Attitude and Position Control of a Quadcopter in a Networked Distributed System

1st Semester Master's Program in Control and Automation
Department of Electronic Systems
Aalborg University

Alejandro Alonso García, Amalie Vistoft Petersen, Andrea Victoria Tram Løvemærke,
Niels Skov Vestergaard and Noelia Villarmarzo Arruñada
Group 733

This report is compiled in LaTeX, originally developed by Leslie Lamport, based on Donald Knuth's TeX. The main text is written in *Latin Modern* pt 12, designed by Bogusław Jackowski and Janusz M. Nowacki. Diagrams are made using Inkscape and Tikz.

AALBORG UNIVERSITY
STUDENT REPORT

1st Semester Project

Master's Program in

Control and Automation

Department of Electronic Systems

Fredrik Bajers Vej 7C, 9220 Aalborg

Title:
Attitude and Position Control of a Quadcopter in a Networked Distributed System

Theme:
Networked Control Systems

Project Period:
Fall 2016

Project Group:
733

Participants:
Alejandro Alonso García
Amalie Vistoft Petersen
Andrea Victoria Tram Løvemærke
Niels Skov Vestergaard
Noelia Villarmarzo Arruñada

Supervisor:
Anders la Cour-Harbo

Pages: 100
Appendices: 6
Attachments: 1

Concluded: 20/12/2016

Synopsis

Quadcopters are becoming increasingly interesting due to the great variety of usage. A design that is able to make the quadcopter hover and move to a desired position is presented. The system's coupled behavior and instability raises a challenging control task. This task is solved by implementing a linear control design, which is based upon a model derived by first principles modeling. The control system is divided into attitude and translational. These are designed using state space and classical control methods, respectively. The quadcopter gets its attitude and position from an external motion tracking system based on infrared cameras, while the control remains in the microcontroller on the quadcopter. This layout constitutes a distributed system, where network issues occur. These are considered in the control design to ensure the stability.

Preface

This first semester project of the Control and Automation Master's Program at Aalborg University has been carried out in the fall 2016 by the student group 16gr733. This project has been supervised by Anders la Cour-Harbo, associate professor at Aalborg University.

The focus of this project has been to design an attitude and position control for a quadcopter in a networked distributed system.

It is expected of the reader to have an engineering level knowledge within mathematics and physics as well as electronics and linear control theory.

For referencing sources the standard ISO 690 is used, noted as [n], where n represents a number in the bibliography. The bibliography is placed at the end of the worksheets, before the appendices. Additional attachments are uploaded along with the worksheets, these are code-files, MATLAB simulation scrips, sources and the documents submitted to SEMCON.

Special thanks to Henrik Schiøler, associated professor at Aalborg University, for providing the quadcopter, his assistance in the laboratory and kind sharing of experience. Thanks as well to Jesper D. Pedersen, engineering worker at Aalborg University, for building test setups for the quadcopter.

Text by:

Alejandro Alonso García

Amalie Vistoft Petersen

Andrea Victoria Tram Løvemærke

Niels Skov Vestergaard

Noelia Villarmarzo Arruñada

Contents

Part I

Preliminary Analysis

1 | Introduction

Drones have a wide variety of potential uses, such as search and rescue, inspection, security, surveillance, research, aerial photography, unmanned cargo systems, military applications, etc. Drones are already widely used, and while some of the aforementioned applications raises ethical issues for debate, there is no doubt that drones also hold a place in the future.

Multicopters constitutes a class of drones with fixed-pitch propellers. This means that the actuation is achieved solely from a difference in speed between the propellers. Among all the different varieties of multicopters, see some examples in Figure 1.1, the quadcopter is the most popular, largely due to a compromise between stabilization capabilities and cost of hardware. [1]

Figure 1.1: A small selection of drones which belongs to the large class of multicopters. [2]

The aim of the project is to investigate the capabilities of a specific design strategy when controlling a quadcopter as part of a distributed system.

The control of a quadcopter has been addressed many times in the recent years. One example is where the quadcopter is controlled using a back-stepping technique and non-linear controllers [3]. Another way of solving the control problem is where the quadcopter attitude is modeled using quaternions and controlled with a PD based non-linear controller [4]. There is a multitude of possible solutions, and while non-linear control is a popular choice for quadcopters, this project addresses the possibility of using linear controllers and Euler angle representation.

The quadcopter obtains its attitude and position over a network from an external motion tracking system. This introduces delay and packet loss as a limiting factor to the control system situated on the quadcopter. The controlling code is implemented in C on a microcontroller using a real time operating system (RTOS), called FreeRTOS.

The system's coupled behavior and instability raises a challenging control task. This task is solved by implementing a controller design, which is based on a model derived by first principle modeling. This is later linearized since it is desired to use linear controllers.

The system is divided into an attitude controller and translational controllers. The attitude control strategy is based in state space design, and makes use of state feedback and integral control using LQR. The translational controllers are designed with classical control methods.

This project provides design and realization of the described system. Subjects in focus are the performance achievable using the chosen linear attitude control strategy, the influence of remote sensing, the influence of the attitude control bandwidth on the translational controllers and the performance of a cascaded control structure with classical linear translational controllers.

2 | System Description

In this project, a prototype of a quadcopter is provided with motors, motor controllers, propellers and a battery. The system in its initial state is seen in Figure 2.1. Apart from the quadcopter and the attached hardware, additional hardware must be selected to make the quadcopter able to fly. First the given hardware is described in detail, hereafter the selected hardware is presented. Last, a description of the prototype is provided.

2.1 Provided Hardware

The provided hardware is described in this section. This is to give an overview of the initial configuration of the system, as it influences the design leading to the final prototype. The provided quadcopter including additional hardware is seen in Figure 2.1.

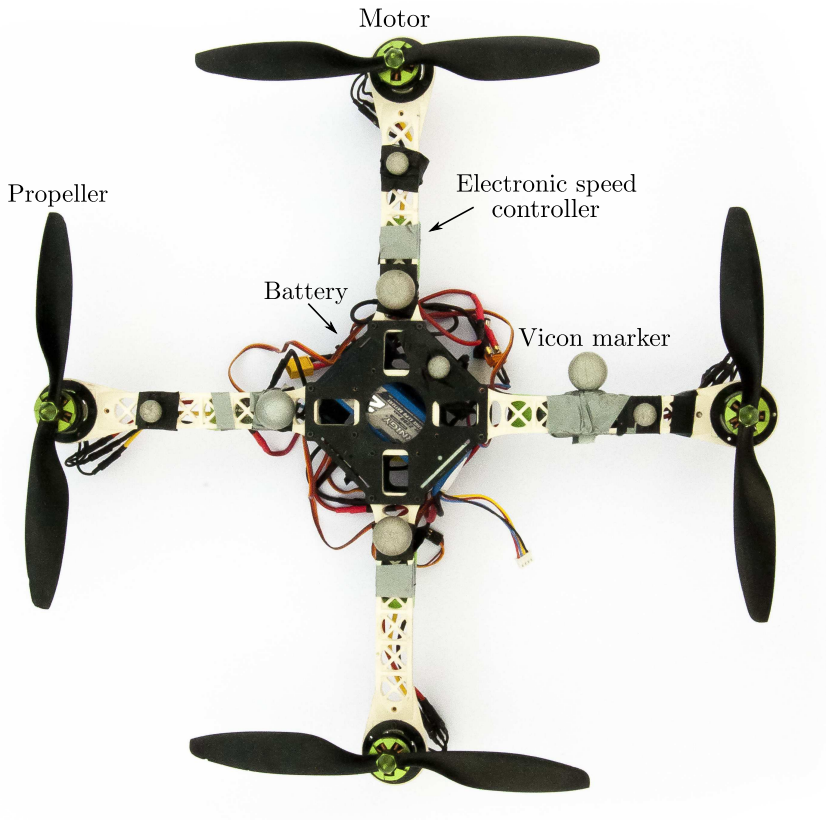

Figure 2.1: The quadcopter with the given hardware.

