempt to validate different results of the simulation. To this end, several types of experiments can be distinguished:

1. **Baseline experiment**: several runs will be executed to generate data that will be used to train the neural network. The quadcopter is in basic configuration (no additional weights, CG location as intended) and no wind is present. Pitch rate limit is violated to enter LOC condition.

2. **Verification experiment**: several runs will be conducted to verify the working of the trained model. The same quadcopter configuration will be used, as well as the same operating conditions. Again, the pitch rate limit is violated

3. **Sensitivity experiments**: different runs will be executed, with slight changes in:

    - Quadcopter weight and CG location
    - Wind speed
    - LOC scenario

    The goal is to identify how well the model can generalise for different configurations, operational conditions and LOC types.

During testing, it is key to create a data set that is as clean as possible, as this is required by the neural network to be able to learn failure features from data. A clean data set shows limited outliers and no gaps. The quadcopter should be fully loaded, such that it can deliver optimal performance, which stimulates obtaining clean data. Furthermore, during experiments data does not only come from on-board measurements, but also from a drone tracking system present in the Cyberzoo: OptiTrack. A set of cameras captures the motion of the UAV using reflection markers on the vehicle. During extreme motion, tracking can get lost, which poses additional challenges in obtaining a clean data set.

5.4. Planning

The first phase of this thesis project, the literature study including research methodologies, last approximately 10 weeks, ending on the 25th of May. After this, phase two starts. The second, third and fourth phase of the project last for approximately 30 weeks, excluding holidays. This means that the green light meeting will take place halfway through December, and graduation will be at the end of January. Taking holidays into account, a preliminary planning is drafted and shown in Figure 5.4.

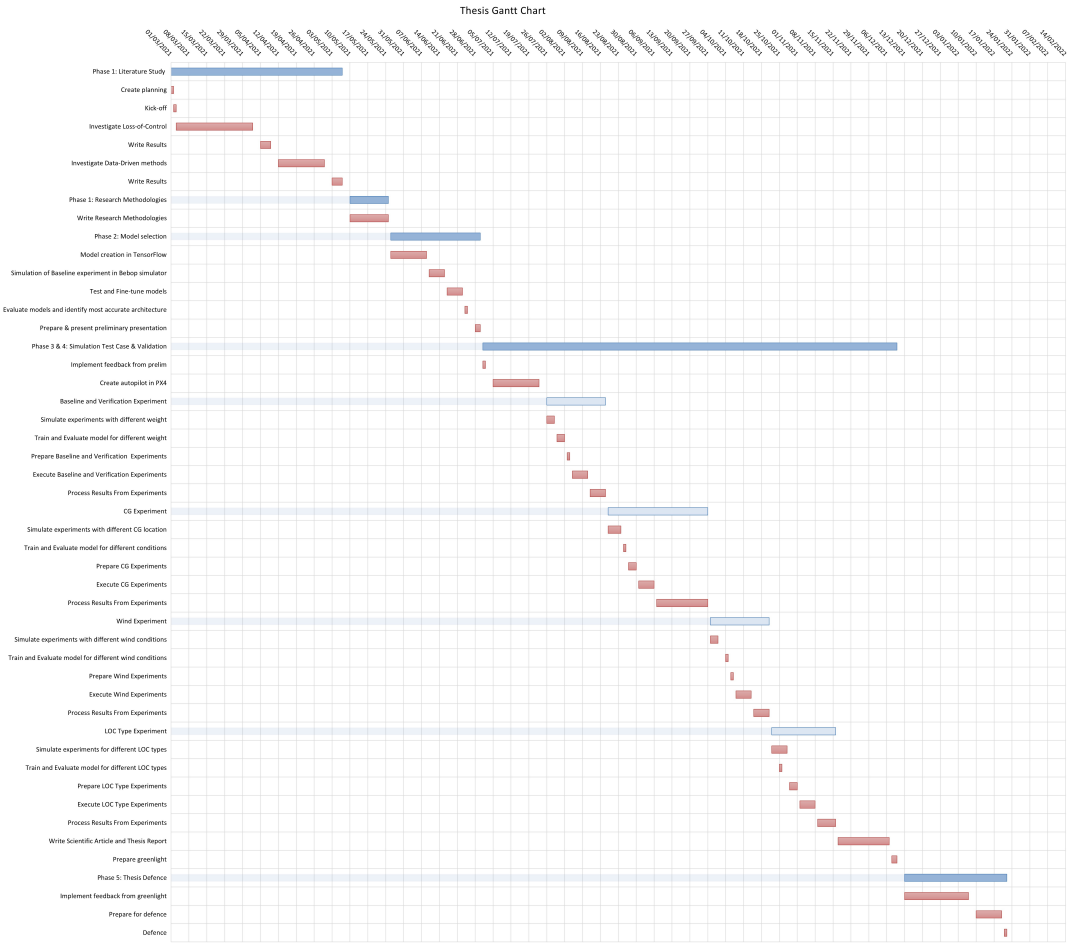


Figure 5.4: Proposed planning for the research project

It should be noted that phase three and four are performed simultaneously. During this period of time, four subphases can be identified that have a similar structure. During each subphase, one experiment is the central point of attention. The layout of each subphase is as follows:

- Simulate experiments in Bebop simulator (4 days)
- Train and evaluate model with this data (2 days)
- Prepare experiments (2 days)
- Execute experiments (5 days)
- Process results from experiments (5 days)

5.5. Risk Identification

During each research, risks are present which can have a bad influence on the results. When risks are identified beforehand, a mitigation approach can be determined to reduce either the probability or the impact. Risks that are identified specifically relating this research and a mitigation plan are the following:

- **Risk**: The planning is too ambitious.
  Mitigation: The research subquestions are defined in a modular fashion. The main goal of this research is to investigate if LOC can be predicted using a data-driven approach. Identifying whether such a model can generalise LOC prediction is a valuable addition, but not the main purpose of this research. When enough time is available, this part of the research question will be assessed, otherwise phase three and four will only focus on the baseline and validation experiment.

- **Risk**: The experiments cannot be executed as planned. Possible redesign on the experiments is necessary.
  Mitigation: In the planning, phase three and four are combined and divided into four subphases, which all contain one week of experimenting. Spreading out the experiments over several, non-consecutive weeks allows for technical issues during the experiments, as well as possible re-design.

- **Risk**: Data obtained from experiments is not clean enough or not enough failure runs can be executed for the neural network to learn failure features.
  Mitigation: Simulations will be done using an existing Bebop 2 simulator, which resembles reality as closely as possible. The amount of runs in a simulator is not limited to availability and less prone to risks. Based upon the results from this, already several research subquestions can be answered. Furthermore, weights of the neural network used in experiments can be initialised using weights from the neural network trained on simulation data. This form of knowledge transfer ensures faster convergence with less data required.

- **Risk**: The drone is not able to perform the desired manoeuvre.
  Mitigation: The Cyberzoo has a wide variety of drones. If the Parrot Bebop 2 is not able to perform the desired manoeuvre, other drones can be chosen. From a safety perspective it is even advised to start with smaller drones, to see if it shows the desired behaviour at the edge of the flight envelope.

6

Conclusion Preliminary

The popularity of Unmanned Aerial Vehicles (UAVs) has grown extensively over the past decade. Due to decreasing prices and extreme manoeuvrability, these vehicles are used in many civil and military applications, as well as for recreational purposes. This increase raises safety concerns, which should be tackled by on-board safety enhancement systems. One solution that will stimulate safe flight is a loss-of-control prediction algorithm.

The goal of this thesis research is to develop a data-driven algorithm that can predict loss-of-control for a quadcopter operating in different conditions, different configurations and for several LOC scenarios. The aim of this report is to answer the first and second research subquestion and the first part of the third research subquestion.

The first question aims at defining loss-of-control for a quadcopter operating nominally. Only one theoretically correct definition exists, however this depends on real-time calculation of the safe flight envelope, which is not possible given the current knowledge. Therefore, for the remainder of this thesis project, loss-of-control is defined by a critical state where it is expected that LOC is present. The moment in time where LOC occurs is the moment prior to this state, which can be observed in sensor data by a sudden change in almost all parameters.

The second question concentrates on identifying data-driven technique which can predict LOC and determine which techniques would be most suitable. As the LOC prediction problem is a regression problem, where large amount of time-series data coming from different sensors will be utilized, recurrent neural network (RNN) is the most suitable model. This is because RNN architectures contain memory cells allowing for identification of time dependent features.

The third research subquestion aims at identifying which NN architecture achieves the highest accuracy for LOC prediction. This can only be answered after testing, however first an overview of the state-of-the-art should be established, as described by question 2a. Literature showed that there are two promising types of RNN networks: long short-term memory (LSTM) and gated recurrent unit (GRU). Apart from these two main variants, more advanced architectures exist, such as a bidirectional LSTM network, where data is processed in both a forward and backward manner. Combinations with other neural network types are also possible, such as an LSTM with a convolutional neural network (CNN). From literature study, it can be concluded that the most promising RNN architectures are:

- LSTM
- CNN-LSTM
- BiLSTM
- GRU

In the next phase of the thesis research, the accuracy of these models will be tested using data from a quadcopter simulator. Once the most accurate model is identified, the remaining research questions can be assessed using the simulator and real-life quadcopter flight tests.

Part II

Thesis Paper

Loss-of-Control Prediction of a Quadcopter using Recurrent Neural Networks

A.V.N. Altena
Graduate Student, Control and Simulation Section
Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands

C.C. de Visser
Assistant Professor, Control and Simulation Section
Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands

***Abstract* - Loss-of-control (LOC) is the main cause of crashes for drones. On-board prevention systems should be designed that require low computing power and memory. Data-driven techniques serve as a solution. This study proposes the use of recurrent neural networks (RNN) for LOC prediction. The aim is to identify which RNN model is most suitable and if this model can predict loss-of-control for changing aerodynamic characteristics, wind conditions, quadcopter types and LOC events. Real-life flight tests using a Tiny Whoop quadcopter were performed where LOC was initiated by demanding a too high yaw rate of ±2000 deg/s. Using data from these failure runs, four RNN networks were trained: long short-term memory (LSTM), bidirectional LSTM (BiLSTM), LSTM preceded by a convolutional neural network (CNN-LSTM) and gated recurrent unit (GRU).**

Only on-board sensor measurements are necessary for LOC prediction. The commanded rotor values provide clearest early warning signals for loss-of-control, as these values show saturation before LOC. Beside this, for optimal performance, an additional parameter should be used as well, which is either the gyroscopic measurement or acceleration. All networks could predict LOC correctly and equally well, leading to no preference for one specific model type. Next to this, when compensating for expected deviations in the prediction, the models can still be used for change in mass and different propellers. To draw conclusions for flying in wind conditions and using different quadcopters, more research is necessary. Finally, the models can only be used to predict loss-of-control due to a too high yaw rate and the time to loss-of-control should be similar to those from the training set.

***Index Terms* - Loss-of-control, quadcopter, recurrent neural network**

I. Introduction

UNMANNED aerial vehicles (UAVs), better known as drones, are used for a variety of applications, such as aerial observation, package delivery, military applications and entertainment. The increasing use raises safety concerns. The extreme agility of UAVs can initiate unsafe situations, leading to damage of the vehicle, or even a loss-of-control (LOC) event, resulting in a crash if left unaddressed. A 2017 study showed that out of 100 reported UAV mishaps, 34 could be categorised as loss-of-control, making it the largest occurrence category of UAV mishaps [1].

Loss-of-control is not only dangerous for UAVs. Between 2011 and 2020, 20.5% of all fatal accidents in commercial aviation resulted from a loss-of-control event [2]. This emphasises the importance of more research on new LOC prevention techniques.

One of the leading studies on loss-of-control prevention was part of NASA's Aviation Safety Program. Belcastro et al. [3] proposed a holistic LOC prevention approach, which points out the value of on-board integrated systems that help avoiding, detecting, mitigating and recovering loss-of-control. A direct result from this proposal is the Envelope-Aware Flight Management System (EA-FMS) designed by Di Donato et al. [4]. The authors present integrated flight envelope estimation, flight planning and flight safety assessment and management to diagnose high-risk events possibly leading to loss-of-control. To prevent this, the system overrides pilot commands just-in-time.

Currently, LOC prevention techniques rely on knowledge about the safe flight envelope (SFE). Especially for quadcopter UAVs it is not straightforward to

determine the exact shape of the SFE, as they manoeuvre differently from traditional fixed-wing aircraft. Sun and De Visser [5] applied a Monte Carlo simulation in combination with reachable set theory to determine the set of states in which a quadcopter can manoeuvre freely and return to safe conditions within a predefined period of time, leading to an accurate estimation of the SFE.

However, there still remain drawbacks about this approach. The flight envelope changes over time due to changing aerodynamic characteristics and operating conditions. Continuous, online recalculation of the SFE is necessary, which implies availability of high computing power. Quadcopters do not have this required power available [6]. Therefore new LOC prevention methods should be designed that do not require high computing power or information from external tracking systems or sensors, but can still deal with varying operating and aerodynamic characteristics. Data-driven techniques can fit these requirements.

Data from on-board sensors can give information about an upcoming LOC event. Van der Pluijm [7] proposed to use critical slowing down (CSD) to identify a tipping point to an alternate state of a quadcoptor suffering from single rotor failure. However, as noted by the author, CSD requires slow and monotonic transitions, whereas damaged quadcopters usually are subject to sudden changes close to tipping points, leading to questions about the suitability of CSD for LOC prediction. Other techniques should be tested that can find patterns from data automatically.

The objective of the presented study is to compare different data-driven techniques for loss-of-control prediction on quadcopter flight data and their ability to generalise LOC prediction for different operating and aerodynamic characteristics. In particular, various recurrent neural network architectures are analysed and compared. Furthermore, different on-board parameters will be used to identify the best early warning signals for LOC, giving more insight in the loss-of-control problem itself.

The remainder of this paper is structured as follows. First, an introduction to recurrent neural networks is provided in section II. Information about the flight tests and data processing is described in section III. Section IV continues with the results, after which they are discussed in section V. The conclusion is presented in section VI.

II. Recurrent Neural Networks

Neural networks have the ability to detect patterns from large amounts of data automatically. This is a favourable characteristic for the loss-of-control prediction problem, as it is unknown what exact combination of aerodynamic characteristics of the quadcopter, operating conditions and vehicle behaviour lead to a LOC event.

Recurrent neural networks (RNN) are in particular suitable for analysis of time-series data and data obtained from sensors fall within this category. In a classical RNN layer, nodes are replaced by memory cells which receive at time t not only the input data, but also its own output produced at time $t-1$ [8]. Due to these cells, RNNs can identify temporal dependencies in sequential data.

When long-term dependencies must be identified, traditional RNNs suffer from vanishing or exploding gradient issues during the training process. To solve this issue, other types of memory cells can be used, where the long short-term memory (LSTM) [9] and gated recurrent unit (GRU) [10] are most common. The idea behind these variants is that the flow of data is handled differently within the memory cell by introducing gates. These ensure that long-term features are maintained for a longer series of steps instead of being overwritten each time-step. GRU has less gates compared to the LSTM network, making it less complex. This means that this network type is computationally more efficient during training while hardly losing performance. On the other hand, for large data sets, LSTM might be better in capturing the longer temporal features [11].

Besides replacing traditional memory cells with more advanced cells, there are two more options to improve the performance of a recurrent neural network. The first option is using a bidirectional network, where a data sequence is analysed both forward and backward, meaning that it extracts features from both past and future states [12]. A final option is to precede the RNN with other types of neural networks, such as a convolutional neural network (CNN) [13]. This additional network extracts high-level features from data, after which the RNN extracts more complex features. However, these advanced RNN structures require more fine-tuning to achieve better performance than its less complex variants. Training also takes longer for these networks.

Table 1. Tiny Whoop characteristics

Characteristic	Details	Characteristic	Details
Mass including batteries	56g	Motor	TC0803 15000kV
Axis to axis diameter	75mm	Batteries	2x BETAFPV 4.35V 300mAh 1S
4-blade Propeller diameter	40mm	Flight Controller	JHEMCU SH50 F4 2S, 8.0MB black box
3-blade Propeller diameter	35mm	FC Software	Betaflight 4.2

Several studies have already proven the power of algorithms containing RNN structures applied on UAV sensor data. Sadhu et al. [14] proposed a complex CNN-BiLSTM network to detect anomalous behaviour of a quadcopter, after which a traditional fully connected neural network classifies the cause or fault leading to this behaviour. Another application of LSTM networks was investigated by Wang et al. [15]. The authors used a long short-term memory network to create an estimation of gyroscope sensor data and used this to detect gyroscope drift and bias. The residuals between the true and estimated data are smoothed to mitigate effects of noise. Once drifted or biased sensor data is detected, the estimated sensor data is used to replace this.

For the studied loss-of-control prediction problem, little information is available about the data obtained from flight tests. It is unknown how obvious the failure features are or how long the temporal dependencies are. Therefore, four network architectures will be tested to identify the complexity of the problem and determine which network deals best with this. These networks are an LSTM network, bidirectional LSTM (BiLSTM) network, LSTM network preceded by a CNN (CNN-LSTM) and a GRU network.

III. Methodology

By analysing sensor data obtained during various flight tests, similar behaviour is observed preceding the start of a loss-of-control event. However, the exact moment where the LOC event starts varies per run. This indicates that there are complex contributions to loss-of-control that are not observable visually. Therefore, recurrent neural networks are used to identify failure features and predict the time to loss-of-control.

