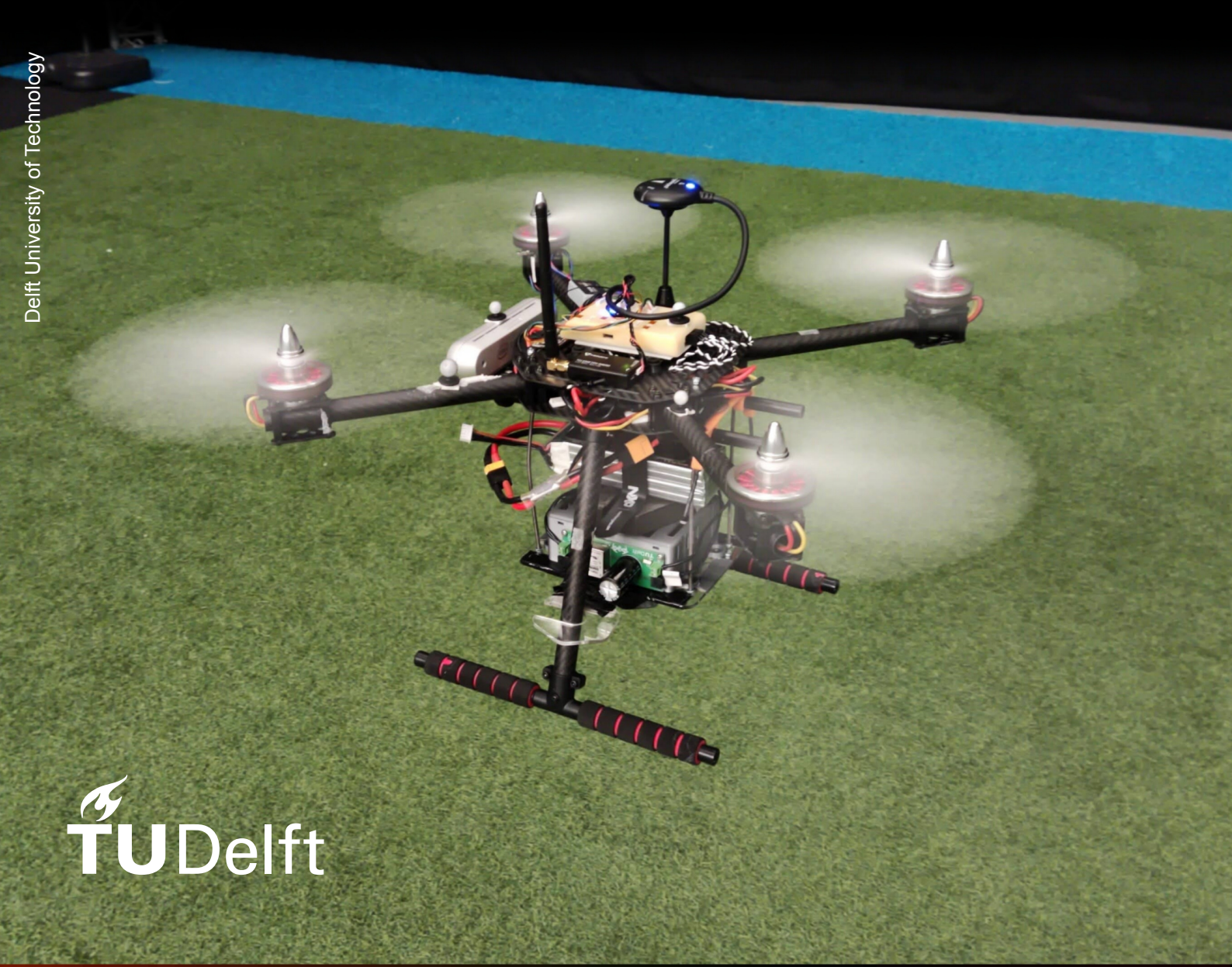
Master Thesis

Improving the PX4 software-in-the-loop multirotor UAV simulator accuracy

Master Thesis Robotics

Michiel te Braake



Delft University of Technology

TU Delft

Master Thesis

Improving the PX4 software-in-the-loop multirotor UAV simulator accuracy

by

Michiel te Braake

Student Name	Student Number
Michiel te Braake	4742486

TU Delft

Abstract

Recent developments have enabled the mass production of cheap, high-performance multirotors. As a result of this, the multirotor has found a large variety of different uses. Testing new algorithms using real-life flights is costly in terms of time and potentially in terms of materials in the case of a crash. Simulators have quickly gained prominence as a tool in algorithm development. The accuracy of these simulators is vital to ensure the step from simulated flights to real-life testing is as small as possible. Current simulator models are held back due to a lack of properly identified multirotor coefficients. Often, the coefficients from a different multirotor are used, even if the new multirotor is vastly different in shape and size. Additionally, most researchers do not have a sufficient overview of which parts of the simulator are most important in providing an accurate simulator environment.

This research will give an overview of the different parts that influence the simulator results. The simulation of the motor controllers and rotor aerodynamics is identified as a major contributor to inaccuracies. The coefficients used in these two components will be found by performing a system identification process of the NXP HoverGames quadcopter. This identification process entails determining the mass, dimensions, inertia, motor response and thrust, torque, rotor drag and rolling moment coefficients. Experiments will be performed to find the correct coefficient values and to evaluate the impact that each effect has on the simulator's performance. A focus is put on ease of repeatability, such that other researchers can use the same process for their specific multirotor.

The theoretical background behind each effect in the motor controller and rotor aerodynamics models is discussed. This theoretical knowledge is used to perform experiments to find the coefficients for each effect and validate them. As a result, new coefficients for the motor controller simulator, thrust, torque and rotor drag were found. Additionally, this process showed that the rolling moment is much smaller than the errors currently existing in the model and this effect has, therefore, been ignored. The motor controller simulation is identified as the largest cause of inaccuracies in the current model. This simulation maps the motor commands to the output rotational velocity of the rotor and simulates the time response of this system. The formulas currently used for this do not provide a sufficient model of real-life behaviour. An analysis is done to gain a better understanding of what variables influence motor behaviour.

Finally, the new coefficients are put to the test in a variety of different trajectories to compare the trajectory tracking performance with the old model. This comparison consists of two parts. In the first part, a qualitative analysis is used to understand the difference in simulator behaviour. In the second part, a set of metrics are defined to quantify the difference in accuracy. This comparison process is used to prove that the new model provides significant improvements compared to the old coefficients, especially in the case of faster, more aggressive manoeuvres.

Contents

Acronyms

BLDC brushless DC 3

CAD computer-aided design 10, 11
CoG centre of gravity 12, 68

DAQ data acquisition 74, 75
DoF Degrees of freedom 77

EKF extended Kalman filter 11
ESC electronic speed control 3, 23

FFT fast Fourier transform 12
FMU Flight Management Unit 4

GPS Global Positioning System 3

HITL hardware-in-the-loop 1

IMU inertial measurement unit 11, 51, 52

MAV Micro Air Vehicle 6
MITL model-in-the-loop 1
MoI moment of inertia 10, 12–14, 65, 68, 73

ODE Open Dynamics Engine 6

PID proportional–integral–derivative 52
PWM pulse-width modulation 61

RC Radio control 72, 75, 76
RMSE root-mean-square error 20, 54, 60, 80
ROS Robot Operating System 76, 78, 80, 81
RPM revolutions per minute 61, 73

SITL software-in-the-loop 1

UART universal asynchronous receiver-transmitter 76
UAV unmanned aerial vehicle 1
UKF unscented Kalman filter 11
uORB micro Object Request Broker 75

List of symbols

Symbol	Definition
ϵ_{z_B}	unit vector pointing in the z direction of the body frame
ϵ	turning direction of the rotor
λ	inflow ratio
μ	advance ratio
Ω	rotor angular velocity
$\mathbf{\Omega}_{sp}$	acceleration setpoint vector
$\overline{C_d}$	drag coefficient of rotor section at the 70% radial station
ρ	air density
σ	rotor solidity ratio
θ_0	angle of incidence
θ_{tw}	twist pitch
a	lift curve slope
A	propeller disk area
$\mathbf{A}_{sp}$	acceleration setpoint vector
$\mathbf{A}_{des}$	desired acceleration vector
C_{R_M}	rolling moment coefficient
C_H	rotor drag coefficient
C_Q	torque coefficient
C_T	thrust coefficient
D	rotor diameter
F_D	drag force
F_T	thrust force
H	rotor drag
M_Q	drag moment
M_R	rolling moment
n	rotor rpm
P_{sp}	pitch setpoint
Q	torque about the rotor shaft
R	rotor radius
R_{sp}	roll setpoint
R_M	rolling moment
T	thrust
T_{sp}	thrust setpoint
$\mathbf{T}_{sp}$	thrust setpoint vector
v	rotor velocity vector
$v_A^{\perp}$	projection of the rotor velocity vector onto the rotor plane
$\mathbf{V}_{sp}$	velocity setpoint vector
X_{sp}	thrust setpoint
$\mathbf{X}_{sp}$	position setpoint vector
z	height above ground

List of Figures

1 Introduction

A multirotor unmanned aerial vehicle (UAV), also called a drone, is an unmanned helicopter with multiple, often four, rotors. Compared to helicopters, their mechanical design is far simpler. The entire system consists of merely one moving part per rotor, the motor. In the past, the sensors, motors and battery were too large, heavy and expensive to construct a commercially viable multirotor. Advancements in technology have drastically improved all of these components, which has led to a massive increase in the research into multirotors. Innovation has gained momentum ever since. In the early 2000s, multirotors were often not much more than remote-controlled toys, but by 2010 they had already evolved into capable camera drones. Since then, multirotors have seen usage in many applications, such as in the production of movies, in the coverage of live events, as a means of crowd control by police, and even in the advent of a new sport: drone racing. Innovation has increasingly moved towards autonomous flying, with the goal of enabling robust, safe flight in any environment.

Multirotors are inherently unstable systems that require constant corrections to maintain stable flight. Having a model that can predict the response of the system is therefore necessary. The theory of flight dynamics has been adapted from existing helicopter knowledge, with works such as that by Fay, 2001 laying the groundwork. Over the years, the control algorithms have been improved to take more aerodynamic effects into account. The more basic models directly relate rotor speeds to thrust and torque, while improved models, such as the one developed by Faessler et al., 2018, take flight speeds into account as well.

Accurate models also make it possible to simulate multirotor flight. This saves time in running experiments, makes the experiments easier to repeat, and avoids damaging flight hardware in the case of a malfunctioning algorithm. Shokry and Hinchey, 2009 made an overview of verification techniques that can be used for embedded software development. Important methods here are model-in-the-loop (MITL), software-in-the-loop (SITL) and hardware-in-the-loop (HITL). Using a more realistic verification system can provide better results. For example, Silano et al., 2019 show that a SITL simulation can be used to detect instabilities that did not show up in a simpler Matlab/Simulink simulation. It shows that a more accurate environment can be used for tuning or training a controller, whereas a basic simulation is only sufficient for testing a planning algorithm.

Simulator accuracy requirements depend on the specific application, as the allowed margin of error will not always be the same. If the flight is performed outdoors, away from obstacles, then the tracking error may be quite large without causing issues. In other applications, such as indoor flight, drone swarms, or high-velocity flight, the allowed margin of error is much smaller. As the usage of multirotors in such environments is increasing, the accuracy requirements of multirotor simulators are also increasing.

While the need for accurate simulators is increasing, the research into this area lags behind. The models used in the simulators are often only validated once and then never updated to represent different or newer types of multirotors. While there is a lot of research into the modelling of aerodynamic effects on multirotors, most of this research only verifies their models in real-life wind tunnel experiments. These results, therefore, can not immediately be used to improve a simulator model. Using a model not updated for the specific multirotor introduces inaccuracies in the simulator. Little is known about how

these inaccuracies influence the simulator results and in which scenarios this could cause a problem. A broader picture of how the different simulator components influence the final result is missing.

1.1. Research Objectives

In this research, the NXP HoverGames quadcopter drone is used as a research platform. For simplicity, this research will talk specifically about quadcopters, but the same principles apply to other types of multirotors. To be used effectively as a research platform, it is essential to have an accurate simulation environment where algorithms can be tested and trained. The values identified in this research will only be valid for this specific quadcopter, but the same methods can be applied to perform the identification on other types. The main goal of the thesis is to answer the following research question:

How can the PX4 Gazebo Plugin Suite[1] *simulator be improved to resemble real drone flight behaviour as much as possible?*

To answer this research question, several subquestions have been defined:

What are the differences between the simulator and real life?
Several different components interact with each other during a flight. A comparison between real-life and simulator flight needs to be made to find the differences between the two setups. This overview will be used to choose the parts that are most likely to introduce inaccuracies and that therefore need to be focussed on. Additionally, this process will show the largest simplifications that had to be made to simulate quadcopter flight. Chapter 2 will give an overview of the quadcopter platform and explain the exact research focus of this thesis.

What coefficients need to be identified?
The formulas used in the simulator model all contain coefficients that are specific to the simulated quadcopter. An analysis needs to be done to find the importance of each coefficient. Current literature using the Gazebo framework has copied all coefficient values from the paper that first identified these coefficients, even if the used drone has very different properties. The importance of each coefficient on flight behaviour needs to be researched. Chapter 2 will explain the aerodynamic effects that influence the quadcopter and describe the coefficients used to model these effects.

How can these coefficients be identified?
Each coefficient has several possible methods that can be used to identify it. A combination of existing research and new experiments will be used to find the identification methods most applicable to the goals of this thesis. Chapter 3 till Chapter 9 will describe the methods used to identify and validate the different coefficients. A more detailed description of the structure of these chapters will be given in Chapter 2.

How can the simulator model be validated?
To prove that the newly identified model is more accurate, a comparison will have to be made between the old and new model. A selection of trajectories needs to be made that demonstrates a variety of flight patterns which the quadcopter may encounter. Additionally, several metrics need to be defined to judge the performance of the models. Chapter 10 will describe the methodology and results of the validation.

[1] https://github.com/PX4/PX4-SITL_gazebo

2

Research methodology

To reduce the discrepancies between real-life flight and simulations, it is vital to have a good understanding of where the differences might come from. The goal of this chapter is to analyse the potential causes and give an explanation of why this research will only focus on certain parts of the simulator. To accomplish this, Section 2.1 will first give an overview of the different parts of the quadcopter that interact with each other and relate these parts to the different modules in the simulator. Section 2.2 will then discuss these different simulator parts and explain which ones will be focused on. The most important parts will be described in more detail in Section 2.3. Finally, Section 2.4 will discuss how the identification experiments will be chosen.

2.1. Architecture overview

This section will give a general overview of the components involved in making a quadcopter fly and will highlight the parts that are replaced to perform a simulation. A short discussion on the differences will be given, and a prediction will be made about which parts are likely the biggest cause of simulator inaccuracies. The focus of the thesis will be on those parts. This section will provide a global overview of how the different software and hardware components of the quadcopter interact to achieve flight. The same overview will then be given for the simulation such that a comparison between the two setups can be made.

This overview is a simplified version of the full system. Therefore, a more detailed description of the setup used in the experiments will be given in Section B.1, and a more detailed description of the simulator will be given in Section B.2.

2.1.1. NXP HoverGames quadcopter

In this research, the NXP HoverGames drone kit (KIT-HGDRONEK66) will be used, from hereon called the HoverGames drone. The HoverGames drone kit is a professional drone development kit that includes a flight management unit (RDDRONE-FMUK66), four brushless DC (BLDC) motors powered by electronic speed control (ESC) motor controllers, a telemetry radio, and a range of sensors including an accelerometer, magnetometer, gyroscope, digital pressure sensor and Global Positioning System (GPS). The frame is specifically designed in a way that makes it easy to attach further electronics, such as a companion computer for data processing and extra sensors. Figure 2.1 shows the base kit. This development kit has already been used extensively in previous research due to its versatility. Dobrea and Dobrea, 2020 is an example of research using this development kit, in which an extra Jetson Nano companion computer and a camera were attached to perform autonomous monitoring and human detection tasks. This versatility does have a downside in the form of relatively large dimensions and weight. The HoverGames drone is approximately 400x400 mm, and depending on the number of extra electronics attached, its weight is about 2 kg. This is much larger than many alternative quadcopters, such as the Parrot Bebop 2 drone used for system identification by Li, 2014, which weighs merely 420 g. It is important to keep in mind that the drone platform used in this thesis will often not be able to reach the acceleration, speed or aggressive flight patterns that other smaller or racing-inspired drones can reach.

Figure 2.1: NXP HoverGames drone (Semiconductors, 2018).

2.1.2. Quadcopter components overview

The heart of the quadcopter is the Flight Management Unit (FMU) running the PX4 autopilot. This is a flight controller that performs the control and stabilization of the quadcopter. To accomplish this, it receives data from various sensors, like an accelerometer, gyroscope, magnetometer and barometer. The flight controller processes the received data to calculate the motor commands required to maintain a stable flight. The caluclated commands are then sent to the motors that spin the rotors.

For an outdoor flight, these components would be enough, as GPS can then be used to get position data. When flying indoors, the GPS signal is unreliable, and additionally, the need for precise position data increases as the available space is limited. An external motion capture system is used to obtain real-time high-accuracy knowledge of the position of the quadcopter. This data is sent over Wi-Fi to a companion computer mounted onto the quadcopter, which relays the data to the flight controller. The flight controller does not have the wireless connectivity to receive large amounts of data, which is why the companion computer is required.

The motors do not have a direct connection to the sensors, but they do have an indirect influence on them. The motors spin the rotors, which produce a thrust that moves the quadcopter and as a result changes the sensor readings. The parts are connected through physics, which will need to be implemented as a new component in the simulator. A diagram of the parts can be seen in Figure 2.2.

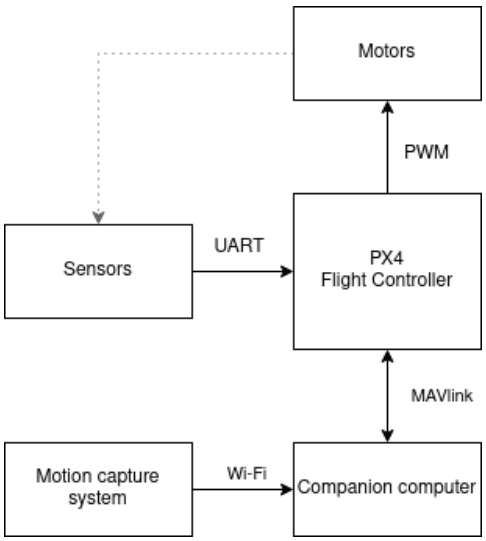


Figure 2.2: Overview of the quadcopter control architecture.

2.1.3. Simulator components overview

The PX4 autopilot has been designed with simulation support in mind. It provides a version of the firmware in which only a few modules are replaced, and the rest of the firmware is exactly the same as the one used in flight. These modules are the ones that receive messages from sensors, or send

commands to the motors. Note that these replaced modules purely perform the communication, and all further processing of the data will be the same. A diagram of the simulator parts can be seen in Figure 2.3.

The diagram of the simulator parts, therefore, shares similarities with the diagram for a real-life flight. Physical components, such as motors and sensors, are replaced with modules that simulate their functionality. The connection between these two parts is now a clear link via a motor model and the Gazebo physics engine. The motor controller simulator converts PX4s motor commands into a rotor velocity. This conversion takes both the expected steady-state RPM as well as the transient response into account to simulate the rotor velocity. The simulated rotor velocity is then used in the motor model to calculate the forces and torques that the rotor generates. The Gazebo physics engine processes these forces to calculate an updated quadcopter state. The simulated sensors read this new quadcopter state and transform it into sensor messages that are sent to PX4. The used simulator modules are from the PX4 SITL_gazebo package, which is a version of the RotorS simulator with added PX4 support (Furrer et al., 2016).

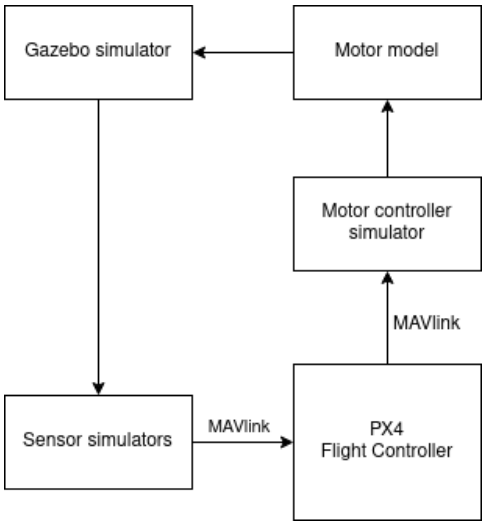


Figure 2.3: Overview of simulator architecture.

It is worth noting that while Gazebo is a powerful simulator, by default, it does not implement many physics effects. Only gravity, collisions and friction are supported out of the box. All other effects, such as lift and drag, must be added using plugins. This means that during flight, all simulated forces either come from gravity or from the motor model.

2.2. Research focus

As seen from the overview, several distinct parts may introduce inaccuracies. Each part will be described in a bit more detail to get a better understanding of how they function. After that, the specific focus of this research will be explained.

Sensor simulators

The sensor simulators read the simulator world state to create the sensor messages for PX4. The behaviour of real sensors is simulated as accurately as possible, which is done by adding noise, drift and bias into the data. The formulas that introduce these effects can be tuned using a set of configurable parameters.

Motion capture system

The motion capture system is simulated through a plugin that reads the quadcopter position in the simulator and creates the relevant vision position estimate message. A notable difference is that in the real-life setup, the motion capture message has to come from an external system. This introduces a small delay between the capture and receival of the motion capture data. The motion capture plugin does not simulate such a delay.

Motor controller simulator
The motor simulator converts motor commands into rotor velocity. It uses a linear relationship to convert from motor output command to rotational velocity of the rotor. Additionally, it models the delayed response when the rotor has to speed up or slow down. The exact behaviour of these two effects can be configured using a set of parameters.

Motor model
The motor model calculates all aerodynamic forces that affect the quadcopter during flight. The used formulas are simplified versions of equations based on aerodynamics theory. Each formula has coefficients that can be configured to better represent the specific quadcopter.

Gazebo
Gazebo uses the Open Dynamics Engine (ODE) physics engine to calculate the effect that applied forces will have on the quadcopter. This physics engine is capable of real-time simulation, emphasising speed and stability over physical accuracy. Several parameters can be set that influence the step size of the simulation and the exact way calculations are done.

Figure 2.4: AscTec Firefly hexacopter, which the default coefficients are based on (https://productz.com/en/asctec-firefly/p/w9DR).

The accuracy of the formulas used in the motor model and motor controller simulator is dependent on the use of the correct coefficients. Currently, many quadcopter models made for this simulator copy the coefficients from the paper that first defined the model (Furrer et al., 2016). Those coefficients are based on the system identification of a hexacopter, which is often not comparable to the quadcopter being modelled. This hexacopter can be seen in Figure 2.4, and it is clearly not comparable to the quadcopter used in this research. The model coefficients strongly depend on the physical characteristics of the quadcopter, such as the size and type of propeller. It is, therefore, likely that the usage of these incorrect coefficients leads to large inaccuracies in the simulated physics. The simulation of the sensors and motion capture system may also be a cause of simulator inaccuracies. Still, it is considered unlikely that these inaccuracies are as significant as the ones introduced by the use of incorrect coefficients.

Due to the reasons outlined above, this research will focus specifically on the motor controller simulator and motor model parts of the simulator. The used coefficients will be identified, and additionally, it will also be checked whether the used formulas are a good enough representation of reality. This analysis will be based on literature and experiments performed in this thesis.

2.3. Motor model

In this research, we will use a Micro Air Vehicle (MAV) simulator model developed by Furrer et al., 2016. The MAV simulator model includes the simulation of the motor controller, as well as the simulation of the aerodynamic effects caused by the rotating rotors. The aerodynamic effects are modelled as two forces and two moments per rotor.

The aerodynamic effects acting on a quadcopter originate from the lift and drag generated by the airfoil. A diagram of the forces acting on an airfoil can be seen in Figure 2.5. The produced lift and

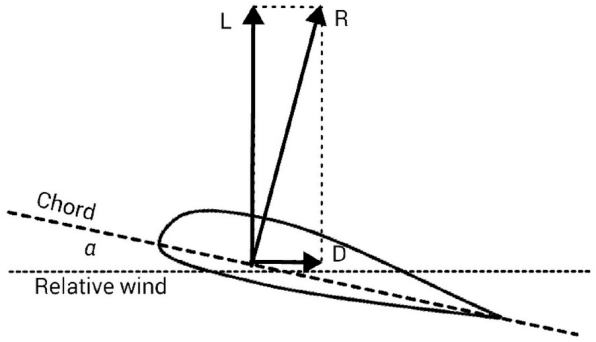


Figure 2.5: Lift and drag forces acting on an airfoil (http://aviation.stackexchange.com/questions/28019).

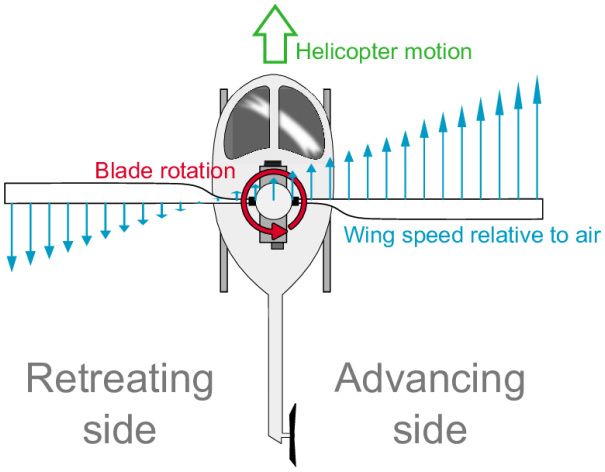


Figure 2.6: Relative wind difference per rotor side when moving (Paternoster et al., 2011).

drag are proportional to the relative wind. Unlike the wings of an airplane, the two sides of a propeller have their relative wind in opposite directions. A diagram of this effect can be seen in Figure 2.6. As a result of the relative wind directions, the two drag forces point in opposite directions, while the lift force of both sides points in the same upward direction. This lift force is called the thrust and is the first force on the quadcopter. Since the two drag vectors point in opposite directions, the resultant force vector is zero. The moment caused by the drag vectors does point in the same direction. This causes a torque around the Z-axis in the opposite direction of the rotor spin. When the quadcopter is not moving and there is no wind, these are the only two aerodynamic effects on the quadcopter. In the case of a moving quadcopter, the forward velocity influences the relative wind experienced by the propeller sections. As seen in Figure 2.6, the side of the propeller moving against the wind will have a higher relative wind, while the other side has a lower relative wind. As a result, the drag vectors on either side of the propeller no longer have the same length. When the two are added, there will now be a resultant force vector pointing in the opposite direction of the velocity vector. This force is called the rotor drag. The lift is impacted similarly, with one side of the propeller producing more lift than the other. This difference causes a moment around the X-axis of the propeller. The name for this moment is the rolling moment.

The four aerodynamic effects on the propeller are shown in Figure 2.7.

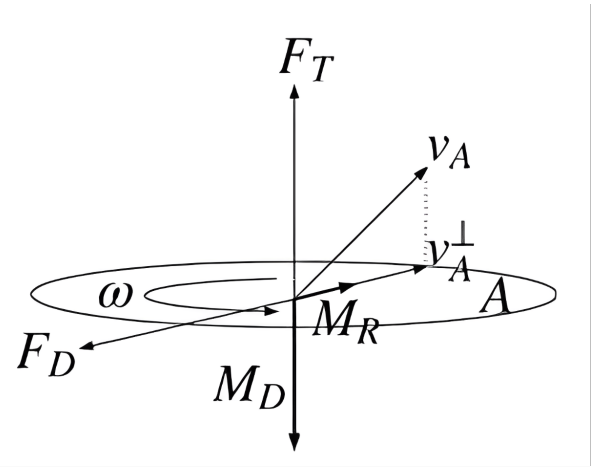


Figure 2.7: Forces and moments acting on the centre of a rotor (Furrer et al., 2016).

The angular velocity of the rotor is shown as ω, the velocity vector as v_A, and the projection of the velocity vector onto the rotor plane is shown as $v_A^\perp$. The thrust is shown as F_T, pointing up along the Z-axis of the rotor. The rotor drag force is shown as F_D, pointing opposite to the projection of the velocity

vector. The torque is shown as M_D, acting along the Z-axis of the propeller. The rolling moment is shown as M_R, acting along the projection of the velocity vector.

The formulas for the aerodynamic effects each have their own coefficient to make them correct for the specific quadcopter that is used. Experiments will be performed to identify these coefficients.

2.3.1. Model shortcomings

The used model contains some omissions and simplifications that may impact the accuracy of the model. This section will list the most important ones which may cause abnormalities in the simulation results.

The model only simulates aerodynamic effects on the rotors and thereby disregards any drag or lift generated by the quadcopter fuselage. Near hover conditions, when the velocity is close to zero, this is a reasonable assumption, but it is not known whether this simplification is also accurate at higher velocities.

Secondly, the aerodynamic effects of propeller wake on the other rotors are ignored. Misiorowski et al., 2019 calculated a 19% reduction in lift from the rear rotors at a horizontal velocity of 10m/s, showing that this has a significant effect at speeds most quadcopter drones can reach. There are no formulas in the model to account for such an effect.

Thirdly, ground or ceiling effects are not incorporated into the model. Due to the short distance from a surface in which these effects are visible, this is of little importance for outside flying. However, for accurate simulations of constrained indoor flying, these effects need to be added.

Ground effect

When flying close to the ground, the air can not flow down freely and is forced outwards to the sides. This changes the induced velocity and therefore increases the rotor thrust and reduces the drag (Matus-Vargas et al., 2021). This effect is only present close to the floor. Powers et al., 2013 has shown that for small quadcopters, the effect can be present up to $z/R = 5$, with a reduction of the RPM required for hover of about 10% close to the floor. A similar increase in thrust occurs near the ceiling. Belatti, 2018 has shown the significant effect that this can have on the trajectory tracking of a quadrotor by adding a tunnel to a trajectory. The variation in thrust when entering and exiting the tunnel would cause the quadcopter to crash unless the flight speed was reduced or controller gains were altered. The ground effect is strongly dependent on the type of floor surface. Aich et al., 2014 has shown that a flat, hard surface will produce the strongest ground effect, with surfaces like grass producing a very minimal ground effect.

2.4. Identification experiments choice

There are generally several different identification methods that can be used to find a specific coefficient. The different options will be compared to find those that are the most applicable. To that end, it is important to clearly define the specific requirements for this thesis.

