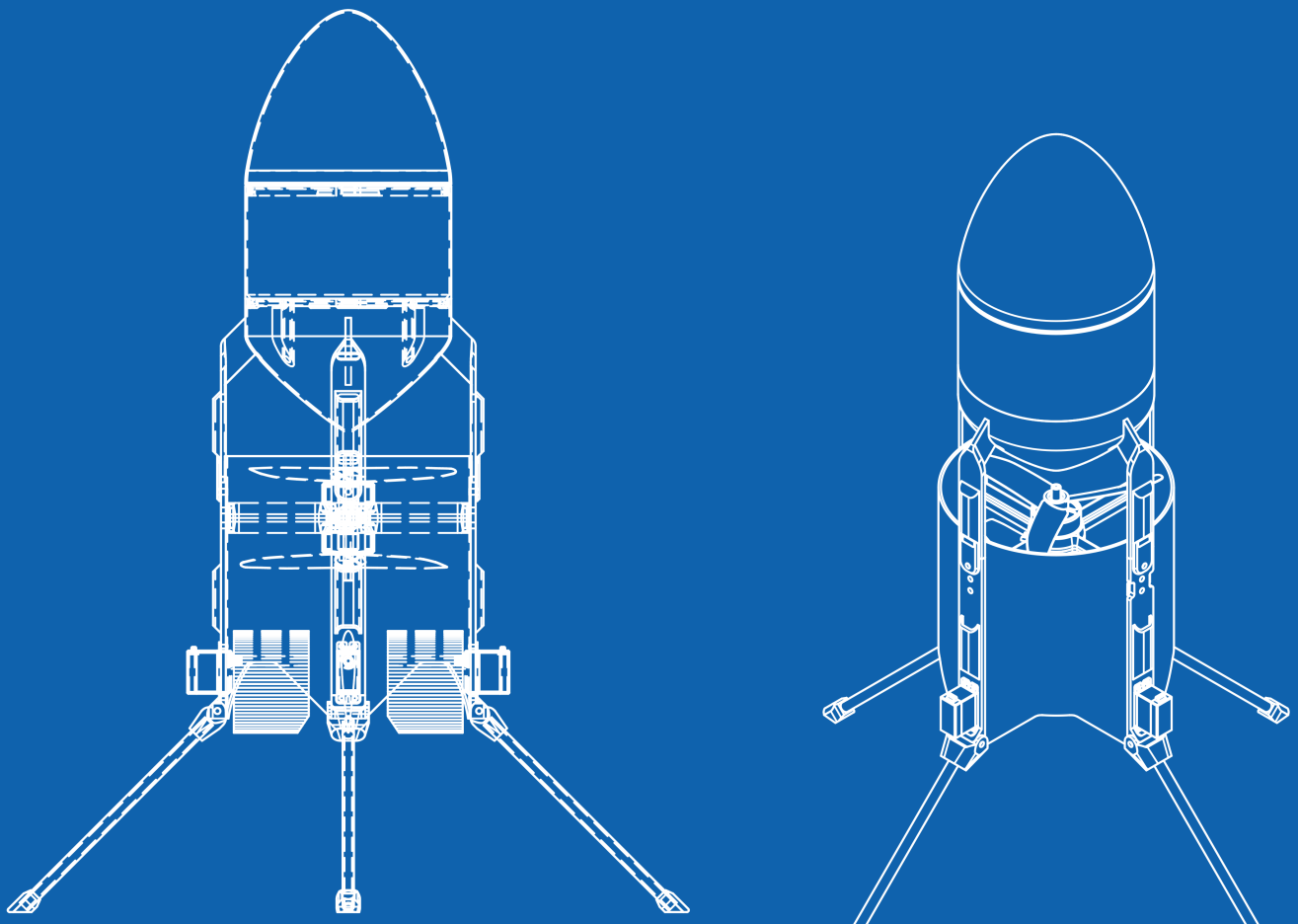
Modelling and Control of Thrust Vectoring Mono-copter

Master Thesis by
Emil Bjerregaard Jacobsen



Aalborg University
Control and Automation

AALBORG UNIVERSITY

Control and Automation
Aalborg University
www.aau.dk

Title:
Modelling and Control of Thrust Vectoring Mono-copter

Theme:
Control and Automation

Project Period:
01/02-2020 - 02/06-2021

Participant(s):
Emil Bjerregaard Jacobsen

Supervisor(s):
Kirsten Mølgaard Nielsen

Copies: 1

Page Numbers: 89

Date of Completion:
June 2, 2021

Abstract:

In recent years, private companies have embraced the space industry to pursue the immense economic potential of space. The private industry has skyrocketed the development of reusable rockets to increase profit and make space economically viable. The sudden growth has raised the demand for new employees with hands-on experience. However, reusable rockets and space technology is out of reach of most scientific institutes, as no educational platform is readily available. This project investigates the concept of mono-copters as a practical way to mimic reusable rockets' behaviour in a safe environment. A custom hardware platform is developed to accomplish this, and a model-based control strategy using LQR is proposed to stabilise the mono-copter. This requires measurements that are not available, for which an optimal estimator is implemented; the Kalman filter. In conclusion the proposed control strategy is able to stabilise the system, enabling the mono-copter to perform vertical take-off, hovering and landings, much like a reusable rocket.

Contents

Preface

This report documents the work done during my master's thesis in control and automation, conducted in the spring of 2021, at Aalborg University.

I want to express my sincere thanks to my supervisor Kirsten Mølgaard Nielsen for supplying guidance and theoretical support from start to finish. I also wish to thank engineering assistant Kenneth Kirke for valuable advice and helpful, practical discussions throughout the semester. In addition, I would like to thank my friends and family, who has supported me in a time of global crisis. Lastly, a big thanks are given to Dane RC Aps, for supplying the electronics and hardware needed for this thesis.

Readers guide

This report is intended to be read in chronological order. Throughout the report, figures and equations are labelled using numbers which is used for reference in the text. Sources and materials used for the project are found in a bibliography at the end, and are referred to using numbers. Lastly the equations composed of physical units, follows the standard of SI-units.

Emil Bjerregaard Jacobsen
thesolidgeek@gmail.com

CHAPTER 1

Analysis

1.1 Introduction

In recent years, the space industry has seen a sky-rocketing increase in public interest, and private companies are accelerating the development of rockets and space technology, comparable to the space-race of the sixties. One of the main drivers of this growth is the economical benefits of reusable rockets [1], which until a few year ago was considered impossible. Companies such as Blue Origin [2] and SpaceX [3], both develops reusable rockets, and the latter has achieved multiple low-earth-orbit missions and performed vertical reentry landings [1], see **Figure 1.1**. This has shown to be a profitable business, as reusable rockets reduce the cost of space-flight by an order of magnitude [4], since the costs are reduced to that of the propellant and refurbishments [1].

Figure 1.1: Reentry of the Falcon 9 rocket, performing a controlled vertical landing [3]

Reusable rockets are characterised by being able to reenter the atmosphere and perform an autonomous landing after a mission [1]. The technology needed to control such rockets are an interesting topic that poses great theoretical and practical challenges. However, the concept of reusable rockets has sparsely been researched in practice, as the technology is out of reach for most people. Rockets are expensive, dangerous and does not fit into an academic environment. This is a problem, as the rapidly growing space industry is in huge demand for new employees with practical and theoretical knowledge [5]. To support the ongoing development of reusable rockets, and give students hands-on-experience with space-related technologies, the academic institutes would benefit from a safer and more accessible platform that replicates the behaviour and dynamics of a reusable rocket. Opportunely, the concept of reusable rockets are not that different from the field of unmanned aerial vehicles (UAVs) which in recent years has been widely researched. Knowledge can especially be drawn from the research on vertical take-off and landing (VTOL) UAVs, as they operate using principles similar to that of reusable rockets.

1.2 Problem Statement

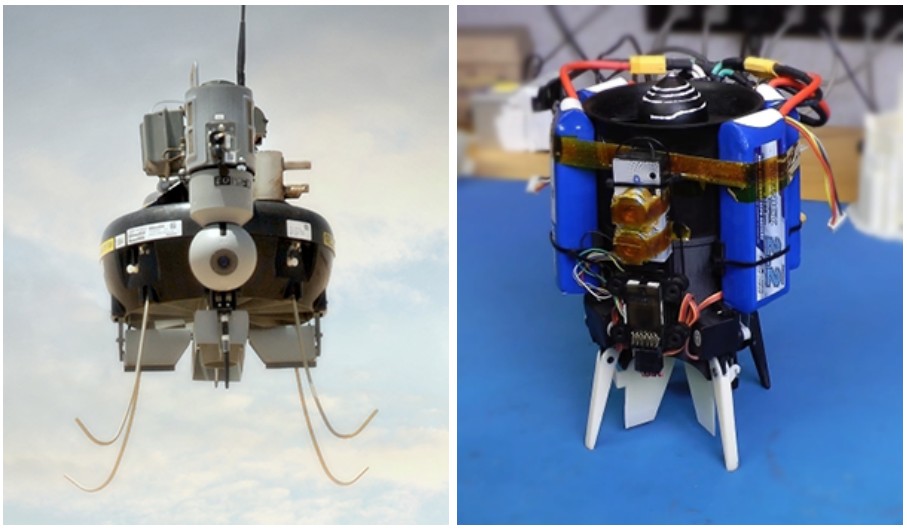
The space industry is in demand for new and skilled employees with experience in the field of space technology. However, because space technology is out of reach for most scientific institutes, engineering students and researchers have a hard time getting hands-on-experience. In order to enable the scientific community to contribute to the advances in reusable rockets, the technology has to be made safe and accessible for everyone. To make the technology more accessible, the principle of autonomous UAVs could be used to mimic the dynamics of reusable rockets, on a safe and smaller scale. However, to capture the flight dynamics of a reusable rocket in a small UAV, the system must be designed using the principles of flight of a rocket. The most significant difference between classical VTOLs (such as quad-copters) and reusable rockets, are the propulsion and control. Quad-copter uses multiple sources of thrust to produce rotational moments, that can be controlled to stabilise the orientation. Reusable rockets, on the other hand, uses only one primary source of thrust, and applies thrust vectoring control (TVC) in order to stabilise its orientation.

This leads to the following problem statement:

> "*How can one design and stabilise an autonomous UAV that uses a single source of thrust, to achieve vertical take-off and landing, similar to that of a reusable rocket.*"

1.3 State of the Art

The field of autonomous UAVs with VTOL capabilities has been widely researched in the last decade [6][7][8]. The underlying goal of this research has been to achieve hovering stability for a large variety of aircraft. In the scientific field of VTOLs, the most frequent design is that of multi-rotors [6] such as quad-copters. Another less researched type of VTOL is the mono-copter, also known as a *single rotor UAV* (SRUAV), which uses thrust vectoring control (TVC) to stabilise the attitude. As the name implies, these aircraft uses just a single source of vectored thrust to achieve stability, much like a reusable rocket. The concept of mono-copters date back to the 60s [8], however, only a few mono-copters has ever been documented, and those that have, are either military projects or amateur projects with no theoretical documentation, see **Figue 1.2**.



(a) Honeywell RQ-16 T-Hawk [9] **(b)** Amatuer Single-Rotor Drone [10]

Figure 1.2: Two mono-copter VTOL designs

The absence of mono-copters in literature is largely contributed to the technological challenges associated with mono-copters, and limitations of past technology. Recent years of improved battery technology and light-weight avionics has, however, made the concept of autonomous mono-copters more accessible [11]. The new possibilities has aroused new interest in mono-copters, and as shown by Carholt et al. [8], the research gap regarding mono-copters is not a result of bad flight performance. On the contrary mono-copters is thought to be a promising alternative to existing VTOL designs such as quad-copters, which poses safety and efficiency issues due to unenclosed propellers [8]. This is further supported by the reduced production cost of mono-copters, due to a large reduction in motors, electronics and propellers.

The controlling principle of mono-copters is based on producing rotational moments, such that the centre of mass (COM) pivots around the principle axes. This is the pervasive principle used to control all airborne vehicles, with the only difference being how the moments are produced. As an example, quad-copters uses differences in motor thrust to produce controlling moments, while the mono-copter uses thrust vectoring. The dynamic behaviour of such system is often modelled as a rotational rigid body, described by the Newton-Euler equations [12]. Carholt et al. [8] uses this approach, to develop a dynamic model used to simulate the behaviour of a generic mono-copter. The model is however not used to design a stabilising controller, and instead a generic model-free PID method is applied (trial-and-error) [8]. A more sophisticated model-based control approach, using state-space and the linear quadratic regulator (LQR) is applied to a quad-copter by Greiff [6] and by Foehn and Scaramuzza [13]. However, because the controlling principle of a quad-copter is quite similar to that of a mono-copter, the same control approach might be considered for the latter.

Using the advances in drone technology, the concept of mono-copters is considered to be applicable in the design of a small scale reusable rocket. Model-based control methods such as LQR might be useful to stabilise the mono-copter, however to do so, an advanced system model is needed. Based on an extensive literature search, this has not been attempted in scientific literature before, and presents an obvious opportunity to produce a novel contribution to the field.

1.4 Summary

This thesis seeks to explore and apply the basic theoretical building blocks of autonomous UAVs, into the development of a platform that mimics the behaviour and dynamics of a reusable rocket. The goal is to design a VTOL platform with a single source of thrust, that can be used in academic projects alike, to test and apply principles of reusable rockets in a safe environment. Simultaneously, the work aspires to test and apply well-known control principles, onto the hardware platform as to validate the performance of the implemented system.

CHAPTER 2

System Design

The goal of this thesis is to replicate the dynamics of a reusable rocket, in a small scale autonomous UAV, using a single source of vectored thrust. This can be achieved purely in simulation, as done by Carholt et al. [8], however even with a good model, simulation can never replace empirical data. In order to gain empirical data, a hardware platform is needed. However, because of the unconventional principle of flight no such drone platform exists, and the construction of a custom platform is needed. The design of this platform is considered a large part of the thesis, and this chapter aims to describe the design and construction of a *Thrust Vectored Mono-copter*. The functional principle of the hardware platform is illustrated in **Figure 2.1**.

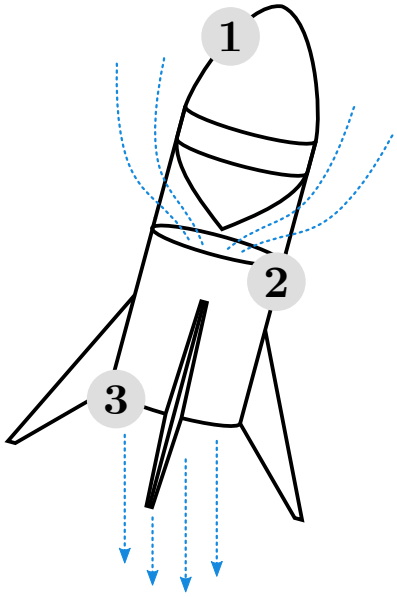


Figure 2.1: Functional illustration of the mono-copter platform. (1) Flight controller and battery. (2) Propulsion, air is sucked through the duct. (3) Exhaust, airflow is controlled by thrust vectoring.

2.1 System Overview

The hardware components needed for a thrust vectoring mono-copter is similar to that of a traditional quad-copter, with only the actuation and propulsion being different. UAVs in general requires a very specific set of electronics, need to stabilise the system and navigate its surroundings. The electronics needed for this project, are grouped into seven subsystem, see **Figure 2.2**.

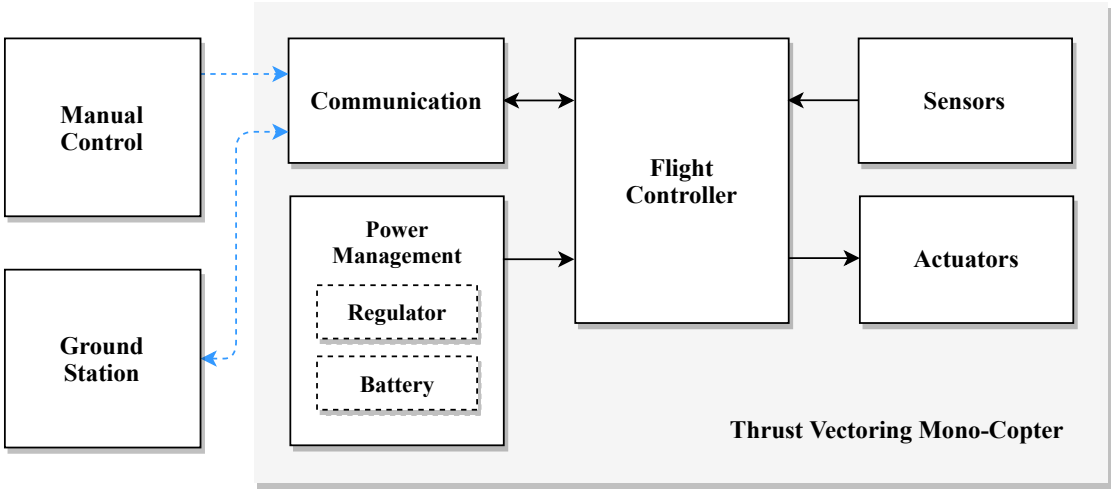


Figure 2.2: Overview of the seven subsystem and their interacting. The black lines illustrate a wired connected, while the blue illustrate a wireless connection.

2.1.1 Functional requirements

The successful design and construction of any airborne vehicle, requires a requirement on the total weight of the system. Furthermore, in order to ensure safety and more manageable working conditions, the hardware platform must be able to fly indoors. To achieve this, the mono-copter should be reasonable small with a mass **less than** 1 kg. By decreasing the weight of the mono-copter, the risk of human injury during a rapid unscheduled disassembly (crash) is reduced. To completely answer the problem statement, and successfully mimic the concept of reusable rockets, three main objectives must be reached. These are:

- Must be able to take-off vertically
- Must be able to enter a stable hover
- Must be able to perform a vertical landing

With this in mind, the design and construction of each of the subsystems, as illustrated by **Figure 2.2**, will be described in the following sections.

2.2 Actuation

The actuation of the mono-copter is composed of two separate systems; the propulsion and the thrust vectoring. The method used as propulsion affects the design of the thrust vectoring system, and is therefor described first.

2.2.1 Propulsion

To counteract the forces of gravity and propel the mono-copter into the air, a method of propulsion is needed. Many methods exists that generates thrust by propelling air, such as: Jet engines, electrical ducted fans (EDF) or conventional motors with propellers. Jet engines produces a large amount of thrust, however requires liquid fuel to operate. This produces exhausts that are unsuitable for indoor flight. EDFs does not have this issue, however produces a large rotational momentum around the axis of rotation, causing the mono-copter to rotate around it self. One way to counteract the rotational moment is by using a second counter-rotating motor. This is not feasible using EDFs, but can be achieved using smaller brushless DC (BLDC) motors and propellers. BLDC motors typical has a lowe than EDFs, however when configured in a counter-rotating setup the nominal thrust is increased [14], and the rotational momentum is reduced.

The motor setup used on the mono-copter must be able to generate more thrust than the maximum total weight of 1 kg. To achieve this, two motors of the name *F40PRO 2600KV* are chosen, which each can generate 1,2 kg of thrust, when paired with a *5045 tri-bladed propeller*. The two motors are mounted back-to-back in a coaxial design, using a custom 3D-printed mount, see **Figure 2.3**. To validate the motor performance, a series of thrust measurements has been made, see **Appendix A**.

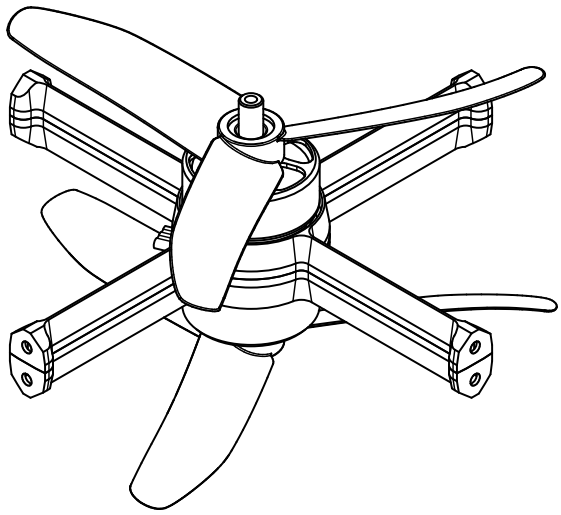


Figure 2.3: 3D-CAD drawing of the two motors mounted back to back.

2.2.2 Speed Controller

BLDC motors are not regular DC motors, and must be controlled using specialised electronics; an electronic speed controller (ESC). The motors chosen is rated at a maximum of 45 A, at a voltage of 16,8 V (a fully charged four celled lithium-ion battery), however because two motors shares the load, the current of one single motor should never reach the rated maximum. Nonetheless, the ESCs chosen for the mono-copter is a *KISS 32A*, which is rated to a peak current of 45 A. The *KISS 32A* ESCs are chosen based on their current rating, but more importantly, because it uses the digital communication protocol DSHOT, and has telemetry capabilities. The digital protocol enables configuration and precise control of the motors, while the telemetry enables measurement of motor RPM, battery voltage and current consumption. Using these ESCs, the motors can be controlled precisely and monitored by the flight controller, see **Figure 2.4**.

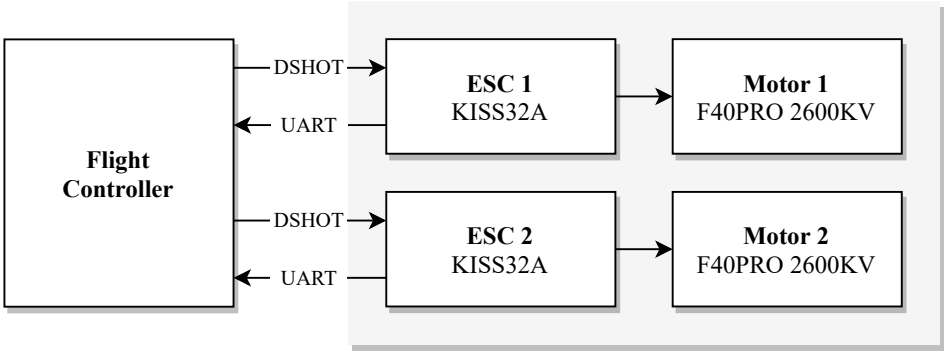


Figure 2.4: Overview of the propulsion system. The two ESCs each controls a single motor, and feeds telemetry back to the flight controller over UART.

2.2.3 Thrust Vectoring

To stabilise the mono-copter using only a single source of thrust, a mechanism that can vector the thrust is needed. This can be achieved with two methods; either by gimballing the thrust or by deflecting the thrust. Gimballing is used on most modern rockets [15], and works by controlling the direction of the exhaust nozzle of the engine, see **Figure 2.5a**. A gimballed motor mount is mechanical complex, and the slightest backlash in the joints could result in uncontrollable vibrations. The second method uses control surfaces that deflects small amounts of thrust to produce a desired movement. The control surfaces (thrust vanes) are less mechanical complex and does not have issues with backlash, as no movable joints are needed, see **Figure 2.5b**. Depending on the configuration, thrust vanes also enable full control of the attitude, which cannot be achieved using a single gimballed motor. Thrust deflectors are considered superior in a mono-copter, and will be used as the actuating force of the mono-copter.

(a) Rocket engine using gimballed TVC [16]

(b) Rocket using TVC deflectors [17]

Figure 2.5: Two methods of performing thrust vector control (TVC)

To get full attitude control, a total of four thrust vanes are necessary. To accurately control the angle of these thrust vanes, four independent servo motors of the type *DAVIGA DS213* are used. These servo motors are chosen, as they offer a high rotational velocity of 750 °/s, and is built using metal gears for high strength [18]. The four servo motors are mounted on the outside of the main body, with only the motor axle perturbing the cylinder, see **Figure 2.6**. The thrust vanes are mounted on the inside, and are shaped according to a commonly known airfoil design, see **Appendix B**.

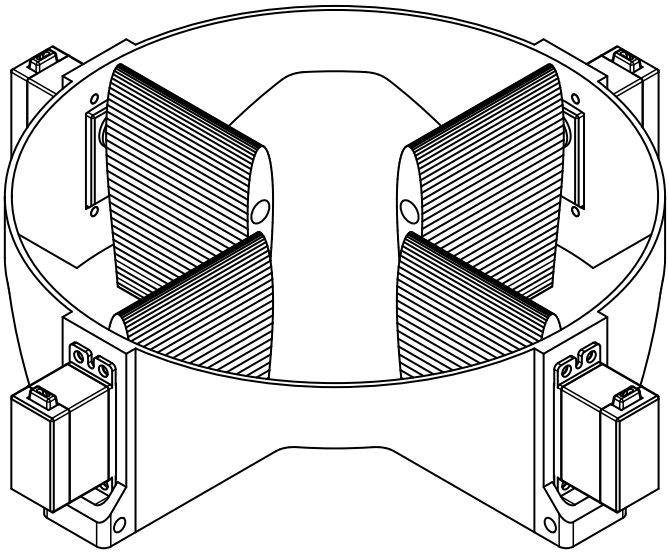


Figure 2.6: Isometric view of the thrust vectoring system, designing in 3D-CAD software.

2.3 Communication

To enable telemetry and wireless control of the mono-copter, a method of wireless communication is needed. Many methods exist to enable wireless communication, such as Bluetooth, WiFi or other RF systems. The WiFi protocol is chosen, as most modern laptops comes with a WiFi-card, enabling the use of a laptop to be the ground station with plenty of range. The WiFi-capable microprocessor *ESP32* is chosen to act as the link between the mono-copter and the ground station, as it comes in a very small and light-weight package (2,2 g). The *ESP32* can be programmed to act a router, and generate a small local network, which any wifi-able device can connect to. The use of WiFi makes it possible to use the data transport protocols TCP or UDP, to send an receive commands and telemetry.

The ground station will acts as a high-level command centre, running on an laptop with multiple processes sharing the resources. In situations where the ground station is unresponsive, a secondary system must be able to take control of the system, to ensure safety. The secondary communication is implemented using standard RC equipment used by hobbyists. A small radio receiver with four PWM-channels is connected to the flight controller, and a radio controller is used to control the four PWM-channels. Using a single PWM-channel, the maximum velocity of the motors is controlled, such that the motors at all times can be stopped. The remaining channels can be used to control the attitude or position in manual mode.

The two communication systems are connected in parallel, see **Figure 2.7**, giving some measure of redundancy, should one of the systems fail.