Figure 1. The black box equipped 75mm, 56g Tiny Whoop used during experiments

A. Flight tests

Data used for this research is obtained during flight tests using an Eachine Trashcan Tiny Whoop 75mm, 56g quadcopter equipped with a JHEMCU SH50 flight controller with 8.0MB black box to store flight logs. The logs contain sensor data, rotor commands and true rotor outputs calculated using bidirectional DShot600. The logging rate is 1 kHz. The maximum power output is limited to 50% of the nominal output, to ensure that a loss-of-control event can occur. Figure 1 shows an image of the Tiny Whoop and Table 1 tabulates its characteristics.

During the experiments, the quadcopter is deliberately crashed by demanding a high yaw rate of ±2000 deg/s. While at first the vehicle can rotate around its z-axis as desired, quickly it starts oscillating where the amplitude of the roll and pitch angle show sinusoidal behaviour. Once these oscillations become too much, which is the moment where either the pitch or roll angle becomes larger than 90°, the quadcopter makes a turn around one of the other axes. The crux for correct loss-of-control prediction is to label the

Table 2. URUAV UZ85 characteristics

Characteristic	Details	Characteristic	Details
Mass including batteries	73g	Motor	1102 10000kV
Axis to axis diameter	85mm	Batteries	2x BETAFPV 4.35V 300mAh 1S
Propeller diameter	52.17mm	Flight Controller	JHEMCU SH50 F4 2S, 8.0MB black box
		FC Software	Betaflight 4.2

moment in time where LOC starts consistently. From both visual inspection and inspection of the failure data, this moment is identified as the moment where the quadcopter cannot continue rotating purely around its z-axis and roll or pitch takes over. Therefore, the starting point of loss-of-control is defined as the moment where the absolute value of either pitch or roll angle is equal to 90° and only becomes larger after this starting point.

Apart from identifying the best performing recurrent neural network architecture for the LOC prediction problem, it is also opportune to investigate their generalisation capabilities. For this purpose, different flight tests are performed:

Test 1: The majority of the executed flights is part of the first test, where sufficient data must be obtained to train neural networks to predict LOC. The test is executed in a windless environment and the quadcopter is in nominal conditions, meaning that no weight is added and four blade propellers are used. 75 Runs leading to a loss-of-control event are performed during this test.

Test 2: The second test consists of flights where the quadcopter mass is altered. Five flights are executed with different masses compared to the nominal quadcopter. The total mass of the quadcopter per run is summarised in Table 3. The LOC scenario remains unchanged.

Table 3. Total quadcopter mass per run

	Nominal	Run 1	Run 2	Run 3	Run 4	Run 5
Mass	56g	62g	66g	72g	76g	82g

Test 3: Another way to change aerodynamic characteristics of the quadcopter is by replacing the propellers by other types. Six flights are performed using 35mm three blade propellers instead of 40mm four blade propellers.

Test 4: The next flights are executed to investigate if networks trained on LOC due to high yaw rate can generalise to other LOC events. 53 Flights are performed leading to a loss-of-control event by demanding a high pitch rate of ±2000 deg/s. The longest two runs are chosen to check the generalisation capabilities.

Test 5: To investigate generalisation performance for different operating conditions, five flights are performed during which the quadcopter flies in a wind stream. Loss-of-control is again achieved by demanding an excessive yaw rate.

Test 6: Lastly, to test to what extent the algorithm can be transferred from one vehicle to another, four flights are executed with a different quadcopter. During this test, a URUAV UZ85 is used. Details of this quadcopter are depicted in Table 2.

All networks are trained using the data obtained during test 1 and after training, data from other tests are fed into the networks to identify which network is best in generalisation for different conditions.

B. Neural Network Architecture

The main goal of the neural networks is to predict the amount of time in seconds left until the LOC event starts. However, to avoid false predictions during nominal flight, another branch is added to the network that detects dangerous manoeuvres. This leads to the architecture shown in Figure 2.

In this figure, the four different architectures that are tested in this research are visualised. For the LSTM, BiLSTM and GRU network, the first three layers are not used. Their first two layers consist of LSTM, BiLSTM and GRU cells respectively. For the CNN-LSTM network, the first three layers are used and the following two layers consist of LSTM cells. For all architectures, the first and second RNN layer contain 50 and 100 neurons respectively. All networks

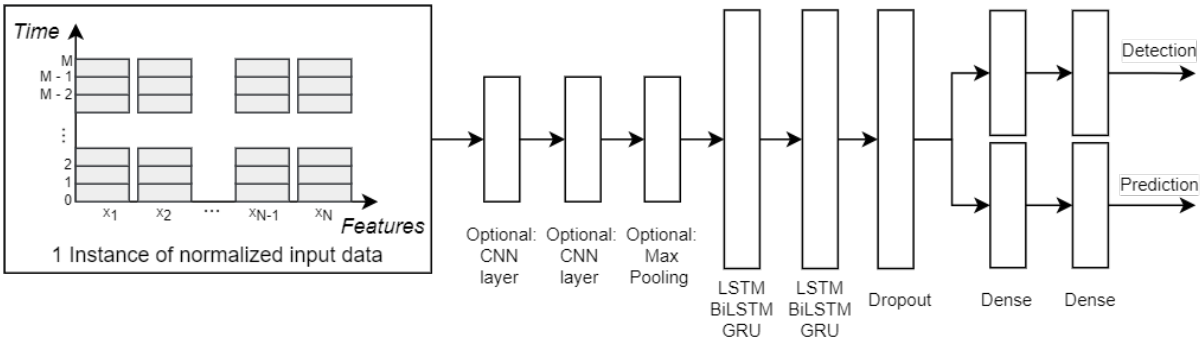
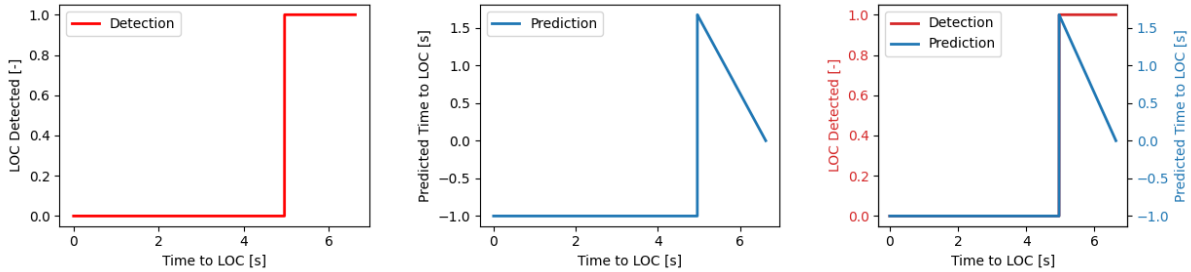


Figure 2. Architecture of the different recurrent neural networks used in this research



(a) Detection branch output **(b) Prediction branch output** **(c) Combined network output**

Figure 3. Output of the two network branches and combined output. The first 5 seconds are nominal flight, after which a dangerous manoeuvre leads to loss-of-control after 1.7s

are followed by a dropout layer with a dropout rate of 0.1, which is a method to prevent overfitting on the training set, allowing for better generalisation [16].

Finally, the networks are split into two branches, both containing two dense layers where the last layer outputs a floating-point value. For the detection branch, the label is either '0' or '1': '0' means that the quadcopter is in nominal flight and '1' means that a dangerous manoeuvre is flown which ends in a LOC event, shown in Figure 3a. For the prediction branch, the label is '-1' before the start of the dangerous manoeuvre. Once this manoeuvre has started, the label changes to the time in seconds left until the LOC event starts, shown in Figure 3b. The output of this branch only becomes relevant once the detection branch detects a dangerous manoeuvre. The combined output is visualised in Figure 3c, where the left y-axis belongs to the detection branch and the right y-axis belongs to the prediction branch.

The primary focus of this research is to identify whether RNNs can predict loss-of-control and if they can do this for various scenarios. Optimising the network architectures, such as the amount of layers and neurons, the activation functions and batch size, is outside the scope of this study.

To finalize, the performance of the trained networks is assessed using the root mean squared error (RMSE). This metric represents the average error between the model's predicted and true time to LOC. Thus, the lower the RMSE value, the better the network performance. The benefit of using this metric is that the unit is the same as the quantity that is being estimated, which is time in seconds for the LOC prediction problem. The RMSE value is calculated by

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2} \tag{1}$$

where $\hat{Y}_i$ represents the model's estimated time to LOC and Y_i the actual time to LOC. Furthermore, n is the total number of time steps, or data points, present in a failure run.

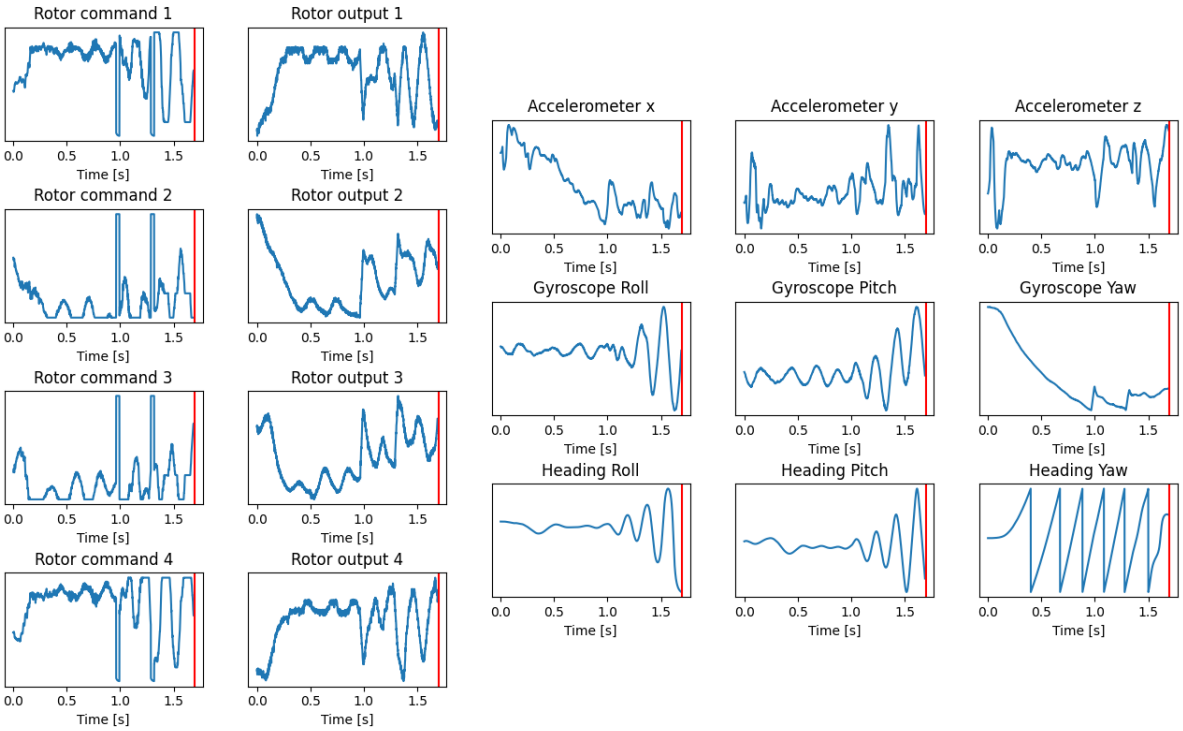


Figure 4. Parameters that show a trend before loss-of-control. The red line indicates the time where the LOC event starts

C. Data processing

Data obtained during the experiment must be processed before it is fed into the networks to improve prediction performance and to reduce training and inference time as much as possible. Inference means using a trained neural network on a new, unknown data set to get a prediction [16].

To prevent high training time, the data of each failure run is stripped. The moment where the dangerous manoeuvre starts is identified by observing the stick input of the pilot and the 5 seconds before this moment are preserved. All data points before this are deleted, as well as all data points after LOC has started. These 5 seconds are visible in Figure 3.

Another way to reduce training and inference time is selecting only sensor measurements that provide early warning signals for loss-of-control. To do so, all parameters that are logged during flight are plotted and those showing a trend before loss-of-control are considered, which are:

- Rotor commands for each of the four rotors: $\omega 1_{cmd}$, $\omega 2_{cmd}$, $\omega 3_{cmd}$ and $\omega 4_{cmd}$
- True rotor outputs for each of the four rotors: $\omega 1_{true}$, $\omega 2_{true}$, $\omega 3_{true}$ and $\omega 4_{true}$
- Smoothed accelerometer measurements: a_x, a_y and a_z
- Measurements from the gyroscope: roll rate p, pitch rate q and yaw rate r
- Heading in roll, pitch and yaw direction

Figure 4 provides plots of these variables from the start of the dangerous manoeuvre until the LOC event starts for one of the failure runs. The difference in behaviour between the rotor command and true rotor output is due to the electronic speed controller (ESC). This device adjusts the rotor commands, which are outputted by the controller using the PID gains, to the true revolutions per minute (RPM) of the rotors. To investigate which parameters show the clearest early warning signals for loss-of-control, a variety of combinations is used during the training phase to find the most optimal mix of parameters. In case of no clear preference for specific combinations, lower amount of parameters are preferred as this results in less processing and inference time.

Next to this, an issue that occurs when input data from different sensors are in different ranges is that neural networks can suffer from the exploding gradient problem during training [16]. To overcome this, the input data is standardised using the Z-Score, which is calculated according to:

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{2}$$

where μ represents the mean and σ the standard deviation. To ensure that the proposed algorithm can run in real-time, these values are determined using all failure runs obtained during test 1.

Apart from using advanced memory cells, there is another way to stimulate the network to identify time-dependent features. The data is cut into chunks of window size M with a step size of 1. This means that the first chunk contains all data points of all used parameters from time $t = 1$ until $t = M$, whereas the second chunk contains all data points from time $t = 2$ until $t = M + 1$. The associated time to loss-of-control is defined as the time from the last data point present in the chunk until LOC. The window size is set to 20.

Finally, as can be observed from the failure data and as is noted by Van der Pluijm [7], rotor saturation happens for most flights prior to loss-of-control. It is therefore hypothesised that including information about rotor saturation improves prediction performance. An additional feature is designed that counts the amount of saturated rotors at each moment in time, using the rotor commands. This is done as follows:

$$\begin{aligned} \text{NumSat} &= \sum_{i=1}^{4} z_i \\ z_i &= \begin{cases} 0 & 165 < \omega i_{cmd} < 1000 \\ 1 & \omega i_{cmd} \leq 165 \vee \omega i_{cmd} \geq 1000 \end{cases} \end{aligned} \tag{3}$$

where the mentioned values are based upon the output range of the rotor commands, which lies between 118 and 1048. A value of 118 means 0% throttle and a value of 1048 means 100% throttle. A margin is included in case the commands remain close to these values but are not exactly equal to them, leading to a saturated rotor in case the command is lower than 165 or higher than 1000.

Beside preprocessing data, results outputted by the networks are post-processed to improve performance. The detection branch outputs a value in the range $(-\infty, +\infty)$, where a negative value represents nominal flight and positive value represents dangerous flight. This output is mapped to '0' or '1'. By default, this value is mapped to '0'. Once the output of this branch is a positive value for 20 consecutive time steps, equalling 20ms, the output is mapped to '1'. This is switched back to '0' if the output of the branch is a negative value for 20 time steps in a row. These 20ms are chosen such that it equals the window size.

The output of the prediction branch is based upon the output of the detection branch according to:

$$Y_{Pred} = \begin{cases} -1 & Y_{Det} = 0 \\ \text{Prediction} & Y_{Det} = 1 \end{cases} \tag{4}$$

where Y_{Pred} equals '-1' in case nominal flight is detected and equals the output of the prediction branch otherwise.

IV. Results

All four network types are trained ten times using different combinations of parameters enumerated in subsection III.C, leading to 40 trained models in total. However, as random processes are involved in training, this is done three times, leading to 120 models. The performance of each model is assessed using one validation failure run that was not part of the training set. This run was performed with the same conditions as the runs from test 1 as described in subsection III.A. During this validation run, the loss-of-control event began 1.7s after the dangerous manoeuvre initiated.

The performance of the models are grouped in several ways to compare them based on different characteristics. Firstly, they are grouped by model type, to identify which type performs best. Secondly, they are grouped by parameter combination used for training to investigate which parameters show the best early warning signals for loss-of-control. Lastly, the results of the generalisation scenarios are presented to identify generalisation capabilities of different types and parameter combinations.

A. Model Performance

For network type comparison, boxplots are created per type, presented in Figure 5. The values used in

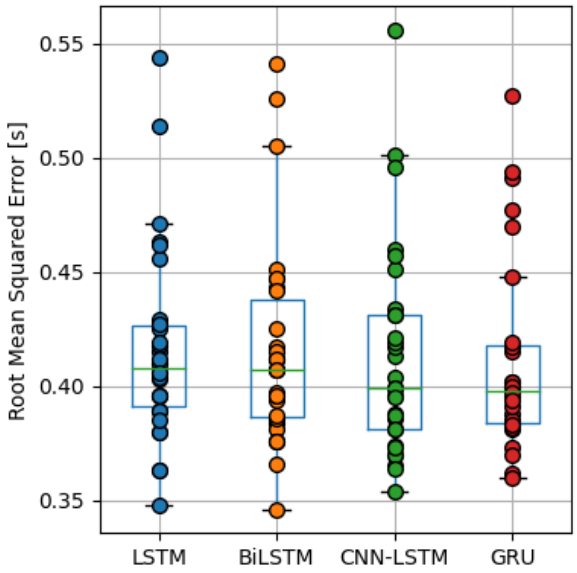


Figure 5. RMSE values in seconds for different model types

this plot are the root mean squared error values in seconds, achieved on the validation failure run. Each model type has 30 data points, due to ten parameter combinations for three trials.

Each model shows a few extreme outliers that are high. These are all related to one specific parameter combination, which will be addressed in the subsequent subsection. Beside this, the GRU model shows more extreme outliers with high values, but these have an RMSE value comparable to the weak outliers of the other models. All other values are within the boundaries of the box or within the weak outliers, which is 1.5 times the interquartile range, defined as the range from the lower boundary of the box to the upper boundary of the box.