1. **Accuracy**
   The main goal is to provide a more accurate simulation environment in which the behaviour of the quadcopter closely matches the real-life behaviour. Therefore, the main requirement of an experiment will logically be that it should be able to provide an accurate measurement.
2. **Repeatability**
   The experiments will need to be repeated by other researchers to get the data for their specific quadcopter and setup. Current robotics research often does not take the time to correctly identify the coefficients. The used experiments should preferably be easy to repeat such that other researchers are more likely to do that for their specific quadcopter.
3. **Offline analysis**
   A large amount of research has focused on performing system identification online in flight. This is not a requirement for this thesis. Online identification experiments often have significant downsides, such as reduced accuracy, that are only worth it if online updates of the model parameters are required.

The following chapters will perform the identification of the different coefficients in the model. These chapters will all follow a similar structure that provides context into the meaning of a coefficient before

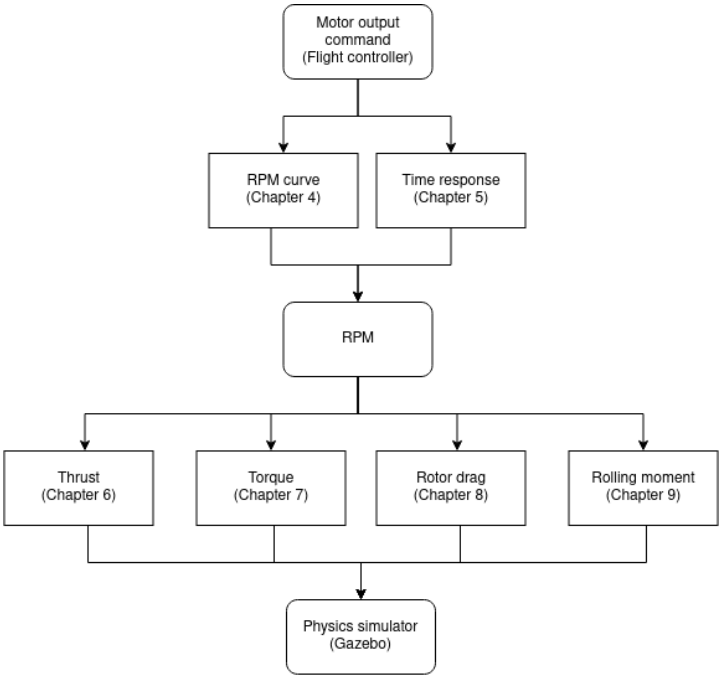


Figure 2.8: Chapter structure of identification process.

identifying and then validating it. The physical parameters, which are the mass, dimensions and inertia, will be identified in Chapter 3. After that, all the coefficients that define the motor controller simulator and the motor model will be identified. Figure 2.8 shows a visual diagram of the chapter structure. First, the coefficients of the motor controller simulator will be identified in Chapter 4 and Chapter 5. After that, the coefficients of the motor model will be identified in Chapter 6, Chapter 7, Chapter 8 and Chapter 9, respectively. The full model with all new coefficients will be validated in Chapter 10. Finally, a conclusion will be given in Chapter 11.

3

Physical parameters

This chapter describes the identification of the mass, dimensions and inertia of the NXP HoverGames quadcopter. As these are well-known parameters that are not specific to quadcopters no extra background theory will be given. Section 3.1 will describe the identification of the mass of the quadcopter, and Section 3.2 will quickly describe the dimensions. Section 3.3 will identify the inertia of the quadcopter and perform a validation experiment for the values that were found.

3.1. Mass

The mass of the drone determines the thrust required to stay in the air, as well as the force required for linear acceleration. The torque needed for an angular acceleration is determined separately by the moment of inertia (MoI) (moment of inertia), which is described in Section 3.3. The mass of the separate parts has been measured using a scale with an accuracy of 1 gram. The most important measurements will be given now, and a complete list of measurements is given in Section A.1. The quadcopter itself, without rotors and battery, weighs 1808 grams. The battery weighs 395 grams, and the rotors weigh 10 grams each. This adds up to a total flight-ready weight of 2243 grams.

3.2. Dimensions

The dimensions of the drone determine where the forces of the motors are applied and additionally determine the collision bounds of the quadcopter. The used simulation model already includes the dimensions of the HoverGames quadcopter. A measuring tape is used to verify the most important dimensions of the quadcopter. The distance from the quadcopter centre to the rotor axis is 25 centimetres. When the rotors are added, this adds up to a quadcopter that is 55 centimetres wide, which is quite large for a quadcopter.

3.3. Inertia

The inertia of the quadcopter influences how quickly it can perform angular accelerations. Several different methods exist to calculate the moment of inertia. First, an overview of these methods will be given. Secondly, an experiment will be performed to identify the inertia. Finally, the found values will be validated.

CAD modelling

The computer-aided design (CAD) software can provide an estimate of the inertia. The quality of this estimate is entirely dependent on how detailed the model is, and creating a highly detailed CAD model with an exact component weight breakdown is very time-intensive (Pegram and Anemaat, 2000). A lumped mass approach can be used to simplify the model. In this approach, the quadcopter is not modelled with each individual part, but instead, nearby components are grouped together as one weight.

Lab measurements
Inertia values can be measured using a bifilar pendulum as explained by Jardin and Mueller, 2007 and Derafa et al., 2006. The setup of the experiment is shown in Figure 3.1. The quadrotor is attached to a frame with two wires and given an initial rotation angle. The quadrotor will then oscillate back and forth while measurements are made by either the onboard inertial measurement unit (IMU) or an external motion capture system. The measured oscillation is fit to the expected oscillation profile to find the inertia.

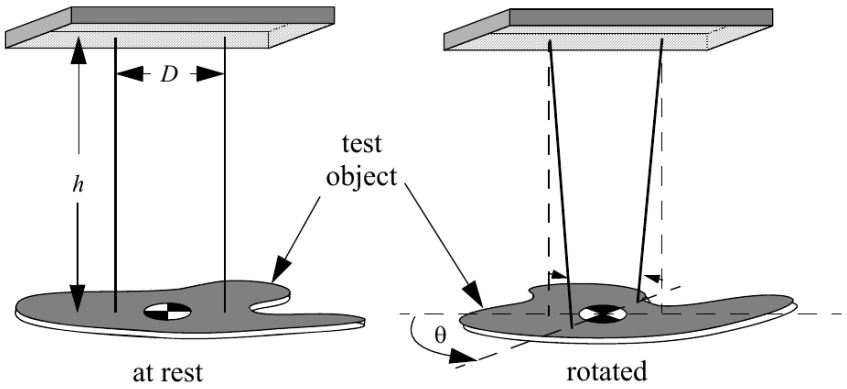


Figure 3.1: Bifilar experiment setup (Benders, 2020).

In-flight estimation
In-flight estimation uses pose measurements during flight to estimate the moment of inertia. Wüest et al., 2019 compared the use of a unscented Kalman filter (UKF), extended Kalman filter (EKF) and CAD model with ground truth data obtained in lab measurements using a bifilar pendulum. These methods of inertia estimation make it possible to perform online parameter estimation. It is shown that this method of in-flight estimation is useful for re-estimating values when they have changed, such as when an object is picked up by the quadcopter. However, the estimates are less accurate than pre-flight measurements with estimates being nearly 20% off of the ground truth data. Burri et al., 2018 focused on offline analysis of flight test data. To accomplish this they used a maximum likelihood parameter estimation system, which makes it possible to calculate the standard deviation of the estimated inertia. Knowing the uncertainty of the estimated parameters is useful, but the calculated inertia values were not more accurate than the online estimated values.

Comparison
The bifilar pendulum is the most accurate, to the point that it is used as the ground truth data to compare to other techniques. Wüest et al., 2019 found an inaccuracy below 10% for the CAD model and an inaccuracy below 20% for the in-flight estimation method. Due to its high accuracy, the bifilar pendulum method will be used to identify the inertia. A simplified CAD model will be used to validate the found inertia values.

3.3.1. Identification
The setup of the experiment is shown in Figure 3.2. The quadrotor is attached to a frame with two wires and given an initial rotation angle. The quadrotor will then oscillate back and forth while angle measurements are recorded using the OptiTrack motion capture system.

In order to measure the inertia of a specific axis, the quadcopter must be mounted such that it is completely level. Additionally, the wires need to be mounted such that the centre of gravity of the quadcopter is directly in between them. A few steps have been undertaken to ensure that these requirements are met. Firstly, the quadcopter does not have mounting points that surround the centre of gravity. Therefore, lightweight bamboo sticks were used to create a frame around the quadcopter that the ropes can be mounted on. This setup is visible in Figure 3.2. Then, the location of the ropes

is determined by finding the location in which the quadcopter hangs level on the ropes. Finally, both wires need to be of the same length. To make it easier to adjust the wire length both wires were looped through a screw hook at the top of the setup and then back down again. This allows for the use of guy line tensioners to adjust the rope length without affecting the parts of the rope that need to move in the experiment. The screw hooks at the top were placed at the same distance as the distance between the ropes at the bottom.

Battery placement
It is important to discuss the influence of the battery on the centre of gravity. The battery is mounted to the quadcopter using a velcro strap on a movable carriage. This means that the position of the battery can differ significantly depending on how it is put in, and as a result affect the centre of gravity (CoG). It was found that the CoG of the quadcopter is near the physical centre of the quadcopter when the battery is moved towards the rear of the quadcopter. This was expected, as there is a camera platform mounted on the front of the drone. With the battery moved towards the rear the weight of the quadcopter is evenly distributed between all rotors, and therefore it was decided to place the battery in this location every time. The location was marked on the battery with a piece of tape to ensure the CoG is in the correct location for all future tests.

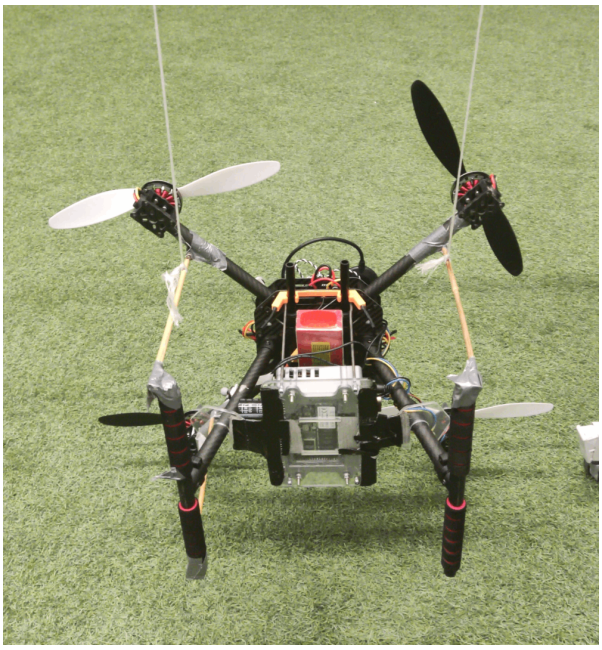


Figure 3.2: Bifilar pendulum frame construction.

Calculation
Krznar et al., 2018 gives an equation that using pendulum equations can determine the inertia from the pendulum period.

$$I = \frac{mgd^2}{16h\pi^2 f^2} \tag{3.1}$$

, where m is the mass, g is the gravitational constant, d is the distance between the wires, h is the length of the wires and f is the oscillation period. The period of the oscillation is extracted from the signal using a fast Fourier transform (FFT).

Table 3.1 gives the MoI values around the three principal axes of the quadrotor, the values found in each separate experiment can be seen in Table A.2.

3.3.2. Validation
A simplified CAD model was constructed to validate the inertia values found in the bifilar pendulum experiment. This CAD model combines certain parts in order to speed up the modelling process. For

Table 3.1: MoI values found using bifilar pendulum experiment.

I_{xx}	2.8E-02
I_{yy}	3.3E-02
I_{zz}	3.8E-02

example, the propeller, motor, and connection to the quadcopter arm are modelled as a single rectangular block. This has a slight influence on the accuracy of the measurement, but it greatly reduces the effort that it takes to create the model.

The CAD model was created in SolidWorks. A mesh model of the HoverGames quadcopter was loaded into SolidWorks for visual reference. Although this mesh model can not be used to calculate the physical properties of the quadcopter, it is useful to use as a guide when reconstructing the model. An image of the model can be seen in Figure 3.3.

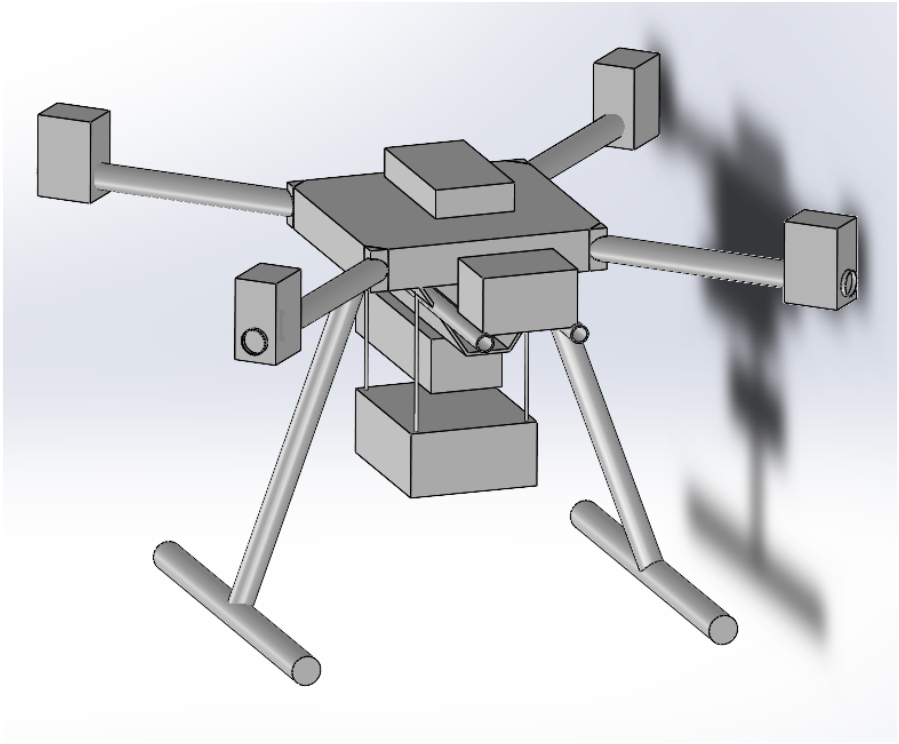


Figure 3.3: CAD model of HoverGames quadcopter.

A downside of SolidWorks for this task is that it is designed to create a full CAD model in which all parts are modelled accurately. A consequence of this is that the weight of a part in the model can not be set directly but is instead calculated using the parts' material and volume. In order to get around this, a custom material is created for each part in which the density is set such that the resulting part weight matches the real-life weight. The inertia values that were calculated using this method can be seen in Table 3.2. The difference in calculated inertia is minimal for the X and Z axes, and the Y axis is within a 10% error margin. The cause of the difference in this axis is not known. Another advantage of using a CAD model for the moment of inertia is that it can also be used to check whether the assumption of all non-diagonal terms being zero was correct. The non-diagonal terms are all at least an order magnitude lower, which is accepted as being low enough to consider them zero.

3.3.3. Discussion

Compared to both identification methods, the original inertia values significantly underestimated the inertia around the X and Y axes. This was to be expected, as the quadcopter has been modified with an extra companion computer. The weight of this computer is centred around the Z axis but not around the X and Y axes. When performing these experiments, it was found that the CAD model was relatively

Table 3.2: Comparison of MoI values found using several methods.

	Original	**Pendulum**	**CAD**
I_{xx}	2.16E-02	2.8E-02	2.76E-02
I_{yy}	2.16E-02	3.3E-02	3.00E-02
I_{zz}	4.00E-02	3.8E-02	3.95E-02
I_{xy}			2.00E-07
I_{xz}			-1.10E-03
I_{yz}			-1.28E-05

quick to implement and has the extra benefit of being able to modify the model in case the quadcopter is modified. The difference between the results of the two methods is small enough that switching between them does not have a noticeable effect on the simulator.

4

RPM Curve

This chapter will identify the steady-state response of a commanded output on the rotor velocity. Section 4.1 will first explain the theoretical background and elaborate on the factors that influence this curve. Section 4.2 will describe the formula currently used to model this effect. Section 4.3 will describe the experiments for the identification. Finally, Section 4.5 will conclude on the results of those experiments.

4.1. Background

The rotor RPM at a specific actuator command depends on the battery's voltage level. A LiPo battery, such as the one used in this research, has a nominal cell voltage of 3.7V. This means that it is 4.2V at full charge and can damage when drained below 3V per cell. Voltage should preferably not go below 3.5V per cell to conserve the battery. The battery has four such cells, which leads to a voltage range of 12V to 16.8V. Figure 4.1 shows example curves where the influence of battery voltage on the produced thrust is visible. The voltages in this plot are higher as the data was collected using a six-cell battery.

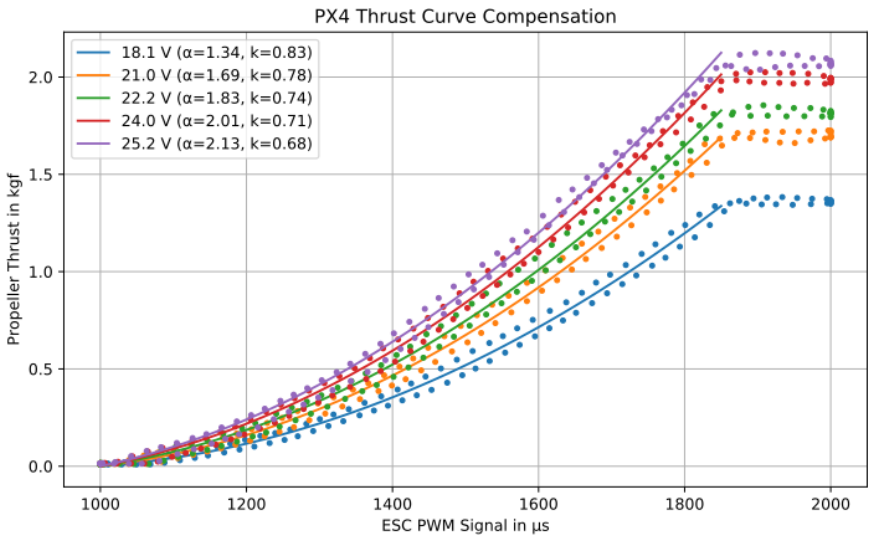


Figure 4.1: Voltage influence on produced thrust (PX4, 2022d).

The voltage of a battery is not purely a function of the percentage remaining in the battery. A high discharge current will lead to a sharp drop in battery voltage. This voltage difference is referred to as V_{edge} and can be modelled as follows (Hoque et al., 2021).

$$V_{edge} = R_b \times \triangle I = R_b \times I - R_b \times I_0, \tag{4.1}$$

where I_0 is the baseline current load, I is the new discharge current, and R_b is the internal resistance of the battery. To minimize the voltage drop, it is, therefore, necessary to use a battery with low internal resistance. The internal resistance of a battery depends on its type and quality. To find a battery suitable for high discharge rates, the manufacturers list a C rating for their batteries. This rating determines the maximum discharge rate of a battery, where 1C corresponds to discharging the total capacity in one hour. Quadcopter batteries generally have a rating above 60C, which indicates that they should be able to fully discharge in one minute without causing damage (Balaji, 2017). It should be noted that the C-ratings listed by the manufacturers, especially in the case of no-name brands, can be far too optimistic about the eventual performance of the battery.

4.2. Model

The default motor controller model assumes a linear relationship between the demanded output command and the produced angular velocity. This relationship is given in Equation 4.2[1].

$$\Omega = (\text{input\_command} + \text{input\_offset}) * \text{input\_scaling} + \text{zero\_position\_armed} \tag{4.2}$$

The input_command is a normalized input command from 0 to 1. input_offset, input_scaling and zero_position_armed are parameters that can be modified for the specific quadcopter model. By default, these values are 0, 1000 and 100, respectively. Meaning that the angular velocity is 100 rad/s when armed and then increases linearly to 1100 rad/s at maximum velocity.

Compared to the theory, it can be seen that the model does not consider battery voltage. A system identification experiment will be performed in which the commanded input, battery voltage and RPM output are measured to verify if this formula sufficiently models reality. If it does not, an attempt will be made to find a better relationship.

4.3. Identification

To identify the actuator output to rotor velocity relationship, an experiment is done where the quadcopter thrust is increased from idle to full power in steps. The RPM is measured using a tachometer, and the battery voltage is obtained from the power module in order to find the relationship. A detailed description of the steps required to perform this experiment is given in Section A.4.5.

The experiment is performed such that the actuator command increases from zero to max and in reverse, where the command decreases from max to zero. Performing the experiment in reverse ensures the results are not influenced by hysteresis. The data points are coloured according to the voltage level at the start of the step. A second-degree polynomial function has been fit onto the mean of the test data for both the normal and reversed test. The collected data can be seen in Figure 4.2. As expected, the data points with a higher voltage achieve a higher RPM at the same motor command. Unfortunately, it can also be seen that the voltage level drops all the way from a full battery to the absolute minimum in just a single test. Preferably, data would be collected such that a curve can be made for several voltage levels, such as Figure 4.1.

The experiment has been modified to prevent the voltage drop. In between motor command steps, the RPM is reduced to idle to give the battery time to recover its voltage level. The data collected in this experiment is plotted together with the data from the previous experiment; the result can be seen in Figure 4.3. This experiment obtained very unexpected results. When increasing from idle, the initial RPM is much higher than expected. Max RPM is already almost achieved at half output power, and after that, the RPM first decreases before increasing again. It is clear that this data is not representative of real-world flight behaviour. A different experiment will have to be performed to better understand the voltage level influence on rotor RPM.

The experiment has again been modified to reduce the voltage drop, but this time by enabling fewer rotors. The experiment is performed with one or two rotors spinning and the other rotors at idle. The reduced total power demand should reduce the experienced voltage drop. It was found that performing the experiment in this setup induced heavy vibrations in the fuselage. To prevent these vibrations, it was decided to have a short idle time of two seconds between output steps. A plot of the data collected in this experiment can be seen in Figure 4.4. This data again contains unexpected behaviour, as the achieved RPM levels are much higher than expected. Around actuator command 0.7, we see that the

[1] https://github.com/PX4/PX4-SITL_gazebo/blob/main/src/gazebo_mavlink_interface.cpp#L1128

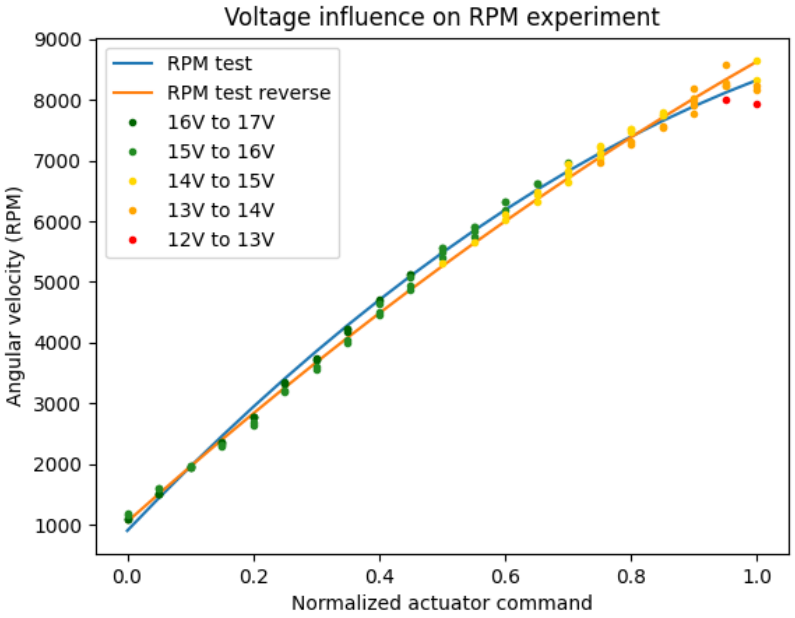


Figure 4.2: Voltage level influence on RPM curve.

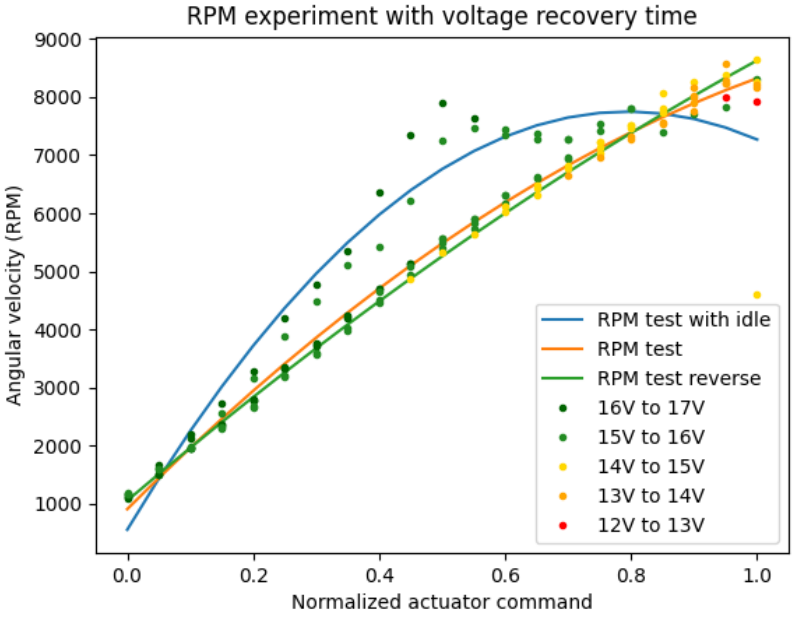


Figure 4.3: Voltage level influence on RPM curve with voltage drop recovery.

lower curve sometimes has a higher measured voltage than the higher curves. The difference between the curves can not be explained by only the battery voltage level. It is possible that the short idle time still influences the results.

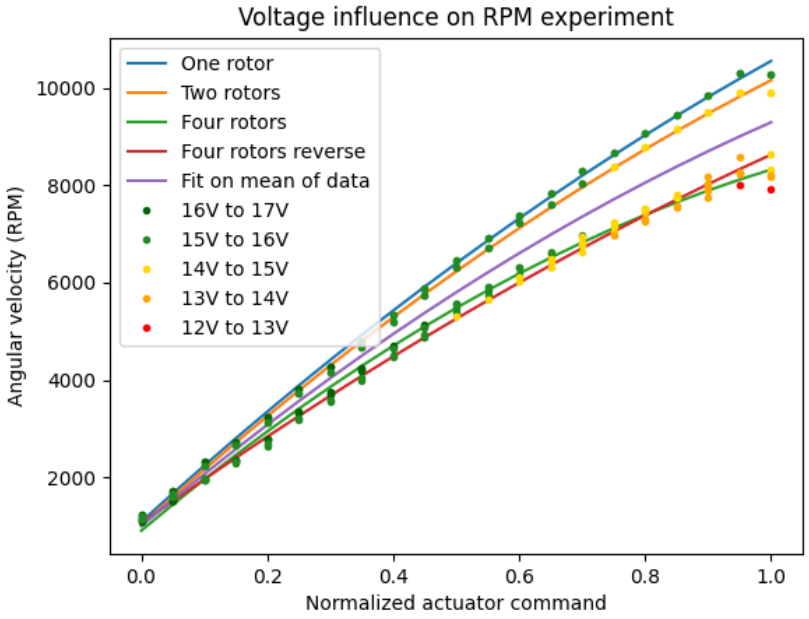


Figure 4.4: RPM curve with fewer rotors enabled.

4.4. Validation

To validate the RPM curve, the performance of several different curves will be evaluated in the simulator. First, a qualitative comparison will be made between two of the curves found in this chapter in order to understand the influence that the RPM curve has on the simulator behaviour. Secondly, a quantitave comparison will be made between a number of RPM curves to find the curve with the best simulator performance.

4.4.1. Qualitative analysis

In this subsection, two different curves will be evaluated in the simulator. The first RPM curve is created from the original test data, as seen in Figure 4.2. A second-degree polynomial function has been fit onto the mean of the data, which leads to the equation $\Omega = -287x^2 + 1070x + 101$. The second RPM curve is constructed from the test data where fewer rotors were used, as seen in Figure 4.4. The polynomial function of this curve is $\Omega = -269x^2 + 1154x + 105$. The first curve will be called the lower RPM curve, and the second one will be called the higher RPM curve.

The original linear relationship in the simulator has been replaced with either of these new curves. To validate their performance during real flight behaviour, a trajectory has been flown in real-life and in the simulator. This trajectory includes a portion at the start where velocities are low, and the rotor RPMs are near hover level. Later in the flight, aggressive pitch manoeuvres will be performed that require the RPM to go much higher. This one trajectory, therefore, tests several parts of the whole RPM curve.

The altitude of the quadcopter during the fast part of this trajectory can be seen in Figure 4.5. When the lower RPM curve is used, the simulated quadcopter is unable to reach the high RPMs that are required for aggressive pitch manoeuvres. This leads to a thrust during pitching that is too low to maintain altitude. The simulated quadcopter, therefore, keeps losing height until it eventually crashes into the floor. When the higher RPM curve is used, the simulated quadcopter can maintain altitude throughout the flight.

The pitch angle during the initial slow part of the trajectory can be seen in Figure 4.6. The downside of the higher RPM curve is clearly visible in the plot showing the pitch angle. The RPM curve predicts

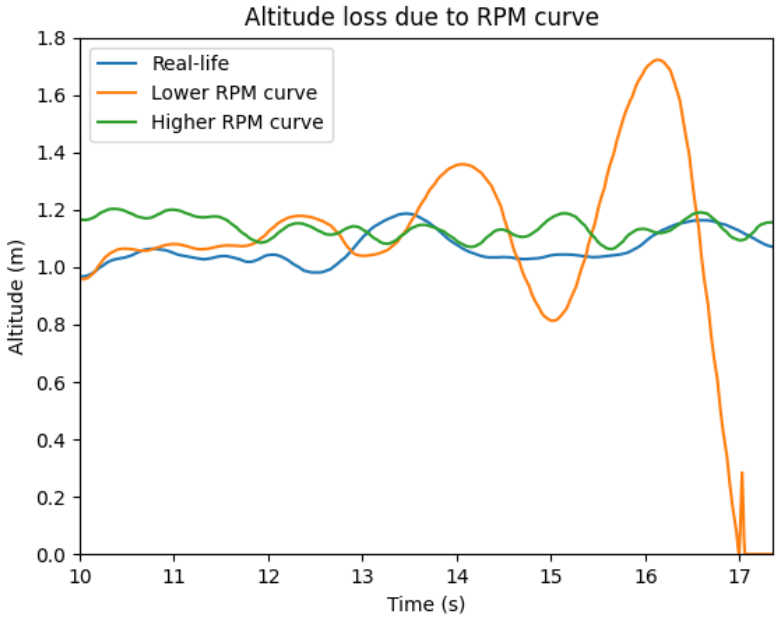


Figure 4.5: Altitude during aggressive flight with different RPM curves.

a too-high RPM near hover levels, which is visible in the oscillations of the pitch angle. The lower RPM curve and real-life flight data are much more stable. This comparison shows that the identified shape of the RPM curve, in which the line flattens out at higher input commands, may not be properly representative of real-life behaviour. The curves in this comparison either had a too-high RPM around hover or a too-low RPM at high output commands. To resolve this, the curve should flatten out less. In the next section, a quantitave comparison will be made with several linear curves.