The provided quadcopter framework measures 45 cm from rotor to rotor and is an X-shaped symmetric quadcopter, with a height of 7.5 cm. With the provided and selected hardware the total mass of the quadcopter is 996 g.

2.1.1 Motors

The quadcopter is equipped with four brushless DC outrunner motors named Turnigy Multistar. Contrary to brushed DC motors these do not rely on a brush which is susceptible to mechanical wear. Outrunner motors have their magnets (rotor) in the outer shell. The rotor spins around the fixed coils that form the stator. Outrunner motors generally have more space for magnets compared to inrunner motors, which is why outrunner motors typically have more poles, and thus, are able to produce more torque for the same physical dimensions.

The motor has a motor velocity constant, k_v, of 935 RPM V^{-1}, has 14 poles and a maximum current of 15 A [5].
The motor used is shown in Figure 2.2.

Figure 2.2: One of the four motors mounted on the quadcopter.[5]

2.1.2 Motor Controllers

The quadcopter comes with four electronic speed controllers, ESCs, one for each motor. The ESCs produce an 8 kHz PWM, are rated for a 2-4 cell Li-Po battery and can handle a constant current of up to 30 A. The ESCs have also a 5.5 V, 4 A output for powering e.g. a controller board. The battery level is only measured at start up, which can be problematic, as the provided power will decrease as the battery level drops over time. [6]

Figure 2.3: One of the four Electronic Speed Controllers mounted on the quadcopter.[6]

2.1.3 Propellers

The propellers have a length of 25.4 cm and a pitch of 11.43 cm. The pitch is the distance traveled by a propeller after one complete rotation. For this to be true the propeller must turn inside a solid surface, the traveled distance is less if it rotates inside a liquid or gas [7]. A higher pitch creates more turbulence and make the quadcopter less steady when flying [8].

A longer propeller has an increase in the quadcopter's speed. The acceleration of a small propeller is greater than that of a larger propeller. Thus it is easier to change the moment of inertia if utilizing a small propeller compared to a larger one. [8]
In this case the given propellers are suitable for the utilized quadcopter, as they have been utilized in multiple cases before.

A picture of the utilized propellers can be seen in Figure 2.4.

Figure 2.4: Two of the four propellers mounted on the quadcopter's motors.[5]

2.1.4 Battery

The battery available for the prototype, is a Turnigy battery. It has a capacity of 2200 mAh, a voltage of 11.1 V and a discharge rate of 20-30 C.[9]

Figure 2.5: The battery mounted on the quadcopter.[10]

2.2 Selected Hardware

The provided quadcopter is not capable of flying as it is. Therefore additional hardware must be implemented. This includes sensors, as the quadcopter will otherwise not be able to navigate. A processor, as a computation device is needed to handle the communication and the control algorithms. And lastly a wireless module, as communication with the quadcopter is required. The chosen sensor solution is now presented.

2.2.1 Sensor

It is possible to implement on board sensors or use a real time attitude and position data capturing system such as Vicon, which is available at Aalborg University in a room of the dimensions: 5.85 m $\times$ 5.72 m $\times$ 3.72 m . As the focus of this project is to design and implement a control system in a distributed network, it is chosen to use the Vicon system. This challenges the control system by introducing network delays and packet loss. As this project develops a prototype, it is not essential that the quadcopter can operate outside of the Vicon system.

The Vicon system provides real-time position and attitude captured with nine infrared cameras.

To use this system, reflective markers are attached to the tracked object. The Vicon system streams the position of the markers and the position and attitude of the object at 100 Hz for a computer to read them[11]. The data can be received by using an SDK (software development kit) plug-in for MATLAB. In this way, data can be received by MATLAB, making it possible to directly perform calculations, which makes it easier to obtain variable derivatives.

The user interface of the Vicon system is shown in Figure 2.6 and an illustration of the Vicon system setup is seen in Figure 2.7.

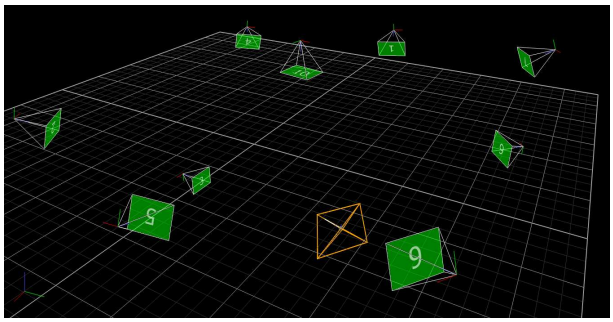

Figure 2.6: User interface of the Vicon System, the Vicon Tracker. An object has been created from the markers placed on the quadcopter..

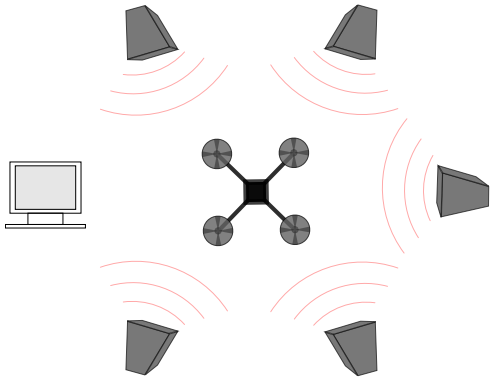

Figure 2.7: An illustration showing the Vicon system's cameras sensing the quadcopter and the computer used to get the data.

The software is called Vicon Tracker and it allows the creation of objects by grouping markers present in the room. It also allows to change the center of gravity of the created objects and rotate the inertial and body reference frames to any desired orientation.