Beside this it is noted that all median values, which are the green lines in the boxplots, lie close to each other and are all within each others interquartile range. Furthermore, the spread around this value is comparable for different model types. From this, it becomes clear that the results are inconclusive on which model performs best on the validation run. As a result, all types will be checked for their generalisation capabilities to identify if one type is preferred above another.

Apart from comparing the performances, the general prediction capabilities must be assessed as well. To get an impression of the average prediction performance, for each model the run resulting in the median RMSE value or closest to this value is plotted in Figure 6. The parameter combination used for these runs are reported as well. First it is noted that all model types detect the manoeuvre leading to loss-of-control correctly. Furthermore, although the time to LOC is overestimated at the beginning of the manoeuvre for all models, all predictions show a downward trend indicating that they observe stronger failure features closer to the loss-of-control event.

Finally, the models can be compared based upon training time, where a lower time is preferred. The average training time per model is depicted in Table 4.

Table 4. Average training time for the different model types trained on 75 failure runs

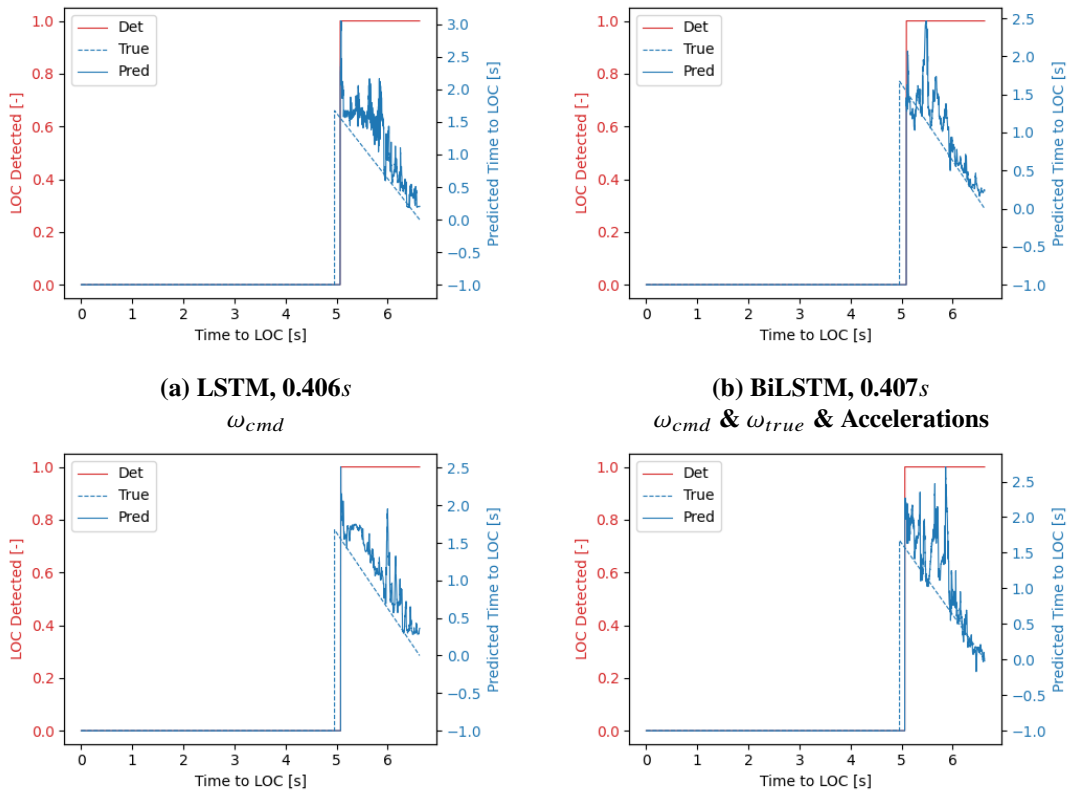
	Time [hh:mm:ss]
LSTM	01:08:00
BiLSTM	01:46:49
CNN-LSTM	01:15:02
GRU	01:04:54

The BiLSTM network has the longest training time, followed by the CNN-LSTM network, which is caused by the complexity of these networks. The GRU model can be trained faster than the LSTM network, which is in line with the expectation due to the lower amount of gates in a GRU cell compared to an LSTM cell.

B. Parameter Performance

Comparing the results for different parameter combinations is done in a similar fashion as for model type comparison. A boxplot is generated for each combination of parameters used during training, visualised in Figure 7. Each combination has twelve data points, which are the RMSE values for the four model types and three random training processes. This is the same data as is used to create the model type comparison boxplots, however grouped differently.

Opposed to the model type comparison, it is possible to observe preferences for specific parameters. By comparing the first three runs it is clear that including the commanded rotor values boosts performance. However, when looking at the fourth run, only us-



(a) LSTM, 0.406s
ω_{cmd}

(b) BiLSTM, 0.407s
ω_{cmd} **&** ω_{true} **& Accelerations**

(c) CNN-LSTM, 0.396s
ω_{cmd} **& Accelerations**

(d) GRU, 0.399s
ω_{cmd} **&** ω_{true} **& Heading**

Figure 6. Best performance for each model on the validation run including RMSE value and used parameters

ing these values is not powerful. When trying the remaining two parameters that show trends before LOC, acceleration and heading, it is observed that the heading does not contain early warning signals and counters the signals present in the commanded rotor values. These RMSE values are the ones that caused the extreme outliers in Figure 5. Combining the rotor command, true rotor output and the heading validate the negative impact of the heading on performance.

Next to this, from the other runs with three or four parameters, it is seen that combining more parameters does not improve performance, as the medians are similar to those with two parameters. This leads to preference for models that use two parameters, which are the commanded rotor values and either the true rotor outputs, gyro measurements or accelerations. The average RMSE value of the run including true rotor outputs is the lowest and therefore this combination will be used for generalisation performance analysis.

C. Generalisation Performance

Five different scenarios are tested where one condition is changed compared to the flights from test 1. To identify the generalisation capabilities, only the performance of the networks trained on commanded rotor values and true rotor outputs will be assessed. However, for those scenarios where there is a clear difference between used parameters, this will be pointed out as well.

1. Changing mass

The nominal mass of the Tiny Whoop including batteries is 56g. Five flights are performed where mass is added to the quadcopter by attaching coins to the frame. As it is assumed that the centre of mass is in the middle of the quadcopter, the coins are attached to this point to change the moment of inertia as little as possible. The resulting RMSE values for the different models and masses are presented in Figure 8. As

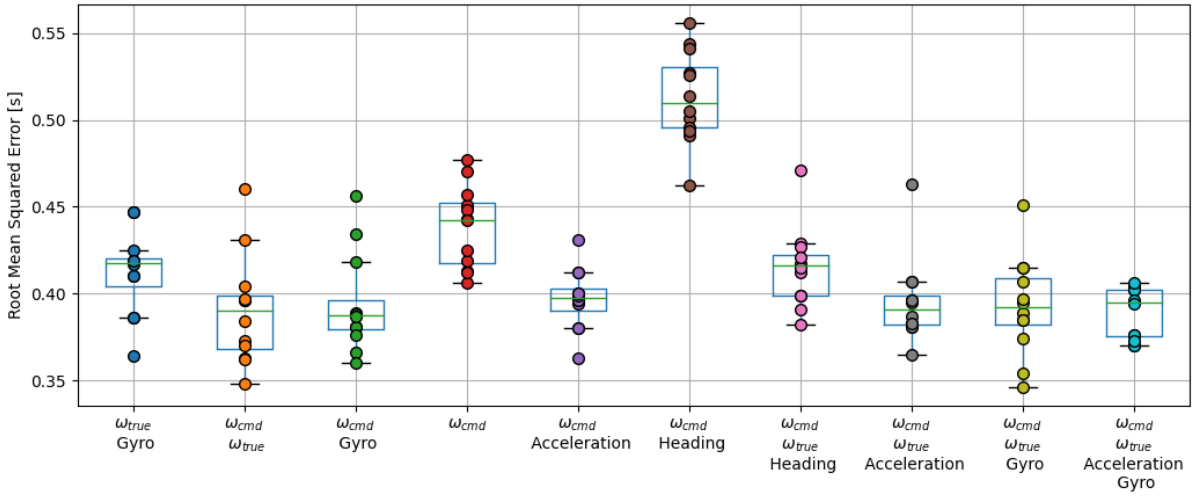


Figure 7. RMSE values in seconds for different parameter combinations

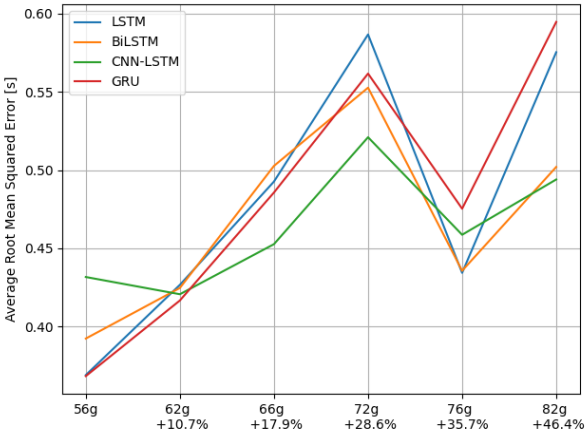


Figure 8. Average RMSE values in seconds for different quadcopter masses

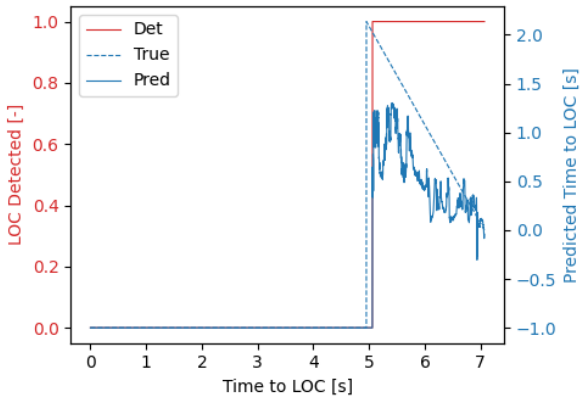


Figure 9. Prediction of the BiLSTM network on the heaviest quadcopter run

each model is trained three times to cancel out the randomness of the training process, the presented RMSE values are the averages per model per mass. The average RMSE values achieved on the validation run for the nominal quadcopter of 56g are depicted as well.

It is expected that for increasing mass, the prediction will be less accurate as higher mass differences change the dynamic behaviour of the quadcopter more. This trend is observed for all models, except for the run with a mass of 76g. This can be explained by looking at the time between the start of the dangerous manoeuvre and loss-of-control. This is between 2.1 and 2.4s for all runs, however only 1.67s for the run with a mass of 76g. This lower time to LOC leads to a lower RMSE score.

To investigate the suitability of these models as LOC predictor when mass changes relative to the nominal case, it is necessary to look at the prediction plots. For this, the heaviest run is assessed and the prediction with the median performance is chosen, which is the BiLSTM model with an RMSE value of 0.501s. Figure 9 shows this plot.

It is observed that the detection of the dangerous manoeuvre still works as desired. Furthermore, the

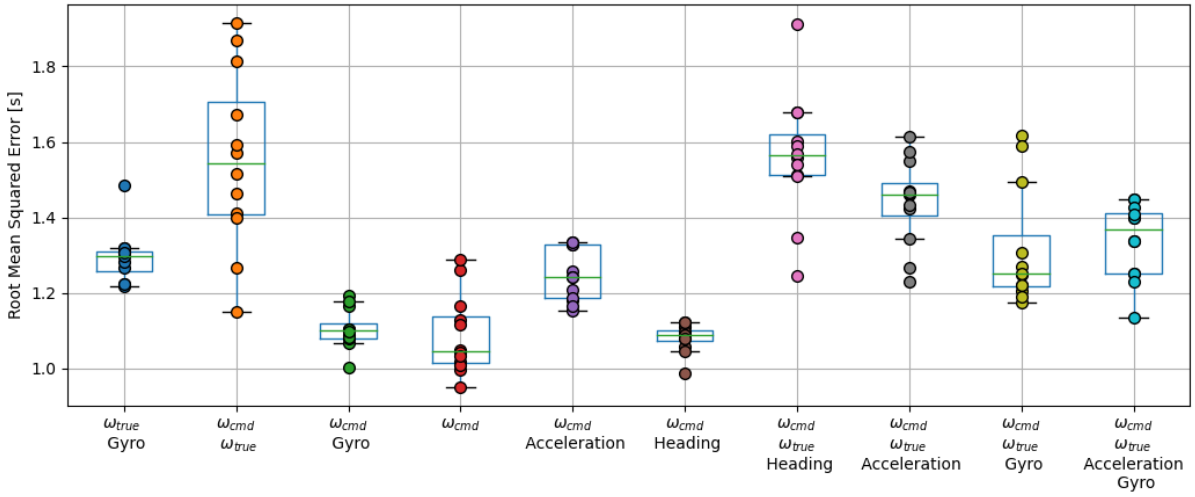


Figure 10. RMSE values in seconds for different parameter combinations on the first run with three blade propellers

prediction is lower than the true time to loss-of-control for the majority of the run, however, there still is a downward trend ending close to 0s at the start of the loss-of-control event.

2. *Different propellers*

In nominal condition, the Tiny Whoop has 40mm four blade propellers. To investigate if different propellers influence prediction capabilities, these were replaced by 30mm three blade propellers. For high speed flights, less blades are preferred as less drag is generated. This leads to the expectation that the quadcopter is able to perform the desired manoeuvre for a longer period of time, meaning that the time to loss-of-control is longer. In total, six flights are performed with these propellers. A first observation is that there is a clear difference in RMSE values between different parameter combinations. Figure 10 visualises this in boxplots, where the RMSE values for the first run with three blade propellers are used.

The runs where the true rotor output is used show higher RMSE values than other runs. Therefore, to assess the suitability of the networks for LOC prediction with different propellers, the models trained on the commanded rotor values and true rotor outputs will not be used. The models trained on commanded values only will be used instead, as these show the best performance on the first propeller run. The average RMSE values for the different propeller runs and different models trained on commanded rotor values only are tabulated in Table 5.

Table 5. Average RMSE values in seconds on different propeller runs using models trained on commanded rotor values

	Nominal	Props 1	Props 2	Props 3	Props 4	Props 5	Props 6
LSTM	0,369	1,043	**1,637**	**0,466**	1,260	0,560	**0,567**
BiLSTM	0,392	1,044	1,670	0,475	1,262	0,549	0,576
CNN-LSTM	0,432	1,239	2,375	0,514	1,516	0,611	0,653
GRU	**0,368**	**1,028**	1,642	0,562	**0,921**	**0,527**	0,634

Table 6. Time to loss-of-control in seconds

	Nominal	Prop 1	Prop 2	Prop 3	Prop 4	Prop 5	Prop 6
Time to LOC	1.700	4.007	5.507	2.142	4.721	2.645	2.027

There is a large difference between the six runs. This can be explained by looking at the time between the start of the dangerous manoeuvre and loss-of-control, summarised in Table 6. As expected, the time before the start of the loss-of-control event is longer than the time to LOC in the validation run. Furthermore, the longer the time to LOC, the higher the RMSE score.

To investigate why this happens, the prediction of one of the models on the second and worst propeller run is analysed. Figure 11 shows the prediction of the BiLSTM model, which scored the median RMSE

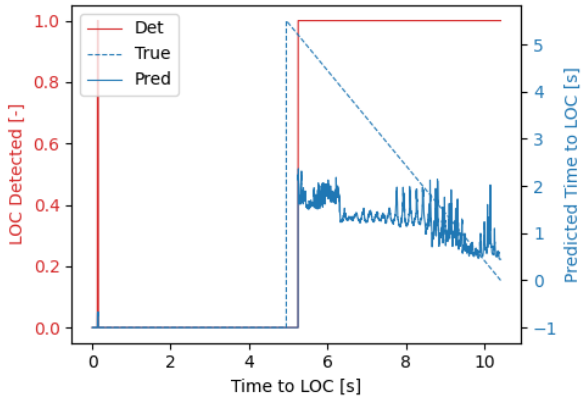


Figure 11. Prediction of the BiLSTM network on the second propeller run

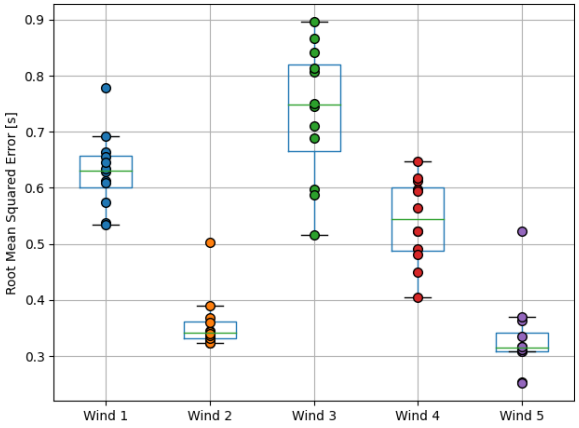


Figure 12. RMSE values in seconds for different runs in wind. Run 1, 2 and 3 represent constant wind, run 4 and 5 represent wind gusts

value on this run, which is 1.660s. Once the dangerous manoeuvre is detected correctly, the prediction time starts at around 2s, after which oscillating or downward prediction behaviour is visible. The reason for this is the average time to loss-of-control in the training set, which is 1.980s. The trained networks are not familiar with runs that take much longer and can therefore not produce correct prediction results for these runs. A longer time to LOC leads to a higher RMSE value because the difference between the prediction and the true time to LOC can be higher in comparison to shorter runs.