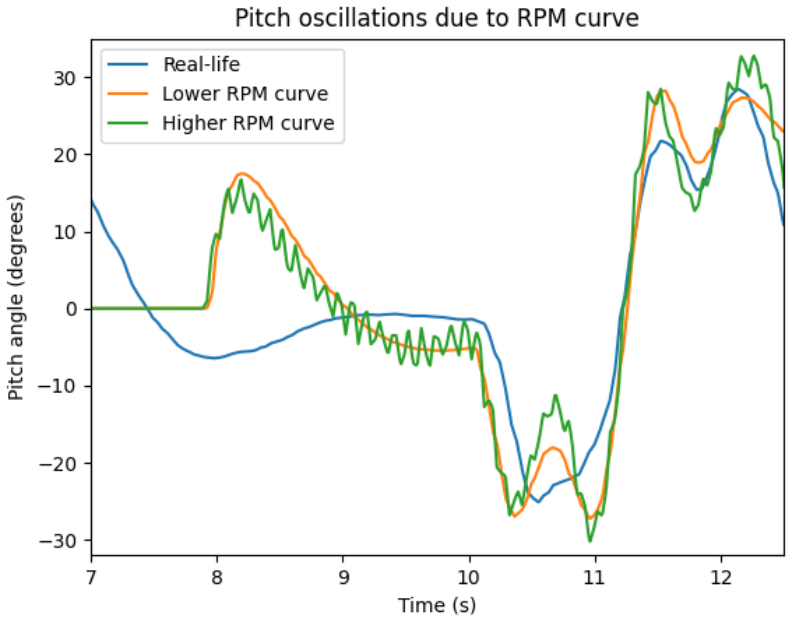


Figure 4.6: Pitch angle during aggressive flight with different RPM curves.

4.4.2. Quantitative analysis

The curves in this section have been selected based on their performance in the simulator. The curves are judged based on two main properties; their stability at low velocities and their trajectory-tracking performance at high velocities. The metrics used to assess this are the root-mean-square error (RMSE) (root mean square error) of the X and Z position and the RMSE of the roll angle. To assess the performance of the curves correctly, a large portion of the RPM curve should be used. To accomplish this, two different trajectories were selected that show the quadcopter performance in relatively low-velocity situations and in high-speed high-acceleration situations. The slower situation is a circular trajectory in which the maximum velocity is $3\,\mathrm{m\,s^{-2}}$, and the maximum acceleration is $5.5\,\mathrm{m\,s^{-2}}$. The second trajectory is a diagonal trajectory, which has a maximum velocity of $5\,\mathrm{m\,s^{-2}}$ and a maximum acceleration of $9.5\,\mathrm{m\,s^{-2}}$. Note that the flight speed is slowly increased after the start of the trajectory, so in the first lap, the velocity and acceleration are much lower.

Three new curves have been added to compare the two curves from the previous section with. These new curves are linear curves defined by Equation 4.2. Their input offset is zero, and their idle rotational velocity when armed is 100. The difference between the curves is in the input scaling variable, which is set to 1000, 950 or 900 for the different curves. Note that the curve with an input scaling factor of 1000 is the curve that was originally used in the model. The different curves can be seen in Figure 4.7.

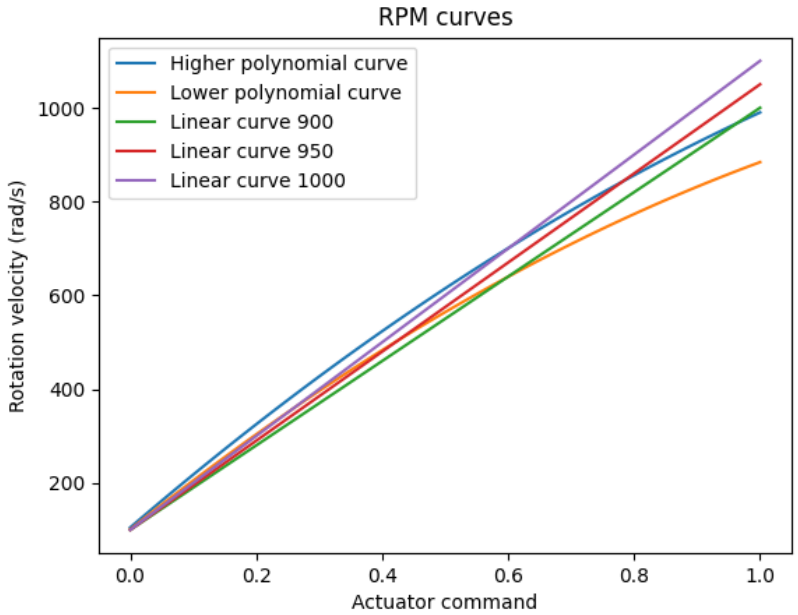


Figure 4.7: RPM curves used in quantitative analysis.

The metrics of the circle trajectory can be seen in Table 4.1. All RPM curves perform roughly the same in terms of position tracking, but two RPM curves cause significant oscillations. This is visible in the RMSE or the roll angle for the highest linear curve and the higher polynomial curve. These oscillations indicate that these curves predict an unrealistically high rotation velocity near the hover output command of approximately 0.6.

The metrics of the diagonal trajectory can be seen in Table 4.2. In this case, two curves failed to track the trajectory and crashed into the floor during the flight. This is the case for the linear curve with an input scaling of 900 and the lower polynomial curve. For the latter, this behaviour was shown in Figure 4.5. These crashes indicate that these curves predict a too-low rotor velocity at high output commands. The performance of the other three RPM curves is roughly the same in this trajectory.

These two tables show the importance of having a correct RPM curve. The linear curve with an input scaling of 1000 was too high and caused oscillations. The linear curve with an input scaling of 900, although only 10% lower than the previous curve, was already too low and unable to perform the aggressive manoeuvres. The curve with an input scaling of 950 performed adequately in both trajectories.

Table 4.1: Trajectory tracking metrics of circle trajectory with different RPM curves.

	X position RMSE	Z position RMSE	Roll RMSE
Linear curve 1000 (original)	0.041	0.083	3.799
Linear curve 950	0.045	0.055	2.060
Linear curve 900	0.055	0.037	1.882
Lower polynomial curve	0.085	0.041	2.012
Higher polynomial curve	0.045	0.079	9.404

Table 4.2: Trajectory tracking metrics of fast diagonal trajectory with different RPM curves.

	X position RMSE	Z position RMSE	Roll RMSE
Linear curve 1000 (original)	0.257	0.116	11.437
Linear curve 950	0.270	0.082	9.550
Linear curve 900	1.844	0.375	44.998
Lower polynomial curve	2.970	0.449	49.734
Higher polynomial curve	0.262	0.085	8.854

4.5. Discussion

The data collected in the identification experiments does not yield a conclusive result. When two experimentally identified RPM curves were tested in the simulator, it was found that neither curve had the expected performance during both low-speed and high-speed flight. A linear RPM curve, with an input scaling of 950, provides consistent performance throughout the two trajectories. This result does not match the findings of the identification experiments. A possible reason for this is that the quadcopter can reach very high RPMs, but only for a short period of time. This effect is visible in Figure 4.3, although in real-life flight, the motors would never go all the way down to idle.

The linear curve with a scaling factor of 950 performed the best in the validation and will be the curve used in the rest of this research. It should be noted that is still a naive solution that purely relates the actuator command to the RPM. This is not sufficient to properly model real-life behaviour. The simulator likely becomes inaccurate if a trajectory is flown with nearly constant high outputs, which could, for example, be achieved by increasing the weight of the quadcopter.

The voltage level of the battery is one of the factors that have an influence on the RPM curve, but it is not the only factor. When spinning up the rotor from idle, the produced RPM is far higher than expected, to the point that max RPM was already almost reached at just half the output power. Additionally, the commanded output of other rotors also seems to have a significant effect on the produced RPM. To get an accurate thrust curve, it is likely necessary to combine the motor command, battery voltage level, commanded output of other rotors, and perhaps even the commanded outputs in the previous few seconds. Finding this relationship is considered outside of the scope of this research. There are a few points of interest that should be looked into in order to improve the model.

Firstly, better experimental data should be obtained to conclusively find the variables influencing the output velocity. The current experimental data indicates that motor command and voltage level alone are not enough to predict the output rotor velocity accurately, but the data used in this analysis has two main issues; the accuracy of the voltage measurements is unknown, and the number of different voltage levels at a specific output command is limited. The voltage measurements used in this experiment were obtained from the power modules' internal sensors. It is possible that this sensor is not accurate or responsive enough for use in this analysis. To be confident about the gathered data, the performance of this sensor should be verified using external equipment. With the voltage measurements verified, it is still required to have RPM measurements at a specific motor output command with different voltages to find its influence. In the current setup, it is difficult to get a measurement at a specific voltage as the battery voltage level changes quickly. So far, no experiment has been found that can avoid voltage drop whilst not otherwise influencing the results. An external power supply that can supply a steady voltage throughout the entire test may be useful in gathering more consistent data.

Secondly, more focus should be put on the part of the curve most relevant in flights. This chapter attempted to find a relationship that properly captures all behaviour. However, in practice, the quadcopter will seldom go from idle to full power or remain at full power for any extended period of time. Only considering output command sequences that can realistically occur during flight should yield data

containing a less extreme voltage drop, and may help in finding the proper relationship. It is also worth noting that PX4 has built-in load compensation that attempts to correct the commanded outputs for the voltage drop. A version of this load compensation was enabled in this experiment, but it was not able to compensate for the extreme voltage drop. This load compensation may perform better in an experiment with more realistic actuator output commands. PX4 also supports more advanced load compensation methods that may perform better. If the unexpected behaviour can be reduced in real-life, then that may simplify the required simulator model.

Finally, there is currently no proper model of battery behaviour in the simulator. The simulated battery decreases its voltage over time, but it does so without taking the command motor outputs into account. Besides the RPM curve, there should also be a model of how the battery voltage changes over time with specific motor commands. An accurate RPM curve model can be identified in the current scenario but will not be as useful as it could be until proper battery simulation is added to the simulator.

5

Motor time response

This chapter will identify the time response of the motor. Section 5.1 will give background information on what this time response is. Section 5.2 will then explain how this response is implemented in the model, and Section 5.3 explains the experiment that was performed to identify the model parameters. Section 5.4 explains the validation process, and finally Section 5.5 will give a discussion on the results.

5.1. Background

When a signal is sent to the motors their response will not be instantaneous. The motor will have to overcome the inertia of the rotor before settling at its target velocity. This delay is described using a time constant. The time constant up is the time it takes to reach 63.2% of the maximum motor speed (Yajima et al., 2000). Conversely, the time constant down will be the time it takes to go down from full speed to 36.8% of the motor speed. Bernard et al., 2017 notes that the time constant down is expected to be significantly higher since the motor ESCs do not have any active brake implemented. The rotor will therefore only slow down due to drag, which makes the thrust response slower.

5.2. Model

The rotor RPM response is modeled as a first order system with the following transfer function (Furrer et al., 2016).

$$X(s) = \frac{1}{\tau * s + 1} U(S) \tag{5.1}$$

This is turned into the following continuous time system.

$$dx(t) = \frac{-1}{\tau} * x(t) + \frac{1}{\tau} * u(t) \tag{5.2}$$

And then finally into the discretized system that is used in the code.

$$x(k+1) = \exp(s * \frac{-1}{\tau}) * x(k) + (1 - \exp(s * \frac{-1}{\tau})) * u(k), \tag{5.3}$$

where $u(k)$ represents the desired rotor velocity, $x(k)$ is the previous state and $x(k+1)$ is the next state, τ is the time constant and s is the time since the last update step. Different time constants are used based on whether the signal is going up or down. The time constant can be identified by measuring the rotor speed response to a step input using a tachometer. Alternatively, Partovi et al., 2012 estimates the time constant by measuring the voltage drop on the battery using an oscilloscope to estimate the rotor speed. Dhaybi and Daher, 2019 also use the voltage drop to estimate the rotor speed, but instead do this using purely onboard sensors.

5.3. Identification

Several data sources can theoretically be used to identify the time constants. The time constant is expected to be a low value, and it is therefore vital that the source data has a high message rate. The original value for the time constants is 0.0125 as time constant up, and 0.025 as time constant down. The battery telemetry is, by default, published at merely 0.5Hz. This message rate can be increased, but the CPU capacity limits of the flight controller made it impossible to increase the rate above 10Hz. Such a message rate is not high enough to identify the time constants.

The tachometer data is, by default, published at 5Hz. The message rate of this message has been successfully increased to 100Hz without hitting the CPU limits of the flight controller. This is by far the highest frequency data source available, and it will therefore be used to identify the time constants. During the experiment, the RPM is increased from idle to max for a few seconds and then decreased back down to idle. The data collected with this experiment can be seen in Figure 5.1. A detailed description of the steps required to perform this experiment is given in Section A.4.5.

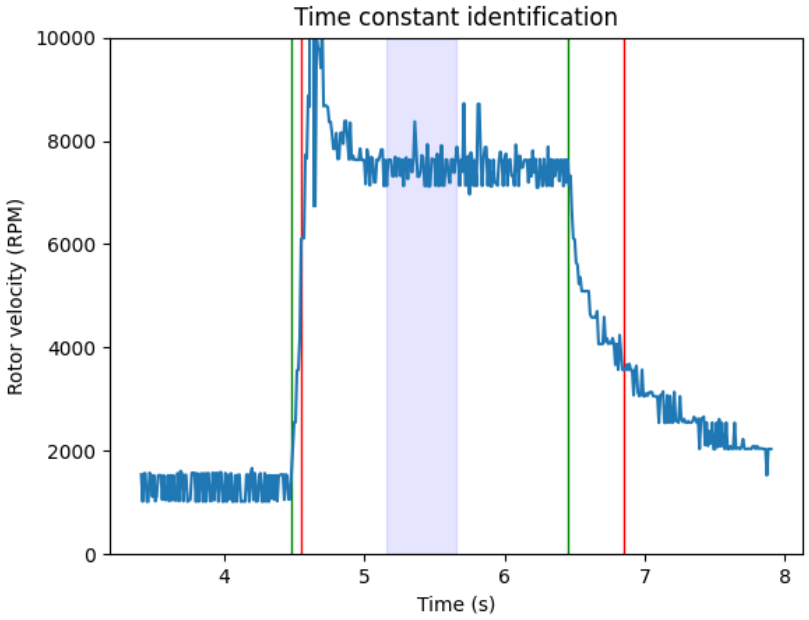


Figure 5.1: Time constant identification using tachometer data.

The shaded area in the middle is used to calculate the max RPM. The RPM has a large peak first before settling, but this is considered a side effect of increasing the RPM from zero to max instantly and therefore not considered as part of the max RPM that the rotor reaches. The green lines indicate the start of the maneuver, where the motor command is either increased to max or dropped back to zero. The red lines indicate the locations where 63.2% of the maneuver has been completed. The time constants are then the time spent between the two lines. Taking the mean over several experiments gives a time constant up of 0.07 seconds and a time constant down of 0.45 seconds. Compared to the original values it can be seen that the time constant up has increased seven fold, and the time constant down almost twenty fold. The next section will validate whether these numbers are realistic.

5.4. Validation

The validation process for the time constants consists of two steps. First, the time constant experiment will be repeated in the simulator to verify whether the resulting RPM curves are similar. Secondly, the found time constants will be used to fly trajectories in the simulator to analyze the influence that it has on the flight performance.

5.4.1. Curve validation

The experiment was repeated in the simulator three different times. The first time with the old values, where time constant up is 0.0125 and time constant down is 0.025. The second time with time constant up of 0.06 and time constant down of 0.45. And then a third time with again a time constant up of 0.06, but a time constant down of 0.2. The curves collected in this experiment can be seen in Figure 5.2.

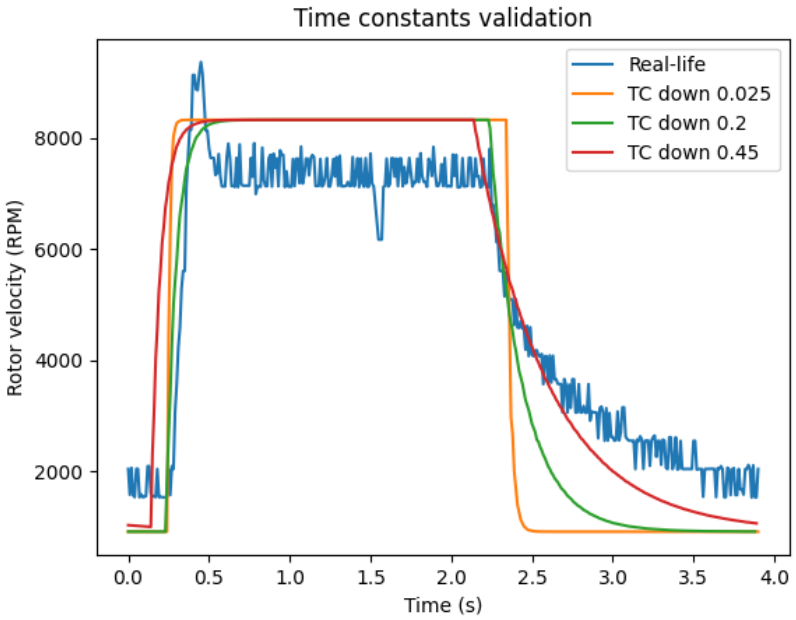


Figure 5.2: Time constant curve validation.

Looking at the orange curve with the original values it is clear that this overestimates both the speed at which the rotor can spin up as well as the speed at which it will spin down. Using the time constant up of 0.06 gives a curve that much more closely resembles the real behaviour. The time constant down of 0.45 that was identified gives a curve that is in the initial part a lot less steep than the real-life data. This initial drop is important for the quadcopter as a slow response in this section can quickly lead to unstable flight behaviour.

A third time constant down of 0.2 was therefore plotted to more closely match the behaviour in the initial portion. This new curve approximately matches the shape of the real-life curve at the start, but then also approaches the idle RPM much faster. The formula used to model the time response is clearly not a perfect representation of the real-life behaviour.

5.4.2. Flight validation

To validate the flight performance of the newly identified time constants a trajectory was flown in real-life and in the simulator to compare the angular acceleration. The trajectory starts with a stable hover, and then accelerates quickly to a 2 rad/s yawing motion. The data collected in this experiment can be seen in Figure 5.3.

It can be seen that none of the trajectories quite tracks the performance of the real-life flight. However, increasing the time constant only makes the performance worse. The extra delay before the thrust is actually reduced causes frequent overshoots. The higher the time constant is made, the higher this overshoot becomes. The plot here only shows one specific angular acceleration as it clearly illustrates the issue, but this same behaviour was found in all trajectories when the time constant is increased.

5.5. Discussion

The curve validation has shown that the used model is not an accurate representation of the real life behaviour when reducing velocity. An attempt was made to visually match the shape of the curve, but in flight validation any increase in time constant down showed a reduction in performance. The exact

cause of this phenomenon is currently not known.

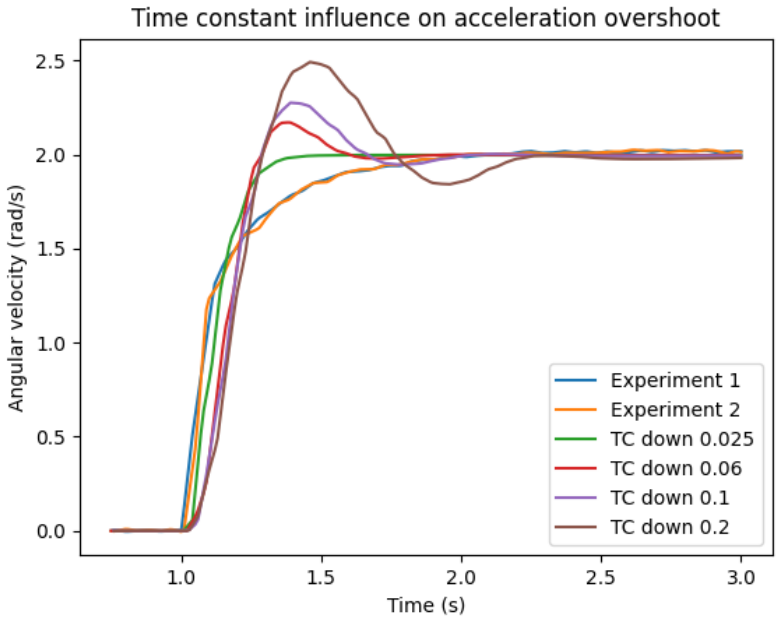


Figure 5.3: Time constant flight validation.

6

Thrust

This chapter describes the thrust generated by the quadcopter. Section 6.1 explains the underlying aerodynamics theory to get an understanding of the factors that influence the generated thrust.

6.1. Background

One of the first model derivations for quadcopters was done by Fay, 2001 and was used to fly a quadcopter. Bouabdallah, 2007 used the same approach and also used the derived equations to implement a simulator in MatLab. Bangura et al., 2016 use the existing theory to create a simplified aerodynamics model that can be used in robotic applications. These papers use a combination of momentum and blade element theory to derive the aerodynamic forces. Momentum theory is used to determine the inflow velocity and advance ratio, and blade element theory is used to determine the aerodynamic forces and torques on the rotors. The exact derivation is outside of the scope of this work, but an understanding of where the forces come from and which effects influence them will be essential to properly understand the research results.

The aerodynamic analysis is largely based on the work of Fay, 2001 and Bouabdallah, 2007. The forces and moments acting on the quadcopter will be analysed from a body-fixed frame. This frame has the Z-axis pointing up parallel to the rotor axis, the X-axis pointing forward in the direction of flight, and the Y-axis pointing towards the left side of the quadcopter. A visual representation of this can be seen in Figure 6.1.

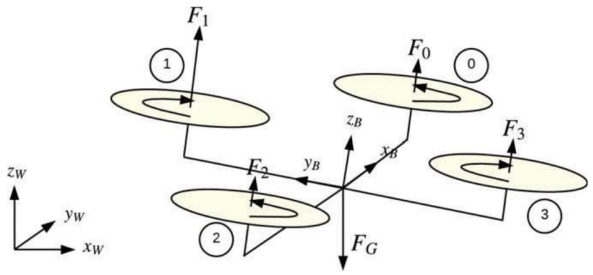


Figure 6.1: Sketch of a quadrotor with the body-fixed frame and the world frame. (Furrer et al., 2016).

Using this frame we can define two dimensionless quantities that will be useful in explaining how the aerodynamic forces relate to the movement of the quadcopter. The first dimensionless quantity is the advance ratio, which relates the horizontal velocity to the rotor speed. It is defined as follows

$$\mu = \frac{V}{\Omega R}, \tag{6.1}$$

where V is the horizontal velocity vector defined as $V = \sqrt{\dot{x}^2 + \dot{y}^2}$, Ω is the rotor angular velocity and R is the rotor radius.

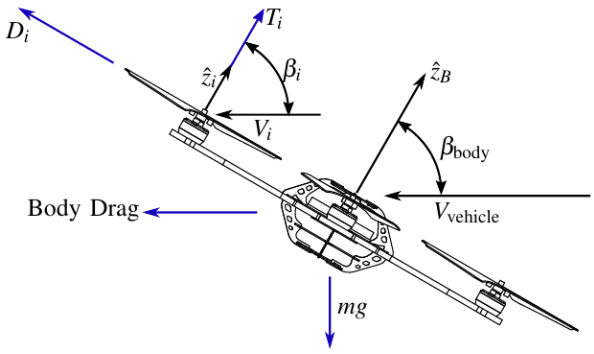


Figure 6.2: Thrust and rotor drag forces visualized. Note that the body-frame Z-axis is placed in the opposite direction from our definition here (Gill and D'Andrea, 2017).

The second quantity is the inflow ratio, which relates the vertical velocity to the rotor speed and is defined as

$$\lambda = \frac{\nu_1 - \dot{z}}{\Omega R}, \tag{6.2}$$

where ν_1 is the inflow velocity.

It is worth noting that a similar advance ratio term, this time defined as J, is also used for airplane propellers. However, since the advance ratio is determined based on the horizontal airspeed of the aircraft, this then refers to the velocity along the axis of the propeller instead of perpendicular to it. This means that the effects of an airplane advance ratio are similar to that of the inflow ratio of a quadcopter. Caution should be taken when reading figures or formulas with an advance ratio in them so that the correct propeller orientation is used.

Blade element theory can be used to find the thrust by integrating all the blade elements over the Z-plane. Figure 6.2 visualizes the thrust and drag forces on a quadcopter, marked as T_i and D_i, respectively. Fay, 2001 uses this technique to obtain the following formula for thrust:

$$\begin{aligned} T &= C_T \rho A \left(\Omega R\right)^2 \\ \frac{C_T}{\sigma a} &= \left(\frac{1}{6} + \frac{1}{4}\mu^2\right)\theta_0 - \left(1+\mu^2\right)\frac{\theta_{tw}}{8} - \frac{1}{4}\lambda \end{aligned} \tag{6.3}$$

, where σ is the solidity ratio, a is the lift slope, ρ is the air density, A is the propeller disk area, θ_0 is the angle of incidence, which is the angle between the chord line of the rotor and the horizontal axis of the drone, and θ_{tw} is the twist pitch, which is the twist of the propeller blade itself.

θ_0 is always larger than θ_{tw}. Therefore, this formula shows that thrust will increase at a higher advance ratio. Reversely, the thrust will decrease when the inflow ratio increases. These results can be tricky to interpret in terms of flight performance since a drone does not stay horizontal during flight. Figure 6.3 shows that, as expected, thrust increases when there is a higher wind speed during horizontal flight. In practice, the quadcopter will have a pitch angle to achieve forward flight, which causes the inflow ratio to increase and the thrust to decrease. The result of this trade-off is that higher horizontal speeds increase thrust at low pitch angles, but reduce thrust at high pitch angles.

Several models are used in literature to simulate thrust behaviour. The model used in the motor model will be described in Section 6.2, but an alternative version will be discussed here to give an overview of the possible implementations. This specific model was defined by Bouabdallah, 2007 and used further by Svacha et al., 2017. Its equation is as follows.

$$T_i = c_1 \Omega_i^2 \left(c_2 \left(1 + \frac{3}{2}\mu_i^2\right) - \lambda_i\right), \tag{6.4}$$

where T_i is the thrust produced by the i^{th} propeller, c_1 and c_2 are positive thrust coefficient constants, μ_i is the advance ratio and λ_i is the inflow ratio. This model incorporates the advance and inflow ratio, which should theoretically lead to better performance than a model that leaves these terms out. In practice, the use of this in a controller led to an improved tracking performance in vertical flight but a slightly reduced performance for horizontal flight.

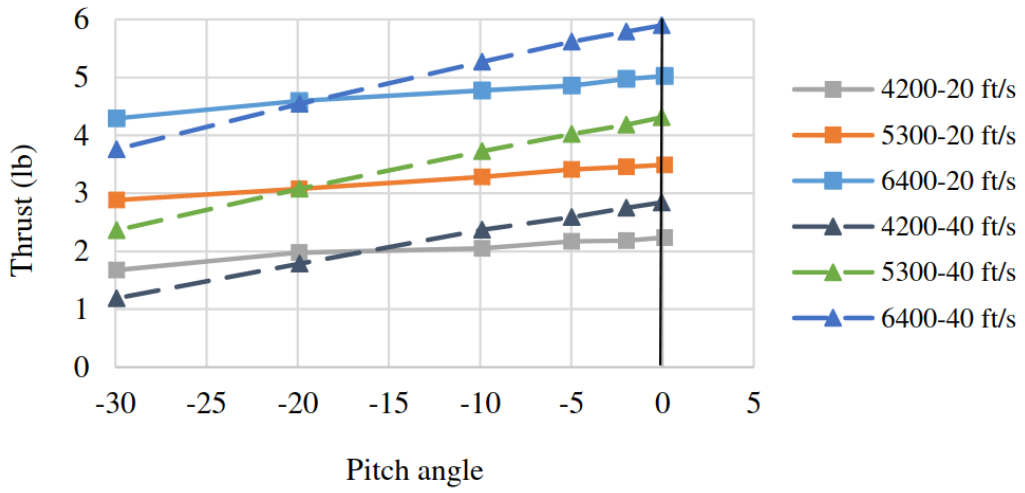


Figure 6.3: Thrust data obtained in a wind tunnel at different wind speeds (Baris et al., 2019).

6.2. Model

In the motor model, the thrust formula is simplified to a version that is often used in simulator models, namely

$$F_T = \Omega^2 C_T \cdot \epsilon_{z_B}, \tag{6.5}$$

where Ω is the angular velocity of the rotor blade, C_T is the thrust coefficient, and ϵ_{z_B} is the unit vector pointing in the z-axis of the body frame.

If we compare this to the theoretical Equation 6.3, we can see this is a simplified version that does not take the air speed over the rotors into account. A simplified approximation of the effect of the vertical velocity on the thrust is added using a scalar. This scalar is defined as

$$T_{scalar} = 1 - (\dot{z} - v_1)/25.0, \tag{6.6}$$

which scales the thrust based on the vertical velocity of the quadcopter minus the vertical wind velocity. This is a simplified model for the effect of the inflow ratio on the produced thrust. It linearly reduces thrust at higher vertical speeds by using a scalar value, which scales the value from full thrust at 0 m/s to zero thrust at 25 m/s of upwards velocity. However, there is no theoretical basis for the use or exact value of this scalar. Additionally, the effect of the advance ratio on the produced thrust is not modelled.

6.3. Identification

Two different methods will be used to find the motor constant. The first method involves a theory-based estimation of the motor constant. The accuracy of this method is not known, but if acceptably accurate, it could provide a very easy way of determining the motor constant. The second method involves performing an experiment in which the thrust is measured using a load cell.