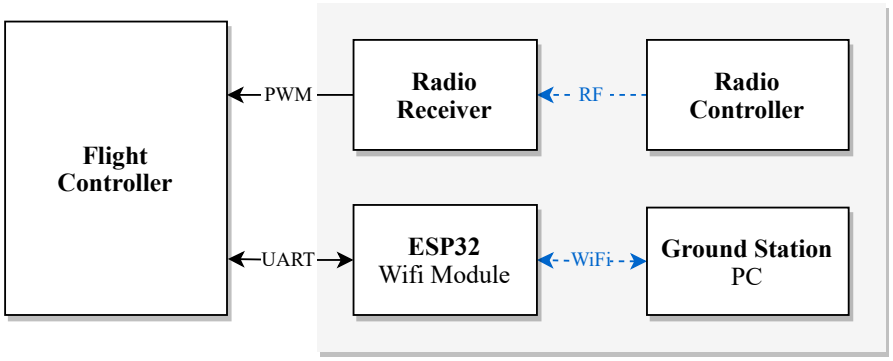


Figure 2.7: Overview of communication subsystem. The ESP32 allows for bi-directional communication, while the radio receiver, only acts as external input.

2.4 Sensors

To stabilise the mono-copter, measurements of the systems attitude and position is needed. To measure this, a multitude of sensors are needed, each providing unique information about the state of the system. The sensors chosen will be described in the following sections.

2.4.1 Orientation

In order to stabilise and navigate an object moving freely in the air, its orientation has to be known. Orientation is often measured using Tait-Bryan notation [6], also know as roll, pitch and yaw. To determine the orientation, measurements from accelerometers, gyroscopes and magnetometers can be combined using sensor fusion techniques [6]. The methods needed to fuse inertial measurements into an orientation estimate, is however not the focus of this project. Instead, an inertial measurement unit (IMU) with an internal orientation estimator is chosen; the *BNO080* from Bosh, see **Figure 2.8**.

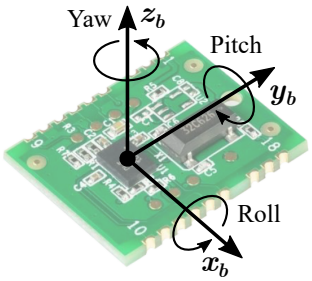


Figure 2.8: Coordinate system of the BNO080 on the breakout board *FSM300*.

The BNO080 is a combined package of multiple inertial sensors, that are sampled by an on-board microprocessor that performs the sensor fusion. The estimated orientation, including the raw measurements from the BNO080 are sampled using a special 7-wire SPI protocol, see **Figure 2.9**.

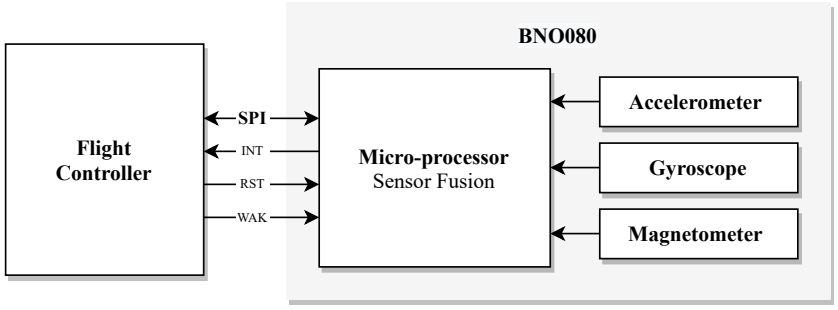


Figure 2.9: Overview of the SPI communication and the internal sensor fusion of the IMU.

2.4.2 Absolute position

When flying indoors, the measurement of position can be problematic, as the primary source of positional measurements, the GPS system, needs a clear view of the sky. Positional measurements are crucial to stabilise the mono-copter, but also to validate the performance of the system. For this reason the indoor tracking system known as Vicon is used instead. Vicon uses a multitude of cameras to track a set of optical features mounted on the system. The Vicon system is connected to the ground station, and positional measurements are transmitted over WiFi to the flight controller, see **Figure 2.10**. Vicon can be sampled at a maximum of 100 Hz, however due to technical limitations, measurements are only transmitted at 5 Hz to the flight controller.

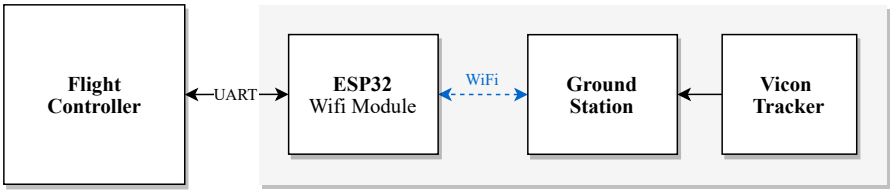


Figure 2.10: Communication between the Vicon system and the flight controller.

2.4.3 Relative Altitude

To achieve stable flight, the relative altitude (height above ground) must be known. The measurements supplied by Vicon could be used for this, however, the slow sample-rate of 5 Hz is not considered sufficient. To enable faster measurements of the altitude the time-of-flight (TOF) sensor *VL53L1X* is used. An on-board altitude sensor, is also necessary if the mono-copter was to be tested outdoors. The TOF sensor can measure up to 4 m at a sample-rate of 50 Hz, and uses the communication protocol I2C. The sensor is connected directly to an I2C port on the flight controller, see **Figure 2.11**. Care has to be taken when sampling the TOF sensor, as the orientation of the mono-copter affects the measured altitude. If the mono-copter tilts, the measurement will have an error proportional to the tilting angle, which must be corrected using the measured attitude.

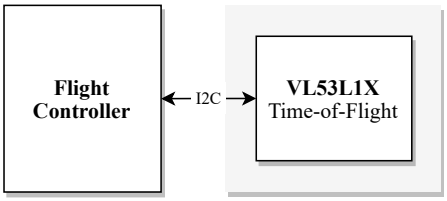


Figure 2.11: Communication between TOF-sensor and the flight controller.

2.4.4 Linear velocity

To supplement the occasional positional measurements from Vicon, measurements of the translational velocity are used. From previous experience [19], it was found that a camera sensor and computer vision algorithms can be used to measure the translational velocity of an object. Drawing on this experience, a small camera sensor, the *PMW3901* is chosen to measure the horizontal velocity. The sensor is a black-box sensor that computes the optical flow components Δx_{of} and Δy_{of} (changes in pixels) and returns the accumulation of these since the last sample. The sensor must have a clear view of the surface it is tracking, and therefor must be mounted at the bottom, looking down, see **Figure 2.12b**. The sensor uses standard 4-wire SPI protocol, and is directly connected to a SPI port on the flight controller, see **Figure 2.12a**.

(a) SPI communication for PMW3901

(b) Optical flow principle

Figure 2.12: Illustration of the working principle of PMW3901

In order to have any use, the unitless optical flow components given by the sensor must be scaled to a physical unit (m/s). To do so, the optical flow is divided by the sample-time Δt, and the pine-hole model is applied to properly scale the values [20]. The pin-hole model describes the projection of 3D-point onto a 2D-point, and requires knowledge about the cameras focal length f and the image depth h. The image depth can be considered the altitude (height above ground), while the focal length is found experimentally. The camera measurements does however not only include linear velocity, and is corrupted by rotational components, as the camera cannot distinguish between the two. To remove the unwanted rotational components, the velocity is compensated using the angular velocities (ω_x, ω_y), see **Equation 2.1** [20].

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \frac{1}{\Delta t} \begin{bmatrix} \frac{h}{f} & 0 \\ 0 & \frac{h}{f} \end{bmatrix} \begin{bmatrix} \Delta x_{of} \\ \Delta y_{of} \end{bmatrix} + \begin{bmatrix} -h & 0 \\ 0 & h \end{bmatrix} \begin{bmatrix} \omega_y \\ \omega_x \end{bmatrix} \tag{2.1}$$

2.5 Flight Controller

The flight controller is the most central part of the system design, as it connects all the sensors and actuators. As a consequence, the flight controller has to sample all sensors, do heavy computations and generate appropriate control signals to stabilise the system. To accomplish this, a reliable and fast computing unit is required. The computing unit must also have all the required input-output (I/O) ports and communication interfaces needed for the sensors. This is best achieved with an embedded platform, where the I/O ports and the hardware peripherals (SPI, UART etc.) are directly accessible through software. The micro-processor chosen is the *Teensy 4.0*, which runs on a *ARM Cortex-M7* at 600 MHz, with a total of 40 configurable I/O pins. The *Teensy 4.0* comes with all the needed peripherals and is directly programmable using the Arduino IDE, making it perfect for fast prototyping.

2.5.1 PCB

The micro-controller has to interface with a lot of different components, by a wired connection, see **Appendix D**. To organise and connect all the components, a custom PCB has been designed. The PCB is named the *carrier board*, and houses the micro-processor, the WiFi module, and a total of eleven connectors, see **Figure 2.13**.

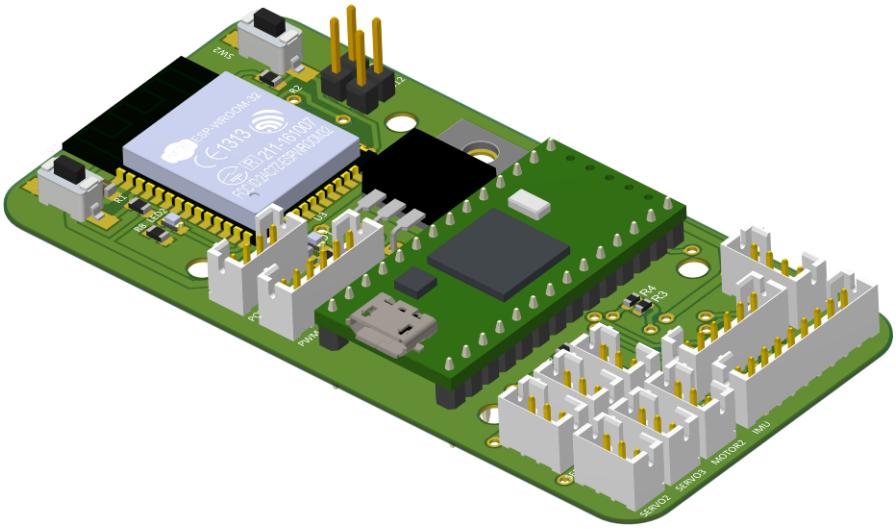


Figure 2.13: Custom PCB designed for the flight controller. The two buttons mounted next to the WiFi-module, are used to flash the ESP32 module using the four pin-headers.

2.6 Power Management

All the electronics of the mono-copter needs a source of power to function, and since the system is airborne, a wired connection to a power-supply is out of the question. Instead a battery must be included. To comply with the high current demand of the motors, a high-capacity high-load battery is used. The battery chosen is a 2200 mA h, composed of four lithium-ion cells yielding a total voltage of 16,8 V.

The ESCs are designed to handle high voltages [21], however the rest of the system runs at either 3,3 V or 5 V. To regulate the voltage, a 5 V DC-DC step-down regulator *D24V22F5* is used [22]. The 5 V is feed to an linear regulator on-board the carrier board, that supplies 3,3 V to the more sensitive components.

2.6.1 PCB

To separate the high-powered and high-voltage electronics from the more sensitive components, a custom PCB is designed for the power management. This PCB is named the *power board*, and houses the ESCs, the DC-DC regulator and the direct battery connection, see **Figure 2.14**. Power is supplied by the *power board* to the *carrier board* by a four-pinned connector; two wires for each terminal (5 V and GND).

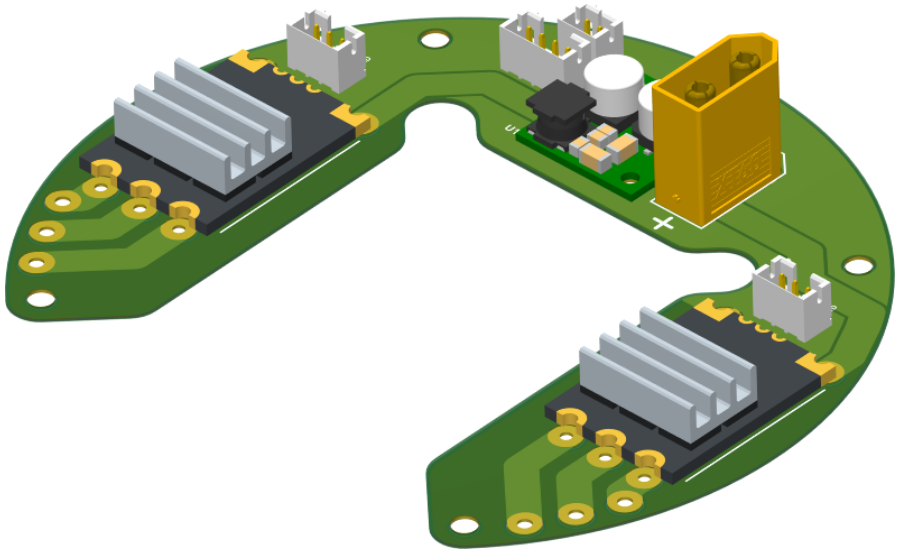
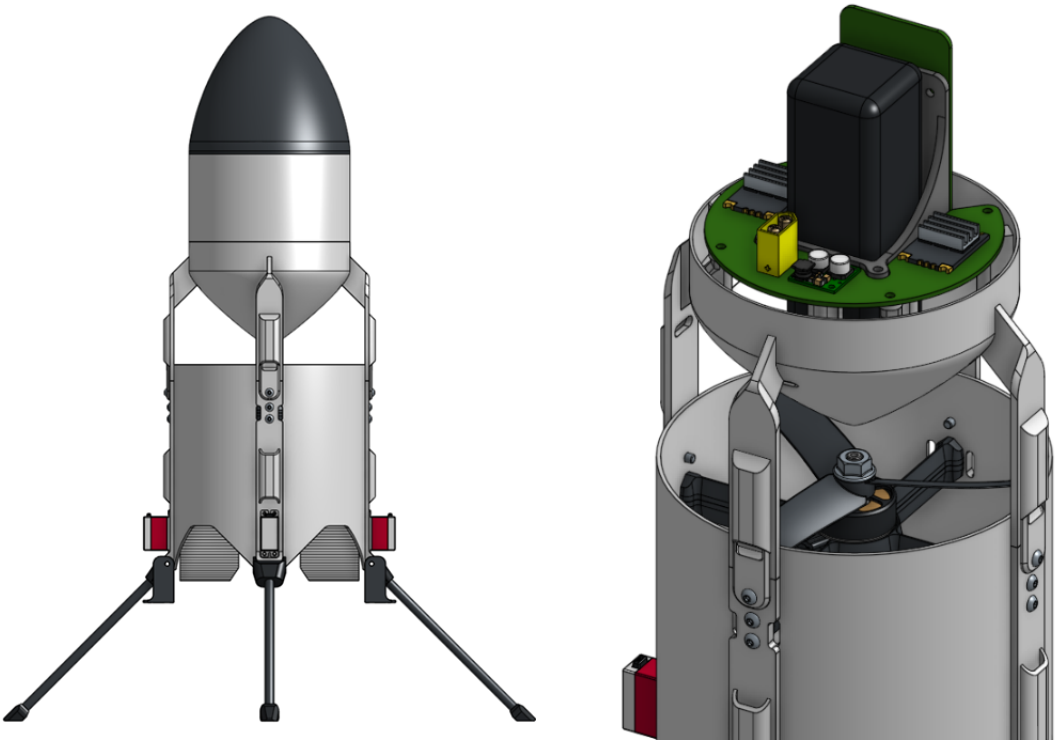


Figure 2.14: Custom PCB designed to power the electronics of the mono-copter. The battery goes in the middle, to better balance the weight of the system.

2.7 Summary

The mono-copter is designed to combine all the hardware and electronics, in a light-weight and sturdy construction. The complete mono-copter and most of its components are modelled in 3D-CAD software, see **Figure 2.15**.



(a) Side-view of entire system **(b)** 3D-view of internal structure

Figure 2.15: 3D model of the complete system

Most of the construction are produced using 3D-printing techniques, allowing for fast prototyping and light-weight plastic parts. Nonetheless, the complete physical system has a weight of 0,968 kg compared to the estimated 0,828 kg of the 3D-model. The increase in weight is mostly contributed to the wiring, which is not included in the 3D-model. To account for the difference in weight, an additional mass is added to the centre of the 3D-model, such that the computed inertia more accurately matches that of the physical system.

With the physical platform developed, a mathematical model of the system can be derived for control and simulation. The following chapter will describe the working principle of the mono-copter in detail and derive a mathematical model that describes the dynamics of the system.

CHAPTER 3

Modelling

In the following chapter, a multi-dimensional model are derived that describes the rigid body dynamics of a mono-copter. This chapter intends to derive a system model that can be used for simulation and can be applied to design a suitable control strategy that stabilises the position and attitude of the mono-copter.

First, the model preliminaries are described, including vector notation, coordinate frames and attitude representation. Next, the forces and torques applied by the thrust vanes and motor throttle are analysed in order to describe the translational and rotational dynamics. Lastly, the nonlinear dynamics are linearized around an operating point, such that a linear model-based controller can be designed.

3.1 Model Preliminaries

Before diving into the model formulations, some basic notation and prerequisites are due for clarification. This includes a description of the coordinate systems used, the defined state vectors and transformations between them.

3.1.1 Coordinate frames

In total, two coordinate frames are used to model the UAV's movement; the body and the world frame. The world frame $\mathcal{W} = \{ x_w, y_w, z_w \}$ is an inertial frame that is fixed to a static point from which the position is measured in x, y and z (flat earth). The body frame $\mathcal{B} = \{ x_b, y_b, z_b \}$ on the other hand is a moving frame that is rigidly attached to the drones centre of mass (COM), and thus moves along with it, see **Figure 3.1**.

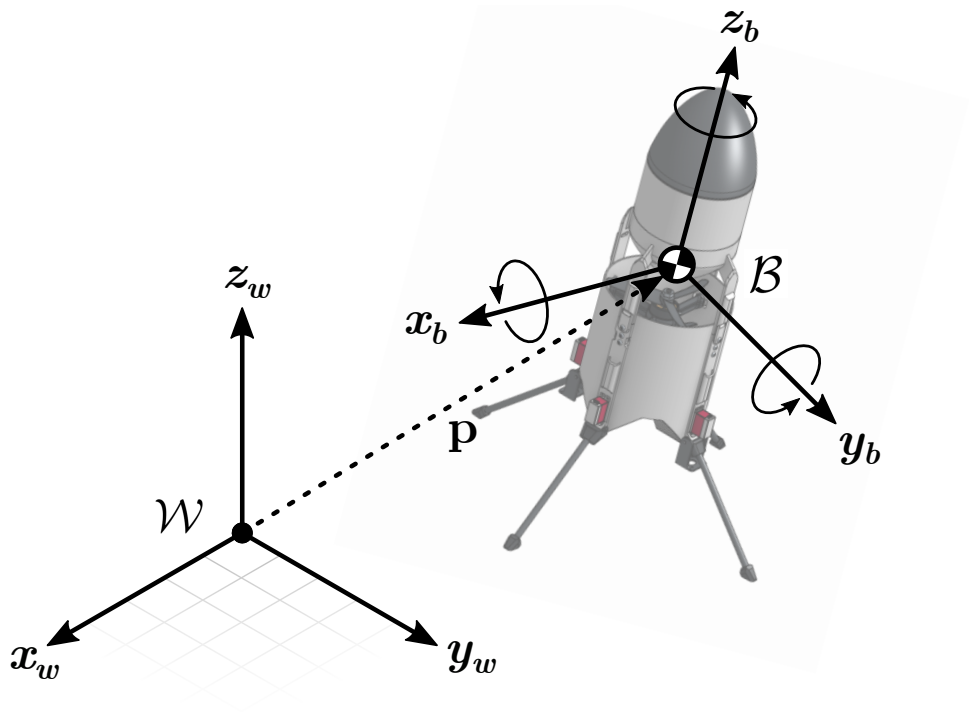


Figure 3.1: The two coordinate frames used to describe the movement of the drone.

To simplify the derivation of the system models, the body frame axes are aligned with the structure of the drone, with the x_b- and y_b-axis being parallel with thrust vanes. The z_b-axis is aligned with the motors centre of rotation and points out through the top, being perpendicular to the xy-plane.

3.1.2 Kinematics

To describe the absolute position and orientation of the UAV, a sequence of kinematic equations are needed. For future reference, let $\boldsymbol{p}$ denote the absolute position of the UAV's centre of mass in the earth fixed world frame, let $\boldsymbol{\eta}$ denote the orientation described using the Tait-Bryan angles (roll, pitch and yaw), and let $\boldsymbol{v}_{\mathcal{B}}$ and $\boldsymbol{\omega}_{\mathcal{B}}$ denote the translational and angular in the body frame, respectively.

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \boldsymbol{v}_{\mathcal{B}} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad \boldsymbol{\omega}_{\mathcal{B}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.1}$$

The orientation of a rigid body with reference to an inertial frame, can be explained in terms of three consecutive rotations, as was introduced by Euler [23]. These rotations can be applied in several different ways, however for this project the ZYX Tait-Bryan notations will be used. Using the Tait-Bryan notation, the orientation can be formulated as a sequence of rotations summarised in the rotation matrix $\mathbf{R}_b^w : \mathcal{B} \Rightarrow \mathcal{W}$ [23].

$$\mathbf{R}_b^w(\psi, \theta, \phi) = \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\theta) \cdot \mathbf{R}_x(\phi) \tag{3.2}$$

$$\mathbf{R}_b^w(\psi, \theta, \phi) = \begin{bmatrix} \mathrm{c}\,\theta\,\mathrm{c}\,\psi & \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{c}\,\psi - \mathrm{c}\,\phi\,\mathrm{s}\,\psi & \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{c}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\psi \\ \mathrm{c}\,\theta\,\mathrm{s}\,\psi & \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi + \mathrm{c}\,\phi\,\mathrm{c}\,\psi & \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi - \mathrm{s}\,\phi\,\mathrm{c}\,\psi \\ -\mathrm{s}\,\theta & \mathrm{s}\,\phi\,\mathrm{c}\,\theta & \mathrm{c}\,\phi\,\mathrm{c}\,\theta \end{bmatrix} \tag{3.3}$$

where $\mathrm{s}\,\phi, \mathrm{s}\,\theta, \mathrm{s}\,\psi = \sin(\phi), \sin(\theta), \sin(\psi)$: Sine function [−]
$\mathrm{c}\,\phi, \mathrm{c}\,\theta, \mathrm{c}\,\psi = \cos(\phi), \cos(\theta), \cos(\psi)$: Cosine function [−]

Using the rotation matrix $\mathbf{R}_b^w$, any vector in the body frame (velocity, acceleration etc.), can be transformed to the world frame by pre-multiplying the rotation matrix [23].

The orientation $\boldsymbol{\eta}$ and its derivative $\dot{\boldsymbol{\eta}}$ are however not vectors in the world frame but rather vectors describing the three rotations around independent reference frames. Consequently the Tait-Bryan rotational rates $\dot{\boldsymbol{\eta}}$ cannot be described using $\mathbf{R}_b^w$ and the body rates $\boldsymbol{\omega}_{\mathcal{B}}$. Instead, to perform the transformation from body to Tait-Bryan rotational rates, the matrix $\mathbf{W}_\eta$ is used [6] [23].

$$\boldsymbol{\omega}_{\mathcal{B}} = \mathbf{W}_\eta\, \dot{\boldsymbol{\eta}} \quad \Rightarrow \quad \dot{\boldsymbol{\eta}} = {\mathbf{W}_\eta}^{-1}\, \boldsymbol{\omega}_{\mathcal{B}} \tag{3.4}$$

$$\mathbf{W}_\eta = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{3.5}$$

It should be noted that the inverse transformation $\mathbf{W}_\eta$ has singularities at $\theta = \{ -90^\circ \; 90^\circ \}$, causing the transformation to fail. To provide a seamless mapping, quaternions rather than Euler angles can be used [6]. However, because a ninety-degree angle of pitch is outside the normal operating condition, this will not be addressed further.

3.1.3 Modelling principle

The UAV is a freely moving object that can move in three spacial and three rotational directions. This is also known as a system with six degrees of freedom (6-DOF). To describe the drones translational and rotational dynamics in six DOF, the forces and moments acting on the drone in three dimensions has to be analysed. This is done using the Newton-Euler equations, which describes the moments and forces acting on a body with respect to a coordinate frame whose origin coincides with the centre of mass (COM), see **Equation 3.6** [12].

$$\begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} m\,\mathbf{I}_3 & 0 \\ 0 & \mathbf{J} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{J}\,\boldsymbol{\omega} \end{bmatrix} \tag{3.6}$$

where

$\boldsymbol{F}$:	$\mathbb{R}^3$	- External force acting on the COM	[N]
$\boldsymbol{\tau}$:	$\mathbb{R}^3$	- External torque acting about the COM	[N/m]
m :	$\mathbb{R}^1$	- Mass of the body	[kg]
$\mathbf{J}$:	$\mathbb{R}^{3\times3}$	- Moment of inertia about the COM	[kg/m^2]
$\dot{\boldsymbol{v}}$:	$\mathbb{R}^3$	- Linear acceleration	[m/s^2]
$\dot{\boldsymbol{\omega}}$:	$\mathbb{R}^3$	- Angular acceleration	[rad/s^2]

The moment of inertia $\mathbf{J}$ is a square matrix describing the rotational inertia around the drones principle axes. The off-diagonal terms in the inertia matrix (products of inertia) can be approximated to zero when the mass of the body are evenly distributed around the local frame of reference's axes [23]. The UAV has been designed in such a way that this is achieved, and thus the product-of-inertia terms becomes zero, see **Equation 3.7**.

$$\mathbf{J} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{bmatrix} \quad \Rightarrow \quad \mathbf{J} \approx \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{3.7}$$

This is validated by the inertia computation performed by the CAD-software, with the diagonal terms being an order of magnitude larger than the off-diagonal terms.

3.2 Moments and forces

To model the rotational and translation dynamics of the mono-copter, a description of the forces and moments acting on the body is needed. To assist in the description thereof a three-dimensional free body diagram has been made, see **Figure 3.2**. The primary forces acting on the body are those produced by the motor and the thrust vanes. Aerodynamic forces are neglected as they are considered inconsequential when the mono-copter operates near the hovering state.