3. LOC due to high pitch rate

Another tested scenario is whether the models can generalise for different LOC events. Instead of creating loss-of-control by demanding a high yaw rate, it is created by demanding a high pitch rate. For this event, loss-of-control is defined similarly as for the nominal scenario. It is set to the moment in time where, after an initial small increase, the roll or yaw rate starts to show a linear increase. At this point, the quadcopter is not able to keep performing flips and starts to rotate around all its axes. This happens faster than for the flights from test 1: on average the time to LOC for the pitch runs is 0.470s. To aid the prediction of the models, the two runs with the longest time to loss-of-control are chosen, with a time to LOC of 1.1s for both runs. Even though these runs are used, all networks already fail in detecting the dangerous manoeuvre correctly.

4. Flying in wind

To create wind, the *Master DF 30P* fan is used, which has a constant airflow of 10200m^3/h, which equals an air velocity of 6.25m/s given the fan diameter of 760mm. Five flights are performed where wind is blowing on the quadcopter from the back left. During the first three flights, the quadcopter is flown into the stream before the manoeuvre starts, which can be seen as flying in constant wind conditions. During these flights, it is noted that the quadcopter is pushed away and upward by the stream. For the last two flights, the manoeuvre is started before it is flown in the stream, which can be seen as a wind gust working on the quadcopter. The vehicle is again pushed away by the wind, however it influences its behaviour a shorter period of time before the LOC event starts. It is expected that the wind gust influences the performance less, as the wind works on the quadcopter for a shorter period of time and for both cases the wind speed is the same. To investigate this, the RMSE values are plotted per run in boxplots, visualised in Figure 12.

Although the fifth run shows the lowest RMSE values, the second run performs much better than the fourth run. The prediction performances vary heavily and do not depend on specific wind conditions. The limited amount of tests in these conditions makes it therefore not possible to draw conclusions based upon specific wind conditions.

To explore the suitability of the models for LOC prediction in windy conditions, the prediction plots

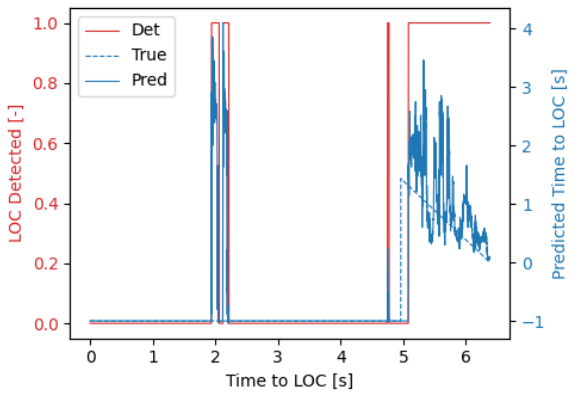


Figure 13. Prediction of the GRU network on the third run in windy conditions

are assessed. The worst condition is chosen, which occurred during the third run, as this one has the highest RMSE values. Figure 13 shows the prediction of the GRU network on this run, which scored the median RMSE value of 0.745s. The detection shows an error after 2s, which is caused by turbulent flight due to the presence of wind. The true dangerous manoeuvre is detected correctly. The prediction branch is primarily overestimating the time to loss-of-control and oscillates more compared to the nominal case, however a downward trend is still visible.

5. Different quadcopter

The final test consists of four runs where a different quadcopter is used: the URUAV UZ85. This quadcopter has a diagonal of 85mm, which is larger than the 75mm of the Tiny Whoop and weighs 73g at take-off compared to 56g for the Whoop. The quadcopter is crashed similarly to the runs from test 1 as described in subsection III.A. It is interesting to look at the scores from models trained on different parameter combinations, as large differences are observed.

The reason for this is the standardisation step that is taken as preprocessing method. For this, a mean and standard deviation are determined using data from the flights performed during test 1. When looking at the scores for all different combinations, as is done for the first flight with this quadcopter in Figure 14, it can be seen that the runs including the true rotor outputs perform worse than those without. The range of true rotor outputs is thus different for the URUAV UZ85 than for the Tiny Whoop.

To identify if the models are suitable for LOC prediction on different quadcopters, the models trained on commanded rotor values only are assessed. The run with the worst RMSE values is the second run. The GRU model scored the median RMSE value of 0.492s. The prediction plot of this model is shown in Figure 15. There are two short wrong detections, caused by the incorrect standardisation process. Apart from this, the detection branch works as desired. The prediction shows an initial downward trend, however, towards the end the network is overestimating the time to loss-of-control.

V. Discussion

COMBINING the outputs presented in previous section leads to new findings about early warning signals, the suitability of recurrent neural networks for loss-of-control prediction and the application of these models.

A. Loss-of-Control Early Warning Signals

The commanded rotor values show the clearest signals of an upcoming loss-of-control event, as noticed in Figure 7. This is an interesting result, as it implies that loss-of-control is driven by a desired behaviour instead of the true behaviour of the quadcopter.

An explanation for this can be found when comparing the commanded rotor values and true rotor outputs in Figure 4. The commanded rotor values show a minimum and maximum value, whereas the true rotor outputs do not hit their minimum or maximum value. Thus, only the commanded values show saturation during this manoeuvre. This validates the hypothesis stated in subsection III.C that rotor command saturation is a clear early warning signal for loss-of-control which can be detected easily by the networks.

Apart from identifying a clear early warning signal, it is also possible to exclude parameters. For the nominal case, including heading reduces the prediction performance of all network types, as shown in Figure 7, making the heading not suitable for LOC prediction. Next to this, for the different propellers and different quadcopter scenarios, models trained on the true rotor outputs show worse performance than other models, which can be seen in Figure 10 and

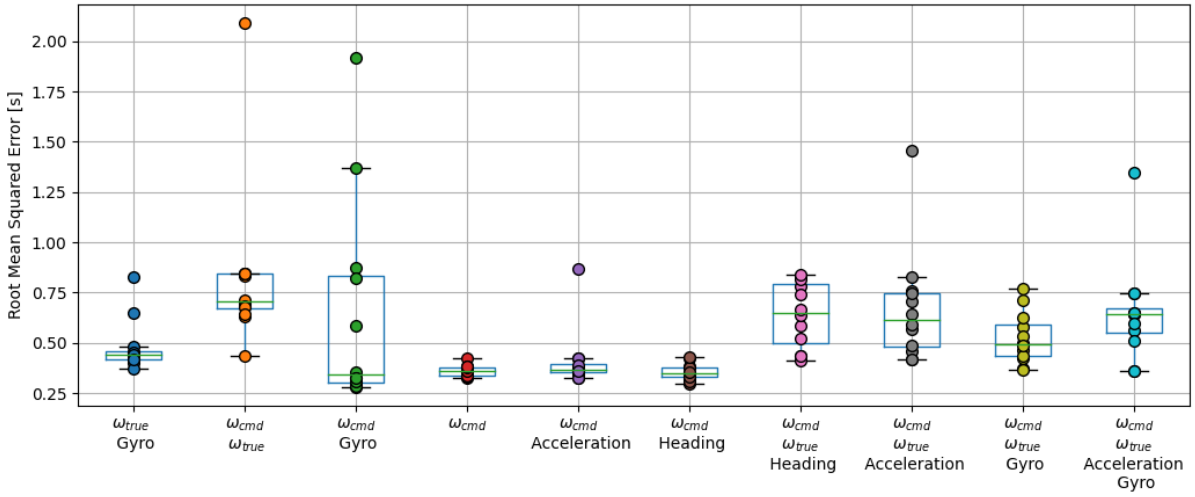


Figure 14. The boxplots are created using the RMSE values of UZ85 run 1

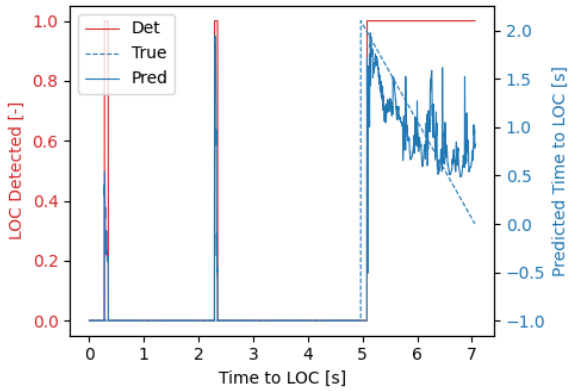


Figure 15. Prediction of the GRU network on the second UZ85 run

Figure 14 respectively. Although the combination of commanded rotor values and true rotor outputs perform best on the nominal run, other parameter combinations follow closely, making this combination not superior to other combinations.

Following this analysis, two combinations turn out to be most useful for loss-of-control prediction: commanded rotor values together with either gyroscopic measurements or the accelerations.

B. RNNs for Loss-of-Control Prediction

Based upon Figure 6, it can be concluded that recurrent neural networks can detect dangerous manoeuvres leading to loss-of-control correctly for scenarios it is trained on. Furthermore, the prediction branch overestimates the time to loss-of-control at the beginning of the manoeuvre, but a clear downward trend can be observed, following the slope of the true time to loss-of-control. Next to this, all networks show an average prediction error close to 0.400s. When using this RMSE value as an uncertainty margin for the output of the network, the prediction provides sufficient information for an (auto)pilot to understand that a loss-of-control event is coming closer and at what pace this is happening. This means that RNNs cannot only be used to detect a dangerous manoeuvre, but also to predict an upcoming loss-of-control event for scenarios it is trained on.

These results can be used for loss-of-control prevention of quadcopters. Especially autopilots serve a role here. The average time to LOC in the training set is almost 2s. Human pilots need time to process a received warning signal and to take action to recover to safe flight. Two seconds can be too short to do this. Therefore, in further research, the usage of a trained model in a closed-loop control system should be investigated. Combining the outputs of both the detection and prediction branch is most useful, to create an algorithm that can override pilot commands just-in-time.

The results on what model type can be used best in this system are inconclusive. All model types show a median RMSE value close to 0.400s on the validation

run and for the different generalisation scenarios the median values of the different models were also comparable to each other. This gives useful insight about the causes of loss-of-control. As both simple and complex RNNs perform similar, the failure features are less time-dependent than expected beforehand. This means that loss-of-control is caused by short-term, more instantaneous behaviour of the quadcopter. To confirm these findings, simple RNN structures or even traditional fully connected neural networks should be tested to see if they can still predict loss-of-control correctly.

C. Applications

For both changing mass and different propellers, the detection and prediction behaviour are similar. Even for the worst run encountered for both scenarios, the detection branch still detects the dangerous manoeuvre correctly, as depicted in Figure 9 and Figure 11 for changing mass and different propellers respectively.

For both scenarios, the prediction branch is underestimating the time to loss-of-control. For changing mass, this can be explained by looking into the consequences of the additional mass. To keep a heavier quadcopter in the air, higher thrust should be generated. The commanded and true rotor outputs will thus be higher, which for the nominal case happens closer to the loss-of-control event, leading to an underestimation. For different propellers, this can be explained by looking at the time to loss-of-control. As was already expected beforehand, the time between the dangerous manoeuvre and LOC is longer for the shorter 35mm, three blade propellers. As the average time to loss-of-control in the training set is only 2s, the outputted sensor values are associated with a shorter time to loss-of-control by the trained models, leading to an underestimation.

Another observation that can be done based upon the results from these two scenarios is that the time to loss-of-control influences the RMSE values. For changing mass, the run with a time to LOC 0.5 seconds lower than the other runs in this test resulted in a lower RMSE value than was expected. For different propellers, all runs were longer than the ones from the training set, which lead to higher RMSE values. These results imply that the application of the trained models is limited to failure runs with similar time to LOC as it is trained for.

On the other hand, for the presented scenarios the incorrect prediction behaviour can be explained and therefore accounted for. When investigating the usage of these models in a closed-loop system, it is advised to incorporate information about the aerodynamic characteristics of the quadcopter and the associated expected deviations of the prediction to the true time to loss-of-control. When these characteristics are changed, the output of the prediction branch can be adjusted according to the expected deviations, which would make these models suitable for LOC prediction for different aerodynamic characteristics.

When looking at the application of the trained models on different loss-of-control scenarios, which is done for the pitch runs in this research, it can be concluded that it is limited to known LOC scenarios. This can be explained by looking at how different scenarios alter the sensor outputs of the quadcopter. Other generalisation scenarios do not lead to complete new sensor outputs. Parameters show similar trends, only with different values. A new LOC scenario shows completely new, unknown behaviour and therefore unknown sensor outputs. One option to overcome this is to create a model for each LOC event, but for the sake of available memory on the quadcopter it is worth investigating if one model can be trained on different failure scenarios.

When flying in wind, the dynamic behaviour of the quadcopter is changed more than for different aerodynamic characteristics. When looking at Figure 13, false detections can be observed, as well as large differences between predicted and true time to LOC. Beside this, the predicted time oscillates with high amplitudes compared to the other scenarios, leading to unreliable predictions. This is due to turbulence caused by wind, which makes the movements of the quadcopter unpredictable. However, the dangerous manoeuvre is detected correctly and the prediction still shows a downward trend. It is therefore expected that for low wind speeds, the prediction will fall within the error margin of the nominal predictions, which makes these models suitable for LOC prediction only for limited wind conditions. To confirm this, more flights in windy conditions should be executed.

Last of all, when flying a different quadcopter, the detection branch can still detect the dangerous manoeuvre correctly, as shown in Figure 15. False

detections are done as well, however these only have a short duration. Initially, the predicted time to loss-of-control follows the true time to loss-of-control, however towards the end of the manoeuvre, the model overestimates the time to LOC. From this result it becomes clear that both quadcopters move similarly and show the same failure features during the dangerous manoeuvre. However, the exact output values of the sensors vary too much to give a correct prediction. The standardisation step taken when preparing the data before it is fed into the networks can serve as a solution for this issue. It is recommended to investigate if it is possible to get mean and standard deviation values for the new quadcopter from only a few failure runs, such that the sensor values are mapped to the same range as for the first quadcopter. When this is possible, the same model can be used for correct detection and prediction of loss-of-control for similar quadcopters.

Finally, similar algorithms can be created for other (autonomous) vehicles suffering from loss-of-control, where one important requirement is that sufficient failure runs can be performed such that the networks can learn failure features. To reduce the amount of necessary crashes, the use of data from simulators or generative models that are capable of mimicking failure data should be investigated. Only when these techniques can create sufficient failure data will it be possible to apply the proposed method on aircraft to make aviation safer.

VI. Conclusion

Loss-of-control remains the number one cause of crashes for both aircraft and drones. To decrease the amount of LOC events, on-board systems that can prevent this should be designed. Recurrent neural networks can be used for loss-of-control prediction in quadcopters using only on-board sensor measurements. The commanded rotor values show best early warning signals for loss-of-control and RNNs can predict a LOC best using these values in combination with either the gyroscopic or accelerometer measurements. However, the application is limited for scenarios that show similar behaviour as the ones from the training set in terms of loss-of-control scenario and time to loss-of-control. On the other side, for changing aerodynamic characteristics, the model prediction can still be used if the expected deviation in prediction behaviour is compensated for. To draw conclusions about the usability of the model in wind conditions and on different quadcopters, additional research is necessary.

References

[1] Belcastro, C. M., Newman, R. L., Evans, J. K., Klyde, D. H., Barr, L. C., and Ancel, E., "Hazards Identification and Analysis for Unmanned Aircraft System Operations," *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017.

[2] Boeing Commercial Airplanes, "Statistical Summary of Commercial Jet Airplane Accidents, Worldwide Operations | 1959 – 2020," , 09 2021. URL `http://www.boeing.com/news/techissues/pdf/statsum.pdf`.

[3] Belcastro, C. M., and Jacobson, S. R., "Future Integrated Systems Concept for Preventing Aircraft Loss-of-Control Accidents," *AIAA Guidance, Navigation, and Control Conference*, 2010.

[4] Di Donato, P. F., Balachandran, S., McDonough, K., Atkins, E., and Kolmanovsky, I., "Envelope-aware flight management for loss of control prevention given rudder jam," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 4, 2017, pp. 1027–1041.

[5] Sun, S., and de Visser, C. C., "Quadrotor Safe Flight Envelope Prediction in the High-Speed Regime: A Monte-Carlo Approach," *AIAA Scitech 2019 Forum*, 2019.

[6] Zhang, Y., Huang, Y., Chu, Q., and de Visser, C., "Database-Driven Safe Flight-Envelope Protection for Impaired Aircraft," *Journal of Aerospace Information Systems*, Vol. 18, No. 1, 2021, pp. 14–25.

[7] van der Pluijm, A., "Early Warning Signals for Loss of Control Prediction of a Damaged Quadcopter," Master's Thesis, Delft University of Technology, The Netherlands, 2020.

[8] Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., and Gao, R. X., "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, Vol. 115, 2019, pp. 213–237.

[9] Gers, F., Schmidhuber, J., and Cummins, F., "Learning to forget: Continual prediction with LSTM," *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, Vol. 2, 1999, pp. 850–855.

[10] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical*

Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

[11] Rezaeianjouybari, B., and Shang, Y., "Deep learning for prognostics and health management: State of the art, challenges, and opportunities," *Measurement*, Vol. 163, 2020, p. 107929.

[12] Zhang, L., Lin, J., Liu, B., Zhang, Z., Yan, X., and Wei, M., "Review on Deep Learning Applications in Prognostics and Health Management," *IEEE Access*, Vol. 7, No. 1, 2019, pp. 162415–162438.

[13] Fink, O., Wang, Q., Svensén, M., Dersin, P., Lee, W. J., and Ducoffe, M., "Potential, challenges and future directions for deep learning in prognostics and health management applications," *Engineering Applications of Artificial Intelligence*, Vol. 92, 2020, p. 103678.

[14] Sadhu, V., Zonouz, S., and Pompili, D., "On-board Deep-learning-based Unmanned Aerial Vehicle Fault Cause Detection and Identification," *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5255–5261.