2.2.2 Processor

As the computer on the ground only handles the Vicon sensor data and the communication of this data, the processor of the quadcopter must be capable of handling the processing of the control while also receiving the attitude and position of the quadcopter. The implementation of the controllers is done in a microprocessor from Atmel, ATmega2560, mounted on the ArduinoMEGA development board, as seen in Figure 2.8. This processor has up to 16 PWM output channels, two 8-bit timers and four 16-bit timers, four USART modules, 8 kB of internal memory and a processing speed of 16M IPS. These capabilities are presumed to be sufficient for the control implementation on the quadcopter.[12]

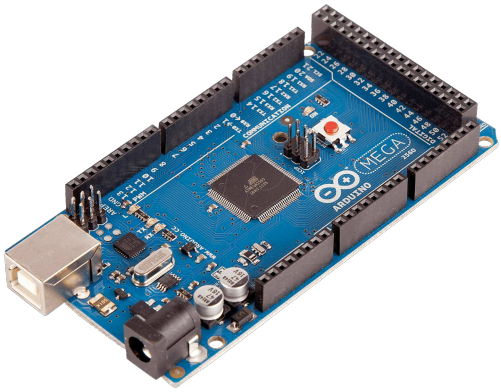

Figure 2.8: AtMega2560 mounted in the ArduinoMEGA development board.[13]

2.2.3 Wireless Modules

The wireless modules used to communicate with the quadcopter are called XBee, see Figure 2.9. These modules need a supply voltage between 2.8 V and 3.4 V, which fits well with the 3.3 V supply provided by the Arduino board[14].

Figure 2.9: XBee wireless modules.[15]

The XBee modules communicate at a frequency of 2.4 GHz and has a range of up to 30 m in indoor environments and a line-of-sight range of up to 90 m. The modules transmit with an RF data rate of 250 Kbps and are capable of communicating with a serial interface data rate of from 1200 bps up to 250 Kbps. The XBee modules take care of all communication layers except the transport layer protocol. The protocol used by the XBee modules is called ZigBee, which follows the IEEE 802.15.4 standard.[14]

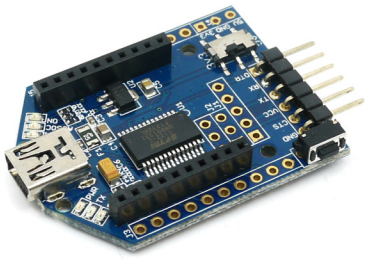

Figure 2.10: UartSBee v4.0 USB to serial adapter module. [16]

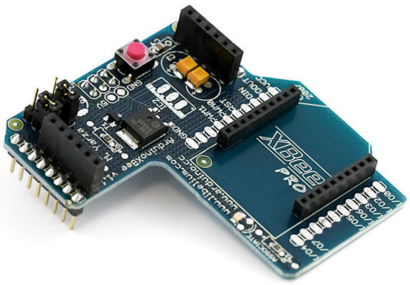

Figure 2.11: XBee Shield. [17]

In order to use these modules, the packet has to be sent through the serial port of the computer to the XBee, through RF to the other XBee and back through serial pins to the microcontroller. In order to interface with the XBee modules, serial connections are required on both ends, and the XBee modules must be powered. For the computer a USB to serial adapter board, called UartSBee v4.0, is used, this module also powers the XBee through the USB [18]. The microcontroller uses an XBee shield to connect the power and serial pins of the Arduino[19].

2.3 Prototype Description

In the following section an overall description of the prototype is given. The prototype is divided into three separate subsystems. The quadcopter, the ground station, and the Vicon system. These can be seen in Figure 2.12.

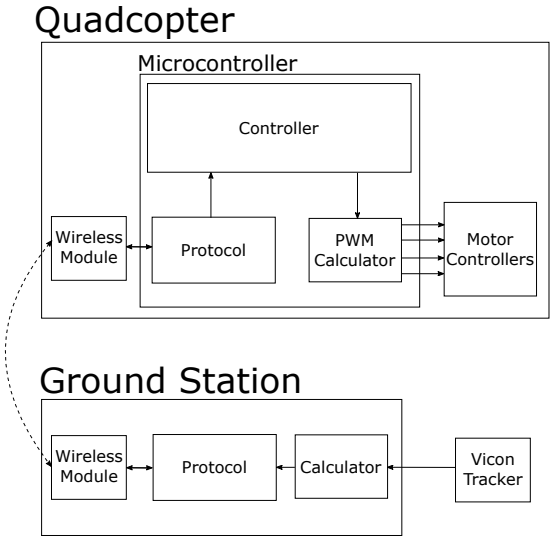

Figure 2.12: Schematic diagram of the prototype, which includes the quadcopter, the ground station and the Vicon system.

The Vicon system captures the quadcopter's position, in form of coordinates, and its attitude, in Euler angles. This information is then transmitted to the Ground Station. It is desired to move the quadcopter in the x, y and z directions. The reference commands are therefore transmitted together with the position, attitude and calculated translational velocities to the controller placed on the quadcopter.

The information is transmitted by means of a network interface through the XBees. The function protocol is implemented in the Ground Station to be able to generate a packet which can be transmitted and decoded on the microcontroller. It is also needed for the microcontroller to detect packets and disregard incorrect packets.

The quadcopter consists of a microcontroller, four motor controllers, four motors, a battery, and a wireless module. The microcontroller's tasks are to handle the network between itself and the ground station, the control calculations and to generate the control signals which should be applied to the four motor controllers.

To be able to calculate the rotational speed needed for the motors, the variables received from the ground station are needed, i.e the control reference, position, attitude and translational velocities of the quadcopter. Hereafter it is possible to calculate the necessary PWM signals, by utilizing the rotational speed calculated by the control system and the relationship between the rotational speed and the PWM signal, found in Appendix C.

3 | Functional Requirements

Based on the prototype description in section 2.3, functional requirements can be established for the prototype.

1) **The quadcopter should be able to receive its own position and attitude from the Vicon system, through a computer at the ground station.**

   This is essential for controlling and navigating the quadcopter. Additionally, it should be possible for the quadcopter to receive the information through a wireless channel and discard corrupted packets.

2) **It shall be possible to control the quadcopter's attitude.**

   To be able to stabilize the quadcopter, it is essential that it is possible to control the attitude. This is essential if the quadcopter is changing position in either the x, y, or z axes or when it needs to maintain its position.

3) **It shall be possible to control the quadcopter's position in the z axis.**

   It should be possible for the quadcopter to keep itself in a fixed z position and be able to move in the z axis.

4) **It shall be possible to control the quadcopter's position in the x and y axes.**

   It should be possible for the quadcopter to keep itself in a fixed x,y position and be able to move in these two axes.

4 | Model

A description of the physical behavior of the quadcopter is necessary in order to proceed with the controller design. From the obtained model, the control system can be designed such that the functional requirements are fulfilled, see chapter 3.

In this chapter, an overview is first presented. Then, the model is derived and a linear approximation is performed. Finally, these two are compared in simulation to ensure that the linear approximation yields acceptable results.

4.1 Model Overview

The model can be split into two submodels, one describing the angular behaviour and the other being a translational model.

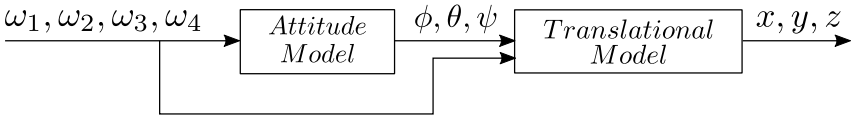

Figure 4.1: Overview of how the two models are related.

Figure 4.1 shows the relation between the two submodels. The attitude model, which describes the angular behavior, takes the velocities of the four motors as input and provides the angles roll, pitch and yaw as output. The translational model takes the angles as input, which also takes the rotational speeds of the motors as input. The output of the translational model is a position given in x, y and z coordinates.