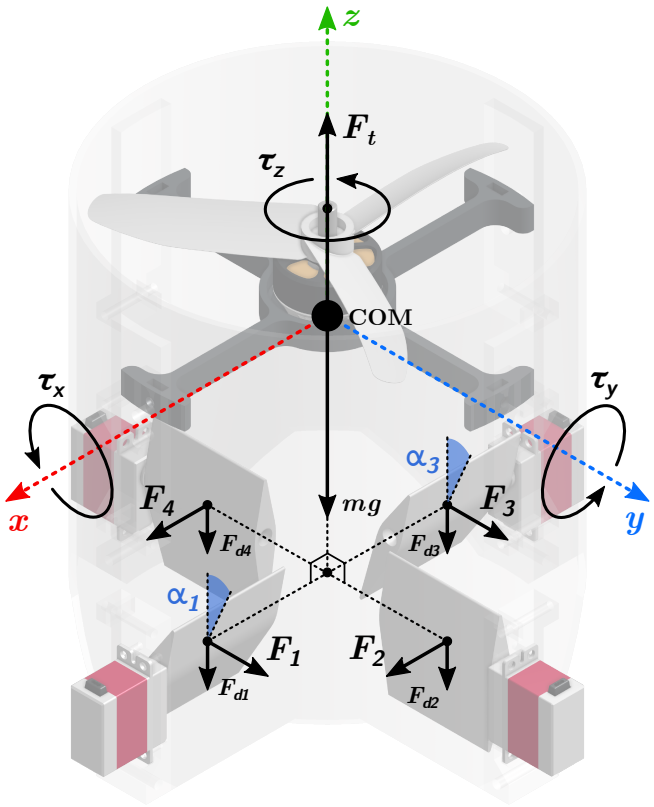


Figure 3.2: The forces and moments acting on the UAV's COM in the body frame. For illustrative purposes only a single motor is included, yet two will be used.

The forces $F_1 \dots F_n$ $(n = 1, 2 \dots 4)$ denotes the controlling forces produced by the thrust vanes, F_t denotes the total force produced by the motors and F_d denotes the combined drag-forces $F_{d1} \dots F_{dn}$ of the four thrust vanes. The offset of the thrust vane forces $F_1 \dots F_n$ to the centre of mass (COM), produces the controlling torques denoted by $\tau_1 \dots \tau_n$. These torques acts around the UAV's principle axes of rotation, causing rotational movement. The motor torque τ_m around the z-axis are not included, as this effectively sums to zero because of the counter-rotating motor setup.

3.2.1 Motor Propulsion Force

To describe the propulsion force F_t in terms of the digital control input u_t, two models are used, one describing the motor dynamics and another describing the propulsion force of the propeller, see **Figure 3.3**.

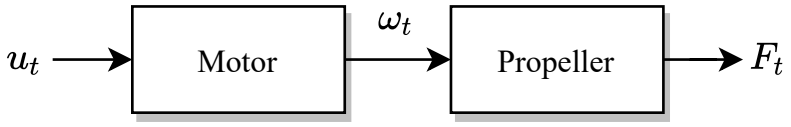


Figure 3.3: Block diagram of the propulsion model

The input to the motor system is a digital signal, while the output can be considered the rotational velocity ω_t of the motor. To describe the dynamics of the motor system, the relationship between input and output must be found. The motors are controlled by electronic speed controllers (ESC), which by the manufacturer has a linear throttle response [21]. The motor does however not actuate instantaneous, as the motor and propeller has inertia. To describe these dynamics, the propulsion system is described in terms of a first order system, with time-constant τ_t and DC-gain K_t.

$$\frac{\omega_t(s)}{u_t(s)} = \frac{K_t}{\tau_t s + 1} \tag{3.8}$$

To find the system parameters, a step is applied on the input u_t, and the output ω_t is monitored. The resulting step-response is fitted to a first order system, using MATLAB's "System Identification Toolbox", see **Appendix A**.

The first order system does however only describe the dynamics of the motor, and not the actual propulsion force. As described by many previous works [6][7], the propulsion force F_t generated by a propeller can be described by a second order polynomial of the rotational velocity ω_t and a constant scaling factor K_f.

$$F_t(\omega) = K_f \cdot \omega_t^2 \tag{3.9}$$

The scaling factor K_f is found by conducting a polynomial fit on a series of measurements, where the rotational velocity of the motor and generated thrust is measured, see **Appendix A**. Considering the value of the time-constant being $\tau_t = 0{,}0345\,\text{s}$, the motor dynamics are considered to be much faster than the combined system dynamics, and can thus be reduced to the DC-gain K_t and the scaling factor K_f.

3.2.2 Thrust Vane Forces

The UAV's rotational movement is controlled by the four thrust vanes located at the exhaust. Each of the thrust vanes are actuated by an independent servo motor, controlled by a PWM signal. To describe the thrust vane forces $F_1 \dots F_n$ in terms of the PWM signals $u_1 \dots u_n$, both the servo and thrust vane dynamics must be considered, see **Figure 3.4**.

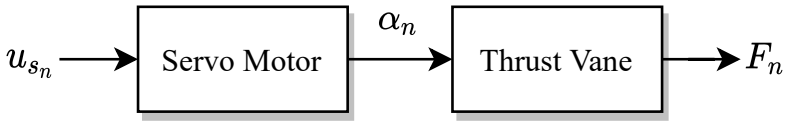


Figure 3.4: Block diagram of the thrust vector model

The servo motors used to actuate the thrust vanes, translates a PWM signal into an absolute rotational position α_n, with very high actuation speed (750 °/s) [18]. Due to the internal servo control system being so fast, and the low inertia of the attached thrust vanes, the rotational dynamics of the servo motors are considered to be negligible. For this reason, the model describing the servo motors can be reduced to a simple gain, that translates the PWM signal to an angular position of the servo motor.

$$\alpha_n = K_a \cdot u_{s_n} \tag{3.10}$$

To describe how the change in servo position $\alpha_1 \dots \alpha_n$ affects the control forces $F_1 \dots F_n$ the aerodynamics must be must considered. The thrust vanes are essentially a symmetrical downwards facing wing (airfoil), with forced air flowing across them. The forces generated by the thrust vanes can be described in terms of lift F_{lift} (controlling force) and drag F_{drag} (loss due to friction and turbulence), see **Figure 3.5**.

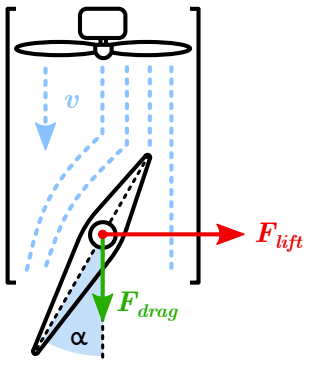


Figure 3.5: Air with velocity v is forced across the thrust vane. The pitching of the thrust vane angle α adjusts the amount of lifting force F_{lift}.

The aerodynamic forces generated by the thrust vanes can be described using the lift and drag equations [24].

$$F_{lift} = \frac{1}{2}\,\rho\,v^2\,C_l\,A_{fin} \tag{3.11}$$

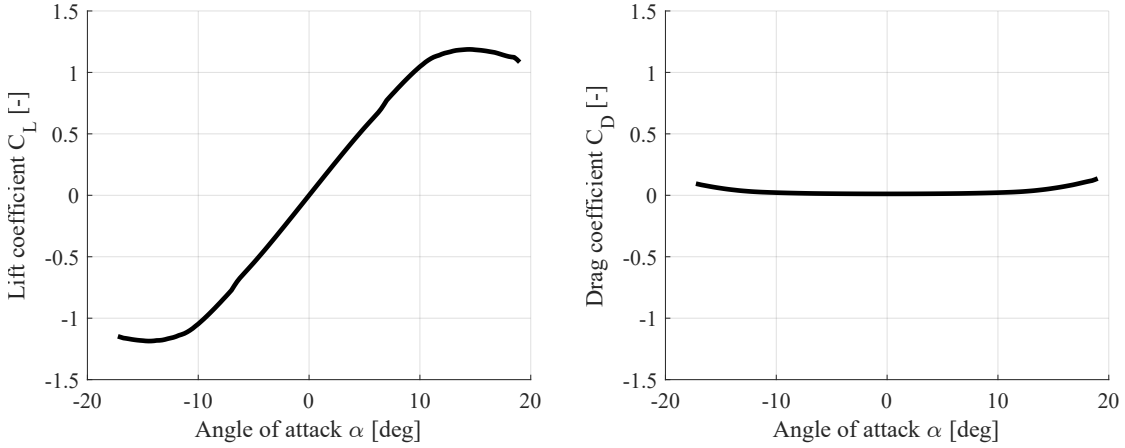
$$F_{drag} = \frac{1}{2}\,\rho\,v^2\,C_d\,A_{fin} \tag{3.12}$$

where	C_l, C_d	:	lift- and drag-coefficient	$[-]$
	ρ	:	air density	$[\mathrm{kg/m^3}]$
	v	:	air velocity	$[\mathrm{m/s}]$
	A_{fin}	:	surface-area of airfoil	$[\mathrm{m^2}]$

The lift and drag coefficients are unit-less, and describes the characteristics of the wing in a single coefficient. These coefficients are cannot be calculated, but are rather found experimentally or using simulation [24].

Lift and drag coefficient

To describe the amount of lift and drag generated by the thrust vanes, the corresponding coefficients must be found. Both coefficients have a nonlinear dependency on the angle of attack α [24]. which can be described in terms of higher order polynomials [24]. However, because of the symmetric airfoil design, the coefficients can be approximated quite accurately using linear functions. Seen from **Figure 3.6a**, the lift coefficient has a linear correlation with the angle of attack α, for small angles ($\alpha < 10°$). Similarly, the drag coefficient can be approximated as a constant in the same interval.



(a) Lift coefficient **(b)** Drag coefficient

Figure 3.6: Simulated values of the lift and drag coefficient vs. angle of attack, found using the interactive simulation tool XFoil, see **Appendix B**.

As seen from **Figure 3.6**, the lift and drag coefficients can be approximated to a linear function and a constant, respectively. This assumes that the thrust vanes operates around 0° and stays within small angles of ±10°. Using this assumption, the lift and drag coefficients can be described as follows:

$$C_l(\alpha) = C_{L\alpha}\,\alpha \tag{3.13}$$

$$C_d(\alpha) = C_{D0} \tag{3.14}$$

where $C_{L\alpha}$: slope of the lift coefficient [−]
C_{D0} : constant bias of the drag coefficient [−]

The coefficients $C_{L\alpha} = 0.008905$ and $C_{D0} = 0.001054$ are found by performing linear regression on the simulated values of the lift and drag coefficients, see **Appendix B**.

Aerodynamic flow

The lift and drag forces generated by the thrust vanes, are described in terms of the air velocity v. The air velocity is unknown unless measured, which is not easily achieved. To simplify this, Newton's second law is applied to describe the motor force F_t, in terms of the air density, the exhaust area and the air velocity squared [25]. This in turn allows the air velocity to be expressed in terms of the motor force F_t, see **Equation 3.15**.

$$F_t = A_{duct}\,\rho\,v^2 \quad \Rightarrow \quad v^2 = \frac{F_t}{A_{duct}\,\rho} \tag{3.15}$$

where A_{duct} : Cross-sectional area of the exhaust [m^2]

Using **Equation 3.15**, the forces generated by the thrust vanes can be described in terms of the motor force F_t, the angle of attack α and two constants C_L and C_D.

$$F_n = F_t \underbrace{\frac{C_{L\alpha}\,A_{fin}}{2\,A_{duct}}}_{C_L}\,\alpha_n \tag{3.16}$$

$$F_{d_n} = F_t \underbrace{\frac{C_{D0}\,A_{fin}}{2\,A_{duct}}}_{C_D} \tag{3.17}$$

3.3 Rotational Dynamics

The rotational dynamics of the mono-copter can be described using Euler's equations for rigid body dynamics. Euler's equations for a rotating body, see **Equation 3.18**, assumes that the UAV is a rigid body with a rotating reference frame fixed to the body's principal axes of inertia. This means that the moment of inertia stays independent of the orientation, and thus greatly simplifies the formulation of the model.

$$\boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} \tag{3.18}$$

When performing rotation around more than one of the principle axis, the second term $(\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega})$ in **Equation 3.18** describes the centripetal moment acting on the body frame. In the simpler case of a single axis rotation, the centripetal moment is not present and the model are easier to visualise, see **Figure 3.7**.

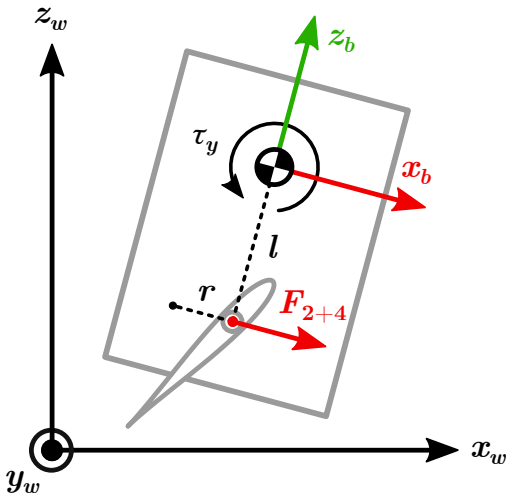


Figure 3.7: Torques acting around the y-axis. The value l is the distance between the COM and the thrust vane joints, while r is the distance between the z-axis and the middle of the thrust vanes.

3.3.1 Rotation in body frame

To describe the rotational dynamics in the body frame, firstly the torque acting on the body must be described. The torque acting on the body, can be described in terms of the thrust vane forces $F_1 \dots F_n$ and the distances r and l, see **Figure 3.7**.

$$\boldsymbol{\tau}_{\mathcal{B}} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} (F_1 + F_3)\, l \\ -(F_2 + F_4)\, l \\ (F_1 - F_2 - F_3 + F_4)\, r \end{bmatrix} \tag{3.19}$$

Next, the body torque $\tau_{\mathbf{B}}$ is inserted into Euler's equation, and the angular acceleration vector $\dot{\boldsymbol{\omega}}_{\mathcal{B}}$ is solved for.

$$\dot{\boldsymbol{\omega}}_{\mathcal{B}} = \mathbf{J}^{-1}(\boldsymbol{\tau}_{\mathcal{B}} - \boldsymbol{\omega}_{\mathcal{B}} \times \mathbf{J}\,\boldsymbol{\omega}_{\mathcal{B}}) \tag{3.20}$$

$$\begin{bmatrix}\dot{\omega}_x\\ \dot{\omega}_y\\ \dot{\omega}_z\end{bmatrix} = \mathbf{J}^{-1}\left(\begin{bmatrix}(F_1+F_3)\,l\\ -(F_2+F_4)\,l\\ (F_1-F_2-F_3+F_4)\,r\end{bmatrix} - \begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} \times \mathbf{J}\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix}\right) \tag{3.21}$$

Lastly, by inverting the inertia matrix $\mathbf{J}$ and multiplying out the above equation, the rotational dynamics in the body frame are found, see **Equation 3.22**.

$$\begin{bmatrix}\dot{\omega}_x\\ \dot{\omega}_y\\ \dot{\omega}_z\end{bmatrix} = \begin{bmatrix}\frac{1}{J_{xx}}(F_1+F_3)\,l\\ \frac{1}{J_{yy}}(F_2+F_4)\,l\\ \frac{1}{J_{zz}}(F_1-F_2-F_3+F_4)\,r\end{bmatrix} + \begin{bmatrix}\frac{1}{J_{xx}}(J_{yy}-J_{zz})\omega_y\omega_z\\ \frac{1}{J_{yy}}(J_{zz}-J_{xx})\omega_x\omega_z\\ \frac{1}{J_{zz}}(J_{xx}-J_{yy})\omega_x\omega_y\end{bmatrix} \tag{3.22}$$

3.3.2 Rotation in inertial frame

With the intention to stabilise the orientation of the drone, the rotational dynamics of the body frame must be described in terms of the inertial frame (Tait-Bryan angles). However, the angular rates of the body frame ($\boldsymbol{\omega}_{\mathcal{B}}$) and angular rate of the Tait-Bryan angles ($\dot{\boldsymbol{\eta}}$) are not the same, but are related by the transformation matrix $\mathbf{W}_\eta$,

$$\dot{\boldsymbol{\eta}} = {\mathbf{W}_\eta}^{-1}\,\boldsymbol{\omega}_{\mathcal{B}} \tag{3.23}$$

Expanding the above equation, by multiplying out the transformation matrix $\mathbf{W}_\eta$ and the angular velocity vector $\boldsymbol{\omega}_{\mathcal{B}}$, yields three additional state equations. The complete model of the rotational dynamic can then be summarised, see **Equation 3.24**.

$$\mathbf{f}_{\mathbf{R}}(\boldsymbol{x},\boldsymbol{u}) = \begin{cases}\dot{\phi} = \omega_x + \omega_z\cos\phi\tan\theta + \omega_y\sin\phi\tan\theta\\ \dot{\theta} = \omega_y\cos\phi - \omega_z\sin\phi\\ \dot{\psi} = \omega_z\dfrac{\cos\phi}{\cos\theta} + \omega_y\dfrac{\sin\phi}{\cos\theta}\\ \dot{\omega}_x = \dfrac{1}{J_{xx}}(J_{yy}-J_{zz})\omega_y\omega_z + \dfrac{1}{J_{xx}}(F_1+F_3)\,l\\ \dot{\omega}_y = \dfrac{1}{J_{yy}}(J_{zz}-J_{xx})\omega_x\omega_z + \dfrac{1}{J_{yy}}(F_2+F_4)\,l\\ \dot{\omega}_z = \dfrac{1}{J_{zz}}(J_{xx}-J_{yy})\omega_x\omega_y + \dfrac{1}{J_{zz}}(F_1-F_2-F_3+F_4)\,r\end{cases} \tag{3.24}$$

3.4 Translational Dynamics

The translational movement of the mono-copter can be described in using Newton's second law of motion; that the acceleration of a body over time is directly proportional to the force applied.

$$\boldsymbol{F} = m\dot{\boldsymbol{v}} \tag{3.25}$$

The UAV is subject to forces in all three dimensions, however to simplify the derivation, the forces are more conveniently illustrated in 2D, see **Figure 3.8**

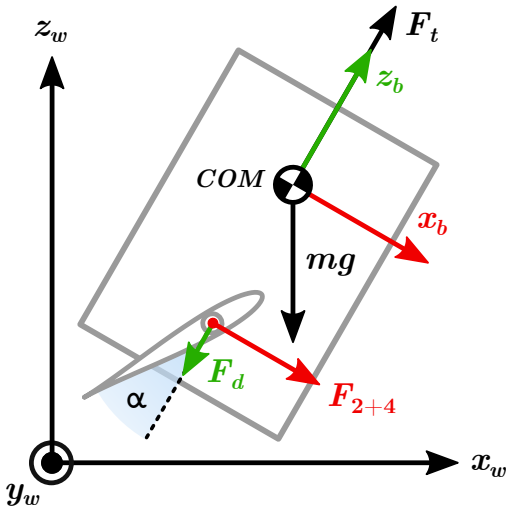


Figure 3.8: Free body diagram, showing the forces acting in the body frame.

3.4.1 Translation in body frame

To describe the translational motion in the body frame, firstly the forces acting on the body must be described. These forces are: The propulsion force F_t, the thrust vane forces $F_1 \dots F_n$ and lastly the gravitational force mg. The gravitational force depends on the orientation of the UAV, as gravity originates in the inertial frame. To describe this, the gravitational force vector is rotated from the inertial to body frame using the inverse rotation matrix $\mathbf{R}_b^{w\mathsf{T}}$, see **Equation 3.26**.

$$\boldsymbol{F}_{\mathcal{B}} = \begin{bmatrix} F_2 + F_4 \\ F_1 + F_3 \\ F_t - F_d \end{bmatrix} - \mathbf{R}_b^{w\mathsf{T}} \begin{bmatrix} 0 \\ 0 \\ m\,g \end{bmatrix} \tag{3.26}$$

where $\mathbf{R}_b^{w\mathsf{T}}$: Rotation matrix from world- to body-frame [−]
g : Gravity acceleration [m/s^2]

Next the body forces $\boldsymbol{F}_{\mathcal{B}}$ are inserted into Newton's second law, and the acceleration is solved for.

$$\dot{\boldsymbol{v}}_{\mathcal{B}} = \frac{1}{m}\,\boldsymbol{F}_{\mathcal{B}} \tag{3.27}$$

$$\begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \frac{1}{m}\left(\begin{bmatrix} F_2 + F_4 \\ F_1 + F_3 \\ F_t - F_d \end{bmatrix} - \mathbf{R}_b^{w\mathsf{T}} \begin{bmatrix} 0 \\ 0 \\ m\,g \end{bmatrix} \right) \tag{3.28}$$

Lastly by expanding the above equation and multiplying out the rotation matrix, the following body-frame model is found:

$$\begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}(F_2 + F_4) \\ \frac{1}{m}(F_1 + F_3) \\ \frac{1}{m}(F_t - F_d) \end{bmatrix} + \begin{bmatrix} g\sin\theta \\ -g\cos\theta\sin\phi \\ -g\cos\theta\cos\phi \end{bmatrix} \tag{3.29}$$

3.4.2 Movement in world frame

The previous body frame model can be used to stabilise the relative velocity of the UAV, however in order to stabilise the absolute position $\boldsymbol{p}$ the dynamics has to be described in the world frame. This is done by rotating the body frame velocity vector $\boldsymbol{v}_{\mathcal{B}}$ into the inertial frame using $\mathbf{R}_b^w$.

$$\dot{\boldsymbol{p}} = \mathbf{R}_b^w\,\boldsymbol{v}_{\mathcal{B}} \tag{3.30}$$

Multiplying out the rotation matrix $\mathbf{R}_b^w$ and velocity vector $\boldsymbol{v}_{\mathcal{B}}$ yields the three remaining state equations, describing the relationship between the body and inertial frame. The complete model of the translational dynamics can then be summarised:

$$\mathbf{f_T}(\boldsymbol{x},\boldsymbol{u}) = \begin{cases} \dot{x} = v_x\,\mathrm{c}\,\psi\,\mathrm{c}\,\theta - v_y(\mathrm{c}\,\phi\,\mathrm{s}\,\psi - \mathrm{c}\,\psi\,\mathrm{s}\,\phi\,\mathrm{s}\,\theta) + v_z(\mathrm{s}\,\phi\,\mathrm{s}\,\psi + \mathrm{c}\,\phi\,\mathrm{c}\,\psi\,\mathrm{s}\,\theta) \\ \dot{y} = v_x\,\mathrm{c}\,\theta\,\mathrm{s}\,\psi + v_y(\mathrm{c}\,\phi\,\mathrm{c}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\psi\,\mathrm{s}\,\theta) - v_z(\mathrm{c}\,\psi\,\mathrm{s}\,\phi - \mathrm{c}\,\phi\,\mathrm{s}\,\psi\,\mathrm{s}\,\theta) \\ \dot{z} = v_x\,\mathrm{s}\,\theta + v_y\,\mathrm{c}\,\theta\,\mathrm{s}\,\phi + v_z\,\mathrm{c}\,\phi\,\mathrm{c}\,\theta \\ \dot{v_x} = \frac{1}{m}(F_2 + F_4) + g\,\mathrm{s}\,\theta \\ \dot{v_y} = \frac{1}{m}(F_1 + F_3) - g\,\mathrm{c}\,\theta\,\mathrm{s}\,\phi \\ \dot{v_z} = \frac{1}{m}(F_t - F_d) - g\,\mathrm{c}\,\theta\,\mathrm{c}\,\phi \end{cases} \tag{3.31}$$

Seen from above system of equations, the translational dynamics are highly dependent on the rotational system. Furthermore, all the system dynamics are highly nonlinear in terms of the trigonometric functions. These non-linearities will have to be addressed in order to use the model for linear control algorithms.

3.5 Linear System Model

The goal of the system modelling, is to design a control system capable of stabilising the mono-copters rotational and translational dynamics. The models formulated in the previous sections, includes all the nonlinear kinematics present in a system with 6-DOF. These non-linear system equations can be formulated in terms of two functions:

$$\dot{\boldsymbol{x}} = \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{3.32}$$

$$\boldsymbol{y} = \mathbf{h}(\boldsymbol{x}, \boldsymbol{u}) \tag{3.33}$$

Where $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{y}$ is the state-, input- and output-vector respectively, $\mathbf{f}$ is the state equations (describing the change in system states) and $\mathbf{h}$ is the output equations (relating the system states to the outputs).

In order to use the non-linear model in the design of a stabilising controller, either a nonlinear control strategy is needed, or the system must be linearized. A non-linear control strategy allows for more aggressive control, as showed by Greiff [6], however when aggressive control outside the stable hover position is not needed, linear approaches performs similarly and are less computational heavy. Therefore, a linear control scheme is deemed sufficient, and the non-linear dynamics has to be linearized in order to design a linear stabilising controller.

3.5.1 State-space representation

In order to apply classical control theory to design a stabilising controller, a linear time-invariant (LTI) model is needed. One way to to describe a linear model, is using the state-space representation, with states $\boldsymbol{x}$, inputs $\boldsymbol{u}$ and output $\boldsymbol{y}$:

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u} \tag{3.34}$$

$$\boldsymbol{y} = \mathbf{C}\boldsymbol{x} + \mathbf{D}\boldsymbol{u} \tag{3.35}$$

Where $\mathbf{A}$ is the system matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the output matrix and $\mathbf{D}$ is known as the feedthrough matrix, see **Figure 3.9**.

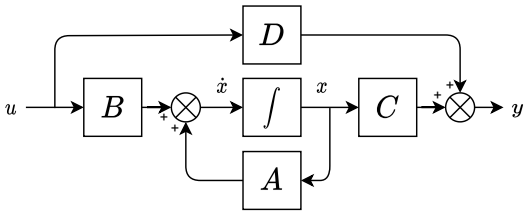


Figure 3.9: Visual representation of a linear system modal on the generalised state-space form.

To describe the entire system, the translational and rotational models are concatenated, resulting in a state vector $\boldsymbol{x}$ that consists of the twelve system states:

$$\boldsymbol{x} = \begin{bmatrix} \phi & \theta & \psi & \omega_x & \omega_y & \omega_z & x & y & z & v_x & v_y & v_z \end{bmatrix}^\mathsf{T} \tag{3.36}$$

The system model describes the dynamics in terms of the forces acting on the system These forces are not considered the direct system inputs, but rather nonlinear functions thereof. Instead, the system inputs are considered the variables of these functions, being the thrust vane angles ($\alpha_1 \dots \alpha_n$) and the motor velocity (ω_t), yielding the input vector:

$$\boldsymbol{u} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \omega_t \end{bmatrix}^\mathsf{T} \tag{3.37}$$

3.5.2 Linearisation

In order to apply classical control algorithms in the design of a stabilising controller, a linear time-invariant (LTI) model is needed. To obtain such a model, the nonlinear system dynamics given by $\dot{\boldsymbol{x}} = \mathbf{f}(\boldsymbol{x}, \boldsymbol{u})$ are linearized around an operating point $\boldsymbol{x}_0$ (with equilibrium input $\boldsymbol{u}_0$). The linearization approximately describes the system dynamics, as long as the system state and input stays close to the operating point with only small perturbations $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{u}}$, being described by:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0 \qquad \tilde{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_0 \tag{3.38}$$

Using the above definition, a linear model describing the small signal dynamics can be written on state-space as:

$$\dot{\tilde{\boldsymbol{x}}} = \mathbf{A}\tilde{\boldsymbol{x}} + \mathbf{B}\tilde{\boldsymbol{u}} \tag{3.39}$$

The small signal system matrix $\mathbf{A}$ and input matrix $\mathbf{B}$ are found by calculating the Jacobian of the nonlinear system $\mathbf{f}(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \mathbf{f_R} & \mathbf{f_T} \end{bmatrix}^\mathsf{T}$, with respect to the states and inputs, and evaluating the partial derivatives at the equilibrium point.

$$\mathbf{A} = \left.\frac{\partial \mathbf{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \qquad \mathbf{B} = \left.\frac{\partial \mathbf{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \tag{3.40}$$

The operating point for the linearization is chosen to be the only stable equilibrium point for the drone; the hover point. In this stable state, the drone is perfectly horizontal $\theta = 0, \phi = 0$ and the motors deliver a total thrust that equals the gravitational pull.