[15] Wang, B., Liu, D., Peng, Y., and Peng, X., "Multivariate Regression-Based Fault Detection and Recovery of UAV Flight Data," *IEEE Transactions on Instrumentation and Measurement*, Vol. 69, No. 6, 2020, pp. 3527–3537.

[16] Géron, A., *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O'Reilly Media, Sebastopol, CA, 2019.

Part III

Discussion

7

Conclusion and Recommendations

The results presented in the paper together with the additional results presented in Part IV lead to conclusions regarding the research question as stated in Chapter 2. Furthermore, the results raise questions about loss-of-control prediction and prevention, which lead to a list of recommendations. Both are presented below.

7.1. Conclusion

The objective of the thesis research was to determine whether data-driven algorithms can generalise loss-of-control prediction of quadcopters for various scenarios. Neural networks were considered the most suitable method for this, leading to the following research question:

> Can a neural network generalise loss-of-control prediction of quadcopters with aerodynamic characteristics, operational conditions and LOC scenarios representative for aircraft in nominal condition?

From the results of the experimental phase it can be concluded that neural networks can generalise loss-of-control prediction only for varying aerodynamic characteristics. However, it is required to compensate the output of the network for expected deviations in predicted time to loss-of-control. For different wind conditions it is expected that a neural network can generalise LOC prediction for low wind speeds only, but this should be investigated further. Neural networks cannot generalise LOC prediction for different LOC scenarios. Lastly, it is expected that networks can generalise LOC prediction for different quadcopter types, however some failure runs with the new quadcopter need to be performed to obtain information for the standardisation step during data preprocessing. A critical note must be made that although the networks can generalise LOC prediction for some different conditions, the time to loss-of-control should be similar to the ones from the failure runs the network is trained on, which puts a large limitation on the usability of this algorithm.

These conclusions are made using the answers to various subquestions that were drafted at the start of the research. The conclusions will be summarised below.

Loss-of-control definition
Based upon literature review it was expected that loss-of-control can be defined by a sudden change in sensor data. During the real-life flight tests, it became clear that this sudden change should be defined for each loss-of-control scenario differently. When looking at sensor data, loss-of-control does not have one general definition, but should be tailored to the specific LOC event. It can be concluded that as long as the definition is applied consistently to all failure runs, the exact definition does not matter.

Data-driven methods and network architectures
Before the start of the research, it was expected that loss-of-control results from a series of actions that

build up to the moment in time where the LOC event starts. Therefore, time-dependent features should be identified from sensor data in order to predict LOC correctly. When comparing data-driven techniques presented in literature, it can be concluded that recurrent neural networks are best in detecting these features in time-series data automatically. Both simple and complex RNN structures exist and from literature the expectations are that complex structures perform better as they can identify more complex failure features.

During the research, surprising results regarding the best architecture were discovered. All RNN structures performed equally well and as an additional test, a fully-connected network was trained on the failure runs. This network performed better than the recurrent neural networks. With these results, it is not possible to draw any conclusions on which architecture performs best in predicting loss-of-control.

However, these results make it possible to draw conclusions about the loss-of-control problem itself. It can be concluded that the LOC prediction problem is less complicated than expected beforehand. Loss-of-control is the result of short-term, instantaneous behaviour of the quadcopter instead of the result of a set of consecutive manoeuvres. This is an important finding, as it implies that a dangerous manoeuvre can be interrupted only slightly before the start of loss-of-control to prevent this from happening.

Generalisation capabilities
Several conclusions can be drawn about the generalisation capabilities of the networks. For different aerodynamic characteristics, the time to loss-of-control is underestimated. However, this deviation can be explained when looking at the expected influence of the change on the behaviour of the quadcopter. Therefore, this can be compensated for in the output of the network, from which it follows that the network can be used for LOC prediction for different aerodynamic characteristics. Wind influences the behaviour more and in an unpredictable way. It is expected that for low wind speeds the network can still predict loss-of-control, however due to the limited amount of flights in wind no conclusions can be draw about this. Considering different LOC events, the conclusion is clear: prediction is not possible for events the network is not trained on.
Apart from these scenarios that were stated in the research subquestions, a final scenario was tested: using a different quadcopter. The results showed that the models overestimated the time to loss-of-control, however the reason for this is that a wrong mean and standard deviation value was used during standardising the data. It is expected that the networks can generalise for different quadcopters if new mean and standard deviation values are determined for new quadcopters, however further research is required before conclusions can be drawn.

7.2. Recommendations

When considering the conclusions presented previously, a list of recommendations can be drafted to help improve the prediction output of the models:

1. The result that the network can be used for events that have similar time to loss-of-control only raises questions about the variation of the training set. This seems to be biased towards a very specific loss-of-control event. It is therefore recommended to create a data set that contains failure runs that are more distinct from each other in terms of time to loss-of-control.
   Furthermore, it should be investigated further if one model can be trained on multiple LOC events. Based upon the results in Appendix B, where one network was trained on both a yaw and pitch LOC event, it is expected that this must be possible. Beside this, it is advised to investigate if runs with different aerodynamic characteristics and wind conditions can be included during training as well, to improve the generalisation capabilities.

2. For the generalisation scenarios, only four to six flights per different condition were performed. For different masses, only one flight per mass was flown and for wind conditions only one wind direction was tested, whereas in real-life wind can come from all directions. For both different propellers and quadcopter type the results among the different flights were inconsistent. Next to this, these flights were performed after the flights necessary for the training set, due to which

the quadcopter was already damaged. It is recommended to execute more flights for the different scenarios with a new quadcopter of the same type, to confirm the findings and expectations presented in this research for these different scenarios.

3. Another surprising result was that all RNN architectures performed similar. However, as noted in the preliminary study, more complex networks need more fine-tuning in order to perform better. It is advised to try various settings of the networks to see if performance can be improved. It is recommended to do this on a dedicated machine with high computing power, as optimising these networks on a standard laptop takes too long. Apart from this, simpler networks should be tested as well to identify if the LOC prediction problem is actually complicated enough for complex networks.

4. One result was that LOC is the result of instantaneous behaviour instead of the result of a series of manoeuvres. This raises questions about the loss-of-control definition that is used. In this research, LOC is defined as the moment in time where the desired manoeuvre cannot be performed anymore as desired. It is recommended to investigate new definitions of loss-of-control where LOC is defined as the moment in time after which recovery is not possible anymore.

Apart from improvements on the presented work, there are also recommendations about the usability of this model:

1. Currently, the model is used off-board and not in real-time. In a closed-loop control system, this model can be used on-board in real-time to help prevent loss-of-control. It is recommended to investigate how this model would fit into this system, where control rules should be designed that convert the output of the model to specific control actions. Possible actions are overwriting the autopilot to recover from the manoeuvre or providing a warning signal to the pilot that the manoeuvre should be ceased. Beside this, the possibility to dampen the pilot input once a dangerous manoeuvre is detected should be investigated as well.

2. It is expected that the presented algorithm cannot only be used for loss-of-control prediction on quadcopters, but also on other vehicles suffering from loss-of-control. This is because the definition of loss-of-control can be tailored for the specific LOC event and therefore for the specific vehicle. It is recommended to test this on different vehicles, such as (small) self-driving carts. However, as it is not always possible to crash vehicles over and over again, techniques should be researched that can create failure data, such as a simulator, or can create additional data from a few known failure runs, such as generative neural networks. Only with working data augmentation techniques it will be possible to apply the created algorithm on aircraft to make aviation safer and more sustainable.

Part IV

Appendices

A

Simulator Details

To prevent unnecessary quadcopter crashes, failure data will be created in a Parrot Bebop 2 simulator. This data will be used to test the suitability of recurrent neural networks for loss-of-control prediction. This appendix provides details about the simulator and the process applied to create the data. Furthermore, the method to preprocess the data is described. Information about the network architecture is written down as well and results that were found using the simulated failure data is provided.

A.1. Data Generation and Processing

A high-level overview of the Bebop simulator is shown in Figure A.1. The quadcopter model is a gray-box model based upon real-life flight test data, due to which complex aerodynamic effects are encapsulated in the behaviour of the quadcopter [35]. The model has been created to correctly predict behaviour of the quadcopter in a speed regime from 0 to 14 m/s, however the pitch and roll angle were limited during the tests, which makes the model less accurate for large pitch and roll angles. This is an important limitation for this research as high angles are required.

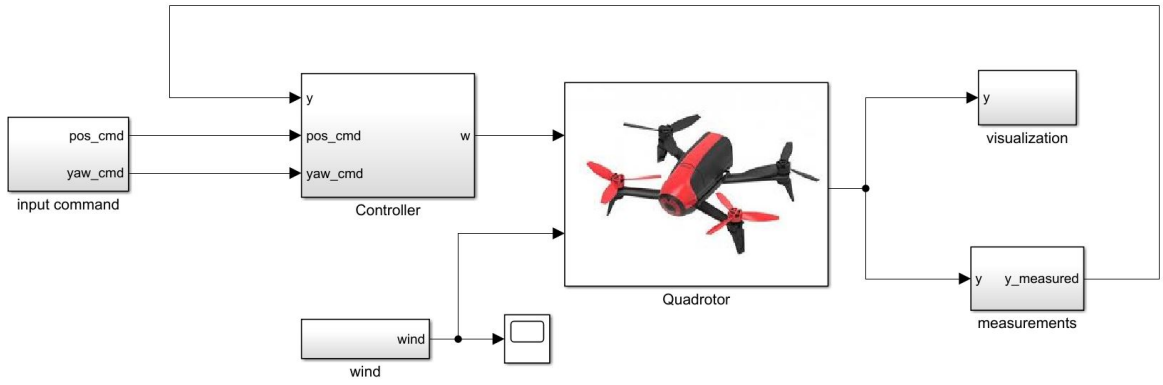


Figure A.1: High-level Simulink model of the Parrot Bebop 2

To simulate increase in pitch rate, the commanded pitch acceleration $\dot{q}$, calculated in the controller, is overruled. To create a diverse set of failure runs, this value was altered each run by adding 1 rad/s^2. A failure run consists of an initial phase of 5 seconds, where the quadcopter is flying in nominal flight. A diverse set of initial conditions was used, such as hovering, flying in circles or eclipses. After 5 seconds, the dangerous pitch manoeuvre started and the pitch rate increased over time. The quadcopter reached a LOC event once the model of the Bebop generated too much drag such that not enough thrust was present to continue flipping. This is not representative for real-life flights, as in real-life the quadcopter starts rotating around all its axes due to rotor saturation. This cannot be simulated, but the results obtained from the simulations are still useful, as it gives a good indication about preprocessing methods and whether these networks can deal with complex sensor data generated by a quadcopter.

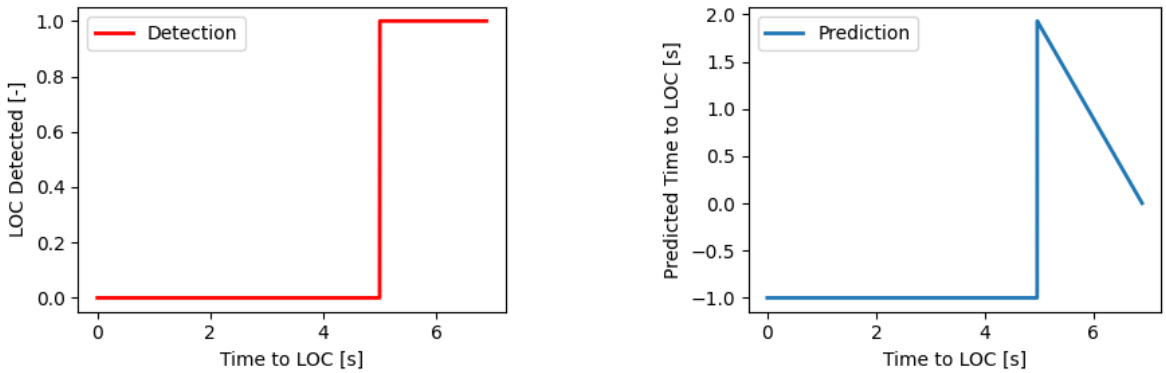
Before feeding the data into the networks, several preprocessing techniques were applied. First, only specific parameters were picked, as these are available on-board during flight and these parameters are most relevant for this type of loss-of-control. These are the vehicle rates, p, q and r, and the rotor outputs, $\omega 1$, $\omega 2$, $\omega 3$ and $\omega 4$. Furthermore, parameters were standardised. This helps to prevent exploding gradients during training of the networks. Lastly, instead of inserting data points one-by-one, clusters of 20 consecutive data points were used. This emphasises the time dependency of the data points and improves network performance.

A.2. Network Architecture

Four RNN types are trained, which are a long short-term memory (LSTM) network, bidirectional LSTM network (BiLSTM), LSTM network preceded by a convolutional network (CNN-LSTM) and gated recurrent unit (GRU) network. Neural networks have different characteristics which need to be chosen upfront and need fine-tuning for optimal network performance. The different characteristics and their default settings are presented in Table A.1. The main purpose of this phase of the research is to investigate if there are clear advantages for one or a few model types. Next to this, network settings are altered to investigate which result in best performance. Due to long training time of the networks, this is limited to a few characteristics only and for limited amount of settings.

Table A.1: Default network settings

Characteristic	Setting
Hidden layers	2
Dimension hidden layers	100, 150
Dimension final layer	600
Dropout rate	0.1
Batch size	16
Training epochs	15
Window Size	20



(a) Detection of dangerous manoeuvre

(b) Prediction to Loss-of-Control

Figure A.2: Outcome of the two different branches of the neural network

Last of all, the output of the networks should be determined in order to label the data. The desired output is a prediction of the time to loss-of-control in seconds. However, to avoid false predictions during nominal flight, the networks should be able to detect nominal and dangerous flight, where nominal flight is represented by '0' and dangerous flight by '1'. This leads to a network with two output branches, where the ideal output of the networks is as visualised in Figure A.2. The first 5 seconds are nominal flight, after which a dangerous manoeuvre starts leading to a LOC event after 1.9 seconds. Figure A.2b shows the output of the detection branch, whereas Figure A.2a shows the output of the prediction branch.

A.3. Nominal Results

In total, 50 failure runs were simulated where the quadcopter was in nominal condition, flying in a windless environment. Each run has a unique combination of $\dot{q}$ and initial condition setting. It should be noted that for comparing different network characteristics, not always all default settings were used. However, when testing different options of one characteristic, the remaining settings were kept unchanged, which still allows for comparison of this characteristic. Only the characteristic that is changed and what options are used will be mentioned in this section.

Amount of hidden nodes

The first characteristic that is tested is the amount of nodes in the hidden layers, as this influences training time as well as the amount of memory that should be available on the quadcopter. The different settings and associated RMSE values per model are plotted in Figure A.3. Except for one high outlier, the LSTM and GRU network show an increase in RMSE value. The BiLSTM and CNN-LSTM perform similar for all different settings, except for one low outlier for the CNN-LSTM network. From this it can be concluded that a lower amount of nodes is preferred, as this also leads to less memory necessary on the quadcopter. Therefore, for the experimental phase, tests will be performed with an even lower amount of nodes of 50 and 100.

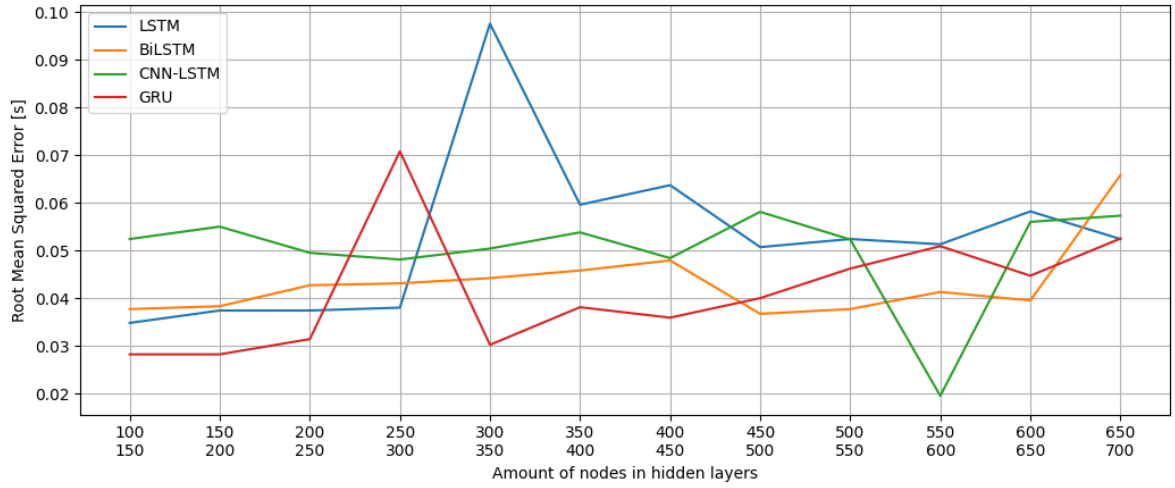


Figure A.3: Root Mean Squared Error values in seconds for different amount of hidden nodes

Window size

Another characteristic that is tested is the window size, as this is problem dependent and can therefore influence prediction performance significantly. Four settings are tested and the resulting RMSE values are tabulated in Table A.2. A clear preference for a window size of 20 is shown, which will therefore be the used setting for the experimental phase.

Table A.2: RMSE values in seconds for different window sizes

Network Type	15	20	25	30
LSTM	0.0433	**0.0427**	0.0891	0.0467
BiLSTM	0.0441	**0.0381**	0.0432	0.0396
CNN-LSTM	0.0644	**0.0459**	0.0581	0.0468
GRU	0.0376	**0.0245**	0.0305	0.0256

Final layer dimension

All networks consist of two RNN layers, which are LSTM, BiLSTM or GRU layers. The CNN-LSTM network has three layers before these two layers, which are two convolutional layers and a pooling layer. All networks end with two branches containing two dense layers, where the final layer has one

node, such that only one value is outputted. The number of nodes in the other final layer can be tuned for optimal performance. The RMSE values are plotted in Figure A.4 for different amount of nodes in the final layer.