6.3.1. Theory based identification

Spakovszky, 2008 defined the thrust coefficient as a function of propeller design parameters. Therefore, using that formula and propeller design values available online, the thrust coefficient can be calculated. The coefficient of thrust is defined as follows.

$$C_T = \frac{T}{\rho n^2 D^4}, \tag{6.7}$$

However, the motor constant is not dimensionless, so the formula needs to be rewritten. Combining this with the motor models Equation 6.5, we get the following formula.

$$T = C_T \rho n^2 D^4 = \omega^2 * motor\_constant = (2\pi n)^2 * motor\_constant \tag{6.8}$$

This can be used to show that the motor constant formula is as follows.

$$motor\_constant = \frac{C_{T0} \rho D^4}{(2\pi)^2} \tag{6.9}$$

As explained in Section 6.1, the thrust coefficient depends on the advance ratio. The motor constant is calculated in static conditions. This is shown in the formula by C_{T0}, which shows that the thrust coefficient is given at an advance ratio J of 0.

This leaves the problem of finding the thrust coefficient of the rotor. Luckily, these values are available in literature for many different rotors. For example, a 9450 propeller, which the HoverGames quadcopter is by default supplied with, has been tested by Deters et al., 2017. However, on the quadcopter used in this research, the propellers have been replaced with larger 1147 propellers. This name means that the rotor has a diameter of 11 inches and a pitch of 4.7 inches. A source that has tested a large variety of propellers is the UIUC propeller database (Brandt et al., 2022). This database contains thrust and power coefficient curves for many types of propellers and is regularly updated with new propellers. The thrust coefficient curve is shown in Figure 6.4.

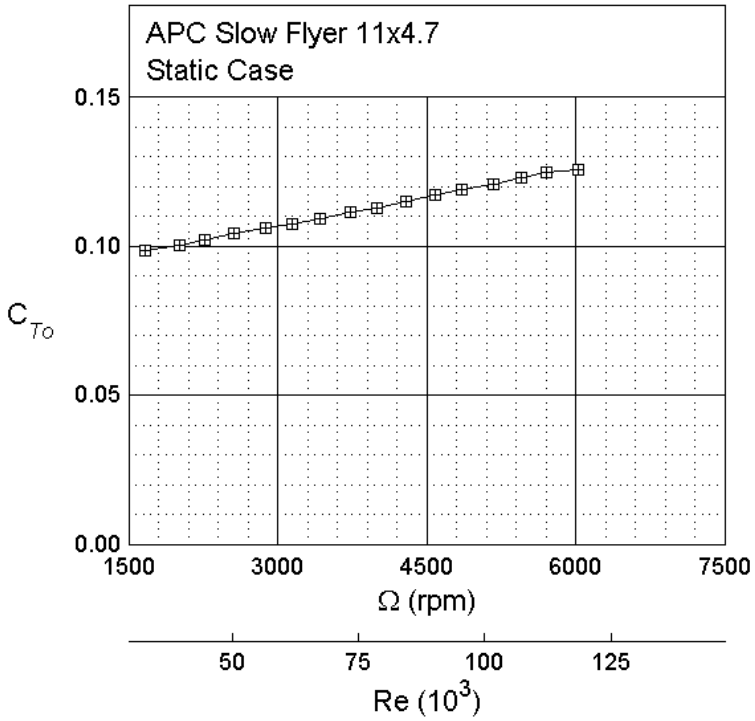


Figure 6.4: APC 11x4.7 thrust coefficient (Brandt et al., 2022).

During hover, the rotor RPM is about 5900, and therefore the thrust coefficient at that point is chosen. The thrust coefficient in this part is about 0.125. Converting the propeller diameter to meters and filling in the average air density gives the following equation.

$$motor\_constant = \frac{0.125 * 1.225 * .2794^4}{(2\pi)^2} \approx 2.364e-5 \tag{6.10}$$

6.3.2. Experimental identification

To identify the motor constant, a test setup has been used in which the rotor velocity and produced thrust can be measured simultaneously. The quadcopter is attached to a seesaw, with a load cell placed under the seesaw on the other side. When it produces thrust, it thus pushes down on the load cell to measure the created thrust. A detailed explanation of this setup is given in Section A.3. A picture of the thrust setup can be seen in Figure 6.5.

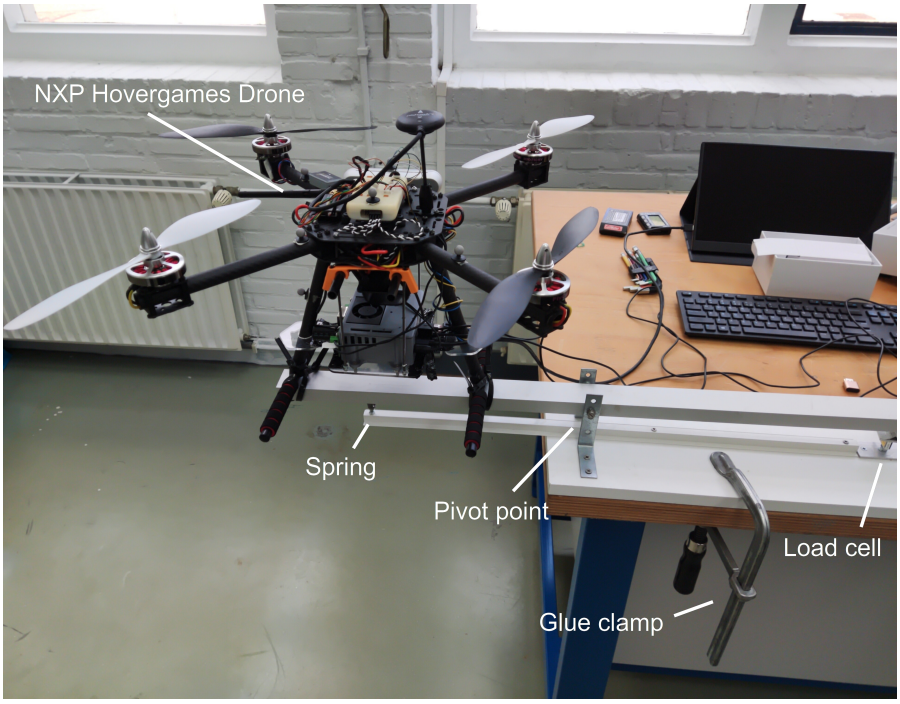


Figure 6.5: Thrust measurement setup.

Figure 6.6 shows the raw data of one of the experiments. The motor output is increased in 21 separate steps, although it can be seen that the final step does not increase rotor velocity any further. The experiment has also been done in reverse to ensure that the battery voltage does not cause this effect, and the same result was seen in that case. As observed in the experiments performed in Chapter 4, the voltage level drops quickly when running the quadcopter at high RPM. In this case, that does not matter much since the motor constant is a relationship with the produced rotor RPMs and not with the motor command.

To make the raw data easier to work with, it is first interpolated at 100Hz. An average rotor velocity and force are calculated per step by looking at the middle 4 seconds of each step. This removes the transient portions of the data where the quadcopter is still increasing its rotor velocity. The outliers in the tachometer data are removed by calculating a z-score for each data point in a section. Any data points with a z-score above three are removed. This score means that all points within three standard deviations are retained, which was found to be a good cutoff point that reliably removes the outliers while retaining all proper data points.

The force measurements need to be scaled due to the physical dimensions of the test setup. The lever arm is longer on the side of the load cell than on the side of the quadcopter. Therefore, the measured values need to be increased with a certain factor to get the real thrust produced by the quadcopter. This factor can be calculated as $37cm/26cm \approx 1.42$. Finally, we need to account for the fact that four rotors generate this thrust, and the motor constant is used for each separate rotor. Therefore, the calculated constant needs to be divided by four. Fitting a quadratic function on the data points leads to a motor constant of $1.45e^{-5}$. A plot of the data can be seen in Figure 6.7.

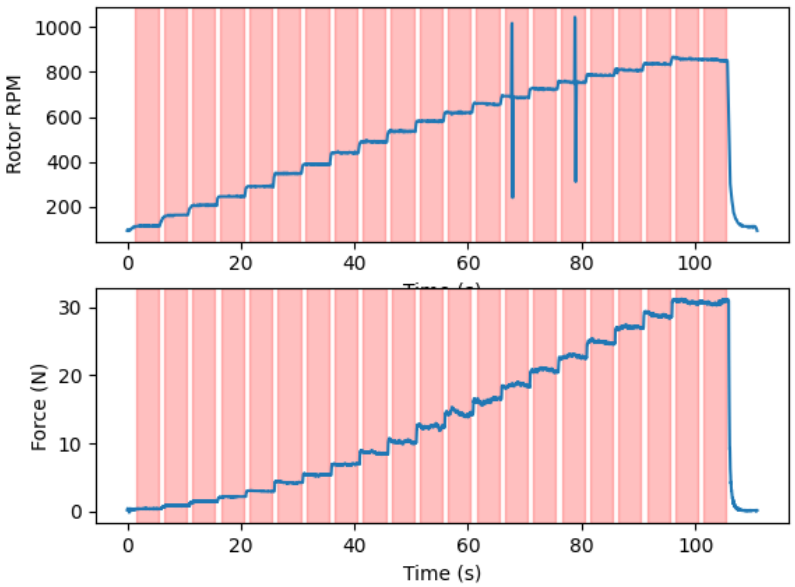


Figure 6.6: Thrust calculation.

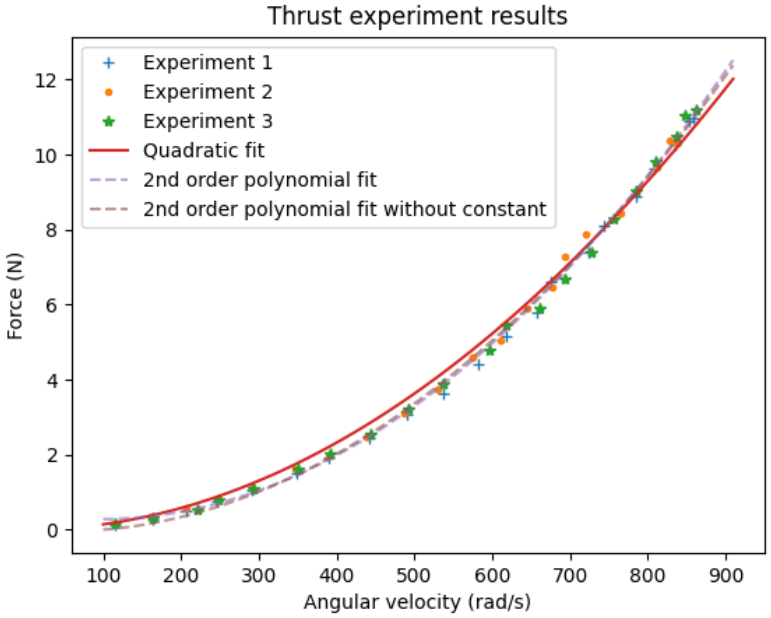


Figure 6.7: Thrust experiment results.

6.4. Validation

The two different methods of calculating the motor constant found very different values, with the theoretically determined motor constant being 60% higher than the experimentally determined constant. The validation of the motor constant consists of two steps. First, hover experiments of the quadcopter will be performed to verify the collected data. Secondly, the experiment will be repeated in the simulator to verify that the processing of the data and calculation of the motor constant is correct. If both parts are validated, then it can be said that the motor constant is correct for sure.

6.4.1. Flight validation

The experiment consists of hovering the quadcopter while measuring the rotor RPM. Since the quadcopter is hovering, the thrust is equal to the weight, giving the RPM to thrust relationship for several operating points. Then, the experiment is repeated with weights attached to the quadcopter. The weight attached is increased by 100 grams each time, going up to 500 grams of extra weight. Each weight is flown twice, with the tachometer being moved to the rotor on the other side of the quadcopter from the first flight. This accounts for a potential offset of the centre of mass.

The RPM to thrust operating points in this experiment are shown as dots in Figure 6.8. The motor constant curves found before are also shown again to compare their accuracy. Additionally, the default motor constant value is also shown for reference. It can be seen that the validation values follow the location and curve of the experimentally determined motor constant quite well, and it is clearly a considerable improvement over the default values. The theoretically determined motor constant, unfortunately, vastly overestimates the thrust the quadcopter produces. The error compared to the validation data points was calculated to quantify their performance. The original thrust curve is, on average, 39% below the validation points. The experimentally determined model is 3.5% above the validation points, and the theoretically determined model is 69% too high.

The validation points are all below the experimentally determined curve. A possible reason for this is that the quadcopter itself causes a downward airflow in the flight arena. The air is pushed down by the quadcopter, moves back up around the sides of the arena, and flows back down in the centre. This effectively increases the inflow ratio and, therefore, slightly reduces the thrust.

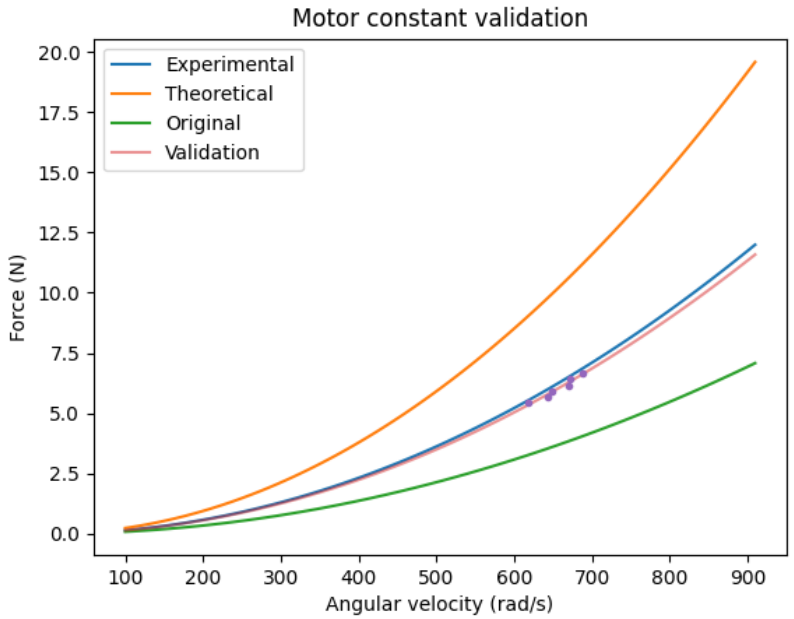


Figure 6.8: Motor constant validation.

6.4.2. Inflow ratio influence

As described in Section 6.1, the inflow ratio can have a significant influence on the thrust produced by the rotors. In the simulator, this is modelled as a linear relationship, defined in Equation 6.6, with the thrust being scaled from full down to 0 at 25m/s of inflow velocity. In this section, an analysis will be made to verify if this is a reasonable assumption.

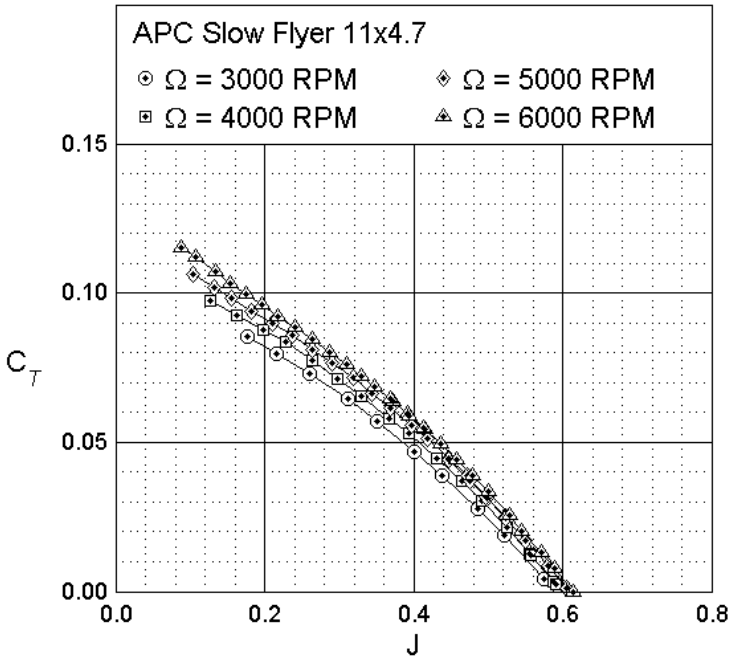


Figure 6.9: Influence of advance ratio on motor constant.

The UIUC database contains not just static performance coefficients but also coefficients that were measured in a wind tunnel and are therefore known at several advance ratios. A plot of this data can be seen in Figure 6.9. This shows that a linear relationship is a reasonable approximation of real-life performance. However, the thrust scalar in the model depends purely on the inflow velocity and not on the actual advance ratio. A calculation can be done to find the maximum velocity near the hover point of 6000 RPM.

The graph uses the airplane variant of the advance ratio, which is as follows.

$$J = \frac{V}{nD}, \tag{6.11}$$

where n is the rotational speed of the propeller in revolutions per second. Therefore, the velocity at which the thrust reduces to zero at 6000 RPM will be as follows.

$$V = nDJ = \frac{6000}{60} * 0.2794 * 0.6 \approx 16.76[m/s] \tag{6.12}$$

This indicates that the thrust will decrease faster than the model predicts. In the performed flight tests, it was found that the maximum velocity the quadcopter was able to reach in the limited space was about 5m/s. Additionally, only a part of that speed will be along the propeller axis as inflow velocity. This can be used to make a rough estimate of the maximum inflow ratio as $J = \frac{3}{(7000/60)*0.2794} \approx 0.092$, which would lead to a thrust reduction of just 10%.

In conclusion, the modelled relationship is simplified but otherwise mostly valid. If more research were to be done in this direction, then higher flight speeds would be required to make the effect more apparent. Since these flight speeds can not be obtained in the available flight space, it was decided not to look into this effect any further.

6.4.3. Experiment validation

The motor constant initially led to unstable behaviour in the simulator. The experiment was therefore repeated in the simulator to ensure that no mistake was made in the calculation of the motor constant. This experiment consists of a simulation of the original motor constant experiment. The Gazebo model has been modified to add a hinge joint between the floor and the quadcopter. This joint has been set to have a rotation limit of 0 degrees, such that it can not move. Any forces and torques generated by the quadcopter will be applied to this joint. A sensor can then be added to this joint, using a force torque sensor plugin with ROS integration from the ‘gazebo_plugins‘ package[1].

This setup makes it possible to measure the generated thrust over a range of RPMs and then process the measured data in exactly the same way as the original experiment. The motor constant calculated from this simulated experiment is the same as the motor constant put into the model SDF file. This definitively validates that the motor constant is correct since the experimental data had already been validated, and this experiment validates that the calculation of the motor constant from this data is also correct.

The cause of the simulator instability, therefore, can not be the motor constant itself but has to be caused by a different effect that is simulated incorrectly. Investigation into this issue eventually showed that this was caused by an incorrect RPM curve that overestimated the RPM that the quadcopter would reach. This did not lead to an issue in the original model, as the overestimated RPM was compensated for by having a too-low motor constant. The issue was resolved once the identification experiments from Chapter 4 had been performed.

6.5. Discussion

The theoretical approximation, unfortunately, did not lead to an acceptable motor constant value. It is about 60% higher than the value found from the load cell experiment. This load cell experiment did provide good results, as proven by the validation experiment. The new motor constant is a significant improvement over the original value. The simulated RPM during flight now matches the real-life behaviour much better.

[1] http://wiki.ros.org/gazebo_plugins

7

Torque

7.1. Background

The formula for the moments is again found using blade element theory. The torque is found by integrating the aerodynamic forces on each propeller section, multiplied by a moment arm equal to the radius from the centre axis of that section. This gives a moment about the Z-axis called the torque. This torque is important as it determines the power required for the motor to keep the rotor spinning (Fay, 2001). Bouabdallah, 2007 gives the formula for the torque as

$$\begin{aligned} Q &= C_Q \rho A \left(\Omega R\right)^2 R \\ \frac{C_Q}{\sigma a} &= \frac{1}{8a}\left(1+\mu^2\right)\overline{C_d} + \lambda\left(\frac{1}{6}\theta_0 - \frac{1}{8}\theta_{tw} - \frac{1}{4}\lambda\right) \end{aligned} \tag{7.1}$$

7.2. Model

In the motor model, the moment constant is defined as follows.

$$M_Q = -\epsilon C_Q * F_T \tag{7.2}$$

Where C_Q is the rotor moment constant and ϵ is the turning direction of the rotor, where ϵ is -1 for clockwise rotation and $+1$ for counter-clockwise rotation.

This is again a simplification of the theory presented in Equation 7.1 as it does not take wind speed or quadcopter speed into account.

7.3. Identification

7.3.1. Theory-based identification

The moment constant can also be calculated using a theory similar to the motor constant. In order to perform this calculation, two more coefficients will need to be used, namely the coefficient of torque and the coefficient of power. There are defined as follows by Spakovszky, 2008.

$$C_Q = \frac{Q}{\rho n^2 D^5} \tag{7.3}$$

$$C_P = \frac{P}{\rho n^3 D^5} \tag{7.4}$$

The moment constant, defined in Equation 7.2, is the thrust force divided by the torque. Equation 6.7 and Equation 7.3 can be rewritten to calculate the thrust and torque, respectively, and then filled into the moment constant formula. Performing these steps leads to the following formula.

$$moment\_constant = \frac{Q}{T} = \frac{C_{Q0}\rho n^2 D^5}{C_{T0}\rho n^2 D^4} = \frac{C_{Q0}}{C_{T0}} D \tag{7.5}$$

The only unknown remaining is the torque coefficient. References, such as the UIUC database, will often not list this coefficient, only listing the power coefficient. As noted in Section 7.1, the torque determines the power required for the motor to keep the rotor spinning. This relationship is formulated as follows.

$$P = Q\omega = Q(2\pi n) \tag{7.6}$$

Combining this with the equation for the torque coefficient gives the complete relationship between the torque and power coefficients.

$$C_Q = \frac{Q}{\rho n^2 D^5} = \frac{1}{2\pi}\frac{P}{\rho n^3 D^5} = \frac{C_P}{2\pi} \tag{7.7}$$

From Figure 7.1 it can be read that in our flight regime the power coefficient is about 0.048. This is used to calculate the torque coefficient, and then finally the moment constant.

$$C_Q = \frac{0.048}{2\pi} \approx 0.0076 \tag{7.8}$$

$$moment\_constant = \frac{C_{Q0}}{C_{T0}} D^4 = \frac{0.0076}{0.12} 0.239 = 0.0152 \tag{7.9}$$

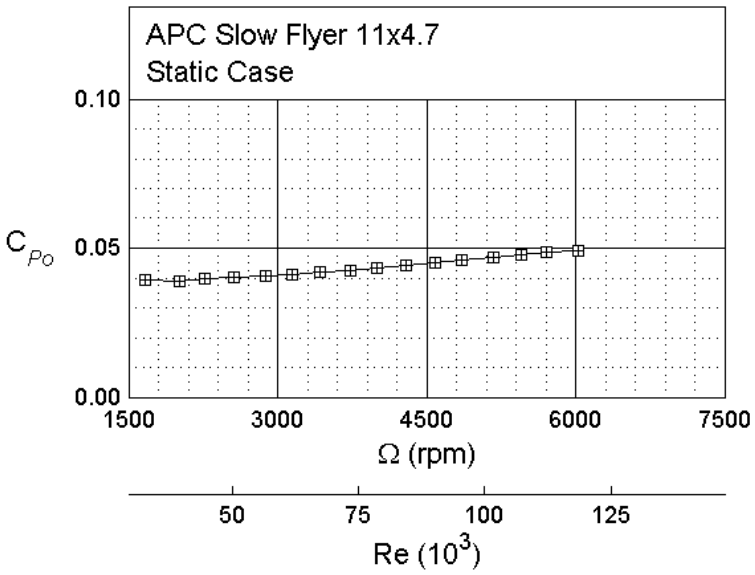


Figure 7.1: APC 11x4.7 power coefficient (Brandt et al., 2022).

7.3.2. Experimental identification

To determine the motor constant, the RPM and resultant torque need to be measured. For this, a setup is used that measures the moment generated by the quadcopter. This setup consists of an arm on which the quadcopter is mounted. The arm can turn horizontally with the hinge below the quadcopter. The other end of the arm pushes on a force sensor, thus measuring the created torque. The quadcopter will have only two of its rotors, which both spin in the same direction, spinning and the other two at idle, as this is when the maximum torque is generated. A picture of the test setup can be seen in Figure 7.2. A more detailed description of the test setup can be found in Section A.4.

Like the thrust, the torque is modelled as a quadratic relationship between the angular velocity and the torque. The data found in the experiment will be used to find $C_Q * C_T$, and therefore needs to be divided by the motor constant to find the moment constant. A plot of the collected data can be seen in Figure 7.3.

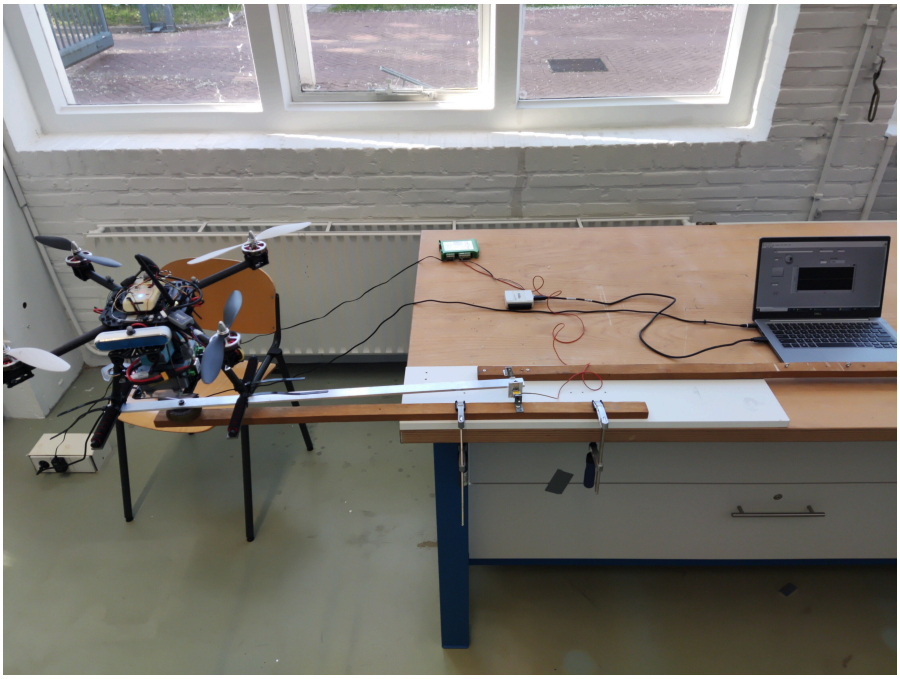


Figure 7.2: Torque setup overview.

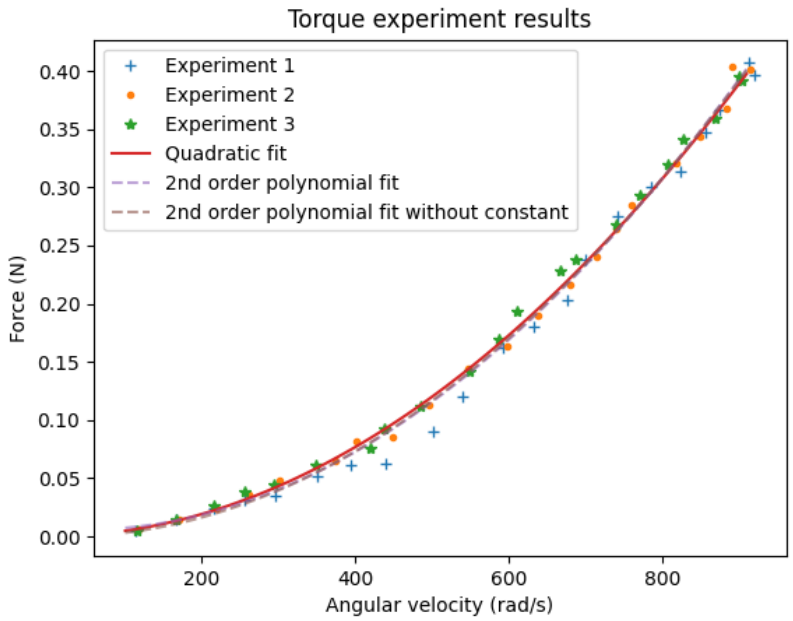


Figure 7.3: Torque experiment results.

7.4. Validation

Validating the moment constant is more challenging than validating the motor constant, as there is no easy way of determining the generated torque in flight. In the thrust validation, the thrust force is counteracted by gravity, which provides accurate knowledge of the generated thrust. When performing a validation experiment for the torque, in which the quadcopter spins around its Z-axis, the produced moment is instead counteracted by a drag force. This drag force is caused by the rotor drag, and the rotor drag coefficient required for this will be identified in the next chapter. A validation of the found moment constant will be performed together with the validation of the rotor drag in Chapter 8. To still gain some insight into how the torque influences the simulator, an analysis of the torque during simulated flights was performed. Additionally, the processing of the experimental data has again been verified using the same method as described in Chapter 6.

7.4.1. Simulator validation

During a flight, the torque generated by the motors should only be around the rotor axis. However, it was found that the used model also simulates a rotor torque around the other axes during flight. According to Equation 7.2, this torque should only be applied along the rotor's body frame. In the code of the motor model, the calculated torque is rotated from the rotor frame into the quadcopter body frame. Due to the implementation of the mathematical physics simulator formulas, the rotor frame does not exactly align with the body frame Z-axis during flight. This connection is via a joint that should only rotate around the Z-axis, but the movement of the quadcopter causes a temporary unaligment of the axes. When two joints are not aligned, a force is applied to bring the two bodies back into alignment (ODE, 2022). This realignment does not happen in a single simulator step. When the torque vector is rotated from the rotor frame to the quadcopter frame, the torque is no longer applied solely around the Z-axis. This vector rotation was added to the code to support arbitrary rotor orientations. As a result, however, it also caused a change for quadcopters that have their rotors aligned with the Z-axis.

A plot of the simulated torque in a trajectory where the quadcopter follows a straight line back and forth is shown in Figure 7.4. The unrotated torque, which acts purely around the Z-axis, is also shown. It can be seen that part of the torque is erroneously applied around the X-axis, and the torque around the Z-axis is, as a result, slightly lower.