When flying a quadcopter two different configurations are generally considered, the cross and the plus configurations, see Figure 4.2 and 4.3. The angles of rotation, roll, pitch and yaw, are generally defined around the three coordinate axes attached to the quadcopter. [20]

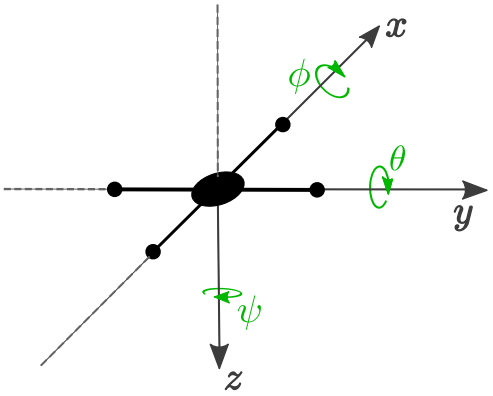

Figure 4.2: The quadcopter with orientation in the coordinate system corresponding to plus configuration.

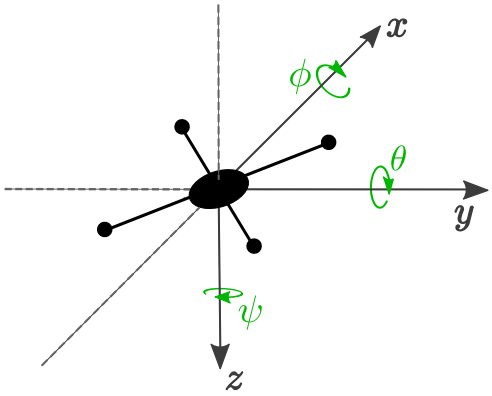

Figure 4.3: The quadcopter with orientation in the coordinate system corresponding to cross configuration.

If the quadcopter is in plus configuration and i.e. the roll angle is increased, it will fly with one rotor in front. However if the same angle is applied to a quadcopter in cross configuration, it will fly with two rotors in front. [20]

Cross configuration provides a more responsive flight since all four rotors are engaged when changing an angle. Less actuation is also required from each motor since there are two to pull the quadcopter around. From a control perspective it is however more forthright to model and control a quadcopter in plus configuration, which is the choice for this prototype. [20]

An aspect to be mentioned before modeling the quadcopter is the usage of two coordinate frames. A body frame, which is fixed to the flying object, and an inertial frame, which is fixed to the Vicon room.

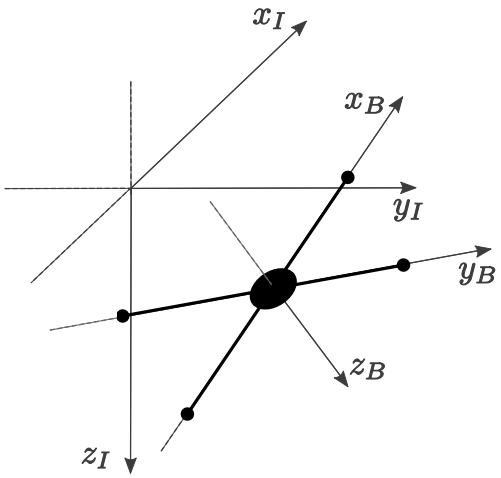

Figure 4.4: The quadcopter with its body frame (B-index) placed in the intial frame (I-index) of the Vicon room.

Figure 4.4 shows the two frames. The inertial frame is oriented in such a way that its z-axis is pointing downwards, where altitude is given in negative numbers and ground is zero.

The attitude of the quadcopter is obtained by knowing the body frame orientation with respect to the inertial frame, yielding the roll, pitch and yaw angles.

The transformation from the body frame to the inertial can be done through a rotation matrix, Equation 4.1, which describes a total rotation in terms of three consecutive rotations.

In this case the rotation matrix is composed with a 1-2-3 convention, that is, first a rotation around x_{B}, then around y_{B} and finally around z_{B}. [21]

$$\mathrm{R_X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \qquad \mathrm{R_Y} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \qquad \mathrm{R_Z} = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_Z R_Y R_X = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ s\phi c\theta & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{4.1}$$

Where:

R_X	is the matrix describing a rotation around the x_{B} axis	
R_Y	is the matrix describing a rotation around the y_{B} axis	
R_Z	is the matrix describing a rotation around the z_{B} axis	
R	is the total rotation matrix	
ϕ	is the roll angle	[rad]
θ	is the pitch angle	[rad]
ψ	is the yaw angle	[rad]

Note that due to the size of the matrix sine and cosine are denoted s and c respectively.

To describe a vector in the inertial frame given its description in the body frame, a matrix-vector multiplication can be done as follows:

$$v_{\mathrm{I}} = Rv_{\mathrm{B}} \tag{4.2}$$

Where:

v_{I}	is a column vector that contains the description with respect to the inertial frame
v_{B}	is a column vector that contains the description with respect to the body frame

Once these considerations have been taken into account, the two submodels can be derived.

4.2 Attitude Model

The attitude model of the quadcopter describes how the roll, pitch and yaw angles evolve according to the forces and torques exerted by the propellers.

The free body diagrams, see Figure 4.5 and 4.6, are the basis for deriving the attitude model.

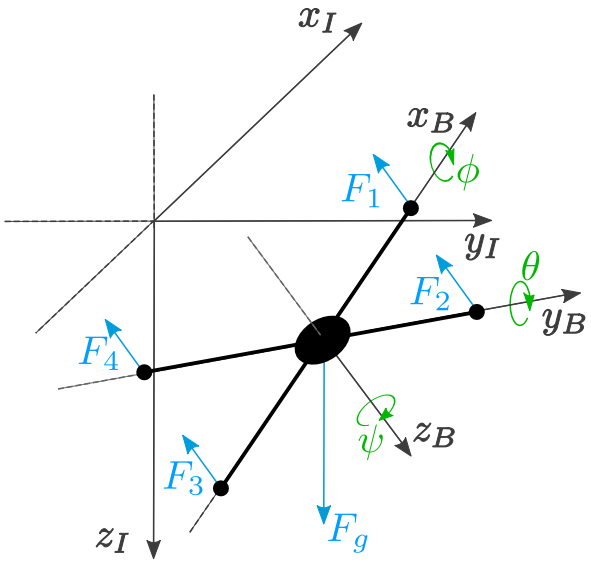

Figure 4.5: Free body diagram that holds both the inertial and body reference systems, as well as the references for the angles, roll, pitch and yaw, along with the thrust forces and the gravitational force.

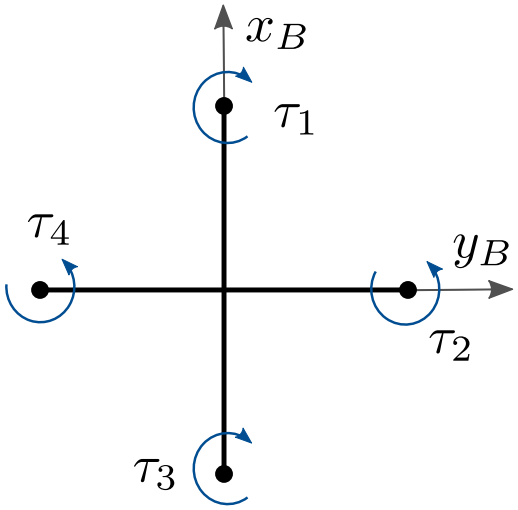

Figure 4.6: Free body diagram with the references for the torques produced by the drag forces at the propellers.