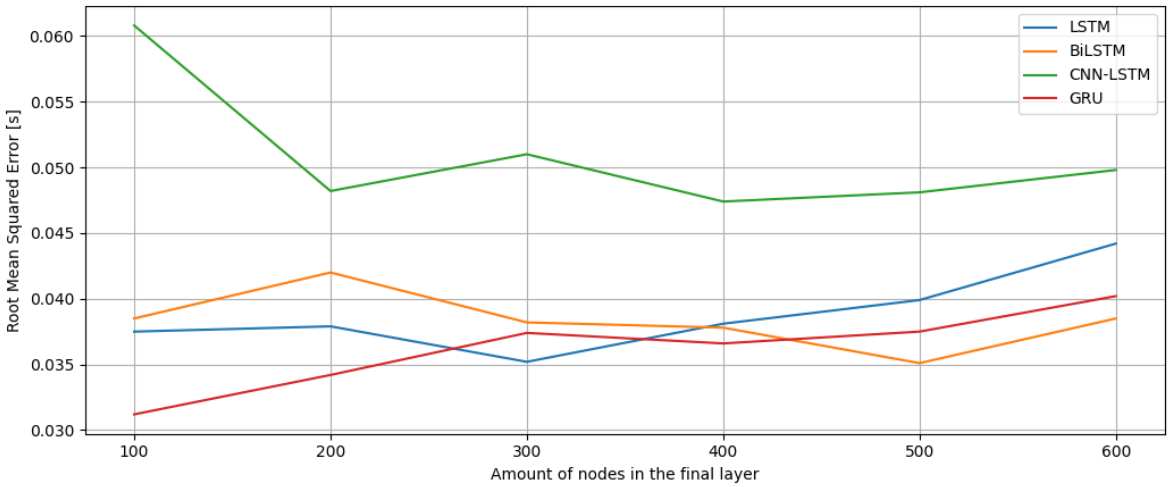


Figure A.4: Root Mean Squared Error values in seconds for different amount of nodes in the final layer

From this figure it becomes clear that the amount of nodes in the final layer does not influence the prediction performance massively, as all models show similar performance for different amount of nodes. The only model that shows larger changes is the CNN-LSTM model. This model performs best with 400 nodes. Next to this, in Table A.3, the average RMSE value of all models for all amount of nodes is tabulated, where 400 nodes also scores best. Therefore, the networks in the experimental phase will have 400 nodes in the final layer.

Table A.3: Average RMSE values in seconds of all models for different amount of nodes in final layer

	100	200	300	400	500	600
Average	0.0420	0.0406	0.0405	**0.0400**	0.0402	0.0432

Table A.4: RMSE values in seconds for a run with and without the additional feature

Network Type	Without additional feature	With additional feature
LSTM	0.0850	0.0379
BiLSTM	0.101	0.0420
CNN-LSTM	0.389	0.0482
GRU	0.146	0.0342

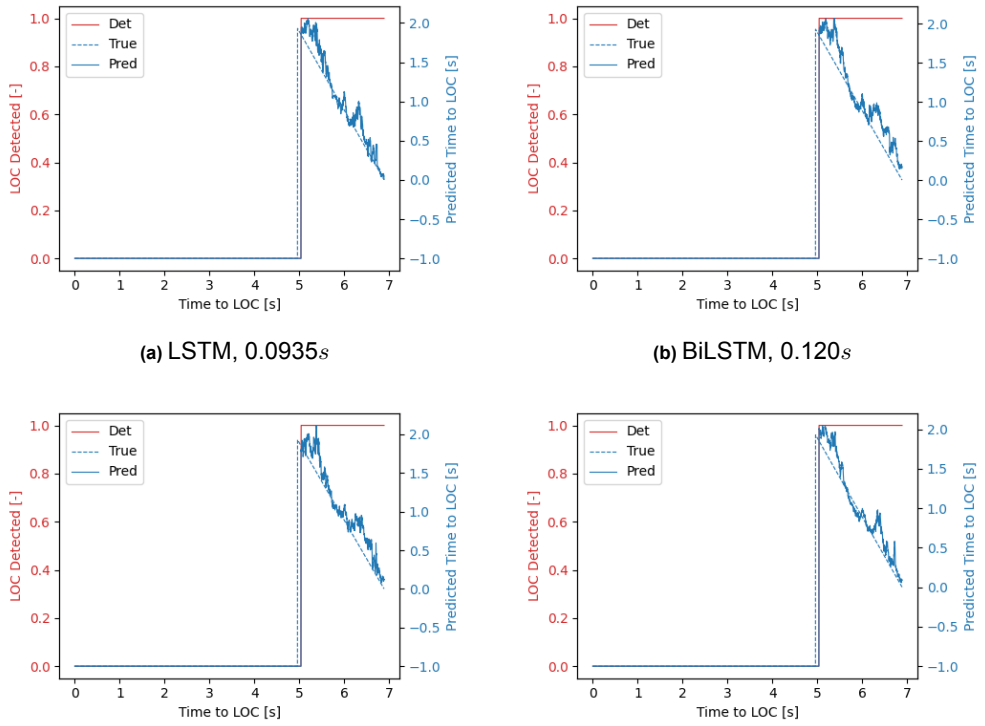
Handcrafted feature

To improve network performance, the influence of providing useful information to the neural network on its performance is investigated. As mentioned by Van der Pluijm [29], rotor saturation occurs prior to loss-of-control. It is expected that providing information about this boosts performance. An additional feature is created which counts the amount of saturated rotors, which is fed into the network together with the sensor data. RMSE values of a run with and without this feature are summarised in Table A.4.

The RMSE values are at least a factor 2 lower, and even go up to a factor of 8 lower for the CNN-LSTM

network when including this feature. These results validate that rotor saturation is a clear sign for loss-of-control and will therefore be included during the experimental phase as well.

Prediction performance
Once most of the settings have been determined, the general prediction performance can be assessed. This will lead to a hypothesis about the suitability of recurrent neural networks for loss-of-control prediction on quadcopters in real-life, which will be validated in the follow phase of the research. For this assessment, a validation run is created where the quadcopter characteristics and operational conditions are the same as the one from the training set, however a different initial condition was used. The prediction plots for all four models and the achieved RMSE values are visualised in Figure A.5.



(a) LSTM, 0.0935s **(b)** BiLSTM, 0.120s

(c) CNN-LSTM, 0.104s **(d)** GRU, 0.0924s

Figure A.5: Performance of all networks on the validation run, including RMSE value in seconds

All models can detect the dangerous manoeuvre correctly and the prediction is also accurate. There is a downward trend following the true time to loss-of-control. This prediction would be sufficient to serve as a warning for an (auto)pilot to recover to safe flight on time. Furthermore, the two more complex networks perform worse than the simpler networks, but the difference is not large. It is not possible to exclude one of the models based upon these results. On a real-life flight data set, it is expected that RNNs can detect a dangerous manoeuvre leading to loss-of-control correctly. The prediction will be slightly off, however sufficient to warn the (auto)pilot on time to recover to safe flight.

A.4. Generalisation Results

Another key topic of this research is to identify whether recurrent neural networks can generalise LOC prediction for various conditions. During the simulation phase, two cases were tested: changing mass of the quadcopter and flying in windy conditions.

Changing mass
The nominal mass of the Bebop 2 is 0.51kg and a mass of 0.55kg, 0.60kg, 0.75kg and 0.80kg were tested. The reason for the larger step between 0.60kg and 0.75kg is that the prediction was still almost

perfect for 0.60kg and therefore higher masses could be tried. The RMSE values for different models for different masses are visualised in Figure A.6. It can be observed that for increasing mass, the RMSE values increase as well. To set an expectation for the generalisation capabilities of RNNs on real-life quadcopter data, the prediction plots should be inspected. It is chosen to compare the ones from the worst performing model to see the influence of changing mass best: CNN-LSTM. The plots are visualised in Figure A.7.

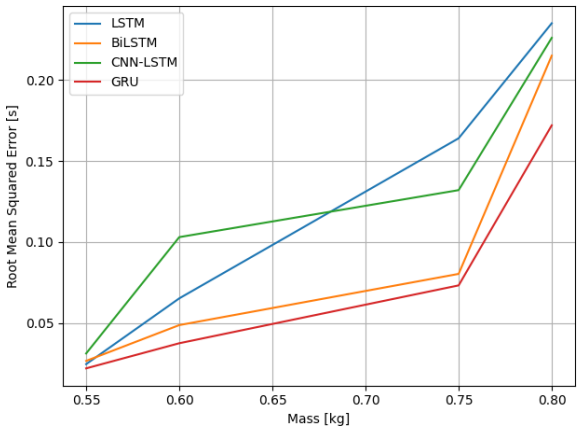
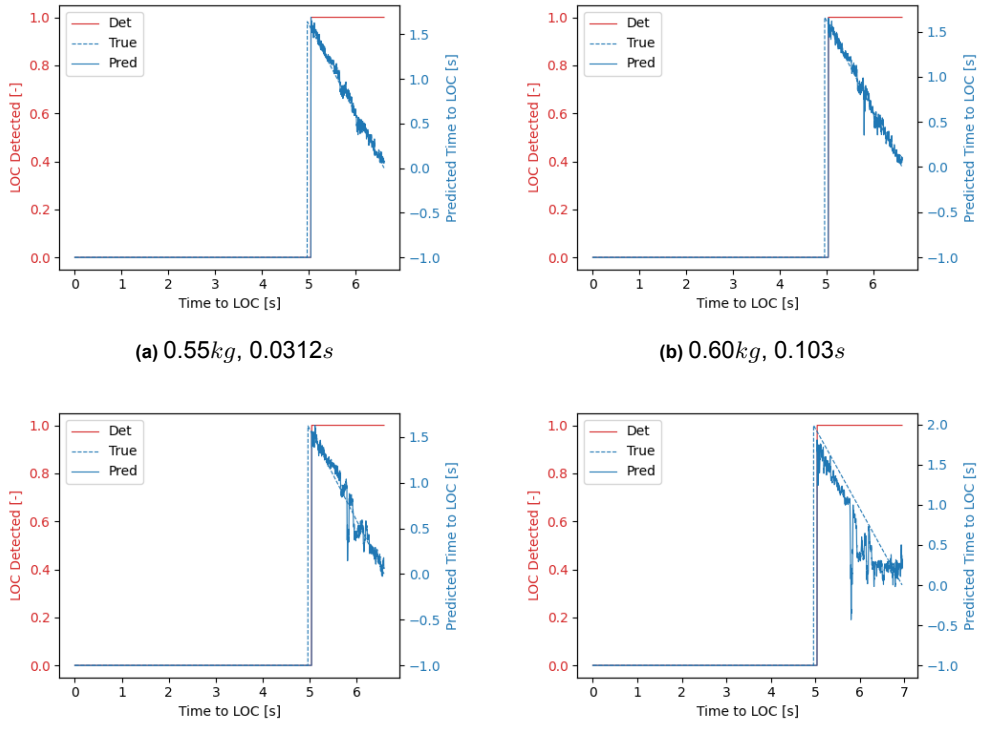


Figure A.6: Root Mean Squared Error values in seconds for different quadcopter masses and different model types



(a) 0.55kg, 0.0312s

(b) 0.60kg, 0.103s

(c) 0.75kg, 0.132s

(d) 0.80kg, 0.226s

Figure A.7: Performance of the CNN-LSTM network on different masses, including RMSE values in seconds

From this figure it can be observed that changing mass influences the behaviour of the quadcopter such that the prediction is too low for large changes in mass. This is shown for a mass of 0.80kg in Figure A.7d. However, it should be noted that a mass of 0.80kg entails an increase in mass of 63.75%, which is unlikely to happen during nominal flight of an aircraft. Next to this, the prediction behaviour

shown here is too optimistic compared to the real-life case, as this data will show more oscillations and noise and therefore influence the prediction more. However, based upon the results here it is expected that the RNN models can still detect a dangerous manoeuvre correctly and predict loss-of-control for changing mass, however the networks will be underestimating the time to loss-of-control. This will be validated during the real-life flight tests.

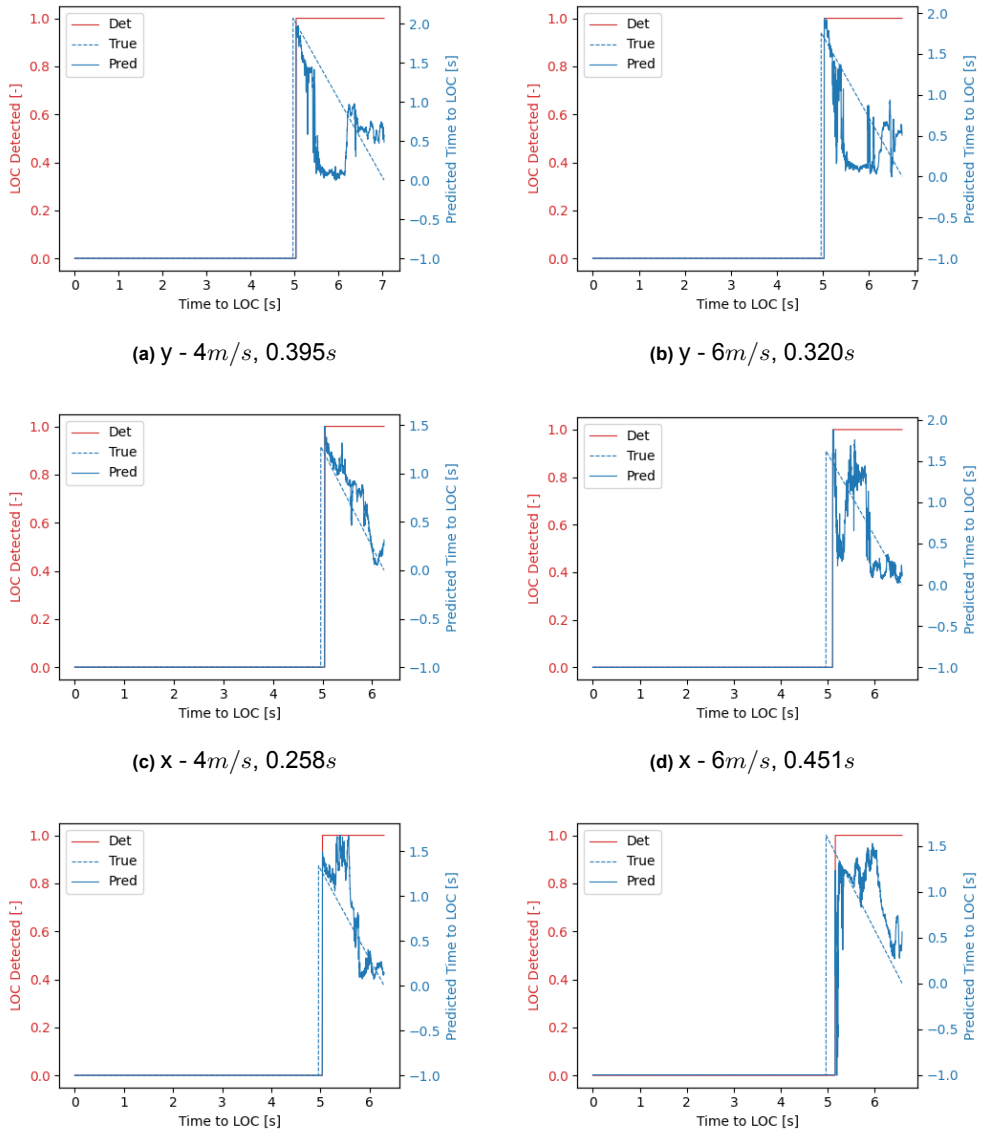
Wind conditions
The Bebop simulator can simulate a constant wind from different directions. Several wind speeds and directions are tested to see if this results in differences in prediction performance. In total, six scenarios are tested. Two tests are performed where wind blows from the y-direction with a speed of 4 and 6 m/s. The same tests are performed with wind blowing from the x-direction. During the final two tests, wind blows from both x- and y-direction with a speed of 3 and 5 m/s. The achieved RMSE values per model are depicted in Table A.5.

Table A.5: RMSE values in seconds for runs with different wind conditions

Network Type	y - 4 m/s	y - 6 m/s	x - 4 m/s	x - 6 m/s	x&y - 3 m/s	x&y - 5 m/s
LSTM	0.395	0.320	0.258	0.451	0.189	0.529
BiLSTM	0.437	0.289	0.122	0.446	0.207	0.748
CNN-LSTM	0.634	0.767	0.144	0.414	0.171	0.685
GRU	0.310	0.249	0.0909	0.283	0.161	0.230

First of all, it is observed that the values are higher than for the validation run or changing mass. Wind influences the dynamic behaviour of the quadcopter more than change in mass, which does not benefit the prediction performance of the RNNs. Second of all, it is not possible to draw conclusions for specific wind conditions. When comparing the tests with wind from either x- or y-direction, they both perform best for one of the two wind speeds. On the other side, when comparing the wind from both directions to only one direction, it is noted that this reduces performance.

To construct a hypothesis about the suitability of these models for LOC prediction when flying in wind conditions, the prediction plots should assessed. The prediction plots of the LSTM network for different wind conditions are drawn up in Figure A.8. This figure makes clear that wind makes the behaviour of the quadcopter unpredictable. The provided time to loss-of-control deviates from the true time in different ways for all wind conditions, making it hard to use the predicted time to LOC in a loss-of-control prevention system. On the other side, the detection of the dangerous manoeuvre still works as desired, except for the conditions with most wind: 5 m/s from both x- and y-direction. The detection is too late here. The hypothesis is therefore that the prediction cannot be used when applied to real-life flight data, as wind is more unpredictable in real-life than in a simulator. However, due to correct detection, for lower wind speeds the algorithm can still serve as an indication that there will be an upcoming loss-of-control event.