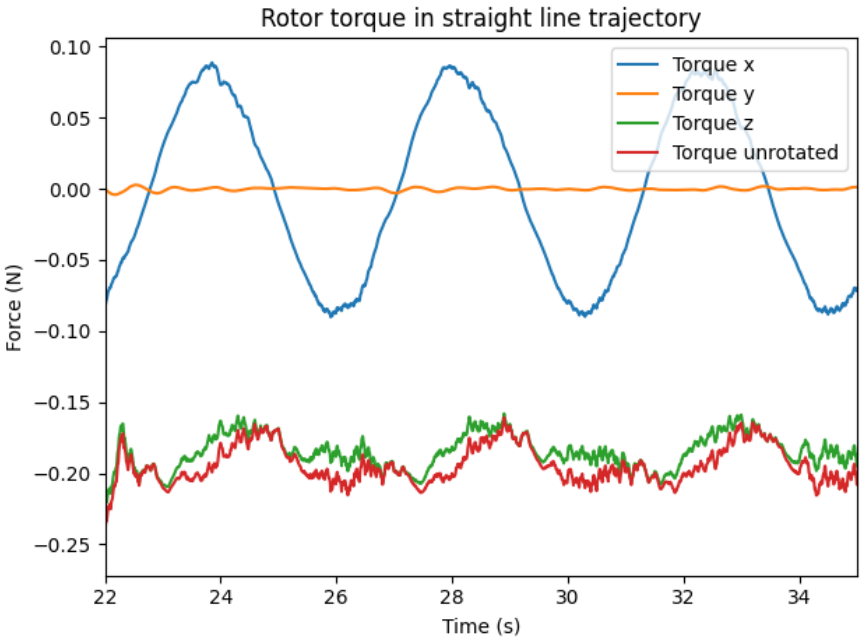


Figure 7.4: Rotor torque in a straight line trajectory.

7.4.2. Experiment validation

The moment constant experiment has been repeated in the simulator using the setup described in Section 6.4.3 to verify that the data processing has been done correctly. The calculated moment constant from this experiment is the same as the moment constant put into the model, verifying that the calculation is correct.

7.5. Discussion

The theory-based identification again produces a number that does not match the one found from the experimental identification. Unlike for the thrust, it was not yet possible to exactly validate the experimentally found number. In the validation process of the torque a bug was found in the motor model, which has a noticeable impact on the flight performance. Validation together with the rotor drag will have to show whether the new moment constant produces realistic flight behaviour.

8

Rotor drag

8.1. Background

The rotor drag, often called H-force or hub force, is a force that lies in the horizontal plane of the body frame. The cause for this force is the drag produced by the rotor. This drag force is applied equally in all directions when there is no air movement on the rotor. As soon as the rotor has a forward velocity, however, the side of the rotor moving into the wind will produce more lift and drag. This means the drag forces no longer cancel each other out and instead result in a drag force vector in the opposite direction of the motion of the rotor. This effect is visualized as D_i in Figure 6.2. Fay, 2001 defines the rotor drag force as follows.

$$\begin{aligned} H &= C_H \rho A \left(\Omega R\right)^2 \\ \frac{C_H}{\sigma a} &= \frac{1}{4a}\mu\overline{C_d} + \frac{1}{4}\lambda\mu\left(\theta_0 - \frac{\theta_{tw}}{2}\right) \end{aligned} \tag{8.1}$$

The rotor drag force increases with both the advance ratio and inflow ratio, and as expected, the rotor drag force will be zero when there is no wind velocity perpendicular to the rotor axis.

8.2. Related work

Rotor drag is an effect that only occurs when there is air velocity perpendicular to the rotor axis, which makes this effect difficult to test in a lab setting. There are two main ways to measure rotor drag. The first option is to place the rotor in a wind tunnel, as done by Gill and D'Andrea, 2017, which gives precise rotor drag data for rotor thrust at any angle of attack and wind velocity. No quantitative conclusions about our rotor drag coefficient can be drawn from this wind tunnel data since the used propeller is not comparable. The data can be used to make a qualitative analysis of the rotor drag behaviour and thereby get a better understanding of how accurate our simplified model is. The first observation is that the obtained measurements prove that rotor drag is zero when there is no velocity perpendicular to the rotor axis. Secondly, it proves that the rotor drag scales roughly linearly with both rotational velocity and air velocity. There is, unfortunately, no available rotor drag data for purely perpendicular air flow. Overall, the data indicates that our simplified model may be sufficient, but at the same time, the data set is too limited to make confident conclusions about the performance of our quadcopter.

An alternative method of rotor drag identification is by performing flight tests. Flight experiments have been performed by several researchers, such as Svacha et al., 2017 and Faessler et al., 2018, who adjusted the rotor drag term in a flight controller to optimize trajectory tracking performance. Since rotor drag becomes more apparent at higher flight speeds, it is important to do such manoeuvres at a decent speed. In practice, the size of the flight path is limited by the dimensions of an indoor drone arena. To reach higher speeds, Faessler et al., 2018 use a racing-inspired drone that is able to achieve speeds of 5 m/s on a small circular trajectory of just 2x2m. Svacha et al., 2017 use a lightweight drone to fly in a straight trajectory at speeds of up to 8 m/s, flying a much larger trajectory than comparable research. Gill and D'Andrea, 2017 use a different approach in which the drone is tethered with a rope

on one side. This tether provides the centripetal force required to fly a circular trajectory, allowing the quadrotor to achieve high speeds exceeding 14 m/s.

Clearly, several approaches exist to reach higher flight speeds. An important advantage of smaller trajectories is that it will be easier to fit into an existing drone arena with a motion capture system. The longer trajectory used by Svacha et al., 2017 does not fit in such a space and only uses the less accurate data from the accelerometer.

All of these approaches determine the rotor drag using an indirect method. The rotor drag term in the controller is adjusted until the flown trajectory matches the target trajectory. Using this same method to determine the rotor drag coefficient will likely be challenging. The trajectory tracking performance is influenced by many factors, several of which exert much larger forces on the quadcopter than the rotor drag.

8.3. Model

The rotor drag force is modelled as follows.

$$F_D = -\Omega C_D \cdot v_A^{\perp} \tag{8.2}$$

Where C_D is the rotor drag coefficient, and $v_A^{\perp}$ is the projection of the body velocity vector v onto the rotor plane. This formula shows that any vertical velocity is not considered for the calculated rotor drag, while Equation 8.1 does have a vertical component in the form of the inflow velocity. Faessler et al., 2018 has done an experiment to identify the rotor drag parameters of a drone in the x, y and z-direction and found no improvement in trajectory tracking performance with a non-zero z coefficient. However, they did find a different coefficient for the x and y direction, which is not possible to use in the current model implementation as it makes no distinction between these axes.

8.4. Experimental identification

This research aims to perform system identification without relying on expensive and specialized equipment, such as wind tunnels. It was, therefore, decided not to perform a wind tunnel test to identify the rotor drag. Using the comparison of flight trajectories for identification is the remaining option. The related work in this area performs relatively difficult flight trajectories in which the overall performance is influenced by too many factors to identify the rotor drag accurately. A slightly different method of identification will need to be found.

Ideally, an experiment would be performed where only the rotor drag coefficient influences the flight behaviour. Due to the reliance on horizontal velocity, this is unfortunately not completely possible, but a method was found where most other variables can be eliminated.

The experiment shares similarities with the hover experiment that was performed to validate the motor constant. A flight will be performed where as many factors as possible are kept constant. This is done by making the quadcopter yaw at a constant angular velocity while hovering. The constant angular velocity largely removes the influence of transient effects that may be caused by quadcopter inertia or the motor time constants. The stable hover of the quadcopter also means that the total produced thrust of the quadcopter is known. This leaves two effects that influence flight behaviour, which are the torque and rotor drag.

To rotate at a constant velocity, the quadcopter will slow down two rotors that spin in the same direction and speed up the other two. The two faster rotors will produce a larger torque and thus cause the quadcopter to rotate. When the quadcopter spins, this causes a velocity perpendicular to the rotor axis, which then causes a rotor drag component in the opposite direction. The angular velocity will remain constant as long as there is an equilibrium between the produced torque and rotor drag. A plot of how the simulated forces counteract each other as the quadcopter accelerates to a constant rotational velocity can be seen in Figure 8.1. This data was collected by simulating moving from a stable hover to a constant velocity spin of 2 radians per second. The moment caused by the rotor drag is calculated by taking the absolute value of the rotor drag vector and multiplying that by the moment arm of 25 centimetres. As can be seen in the plot, the torque at first exceeds the drag while the quadcopter is accelerating. The drag moment is zero during hover and increases as the rotational velocity increases. Once the quadcopter approaches a constant velocity, the rate controller will command a lower torque to be generated until the torque and drag moment balance each other out. The variation seen in the

rotor drag moment is due to the small horizontal movements of the quadcopter.

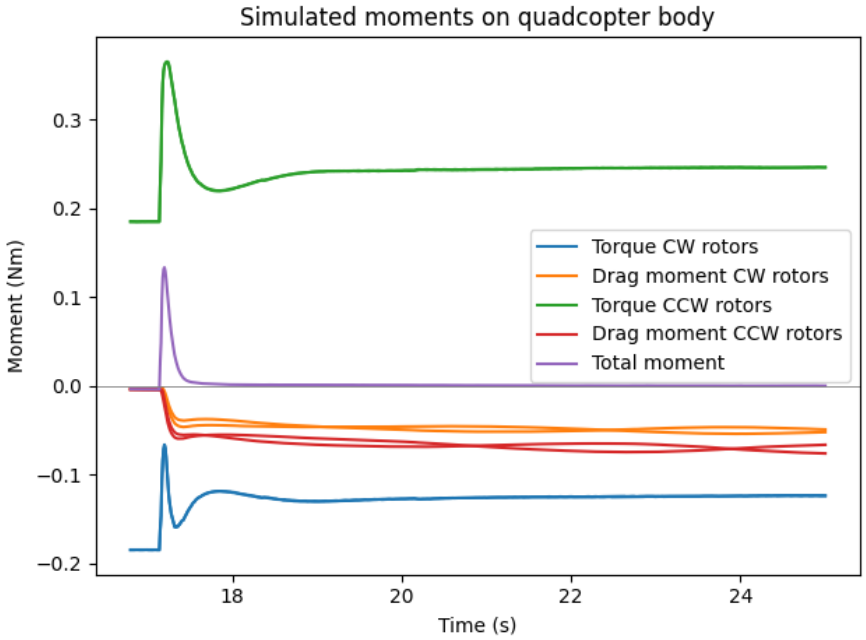


Figure 8.1: Simulated moments on quadcopter during constant angular velocity around the Z axis.

The meaningful conclusion from this equilibrium is that the torque and rotor drag together determine the rotor RPM required to maintain a certain angular velocity. And this relationship can also be used the other way around to calculate the rotor drag coefficient when the torque and rotor RPM are known.

$$M_D = M_{RD}$$
$$C_M * C_T * (\omega_{CW}^2 - \omega_{CCW}^2) = 0.25 * C_D * v_A^{\perp} * (\omega_{CW} + \omega_{CCW})$$

The total drag moment is calculated by subtracting the drag moment created in one direction from the drag moment in the other direction. For the rotor drag, the forces all apply in the same direction, so the rotor velocities are added together. A slight rewrite of this formula gives the equation to calculate the rotor drag coefficient.

$$C_D = \frac{C_M * C_T * (\omega_{CW}^2 - \omega_{CCW}^2)}{0.25 * v_A^{\perp} * (\omega_{CW} + \omega_{CCW})} \tag{8.3}$$

This formula does depend on knowing the rotational velocity of both rotor directions, and the experimentally obtained data only contains RPM data from a single rotor. Luckily the RPM of the other rotors can be calculated since the total thrust at hover is known. A clockwise and counterclockwise spinning rotor together have to support half the weight of the quadcopter.

$$\omega_{CCW} = \sqrt{\frac{0.5 * m * g}{C_T} - \omega_{CW}^2} = \sqrt{\frac{0.5 * 2.24 * 9.81}{1.45e-5} - 599^2} \approx 632[rad/s]$$

Equation 8.3 now contains only known values, which can be filled in to calculate the rotor drag coefficient.

$$C_D = \frac{0.033 * 1.45e-5 * (643^2 - 599^2)}{0.25 * 0.5 * (643 + 599)} \approx 1.25e-04 \tag{8.4}$$

This equation can be repeated with the data from the experiments with a faster rotational speed of 3 and 4 radians per second and rotor velocity measurements of 586 and 571 radians per second, respectively. The resulting drag coefficients are then 1.45e-05 and 1.63e-04, which shows a slight increase in the calculated rotor drag coefficient as the angular velocity increases. A possible explanation

for this is that the quadcopter exhibits more unwanted horizontal drift when rotating at a higher velocity, but the currently available test data can not give a conclusive answer to this discrepancy.

Another factor that may reduce the accuracy of this calculated rotor drag coefficient is that the moment constant has not yet been validated. In the next section, transient flight behaviour will be used to investigate the accuracy of the rotor drag coefficient and moment constant.

8.5. Validation

To validate the found rotor drag coefficient and moment constant, the experiment above has been repeated in the simulator, and the data will be compared based on the rotor RPM and the angular velocity. The performance at a constant angular velocity will be validated first, and then the impact of these coefficients on the angular acceleration will be investigated.

8.5.1. Simulator analysis

The two forces applied to the quadcopter are the thrust and rotor drag. A plot of these values for two different flights can be seen in Figure 8.2. Note that the rotor drag force is along the X-axis of the body frame, and the thrust force is along the Z-axis.

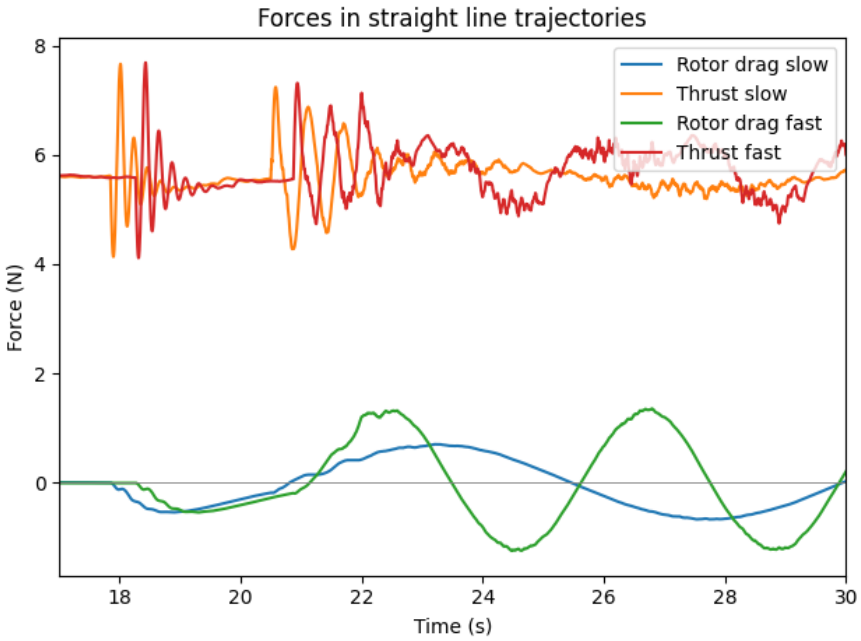


Figure 8.2: Simulated forces in straight line trajectories.

It can be seen here that at slow speeds, the rotor drag force is about 15% of the thrust force. When the trajectory speed is doubled, the rotor drag force becomes more apparent. While the thrust increases by merely 10%, the rotor drag force nearly doubles. The reason that the rotor drag does not completely double is that the rotor drag is based on the air velocity perpendicular to the rotor. To reach higher velocities, the quadcopter will fly at a higher pitch angle, and therefore less of the total velocity will be perpendicular to the rotor axis. As expected, the quadcopter should be flown at an as high as possible velocity to identify the rotor drag coefficient.

8.5.2. Rotor RPM validation

The simulator has been run in three different configurations: with completely original settings, with the coefficients identified in this research but the old rotor drag coefficient, and with all identified coefficients. The results from this experiment can be seen in Figure 8.3. As expected, the original model overestimates the required RPM for both a stable hover and for constant velocity rotation. The newer model correctly estimates the RPM during hover, but the too-high rotor drag coefficient means a higher

difference between the rotors is calculated to maintain the angular velocity. The model with the new rotor drag coefficient correctly estimates the RPM at hover and during the spin.

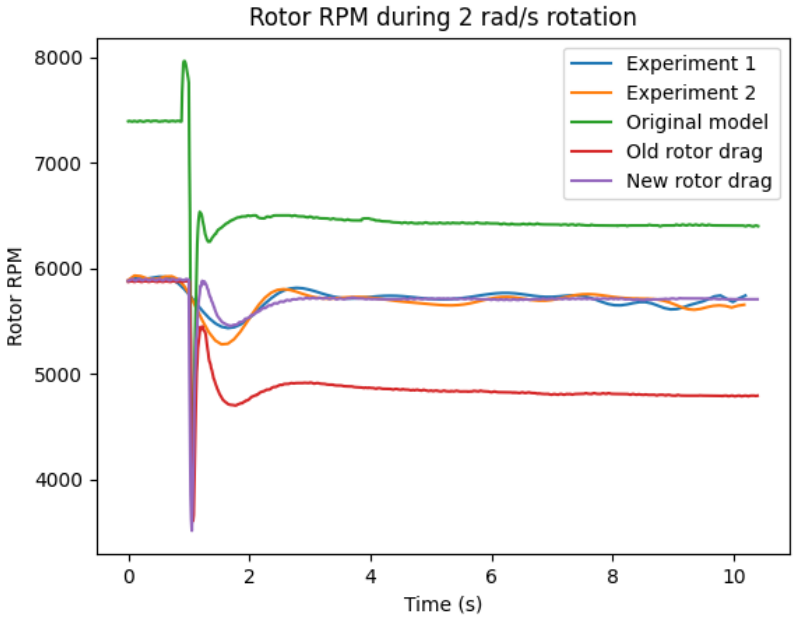


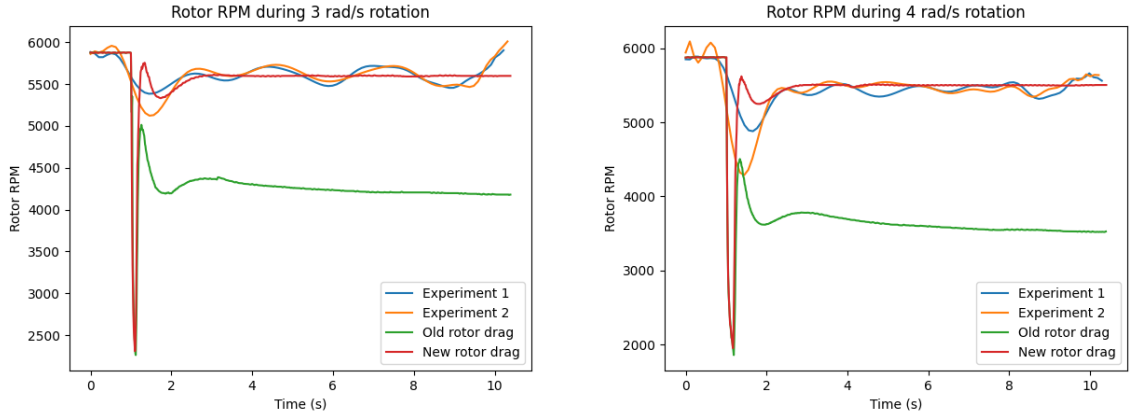
Figure 8.3: Rotor RPM during 2 radians per second spin with different rotor drag coefficients.

Compared to the real-life data, all three models vastly overestimate the initial reduction in rotor RPM. While the real-life data shows a slow decrease in rotor RPM over a period of about half a second, the simulated models almost halve the rotor RPM in a fraction of a second. This is likely due to the use of incorrect time constants, but higher time constants can not be used as they cause oscillations in the simulator quadcopter.

Running the experiment at higher velocities gives comparable results. The plots for these experiments can be seen in Figure 8.4.

8.5.3. Angular velocity validation

In this section, the simulator accuracy will be assessed based on the angular velocity during the spin. The angular rate is controlled by a PID controller, which, in most circumstances, will be able to reach



(a) Spin at 3 radians per second.

(b) Spin at 4 radians per second.

Figure 8.4: Rotor RPM during spin at higher RPMs.

and maintain the desired angular velocity. This analysis will therefore focus on the transient behaviour that happens while the quadcopter is accelerating. When looking at the angular acceleration, more effects influence the performance, such as the RPM curve, time constants and inertia. It is therefore challenging to exactly identify specific coefficients from this data, but it can still be used to examine the influence that specific coefficients have.

A plot of the angular velocity can be seen in Figure 8.5. This plot shows that none of the simulator models follow the real-life curve.

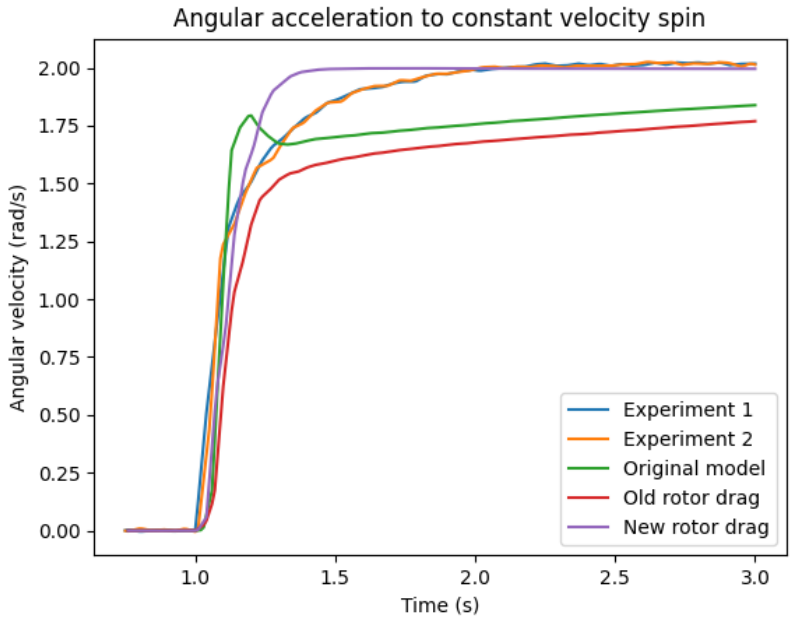


Figure 8.5: Angular acceleration towards 2 radians per second spin.

To understand the behaviour of the different models, some knowledge about the angular rate controller is required. This is a PID controller configured with a proportional term of 0.20, an integral term of 0.10 and a derivative term of 0.0. The original model has a comparatively high moment constant and quickly accelerates towards the desired angular velocity. As it approaches this velocity, the proportional term of the controller decreases while the rotor drag increases. The actually produced torque slightly lags behind the torque requested by the rate controller due to the time constants delay. This leads to the situation where the rotor drag becomes higher than the produced torque, and the quadcopter will decelerate. Over time the integral term of the rate controller becomes larger, and the quadcopter starts to accelerate again. The model eventually reaches the target angular velocity after about 10 seconds.

The updated model, with the old rotor drag, has a smaller moment constant and therefore takes longer to accelerate towards a point where the rotor drag becomes large. At this point, the integral term has increased to the point that the quadcopter will keep accelerating. This model also takes very long to converge to the target velocity.

The new model is able to converge to the target velocity quickly due to the much smaller influence of the rotor drag. This model now shows the opposite problem, where it actually reaches the target velocity faster than it should. This plot gives the impression that there is a combination of the moment constant and rotor drag coefficient where the acceleration accurately follows the curve. To investigate this, the rotor drag coefficient at different moment constants was calculated, and the experiment was repeated with those values. Figure 8.6 shows a plot of these different curves. Each curve has a rotor drag and moment constant combination that produces the correct rotor RPM during a constant spin. The rotor drag identified in this chapter is close to the purple line and will be used to compare the other curves to. This plot shows that increasing the rotor drag, and therefore also increasing the moment constant, causes an even steeper line that reaches the target velocity unrealistically fast. Reducing the rotor drag, and thus also reducing the moment constant, leads to a curve where the initial acceleration is too slow. It is clear that the curve can not be matched by only modifying the rotor drag and moment

constant coefficients.

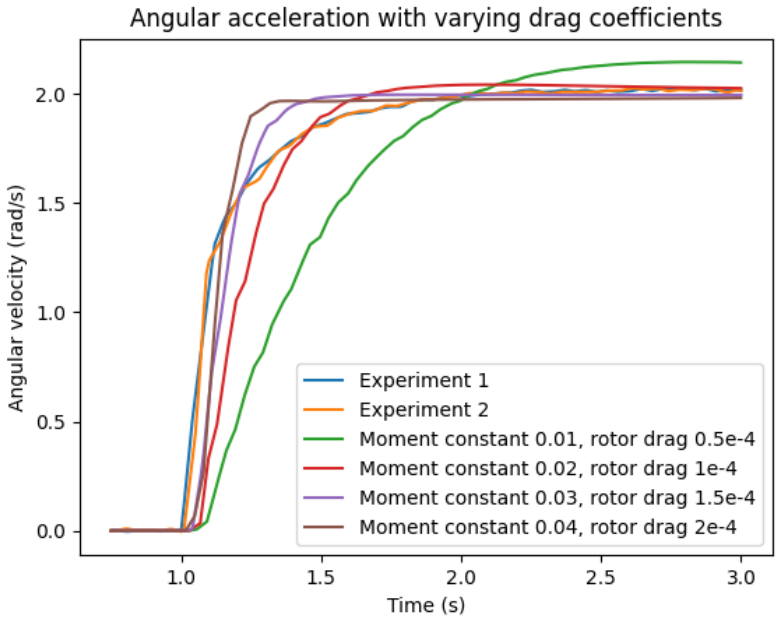


Figure 8.6: Angular acceleration with varying moment constant - rotor drag coefficient combinations.

8.6. Discussion

Without a wind tunnel, it is difficult to isolate the rotor drag from other effects to identify it accurately. Additionally, it makes identifying the rotor drag at different angles of attack virtually impossible. The method used in this chapter gives a rough estimation of the rotor drag coefficient. A validation in the simulator shows a clear improvement over the original model, although there are still discrepancies caused by other factors that influence the results. The current implementation of the time constant is a likely cause for the mismatch in behaviour since, at the start of the manoeuvre, a significant difference in RPM between the real-life data and the simulator is observed.

9

Rolling moment

9.1. Background

In forward flight, there will be wind blowing over the rotor blades. The advancing blade of the rotor is moving against this wind and therefore generates lift based on the wind speed plus its own velocity. Because of this, the advancing blade will see a higher wind speed than the retreating blade. This wind speed difference causes a difference in lift, which results in a rolling moment. This moment, therefore, only exists when the quadcopter has a horizontal velocity, and the moment becomes larger as the speed increases. Bouabdallah, 2007 defines the rolling moment as follows.

$$\begin{aligned} R_m &= C_{R_m} \rho A \left(\Omega R\right)^2 R \\ \frac{C_{R_m}}{\sigma a} &= -\mu \left(\frac{1}{6}\theta_0 - \frac{1}{8}\theta_{tw} - \frac{1}{8}\lambda \right) \end{aligned} \tag{9.1}$$

Like the rotor drag, the rolling moment increases with horizontal velocity and is zero during hover.

9.2. Related work

It was found that not much research has been done on the rolling moment. This makes it unclear how significant the effect is on the flight of a quadcopter. Kolaei et al., 2018 did wind tunnel tests on a single rotor to measure the rolling moment using load cells. It was found that there was a rolling moment for angles of attack between -30 and 15 degrees. Outside of this range, the rolling moment was very minimal. Even within this range, the rolling moment exhibits significant nonlinear effects, which make our model implementation a rough estimation at best. For more extreme flight manoeuvres at higher angles of attack, the estimation is wrong and should approach zero instead. It is important to note that so far, we have only considered the rolling moment on one single rotor. Kopyt et al., 2020 note that because the front-left rotor spins in the opposite direction from the front-right rotor, the rolling moments of the rotors will cancel each other out. Therefore, a rolling moment on the body will only occur when the wind speed over the rotors is not equal. This will only happen during a turn in an indoor environment without wind. Misiorowski et al., 2019 did a computational analysis of the airflow over an entire quadcopter in flight. At a speed of 10 meters per second, it was found that the rear rotors experienced a decrease in thrust of about 19%, while the rolling moment of these rotors increased by about 9%. No literature was found that verifies these results experimentally or attempts to model them.

In conclusion, further research is required to more accurately assess the influence of the rolling moment, create a better model of the effect, and find ways to identify the associated rolling moment coefficient correctly. Additionally, it is important to note that the generated rolling moments of the different rotors essentially cancel each other out, making it likely that the total rolling moment is a relatively insignificant effect. An analysis of the importance of the rolling moment will be made in this chapter to find the relevance of this effect.

9.3. Model

The rolling moment is modelled as follows.

$$M_R = \Omega C_R * v_A^{\perp} \tag{9.2}$$

Where C_R is the rolling moment coefficient. Compared to the theory presented in Section 9.1, this is again a simplified model that does not take the vertical velocity into account. However, the inflow ratio term in Equation 9.1 is much smaller than the advance ratio term, which makes sense considering the reasoning behind the rolling moment. Little research has been done on the importance of the rolling moment. Misiorowski et al., 2019 did look at the effects of propeller wake on the quadrotor. Propeller wake is the vortices created by the rotor that trail behind it. These vortices disrupt the airflow and therefore reduce or increase the thrust of other rotors, which is an effect that is not modelled at all. The default value for the rolling moment coefficient in the model is $1e-6$, which indicates that this is an approximation and not a measured value. No other literature that uses the framework has changed the value of this constant. Further research will have to be done to identify the importance of this value on the accuracy of simulations.

9.4. Simulator analysis

To gain a better understanding of the rolling moment, the influence that it currently has on the simulator is analysed. Like the rotor drag, the rolling moment is dependent on wind velocity perpendicular to the rotor axis, and it will therefore increase in magnitude with increasing speed in the same way. Unlike the rotor drag, however, the rolling moment torques depend on the turning direction of the rotor. A plot of the rolling moment torques when the quadcopter is flying back and forth along a straight line can be seen in Figure 9.1. To perform this manoeuvre, the quadcopter only has to pitch up and down. This is done by changing the RPM of either the front two rotors or the back two rotors at the same time. Since adjacent rotors have opposing turning directions, the generated rolling moment is essentially cancelled out.