3.6 Summary

Using the nonlinear dynamics, a linear small signal model is found by linearizing around the hover point, using the Jacobian linearzation. By evaluating the partial derivatives at the equilibrium point, the following linear system dynamics are found:

$$\begin{cases} \dot{\phi} = \omega_x & \dot{x} = v_x \\ \dot{\theta} = \omega_y & \dot{y} = v_y \\ \dot{\psi} = \omega_x & \dot{z} = v_z \\ \dot{\omega}_x = \dfrac{l\, C_L C_F}{J_{xx}}(\alpha_1 + \alpha_3) & \dot{v_x} = \dfrac{C_L C_F}{m}(\alpha_2 + \alpha_4) - g\theta \\ \dot{\omega}_y = \dfrac{l\, C_L C_F}{J_{yy}}(\alpha_2 + \alpha_4) & \dot{v_y} = \dfrac{C_L C_F}{m}(\alpha_1 + \alpha_3) + g\phi \\ \dot{\omega}_z = \dfrac{r\, C_L C_F}{J_{zz}}(\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4) & \dot{v_z} = \dfrac{2K_f\omega_0(1 - C_D)}{m}\omega_t \end{cases} \tag{3.41}$$

The linear system equations can then be written on state-space form, with $\mathbf{A}$ and $\mathbf{B}$ being composed of the linearized system dynamics.

$$\mathbf{A} = \begin{bmatrix} & \mathbf{0}_3 & & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ & \mathbf{0}_3 & & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ & \mathbf{0}_3 & & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \\ \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & & & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \in \mathbb{R}^{12\times 12} \tag{3.42}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 1} \\ \begin{bmatrix} \frac{C_L C_F l}{J_x} & 0 \\ 0 & -\frac{C_L C_F l}{J_y} \\ \frac{C_L C_F r}{J_z} & -\frac{C_L C_F r}{J_z} \end{bmatrix} & \begin{bmatrix} \frac{C_L C_F l}{J_x} & 0 \\ 0 & -\frac{C_L C_F l}{J_y} \\ -\frac{C_L C_F r}{J_z} & \frac{C_L C_F r}{J_z} \end{bmatrix} & \mathbf{0}_{3\times 1} \\ \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 1} \\ \begin{bmatrix} 0 & \frac{C_L C_F}{m} \\ \frac{C_L C_F}{m} & 0 \end{bmatrix} & \begin{bmatrix} 0 & \frac{C_L C_F}{m} \\ \frac{C_L C_F}{m} & 0 \end{bmatrix} & \mathbf{0}_{2\times 1} \\ \mathbf{0}_{1\times 2} & \mathbf{0}_{1\times 2} & \frac{2K_f\omega_0(1-C_D)}{m} \end{bmatrix} \in \mathbb{R}^{12\times 5} \tag{3.43}$$

The output matrix $\mathbf{C}$ is an identity matrix $\mathbf{I}_{12}$, however with zeros in the places matching an unmeasured system state. The next chapter will introduce the concept of full-state feedback and describe the control law that will be utilised to stabilise the system dynamics based on the linear system dynamics described by $\mathbf{A}$ and $\mathbf{B}$.

CHAPTER 4

Control

Many methods can be applied to stabilise UAVs in general, however, the most common strategy is that of the proportional–integral–derivative (PID) control. The PID approach, however, is based on classical control theory, and does only apply to single-input single-output (SISO) systems. This makes the PID controller difficult and time-consuming to apply on a 6-DOF system, as each control loop has to be designed separately.

Another established control strategy is that of full-state feedback, that relies on either pole-placement or the theory of linear quadratic regulator (LQR) to compute a controller that stabilises the entire system. Full-state feedback enables the control of multiple-input multiple-output (MIMO) systems, which greatly simplifies the controller design of an UAV. The method presented by Foehn and Scaramuzza [13], uses LQR and the partial differentials of a nonlinear model, to build a linear model and to compute the controller gains at each time-step. This approach is very computational expensive, which does not fit well on a digital platform that has to run in real-time. Instead, the regular LQR method, as proposed by Greiff [6], is considered sufficient, where the controller is computed offline.

In the this chapter, the linear state-space model derived in **Chapter 4** will be used to design a model-based control strategy, with the goal of stabilising both attitude and position. First, the principle of full-state feedback is introduced, and the LQR formulation is described. Next, the full-state vector is separated into two subsets and the proposed control strategy is described. Lastly, the controller gains are computed, and steps are applied to test the controller response on the nonlinear simulation.

4.1 Full-state feedback

The stability of a linear system depends entirely on the location of the eigenvalues in the complex plane [26]. The open-loop dynamics described by the system matrix **A** from the state-space representation, see **Equation 4.1**, has all-zero eigenvalues (poles at origin). This means that the linearized open-loop system has marginal stability.

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u} \tag{4.1}$$

The system therefore cannot achieve stability on its own, as any slight disturbance will excite the system towards instability. To achieve stability the systems open-loop poles must be moved to the left half of the complex plane. One way to achieve this is by using full-state feedback, with the controller gain **K** [26], see **Figure 4.1**.

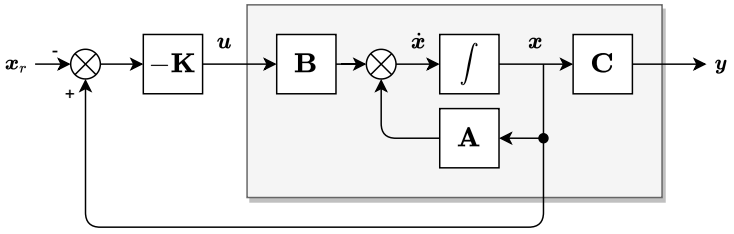


Figure 4.1: Full-state error feedback with controller **K** and reference $\boldsymbol{r}$.

Assuming all system states are measurable and known at all times, the full-state feedback law is described by:

$$\boldsymbol{u} = -\mathbf{K}(\boldsymbol{x} - \boldsymbol{x}_r) \tag{4.2}$$

Where $\boldsymbol{x}_r$ is a vector of desired state setpoints, which serves as a external input to the closed-loop system [27]. Using the above control-law, the open-loop system is closed, and the feedback-term controlled by the gain matrix **K** can be used to place the system poles at the desired location, yielding the closed loop dynamics:

$$\dot{\boldsymbol{x}} = (\mathbf{A} - \mathbf{BK})\boldsymbol{x} + \mathbf{BK}\boldsymbol{x}_r \tag{4.3}$$

The poles of the closed-loop system are then given by the characteristic equation of the matrix $(\mathbf{A} - \mathbf{BK})$ while the matrix **BK** acts as input matrix to the closed-loop system.

Full-state feedback are a powerful tool, that enables absolute control of the closed-loop system poles [26], however it comes with an obvious problem; the choice of the gain matrix **K**. To determine **K**, two methods are often utilised; pole placement and the linear quadratic regulator (LQR).

The pole placement method computes $\mathbf{K}$ such that the close-loop poles are placed at specific locations. This requires knowledge about the system, and how the system characteristics (overshoot, rise-time etc.) should behave. LQR on the other hand computes $\mathbf{K}$ such that the closed-loop poles are placed optimal, given a quadratic cost function that punishes state errors and the actuation effort. Tuning of $\mathbf{K}$ using LQR are therefore much more intuitive, and enables faster development of a stabilising controller for larger and complex systems [26].

4.1.1 Controllability

In order to design the stabilising feedback gain $\mathbf{K}$ using any of the above mentioned methods, the system it self must be controllable. Only the poles of a fully controllable system can be moved to any arbitrary place within the complex plane using full-state feedback. A system is said to be controllable if a sequence of control inputs $\boldsymbol{u}$ can transfer any initial state to the origin in a finite amount of time [26].

For linear time-invariant systems, a system is controllable if its controllability matrix, $\mathcal{C}$, has a full row rank of n, where n is the dimension of the systems matrix $\mathbf{A}$ [26]:

$$\text{rank}(\mathcal{C}) = n \tag{4.4}$$

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \dots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix}^{\mathsf{T}} \tag{4.5}$$

The controllabillity matrix is computed for the linearized system dynamics, and the rank is found to be $n = 12$. This implies that the 12 system states given by $\boldsymbol{x}$, are fully controllable from the 5 control inputs $\boldsymbol{u}$.

4.1.2 Linear Quadratic Regulator

The linear quadratic regulator is a full-state feedback controller, that is computed by minimising a quadratic cost function J that punishes state errors and actuation effort. The continuous-time variant of the cost function is defined as [13]:

$$J(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) = \int_0^\infty (\tilde{\boldsymbol{x}}^{\mathsf{T}}\mathbf{Q}\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^{\mathsf{T}}\mathbf{R}\tilde{\boldsymbol{u}})\mathrm{d}t \tag{4.6}$$

Where $\mathbf{Q}$ and $\mathbf{R}$ are positive definite matrices, that punishes the state errors and actuation effort respectively.

The intent of the cost function is to bring the state error close to zero using the term $\tilde{\boldsymbol{x}}^{\mathsf{T}}\mathbf{Q}\tilde{\boldsymbol{x}}$. This might require very large control inputs, which the additional term $\tilde{\boldsymbol{u}}^{\mathsf{T}}\mathbf{R}\tilde{\boldsymbol{u}}$ penalises, such that a more realistic design is found [28].

Bryson's Rule

The choice of the weighting matrices **Q** and **R** are however not trivial. Practically the LQR formulation, translates classical control problems where specifications are given in terms of settling time, overshoot etc. into the choice of the coefficients of the cost function. To aid in the choice of the coefficients, Bryson's rule is applied, describing the coefficients in terms of maximum allowed deviation of the states and input [28] by choosing the matrices as diagonal matrices:

$$\mathbf{Q} = \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_n \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} R_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R_n \end{bmatrix} \tag{4.7}$$

Where the diagonal entities Q_n and R_n are defined as the reciprocal of the maximum allowable value of state and inputs squared.

$$Q_n = \frac{1}{\max(x_i)^2} \qquad R_n = \frac{1}{\max(u_i)^2} \tag{4.8}$$

Even though Bryson's rule usually gives good initial results, it is merely the starting point for a trial and error iterative design process, aimed at achieving desirable properties for the closed-loop system.

4.1.3 Integral action

An inherent disadvantage of the standard LQR-controller is that, in essence, it is a proportional state feedback controller, that may result in steady-state errors. To remove steady-state error integral action can be introduced into the feedback loop [6], see **Figure 4.2**.

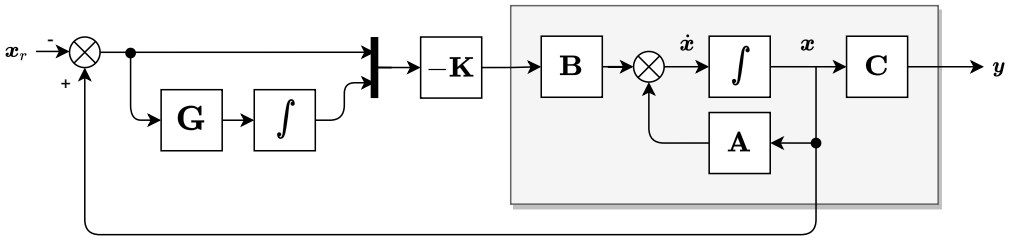


Figure 4.2: Full-state feedback with integral action on a selection of the state errors.

The **G** matrix has the same columns as the size of the state vector, and is used to select the states errors that need integral action. The same matrix is appended to the system when computing the feedback gain, augmenting the system with additional states.

4.2 Control strategy

Before designing the actual controller using LQR and Bryson's rule, first the control strategy will be explained. In order to ease the design and implementation, the control system is separated into two subsystems, each consisting of a subset of the total system dynamics, this being:

$$\boldsymbol{x}_{\text{hov}} = \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{\omega} \\ z \\ v_z \end{bmatrix} \in \mathbb{R}^{8\times 1} \qquad \boldsymbol{x}_{\text{pos}} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \in \mathbb{R}^{4\times 1} \tag{4.9}$$

The first subsystem consists of all the rotational dynamics, including the altitudinal states (z-axis). The controller for this subsystem is named the "Hover controller", as the objective of this controller is to stabilise the drone in the hover point. The second subsystem consists of the translational states in the horizontal plane, and is named "Position controller", having the goal of stabilising the horizontal position by supplying roll and pitch set-points to the hover controller, see **Figure 4.3**.

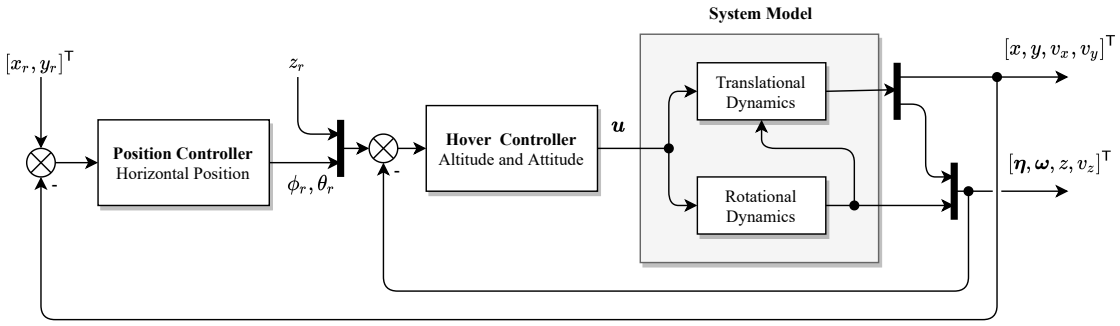


Figure 4.3: Block diagram of the proposed control strategy.

The division of the state-vector separates the faster dynamics ($\boldsymbol{x}_{\text{hov}}$) from the slower dynamics ($\boldsymbol{x}_{\text{pos}}$), which allows for the design of an independent hover controller. With an independent hover controller, different position control strategies can be tried out, while maintaining the same level of hovering stability.

4.3 Hover Controller

The hover controller acts on the truncated system dynamics described by the state vector $\boldsymbol{x}_{\text{hov}} = \begin{bmatrix} \phi & \theta & \psi & \omega_x & \omega_y & \omega_z & z & v_z \end{bmatrix}^{\mathsf{T}}$, and from this generates the control signals $\boldsymbol{u} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha 4 & \omega_t \end{bmatrix}^{\mathsf{T}}$, that stabilises the attitude and altitude, see **Figure 4.4**.

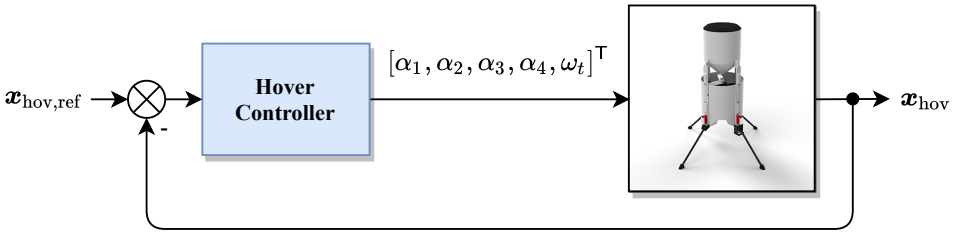


Figure 4.4: Block diagram of the hover controller. The vector $\boldsymbol{x}_{\text{hov,ref}}$, defines the desired state reference, such as ϕ_r, θ_r, ψ_r and z_r.

4.3.1 Control considerations

As the system dynamics are linearized around the hover point, the system inputs (control signals) has reached steady-state. At steady-state, the only input not being zero is the rotational velocity of the motors ω_t. In order to produce the needed thrust, the steady-state motor velocity ω_0 must be added to the controller output. This, however, would cause a very sudden step on the z-state, causing uncontrollable behaviour. Another way to reach ω_o is by augmenting the system with integral action on the z-state, such that the control output slowing raises towards the steady-state value.

Integral action could similarly be implemented on the attitudinal states, to reduce steady-state errors caused by misalignments and unbalanced weight. However, this could potentially reduce performance, as the integral action would slow down the controller response. No integral action is therefore applied to the attitude, and any steady-state error present on the real system must therefore be handled by the position controller. The augmented hover controller with integral action on the z-state, thus becomes:

$$\mathbf{A}_{\text{hint}} = \begin{bmatrix} \mathbf{A}_{\text{hov}} & \mathbf{0}_{9\times 1} \\ \mathbf{G}_{\text{hov}} & \mathbf{0}_{1\times 1} \end{bmatrix} \in \mathbb{R}^{9\times 9} \quad \mathbf{B}_{\text{hint}} = \begin{bmatrix} \mathbf{B}_{\text{hov}} \\ \mathbf{0}_{1\times 5} \end{bmatrix} \in \mathbb{R}^{9\times 5} \tag{4.10}$$

With $\mathbf{A}_{\text{hov}}$ and $\mathbf{B}_{\text{hov}}$ being the truncated system and input matrices. The integral action matrix $\mathbf{G}_{\text{hov}}$ is defined as,

$$\mathbf{G}_{\text{hov}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{1\times 8} \tag{4.11}$$

Using the augmented system, the control gain $\mathbf{K}_{\text{hov}}$ is designed using MATLABs `lqr()` command, with the $\mathbf{Q}_h$ and $\mathbf{R}_h$ initially being defined using Bryson's rule. After the initial run, $\mathbf{R}_h$ are kept constant, and $\mathbf{Q}_h$ are tuned to achieve desirable response. The maximum allowable actuation are chosen to be within reasonable limits of the operating point; $\max(\alpha_t) = 10^\circ$ and $\max(\omega_t) = 1000\,\text{rpm}$, yielding the $\mathbf{R}_h$ coefficients:

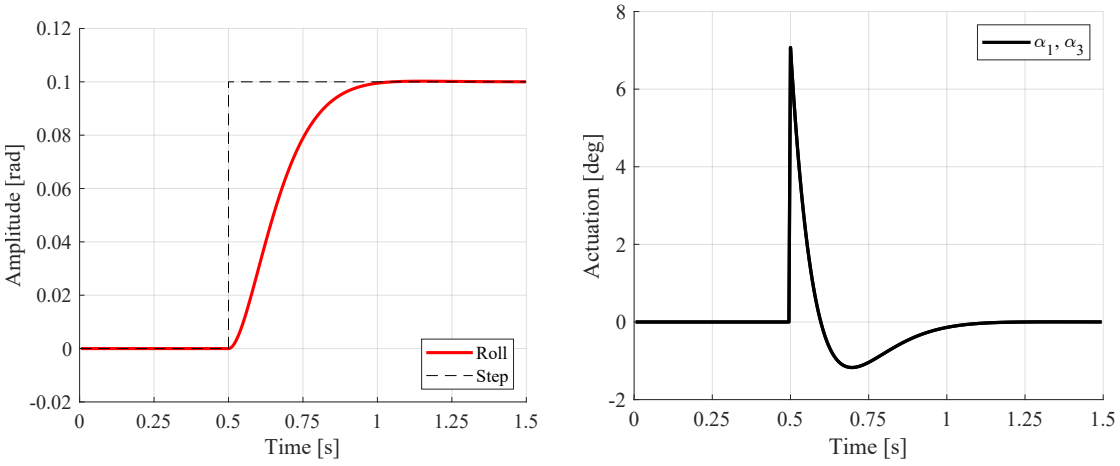
$$R_{h1} \dots R_{h4} = \frac{1}{(10^\circ)^2} \qquad R_{h5} = \frac{1}{(1000\,\text{rpm})^2} \tag{4.12}$$

4.3.2 Roll and pitch control

For the purpose of stabilising the mono-copter, fast and robust control of roll and pitch is required. Any small error in roll or pitch propagates into the translational states, which affects the overall performance. Therefore, the penalty of errors in roll and pitch must be high, such that state-errors quickly are removed. The state-error penalty described by the diagonal coefficients of $\mathbf{Q}_h$ are found iterative, to be a maximum allowable deviation of 0,1 rad for roll and pitch, and 1 rad/s for the angular rates.

$$Q_{h1} = Q_{h2} = \frac{1}{(0{,}1\,\text{rad})^2} \qquad Q_{h4} = Q_{h5} = \frac{1}{(1\,\text{rad/s})^2} \tag{4.13}$$

The resulting controller achieves a very quick response with almost zero overshoot, and with no steady-state error, see **Figure 4.5**. Furthermore, as seen from **Figure 4.5b** the actuation effort is kept below 10° as desired.



(a) Roll with a step of 0,1 rad.

(b) Actuation from a step of 0,1 rad on roll.

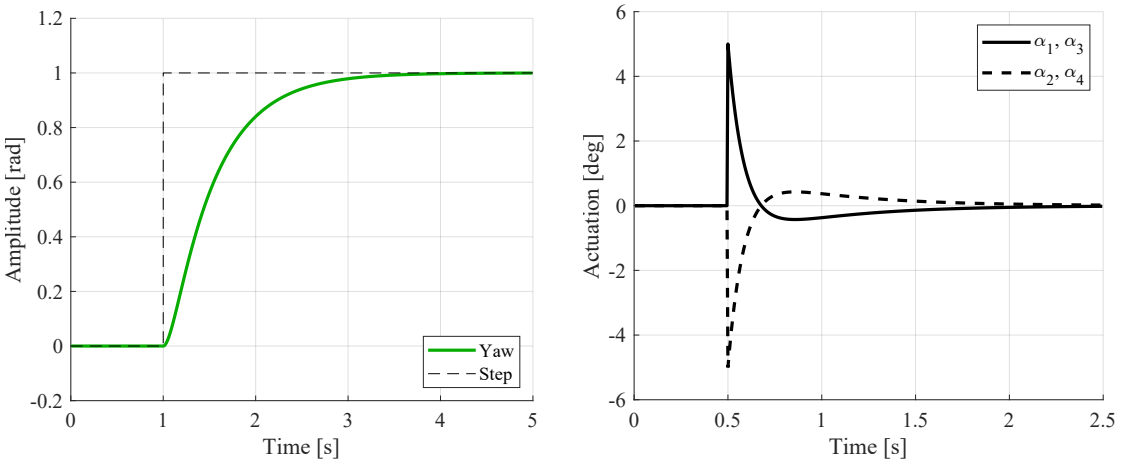
Figure 4.5: Step-response and actuation effort from a step on roll. Similar results are seen with pitch, as the mono-copter is symmetric.

4.3.3 Yaw control

The rotational motion around the z-axis (yaw) differs from the roll and pitch, both in terms of inertia but also because no specific value of yaw causes instability. Although all values of yaw are stable, a controller is still needed to counteract the small net momentum caused by differences in motor velocity and inertia. In the absence of yaw control, the mono-copter would keep rotating in the air, which eventually could lead to instability. Considering that fast control of yaw is not critical, the yaw control can be tuned to perform considerable slower than roll and pitch, to save actuation effort. This is important, as the thrust vanes has to actuate significantly more, in order to produce torque around the z-axis (τ_z), than with roll and pitch (difference in moment arm). It is therefore important to penalise the yaw state less, as to not saturate the thrust vanes and in turn losing control of roll and pitch. The maximum allowable deviation of yaw is chosen to be 1 rad and the yaw rate to be 2 rad/s.

$$Q_{h3} = \frac{1}{(1\,\text{rad})^2} \qquad Q_{h6} = \frac{1}{(2\,\text{rad/s})^2} \tag{4.14}$$

This yields a considerable slower step-response than roll and pitch, see **Figure 4.6a**, yet still uses quite a lot of actuation effort, see **Figure 4.6b**.



(a) Step-response of yaw with a step of 1 rad.

(b) Actuation signals from a step of 1 rad on yaw

Figure 4.6: Step-response of the yaw and the resulting actuation, using the nonlinear system model.

It is important to notice, that the attitude is measured in polar coordinates, and switches sign at $\pm 180°$. This does not work well with the cartesian definition of state errors $e = x_r - x$, causing the yaw controller to take the wrong route in certain intervals. To account for this, 2π is subtracted from the state-error, if the yaw error is larger than π, which forces the control to always rotate in the correct direction.

4.3.4 Altitude control

The altitude is controlled entirely by the motor thrust, which in turn is controlled by the rotational velocity of the motors ω_t. The hover controller computes the motor velocity ω_t using feedback from both the altitude z and the velocity v_z, and from the augmented integral state (summation of errors in z). The integral term is used to slowly raise the motor velocity to the steady-state value ω_0. This has the benefit of also removing any steady-state error, which otherwise would be present due to gravity. However, because of the integral action, the z motion is also inclined to overshoot, as the actuation output mostly consist of the slower integral state. The overshoot can be reduced by punishing the z state more, however a preferable solution is to implement anti-windup on the integral. This is done by setting an upper limit on the integral, that limits how much the integral state can add to the motor velocity.

Now, using Bryson's rule, the maximum allowable deviation of z is chosen to 0,5 m and v_z to 1 m/s. The integral state is tuned to achieve a lift-of delay of approximately 2 s.

$$Q_{h7} = \frac{1}{(0{,}25\,\mathrm{m})^2} \qquad Q_{h8} = \frac{1}{(1\,\mathrm{m/s})^2} \qquad Q_{h9} = \frac{1}{(0.15)^2} \tag{4.15}$$

As seen from **Figure 4.7**, this yields a rather slow rise-time with a slight overshoot. Furthermore no change in altitude is observed before 2 s after the step of 1 m is applied. This behaviour is the result of the slowly growing integral, and not a result of slow system dynamics. This is an intended feature, which only is observed at initial take-off and not at any subsequent steps performed afterwards.

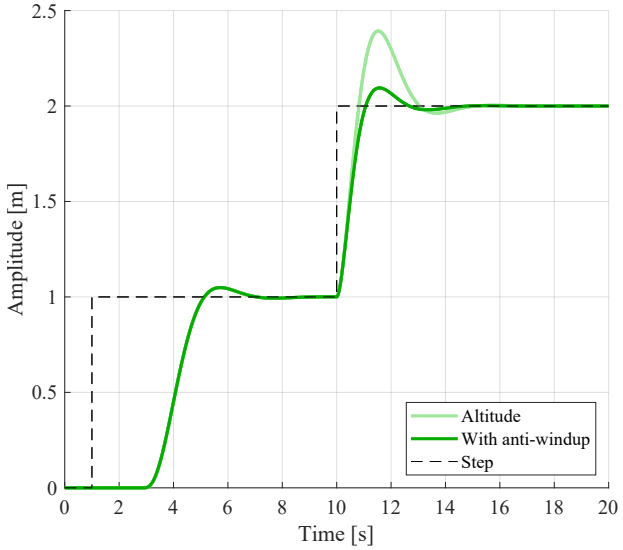


Figure 4.7: Simulated step-response of the altitude with and without anti-windup. The anti-windup reduces overshoot by stopping the integral from growing to large.