In Figure 4.5 a free body diagram of the quadcopter is illustrated. This diagram is an extended diagram of Figure 4.4 shown in section 4.1. Each of the thrust forces exerted by the motor and propeller is included, F_1, F_2, F_3 and F_4. Besides these four forces, the gravitational force is included as well. In Figure 4.6 the quadcopter is seen from above. Here the four torques produced by the drag forces at the propellers are illustrated. As seen in the figure, τ_1 and τ_3 are opposing τ_2 and τ_4. The motor pairs turn in opposite directions to compensate the drag torques.

From the diagrams above, the equations of angular motion around the three body axes can be derived by first principles modeling, that is, using only laws of physics. In this case, Newton's Second Law for rotational motion.

$$J\alpha = \sum \tau \tag{4.3}$$

Where:

J	is the moment of inertia	[kg m^2]
α	is the angular acceleration	[rad s^{-2}]
τ	are the torques applied to the system	[N m]

From applying Newton's Second Law on all three axes in the body-frame, Equation 4.4, 4.5 and 4.6 are obtained. As it is seen, the roll and pitch angular accelerations depend on the thrust forces exerted by the propellers placed along the y_B and x_B axes of the quadcopter, respectively. Hence the roll angular acceleration, $\ddot{\phi}$, depends on the difference in thrust force exerted by propeller four and two. Similar the pitch angular acceleration, $\ddot{\theta}$, depends on the difference in thrust force exerted by propeller one and three. The yaw angular acceleration, $\ddot{\psi}$, changes due to the torques created by the drag forces in the propellers. As the torques, τ_1 and τ_3, see Figure 4.6, are pointing in the opposite direction of τ_2 and τ_4, the latter is subtracted from the former yielding the total torque, $J_z\ddot{\psi}$.

$$J_x\ddot{\phi} = (F_4 - F_2)L \tag{4.4}$$
$$J_y\ddot{\theta} = (F_1 - F_3)L \tag{4.5}$$
$$J_z\ddot{\psi} = \tau_1 - \tau_2 + \tau_3 - \tau_4 \tag{4.6}$$

Where:

J_x	is the inertia around x_B	[kg m^2]
J_y	is the inertia around y_B	[kg m^2]
J_z	is the inertia around z_B	[kg m^2]
$\ddot{\phi}$	is the roll angular acceleration	[rad s^{-2}]
$\ddot{\theta}$	is the pitch angular acceleration	[rad s^{-2}]
$\ddot{\psi}$	is the yaw angular acceleration	[rad s^{-2}]
F_i	is the thrust force from each propeller	[N]
L	is the length from center of mass to motor	[m]
τ_i	is the drag torque from each propeller	[N m]

The thrust forces and drag torques from the propeller can be assumed to be proportional to the square of the rotational speed of the motor, related by the coefficients obtained as described in Appendix A and B. The equations above can be rewritten to include these terms.

$$J_x\ddot{\phi} = k_{\text{th}}(\omega_4^2 - \omega_2^2)L \tag{4.7}$$
$$J_y\ddot{\theta} = k_{\text{th}}(\omega_1^2 - \omega_3^2)L \tag{4.8}$$
$$J_z\ddot{\psi} = k_{\text{d}}(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{4.9}$$

Where:

k_{th}	is the thrust coefficient	[N s^2 rad^{-2}]
k_{d}	is the drag coefficient	[N m s^2 rad^{-2}]

Equation 4.7, 4.8 and 4.9 are the final attitude model expressions.

4.3 Translational Model

The aim of this section is to model the translational behavior of the quadcopter. This can be done from Figure 4.5 in section 4.2. An equation for translational accelerations of each specific axis in the inertial frame can be derived by using Newton's Second Law.

$$ma = \sum F \tag{4.10}$$

Where:

m	is the mass	[kg]
a	is the translational acceleration	[m s^{-2}]
F	are the forces applied to the system	[N]

All the forces that are applied to the quadcopter create an acceleration on it. The forces include thrust and gravitational forces.

As seen in Figure 4.5 in section 4.2 the gravitational force has effect only in the z_B direction and the four forces from the motors need to be transformed into the inertial system. This is done through the rotation matrix showed in Equation 4.1.

Taking into account this transformation, the movement can be described using Equation 4.11, 4.12 and 4.13.

$$m\ddot{x}_I = -(F_1 + F_2 + F_3 + F_4)(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi) \tag{4.11}$$
$$m\ddot{y}_I = -(F_1 + F_2 + F_3 + F_4)(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi) \tag{4.12}$$
$$m\ddot{z}_I = F_g - (F_1 + F_2 + F_3 + F_4)\cos\phi \cos\theta \tag{4.13}$$

Where:

$\ddot{x}_I$	is the translational acceleration in the x_I direction	[m s^{-2}]
$\ddot{y}_I$	is the translational acceleration in the y_I direction	[m s^{-2}]
$\ddot{z}_I$	is the translational acceleration in the z_I direction	[m s^{-2}]
F_g	is the gravitational force acting on the quadcopter	[N]

These equations can also be rewritten taking into account that the forces are assumed to be proportional to the square of the speeds of the motors.

This assumption results in Equation 4.14, 4.15 and 4.16.

$$m\ddot{x}_{\mathrm{I}} = -k_{\mathrm{th}}({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \tag{4.14}$$

$$m\ddot{y}_{\mathrm{I}} = -k_{\mathrm{th}}({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \tag{4.15}$$

$$m\ddot{z}_{\mathrm{I}} = F_g - k_{\mathrm{th}}({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)\cos\phi\cos\theta \tag{4.16}$$

Equation 4.14, 4.15 and 4.16 are the final model expressions of the translational model.

Now that both the attitude model and the translational model are derived, it is desirable to linearise the model expressions.

4.4 Linearization

Designing a linear controller requires linearization of the model equations around an equilibrium point. This is done for each equation using the first order Taylor approximation, see Equation 4.17.

$$f(x) \approx f(\overline{x}) + f'(\overline{x})(x - \overline{x}) \rightarrow \ \Delta f(x) \approx f'(\overline{x})\Delta x \tag{4.17}$$

In this equation, $\overline{x}$ represents the linearization point.

To apply the approximation, the function to be approximated must be differentiated with respect to each of the present variables.

$$\begin{aligned} f &= f(x_1, x_2, ..., x_n) \\ \Delta f &= \left.\frac{\partial f}{\partial x_1}\right|_{\overline{x}_1,\overline{x}_2,...,\overline{x}_n} \Delta x_1 + \left.\frac{\partial f}{\partial x_2}\right|_{\overline{x}_1,\overline{x}_2,...,\overline{x}_n} \Delta x_2 + ... + \left.\frac{\partial f}{\partial x_n}\right|_{\overline{x}_1,\overline{x}_2,...,\overline{x_n}} \Delta x_n \end{aligned} \tag{4.18}$$

Once linearized, the function is expressed in terms of variations from the linearization point. This is represented by the symbol Δ in all linearized equations.