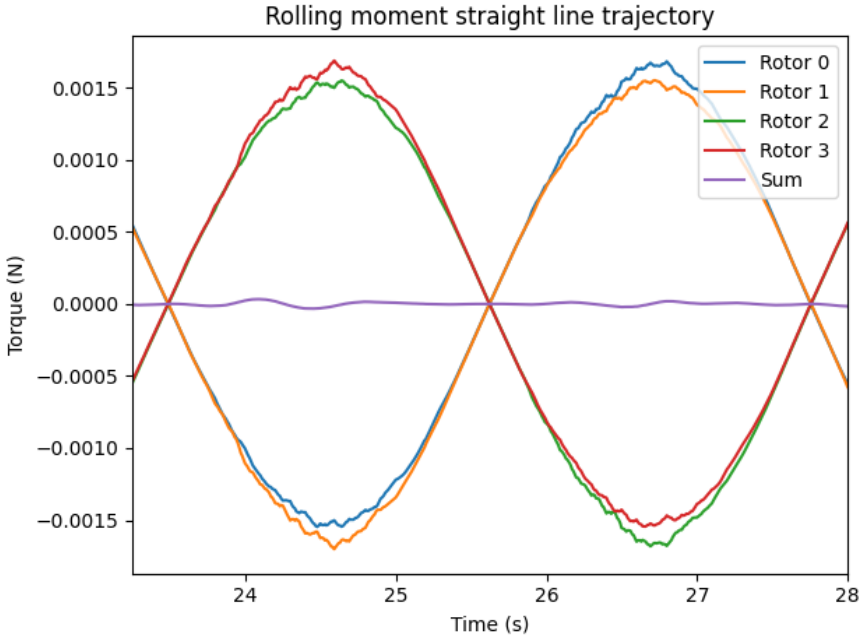


Figure 9.1: Rolling moment torque in a straight line trajectory.

When tracking more complicated trajectories that include yawing motions, there will be a difference in the rotor velocities. A plot of the rolling moment torques when flying a lemniscate trajectory can be seen in Figure 9.2. The generated torques no longer cancel each other out perfectly and will therefore lead to a resultant torque on the quadcopter body. This torque is still several orders of magnitude smaller than the other forces and torques applied on the quadcopter.

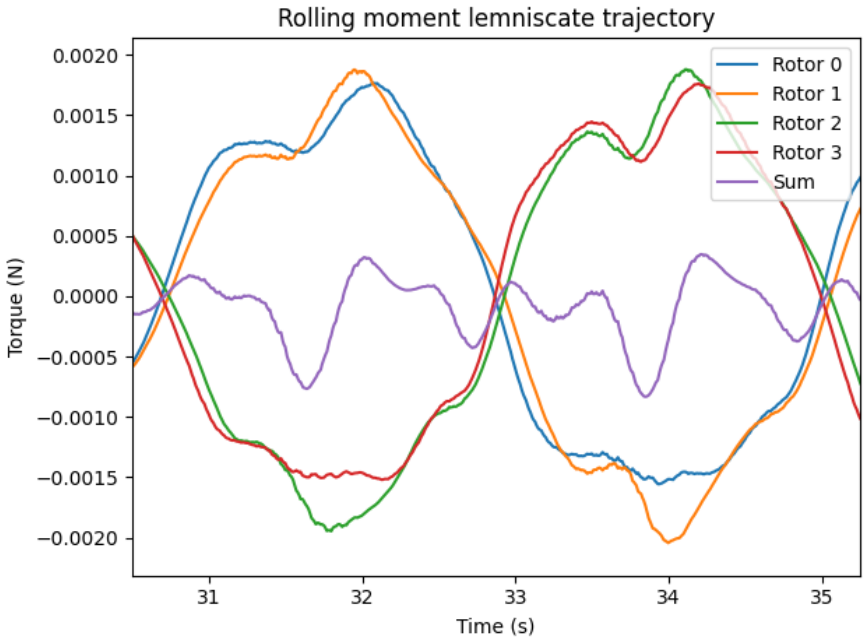


Figure 9.2: Rolling moment torque in a lemniscate trajectory.

9.5. Discussion

The rolling moment is a real and measurable effect within a wind tunnel. Compared to the other aerodynamic effects, however, it has only a very small influence. The simulator currently contains inaccuracies within the simulation of some of these other effects that overpower any moment caused by the rolling moment. It has therefore been decided not to look into the rolling moment any further, as any inaccuracies here will have only a minor influence on the total model accuracy.

10

Model validation

In the previous chapters experiments were performed to identify the system parameters. Theoretically, this identification should lead to a more accurate simulation environment. In this chapter a validation process will be performed to verify that the accuracy of the simulator has improved. First, Section 10.1 will give some background info on how such comparisons have been performed in existing literature. Section 10.2 will then explain some of the factors that make this validation process challenging. Next, Section 10.3 describes the metrics used to judge the simulator models. Section 10.4 will quickly describe which coefficients are used in the final model version. Section 10.5 will perform the validation using a qualitative and quantitative analysis of the performance. Finally, Section 10.6 will describe an approach that may simplify the identification and validation process in the future.

10.1. Background

Several papers validate their models by performing flight tests in which the quadcopter performs pitch, yaw and roll movements from a hover. Such movements can be used to validate things such as the modelled thrust, torque, inertia or motor response. Partovi et al., 2012 and Minervini et al., 2021 perform this validation using data from the onboard IMU, and Capello et al., 2015 uses motion capture data for a similar experiment. Each of these papers defined a simplified quadcopter model, which does not take relative wind velocities into account. While these papers perform model validation, their approach does not validate the model in realistic flight scenarios. Most notably, all validation experiments are performed around hover with negligible velocities. As a result, the effect of omitting relative wind from the models is not analyzed.

Faessler et al., 2018, on the other hand, flies several different trajectories in order to tune model parameters. A comparison is made between the flight path requested by the controller and the flight path flown by the quadcopter. Rotor drag is part of the model, and its coefficient is one of the parameters being tuned. This research does not validate a simulator model, but instead tunes a controller during real-life flights.

Finally, Gill and D'Andrea, 2017 used flight data to identify the model coefficients. Their approach could use be used to instead validate existing model coefficients, however they only model the relationship between rotor rotational velocity and the aerodynamic forces and torques.

To sum up the findings of this section, model validation using flight data is a proven concept, but none of these papers perform the full validation. They either perform their validation at such a low velocity that drag does not yet have an influence, or do not consider the motor dynamics.

Faessler et al., 2018 used a similar approach, although the flight tests here were done to tune a controller instead of to validate a model. In this research, The data is collected using an OptiTrack motion capture system and analyzed based on the quadcopter position.

Most literature performs its validation near hover, which means that several aerodynamic effects such as rotor drag and rolling moment are not relevant. For a proper validation of the complete model, a flight path should be executed at a higher speed where such effects play a larger role. The other differences between the validation techniques come from whether internal IMU data or external motion capture data is used, and the specific parameters used for the validation. As noted by Minervini et al.,

2021 the use of IMU data can introduce noise or a bias in the captured data. Therefore, a motion capture system is preferable. The data is generally compared based on angular rate, angular acceleration, or a combination of both.

10.2. Complicating factors

There are several effects that, especially when combined, significantly complicate all comparisons. This section will give an overview of these effects and discuss potential ways to mitigate them.

Nondeterministic behaviour

By its nature quadcopter flights are non-deterministic, which means that a trajectory will never be flown in exactly the same way if it is repeated. A large cause of this is that onboard pose estimation is notoriously difficult, and variations in the measured pose will lead to different controller commands. The use of OptiTrack is essential to improving this pose estimation, but there will always be a short delay before the pose measured by OptiTrack reaches the drone to correct its internal pose estimation. A second factor is variations in wind velocity and turbulent air. The flights are done indoors, but the quadcopter itself produces a large amount of wind. This wind will then bounce off of the floor and walls back to the quadcopter and influence its behaviour.

When comparing trajectories, the comparison should always be made based on several flights, as only comparing two flights can often lead to incorrect conclusions.

Controller compensates for dynamics

During a flight, there are three controllers at work simultaneously. The geometric controller is based on Faessler et al., 2018, the geometric attitude controller is based on Lee et al., 2010, and the angular rate controller is from PX4 (PX4, 2022a). Jaeyoung, 2019 combined the separate controllers into one package for use with PX4.

A diagram of the controller architecture can be seen in Figure 10.1. The position, velocity and acceleration setpoints, $\mathbf{X}_{sp}$, $\mathbf{V}_{sp}$ and $\mathbf{A}_{sp}$, are calculated by the trajectory publisher based on the time since the start of the trajectory. These setpoints are sent to the position controller, where the position and velocity errors are used to calculate the desired acceleration vector. The desired acceleration vector, $\mathbf{A}_{des}$, is calculated and sent to the geometric attitude controller. This controller, which is based on Lee et al., 2010, computes an angular body rate setpoint vector, $\mathbf{\Omega}_{sp}$, and a normalized thrust setpoint, T_{sp}. The angular rate controller consists of three separate proportional–integral–derivative (PID) controllers for the roll, pitch and yaw body rates. It calculates a normalized roll, pitch and yaw setpoints, R_{sp}, P_{sp}, Y_{sp}, which are finally combined in the mixer to calculate the thrust setpoint vector $\mathbf{T}_{sp}$ for the motor controllers.

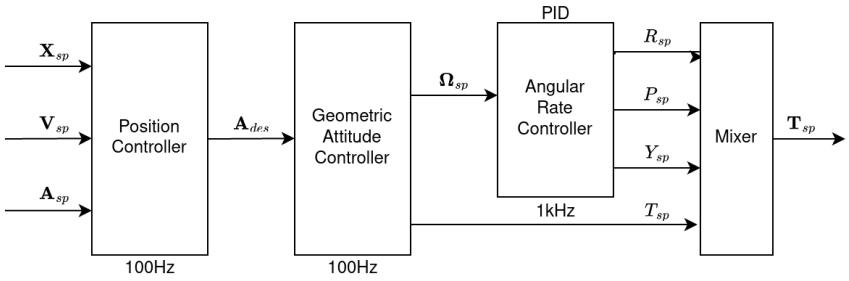


Figure 10.1: Flight experiments controller architecture.

Quadcopters are inherently unstable and cannot fly without such electronic controllers. For our validation experiments, this is an issue, as the controllers will attempt to compensate for the changes made to the dynamics model. This leads to a situation where it is impossible to directly measure the difference in quadcopter pose caused by a change in the model. Instead, what is being measured is the ability of the quadcopter controller to track the requested pose, despite any model changes that might make this task more difficult.

Two workarounds were used to get better data for the validation process. The first is to compare parameters that are not directly being optimized for by the controllers. For example, the quadcopter

will still be able to fly most trajectories in the simulator when the motor constant is reduced by 20%. A comparison of the rotor RPM will then reveal that the controller has increased the RPM to compensate for the lower motor constant. The second workaround is to fly more aggressive trajectories. This has a double advantage of both reducing the time the controller has to compensate for the dynamics and simultaneously also increasing the forces that the controller has to compensate for. All modelled forces scale with either the rotor RPM, velocity perpendicular to the rotor axis, velocity parallel to the rotor axis, or a combination of these. Therefore, a trajectory should always be chosen in such a way that the coefficient that is being validated has an as high as possible influence.

Interdependence of variables
In an ideal situation, the coefficients could all be validated independently, and a coefficient modification could immediately be analysed to see if it is an improvement. In practice, this is unfortunately not the case, as all modifications have an effect on the same system. This leads to the situation where setting a coefficient to the correct value can initially worsen the simulator's accuracy until other parts of the model are also modified.

To deal with this interdependence, the different coefficients should be identified in situations where other effects are either zero or as low as possible.

10.3. Trajectory comparison metrics

Before a validation process can be chosen, it is necessary to precisely define what improvements in the simulator have hopefully been achieved by the system identification. So far, this improvement has been described in one word; accuracy. An overview will be given in this section of the exact simulator results that will be looked at. For all options, the comparison will be made based on whether the updated simulator is more similar to the real-life results than the original simulator.

Position error
Likely the most important result for a researcher will be whether the simulated quadcopter position during a flight now more closely matches the position in a real-life flight. An incorrectly predicted position for a flight manoeuvre could lead to a collision with nearby objects, and it is, therefore, vital that this error is as low as possible. A downside of comparing based on position error is that the used controller tries to track a position and modify thrust and orientation to achieve this. As a result, a large difference in the simulated dynamics may only lead to a small difference in position error.

Orientation error
Similar to the position error, a comparison can also be made to verify that the simulated orientation matches the real-life performance better. As mentioned, the geometric controller is, at its core, a position-tracking controller. It is, therefore, likely that a model dynamics change is more noticeable in the orientation.

Rotor RPM
A comparison can also be made using the rotor RPM. A change in the model dynamics will likely also cause a change in the rotor RPM. A downside, however, is that the RPM can only be measured of one rotor at a time. When flying more complicated trajectories where the rotor RPMs vary a lot, this may give an incomplete picture of the actual situation.

PWM output command
As the final option, a comparison could be made by comparing the PWM command sent to the motors. This is similar to capturing the RPM but instead captures the data before the time constants and RPM curve are applied.

10.4. Final model

The coefficients used in the final model, and the original values that they had, can be seen in Table 10.1. The only identified values that were not used are the time constants, as they made performance worse. The rolling moment coefficient has been left at its original value, but as noted before this moment is so small that it has practically no influence on the model.

Table 10.1: Complete list of old and new model coefficients.

	Original	Updated
Mass (g)	2079	2243
I_{xx}	2.10E-02	2.8E-02
I_{yy}	2.10E-02	3.3E-02
I_{zz}	4.00E-02	3.8E-02
Motor input scaling	1000	950
Motor zero position	100	100
Time constant up	0.125	0.125
Time constant down	0.025	0.025
Motor constant	8.55E-06	1.45E-05
Moment constant	0.06	0.033
Rotor drag coefficient	8.06E-04	1.45E-04
Rolling moment coefficient	1.00E-06	1.00E-06

10.5. Complete model validation

Finally, an analysis will be made on the accuracy of the complete model with all changed coefficients. The complete list of used coefficients can be seen in Table 10.1. This analysis will consist of a qualitative part where the model's behaviour is analysed, as well as a quantitative part in which metrics based on two data sources are used to judge the performance. Rotor RPM and PWM output messages were eventually not used within certain metrics. The captured RPM data turned out to have a too-low message rate to create a meaningful comparison. The PWM output command was, by accident, not captured in the test flight data. Instead, the RMSE of the position and orientation of each axis is used as a metric. A detailed description of the setup used to perform flights in the lab can be found in Section B.1, and the equivalent simulator components can be seen in Section B.2. A description of the software used to fly and compare trajectories is given in Appendix C

10.5.1. Circle trajectory

The first comparison trajectory is a circle pattern flown at a medium speed. The X position of the different trajectories can be seen in Figure 10.2, and the pitch angle can be seen in Figure 10.3.

The difference between the old and new model is likely caused mainly by the difference in rotor drag. When looking at the pitch angle plot, it is clear that the old model has to move to a higher pitch angle to keep up with the trajectory. This higher pitch angle will be instructed when the position error increases. It can thus also be seen in the position plot that the old model lags behind the position that it is supposed to be at. The new model, on the other hand, is slightly faster than it should be and therefore makes a larger circle than the real-life trajectory. This behaviour can be seen in Figure 10.4 and Figure 10.5. The new model consistently flies the circle with a larger radius. It is also noteworthy that the real-life trajectory flies a larger radius at the positive Y coordinate than the negative Y coordinate. Such a difference is not present in the simulator trajectories at all. This may indicate that certain effects, such as the rotor drag, do not affect the quadcopter similarly in all directions. Faessler et al., 2018 identified the rotor drag for the X and Y direction separately and found two values that differ by about 60%.

To perform a quantitative analysis, the RMSE of the difference between the real-life and simulated trajectories has been calculated. This is done for the position as well as the orientation errors. The calculated metrics can be found in Table 10.2a. The new model has a much lower error in nearly all categories. Note that the positions are compared based on time. The old model lags behind the real-life trajectory, which causes a large part of the error of the old model.

10.5.2. Lemniscate trajectory

The new model tracks the target trajectory much better than the old model, but it is actually too fast. In the X direction, the new model comes close to the target trajectory while the real-life trajectory cuts the corner. This is the cause for the still relatively high X RMSE shown in Table 10.3. A plot of the quadcopter position during this trajectory can be seen in Figure 10.6. The errors are reduced significantly, but the error in the X-direction still remains quite high. An attempt was made to slow

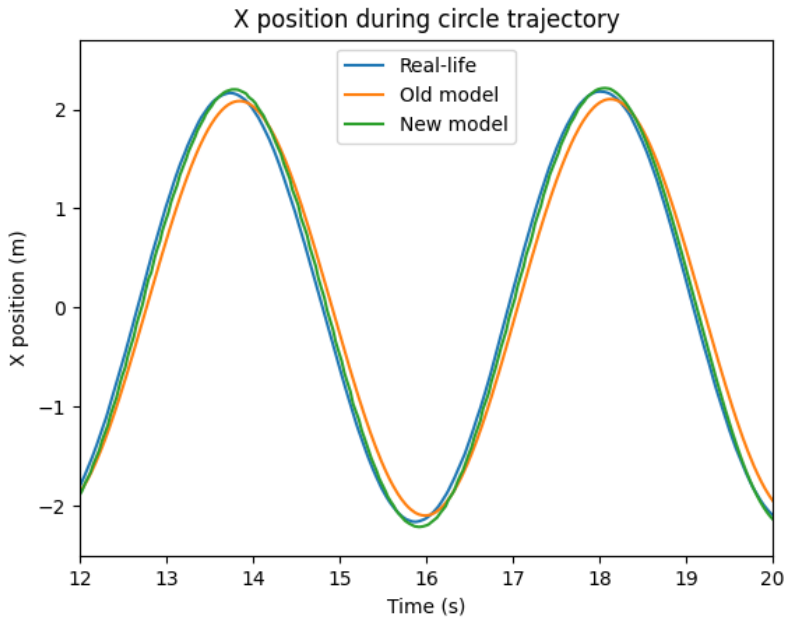


Figure 10.2: X position during circle validation trajectory.

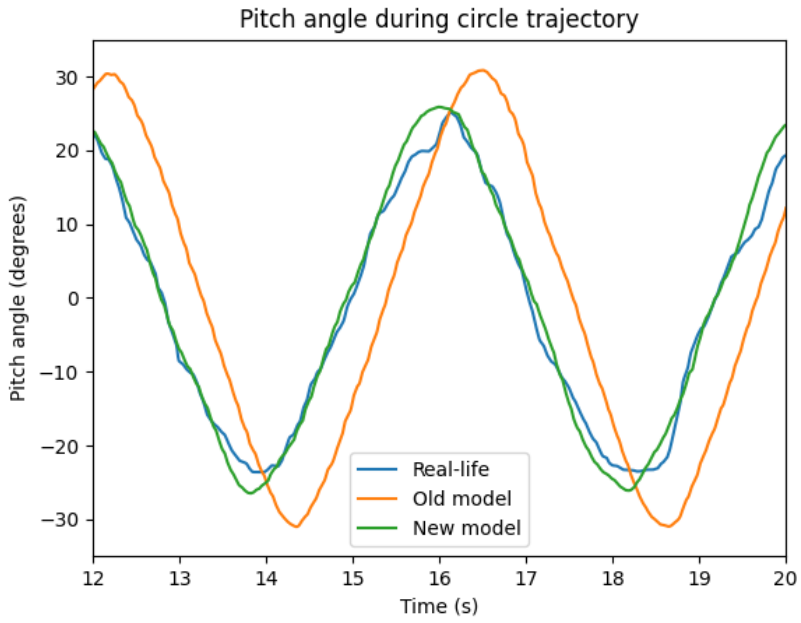


Figure 10.3: Pitch angle during circle validation trajectory.

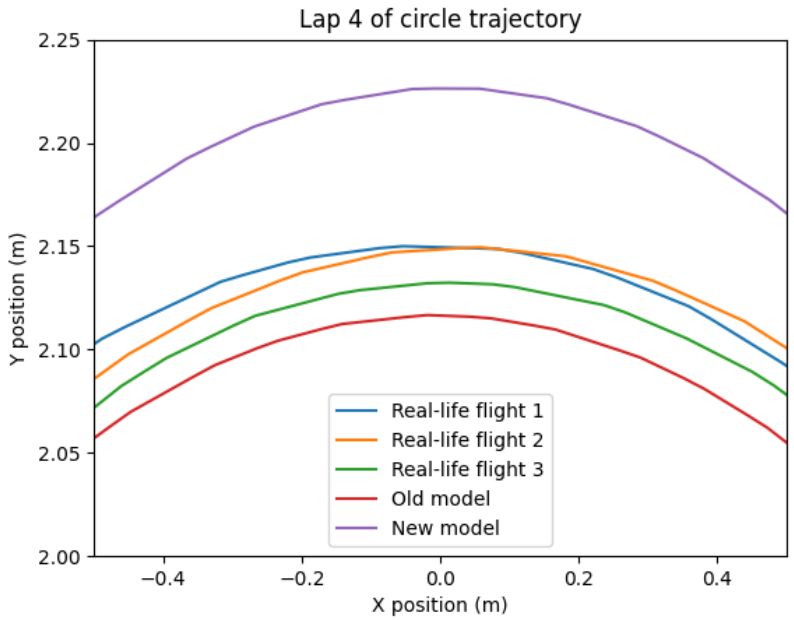


Figure 10.4: Quadcopter position at positive Y coordinate lap 4 of circle trajectory.

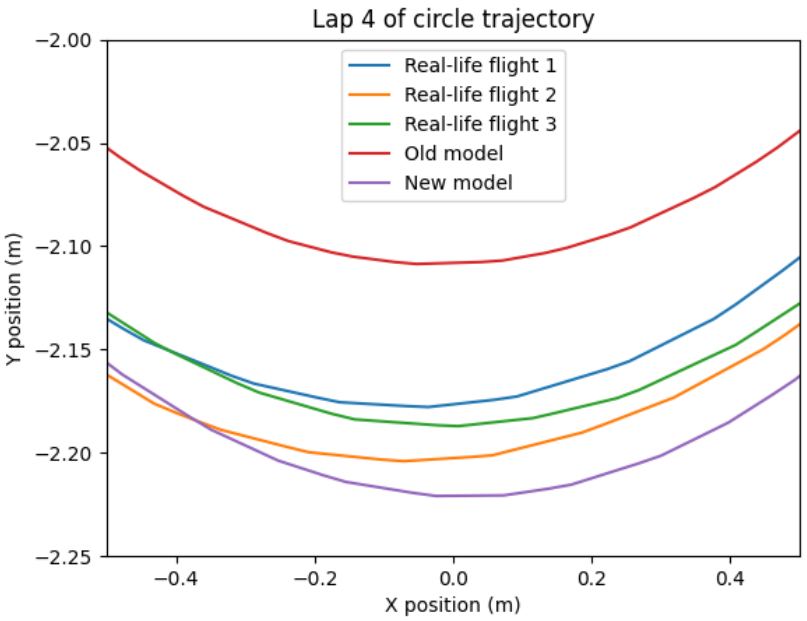


Figure 10.5: Quadcopter position at negative Y coordinate lap 4 of circle trajectory.

Table 10.2: Metrics of circle trajectory at different velocities.

(a) Maximum velocity of $3\,m\,s^{-2}$.

	Old	New
X RMS	0.202	0.046
Y RMS	0.197	0.045
Z RMS	0.037	0.064
Roll RMS	12.501	2.119
Pitch RMS	10.824	3.837
Yaw RMS	5.978	0.989

(b) Maximum velocity of $4.5\,m\,s^{-2}$.

	Old	New
X RMS	0.345	0.137
Y RMS	0.328	0.129
Z RMS	0.049	0.068
Roll RMS	18.950	8.334
Pitch RMS	12.065	5.611
Yaw RMS	14.565	4.768

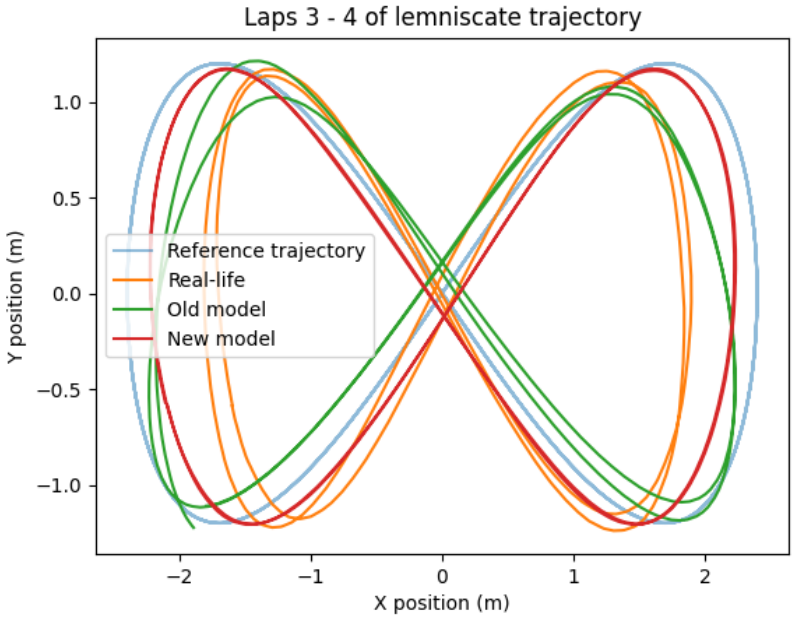


Figure 10.6: Quadcopter position during lemniscate trajectory.

Table 10.3: Metrics of lemniscate with maximum velocity of $4\,m\,s^{-2}$.

	Old	New
X RMS	0.48	0.31
Y RMS	0.55	0.11
Z RMS	0.07	0.07
Roll RMS	32.10	4.87
Pitch RMS	13.60	5.55
Yaw RMS	10.33	2.75

the simulated quadcopter by increasing the rotor drag, but it was not possible to match the real-life trajectory this way. In Chapter 5 it was found that the simulated quadcopter achieves faster angular accelerations than it should, which is a likely cause for the error seen in this trajectory.

10.6. Automated tuning

In order to easily run a large number of experiments an addition to the simulator setup has been made. The motor model has been expanded to support live updates of the model coefficients. This makes it possible to send a set of new coefficients without having to restart the simulator. A script was created to make use of this. When a trajectory ends, the next trajectory is automatically started. Once all trajectories have been flown, a new set of coefficients is sent to the simulator model and the first trajectory is started again. After each trajectory its data is saved to a file, which can immediately be analysed to calculate the metrics. The goal of this is to perform automated tuning in which a large number of coefficient combinations can automatically be tested to find the ones with the lowest metrics. The usefulness of this approach has been limited, because the model implementations for the time constants and RPM curve are not sufficient. Due to this, there is currently no combination of coefficients that would reduce the error much further.

Still, this approach is promising for two separate reasons. First of all, it can be used to make small improvements to the coefficients when they have already been identified using other experiments. But secondly, it can also be used to tune coefficients that have not yet been identified. Until now, performing such tuning in the simulator was far too time-consuming to be a viable approach. Automating this process, therefore, opens up new possibilities. This approach is in some ways similar to the one used in Gill and D'Andrea, 2017. They collected a large amount of flight data at different speeds, and then used a minimization algorithm on that data to find the correct coefficients for a model predicting the thrust, torque and rotor drag. This method could potentially remove the reliance on custom test rigs, such as the ones used for the thrust and torque experiments, and could therefore significantly reduce the complexity of identifying a new quadcopter.

11 Conclusions

This chapter will provide the final conclusions of the research. Section 11.1 will give a short summary of the research. The main research question will be answered in Section 11.2. Section 11.3 will highlight the main contributions of this paper, and finally Section 11.4 talks about the future work to further improve the simulator performance.

11.1. Summary

The goal of this research was to provide a better simulation environment for the NXP HoverGames quadcopter, as well as provide repeatable methods for the identification of alternative quadcopters. To improve the simulator, extensive system identification has been performed of the quadcopter. The motor model consists of two parts; the part concerning aerodynamic effects and the part concerning motor dynamics. The formulas used to represent the four aerodynamic effects are well-founded in the existing literature and properly match the behaviour of the real-life effects in the performed flights. Using system identification experiments, the correct coefficient values have been found, and validation experiments have verified the new values. The motor dynamics formulas, on the other hand, lack a strong theoretical background. Most previous quadcopter research has focused on finding formulas that match rotor velocity with a specific force or torque. This has resulted in a research gap considering motor dynamics. The model implementations of the time constants and RPM curve have been proven to be oversimplified, and this incomplete modelling hampers the simulator's accuracy. A validation process based on trajectory comparison has been used to analyse the performance of the simulator model. This analysis consists of two parts. The first part is a qualitative analysis which provides insights into the impact that the new model has on the simulator performance. The second part is a quantitative analysis that objectively scores the performance of a model on a set of metrics. These metrics are based on the position and orientation error of the quadcopter. This quantitative analysis has proven that the new model performs significantly better than the old model, especially in the case of trajectories with higher accelerations. The tooling developed for this process also makes it possible to perform automated tuning of the coefficients, which is a promising way of reducing the complexity of performing the system identification of a new quadcopter. The importance of these findings for other researchers strongly depends on the type of flights that need to be simulated. The old model underestimated the capabilities of the quadcopter, but it did provide stable flight. When only flying relatively low-velocity trajectories with a maximum velocity of less than $3\,\mathrm{m\,s^{-2}}$, the old model is likely already adequate. When faster trajectories are flown, with a maximum velocity above $4\,\mathrm{m\,s^{-2}}$, the old model becomes so inaccurate that the newly identified model is required to perform useful simulations. There is no exact cut-off point here, as it depends on the type of trajectory being flown as well as the accuracy requirements of the researcher. A 20 cm position error in the simulation may not matter in most scenarios. However, when simulating flight in enclosed spaces, such as flying between trees in a forest, this difference becomes more important.

11.2. Answering the main research question

The main research question of this research is:

How can the RotorS simulator framework be improved to resemble real drone flight behaviour as much as possible?

This improvement was achieved by performing a system identification of the quadcopter to find the correct coefficient values. In a comparison between a real-life trajectory and a simulated trajectory, the new coefficient values provide a significant reduction in the position and orientation errors. To quantify this result, the RMSE of the position and orientation has been reduced to about a third of the old model. Only the error in the Z axis, which was by far the smallest error in the old model, was not reduced.