(a) y - $4m/s$, $0.395s$

(b) y - $6m/s$, $0.320s$

(c) x - $4m/s$, $0.258s$

(d) x - $6m/s$, $0.451s$

(e) x&y - $3m/s$, $0.189s$

(f) x&y - $5m/s$, $0.529s$

Figure A.8: Performance of the LSTM network for different wind conditions, including RMSE values in seconds

B

Experiment Details

The final phase of the thesis research consists of validating results found in the simulator phase. To confirm that recurrent neural networks can predict a loss-of-control event, a series of real-life flight tests is conducted to collect failure data. This appendix provides the validation of the simulation phase and additional results that were not written down in the thesis paper.

The aim was to fly with the Bebop 2, however for the sake of safety and damage prevention a smaller quadcopter was used: the Eachine Trashcan Tiny Whoop. Less damage occurs on this vehicle when it is crashed into the ground hundreds of times compared to the larger and heavier Bebop 2. Furthermore, the models that are used to produce the results shown in this appendix are trained on rotor commands and accelerations, as this was one of the two combinations that turned out to perform best.

B.1. Validation of Simulation Results

From the simulation phase, hypotheses were formed about the suitability of the models for prediction of LOC for changing mass and in wind conditions. The quadcopter used during experiments is not the same as the one in the simulator, but as the dynamics of all quadcopters is the same, the expectation is that the hypotheses are still valid. However, as the Tiny Whoop is lighter, the wind might have a larger influence on this vehicle than on the Bebop 2. These expectations will be validated here.

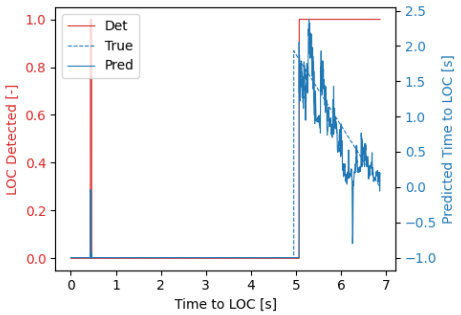
Changing mass

The expectation for prediction of loss-of-control for changing masses of the quadcopter is that the RNNs are still capable of doing this, however the prediction will be lower than the true time to LOC. To validate this, the prediction plots for all mass configurations for the GRU model are presented in Figure B.1.

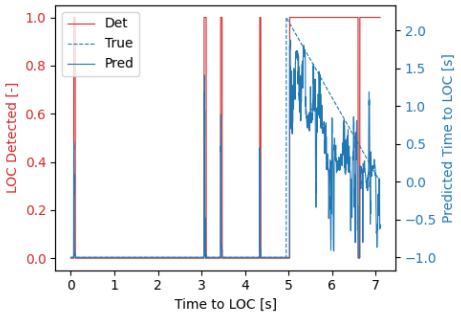
For all mass conditions, the model is detecting the dangerous manoeuvre correctly. The only incorrect behaviour that is visible is wrong detections of dangerous manoeuvres before the true manoeuvre, which becomes more frequent for higher masses. The reason for this is that the larger the mass difference with the nominal condition, the more the dynamic behaviour of the quadcopter is influenced, leading to sensor data which the network associates with dangerous manoeuvres. Beside this, for all different masses the model shows the expected prediction behaviour, where the network is underestimating the true time to loss-of-control. Both these observations are in line with the hypothesis, validating the results obtained during the simulation phase.

Wind conditions

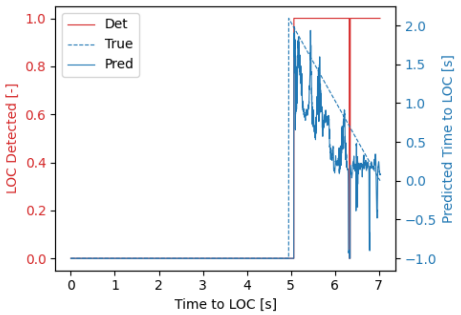
The results from previous phase indicated that wind influences the dynamic behaviour of the quadcopter more than changing mass, leading to predictions that are more off compared to this scenario. The expectation is therefore that the prediction branch of the networks is too unreliable to be used. However, the detection branch should still be able to detect the dangerous manoeuvre correctly, such that the networks can serve as an indication that a LOC event is upcoming. To validate this, the prediction plots for the GRU model on the wind runs are provided in Figure B.2. During the first three runs,



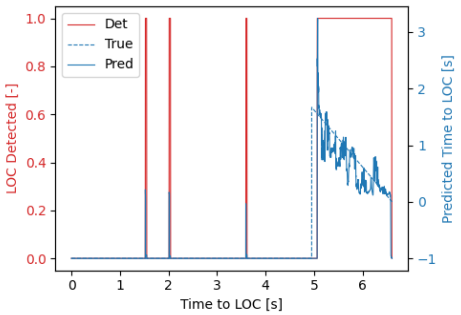
(a) 62g (+10.7%), 0.417s



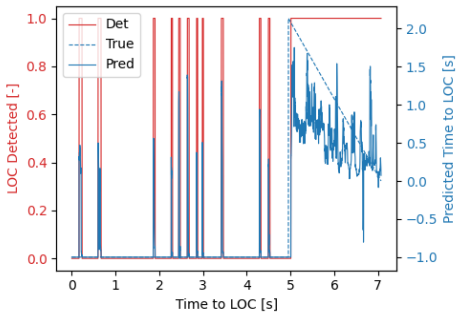
(b) 66g (+17.9%), 0.520s



(c) 72g (+28.6%), 0.497s



(d) 76g (+35.7%), 0.382s

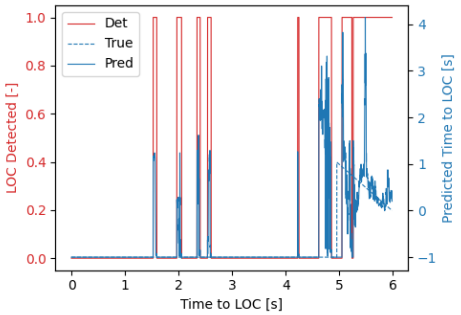


(e) 82g (+46.4%), 0.525s

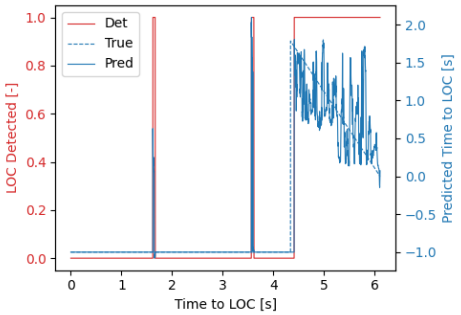
Figure B.1: Performance of the GRU network for different masses, including RMSE values in seconds

the quadcopter flew in a wind stream before the dangerous manoeuvre started, which represents a constant wind condition. During the last two runs, the quadcopter was flown into the wind stream after the start of the manoeuvre, representing a wind gust.

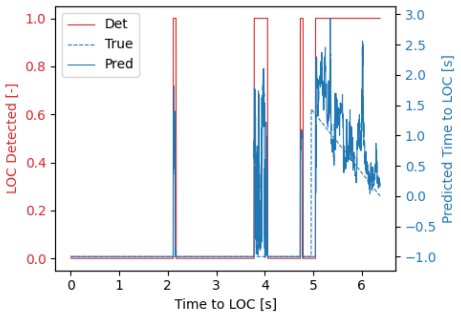
The results vary among the different runs. Although the dangerous manoeuvres are detected correctly for the majority of the time, there are also wrong detections, both false positives and false negatives. These result from the turbulent behaviour of the quadcopter caused by the wind. Furthermore, once the dangerous manoeuvre has started, the prediction shows fast oscillations with large amplitudes around the true time to loss-of-control. The latter confirms that the prediction time is too unreliable to be used. The detection branch however shows inconsistent behaviour with the hypothesis, which can be attributed to the difference in quadcopter type in the simulator and in real-life. On the other side, when looking at the difference between both vehicles, it was expected that the influence of wind is larger than was hypothesised. In conclusion, the differences between the simulator and real-life are too large to validate the simulator results.



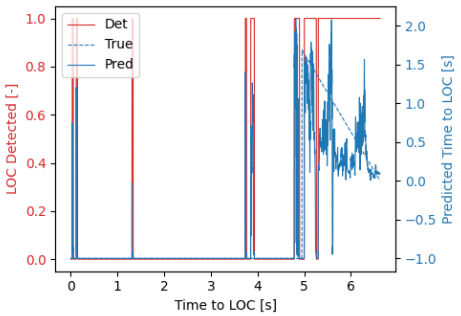
(a) Constant wind 1, 0.743s



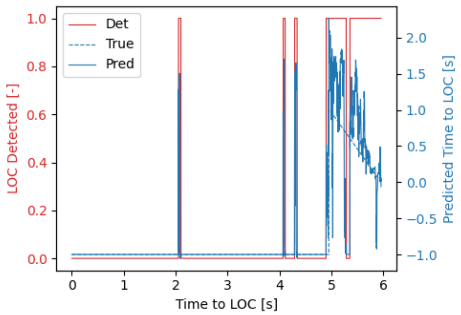
(b) Constant wind 2, 0.417s



(c) Constant wind 3, 0.552s



(d) Wind gust 1, 0.505s



(e) Wind gust 2, 0.382s

Figure B.2: Performance of the GRU network for different wind runs, including RMSE values in seconds

B.2. Additional Results

Apart from the results presented in the thesis paper, additional research was performed. These focused on some of the findings and recommendations that were done in the paper.

Fully-connected neural network

Recurrent neural networks can be used for analysis of sequential data, as they are capable of detecting time-dependent features. However, during the research it became clear that there is no difference in performance between simple and complex RNNs, indicating that failure features are less temporal dependent than expected beforehand. To confirm this, one recommendation is to test simpler networks. Therefore, a fully-connected neural network is created, which has the same lay-out as the RNNs, however the RNN layers are replaced by dense layers. The average RMSE values over three random training processes for the five different networks achieved on the validation run are depicted in Table B.1.

Table B.1: RMSE values in seconds on the validation run for a fully-connected neural network and four RNNs

Network Type	Validation run
Normal Network	0.341
LSTM	0.374
BiLSTM	0.406
CNN-LSTM	0.409
GRU	0.399

The fully-connected network performs best on the validation run. This confirms that there are less prominent time-dependent failure features than was anticipated before the start of this research. Loss-of-control is thus the result of short-term, more instantaneous behaviour instead of induced by a series of actions. However, it should be noted that the hyperparameters of the RNN models are not optimised. Default network settings are used, whereas complex networks usually need more fine-tuning to detect complex features and perform better than simpler networks. Before disregarding the RNNs, this should be tried first.

Handcrafted feature

Based upon the research performed by Van der Pluijm [29], the results from the simulation phase and the result that the commanded rotor values provide clearest early warning signals for loss-of-control, it is expected that the additional feature counting the amount of saturated rotors improves prediction performance. To confirm this, a run is performed where this feature was excluded. Table B.2 depicts the RMSE values of this test run, as well as the average RMSE values over the three random runs trained on commanded rotor values and accelerations including this feature.

Table B.2: RMSE values in seconds for a run with and without the additional feature

Network Type	Without additional feature	With additional feature
LSTM	0.389	**0.374**
BiLSTM	**0.380**	0.406
CNN-LSTM	**0.393**	0.409
GRU	**0.379**	0.399

The results are surprising, as the run without the additional feature performs better for three out of four networks. It should be noted that this is only one run, whereas the RMSE values including the feature are an average over three runs. However, the result raises questions about the additional value of this feature, as they indicate that the quadcopter data already provides sufficient information about an upcoming LOC event. More training runs should be executed to confirm this. Furthermore, the generalisation performance should be assessed as well when the additional feature is excluded.

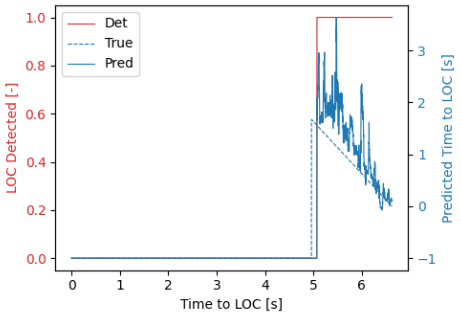
Pitch and yaw

One recommendation following from the research is to investigate if one model can be trained to predict loss-of-control correctly for multiple loss-of-control events. Beside 75 successful failure runs by demanding a high yaw rate, 53 runs were executed where a LOC event was successfully initiated by demanding a high pitch rate. This gives the opportunity to explore if a model can be trained to detect two loss-of-control events. First, the performance on the yaw validation run will be compared for a model trained on yaw runs only and a model trained on both yaw and pitch runs. The resulting RMSE values for both are depicted in Table B.3.

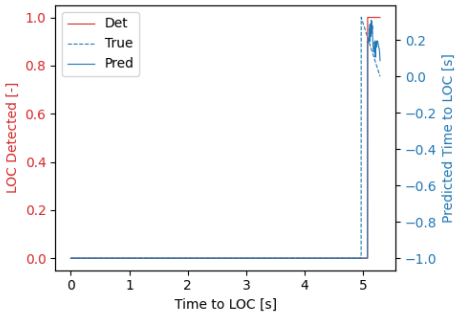
Table B.3: RMSE values in seconds on the yaw validation run for a network trained on yaw runs only and trained on both yaw and pitch runs

Network Type	Only Yaw	Yaw and Pitch
LSTM	0.389	0.429
BiLSTM	0.380	0.394
CNN-LSTM	0.393	0.465
GRU	0.379	0.449

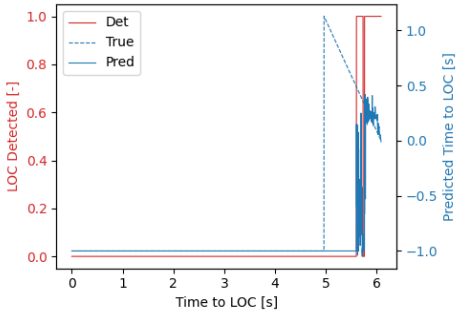
For all models, the one trained on yaw only performs better. The prediction performance is thus influenced negatively by the additional data from the pitch runs. However, to draw conclusions about the prediction performance, prediction plots should be assessed. The prediction on the yaw validation run, pitch validation run and pitch generalisation run are shown in Figure B.3a, Figure B.3b and Figure B.3c respectively for the GRU model.



(a) Prediction on yaw validation run, 0.449s



(b) Prediction on pitch validation run, 0.184s



(c) Prediction on pitch generalisation run, 0.618s

Figure B.3: Performance of the GRU network trained on both pitch and yaw runs, including RMSE values in seconds

The RMSE value is higher on the yaw validation run when trained on both yaw and pitch runs than trained on yaw only, but the detection is still correct and the prediction behaviour is identical to the one observed in the thesis paper. As a result, this model is still suitable for LOC detection and prediction on runs similar to the ones from the training set.

Beside this, for the pitch validation run, the network detects the dangerous manoeuvre correctly. The short time between the start of the manoeuvre and loss-of-control makes it hard to draw conclusions about the prediction performance. There is a downward trend visible, however it is overestimating the true time to LOC. The output of the prediction branch can therefore not be used, as for such a short time to LOC an overestimation will be fatal. On the other hand, for short manoeuvres between 0 and 1 seconds, a detection is already sufficient to serve as a warning that a LOC event is almost starting

and action must be taken, which means that this model can be used in an on-board loss-of-control prevention system.

Finally, the most interesting plot is the last one. Models trained on yaw runs only cannot detect a dangerous manoeuvre leading to loss-of-control due to high pitch rate. When trained on both runs, this is different. A dangerous manoeuvre is detected, although too late and including a short false negative detection. This can be explained by looking at one of the findings of the research. As concluded in the paper, the networks can detect and predict loss-of-control correctly for scenarios that are similar to the ones from the training set in terms of LOC event and time to loss-of-control. The average time to loss-of-control in the training set of pitch runs is 0.470s. The detection of the dangerous manoeuvre happens approximately half a second before loss-of-control and after the short false negative detection, the prediction matches the true time to LOC correctly, meaning that it is overfitting the training set.

Although the output of the models are overfitting the training set, it becomes clear that a model trained on two scenarios can correctly predict LOC for both. This is a promising result, as it means that not every LOC scenario requires its own model.

Prediction bias

The results from the presented research show a bias in the predicted time to loss-of-control to the average time to LOC of the training set, which is 2 seconds. However, this does not mean that the created models always predict a time to loss-of-control of 2 seconds. A first reason for this is that the detection and prediction branch are trained separately and the output of the prediction branch is thus not influenced by the output of the detection branch. The prediction branch does not know when a dangerous manoeuvre is detected and can therefore not associate the start of this manoeuvre with a time to LOC of 2 seconds.