In the quadcopter model, the attitude equations, Equation 4.7, 4.8 and 4.9, and the translational equations, Equation 4.11, 4.12, and 4.13, must be linearized since these contain trigonometric functions and squared velocities.

The linearization point is chosen to be where all the angular and translational velocities and accelerations are equal to zero. In Equation 4.14 and 4.15 it is seen that to obtain a zero in x_{I} and y_{I} accelerations, the pitch and roll of the quadcopter should be zero as well. To keep the acceleration in the z_{I} axis in the inertial frame equal to zero, the rotational speeds of the motors, need to be calculated. This can be done from Equation 4.16.

$$m\overline{\ddot{z}}_{\mathrm{I}} = F_g - k_{\mathrm{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\cos\overline{\phi}\cos\overline{\theta} \tag{4.19}$$

Since all $\overline{\ddot{z}_I}$, $\overline{\phi}$ and $\overline{\theta}$ are equal to zero, Equation 4.19 can be rearranged to Equation 4.20.

$$\overline{\omega}_i = \sqrt{\frac{mg}{4k_{th}}} \tag{4.20}$$

Applying the linearization procedure to the attitude model equations around the linearization point, Equation 4.21, 4.22 and 4.23 are obtained.

$$J_x \Delta\ddot{\phi} = 2k_{\text{th}} L\overline{\omega}_4 \Delta\omega_4 - 2k_{\text{th}} L\overline{\omega}_2 \Delta\omega_2 \tag{4.21}$$
$$J_y \Delta\ddot{\theta} = 2k_{\text{th}} L\overline{\omega}_1 \Delta\omega_1 - 2k_{\text{th}} L\overline{\omega}_3 \Delta\omega_3 \tag{4.22}$$
$$J_z \Delta\ddot{\psi} = 2k_{\text{d}}\overline{\omega}_1 \Delta\omega_1 - 2k_{\text{d}}\overline{\omega}_2 \Delta\omega_2 + 2k_{\text{d}}\overline{\omega}_3 \Delta\omega_3 - 2k_{\text{d}}\overline{\omega}_4 \Delta\omega_4 \tag{4.23}$$

Where:

$\Delta\ddot{\phi}$	is the change in roll angular acceleration from equilibrium	[rad s^{-2}]
$\Delta\ddot{\theta}$	is the change in pitch angular acceleration from equilibrium	[rad s^{-2}]
$\Delta\ddot{\psi}$	is the change in yaw angular acceleration from equilibrium	[rad s^{-2}]
$\overline{\omega}_i$	is the angular velocity of each motor in equilibrium	[rad s^{-1}]
$\Delta\omega_i$	is the change in angular velocity from equilibrium of each motor	[rad s^{-1}]

In a similar way as with the attitude equations, the translational model can be linearized, leading to Equation 4.24, 4.25 and 4.26.

$$m\Delta\ddot{x}_{\text{I}} = -k_{\text{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\Delta\theta \tag{4.24}$$
$$m\Delta\ddot{y}_{\text{I}} = k_{\text{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\Delta\phi \tag{4.25}$$
$$m\Delta\ddot{z}_{\text{I}} = -2k_{\text{th}}\overline{\omega}_1 \Delta\omega_1 - 2k_{\text{th}}\overline{\omega}_2 \Delta\omega_2 - 2k_{\text{th}}\overline{\omega}_3 \Delta\omega_3 - 2k_{\text{th}}\overline{\omega}_4 \Delta\omega_4 \tag{4.26}$$

Where:

$\Delta\ddot{x}_{\text{I}}$	is the change in linear acceleration from equilibrium in x_{I} direction	[m s^{-2}]
$\Delta\ddot{y}_I$	is the change in linear acceleration from equilibrium in y_{I} direction	[m s^{-2}]
$\Delta\ddot{z}_I$	is the change in linear acceleration from equilibrium in z_{I} direction	[m s^{-2}]
$\Delta\phi$	is the change in roll from equilibrium	[rad]
$\Delta\theta$	is the change in pitch from equilibrium	[rad]
$\Delta\psi$	is the change in yaw from equilibrium	[rad]
$\overline{\phi}$	is the roll in equilibrium	[rad]
$\overline{\theta}$	is the pitch in equilibrium	[rad]
$\overline{\psi}$	is the yaw in equilibrium	[rad]

4.5 Model Simulation

The found models, i.e. the attitude and translational model of the system, are presented with the constituted non-linear equations followed by the same equations linearized. Hereafter, to give a better overview, two block diagrams illustrating the linearized equations are displayed. In the last part of the section the two models are simulated, this aims to analyze the angular and translational behavior. The simulation is also used to see if the linear approximation of the equations give a valid result compared to the non-linear equations.

4.5.1 Attitude Model Equations

- Model equations

$$J_x\ddot{\phi} = k_{\text{th}}(\omega_4^2 - \omega_2^2)L \tag{4.27}$$
$$J_y\ddot{\theta} = k_{\text{th}}(\omega_1^2 - \omega_3^2)L \tag{4.28}$$
$$J_z\ddot{\psi} = k_d(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{4.29}$$

- Linearized model equations

$$J_x\Delta\ddot{\phi} = 2k_{\text{th}}L\overline{\omega}_4\Delta\omega_4 - 2k_{\text{th}}L\overline{\omega}_2\Delta\omega_2 \tag{4.30}$$
$$J_y\Delta\ddot{\theta} = 2k_{\text{th}}L\overline{\omega}_1\Delta\omega_1 - 2k_{\text{th}}L\overline{\omega}_3\Delta\omega_3 \tag{4.31}$$
$$J_z\Delta\ddot{\psi} = 2k_{\text{d}}\overline{\omega}_1\Delta\omega_1 - 2k_{\text{d}}\overline{\omega}_2\Delta\omega_2 + 2k_{\text{d}}\overline{\omega}_3\Delta\omega_3 - 2k_{\text{d}}\overline{\omega}_4\Delta\omega_4 \tag{4.32}$$

4.5.2 Translational Model Equations

- Model equations

$$m\ddot{x}_{\text{I}} = -k_{\text{th}}({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \tag{4.33}$$
$$m\ddot{y}_{\text{I}} = -k_{\text{th}}({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \tag{4.34}$$
$$m\ddot{z}_{\text{I}} = F_g - k_{\text{th}}\ ({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2)\cos\phi\cos\theta \tag{4.35}$$

- Linearized model equations

$$m\Delta\ddot{x}_{\text{I}} = -k_{\text{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\Delta\theta \tag{4.36}$$
$$m\Delta\ddot{y}_{\text{I}} = k_{\text{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\Delta\phi \tag{4.37}$$
$$m\Delta\ddot{z}_{\text{I}} = -2k_{\text{th}}\overline{\omega}_1\Delta\omega_1 - 2k_{\text{th}}\overline{\omega}_2\Delta\omega_2 - 2k_{\text{th}}\overline{\omega}_3\Delta\omega_3 - 2k_{\text{th}}\overline{\omega}_4\Delta\omega_4 \tag{4.38}$$

From now on, the linearized variables are represented without the symbol Δ, to avoid excessive notation, even though they refer to changes around the linearization point.