4.4 Position Controller

With the hover controller being able to stabilise the attitude and altitude, the position controller can be designed. The position controller acts on the translational states in the horizontal plane, $\boldsymbol{x}_{\text{pos}} = [\, x \;\; y \;\; v_x \;\; v_y \,]$, and outputs roll and pitch references $(\phi_r,\, \theta_r)$ to the hover controller, see **Figure 4.8**.

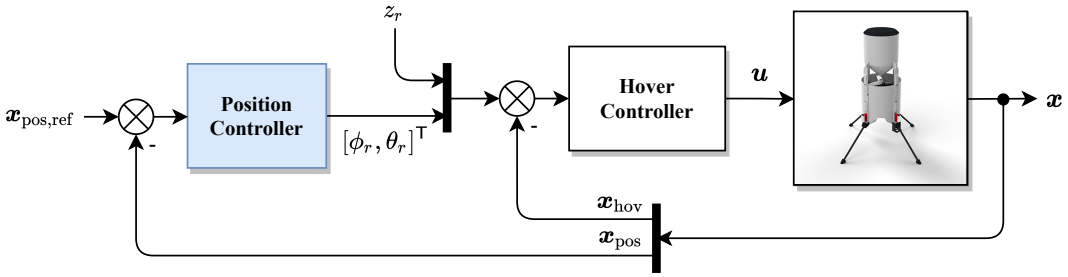


Figure 4.8: Block diagram of the postion and hover controller. The vector $\boldsymbol{x}_{\text{pos,ref}}$, defines the desired state reference, being the target position x_r and y_r

4.4.1 Control considerations

The first and foremost goal of the position controller is to stabilise the positional drift caused by steady-state errors in the hover controller caused by imperfections in the physical system. To account for this, the position controller is augmented with an integral state on both positional states x and y. The augmented system with integral action becomes:

$$\mathbf{A}_{\text{pint}} = \begin{bmatrix} \mathbf{A}_{\text{pos}} & \mathbf{0}_{4\times 2} \\ \mathbf{G}_{\text{pos}} & \mathbf{0}_{2\times 2} \end{bmatrix} \in \mathbb{R}^{6\times 6} \quad \mathbf{B}_{\text{pint}} = \begin{bmatrix} \mathbf{B}_{\text{pos}} \\ \mathbf{0}_{2\times 2} \end{bmatrix} \in \mathbb{R}^{9\times 5} \tag{4.16}$$

With the integral action matrix $\mathbf{G}_{\text{pos}}$ being defined as,

$$\mathbf{G}_{\text{pos}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{1\times 8} \tag{4.17}$$

Using the augmented system, the control gain $\mathbf{K}_{\text{pos}}$ is computed using the same approach as for the hover controller, with $\mathbf{Q}_p$ and $\mathbf{R}_p$ being defined using Bryson's rule. The maximum allowable actuation is chosen to $0{,}1\,\text{rad} \approx 5^\circ$ for both ϕ_r and θ_r.

$$R_{p1} = R_{p2} = \frac{1}{(0{,}1\,\text{rad})^2} \tag{4.18}$$

4.4.2 Position control

The only way to control the position, is by pivoting the mono-copter such that a component of the thrust vector lies in the horizontal plane. This is achieved by supplying roll and pitch references to the hover controller, which in turn actuates the thrust vanes. However, because the position is measured in the inertial frame, and the roll and pitch actuation is done in the body frame, the values are not directly compatible.

The error in position must therefore be rotated to the body frame, in order to generate appropriate control signals for the hover controller. Without rotating the error any non-zero yaw angle will make the control system unable to track the global position precisely. To rotate the positional error, the rotation matrix $\mathbf{R}_b^{w\mathsf{T}}$ are applied with $\phi = 0$ and $\theta = 0$. This is possible because the absolute position is measured horizontally in two dimensions, and thus the error is not affected by roll and pitch.

$$\mathbf{R}_b^{w\mathsf{T}} \approx \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_{2\mathrm{D}} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \tag{4.19}$$

The error in position and velocity can then be rotated to the body frame as:

$$\boldsymbol{e}_p = \left(\begin{bmatrix} x_r \\ y_r \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right) \mathbf{R}_{2\mathrm{D}} \qquad \boldsymbol{e}_v = \left(\begin{bmatrix} v_{x_r} \\ v_{y_r} \end{bmatrix} - \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right) \mathbf{R}_{2\mathrm{D}} \tag{4.20}$$

Yielding the state-error vector: $\boldsymbol{e}_{\mathrm{pos}} = [\, \boldsymbol{e}_p \; \boldsymbol{e}_v \,]^{\mathsf{T}}$

Now, using Bryson's rule the state-penalty coefficients are found. The maximum allowable deviation of x and y is chosen to 0,5 m while the value of v_x and v_y is chosen to 1 m/s. The penalty of the integral states are chosen such that the integral part quickly reduces, which unfortunately increases the overshoot.

$$Q_{p1} = Q_{p2} = \frac{1}{(0{,}5\,\mathrm{m})^2} \qquad Q_{p3} = Q_{p4} = \frac{1}{(1\,\mathrm{m/s})^2} \qquad Q_{p5} = Q_{p6} = \frac{1}{(1)^2} \tag{4.21}$$

As seen from **Figure 4.9**, this results in a rather fast step-response, with a large overshoot. The overshoot is largely caused by the integral, which grows without constraints when a step is applied. To suppress this behaviour, the principle of anti-windup is again utilised by setting a limit on the integral. The integral limit is found experimentally, however, it is important to note that this limit might need adjusting to correct for larger steady-errors, found on the physical system.

To evaluate the integral action, an offset of 1° is introduced on all thrust vanes in the simulation. This offset produces a steady-state error is the hover controller, which would have propagated into an error of 0,1 m in the position, see **Figure 4.9**. Similar steady-state errors in the actuation or attitude are likely to be present on the physical system, which strongly argues for the implemented integral action.

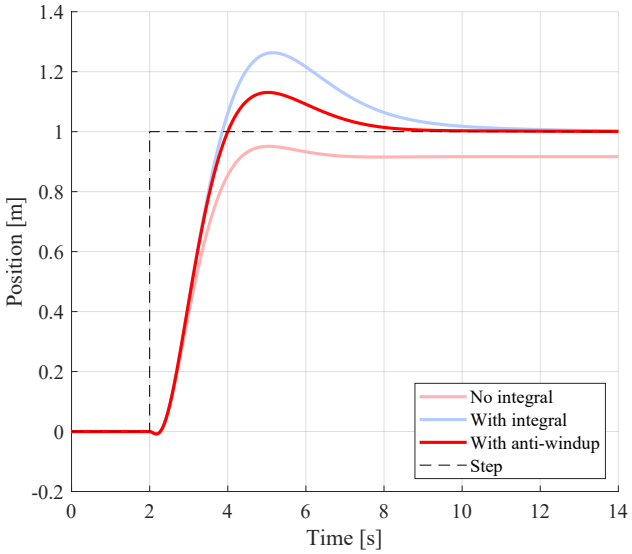


Figure 4.9: Three simulated step-responses with a step of 1 m on the x-position.

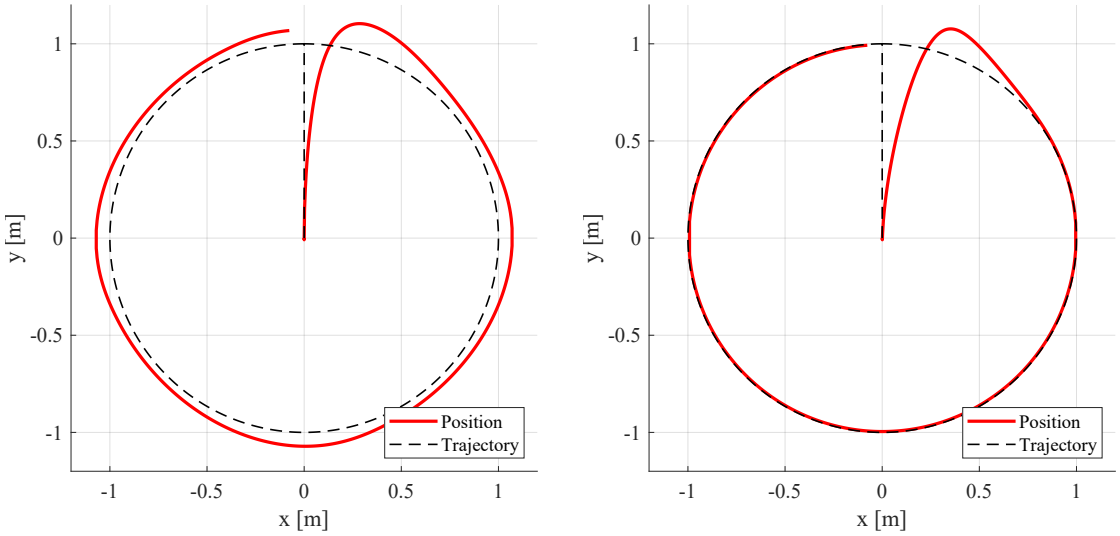
As might be observed from **Figure 4.9**, the position has a small dip in the opposite direction (non-minimum phase) before moving towards the step. This is the result of the thrust vanes producing a opposite force of the desired translational direction, in order to tilt (roll/pitch) the system. The small dip can be seen as a time-delay, and in a sense, sets a fundamental limit on how fast the system can be actuated.

4.4.3 Time-delay margin

The mono-copter is a fast moving system, with relative quick dynamics. Positional measurements are however prone to be slow, causing a transport delay in the feedback loop. This could potentially cause instability, because the controller reacts on old measurements. To estimate how much transport delay the positon controller can handle, a time delay is introduced in the feedback-loop (simulation). The delay is adjusted until the closed-loop system is on the verge of instability, defining the *time delay margin* of the controller. Using this approach, a maximum allowable time-delay of 0,4 s is found. To increase the allowable time-delay, the controller gains can be reduced, however, the achieved *time delay margin* is considered sufficient.

4.4.4 Trajectory tracking

The problem of tracking a trajectory differs slightly from the regular problem of doing a step along a single axis. This is especially noticeable when tracking a nonlinear continuous trajectory, such as a circular path. Nonetheless, the position controller are capable of tracking a nonlinear path to some degree, as shown in **Figure 4.10a**. Here the position is controlled entirely by linear x and y feedback, causing the system to never reach the trajectory due to a steady-state error. This steady-state error is not present in the linear case, and is probably the result of the integral states being a summation of errors with different origin. To resolve this, the front of the mono-copter should point in the direction of movement (heading) such that the body-frame is aligned with the path, and the accumulated errors translates directly to the body-frame, see **Figure 4.10b**.



(a) Constant yaw

(b) Yaw equal to heading

Figure 4.10: Simulated trajectory tracking, with a circular path. The yaw direction has a large effect when tracking nonlinear trajectories.

In order to align the system with the heading, the yaw reference of the hover-controller is used. To control the yaw, a nonlinear control block is introduced that computes the heading from the desired trajectory and feeds this to the yaw controller.

$$\psi_r = \text{atan2}\left(\frac{y_n - y}{x_n - x}\right) \tag{4.22}$$

Where x_n and y_n are the next set-point values for the controller. A better method for following nonlinear paths (with multiple waypoints and sharp corners) would be pure-pursuit or carrot-chasing [29], however that is out of the scope for this project.

With the nonlinear heading control added, the linear control scheme shows great promise, as it is capable of following a nonlinear trajectory in three dimensions, see **Figure 4.11**, while eliminating steady-state errors in the hover controller introduced by actuator biases or unbalanced weight.

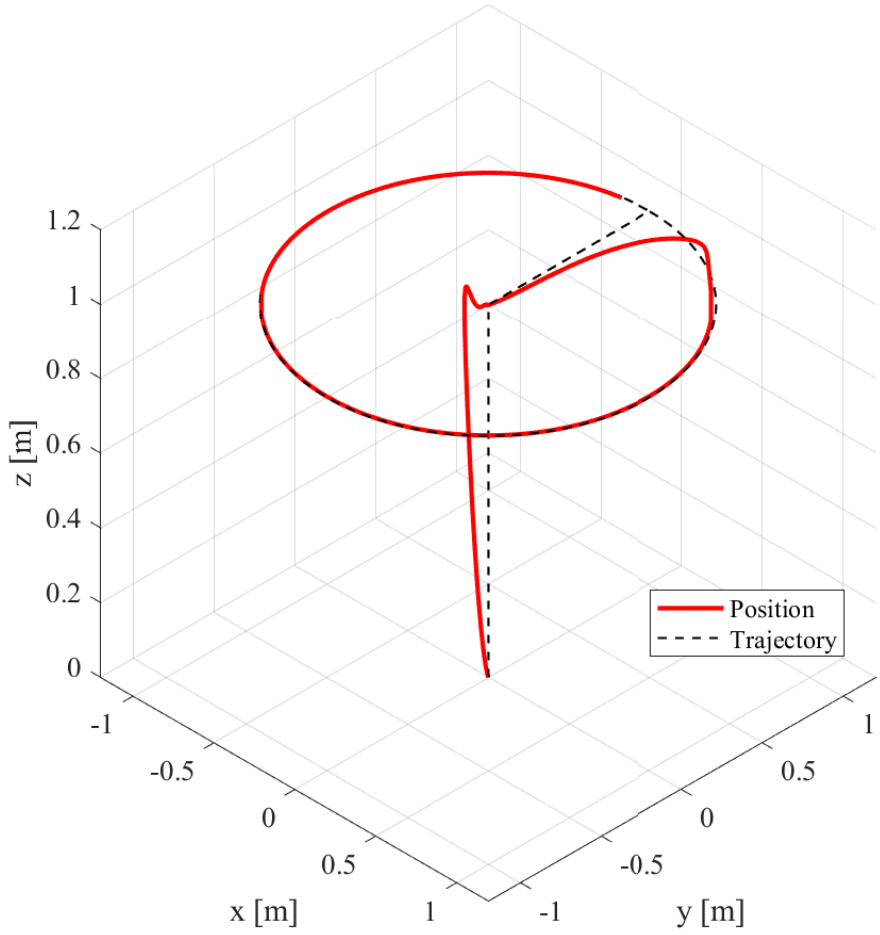


Figure 4.11: Simulated step-response of the 3D-position following a circular trajectory. Actuators are biased by 1° of offset, and integral action is active.

4.5 Summary

Simulation shows that the proposed linear control strategy are able to stabilise the entire nonlinear system dynamics. Furthermore, by addition of a small nonlinear control block, nonlinear trajectories can be followed with zero steady-state error.

The proposed full-state feedback controller is designing using a linear continuous-time model, and are tested as such in the simulation. However, the controller is eventually going to be implemented on a digital system, and thus the effects of sample-time, bit-resolution and controller loop frequency should be accounted for. If the discrete system differs to much from the continuous, the controller will not perform as expected on the real physical system. The effects of discretization does however become very small if the loop frequency gets very high (low sample-time), and the difference between the continuous- and discrete-time controller gains converges towards zero. This is assumed to be the case, as long as the control loop is run above 200 Hz.

During the design of the full-state controller, it was assumed that all system states are available at all times. This however, is not the case, as multiple states of the system are measured at different sample rates while other states are not measured at all. To account for the missing states used by the full-state feedback, a method of estimation must be applied. To perform the state estimation, the well-established concept of the Kalman filter is applied, which will be described in the following chapter.

CHAPTER 5

State Estimation

To implement the full-state feedback controller described in the previous chapter, measurements of the entire state vector $\boldsymbol{x}$ are needed. However, as previously described, some of the system states are not measured, while most are sampled at different frequencies. To remedy this issue, the use of an estimator is proposed; both to estimate the unmeasured system states, but also to interpolate in-between samples. The goal of the estimator is to produce estimates of the unmeasured system states, using indirect and noisy measurements and a system model. To deal with the noisy measurements, the Kalman filter is applied, which combines knowledge about the process and measurement noise, to produce an optimal state estimate. This chapter describes the basic concept of Kalman filtering, and how this is applied to estimate and interpolate the needed state vector.

5.1 State measurements

The entire state vector $\boldsymbol{x}$ must be known in order to implement the full-state feedback controller. Most of the states are directly measurable by the available sensors, while some must be found by estimation. As an example, the IMU supplies attitude measurements $\boldsymbol{\eta}$ and drift free gyro measurements $\boldsymbol{\omega}$, at very high sample-rates. The translational states on the other hand, are measured by sensors with a low sample-rate and one of the states (v_z) is unmeasured. This poses an issue in terms of the feedback control, as the controller cannot operate faster than the rate of the measurements and because all states are needed to close the loop. An overview of the system states availability and their sample-rates are given in **Table 5.1**.

	IMU						VICON		TOF	FLOW		
State	ϕ	θ	ψ	ω_x	ω_y	ω_z	x	y	z	v_x	v_y	v_z
Measured	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Rate [Hz]	400	400	400	400	400	400	5	5	30	50	50	-

Table 5.1: Overview of the system states and their sample-rates

From **Table 5.1**, it is clear that all the translational states $\boldsymbol{x}_{\text{pos}}$ require interpolation if the entire control loop is to be run at the same frequency. In order to interpolate and estimate the unmeasured system states, an estimator is introduced in the feedback-loop. The goal of the estimator is to produce an estimate of the translational state vector $\boldsymbol{x}_{pos}$, by combining available measurements with a system model, see **Figure 5.1**.

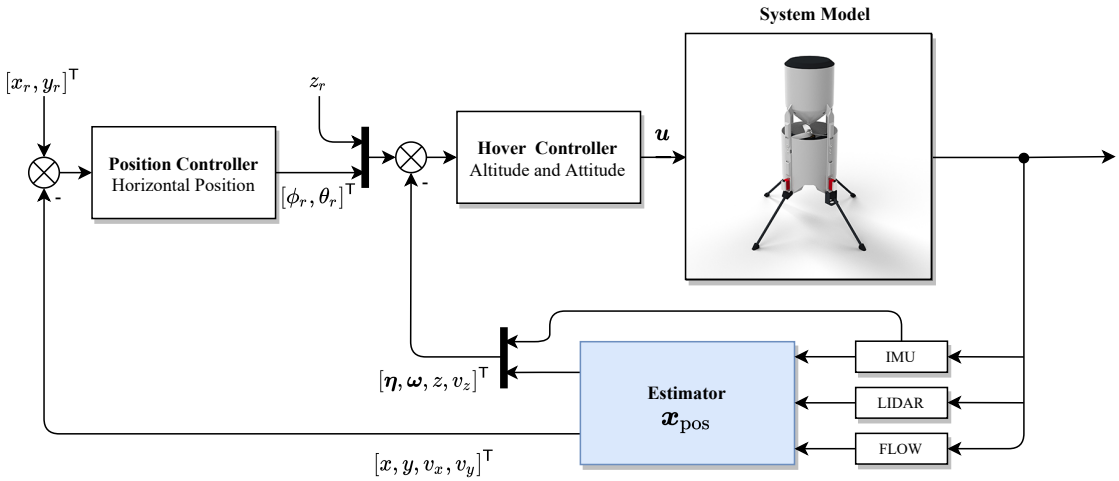


Figure 5.1: Block diagram of the control structure, with the estimator in the feedback.

5.2 Estimator Model

In order to predict the evolution of the system states, the state estimator requires a linear time-invariant model described on state-space form. The obvious choice would be the dynamic system model derived for the controller design. However, because of the linearization, the kinematic relationship between the body- and world-frame are lost, and the model would yield inferior tracking performance. A solution to this would be implementing a nonlinear estimator (such as an extended Kalman filter) that linearizes the nonlinear dynamics at each sample. However, due to the computational limitations of the digital platform, this is not considered feasible at the desired update frequency.

Instead, a slightly different model is used, one described entirely in terms of the equations of motion. Any object moving in a frictionless environment can be described using the kinematic equations for a moving body. The velocity v of this object is defined as the derivative of the position p with respect to time t. In like manner, the acceleration a is defined as the derivative of v with respect to time t. This results in the following differential equations:

$$v = \dot{p} = \frac{\mathrm{d}p}{\mathrm{d}t} \qquad a = \dot{v} = \frac{\mathrm{d}v}{\mathrm{d}t} \tag{5.1}$$

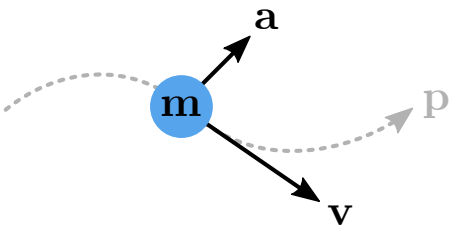


Figure 5.2: Illustration of the kinematic model described by the equations of motion.

The two differential equations described by **Equation 5.1** can be written on state-space form. The position p and velocity v is chosen as the system states, while the acceleration a is considered the system input. Any acceleration of the object must result from an external force acting on the body, and per definition, this is to be considered an external input. Formulated on state-space, this looks like:

$$\begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a \tag{5.2}$$

Using the motion model, the translational states can be described entirely in terms of position, velocity and acceleration measured in the world-frame. The general idea is to combine slow optical flow measurements, with fast acceleration and occasional positional measurements, to shape a solid estimate for the translational states.

5.2.1 Discrete-time model

The state-space model presented in **Equation 5.2** are in continuous-time, however the estimator is to be implemented on a discrete-time system. Computers cannot compute continuous-time differentials directly, and instead a numerical solution must be defined. The discrete-time state-space representation can be written as:

$$x_{k+1} = \mathbf{A_d}\, x_k + \mathbf{B_d}\, u_k \tag{5.3}$$

$$y_k = \mathbf{C}\, x_k \tag{5.4}$$

where $\mathbf{A_d}$: discretization of the state space matrix A
$\mathbf{B_d}$: discretization of the state space matrix B

The discrete state-space representation is described in terms of finite differences. To fit this representation, the continuous-time model, described in **Equation 5.2** are approximated using a finite difference (second order Taylor series) with step-time Δt, yielding the discrete-time model:

$$\begin{bmatrix} p_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_k \\ v_k \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a_k \tag{5.5}$$

The model described by **Equation 5.5** only includes the motion in one dimension, however as the objective is to estimate all six translational states $\boldsymbol{x}_{\text{pos}} = (x, y, z, v_x, v_y, v_z)$, the model is expanded to three dimensions, yielding:

$$\begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{A_d}} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}_k + \underbrace{\begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix}}_{\mathbf{B_d}} \begin{bmatrix} a_x \\ a_y \\ a_x \end{bmatrix}_k \tag{5.6}$$

The output of the system y_k should depict which measurements are available at time k, meaning that the $\mathbf{C}$ matrix is not constant. As a consequence, the system is not always fully observable, and depending on which measurements are available the positional states are not guarantied to converge.

5.3 Kalman Filter

The Kalman filter (KF) is an optimal full-state estimator, that combines a linear system model perturbed by noise with noisy measurements, to estimate unmeasured system states and reduce measurement noise [30]. The Kalman filter is a great tool to infer missing information from indirect and noisy measurements. The goal of the Kalman filter is to produce a state estimate $\hat{\boldsymbol{x}}_k$ that minimises the error of the estimate and the true state $\boldsymbol{x}$, using noisy measurements $\boldsymbol{y}_k$, described by the discrete-time linear system:

$$\boldsymbol{x}_{k+1} = \mathbf{A}_d \boldsymbol{x}_k + \mathbf{B}_d \boldsymbol{u}_k + w_k \tag{5.7}$$

$$\boldsymbol{y}_k = \mathbf{C}\boldsymbol{x}_k + v_k \tag{5.8}$$

Where $\boldsymbol{w}_k$ is the process noise, and $\boldsymbol{v}_k$ is the measurement noise. Both $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are assumed to be zero-mean gaussian noise, with known covariances $\mathbf{Q}_f$ and $\mathbf{R}_f$ [30]:

$$w_k \sim \mathcal{N}(0, \mathbf{Q}_f) \qquad v_k \sim \mathcal{N}(0, \mathbf{R}_f) \tag{5.9}$$

5.3.1 Algorithm

The Kalman filter is an estimator, and runs in parallel with the system, using the system input $\boldsymbol{u}_k$ and output $\boldsymbol{y}_k$ to estimate the true internal state $\boldsymbol{x}_k$ of the system. The Kalman filter are often separated into two steps; the prediction and the update step. The prediction is based on the system model, while the update uses the residual between the prediction and observations to correct the estimate, see **Figure 5.3**

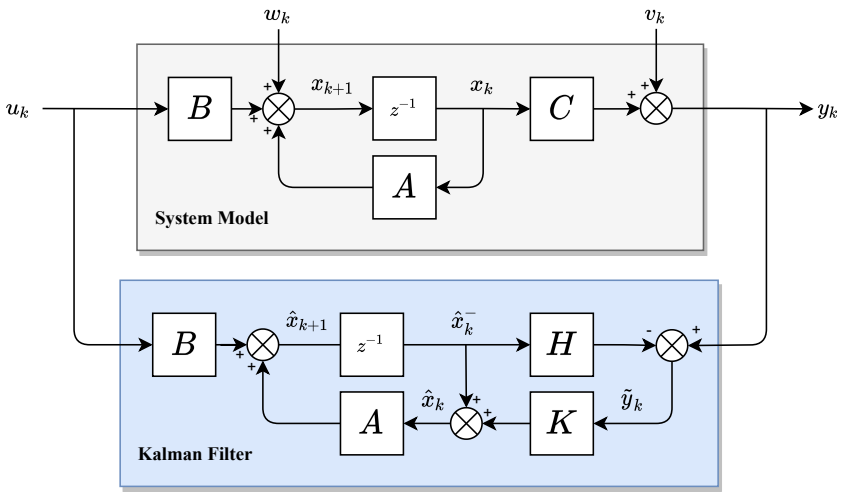


Figure 5.3: Diagram of the discrete system model in combination with the Kalman filter [30].