11.3. Thesis contributions

1. Providing a global overview of where the errors in quadcopter simulations can come from. This research has mapped the differences between real-life and simulator setups. Within the simulator setup, a list was made of the components influencing the simulator results. This has led to a better insight into where simulator inaccuracies originate.
2. Performing system identification of the NXP HoverGames quadcopter (including mass, dimension, moment of inertia, time constants, RPM curve and thrust, torque and rotor drag coefficients). A large number of system identification experiments have been performed, and all found coefficients have been validated.
3. Adding knowledge on the factors that influence the RPM curve. The RPM curve model is overly simplified, and knowledge about the factors influencing this curve is lacking. The identification experiments performed in this experiment give new insights into how the output command of other rotors, and the output commands in the previous seconds, can also influence the output RPM. Although the complete model has not yet been identified, these insights are helpful in designing better identification experiments.
4. Using a novel method for identifying the rotor drag coefficient via an equilibrium between the torque and rotor drag. A new method of identifying the rotor drag has been performed, in which a wind tunnel is not required. This reduces the time and equipment required to perform system identification of a new quadcopter.
5. Creating a CAD model of the NXP HoverGames quadcopter. In order to validate the inertia, a CAD model was created. This model is now available to other researchers.
6. Upgrading the simulator setup to enable automated tuning. The simulator setup has been expanded to allow changes to the motor model coefficients at any time. This makes it possible to automate the coefficient tuning process, which may lead to easier system identification methods.

11.4. Future work

While the simulator accuracy has been increased, there are still further improvements possible. This section will discuss the limitations of this research and recommend steps that should be undertaken to reduce the inaccuracies still present in the simulator. These steps are roughly ordered from most important to least important.

Time constants

The time constants implementation is the only part of the model that does not behave like real-life behaviour in any scenario. Figure 8.3 showed a large discrepancy between real-life RPM and simulated RPM in a simple flight scenario. As a result of this discrepancy, the simulated quadcopter performs faster angular accelerations than its real-life counterpart. This is a likely cause for the oversheet seen in flight trajectories, such as in Figure 10.6.

RPM curve

The RPM curve is a significant simplification of real-life behaviour. The linear curve chosen in this thesis is a reasonable approximation for most scenarios but will not hold up in all cases. For example, a longer flight will cause the voltage to drop over time. This is not modelled and will cause a significant error in the estimated RPM. A better model that properly captures all factors influencing the rotor RPM must be found.

Use more metrics for validation process
As mentioned in Section 10.3, the rotor revolutions per minute (RPM) and pulse-width modulation (PWM) output messages are potentially useful in metrics that judge the simulator performance. This data was not available at a high enough message rate from the experimental data but could be properly captured if the experiment was repeated.

Test higher velocities
Based on the theory, the aerodynamic effects should be influenced by the advance and inflow ratios. This is, besides a scalar on the thrust, disregarded in the quadcopter model. In the experiments in this research, these effects were not visible, but this is likely because the velocities were too low. The size of the used quadcopter arena makes faster flights impossible. A larger area to fly in would enable the verification of the high-speed performance of the simulator.

Simplify identification experiments
The identification experiments performed in this experiment are still quite time-consuming. Ideally, the whole identification process would be automated as much as possible. The thrust and torque tests are particularly complicated, as they rely on custom test setups and use specialized hardware to measure the loads. Identifying these coefficients via flight experiments can significantly simplify the process. The trajectory comparison approach outlined in this research opens the possibility of performing automated coefficient tuning in the simulator. The reliance on the load cell experiments may potentially be removed using this approach.

Migrate to Ignition Gazebo
The version of Gazebo used in this thesis is nearing the end of its life, and a newer version with major architectural changes is replacing it. The quadcopter simulator model has already been ported over to this newer Gazebo version[1]. The old Gazebo version will be supported until 2025, so the migration will need to be performed before thenGazebo, 2022.

[1] https://github.com/gazebosim/gz-sim/tree/gz-sim7/src/systems/multicopter_motor_model

References

Aich, S., Ahuja, C., Gupta, T., & Arulmozhivarman, P. (2014). Analysis of ground effect on multi-rotors. *2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE)*, 236–241. https://doi.org/10.1109/ICECCE.2014.7086619

Balaji, J. (2017). Electric Motor Propulsion, 42.

Bangura, M., Melega, M., Naldi, R., & Mahony, R. (2016). Aerodynamics of Rotor Blades for Quadrotors [arXiv: 1601.00733]. *arXiv:1601.00733 [physics]*. Retrieved November 24, 2021, from http://arxiv.org/abs/1601.00733

Baris, E., Britcher, C. P., & Altamirano, G. (2019). Wind Tunnel Testing of Static and Dynamic Aerodynamic Characteristics of a Quadcopter. *AIAA Aviation 2019 Forum*. https://doi.org/10.2514/6.2019-2973

Belatti, T. (2018). Quadrotor flight in constrained environments. Retrieved March 11, 2022, from https://sunfest.seas.upenn.edu/wp-content/uploads/2018/07/12-belatti.pdf

Benders, D. (2020). AR.Drone 2.0 state estimation using Dynamic Expectation Maximization: Bringing brain perception theory to practice. Retrieved December 3, 2021, from https://repository.tudelft.nl/islandora/object/uuid%3A156157c6-d7f0-4dc1-a55a-b2e4ed66f1c2

Bernard, D. D. C., Riccardi, F., Giurato, M., & Lovera, M. (2017). A dynamic analysis of ground effect for a quadrotor platform. *IFAC-PapersOnLine*, *50*(1), 10311–10316. https://doi.org/10.1016/j.ifacol.2017.08.1500

Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying* (Doctoral dissertation). EPFL. Lausanne. https://doi.org/10.5075/epfl-thesis-3727
Jean-Daniel Nicoud, Peter Corke, Philippe Müllhaupt

Brandt, J., Deters, R., Ananda, G., Dantsker, O., & Selig, M. (2022). UIUC Propeller Database, Vols 1-4. https://m-selig.ae.illinois.edu/props/propDB.html

Brescianini, D., Hehn, M., & D'Andrea, R. (2013). *Nonlinear Quadrocopter Attitude Control: Technical Report* (tech. rep.) [Medium: application/pdf,Online-Ressource]. ETH Zurich. https://doi.org/10.3929/ETHZ-A-009970340

Burri, M., Bloesch, M., Taylor, Z., Siegwart, R., & Nieto, J. (2018). A framework for maximum likelihood parameter identification applied on MAVs [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21729]. *Journal of Field Robotics*, *35*(1), 5–22. https://doi.org/10.1002/rob.21729

Capello, E., Park, H., Tavora, B., Guglieri, G., & Romano, M. (2015). Modeling and experimental parameter identification of a multicopter via a compound pendulum test rig. *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, 308–317. https://doi.org/10.1109/RED-UAS.2015.7441021

Derafa, L., Madani, T., & Benallegue, A. (2006). Dynamic Modelling and Experimental Identification of Four Rotors Helicopter Parameters. *2006 IEEE International Conference on Industrial Technology*, 1834–1839. https://doi.org/10.1109/ICIT.2006.372515

Deters, R., Kleinke, S., & Selig, M. (2017). *Static Testing of Propulsion Elements for Small Multirotor Unmanned Aerial Vehicles*. https://doi.org/10.2514/6.2017-3743

Dhaybi, M., & Daher, N. (2019). Real-time Estimation of the Inertia Tensor Elements of a Quadcopter Hover Platform [ISSN: 2159-6255]. *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 1347–1352. https://doi.org/10.1109/AIM.2019.8868641

Dobrea, D.-M., & Dobrea, M.-C. (2020). An autonomous UAV system for video monitoring of the quarantine zones, 15.

Faessler, M., Franchi, A., & Scaramuzza, D. (2018). Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories [Conference Name: IEEE Robotics and Automation Letters]. *IEEE Robotics and Automation Letters*, *3*(2), 620–626. https://doi.org/10.1109/LRA.2017.2776353

Fay, G. (2001). Derivation of the Aerodynamic Forces for the Mesicopter Simulation, 8.

Furrer, F., Burri, M., Achtelik, M., & Siegwart, R. (2016). *RotorS—A modular gazebo MAV simulator framework* (Doctoral dissertation). Springer.

Gazebo. (2022). A new era for Gazebo. Retrieved November 3, 2022, from https://www.openrobotics.org/blog/2022/4/6/a-new-era-for-gazebo

Gill, R., & D'Andrea, R. (2017). Propeller thrust and drag in forward flight. *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 73–79. https://doi.org/10.1109/CCTA.2017.8062443

Hoque, M. A., Nurmi, P., Kumar, A., Varjonen, S., Song, J., Pecht, M. G., & Tarkoma, S. (2021). Data driven analysis of lithium-ion battery internal resistance towards reliable state of health prediction. *Journal of Power Sources*, *513*, 230519. https://doi.org/10.1016/j.jpowsour.2021.230519

Jaeyoung, L. (2019). *Mavros_controllers - Aggressive trajectory tracking using mavros for PX4 enabled vehicles* (Doctoral dissertation). https://doi.org/10.5281/zenodo.2652888

Jardin, M., & Mueller, E. (2007). Optimized Measurements of UAV Mass Moment of Inertia with a Bifilar Pendulum. *AIAA Guidance, Navigation and Control Conference and Exhibit*. https://doi.org/10.2514/6.2007-6822

Kolaei, A., Barcelos, D., & Bramesfeld, G. (2018). Experimental Analysis of a Small-Scale Rotor at Various Inflow Angles [Publisher: Hindawi]. *International Journal of Aerospace Engineering*, *2018*, e2560370. https://doi.org/10.1155/2018/2560370

Kopyt, N., Niemiec, R., & Gandhi, F. (2020). Quadcopter Rotor Phasing for Minimization of Aircraft Vibratory Loads, 12.

Krznar, M., Kotarski, D., Piljek, P., & Pavkovic, D. (2018). On-line Inertia Measurement of Unmanned Aerial Vehicles using on board Sensors and Bifilar Pendulum. *Interdisciplinary Description of Complex Systems*, *16*(1), 149–161. https://doi.org/10.7906/indecs.16.1.12

Lee, T., Leok, M., & McClamroch, N. H. (2010). Geometric tracking control of a quadrotor UAV on SE(3) [ISSN: 0191-2216]. *49th IEEE Conference on Decision and Control (CDC)*, 5420–5425. https://doi.org/10.1109/CDC.2010.5717652

Li, Q. (2014). Grey-box system identification of a quadrotor unmanned aerial vehicle [Publisher: Citeseer].

Matus-Vargas, A., Rodriguez-Gomez, G., & Martinez-Carranza, J. (2021). Ground effect on rotorcraft unmanned aerial vehicles: A review. *Intelligent Service Robotics*, *14*(1), 99–118. https://doi.org/10.1007/s11370-020-00344-5

Minervini, A., Godio, S., Guglieri, G., Dovis, F., & Bici, A. (2021). Development and Validation of a LQR-Based Quadcopter Control Dynamics Simulation Model. *Journal of Aerospace Engineering*, *34*(6), 04021095. https://doi.org/10.1061/(ASCE)AS.1943-5525.0001336

Misiorowski, M., Gandhi, F., & Oberai, A. A. (2019). Computational Study on Rotor Interactional Effects for a Quadcopter in Edgewise Flight [Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J058369]. *AIAA Journal*, *57*(12), 5309–5319. https://doi.org/10.2514/1.J058369

ODE. (2022). ODE Manual - Joint error and the Error Reduction Parameter (ERP). Retrieved October 31, 2022, from http://ode.org/wiki/index.php/Manual#Joint_error_and_the_Error_Reduction_Parameter_.28ERP.29

Partovi, A. R., Zong Yao Kevin, A., Lin, H., Chen, B., & Cai, G. (2012). Development of a Cross Style Quadrotor. *AIAA Guidance, Navigation, and Control Conference*. https://doi.org/10.2514/6.2012-4780

Paternoster, A., Loendersloot, R., de Boer, A., & Akkerman, R. (2011). Geometric Optimisation of Hinge-Less Deployment System for an Active Rotorblade. *ASME 2011 Conference on Smart Materials, Adaptive Structures and Intelligent Systems, Volume 2*, 171–177. https://doi.org/10.1115/SMASIS2011-5025

Pegram, J. P., & Anemaat, W. A. (2000). Preliminary Estimation of Airplane Moments of Inertia using CAD Solid Modeling, 2000–01–1700. https://doi.org/10.4271/2000-01-1700

Powers, C., Mellinger, D., Kushleyev, A., Kothmann, B., & Kumar, V. (2013). Influence of Aerodynamics and Proximity Effects in Quadrotor Flight [Series Title: Springer Tracts in Advanced Robotics]. In J. P. Desai, G. Dudek, O. Khatib, & V. Kumar (Eds.), *Experimental Robotics* (pp. 289–302). Springer International Publishing. https://doi.org/10.1007/978-3-319-00065-7_21

PX4. (2022a). Controller Diagrams | PX4 User Guide. Retrieved November 9, 2022, from https://docs.px4.io/main/en/flight_stack/controller_diagrams.html

PX4. (2022b). Flying with Motion Capture (VICON, NOKOV, Optitrack) | PX4 User Guide. Retrieved September 29, 2022, from https://docs.px4.io/main/en/tutorials/motion-capture.html

PX4. (2022c). Mixing and Actuators | PX4 User Guide. Retrieved September 13, 2022, from https://docs.px4.io/main/en/concept/mixing.html

PX4. (2022d). Multicopter PID Tuning Guide (Manual/Advanced). Retrieved October 18, 2022, from https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter.html

Semiconductors, N. (2018). KIT-HGDRONEK66: NXP drone kit. Retrieved April 1, 2022, from https://www.nxp.com/design/designs/nxp-hovergames-drone-kit-including-rddrone-fmuk66-and-peripherals:KIT-HGDRONEK66

Shokry, H., & Hinchey, M. (2009). Model-Based Verification of Embedded Software [Conference Name: Computer]. *Computer*, *42*(4), 53–59. https://doi.org/10.1109/MC.2009.125

Silano, G., Oppido, P., & Iannelli, L. (2019). Software-in-the-loop simulation for improving flight control system design: A quadrotor case study [ISSN: 2577-1655]. *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 466–471. https://doi.org/10.1109/SMC.2019.8914154

Spakovszky, Z. S. (2008). Performance of Propellers. Retrieved December 8, 2021, from https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html#SECTION06374100000000000000

Svacha, J., Mohta, K., & Kumar, V. (2017). Improving quadrotor trajectory tracking by compensating for aerodynamic effects. *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 860–866. https://doi.org/10.1109/ICUAS.2017.7991501

Wüest, V., Kumar, V., & Loianno, G. (2019). Online Estimation of Geometric and Inertia Parameters for Multirotor Aerial Vehicles [ISSN: 2577-087X]. *2019 International Conference on Robotics and Automation (ICRA)*, 1884–1890. https://doi.org/10.1109/ICRA.2019.8794274

Yajima, H., Wakiwaka, H., Minegishi, K., Fujiwara, N., & Tamura, K. (2000). Design of linear DC motor for high-speed positioning. *Sensors and Actuators A: Physical*, *81*(1), 281–284. https://doi.org/10.1016/S0924-4247(99)00175-2

A

Ground experiments

A.1. Mass measurements

Part	Weight (g)	Amount	Total (g)
Motor assembly	110	4	440
Arm	13	4	52
Arm attachment bracket	14	8	112
Landing gear	28	2	56
Leg w/ attachment	24	2	48
Mounting plates	64	1	64
Battery mount	30	1	30
Rails w/ mounts	60	1	60
Flight controller	55	1	55
Front plate	12	1	12
Camera	120	1	120
Telemetry radio	20	1	20
Tachometer	20	1	20
GPS	50	1	50
Propeller	10	4	40
Companion computer	386	1	386
Computer rods	5	4	20
Motor controllers	263	1	263
Battery	395	1	395
Subtotal			1848
Total for drone			2243

A.2. Inertia

A wider view of the bifilar pendulum setup can be seen in Figure A.1. In this image, the setup is ready to measure the inertia in the Z-axis. An initial offset angle is set by hand and then let go such that the quadcopter will oscillate around one axis. The pitch angle during this experiment, obtained using the OptiTrack system, can be seen in Figure A.1. The pitch angle is analyzed using a fast Fourier transform to find the period, this can be seen in Figure A.3. Measurements of the length of the wires and the distance between them for each axis can be found in Table A.1. The oscillation period and wire parameters are used in Equation 3.1 to calculate the MoI. The calculated MoI for each separate experiment are shown in Table A.2.

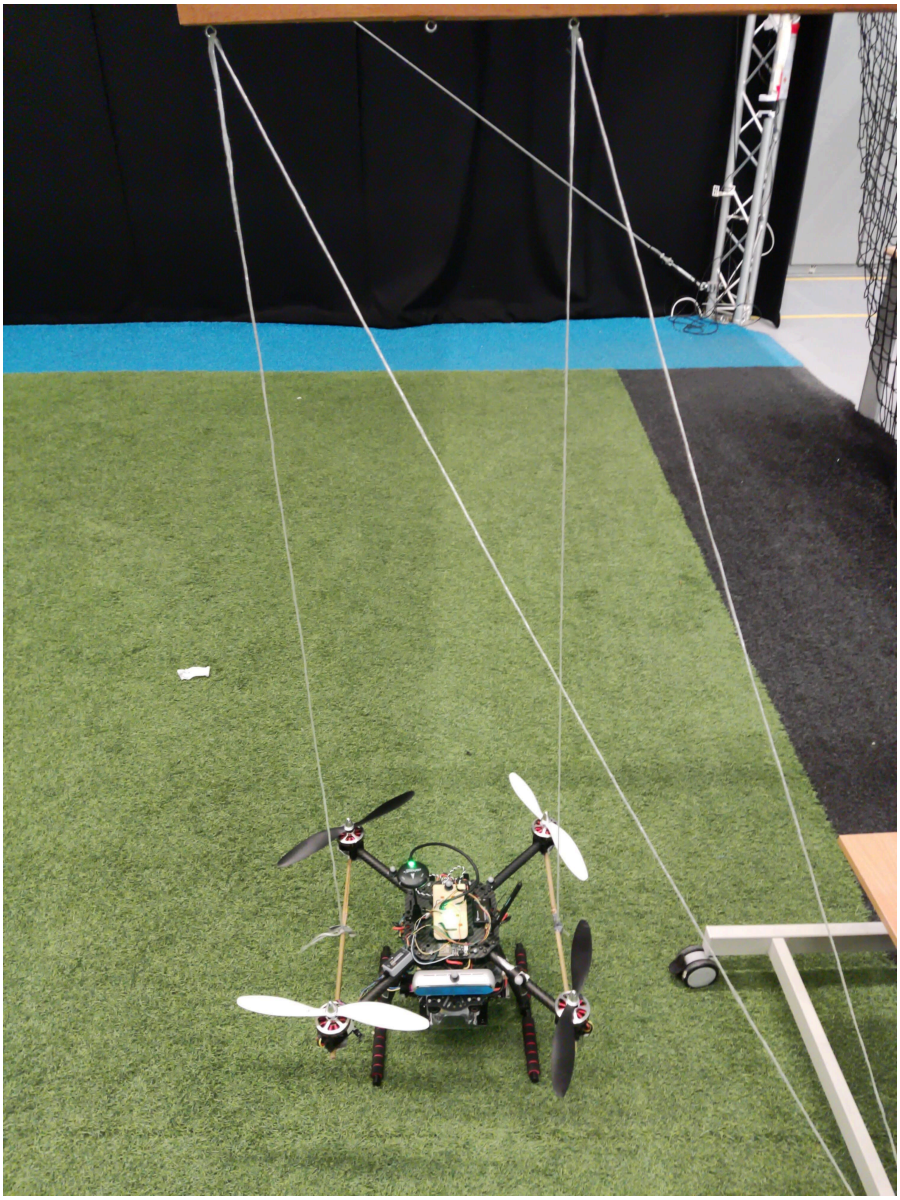


Figure A.1: Full bifilar pendulum experiment setup.

Table A.1: Wire length and distance in the bifilar pendulum experiment for the three axes.

	x	y	z
d	0.34	0.245	0.275
l	1.14	0.97	1.05

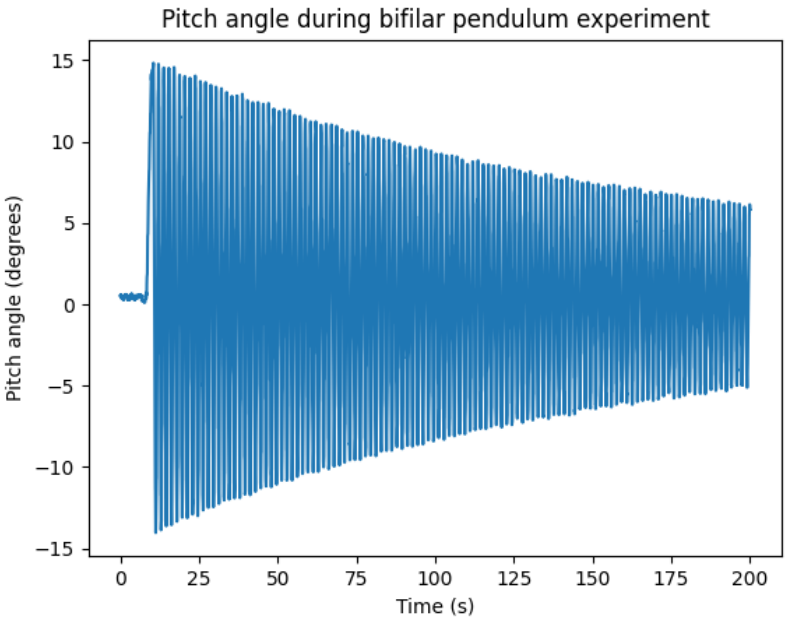


Figure A.2: Pitch angle during bifilar pendulum experiment.

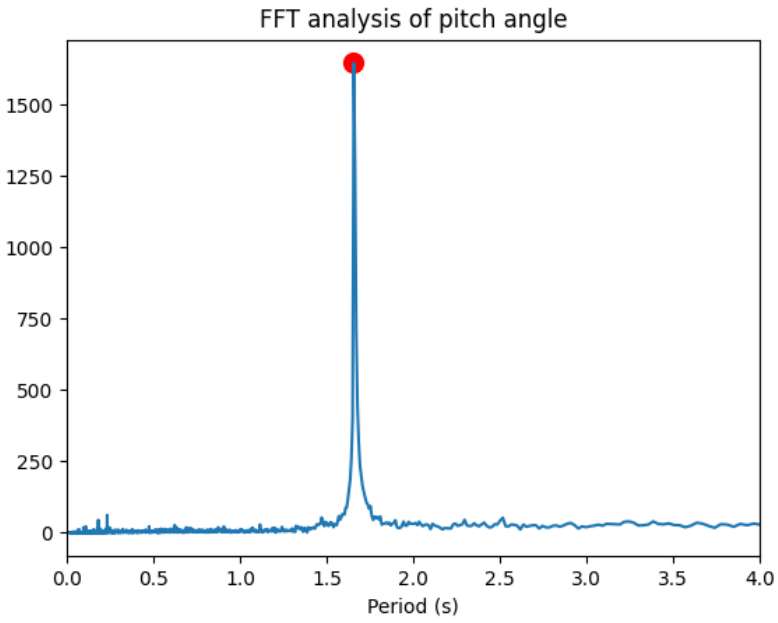


Figure A.3: FFT analysis of bifilar pendulum data.

Table A.2: MoI values found in each separate bifilar pendulum experiment.

		Inertia component		
		I_{xx}	I_{yy}	I_{zz}
Experiment	**1**	2.8E-02	3.3E-02	3.8E-02
	2	2.8E-02	3.3E-02	3.8E-02
	3	2.8E-02	3.3E-02	3.8E-02
	4	2.8E-02	3.3E-02	3.8E-02
	5	2.8E-02	3.3E-02	3.9E-02
	Average	2.8E-02	3.3E-02	3.8E-02

A.2.1. Measurement plan

Perform the following steps when starting measurements of a new axis:

1. Attach bamboo sticks to the quadcopter perpendicular to the axis that is to be measured, such that the rope positions easily be adjusted to align the ropes around the CoG
2. Attach ropes to the bamboo sticks
3. Shorten wires until the quadcopter is a few centimetres above the ground and the wires are of equal length
4. Adjust wire attachment points such that the CoG is exactly between the two wires
5. Adjust wire lengths such that the quadcopter is hanging level
6. Create a new solid body in OptiTrack for all the available markers
7. Ensure OptiTrack has stable tracking and adjust markers as necessary, this is necessary because the abnormal angle may cause some markers to be obstructed

Perform the following steps for each separate measurement:

1. Start rosbag recording of the OptiTrack data
2. Manually give the quadcopter a small offset angle of fewer than 15 degrees
3. Visually verify that there is only a rotational motion and not a translational motion.
4. Let the quadcopter oscillate for at least three minutes
5. Stop the rosbag recording

A.3. Thrust

This section will describe how the thrust experiment was performed. Section A.3.1 will explain how the seesaw setup was constructed. The rest of the experiment steps are described in Section A.4.5, as the steps are the same for all direct actuator control experiments.

A.3.1. Setup design

The experiment setup is a slightly modified version of the setup originally used by Benders, 2020. It consists of a base platform with a seesaw on top of it. The seesaw is constructed from a hollow aluminium bar and is mounted on the base platform using two L-shaped aluminium profiles to construct a pivot point. The HoverGames quadcopter is attached to one side of the seesaw using reusable zip-ties on the quadcopter legs. The other side of the seesaw pushes down on a load cell to measure the thrust generated by the quadcopter. A spring is placed under the quadcopter to keep the seesaw pressed onto the load cell at all times. This makes it possible to also measure the thrust produced at low RPMs when the quadcopter does not yet produce enough thrust to take off. The spring force can be chosen by adjusting a nut under the spring. This is used to select a spring force that is high enough to lift the quadcopter while not being so high that the total force would overload the load cell during the experiment.

To get an accurate measurement, the load cell must be loaded purely vertically and in the centre of the load cell. The load cell contact point has been placed near the same height as the lever arm. This small height differential, combined with the length of the lever arm, ensures that the force will be nearly purely vertical. To get the force centred on the load cell, two perpendicularly placed triangular aluminium profiles are used. One of these is mounted on the bottom of the lever arm, and the other is mounted on top of the load cell. An image of the setup is shown in Figure A.4.

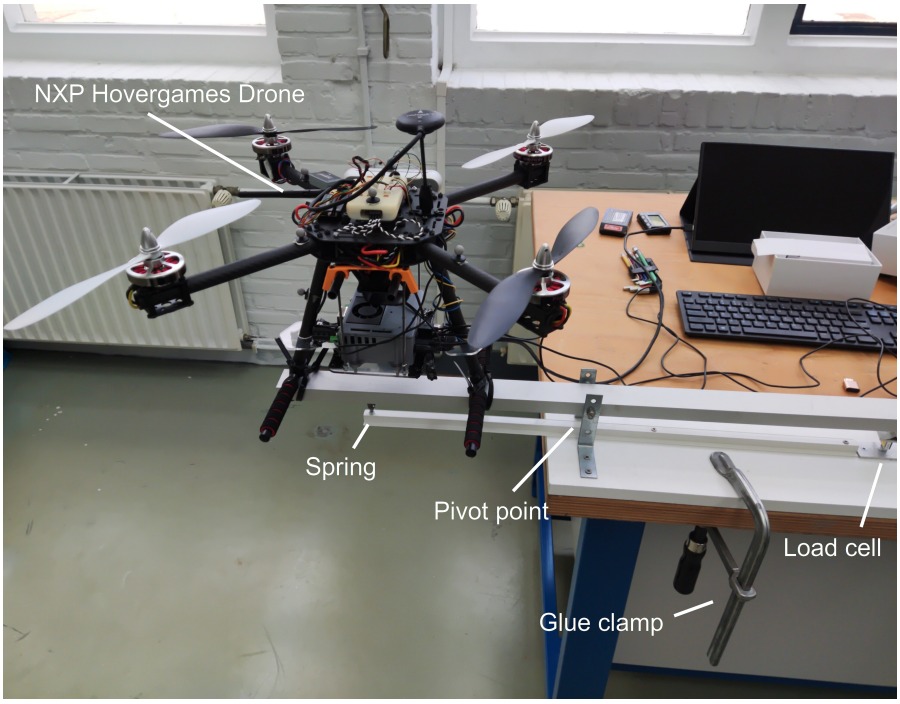


Figure A.4: Thrust measurement setup.

A.4. Torque

This section will describe how the torque experiment was performed. Section A.4.1 will explain how the seesaw setup was constructed. The rest of the experiment steps are described in Section A.4.5, as the steps are the same for all direct actuator control experiments.

A.4.1. Setup design

The experiment setup is again a modified version of the setup from Benders, 2020, this time with a few more modifications made to it. The setup consists of a wooden bar with an inline skate wheel at the end of it. The bearings in this inline skate wheel are used as a low-friction pivot point. A metal bar is mounted on the axle of the wheel, with the quadcopter on it centred around the pivot point. Two extra metal bars are added to support the legs of the quadcopter. This is visible in image Figure A.5. These extra supports are required because without them the generated torque would angle the quadcopter slightly sideways off the side of the metal bar and therefore influence the results.

Figure A.5: Torque setup quadcopter leg supports.

The end of the metal bar contacts the load cell on one side and has a spring on the other side. This spring has been added to keep the metal bar in contact with the load cell at all times, as this reduces the noise measured by the load cell. A small piece of double-sided tape is used to keep the spring from sliding on the metal bar due to the vibrations caused by the quadcopter, as this would otherwise slightly change the length of the spring and therefore the imparted spring force during the experiment. A closeup of the load cell setup can be seen in Figure A.6.

Compared to the original setup, the bar with supports and the spring have been added to improve the measurement accuracy.

A.4.2. Direct actuator control experiments

For specific experiments, it is required to be able to set the motor output command directly. Such direct control is, by default, not possible with PX4. Their reasoning for not supporting this is that it would never be possible to achieve stable flight when sending output commands from a companion computer. When performing experiments in which the quadcopter is mounted securely to a test rig, this is no longer a concern. Nonetheless, a few extra steps must be performed to enable direct motor command control. First, the general setup of direct output control is discussed. After that, the specific setup for real-life and simulator use is described.