4.5.3 Linearized Block Diagrams

The block diagrams of the linear approximated attitude and translational equations are illustrated in Figure 4.7 and 4.8.

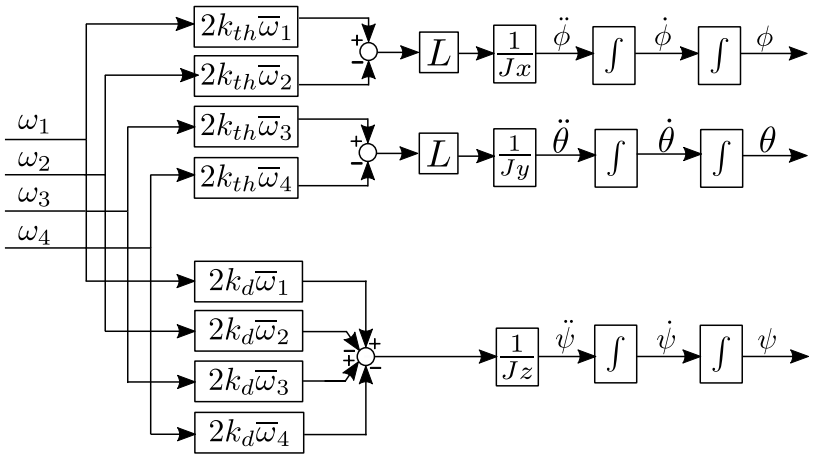

Figure 4.7: A block diagram of the linearized attitude model. The input are the angular velocities of the four motors and the output are the three angles roll, pitch and yaw.

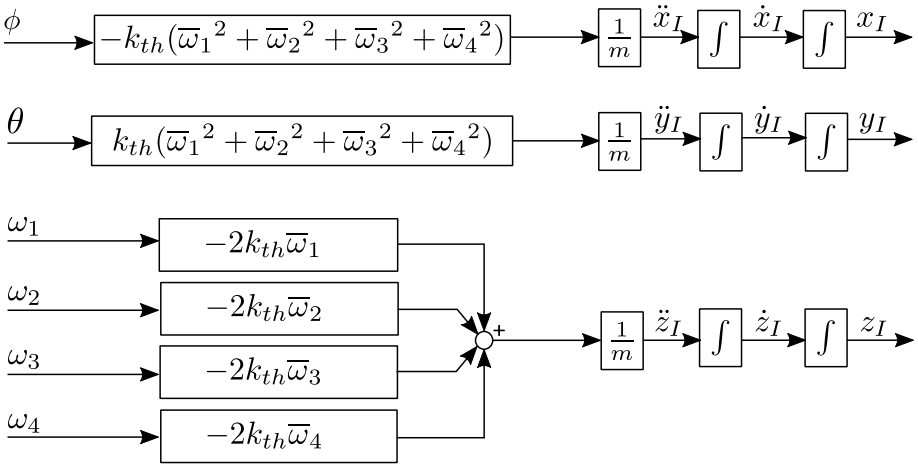

Figure 4.8: A block diagram of the linearized translational model. The input are the angular velocities of the four motors and the output are the three positions in x_I, y_I and z_I.

4.5.4 Simulation

A simulation of the different models is done using Simulink. The parameters needed for the simulations, that come from the model equations, are seen in Table 4.1.

Parameter Name	Symbol	Value	Units
Mass of the quadcopter	m	0.996	kg
Length of quadcopter arm	L	0.225	m
Moment of inertia around x_B axis	J_x	0.01073	kg m^2
Moment of inertia around y_B axis	J_y	0.01073	kg m^2
Moment of inertia around z_B axis	J_z	0.0208	kg m^2
Thrust force coefficient	k_{th}	1.32922 10^{-5}	N s^2 rad^{-2}
Drag torque coefficient	k_d	9.39741 10^{-7}	N m s^2 rad^{-2}

Table 4.1: Parameters used in the simulation of the model.

The mass of the quadcopter and the arm length are measured directly on the system. The three inertias are calculated analytically by dividing the quadcopter into different bodies for which the inertia is known, see Appendix E. Finally, the thrust and drag coefficients are found by measuring the output force and torque when turning the propeller at a known speed, see Appendix A and B.

Attitude Model Simulations

The angular behavior of the model is analyzed, where the input to the system are the four motor velocities. They are defined such that the response of the model is predictable and the performance can be evaluated.

The behavior around x_B axis is shown in Figure 4.9.

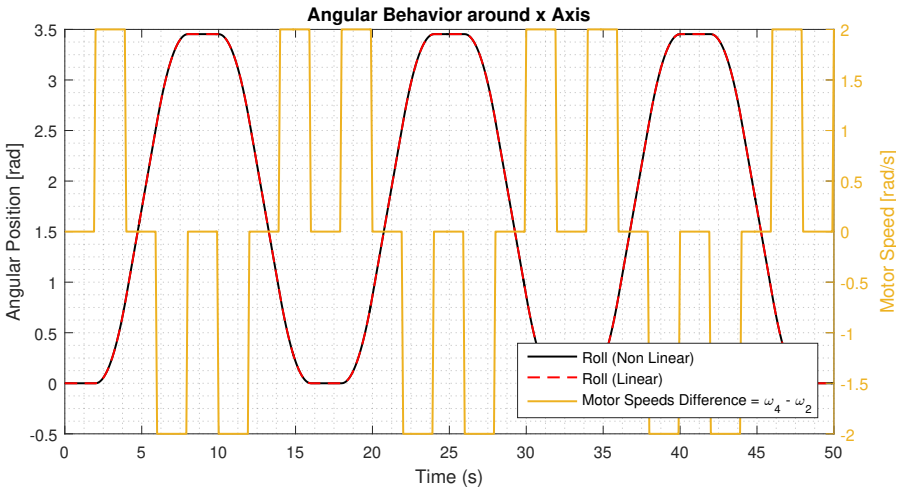

Figure 4.9: Behavior of the non linear and linear models in roll angle.

Only the speeds of motor 2 and 4 are changed as they are supposed to be the ones that affect the roll angle. It is seen that a positive difference in the rotational speeds of these

motors generates a positive acceleration in the roll direction, which is consistent with Equation 4.27.

It can also be observed that the linear approximation of the equations yields an accurate result as the two responses show no perceptible difference.