Prediction step

The prediction step utilises the system model, the input and the prior system state to predict the system state and uncertainty at the newest time-step. This is called the prior estimate, see **Equation 5.10**.

$$\hat{x}_k^- = \mathbf{A_d}\hat{x}_{k-1} + \mathbf{B_d}u_k \tag{5.10}$$

Doing the prediction step, the uncertainty of the prediction $\mathbf{P}_k^-$ is also computed,

$$\mathbf{P}_k^- = \mathbf{A_d}\mathbf{P}_{k-1}\mathbf{A_d}^T + \mathbf{Q}_f \tag{5.11}$$

Update step

The update step improves the predicted state estimate $\hat{\boldsymbol{x}}_k^-$, by combining the prediction with sensor measurements $\boldsymbol{y}_k$. In order to perform the update-step, there must however be new measurements available. This is described by the measurement matrix $\mathbf{H}$ (a variation of the $\mathbf{C}$ matrix), which relates the current system states, with the measurements. Using $\mathbf{H}$ the predicted estimate $\hat{\boldsymbol{x}}_k^-$ is corrected using the measurements by a factor $\mathbf{K}_f$ [30]. The factor $\mathbf{K}_f$ is known as the optimal Kalman gain [30], and is computed as:

$$\mathbf{K}_f = \mathbf{P}_k^-\mathbf{H}^T(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R}_f) \tag{5.12}$$

Using the Kalman gain $\mathbf{K}_f$, the state and process covariance is updated:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbf{K}_f(\boldsymbol{y}_k - \mathbf{H}\hat{\boldsymbol{x}}_k^-) \tag{5.13}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K_k}\mathbf{H})\mathbf{P}_k^- \tag{5.14}$$

If there is no new observations, the measurement matrix $\mathbf{H}$ matrix is set to zero, and the update step does nothing. In such as case, the estimator relies solely on the prediction, and the process covariance (uncertainty in the prediction) would increase until new measurements are ready.

5.3.2 Steady-State filter

The Kalman filter algorithm described above computes a new optimal filter gain at each sample [28] [30]. This process is computational heavy, and does not fit well in an embedded system for real-time state-estimation. A well known feature of the Kalman filter is that for time-invariant systems with constant uncertainty, the filter gain $\mathbf{K}_f$ tends to converge towards a steady state value after the initial transient process [31].

This means that the filter gain can be computed offline, and be used in an digital implementation with ease. With $\mathbf{K}_f$ being constant, there are no longer any need to compute the process error covariance, and only the state prediction and update equations are needed. The steady-state assumption of the estimator is however not perfectly true, as the observations arrives at different intervals ($\mathbf{H}$ is rarely equal to $\mathbf{I}_6$). This would change the process error covariance $\mathbf{P}_k$ in the regular Kalman filter, which in turn would change the optimal Kalman filter gain. The effects of these fluctuations in the uncertainty is however considered negligible, and the computational advantages of the steady-state Kalman filter are utilised.

The steady-state gain $\mathbf{K}_f$ is computed using MATLAB's `dlqe()` command (discrete linear quadratic estimator), using the discrete-time system matrix $\mathbf{A}_d$ and the noise matrices $\mathbf{Q}_f$ and $\mathbf{R}_f$. The measurement noise matrix $\mathbf{R}_f$ describes the covariance of the sensors noise, and is estimated by a diagonal matrix with the entries being the variance of each state measurement, see **Figure 5.15**.

$$\mathbf{R}_f = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{vz}^2 \end{bmatrix} \tag{5.15}$$

The process noise covariance $\mathbf{Q}_f$ is similarly approximated, but rather than keeping the diagonal entries constant, these are used to tune the response of the Kalman filter. If $\mathbf{Q}_f$ are made to small, the filter becomes overconfident in the prediction, causing slow and less responsive behaviour. On the other hand, if the coefficients are made to high, the estimate will be highly influenced by measurement noise. Using these guidelines, a trial-and-error approach is used to compute a filter gain that yields the most desirable attributes.

5.4 Estimation evaluation

To evaluate the performance of the steady-state Kalman filter, a preliminary test has been performed. In this test, the physical platform has been manually carried around in a square pattern, while all measurements are logged, see **Figure 5.4**.

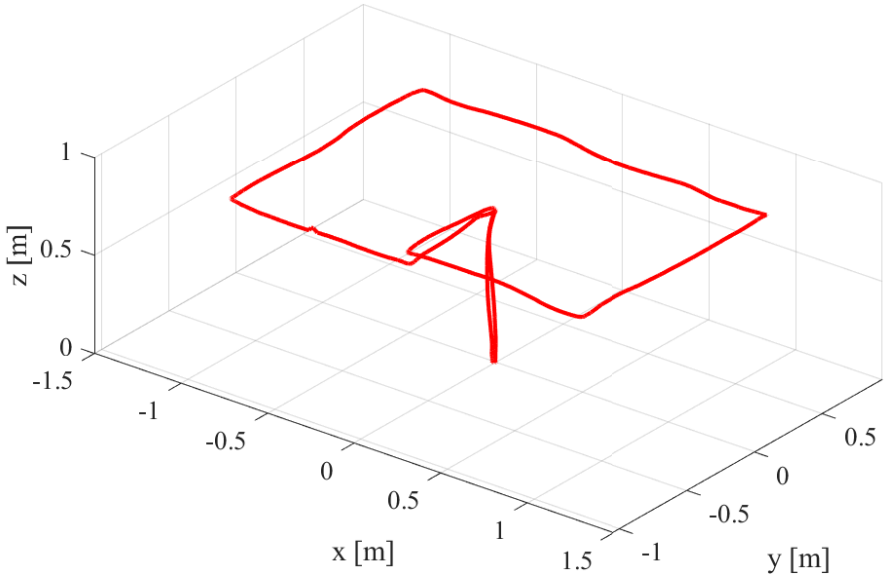


Figure 5.4: Position measured by Vicon (ground truth) during the preliminary test

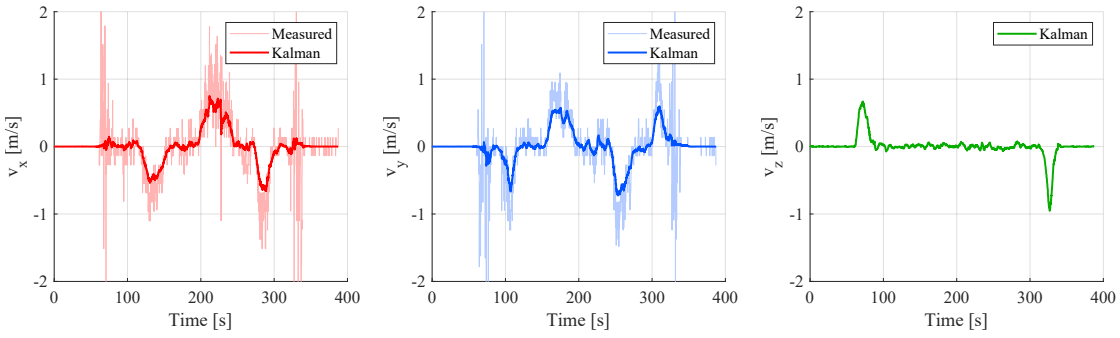
While performing the test, all sensor measurements were logged, and the resulting data has been used to tune the Kalman filter offline. Before the measurements are feed to the estimator, they are rotated to the world frame, using the rotation matrix described by **Equations 3.3**.

Using the data logged from this test, two offline simulations has been performed; one where no positional measurements are supplied, and one where the position is updated occasionally. The first simulation is made to evaluate how the filter performs, when no position measurements are available. Doing this simulation, the system is not fully observable and the positional estimate are certain to drift. The second simulation is made to evaluate the overall performance of the estimator, with positional measurements. Altitude measurements from the LIDAR are supplied in both simulations, as the optical flow relies on this to scale the velocity measurements correctly.

Both offline simulations are run at 200 Hz, and the results thereof are presented in the following sections.

5.4.1 Visual aided inertial estimation

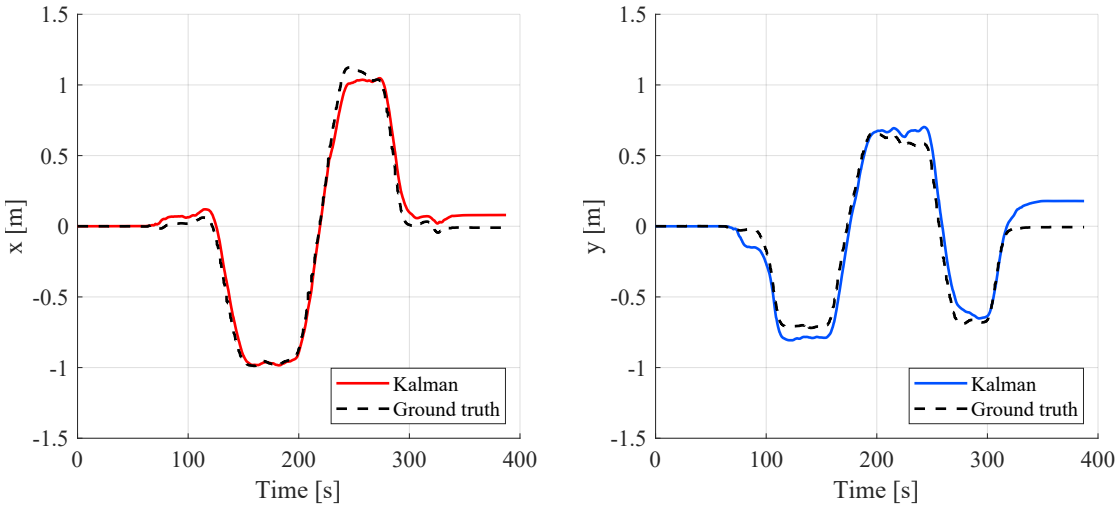
In the first simulation, the estimator is feed with measurements of altitude (z), velocity (v_x and v_y) and with acceleration (a_x, a_y and a_z) as input. Since the velocity measurements are based on optical flow, they are heavily corrupted by noise, however combined with the measurements of acceleration, two smooth velocity estimates (v_x and v_y) are produced, see **Figure 5.5**. The velocity v_z is based on the derivative of the altitude measurements, and is similarly improved by the measurements of acceleration, see **Figure 5.5c**.



(a) x-velocity **(b)** y-velocity **(c)** z-velocity

Figure 5.5: Velocity estimates produced by steady-state kalman filter.

The position estimate is the result of integration, and due to errors in the velocity estimates, the position drifts. However, as seen from **Figure 5.6** the overall translational movement is captured, and the position drifts less than 0,2 m doing the entire test.

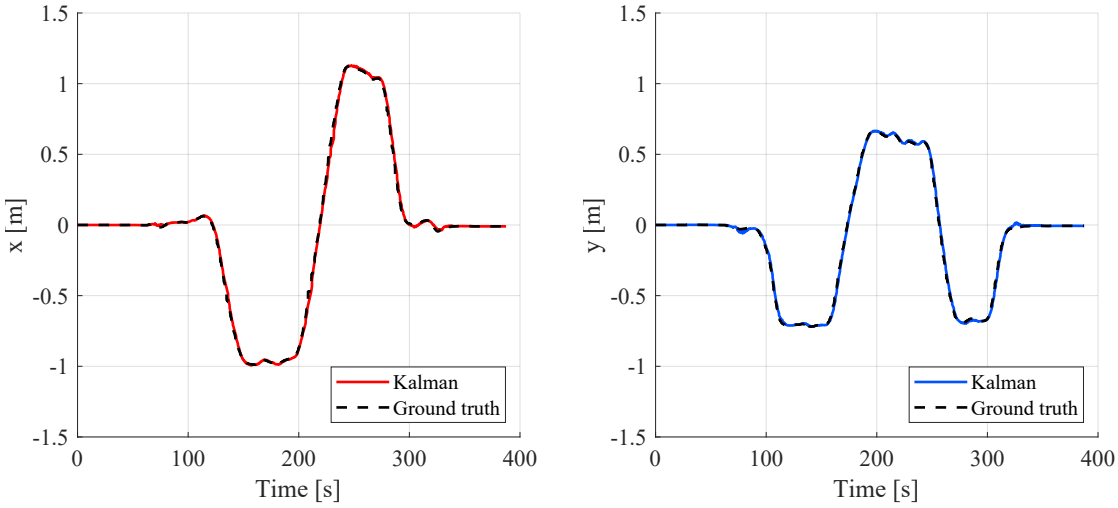


(a) x-position **(b)** y-position

Figure 5.6: Position estimate using optical flow and acceleration.

5.4.2 Position updates

In the second test, the exact same information is feed to the Kalman filter, however with the occasional position measurement (5 Hz) to correct the positional drift. As seen from **Figure 5.7**, the position estimates now follows the ground truth, and the positional drift is eliminated, see **Figure 5.8**.



(a) x-position **(b)** y-position

Figure 5.7: Position estimate updated using Vicon measurements at 5 Hz

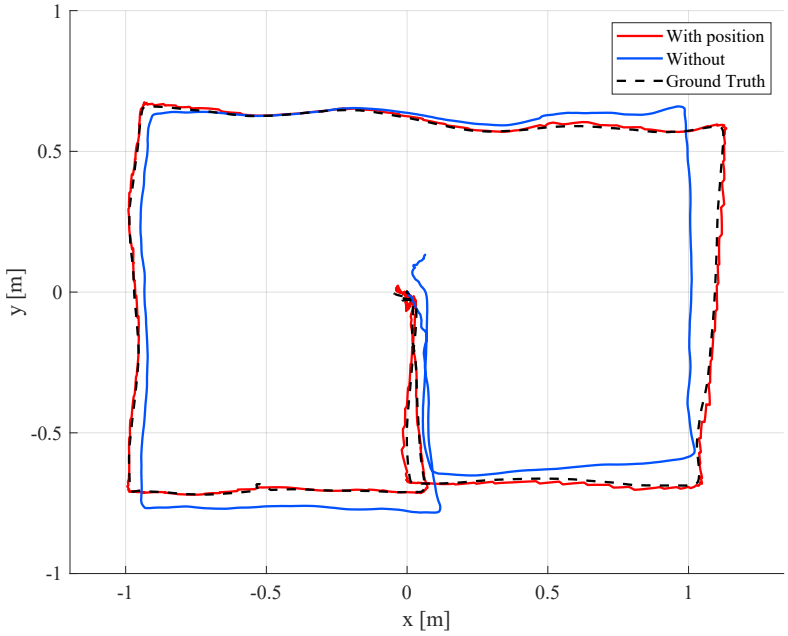


Figure 5.8: Comparison between the position estimate, with and without positional updates.

5.5 Summary

The offline simulations of the steady-state Kalman filter shows that the estimator is able to produce a solid estimate of both position and velocity, even if no positional measurements are available. The estimator can continue to estimate the position in short periods of time even if the positional system is unresponsive.

However, if the estimate drifts to much, the residual of an position observation, could potentially cause instability in the estimator. This is the result of the steady-state assumption, as the filter applies the same gain, no-matter how much uncertainty has been accumulated. To account for this, the steady-state kalman filter is tuned to perform sub-optimal, such that a positional measurement does not cause to big of a update. The estimator does however still produce a very solid estimate, that in general performs better than the Vicon measurements, see **Figure 5.9**.

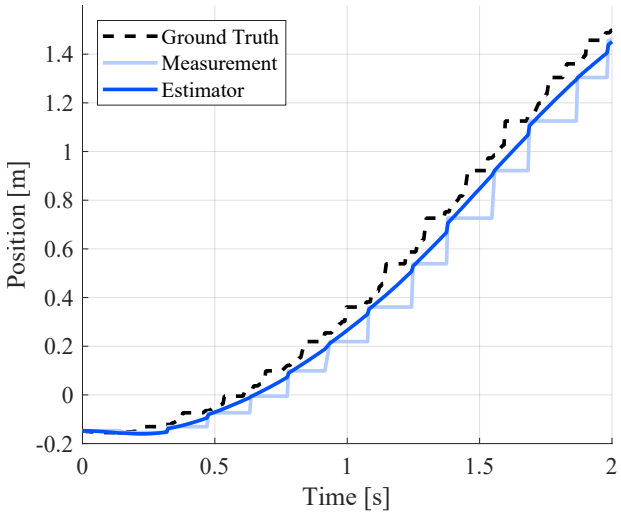


Figure 5.9: Comparison between the ground truth, the positional measurement update and the estimate. The estimator interpolates the position in-between measurements, however lags behind the ground truth by 0,1 s.

The sampling and wireless transmission of the Vicon measurements, does however introduce a small time-delay of approximately 0,1 s in the estimate. This is barely noticeable, however, will have an noticeable effect on the position controller, which only has a *time delay margin* of 0,4 s, see **Section 4.4.3**. Nonetheless, the filter performs well, and the time-delay is within the acceptable margin, and therefore no other action is needed in order to implement the control system. The next chapter will describe and discuss the considerations taken in order to implement the control system on the digital platform.

CHAPTER 6

Software

This chapter will describe the structure of the digital implementation, and explain the software needed to implement the full-state feedback and estimation. The software is a crucial part of the project, as it enables the evaluation of the designed control system in practice.

The software developed for the project is freely available under the open-source MIT license, and can be accessed online at Github: github.com/SolidGeek/SingleRotorUAV.

6.1 Software structure

The software developed for the mono-copter is separated into two different solutions; the ground station and the flight controller. The flight controller software is written in C++, and is divided into a subset of classes, each handling a particular function. The ground station on the other hand, is written in MATLAB, and is developed using MALTABs app designer that provides the primary graphical features. An overview of the software structure is shown in **Figure 6.1**.

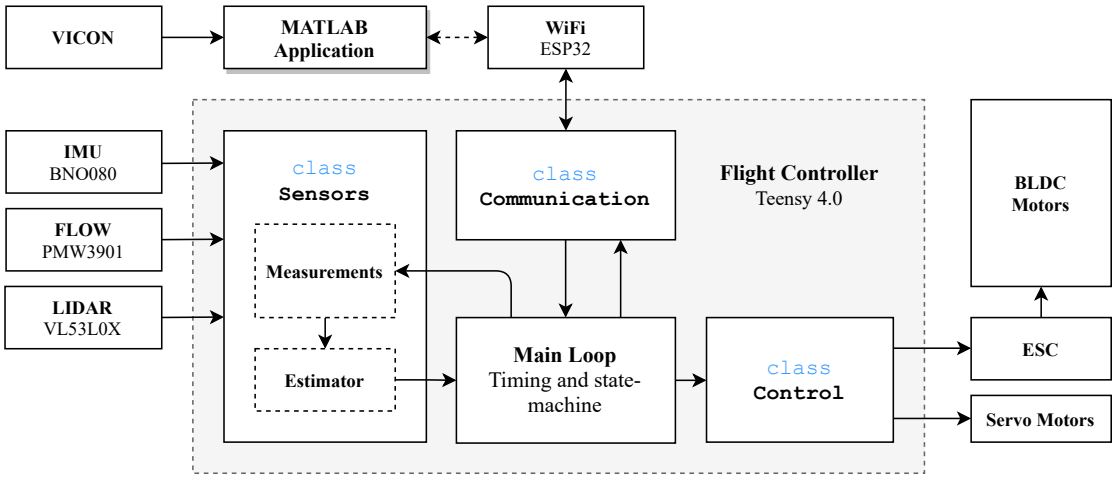


Figure 6.1: An overview of the two software solutions and their interaction.

6.2 Flight Controller

The flight software is rather complex, and is therefor divided into a subset of classes to better organise and reuse the code. A total of five classes are written, these being:

- `Sensors()`
  Samples each sensor and fuses the measurements in the estimator
- `Control()`
  Implements full-state feedback control to generate appropriate control signals.
- `Communication()`
  Handles the WiFi communication between ground station and mono-copter.
- `Config()`
  Enables storage of non-volatile configuration and calibration parameters.
- `DShot()`
  Configures two DShot outputs to control the motor ESCs

6.2.1 Main Loop

The flight software runs in an infinite loop, that periodically calls the functions of the software classes. The timing of these function-calls are handled by keeping track of only two timers; the control/estimator timer and the sensor timer, see **Source 6.1**. The sensor timer is used to sample the TOF and FLOW sensor, while the IMU is sampled as often as possible. The continuously sampling of the IMU is possible, because the IMU is configured to pull a pin high when a new measurement is available, and therefor no additional computation-time is wasted.

```
1  void loop() {
2      // Sample IMU as fast as possible (sample-rate is limited by sensor)
3      sensors.sample_imu();
4
5      // Sample FLOW and TOF at 50 Hz
6      if( micros() - sensor_timer >= 20000 ){
7          sensor_timer = micros();
8
9          sensors.sample_flow();
10         sensors.sample_tof();
11     }
12
13     // Run control and estimator at 200 Hz
14     if( micros() - control_timer >= 5000 ){
15         control_timer = micros();
16
17         sensors.run_estimator();
18         control.run( sensors.data, sensors.estimate );
19     }
20 }
```

Source 6.1: The basic timing performed in the infinite loop. The function `micros()` returns the time in microseconds since startup.

Correct timing is important to ensure stability of any control system, as the discrete system poles changes with sample-frequency. Therefore, to guarantee stability, the sample-frequency must be kept constant. However, the timing method used in the flight controller does not guaranty perfect timing, as nothing stops a process, should it be using to much processing-time. Nonetheless, to ensure that the sample-frequency is kept within an acceptable margin, the software is developed using non-blocking functions, and the communication is kept in buffer until it can be read. A better approach would be the use of a real-time kernel such as the Free real-time operating system (Free-RTOS) [32].

6.2.2 Class: Sensors

The sensor class handles the setup and sampling of each sensor and combines the measured data in the estimator. The purpose of this class, is to provide the control system with measurements and estimates of the system states. Each time a new measurement becomes available, a data structure is updated with the measurement and a boolean flag regarding the state of that measurement is updated, see **Source 6.2**. Using this flag, next time the estimator is called the new measurement will be used as an observation to update the state estimate.

```
typedef struct {
    float gx, gy, gz;
    float roll, pitch, yaw;
    float ax, ay, az;
    float vx, vy;
    float x, y, z;
    struct{ // Struct to represent if new measurements are available
        uint8_t imu     : 1;
        uint8_t flow    : 1;
        uint8_t tof     : 1;
        uint8_t pos     : 1;
    } status;
} sensor_data_t;
```

Source 6.2: The sensor data structure used to hold the newest measurements.

The estimator consists of multiple matrix operations, which is time-consuming to implement as multiple 1-dimensional computations. Instead the library *BasicLinearAlgebra* [33] is applied, as this implements standard matrix notion in C++. Using this library, the $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{K}_f$ matrices of the estimator can be implemented directly in software, see **Source 6.3**, and can be used for multiplication, addition and substraction.

```
Matrix<6,6> A = {   1,  0,  0,  DT, 0,  0,
                    0,  1,  0,  0,  DT, 0,
                    0,  0,  1,  0,  0,  DT,
                    0,  0,  0,  1,  0,  0,
                    0,  0,  0,  0,  1,  0,
                    0,  0,  0,  0,  0,  1 };
Matrix<6,3> B = { ... };
Matrix<6,6> Kf = { ... }
```

Source 6.3: Declaration of the estimator model ($\mathbf{A}$, $\mathbf{B}$) and the steady-state kalman gain ($\mathbf{K}_f$)

6.2.3 Class: Control

The control class, implements the two full-state feedback controllers designed in **Chapter 4**; the hover and position controller. This is implemented using the same matrix library as the estimator, enabling direct declaration of the gain matrices $\mathbf{K}_{\text{hov}}$ and $\mathbf{K}_{\text{pos}}$. Each controller is implemented as its own function, such that they can be called individually. The input arguments are the values of the system states, see **Source 6.4**, while the set-points are kept as private variables in the control class.

```
1 // Hover controller
2 void control_hov( float roll, float pitch, float yaw, float gx, float
  ↪  gy, float gz, float z, float vz );
3
4 // Position controller
5 void control_pos( float x, float y, float vx, float vy, float yaw );
```

Source 6.4: Header declaration of the hover and position controller. The position controller uses yaw to rotate the position error to body frame.

Changing of set-points is only needed for attitude and position, and are done using the function `void set_reference( uint8_t setpoint, float value );`. Using this function, the attitude and position of the mono-copter can be controlled, allowing for full 3D-movement.

6.2.4 Class: Communication

The communication class, handles the communication between the ground station and the mono-copter. The WiFi module is programmed to generate a local WiFi network, on which it listens for UDP commands from the ground station. When a command is received, the data is relayed to the flight controller over UART. The same goes in the opposite direction, where the WiFi module listens for UART packages (telemetry) and broadcasts these as UDP messages to the ground station, see **Figure 6.2**.

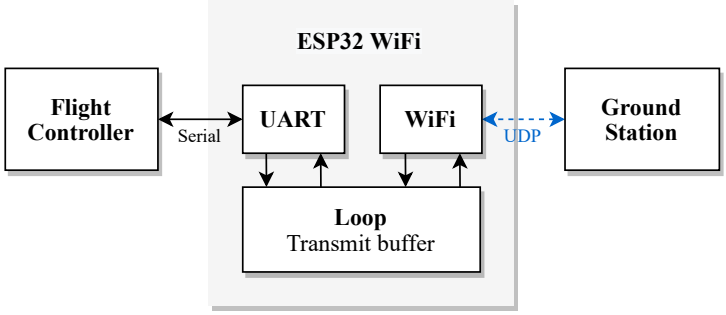


Figure 6.2: The WiFi module acting as a communication relay.

6.2.5 Class: Config

The config class, enables non-volatile storage of configuration and calibration parameters. This is done by reading and writing a data structure to a region in the flash memory that is not used for the application. The data structure can be as large as 2 kB, however only 8 B are used to store the calibrated servo offsets, see **Source 6.5**. Since the data is stored in a structure, other parameters could easily be added such as the controller gains, or initial values for controller integrals computed on the fly.

```
1 struct config{
2     int16_t servo_offset[4];
3 } params;
```

Source 6.5: Header declaration of the configuration data structure.

6.2.6 Class: DShot

The DShot class, contains the functionality and configuration needed to generate two DShot signals to control the ESCs. This is achieved using internal hardware timers and interrupt driven direct memory access (DMA), to offload the processor. DShot is a digital protocol that offers CRC checksum, a high resolution and does not require throttle calibration [34]. The DShot packet contains a total of 16 bits, with each bit being represented by a PWM pulse. The bit-value 0 and 1 are distinguished by different pulse lengths. The DShot data packet is divided into three elements [34]:

- Throttle (11 bits)
- Telemetry request (1 bit)
- Checksum (4 bits)

To request telemetry from the ESCs, the telemetry request bit is set high, and the telemetry is received as serial data on the telemetry wire. The data received are unpacked and stored in a data structure, see **Source 6.6**.

```
1 typedef struct {
2     uint8_t temp;
3     uint16_t voltage;
4     uint16_t amps;
5     uint16_t ampHours;
6     uint16_t rpm;
7 } DSHOT_telemetry;
```

Source 6.6: Header declaration of the telemetry structure

6.3 Ground Station

The ground station has three important functions; sending commands, displaying telemetry and logging. The transmission of commands and logging, could be achieved using a simple terminal script, however the live plotting of telemetry requires a graphical user interface (GUI). MATLABs App designer was chosen for this, as MATLAB comes with a lot of pre-built features such as plots, inputs and an UDP server class (for telemetry). As seen from **Figure 6.3**, the designed GUI includes both plots and controls.