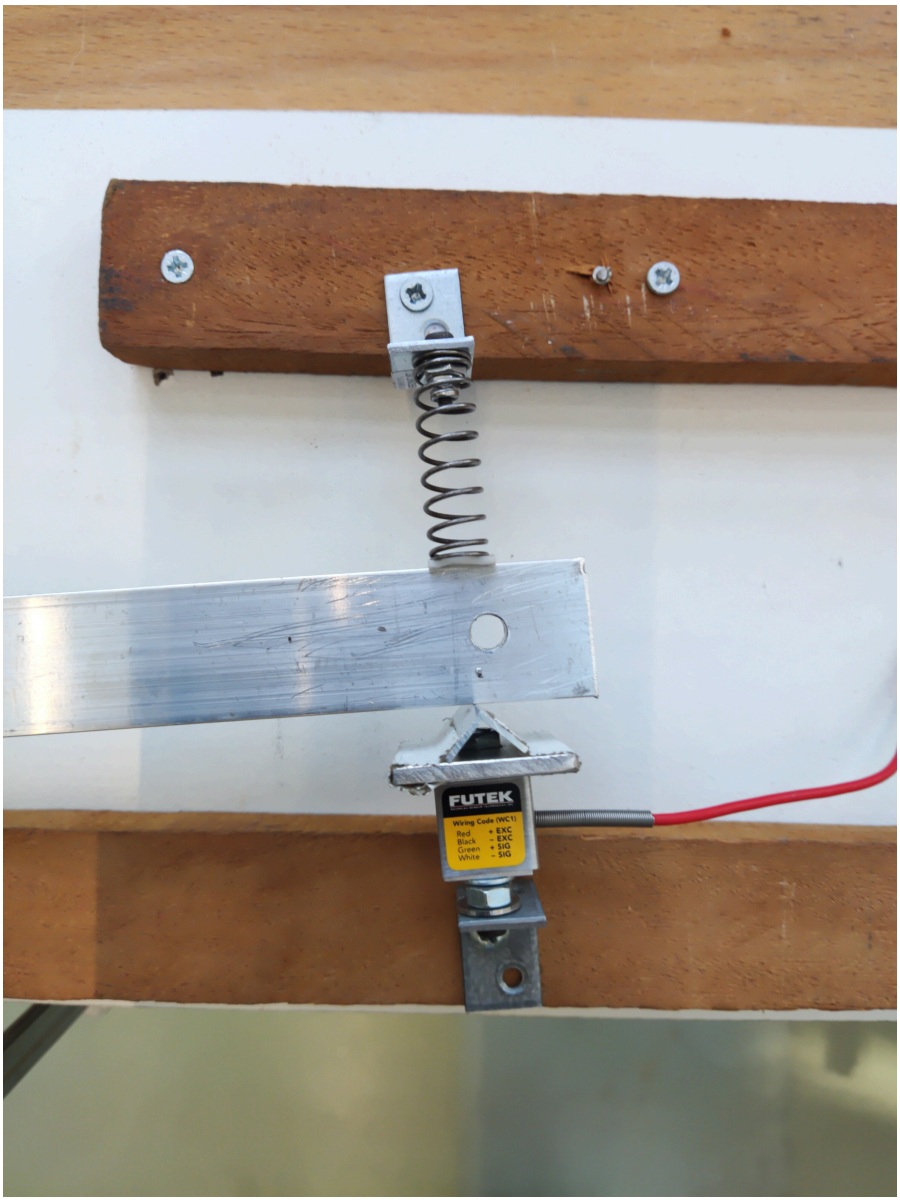


Figure A.6: Torque setup closeup of load cell.

Sending actuator commands
Actuator commands are sent via the `mavros/actuator_control` topic. In this message, an output command is a normalized number between -1 and 1[1] . This message is passed through to the PX4 mixer, which maps the actuator command onto specific outputs (PX4, 2022c). In regular operation, the mixer receives the commands roll, pitch, yaw and thrust. These commands are then mapped onto the different motors of the drone. This extra mixer step ensures that an attitude controller does not need to consider the physical location or number of rotors. To directly command specific motors, this mixer needs to be bypassed, which can be done by replacing the mixer file. The new mixer file takes the input commands and passes them straight through to the respective output motor without performing any mixing. The full mixer file can be seen in Section A.4.2.

To load the mixer onto the quadcopter, the file must be added to the SD card at `/etc/mixers/quad_x_direct.main.mix`. The mixer file can then be loaded by executing

```
mixer load /dev/pwm_output0 /fs/microsd/etc/mixers/quad_x_direct.main.mix
```

in the MAVLink console of QGroundControl. In the simulator, the mixer file can be loaded by executing

```
build/px4_sitl_default/bin/px4-mixer load /dev/pwm_output0
↪ ROMFS/px4fmu_common/mixers/quad_x_direct.main.mix
```

from the root of the PX4-Autopilot folder.

Radio control (RC) remote interference
The RC remote of the drone interferes with the actuator control commands. When the remote is left on during an experiment, the output RPM will show erratic behaviour as it seems to switch between the output commanded by the remote and the output command from the actuator control messages. The remote should therefore stay off during these experiments. PX4 checks for the presence of an RC remote when arming and will refuse to arm when the signal has been lost. This can be prevented by changing `COM_RCL_EXCEPT` to specify that RC loss should not be checked in offboard mode[2]. The experiment script will first change PX4 to offboard mode, and then arm it afterwards. Flight experiments in the lab have the opposite sequence, as the quadcopter can not be armed anymore using the RC when it is already in offboard mode. For good measure, however, the setting should be returned to default after the experiment.

Disabling auto disarm
In all direct actuator control experiments, the quadcopter will be mounted onto some form of a test rig to prevent it from flying away. This setup somewhat confuses PX4 and leads to an inconsistent state where the quadcopter can change between `FLYING` and `LANDED` in the middle of an experiment. PX4 will usually auto-disarm the quadcopter when it does not take off after arming or after it has landed. This behaviour causes the quadcopter to sometimes disarm in the middle of an experiment and therefore has to be disabled. These checks can be disabled by changing the `COM_DISARM_PRFLT` and `COM_DISARM_LAND` parameters to 0 using QGroundControl[3].

PX4 actuator control bug
Finally, it should be noted that the stable version of PX4 used in this research (v1.12.3) contained a bug that made it impossible to use offboard actuator setpoints. This was fixed in pull request #18581[4]. Therefore, the PX4 firmware on the quadcopter was upgraded to v1.13.0 beta temporarily for the experiments with PWM control.

Mixer file

```
MAIN1 Passthrough
M: 1
S: 3 0   10000   10000        0 -10000   10000
```

[1] http://docs.ros.org/en/api/mavros_msgs/html/msg/ActuatorControl.html
[2] https://docs.px4.io/main/en/advanced_config/parameter_reference.html
[3] https://docs.px4.io/main/en/advanced_config/parameter_reference.html
[4] https://github.com/PX4/PX4-Autopilot/pull/18581

```
MAIN2 Passthrough
M: 1
S: 3 1 10000 10000 0 -10000 10000

MAIN3 Passthrough
M: 1
S: 3 2 10000 10000 0 -10000 10000

MAIN4 Passthrough
M: 1
S: 3 3 10000 10000 0 -10000 10000

AUX1 Passthrough
M: 1
S: 3 5 10000 10000 0 -10000 10000

AUX2 Passthrough
M: 1
S: 3 6 10000 10000 0 -10000 10000

Failsafe outputs
The following outputs are set to their disarmed value
during normal operation and to their failsafe falue in case
of flight termination.
Z:
Z:
```

A.4.3. RPM measurement

On the real-life quadcopter, the rotor RPM is measured using a tachometer. In the simulator, a modification has been made to receive rotor velocity data. The real-life setup will be explained in Section A.4.3, and the simulator version will be explained in Section A.4.3.

Real-life tachometer

To measure the rotor velocity, the ThunderFly TFRPM01[5]tachometer with TFPROBE01[6]sensor probe is mounted onto the quadcopter. This probe supports both reflective optical sensing as well as magnetic sensing. It was found that the magnetic sensing capabilities of the probe can be used to measure the small magnets inside of the electric motors. This provides 14 pulses per full rotation. In the PX4 firmware, the `PCF8583_MAGNET` parameter can be adjusted to set the number of pulses per full rotation of the rotor.

The tachometer has a pulse frequency range of up to 20kHz. A simple verification was done to ensure the pulses per second stayed within the tachometer specifications. The maximum possible rotor RPM is when the battery is fully charged and only the rotor being measured is at full power, with the rest of the rotors at idle. In this case, the rotor RPM approached, but never quite reached, 10000 RPM. This means that at max RPM, there are approximately $10000/60 * 16 \approx 2667$ pulses per second, staying well within the tachometer design specifications of 20kHz.

The tachometer and sensor probe are very lightweight, coming in at a total of about 15 grams. The tachometer is placed near the centre portion of the quadcopter to influence the MoI as little as possible, with only the sensor probe being mounted near the rotor. Because of this, the RPM measurement setup will have a negligible impact on total quadcopter mass and MoI.

Finally, some modifications on the software side need to be made. The driver for the tachometer needs to be started, and the rate at which the sensor reports the RPM needs to be modified. By default, this is only every half a second, but for some of the experiments, this will need to be higher. Therefore, the `PCF8583_POOL` parameter, which sets how often the tachometer is read out, was changed. Additionally, the rate of the `RAW_RPM` MAVLink message was increased.

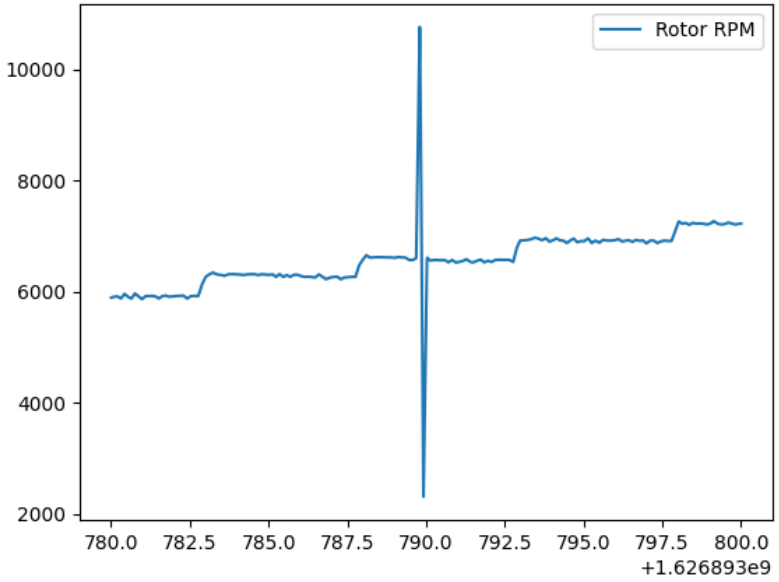


Figure A.7: RPM measurement with an artefact.

This setup provides accurate RPM measurements with little noise. Occasionally, the measured data will contain an artefact. An example of this is shown in Figure A.7. The exact cause of these artefacts is not known. Since they only occur occasionally, it was found that they could be filtered out successfully using outlier detection, and the artefact issue was not investigated further.
[7] [8]

Simulator tachometer
To obtain RPM data in the simulator, a new Gazebo plugin was added. This plugin subscribes to Gazebo world updates to read the rotor joint velocity and publishes this as a ROS message. The name of the joint to monitor and the topic to publish can be changed in the model SDF file.

A.4.4. Force measurement
To measure the force generated by the quadcopter, a miniature S-beam load cell, the Futek LSB200[9], is used. This load cell has been calibrated before use up to 40N. The load cell is connected to an analogue signal conditioner, the Feteris Scaime CPJ2[10], to amplify the measured signal. The amplified signal is fed into a National Instruments USB-6008 data acquisition (DAQ)[11], which sends the data to a virtual instrument in Labview 2018, where it is visualized and can be saved to a file. Due to the dependence on Labview 2018, this was done on a separate Windows laptop.

A.4.5. Ground experiments measurement plan
The measurement plan of all ground-based experiments is very similar and has therefore been bundled together here. Section A.4.5 lists the steps that need to be performed before the start of the first experiment. Section A.4.5 lists the steps that need to be performed after every battery change.

One-time setup steps
The thrust and torque experiments require some extra steps to set up the load cell. The following steps need to be executed if load cell data is required:

[7] https://github.com/ThunderFly-aerospace/TFRPM01
[8] https://github.com/ThunderFly-aerospace/TFPROBE01
[11] https://www.futek.com/store/load-cells/s-beam-load-cells/miniature-s-beam-LSB200/FSH01559
[11] https://scaime.com/product/post/cpj—cpj2s
[11] https://www.ni.com/nl-nl/support/model.usb-6008.html

1. Connect load cell via the signal conditioner and DAQ to the laptop running LabView 2018.
2. Adjust the spring nut to create between 2N and 5N force on the load cell.
3. Set measured force to zero in LabView.
4. Measure the distance from the quadcopter legs to the pivot point and from the pivot point to the load cell.

1. Add `pcf8583 start -X -b 1` to `/etc/extras.txt` to enable the tachometer driver.
2. Add `mavlink stream -d /dev/ttyS4 -s RAW_RPM -r 100` to `/etc/extras.txt` to increase the tachometer micro Object Request Broker (uORB) message rate.
3. Add `mavlink stream -d /dev/ttyS4 -s BATTERY_STATUS -r 10` to `/etc/extras.txt` to increase the battery status uORB message rate.
4. Attach setup to the table using glue clamps.
5. Mount quadcopter legs firmly on the lever arm using cable ties.
6. Connect a fully charged battery to the quadcopter.
7. Ensure PX4 firmware version v1.13.0 or newer has been flashed onto the quadcopter.
8. Set `PCF8583_POOL` parameter to 100 000 µs.
9. Set `COM_RCL_EXCEPT` to 4 to disable RC connection safety check.
10. Disable auto disarm parameters `COM_DISARM_LAND` and `COM_DISARM_PRFLT` to prevent PX4 disarming the quadcopter during the experiment.

Per experiment setup steps

1. Load custom mixer by running:

```
mixer load /dev/pwm_output0
↪ /fs/microsd/etc/mixers/quad_x_direct.main.mix
```

2. Adjust MAVLink `RAW_RPM` message interval by running:

```
rosservice call /mavros/set_message_interval 339 100
```

3. Adjust MAVLink `BATTERY_STATUS` message interval by running:

```
rosservice call /mavros/set_message_interval 147 10
```

4. In the case of a thrust or torque test, start LabView recording.
5. Run ROS launch file that starts the experiment script and rosbag recording.

The quadcopter will perform the experiment and automatically disarm at the end. Once this has been completed, perform the following steps:

1. Stop the LabView recording.
2. Save the LabView data to a file with a date and description of the experiment.
3. Rename the created rosbag to have the same date and description.
4. Remove the battery from the quadcopter for charging.

When done with all the experiments, return the `COM_DISARM_LAND`, `COM_DISARM_PRFLT` and `COM_RCL_EXCEPT` parameters to their default setting.

B

Flight experiments setup

B.1. Lab software setup overview

The software setup can be divided into three main parts; the flight controller, the companion computer and the ground station. The flight controller is the heart of the HoverGames quadcopter and runs on the PX4 Autopilot. This autopilot is a widely used open-source project that performs the control and stabilization of the quadcopter. It uses various sensors to accomplish this, like an accelerometer, gyroscope, magnetometer and barometer. The flight controller has limited wireless connectivity and processing power, and therefore certain tasks are offloaded to a companion computer. This companion computer is mounted onto the quadcopter frame and communicates with the flight controller using a universal asynchronous receiver-transmitter (UART) connection. Any control algorithms that need to run to perform experiments will be executed on this companion computer.

Finally, there is an RC remote and a laptop connected to the flight controller's telemetry radio. The RC remote is used to change the flight mode and take over control in case anything goes wrong. The laptop is not necessarily required to perform a flight, but it is used to adjust any PX4 parameters before the flight and monitor the state during the flight. Any error messages that may occur during the flight, either on the flight controller or the companion computer, can be read here.

A diagram of the setup can be seen in Figure B.1. The parts that are physically connected are inside container boxes, while the wireless connections are made with dotted lines.

B.1.1. Protocols

This section will list the different protocols used to communicate between the different components. A single message between components will often pass through most of these protocols. This may seem like a convoluted method, but it is done this way to make the entire system as modular as possible.

For example, an OptiTrack message in the simulator will be created by a Gazebo plugin which gains access to the world state via a Gazebo event. The message is then sent over a Gazebo topic and read by the `gazebo_mavlink_interface`. This interface will translate the message to MAVLink, and send it to PX4. Within PX4, the message is translated to uORB and processed to update the internal position estimate. This new position estimate is broadcast via another MAVLink topic, which is read by MAVROS. Finally, MAVROS republishes the message as a ROS message which is used within our own code. A short description of the different protocols will be given now.

Robot Operating System (ROS)

ROS has several methods available for communication. The main method is via topics. Any ROS node can publish a message on a specific topic, and any other node can subscribe to messages on that topic. This leads to anonymous many-to-many communication where nodes can easily be replaced. Sometimes, it may also be relevant to get a response to a message. For that use case, an alternative method exists, namely, services. These are one-to-one connections where each sent message will also get a reply.

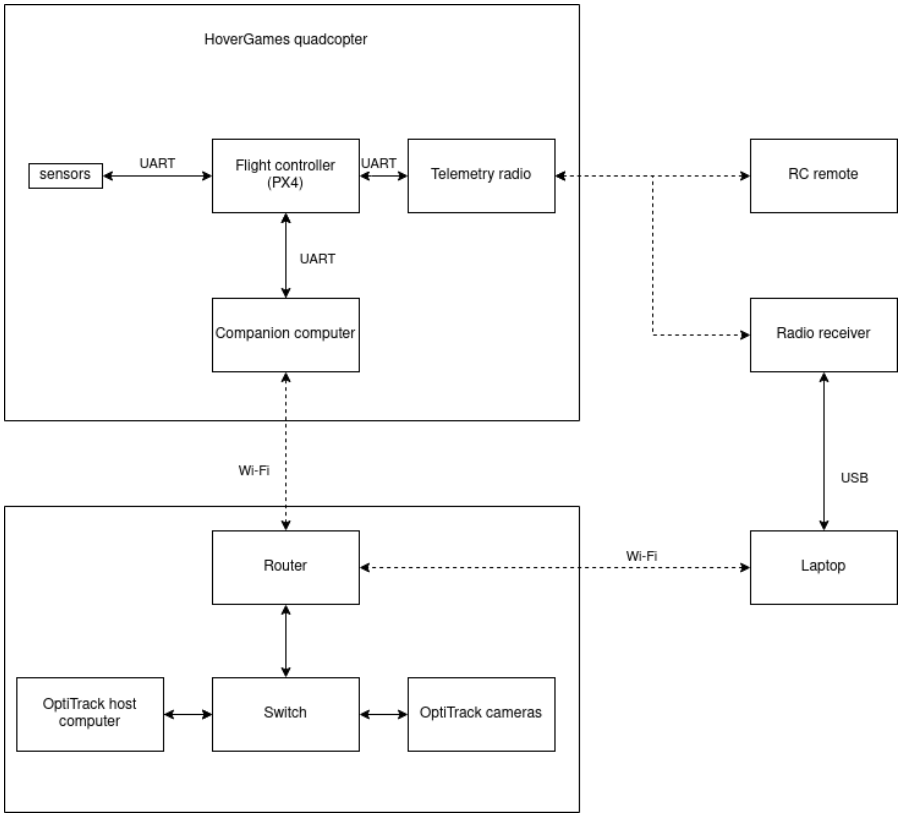


Figure B.1: Lab setup overview.

Gazebo
Gazebo uses a messaging system with topics similar to that of ROS. On top of that, it adds an event system where plugins can subscribe to specific events that happen in the simulation. Most notably, plugins can subscribe to an event that is called at the start of each simulator update. This is how the motor model is able to add aerodynamic forces to the simulation.

MAVLink
MAVLink is a messaging protocol which PX4 uses for its external communication. In our setup, this protocol is therefore used by both the radio link to QGroundControl, and also the UART connection to the companion computer. MAVLink also uses a topics system for most communication, with an alternative system for one-to-one communication in place to set flight controller parameters and mission data.

uORB
uORB is, like the other protocols, a topic-based messaging API. This is the protocol that PX4 uses for its internal communications.

B.1.2. OptiTrack

The lab uses OptiTrack as a motion capture system capable of precise, real-time tracking in 6 Degrees of freedom (DoF)[1]. This system consists of a series of cameras around the edges of the arena that emit IR light and capture the reflections. Spherical reflective markers are used to provide accurate reflections from all sides. Three reflective markers are required to reconstruct the quadcopter orientation in 3D space. In practice, one of the markers may become occluded during flight and not be trackable for a short period of time. Therefore, four reflective markers are placed on the quadcopter to be less vulnerable to marker occlusions.

The camera data is sent to an OptiTrack host computer running Motive[2] that processes the camera

[1] https://docs.px4.io/main/en/tutorials/motion-capture.html
[2] https://optitrack.com/software/motive/

data to obtain the quadcopter position and orientation. This data is sent over Wi-Fi to the companion computer, which uses the `mocap_optitrack` ROS node to transform the OptiTrack NatNet packages to ROS messages. These messages are throttled at 50Hz, to prevent overloading the flight controller and then sent to the flight controller via UART. Direct communication from the Optitrack host computer to the flight controller would not be possible, as the telemetry radio does not provide a high enough bandwidth for motion capture usage PX4, 2022b.

OptiTrack issues
During experiments, it was found that OptiTrack was unstable, and the quadcopter would often lose position data for several seconds. Without OptiTrack data, the quadcopter's internal position estimate will almost immediately start drifting, which can lead to it crashing into the flight arena boundaries in mere seconds. The cause of these connection issues was pinpointed to the Wi-Fi connection to the companion computer. This connection was via a 2.4 GHz Wi-Fi network but has been changed to a 5 GHz network. Connection issues were reduced significantly with this change, although it still occurs occasionally. To further reduce these issues, the connection should be improved by improving the Wi-Fi router setup, or onboard vision odometry should be used that removes the requirement for an external connection altogether.

B.2. Simulator

A diagram of the communication paths between different simulator components can be seen in Figure B.2. Internal communication paths within PX4 are omitted for clarity. The files containing the motor controller simulator and motor model are shown within the `sitl_gazebo` block.

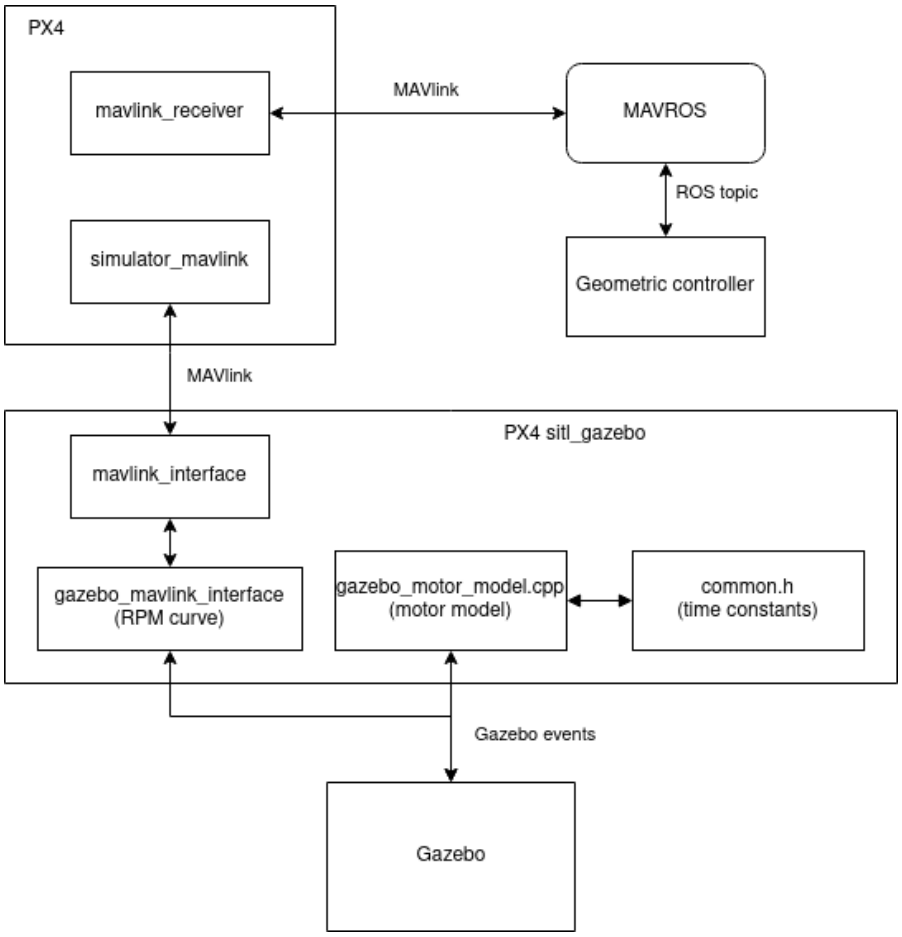


Figure B.2: Overview of main modules involved in simulator flight.

C

Software setup

To perform flight validation of the simulation, a trajectory tracking controller capable of flying aggressive trajectories is required. A modified version of the mavros_controllers package by Jaeyoung, 2019 was used to accomplish this. This controller package has been integrated with the PX4 Tools package developed by Dennis Benders. This section will give an overview of these packages and discuss the changes that have been made to perform the experiments.

C.1. PX4 Tools

The PX4 Tools package provides a control interface that handles tasks required for all flight experiments[1]. For example, to simulate an experiment, the PX4 SITL package needs to be loaded, the Gazebo simulator needs to be launched with our quadcopter model added, and the quadcopter needs to be armed for takeoff programmatically. For real-life experiments, only the OptiTrack settings need to be loaded. In either case, the control interface will also handle takeoff and landing and monitor the lab safety boundaries during flight. These boundaries consist of a virtual box that the quadcopter must stay within. If the quadcopter leaves this area, the control interface will take back control and abort the experiment. This was added to reduce the risk of crashes in the case of a malfunctioning experiment. In conclusion, the control interface provides often-needed functionality that reduces duplicated code and allows for a better-tested and more robust implementation of those tasks.

C.2. Mavros controllers

The mavros_controllers package consists of two separate packages[2]. The first package, the trajectory publisher, generates a trajectory and publishes setpoints to follow. The second package, the geometric controller, subscribes to the published setpoints and converts them to body rate commands that are sent to PX4. The two packages will be discussed in more detail now, including the changes made to them to add the functionality required for the flight experiments.

C.2.1. Trajectory publisher

This package provides support for generating a variety of trajectories. A circle and lemniscate-shaped trajectory already existed, and a straight line and diagonal trajectory have been added to perform specific experiments. Additionally, the trajectory radius and velocity have been made dynamically configurable to make it easier to test different configurations in the lab. A lap counter was added to stop the experiments automatically. Finally, a windup ratio was implemented, which slowly speeds up the trajectory velocity at the start of the flight. This was done to prevent the jerky motion that otherwise occurred when the quadcopter had to accelerate from a hover to full trajectory speed instantly.

[1] https://github.com/cor-drone-dev/drone_toolbox
[2] https://github.com/michieltebraake/mavros_controllers/

C.2.2. Geometric controller

The geometric controller package receives trajectory position and velocity setpoints and processes these to calculate a body rate command for PX4. This control method was defined by Brescianini et al., 2013. It functions by calculating the desired target orientation based on the position and velocity error of the quadcopter compared to the trajectory setpoint. The controller has been proven to have global asymptotic stability and is, therefore, able to recover to a stable hover from any arbitrary attitude. This makes it possible to track aggressive trajectories, in which the flight dynamics have a larger impact on the quadcopter position.

The geometric controller package has been modified to work with the PX4 Tools package. Additionally, support has been added for setting a static body rate command. This will be used to command exact yaw velocities and thereby test the torque the rotors can generate.

C.3. Data processing

During experiments, a large amount of data is recorded into a rosbag file. To efficiently analyse this data, a tool was created to process it into an easier-to-work-with state. An overview of the functionality will be given here.

Interpolation

Different topics are generally not published at the same rate or same timestamp. To make it possible to compare different topics at a specific timestamp, an interpolation step has been added. Depending on the message type, this is done using linear or nearest-neighbour interpolation. For example, a position message is interpolated, but the lap counter isn't.

Simulator specific topics

Some messages in the simulator are sent under a different topic name and sometimes with a different data type. For example, the tachometer data on the quadcopter is reported in RPM, while the rotor velocity data in the simulator is published as radians per second. Such differences are equalized here to ensure that the analysis algorithms can stay the same regardless of where the data was recorded.

Data caching

Loading and processing large rosbag files can take a significant amount of time. When repeatedly running analysis scripts on the data, this processing time can quickly add up. Therefore, a caching functionality was implemented that serializes the processed data such that it can quickly be loaded again.

C.4. Trajectory comparison

A script was constructed to compare two different trajectories and calculate metrics by which the simulator models can be judged[3]. Different trajectories are compared based on their position and orientation at a specific time in the trajectory. The different data files are aligned based on the start of the reference trajectory. Their position and orientation can then be compared at any timestamp. These comparisons are used to calculate a RMSE for each position and orientation axis. The time-based comparison means a time delay causes a significant calculated error metric, even if the final trajectory roughly follows the same path. It was decided that a time delay is a relevant inaccuracy in the simulation, although its relevancy does depend on the specific scenario that is simulated. If the optimal position-based trajectory tracking performance is required, then a different metric should be used.

C.5. Motor model communication

To enable communication with the motor model, the `SITL_gazebo` package has been built with catkin support enabled. This makes it possible to add ROS subscribers and publishers within the motor model. This has been used for two different functions. The first use is the addition of a publisher that sends the calculated model forces and moments. This has, for example, been used to analyze the influence

[3] https://github.com/cor-drone-dev/drone_toolbox/blob/sys_id_experiments/hovergames/hovergames_sim_identification/testbench_identification/src/analysis/trajectory_comparison/compare_trajectories.py

of the rolling moment in Section 9.4. The second use is to listen for new model parameters, such that they can be updated without restarting the simulator. In the future, this could enable automated tuning of the model coefficients.

C.6. Software

This section will list the software packages used in this research. First, it will list all the software packages that have been modified. Secondly, it will list the packages of which the default version was used. Finally, it will list the Python packages that were used for the experiment scripts and analysis.

C.6.1. Modified software packages

- PX4 (https://github.com/cor-drone-dev/PX4-Autopilot/tree/hovergames_drone)
- SITL_gazebo (https://github.com/cor-drone-dev/px4-sitl_gazebo/tree/hovergames_drone)
- MAVROS (https://github.com/cor-drone-dev/mavros/tree/rpm_support)
- drone_toolbox (https://github.com/cor-drone-dev/drone_toolbox/tree/hovergames_drone)
- mavros_controllers (https://github.com/cor-drone-dev/mavros_controllers/tree/hovergames_drone)

C.6.2. Unmodified software packages

- Python 3.8.10
- ROS Noetic
- Gazebo 11
- Rviz 1.14.14
- PlotJuggler 3.6.0
- mocap_optitrack (https://github.com/ros-drivers/mocap_optitrack)
- gazebo_ros_packages (https://github.com/ros-simulation/aazebo_ros_pkgs)

C.6.3. Python libraries

- bagpy 0.4.7
- matplotlib 3.2.1
- mavros 1.1.40
- numpy 1.21.4
- pandas 1.3.5
- pymavlink 2.4.19
- scipy 1.7.2
- scikit-learn 1.1.1
- tf 1.13.2