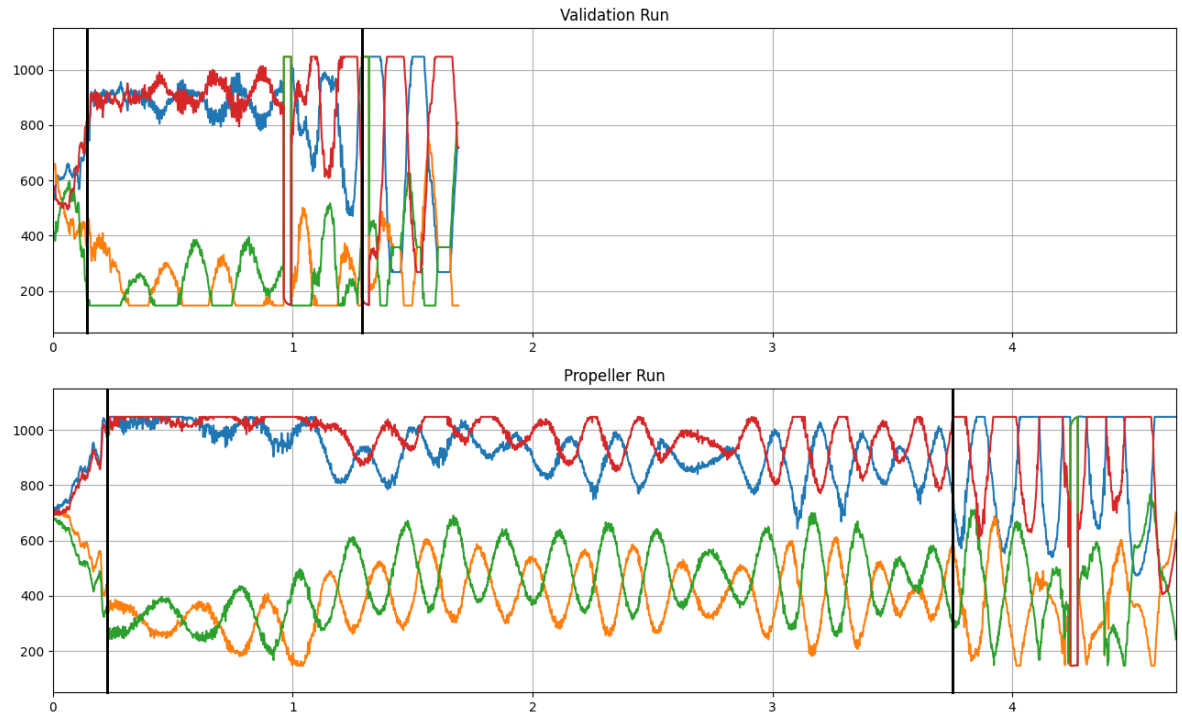


Figure B.4: Rotor commands during the dangerous manoeuvre for the validation and propeller run

However, to confirm that the networks actually learned to associate specific states with a specific time to loss-of-control, it is useful to compare the sensor data and predicted time for two runs: the validation run and a run with different propeller. During the validation run, loss-of-control occurred 1.7s after start of the dangerous manoeuvre, whereas for the propeller run this was 4.721s. The rotor commands

during this manoeuvre are shown in Figure B.4 for both runs.

Two parts can be distinguished in these runs. After an initial rise, the first part shows oscillating rotor commands, which lasts about 0.8s for the validation run and 3.5s for the propeller run. The second part shows increasing amplitude of these oscillations where rotors are saturated continuously. For the validation run this lasts around 0.7s, whereas for the propeller run this lasts about 1.0s. From this it can be observed that especially the first part is stretched for the propeller run compared to the validation run. The rotor commands oscillate for a longer period of time before the quadcopter reaches the next part where the oscillations lead to saturation. To see how the networks deal with this, the prediction of the GRU model trained on rotor commands and accelerations is visualised in Figure B.5.

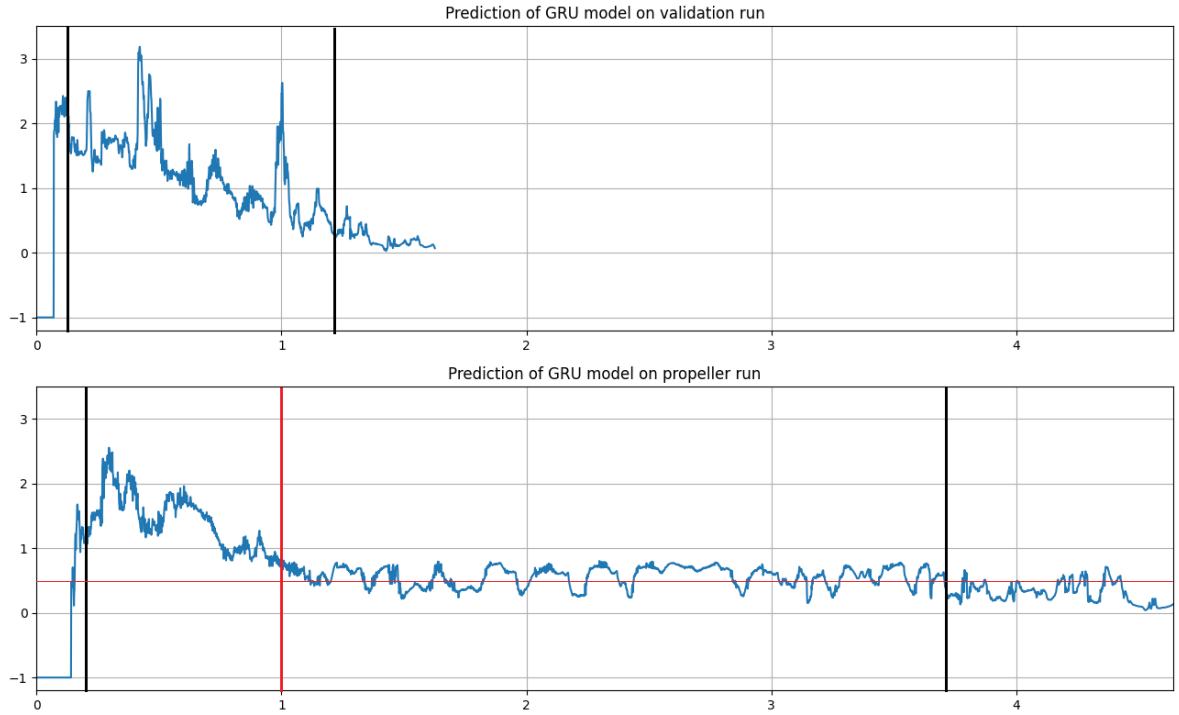


Figure B.5: Predicted time to loss-of-control during the dangerous manoeuvre for the validation and propeller run

The prediction for both runs are similar. Once a dangerous manoeuvre is detected, the prediction shows a downward trend. For the validation run, this continues until LOC starts. However, for the propeller run, the prediction shows an oscillation around a predicted time of 0.5s, indicated by the horizontal red line, starting 1 second after the beginning of the manoeuvre, indicated by the vertical red line, until the start of the second part. This is as expected when comparing both the rotor commands in Figure B.4 and the predicted time in Figure B.5. For the validation run, the rotor command oscillations, especially those with larger amplitude after 1 second and before the start of the second part, are associated with a predicted time to LOC between 0 and 1 second. As these oscillations last longer in the propeller run, it is expected that the predicted time to LOC will oscillate between 0 and 1 second during this part. This proves that the networks have learned to associate specific states with specific time to LOC instead of always predicting 2 seconds.

References

[1] Matthias Baert. 'Quadrotor Upset Recovery after Rotor Failure'. MA thesis. The Netherlands: Delft University of Technology, 2019.

[2] Oguz Bektas, Jane Marshall and Jeffrey A Jones. 'Comparison of Computational Prognostic Methods for Complex Systems Under Dynamic Regimes: A Review of Perspectives'. In: *Archives of Computational Methods in Engineering* 27 (2020), pp. 999–1011. DOI: 10.1007/s11831-019-09339-7.

[3] Christine M Belcastro and Steven R Jacobson. 'Future Integrated Systems Concept for Preventing Aircraft Loss-of-Control Accidents'. In: *AIAA Guidance, Navigation, and Control Conference*. 2010. DOI: 10.2514/6.2010-8142.

[4] Christine M Belcastro et al. 'Aircraft Loss of Control Problem Analysis and Research Toward a Holistic Solution'. In: *Journal of Guidance, Control, and Dynamics* 40.4 (2017), pp. 733–775. DOI: 10.2514/1.G002815.

[5] Christine M Belcastro et al. 'Aircraft Loss of Control: Problem Analysis for the Development and Validation of Technology Solutions'. In: *AIAA Guidance, Navigation, and Control Conference*. 2016. DOI: 10.2514/6.2016-0092.

[6] Christine M Belcastro et al. 'Hazards Identification and Analysis for Unmanned Aircraft System Operations'. In: *17th AIAA Aviation Technology, Integration, and Operations Conference*. 2017. DOI: 10.2514/6.2017-3269.

[7] Boeing Commercial Airplanes. *Statistical Summary of Commercial Jet Airplane Accidents, Worldwide Operations | 1959 – 2019*. Tech. rep. Seattle, Washington 98124-2207: Boeing, Dec. 2020. URL: http://www.boeing.com/news/techissues/pdf/statsum.pdf.

[8] Michael A Bromfield and Steven J Landry. 'Loss of Control In Flight-time to re-define?' In: *AIAA Aviation 2019 Forum*. 2019. DOI: 10.2514/6.2019-3612.

[9] Murat Bronz et al. 'Real-time Fault Detection on Small Fixed-Wing UAVs Using Machine Learning'. In: *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. 2020, pp. 1–10. ISBN: 9781728198255. DOI: 10.1109/DASC50938.2020.9256800.

[10] Zhenghua Chen et al. 'Machine Remaining Useful Life Prediction via an Attention-Based Deep Learning Approach'. In: *IEEE Transactions on Industrial Electronics* 68.3 (2021), pp. 2521–2531. ISSN: 15579948. DOI: 10.1109/TIE.2020.2972443.

[11] Kyunghyun Cho et al. 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[12] June Chongvisal et al. 'Loss-of-Control Prediction and Prevention for NASA's Transport Class Model'. In: *AIAA Guidance, Navigation, and Control Conference*. 2014. DOI: 10.2514/6.2014-0784.

[13] Pedro F.A. Di Donato et al. 'Envelope-aware flight management for loss of control prevention given rudder jam'. In: *Journal of Guidance, Control, and Dynamics* 40.4 (2017), pp. 1027–1041. ISSN: 15333884. DOI: 10.2514/1.G000252.

[14] Olga Fink et al. 'Potential, challenges and future directions for deep learning in prognostics and health management applications'. In: *Engineering Applications of Artificial Intelligence* 92 (June 2020), p. 103678. ISSN: 09521976. DOI: 10.1016/j.engappai.2020.103678.

[15] Forecasts and Performance Analysis Division. *FAA Aerospace Forecast Fiscal Years 2020 - 2040*. Tech. rep. Federal Aviation Administration, 2020. URL: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2020-40_FAA_Aerospace_Forecast.pdf.

[16] G.J.J. Ruijgrok. *Elements of airplane performance*. Delft University Press, 1996. ISBN: 9062756085.

[17] Felix A. Gers, Jürgen Schmidhuber and Fred Cummins. 'Learning to forget: Continual prediction with LSTM'. In: *Neural Computation* 12.10 (2000), pp. 2451–2471. ISSN: 08997667. DOI: 10.1162/089976600300015015.

[18] Ian Goodfellow et al. 'Generative Adversarial Networks'. In: *Advances in Neural Information Processing Systems* 3 (June 2014). DOI: 10.1145/3422622.

[19] Jarosław Gośliński, Wojciech Giernacki and Andrzej Królikowski. 'A Nonlinear Filter for Efficient Attitude Estimation of Unmanned Aerial Vehicle (UAV)'. In: *Journal of Intelligent & Robotic Systems* 95.3-4 (2019), pp. 1079–1095. DOI: 10.1007/s10846-018-0949-7.

[20] Agus Hasan and Tor Arne Johansen. 'Model-Based Actuator Fault Diagnosis in Multirotor UAVs'. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 1017–1024. ISBN: 9781538613535. DOI: 10.1109/ICUAS.2018.8453420.

[21] Cheng Geng Huang, Hong Zhong Huang and Yan Feng Li. 'A Bidirectional LSTM Prognostics Method Under Multiple Operational Conditions'. In: *IEEE Transactions on Industrial Electronics* 66.11 (2019), pp. 8792–8802. ISSN: 15579948. DOI: 10.1109/TIE.2019.2891463.

[22] Guokun Lai et al. 'Modeling Long-and Short-Term Temporal Patterns with Deep Neural Networks'. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. Association for Computing Machinery, 2018, pp. 95–104. DOI: 10.1145/3209978.3210006.

[23] A A Lambregts et al. 'Airplane Upsets: Old Problem, New Issues'. In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*. 2008. DOI: 10.2514/6.2008-6867.

[24] Jialin Li, Xueyi Li and David He. 'A Directed Acyclic Graph Network Combined With CNN and LSTM for Remaining Useful Life Prediction'. In: *IEEE Access* 7 (2019), pp. 75464–75475. DOI: 10.1109/ACCESS.2019.2919566.

[25] Thomas Lombaerts et al. 'On-Line Safe Flight Envelope Determination for Impaired Aircraft'. In: *Advances in Aerospace Guidance, Navigation and Control*. Springer International Publishing, 2015, pp. 263–282. DOI: 10.1007/978-3-319-17518-8_16.

[26] Seema Mallavalli and Afef Fekih. 'A fault tolerant tracking control for a quadrotor UAV subject to simultaneous actuator faults and exogenous disturbances'. In: *International Journal of Control* 93.3 (2020), pp. 655–668. ISSN: 1366-5820. DOI: 10.1080/00207179.2018.1484173.

[27] Sina Sharif Mansouri et al. 'Remaining Useful Battery Life Prediction for UAVs based on Machine Learning'. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 4727–4732. ISSN: 24058963. DOI: 10.1016/j.ifacol.2017.08.863.

[28] Anush Manukyan et al. 'Real time degradation identification of UAV using machine learning techniques'. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017, pp. 1223–1230. ISBN: 9781509044948. DOI: 10.1109/ICUAS.2017.7991445.

[29] Anthony van der Pluijm. 'Early Warning Signals for Loss of Control Prediction of a Damaged Quadcopter'. MA thesis. The Netherlands: Delft University of Technology, 2020.

[30] Behnoush Rezaeianjouybari and Yi Shang. 'Deep learning for prognostics and health management: State of the art, challenges, and opportunities'. In: *Measurement* 163 (Oct. 2020), p. 107929. ISSN: 02632241. DOI: 10.1016/j.measurement.2020.107929.

[31] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. In: *Nature* 323 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[32] Paul Russell and Jay Pardee. *Joint Safety Analysis Team- CAST Approved Final Report Loss of Control JSAT Results and Analysis*. Tech. rep. Federal Aviation Administration: Commercial Aviation Safety Team, Dec. 2000. URL: https://www.cast-safety.org/pdf/jsat_loss-control.pdf.

[33] Vidyasagar Sadhu, Saman Zonouz and Dario Pompili. 'On-board Deep-learning-based Unmanned Aerial Vehicle Fault Cause Detection and Identification'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 5255–5261. DOI: 10.1109/ICRA40945.2020.9197071.

[34] S. Sun and C. C. de Visser. ‘Quadrotor Safe Flight Envelope Prediction in the High-Speed Regime: A Monte-Carlo Approach’. In: *AIAA Scitech 2019 Forum*. 2019. ISBN: 9781624105784. DOI: 10.2514/6.2019-0948.

[35] Sihao Sun, Coen C De Visser and Qiping Chu. ‘Quadrotor Gray-Box Model Identification from High-Speed Flight Data’. In: *Journal of Aircraft* 56.2 (2019), pp. 645–661.

[36] E.R. Van Oort. ‘Adaptive Backstepping Control and Safety Analysis for Modern Fighter Aircraft’. PhD thesis. Delft University of Technology, 2011. ISBN: 9789085707356.

[37] Jiujian Wang et al. ‘Remaining Useful Life Estimation in Prognostics Using Deep Bidirectional LSTM Neural Network’. In: *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*. IEEE, 2019, pp. 1037–1042. ISBN: 9781538653791. DOI: 10.1109/PHM-Chongqing.2018.00184.

[38] James E Wilborn and John V Foster. ‘Defining Commercial Transport Loss-of-Control: A Quantitative Approach’. In: *AIAA Atmospheric Flight Mechanics Conference and Exhibit*. 2004. DOI: 10.2514/6.2004-4811.

[39] Zhaoyi Xu, Joseph Homer Saleh and J H Saleh. ‘Machine Learning for Reliability Engineering and Safety Applications: Review of Current Status and Future Opportunities’. In: *Reliability Engineering and System Safety* 211 (2021), p. 107530. DOI: https://doi.org/10.1016/j.ress.2021.107530.

[40] Liangwei Zhang et al. ‘Review on Deep Learning Applications in Prognostics and Health Management’. In: *IEEE Access* 7.1 (2019), pp. 162415–162438. DOI: 10.1109/ACCESS.2019.2950985.

[41] Ye Zhang et al. ‘Database-Driven Safe Flight-Envelope Protection for Impaired Aircraft’. In: *Journal of Aerospace Information Systems* 18.1 (2020), pp. 14–25. DOI: 10.2514/1.I010846.

[42] Youmin Zhang and Jin Jiang. ‘Bibliographical review on reconfigurable fault-tolerant control systems’. In: *Annual Reviews in Control* 32.2 (Dec. 2008), pp. 229–252. ISSN: 13675788. DOI: 10.1016/j.arcontrol.2008.03.008.

[43] Rui Zhao et al. ‘Deep learning and its applications to machine health monitoring’. In: *Mechanical Systems and Signal Processing* 115 (Jan. 2019), pp. 213–237. ISSN: 10961216. DOI: 10.1016/j.ymssp.2018.05.050.

[44] Rui Zhao et al. ‘Machine Health Monitoring Using Local Feature-Based Gated Recurrent Unit Networks’. In: *IEEE Transactions on Industrial Electronics* 65.2 (2018), pp. 1539–1548. DOI: 10.1109/TIE.2017.2733438.