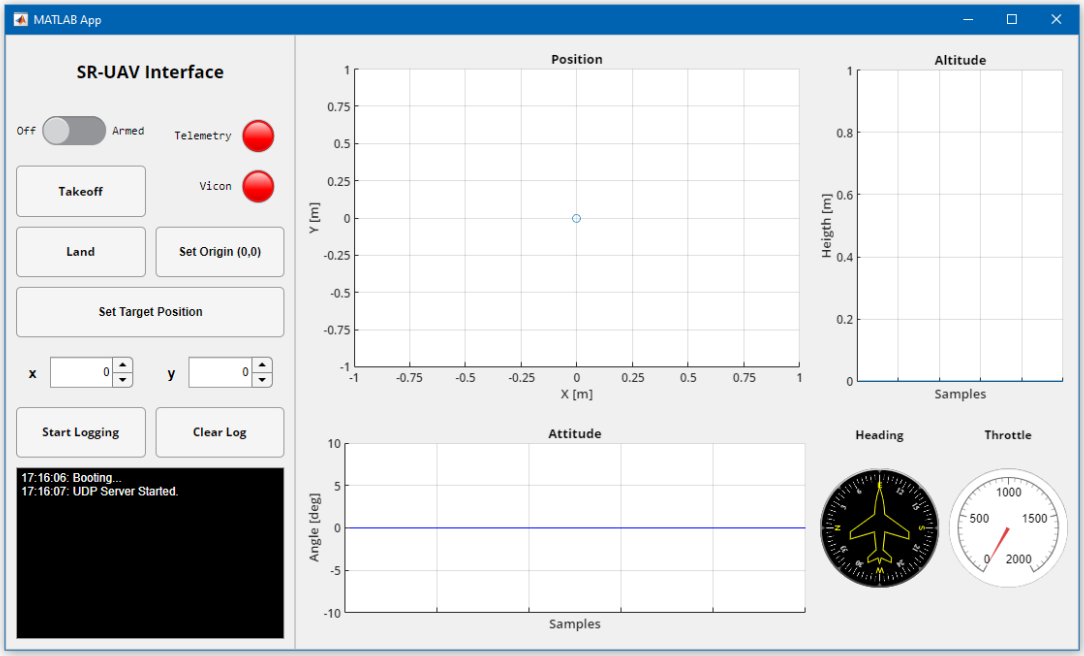


Figure 6.3: Graphical User Interface developed using MATLAB App designer. Three plots are used to illustrate the movement of the mono-copter; horizontal position, altitude and attitude.

6.3.1 Command structure

To send commands from the GUI to the mono-copter, a command structure had to be established. The commands needed are generally simple, and does not have to contain more than an identifier, however some commands, such as changing a set-point, requires additional data. The command structure is therefor composed of an command identifier of type `uint8_t` and a array of `float` for additional data, see **Source 6.7**.

```
1 typedef struct{ uint8_t command; float value[3]; } command_t;
```

Source 6.7: Header declaration of the command structure.

6.3.2 Telemetry and logging

Logging of the received telemetry is critical to document the response of the system, but also to gain knowledge from every flight. By logging the telemetry on the ground station, a new test can be initiated immediately, rather than extracting the data from a SD card or similar. To perform the logging, a data structure is created in MATLAB, that can hold all the measurements, see **Source 6.8**. For each telemetry package received, a complete lines of measurement are added to this structure. However, because the telemetry is transmitted at the same frequencye as the control loop runs (200 Hz), data can easily be lost. To limit the data-loss, a buffer is used, that holds the UDP packages until the GUI has time to write them all at once.

```
log_fields = { ...
    'time', ...                 % Timestamp
    'gx','gy','gz', ...         % Angular-rate (imu)
    'roll','pitch','yaw', ...   % Attitude (imu)
    'ax','ay','az', ...         % Acceleration (imu)
    'vx','vy','vz', ...         % Velocity  (flow sensor)
    'x','y','z', ...            % Internal position sensor (no sensor)
    'vxt','vyt','vzt', ...      % Velocity estimates (kalman)
    'xt','yt','zt', ...         % Position estimates (kalman)
    'a1','a2','a3','a4','dshot', ...  % Actuation signals (lqr)
    'xe','ye','ze', ...               % External position (vicon)
    'stat_imu','stat_flow','stat_tof','stat_pos' ... % Status
};
```

Source 6.8: MATLAB data structure used for logging.

6.4 Summary

The software developed for both the flight controller and ground station, are described at a minimal, trying to give an general idea of the underlying processes of the digital implementation. The reader is referred to the Github repository published for this project, if a higher level of detail is needed.

With the digital implementation working, the controller, estimator and hardware can be tested in practise. The next chapter will describe how the combined system is tested and the results thereof.

CHAPTER 7

Results

In order to validate the performance of the physical system, a series of tests has been performed that aims to evaluate the designed controllers and validate the three main objectives of this thesis; vertical take-off, hovering and landing. This chapter will describe the results of these tests and compare the empirical data with the nonlinear simulation results.

7.1 Attitude Control

It is essential to evaluate the attitude control (roll and pitch) before reaching any of the main objectives, which requires actual flight. Without reasonable attitude control, the mono-copter will never achieve stable flight, as just a tiny error in attitude has cascading effects on the rest of the system.

7.1.1 Test Setup

To evaluate the stabilisation of roll and pitch, without being in flight, a test bench has been constructed. The test bench is a dual-axis gyroscope construction that allows the mono-copter to move in two degrees of freedom (roll and pitch), see **Figure 7.1**.

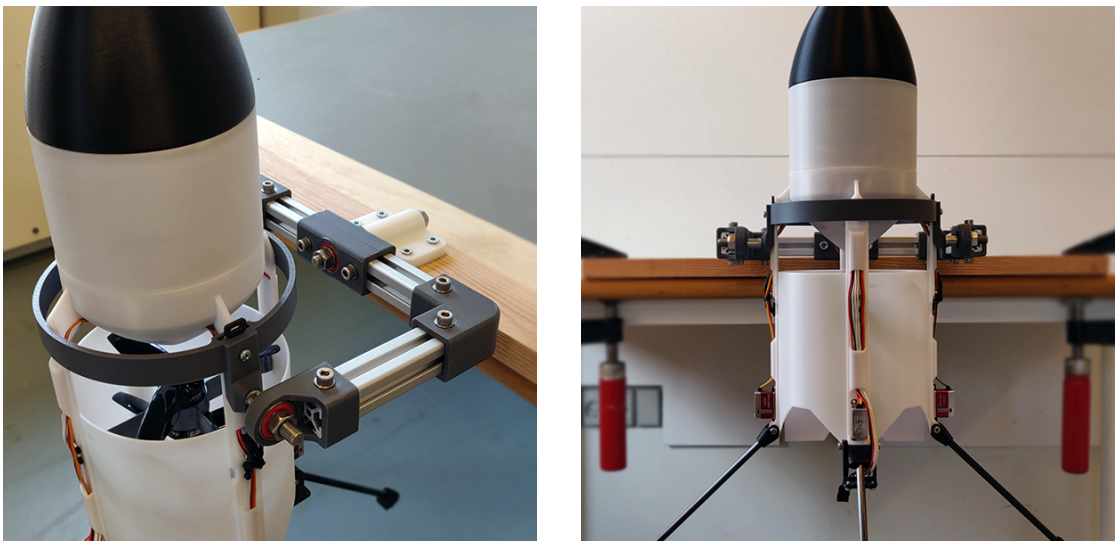
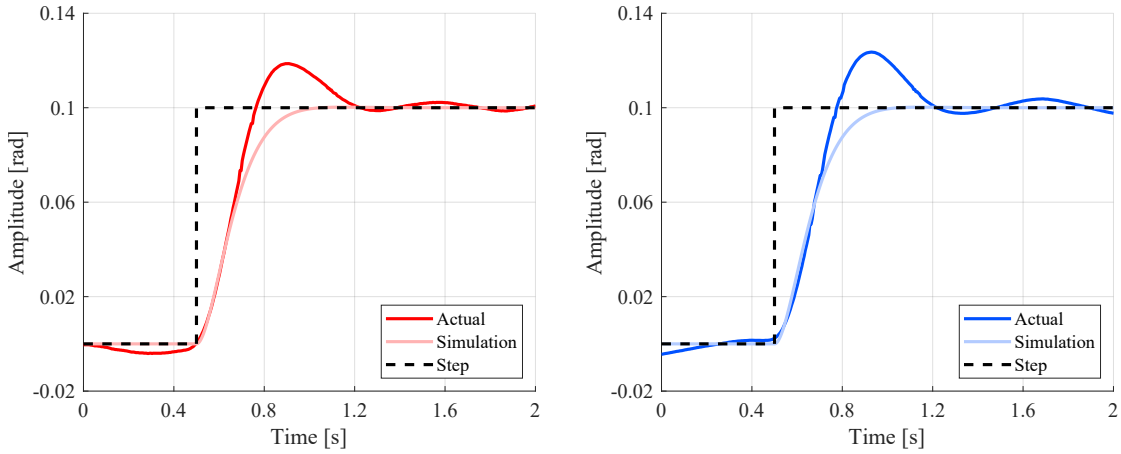


Figure 7.1: 2-DOF test bench, constructed to evaluate the control of roll and pitch. The rotational joints uses ball-bearings to reduce friction, and the structure is made using aluminium extrusions and 3D-printed brackets. The entire setup is mounted on a table, using screw clamps.

Using the test bench, the mono-copter is securely mounted, and the ESCs are commanded to spin the motors at a velocity equal to the value of the hover-point. When the motors reach the desired velocity, a step is performed on each axis separately, and the results are logged wireless by the ground station.

7.1.2 Results

A step of 0,1 rad ≈ 6° is applied to the roll and pitch control (ϕ_{ref}, θ_{ref}) and the step-responses thereof are plotted alongside the simulated, see **Figure 7.2**. As seen from the plots, the attitude is quickly stabilised, and the rise-time of 0,2 s for both steps are near identical to their simulated counterpart. Both tests does, however, generate an overshoot of roughly 20 %, which is not the case in simulation.



(a) Roll with a step of 0,1 rad.

(b) Pitch with a step of 0,1 rad.

Figure 7.2: Comparison between simulated and real step-response of 0,1 rad on roll and pitch.

To further test the attitude control, a sequence of varying steps is applied to the roll direction, see **Figure 7.3**. This test shows that the attitude controller is capable of stabilising roll for varying step inputs, with the same overshoot of approximately 20 %.

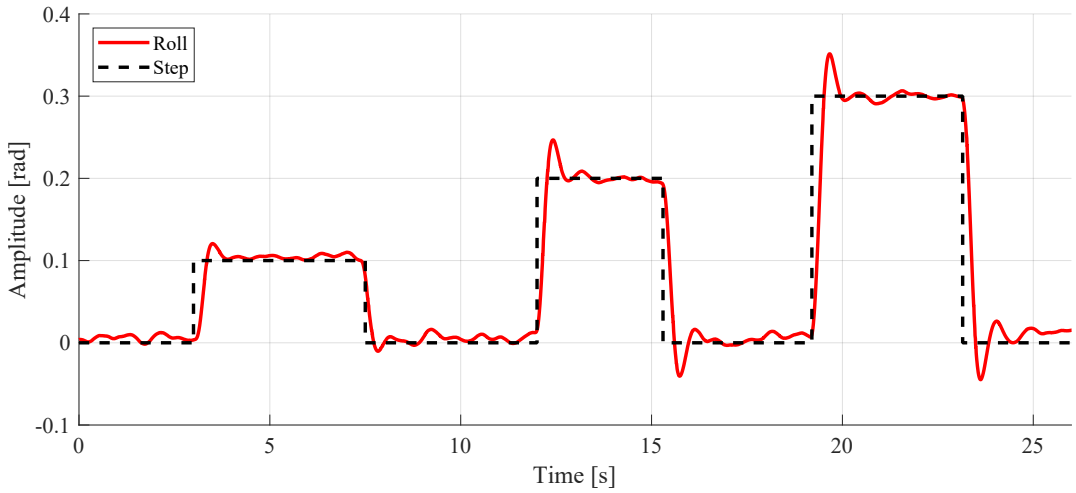


Figure 7.3: Sequence of steps of varying size (0.1, 0.2 and 0,3 rad) on the roll control ϕ_{ref}.

7.2 Altitude

With the attitude control being stable, a step on the altitude can be used to evaluate two of the main objectives; vertical take-off and landing. In order to achieve vertical take-off, the motors has to produce enough thrust to counteract gravity. This is handled by the hover controller and to test this, a step has to be applied on altitude (z_{ref}). The landing phase is more complicated, as changing the reference to zero, would result in the system crashing to the ground. Instead, when a landing is initiated, the previous reference is kept constant, and the integral is reduced slowing.

7.2.1 Test Setup

To evaluate the altitude control, without drifting to far from the origin, a cable is connected to the mono-copter through a hook in the ceiling. The cable helps to limit the uncontrolled positional movement and to catch the mono-copter should something go wrong, without limiting the z-movement, see **Figure 7.4**.

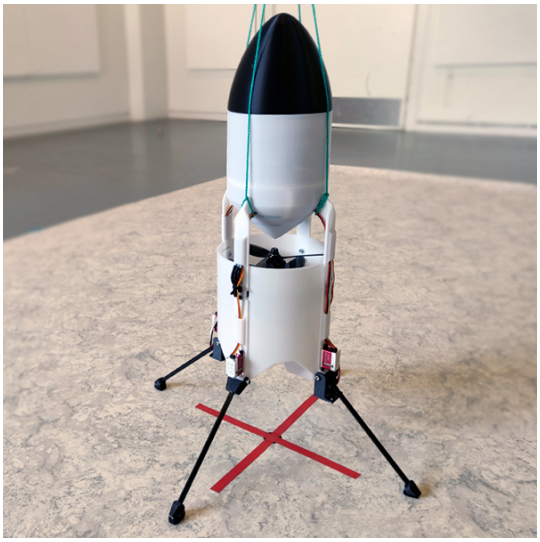


Figure 7.4: 4-DOF setup (x and y restricted) used to evaluate the control of altitude.

As this is an actual flight test, the Vicon system is used to keep track of the position, and will be used as ground truth. The on-board estimator is used as the control feedback, and no positional data is feed to the mono-copter from Vicon.

7.2.2 Results

A step of 0,5 m is applied to the altitude control (z_{ref}), and the step-response thereof is plotted alongside the corresponding response of the simulation, see **Figure 7.5**. Seen from the graph, the step-response of the physical system closely matches that of the simulation, with a rise-time of 1,39 s. Furthermore, the slowly growing integral of the altitude, has nearly the same time-delay as in the simulation. The small increase in over- and undershoot is considered insignificant.

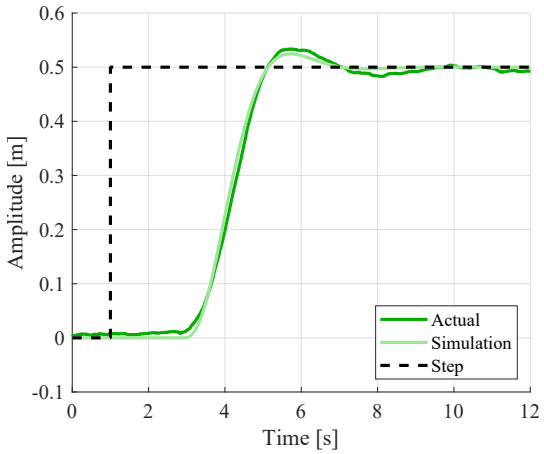


Figure 7.5: Comparison between actual and simulated step-response of 0,5 m on the altitude z_{ref}.

After the vertical take-off, the landing sequence is initiated, see **Figure 7.6**. The controlled vertical landing is performed successfully, however, with to high velocity causing a small bump at landing. The rough landing causes an sudden change in acceleration, which by the estimator is interpreted as negative motion.

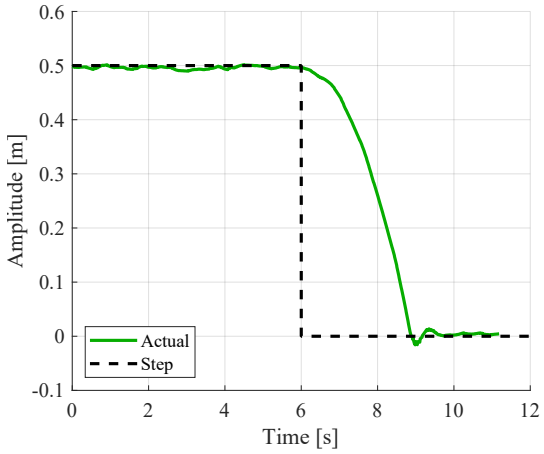


Figure 7.6: The landing procedure, after an initial step of 0,5 m on the altitude.

7.3 Position

With both the attitude and altitude stabilised, the semi-stable hover state is achieved, however, this does not guarantee stability of the horizontal translation. The position control is needed to stabilise the horizontal system states, as would otherwise drift due to disturbances and steady-state errors. To evaluate the entire system stability, the positional controller is evaluted by two individual tests. The first test performs a standard step on a single axis, while the second test uses a nonlinear trajectory.

7.3.1 Test Setup

To evaluate the position control, the same setup as for the altitude test is used. This time, however, the Vicon system is used to supply positional updates to the on-board estimator at 5 Hz. The nonlinear trajectory used as reference for the last test, is generated by the flight controller, which after the initial take-off phase, feeds x- and y-coordinates to the position controller (x_{ref}, y_{ref}).

7.3.2 Results

For the first test, a step of 2 m is applied to the y-axis (y_{ref}). The step-response is plotted alongside the simulation result, see **Figure 7.7**. From the plot, it is clear that the position is successfully stabilised, however, that the implemented controller has a steady-state error and undershoots considerable. Additionally, it can be observed that the implemented controller is slightly slower with a rise-time of 1,41 s, rather than the 1,20 s of the simulated.

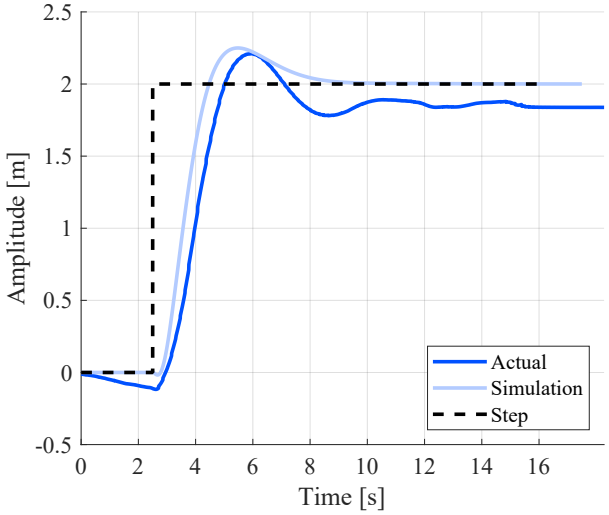


Figure 7.7: Comparison between actual and simulated step-response of 2 m on the y-position y_{ref}.

The second test, is done to validate the combined performance of the mono-copter, by supplying a nonlinear path as reference; a circle with a radius of 0,5 m. The actual trajectory (position) is plotted alongside the desired path, see **Figure 7.8**.

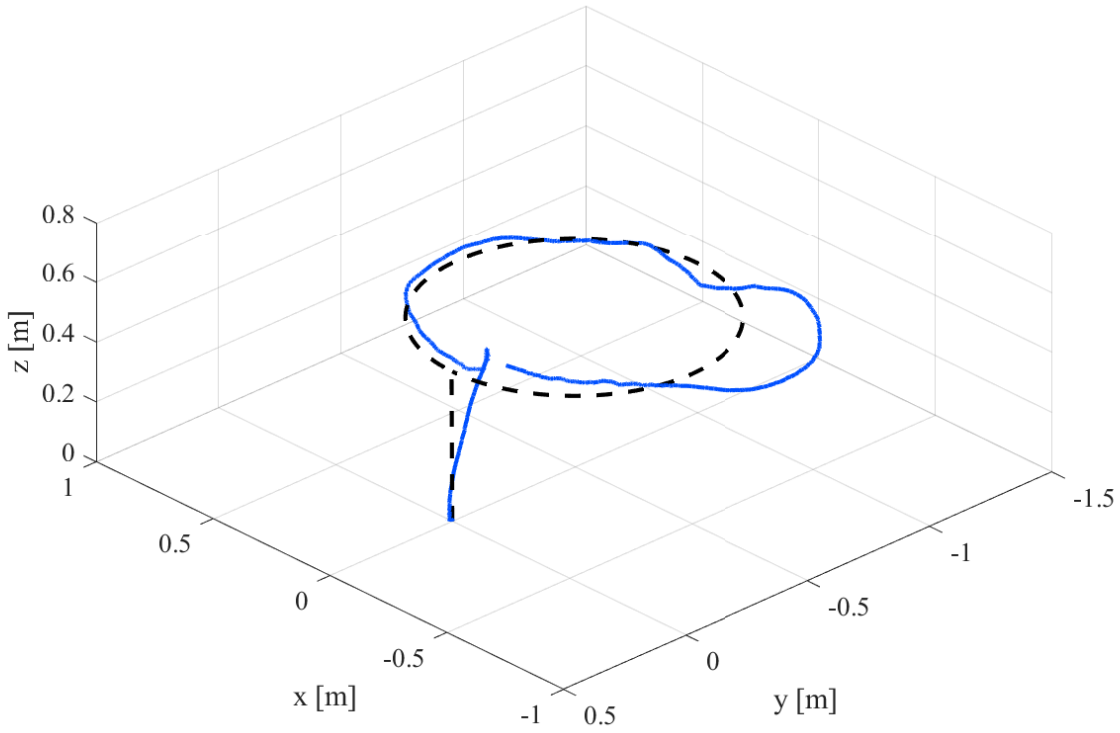


Figure 7.8: Trajectory of the mono-copter following a circular path with radius 0,5 m.

As seen from **Figure 7.8**, the nonlinear path is tracked with varying success. Around half-way, the tracking suddenly performs poorly. Doing this time, it was observed that the yaw angle did not follow its reference, but slowly drifted. This has to some degree been observed in all flights, however only causes problems over longer flights. The problem is observed even if the mono-copter are stationary on the ground, but only with the motors spinning.

7.4 Summary

The results presented in this chapter, showcases the empirical performance of the implemented control system. Multiple tests has been performed, to evaluate the proposed control strategy and the performance of the designed system. The next chapter will discuss the observations made from these results, and discuss the decisions and methods used to reach the objectives of this thesis.

CHAPTER 8

Discussion

This chapter will evaluate the results presented in the previous chapter, and discuss the overall performance of the proposed mono-copter system.

8.1 Results

The results presented in **Chapter 7**, demonstrates that the proposed control strategy, is able to stabilise the unstable dynamics of the mono-copter. The data acquired from these tests is near identical to that of the simulation, which suggests that the assumptions and simplifications made to develop the dynamic model has been well founded. Nonetheless, deviations of the empirical data from the simulated response is present, which is to be expected as the controller has to deal with measurement noise, time-delays, and dynamics which has not be included in the model. The difference between the simulation and the empirical results will be used to discuss the performance of the control system in the following sections.

8.1.1 Attitude control

The overshoot experienced by the roll and pitch control, see **Figure 7.2**, are not experienced in simulation or during actual flight. As a result, the unexpected behaviour is thought to be the effects of the added inertia of the test bench. The added inertia (mass) changes the dynamics of the mono-copter, making the controller, which is designed for a lower mass, perform a little under-damped. Similar overshoot is experienced in simulation, when the moment of inertia terms (J_{xx} and J_{yy}) are increased, while keeping the controller gains constant. The attitude control (roll and pitch) is therefor considered to perform better than the presented results, when the mono-copter is in flight. The yaw control on the other hand, proved difficult to evaluate, as the yaw estimate from the IMU quickly drifts after take-off. The controller stabilises the yaw rotation, however, because the estimate drifts, the controller does not have a solid reference.

8.1.2 Altitude control

The altitude control performs almost identical to that of the simulation, see **Figure 7.5**. This is the result of an accurate model and a relatively slow controller, that keeps the system close to the operating point. The landing on the other hand, could be improved considerable, f.x. by designing a separate controller used only in the landing phase.

8.1.3 Position control

The slower rise-time and under-damped response experienced on the implemented position controller, see **Figure 7.7**, is not observed in simulation, unless a time-delay is introduced in the feedback loop. In consequence, the less ideal response of the position controller might be contributed to the time-delay introduced by measuring, transmitting and estimating the position. To reduce the time-delay, the regular Kalman filter (not steady-state) might be utilised, where the gains can be tuned more aggressively. An alternative method, would be to use an on-board positioning system such as GPS, such that the transmission delay from one system to another is minimised. Nonetheless, the delay is small enough to keep the system stable, and reduction is only needed to improve performance.

The most noticeable problem with the position controller, is that of the steady-state error. This should not be the observed, as the controller is augmented with integral action, which is designed to suppress any steady-state errors. The occurrence of steady-state error is considered the result of the used anti-windup method, which for the implemented system is configured with a too low limit. However, increasing the integral limit, is likely to increase the overshoot, which is not desired. Another approach, is to reduce the effects causing the steady-state error, such as balancing the weight distribution or reducing the bias on the actuators.

8.2 Implementation

The hardware presented in **Chapter 2**, is based on the concept of a counter-rotating motor setup, that in theory should counteract the rotational momentum of one single motor. In practise this did not work, and yaw control had to be implemented to remove the net-torque generated by the differences in motor velocity. Furthermore, the dual motor setup generates a lot of noise, caused by increased turbulence, drag and vibrations. The principle of counter-rotating motors is good, however, in practise a better choice might be a single motor ducted fan (EDF).

Another more prominent issue of the hardware design, is the placement of the IMU. As previously described, the yaw estimate drifts whenever the motors are running. The IMU fuses angular velocity measurements with measurements of the earths magnetic

field, to produce a yaw estimate with a constant reference; magnetic north. However, due to the close proximity of the IMU to the motors, the magnetic measurements are corrupted by magnetic noise produced by the motors. To reduce the influence of the motors, the IMU must be moved, or shielded somehow.

8.3 Appliance

All the tests performed has been in perfect conditions, with no wind and with very precise positional measurements. Reusable rocket, however, operate outdoors with varying weather and wind gusts. A small body such as the proposed mono-copter is much more susceptible to wind and air resistance, and is therefor not suitable for outdoor flight in its current state. However, the small form factor does have its merits in an indoor and wind-free environment, as it originally is intended.

The mono-copter platform is however not only usable in the pursuit of resuable rockets, but shows great general flight capabilities that might have merits in other fields. One inherent feature of the mono-copter is the reduced risk of human injuries, as the propellers are enclosed.

8.4 Literature

While previous research performed by Carholt et al. [8] purely focused on simulation, the results provided in this thesis uses a practical approach to demonstrate that the mono-copter is a robust platform, that is capable of autonomous flight. The model derived for the mono-copter expands on the modelling principles used to describe multi-rotors [6], and contributes with a simple approach to describe the forces generated by the thrust vectoring system. Considering the large gap in research on mono-copters, this thesis aims to provide a novel contribution in terms of combining the needed theoretical building blocks, in designing, modelling and stabilising a mono-copter.

CHAPTER 9

Conclusion

The focus of this thesis, has been to develop a mono-copter that resembles a reusable rocket, and design a control system capable of stabilising the unstable dynamics thereof. This has been attempted using the linear quadratic regular (LQR) based on a dynamic system model, derived using the Newton-Euler formulation of a rigid rotating body. Now recall the problem-formulation:

> "*How can one design and stabilise an autonomous UAV that uses a single source of thrust, to achieve vertical take-off and landing, similar to that of a reusable rocket.*"

The principle of thrust vectoring control (TVC) has been applied to an unmanned aerial vehicle (UAV), in order to stabilise the attitude and position using only a single source of thrust. This concept is named a mono-copter, and closely resembles the flight principle used by modern reusable rockets. The mono-copter platform is custom built, using 3D-printing and low cost and readily available avionics and actuators. The stabilising is achieved using two full-state feedback controllers, designing using a linearized system model and the well-known linear quadratic regulator (LQR). In order to stabilise the system using full-state feedback, a steady-state Kalman filter is designed and implemented, supplying state estimates for both controllers.

In conclusion, the designed mono-copter is capable of performing vertical take-off, hovering and landings, using a model-based control strategy designed using LQR. The proposed control strategy is capable of stabilising the mono-copter in mid air, and can recover from large steps and disturbances. On the basis of that, the proposed system is able to replicate key aspects of reusable rockets, while providing the groundwork used to test and apply concepts of reusable rockets in a safe environment.

Bibliography

[1] Steinar Lag. *Reusable rockets: revolutionizing access to outer space*. URL: https://www.dnv.com/to2030/technology/reusable-rockets-revolutionizing-access-to-outer-space.html.

[2] *New Shepard*. URL: https://www.blueorigin.com/new-shepard/.

[3] *Falcon 9*. URL: https://www.spacex.com/vehicles/falcon-9/.

[4] Michael Belfiore. *The Rocketeer*. 2013. URL: https://foreignpolicy.com/2013/12/09/the-rocketeer/.

[5] Caleb Henry. *America's space industry has a hiring problem, and it must battle the Silicon Valley to solve it*. May 2018. URL: https://spacenews.com/americas-space-industry-has-a-hiring-problem-and-it-must-battle-the-silicon-valley-to-solve-it/.

[6] Marcus Greiff. "Modelling and Control of the Crazyflie Quadrotor for Aggressive and Autonomous Flight by Optical Flow Driven State Estimation". MA thesis. Department of Automatic Control: Lund University, 2017.

[7] Guilherme V. Raffo, Manuel G. Ortega, and Francisco R. Rubio. "An integral predictive/nonlinear H-infinity control structure for a quadrotor helicopter". In: *Automatica* 46.1 (2010), pp. 29–39. ISSN: 0005-1098.

[8] O. C. Carholt et al. "Design, modelling and control of a Single Rotor UAV". In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. 2016, pp. 840–845.

[9] Army-technology.com. *Honeywell T-Hawk Micro Air Vehicle (MAV)*. 2007. URL: https://www.army-technology.com/projects/honeywell-thawk-mav-us-army/.

[10] Dan Maloney. *Single-Rotor Drone: A Thrust-Vectoring Monocopter*. 2018. URL: https://hackaday.com/2018/08/31/single-rotor-drone-a-thrust-vectoring-monocopter/.

[11] A. Bacchini and E. Cestino. "Key aspects of electric vertical take-off and landing conceptual design". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 234 (2020), pp. 774–787.

[12] Eric Stoneking. "Newton-Euler Dynamic Equations of Motion for a Multi-Body Spacecraft". In: (Aug. 2007).

[13] P. Foehn and D. Scaramuzza. "Onboard State Dependent LQR for Agile Quadrotors". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6566–6572. DOI: 10.1109/ICRA.2018.8460885.

[14] Wojciech Giernacki et al. "Performance of Coaxial Propulsion in Design of Multi-rotor UAVs". In: vol. 440. Mar. 2016, pp. 523–531.

[15] National Aeronautics and Space Administration (NASA). *Gimballed Thrust*. Visited: 19-05-2021. URL: https://www.grc.nasa.gov/www/k-12/rocket/gimbaled.html.

[16] URL: https://www.rocketlabusa.com.

[17] Hunini. *JASDF AAM-5 Kai TVC behn right rear view at Gifu Air Base November 19, 2017*. 2017. URL: https://upload.wikimedia.org/wikipedia/commons/0/0e/JASDF_AAM-5_Kai_TVC_behn_right_rear_view_at_Gifu_Air_Base_November_19%2C_2017.jpg.

[18] *KST DS213 DAVIGA*. URL: https://wiki.rc-network.de/wiki/KST_DS213_DAVIGA.

[19] Emil Bjerregaard Jacobsen. *Position Estimation of Unmanned Aerial Vehicles in GPS denied locations*. 9th semester internship report. 2021.

[20] D. A. Mercado et al. "GPS/INS/optic flow data fusion for position and Velocity estimation". In: (2013), pp. 486–491.

[21] *KISS ESC 3-6S 32A*. Visited: 05-04-2021. URL: https://www.flyduino.net/en_US/shop/product/pr2200-kiss-esc-3-6s-32a-45a-limit-32bit-brushless-motor-ctrl-2961.

[22] *Pololu 5V, 2.5A Step-Down Voltage Regulator D24V22F5*. Visited: 05-04-2021. URL: https://www.pololu.com/product/2858.

[23] Kimon P. Valavanis and George J. Vachtsevanos. *Handbook of Unmanned Aerial Vehicles*. Springer Publishing Company, Incorporated, 2014. ISBN: 9048197066.

[24] E.L. Houghton et al. *Aerodynamics for Engineering Students*. Seventh Edition. 2017. ISBN: 978-0-08-100194-3. URL: https://www.sciencedirect.com/science/article/pii/B9780081001943000018.

[25] National Aeronautics and Space Administration (NASA). *Momentum effects on Aerodynamic Forces*. Visited: 05-04-2021. URL: https://www.grc.nasa.gov/www/k-12/airplane/momntm.html.

[26] William L. Brogan. *Modern Control Theory (3rd Ed.)* Prentice-Hall, Inc., 1991. ISBN: 0135897637.

[27] M.S. Triantafyllou and F.S. Hover. *Maneuvering and Control of Marine Vehicles*. Massachusetts of Institute of Technology.

[28] João P. Hespanha. *Linear Systems Theory: Second Edition*. Princeton University Press, 2018. ISBN: 9780691179575.

[29] Ross A. Knepper et al. *Foundations of Robotics Course Notes*. Cornell University, 2019. URL: https://rpal.cs.cornell.edu/foundations/.

[30] *Kalman Filtering: Theory and Practice Using MATLAB*. John Wiley & Sons, Ltd, 2008. ISBN: 9780470377819.

[31] Nicholas Assimakis, Emmanouil Psarakis, and Demetrios Lainiotis. "Steady State Kalman Filter: A New Approach." In: *Neural Parallel & Scientific Comp.* 11 (Jan. 2003), pp. 485–490.

[32] *Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions*. May 2021. URL: https://www.freertos.org/.

[33] *Basic Linear Algebra*. URL: https://github.com/tomstewart89/BasicLinearAlgebra.

[34] Blake West. *DSHOT on Mbed*. Visited: 25-05-2021. URL: https://os.mbed.com/users/bwest32/notebook/dshot.

[35] Airfoiltools.com. *Joukowski 9% Symmetrical Airfoil*. URL: http://airfoiltools.com/airfoil/details?airfoil=joukowsk0009-jf.

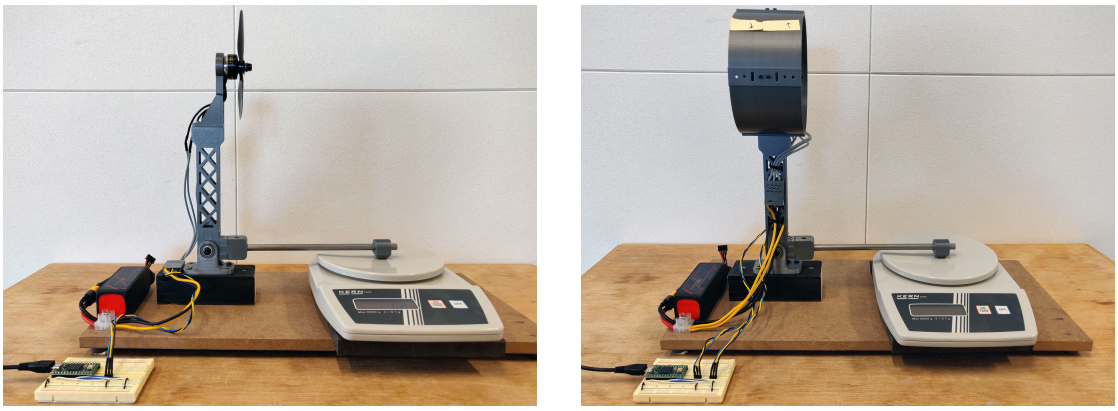
APPENDIX A

Motor Thrust Tests

This appendix describes a series of test performed to fidn the correlation between motor thrust and RPM (thrust curve). This is useful, to compute the steady-state motor velocity, needed to counter-act the gravity in the hover point. Furthermore, by analysing the response of a single step on the motor, a model describing the motor dynamics can be approximated.

A.1 Test setup

To measure the thrust F_t generated by the motors, the principle of moment arm is utilised to convert the motor force F_t to a equal buts 90° rotated force. In order to measure the generated thrust a custom setup has been produced, see **Figure A.1**. The thrust is manually measured, while the motor RPM, current and voltage is measured by requesting telemetry from the ESCs.



(a) Test of a single motor in free air

(b) Test of single and dual motor in duct

Figure A.1: Comparison between the two test setup used.

Using this setup a series of tests is performed.

A.2 Results

To generate a thrust curve, the motor velocity is increased in steps and the throttle is measured. Four tests in total is made, one for each type of propulsion system, see **Figure A.2**.

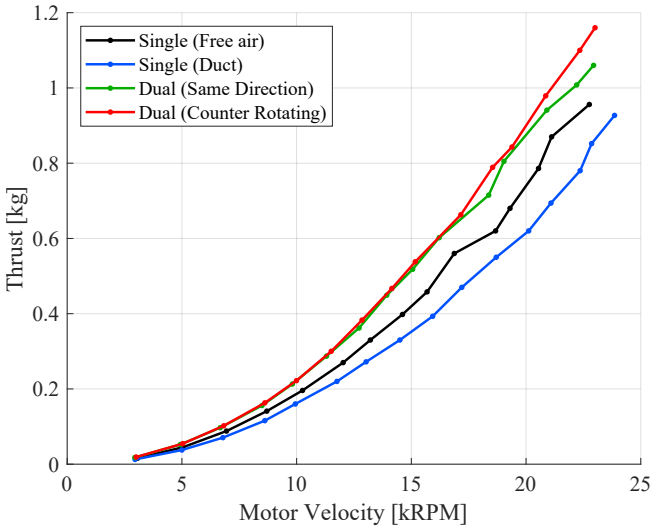


Figure A.2: The results of the four separate motor tests. All four setup performs similarly, however, the counter-rotating setup yields the most thrust.

To approximate a dynamic model of the motor, a step is applied to the motor ESC, and the output is monitored, see **Figure A.3**.

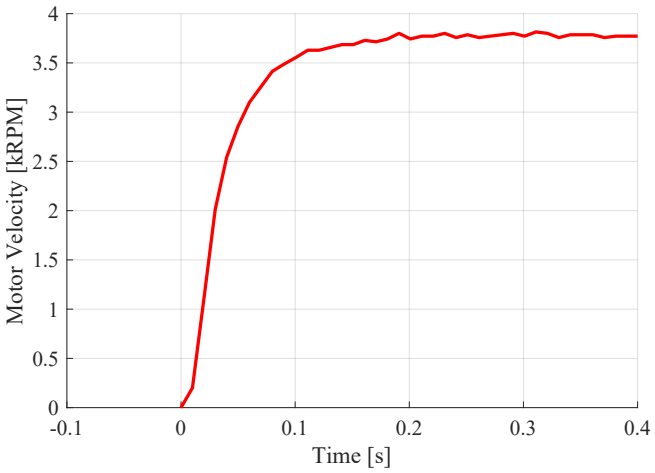


Figure A.3: The step-response of the motor velocity, with a DSHOT step of 200.

A.3 Parameter Estimation

As described in **Chapter 3**, the motor thrust can be described as a second order polynomial of the motor velocity.

$$F_t(\omega) = K_f \cdot \omega_t^2 \tag{A.1}$$

From the motor test the same is observed. Using the measurements acquired from motor tests, a second order polynomial fit is made using the MATLAB function `polyfix()`, which performs a regular polynomial fit, while forcing the fit specific points (0,0).

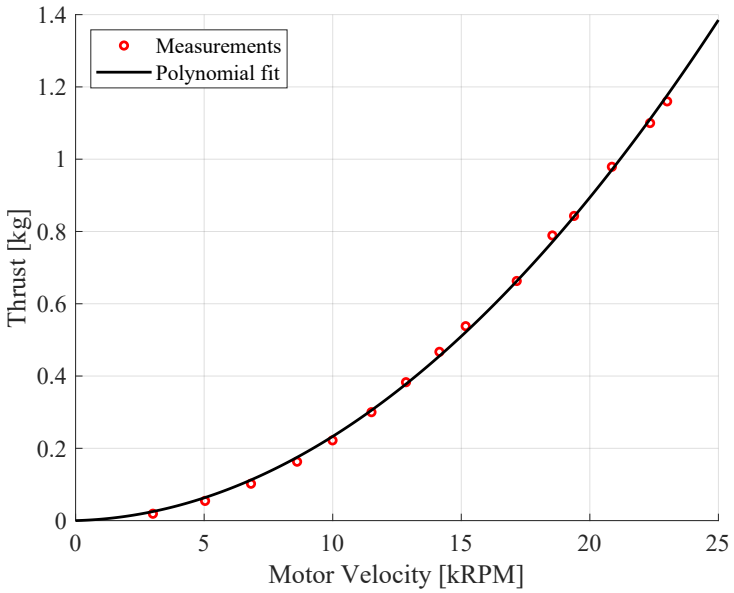


Figure A.4: Polynomial fit of motor thrust, which is forced through origin. The x-axis unit is kilo RPM (a thousand RPM).

The polynomial fit, yields the thrust coefficient $K_f = 0.002140127$.

A.4 System Identification

Using MATLABs "System Identification Toolbox", the step-response, see **Figure A.5** is fitted to match a first order system.

$$\frac{\omega_t(s)}{u_t(s)} = \frac{K_t}{\tau_t s + 1} \tag{A.2}$$

With the DC gain $K_t = 19.89 \cdot 10^{-3}$ and the time-constant τ_t =0,0345 s.

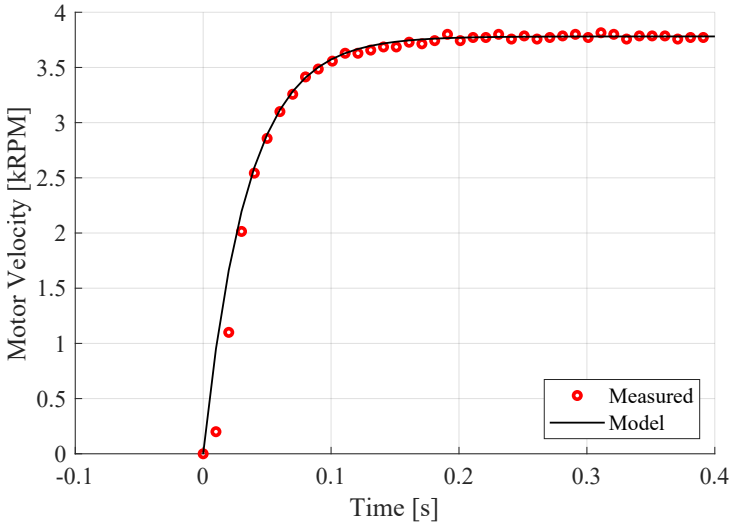


Figure A.5: System Identification applied on the motor velocity step-response.

The DC gain effectively transfers DSHOT throttle to a corresponding RPM value, while the time-constant describes the dynamics of the motor.

APPENDIX B

Thrust Vane Design

To describe the amount of thrust reflected by the thrust vanes, it is important to know the lift and drag coefficients. These coefficients cannot be calculated theoretically and must instead be obtained experimentally using a wind-tunnel or by running aerodynamic simulations.

B.1 Shape

To reduce the time-consuming task of achiveing these coefficients for a custom thrust vane, a preexisting airfoil design has been chosen as the shape of the thrust vanes. The specific airfoil chosen is a symmetrical airfoil known as the Joukovsky airfoil, see **Figure B.1**.

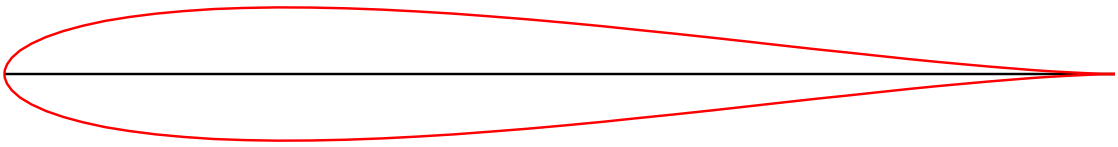


Figure B.1: Joukovsky airfoil shape [35]

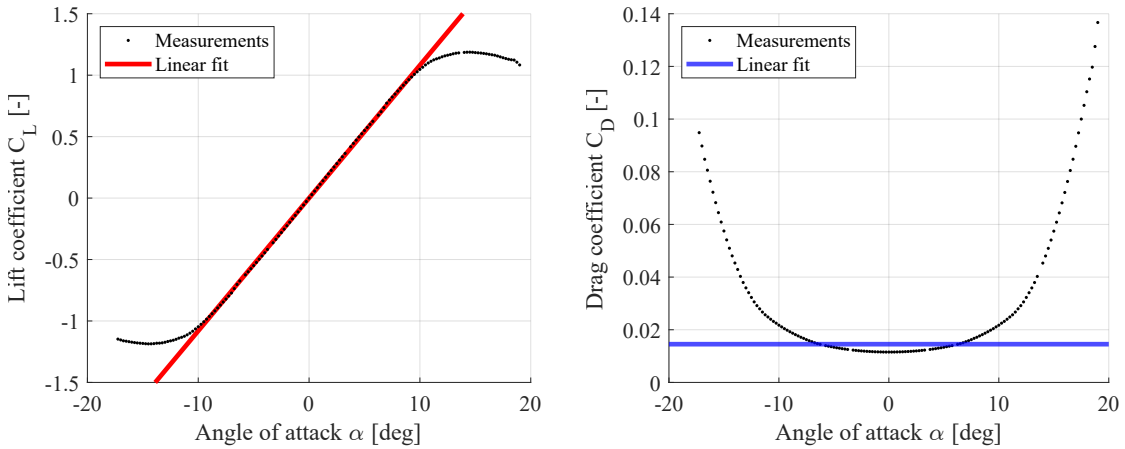
The shape and characteristics for the Joukovsky shape is freely available on the internet. The freely available data includes lift- and drag-coefficients measured at different angles of attack (α). These measurements will be applied to find an approximate model for the lift and drag coefficients that can be applied in a linear model.

B.2 Characteristics

The coefficients of lift and drag, for varying angle of attack are found using the iterative aerodynamic simulation tool called XFoil [35]. The data is made freely available by Airfoiltools.com [35].

B.3 Parameter estimation

From the plots shown in **Figure B.2**, it is clear that for small angles the lift coefficient can be modelled as a linear function and the drag coefficient can be modelled as a constant. Using these assumptions, the measurements in the interval 0° to 10° is fitted to the linear model, see **Figure B.2a**. This results in a slope of $C_{L\alpha} = 0.1081$ for the lift coefficient, and a constant of $C_{D0} = 0.0145$ for the drag coefficient.



(a) Linear fit of lift coefficient

(b) Linear fit of drag coefficient

Figure B.2: Linear models for both lift and drag coefficient

B.4 Implementation

The shape of the Joukovsky airfoil is accessible as a set of coordinates or as a vector-file. Using the vector-file the shape is imported into 3D-CAD modelling software, where it can be used to extrude the shape, see **Figure 3.5**.

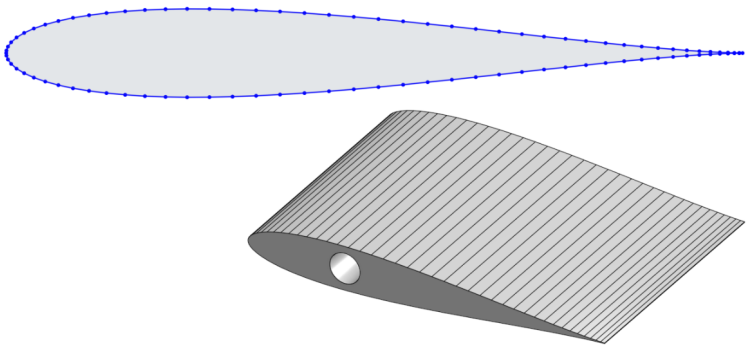


Figure B.3: Import of airfoil shape into CAD software

APPENDIX C

Nonlinear Simulation

In order to use the derived system model, a simulation build in MATLAB Simulink is constructed, see **Figure C.1**. The simulation is available at the Github repository, as the rest of the software, see **Chapter 6**.

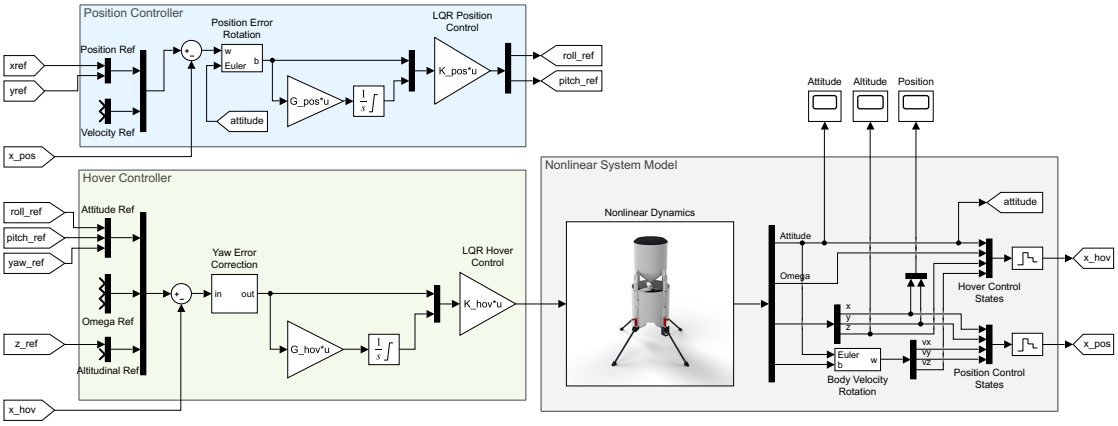


Figure C.1: Overview of the MATLAB Simulink simulation, divided into three sections; nonlinear system model, hover controller and position controller.

The simulation uses the *S-Function* block, to call the nonlinear differential equations describing the dynamics of the mono-copter. This includes all the rotational and translational dynamics, motor model and the nonlinear lift- and drag-equations. The rest of the simulation is done purely in Simulink, however with the gain-matrices being computed beforehand in a separate matlab-script.

APPENDIX D

Schematics

D.1 Carrier board

The *carrier board* houses the main micro-processor and the WiFi-module, and all the connectors needed to interface with the sensors and actuators, see **Figure D.1**.

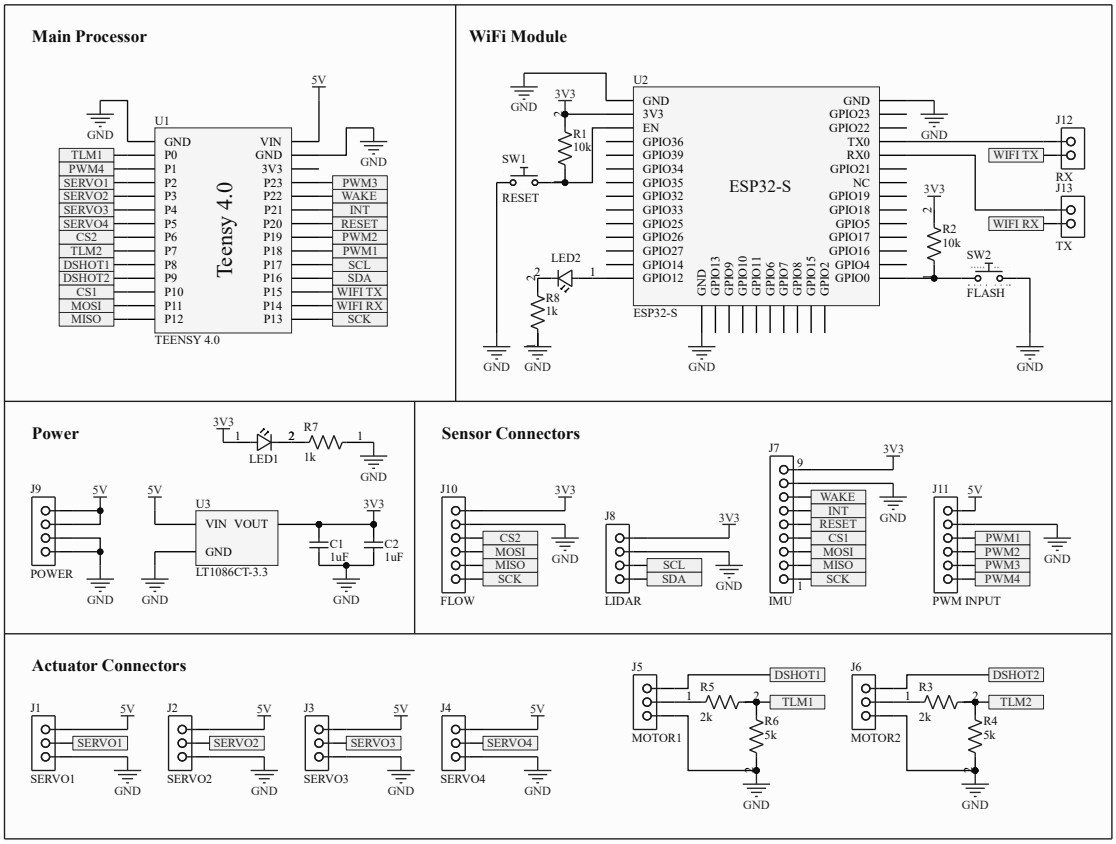


Figure D.1: Schematic overview of the carrier board, used for the flight controller.