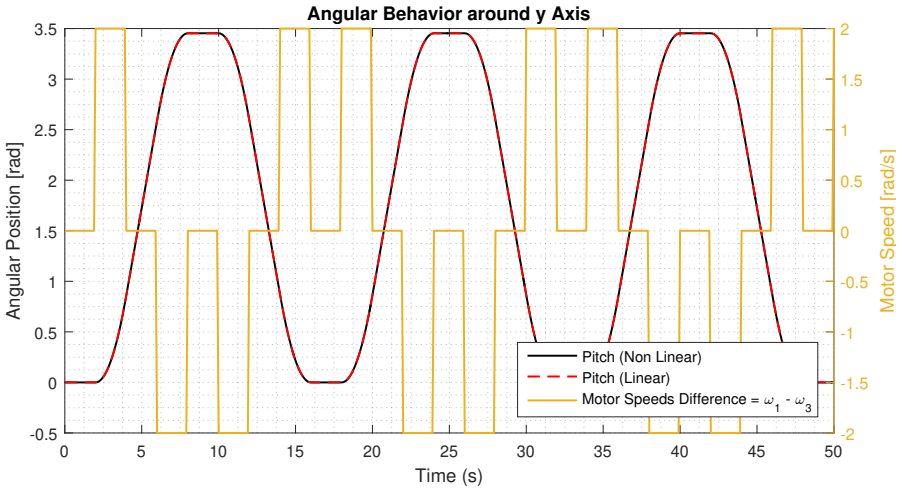

Figure 4.10: Behavior of the non linear and linear models in pitch angle.

The behavior around the y_B axis is very similar to that of the roll angle, see Figure 4.10. In this case, the motor velocities modified are those from motors 1 and 3. Equation 4.28 states that a positive rotational speed difference creates a positive pitch acceleration. This is confirmed by the simulations in both models. Again, the linear approximation can be considered adequate. The last attitude response simulated is around the z_B axis and is displayed in Figure 4.11.

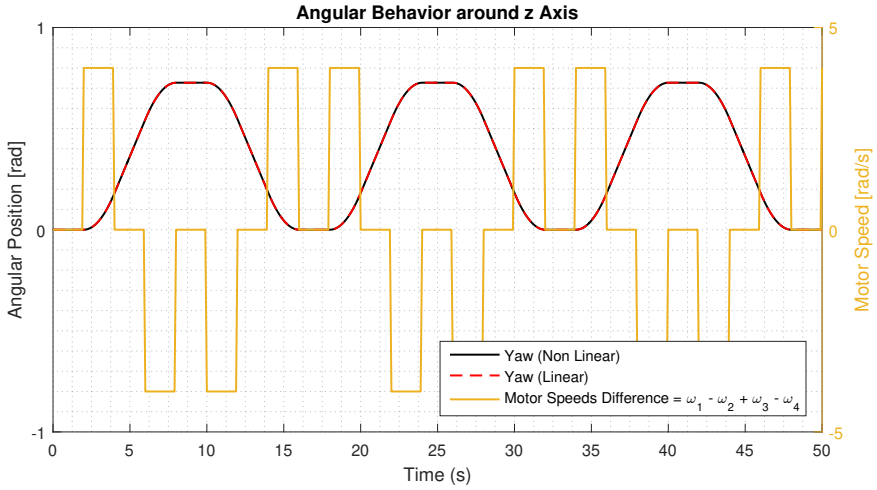

Figure 4.11: Behavior of the non linear and linear models in yaw angle.

In this case all motor velocities affect the yaw angle. In the simulation, velocities of motor 1 and 3 are increased while those of motor 2 and 4 are decreased and vice versa. This creates yaw accelerations according to Equation 4.29. The response of the model is also correct and the linear approximation is accurate.

As the linear attitude approximations are verified, the linear translational approximations are to be evaluated.

Translational Model Simulations

The translational behavior of the model is analyzed, where the inputs are the roll and pitch angles and the addition of the four motor velocities.

Figure 4.12 shows how the position in the x_I axis evolves with constant velocities of the motors and a change in the pitch angle.

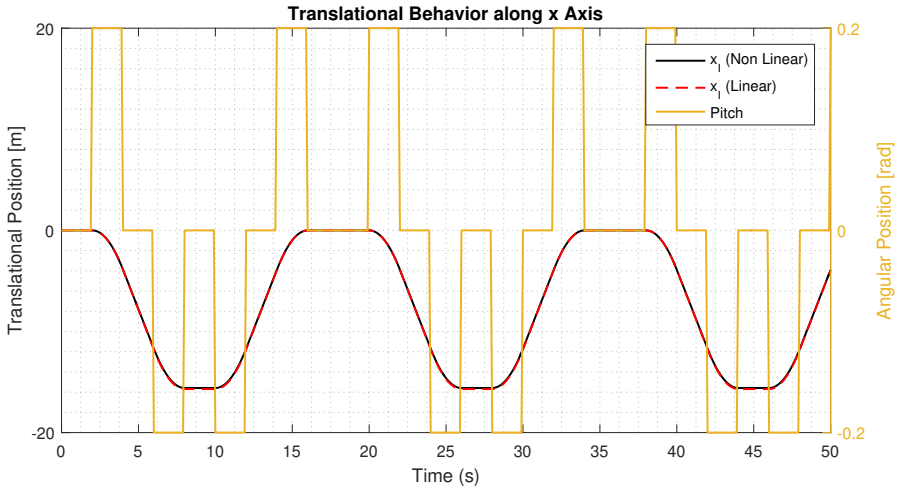

Figure 4.12: Behavior of the non linear and linear models along the x_I axis.

As expected, a negative translational acceleration is generated for a positive angle change, which drives the position to negative values. This response follows Equation 4.33. The linear approximation is considered suitable when looking at the linear model response.

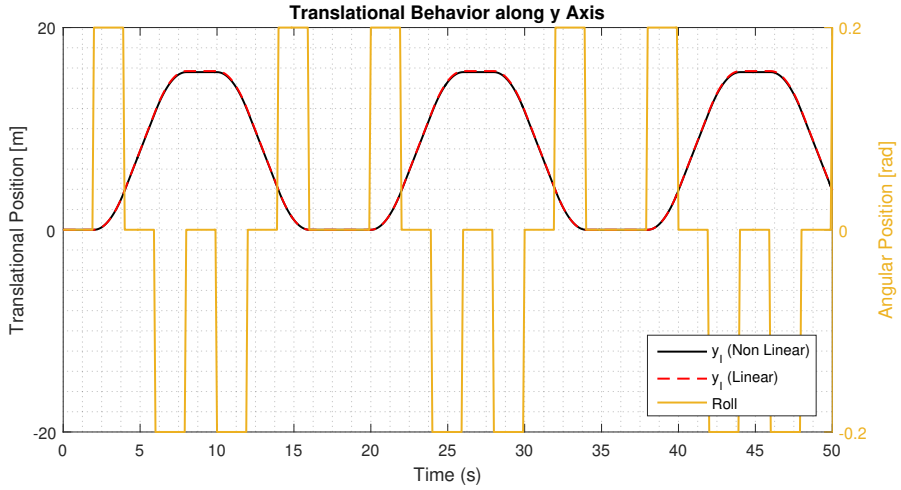

Figure 4.13: Behavior of the non linear and linear models along the y_I axis.

The response along the y_I inertial direction is depicted in Figure 4.13. It is similar to that of the x_I direction but in this case a change in roll is needed to change the acceleration.

Now, a positive roll angle generates a positive acceleration, as opposed to the x_I axis case. All goes in accordance with Equation 4.34. As previously, the linear approximation performs well in the simulation.

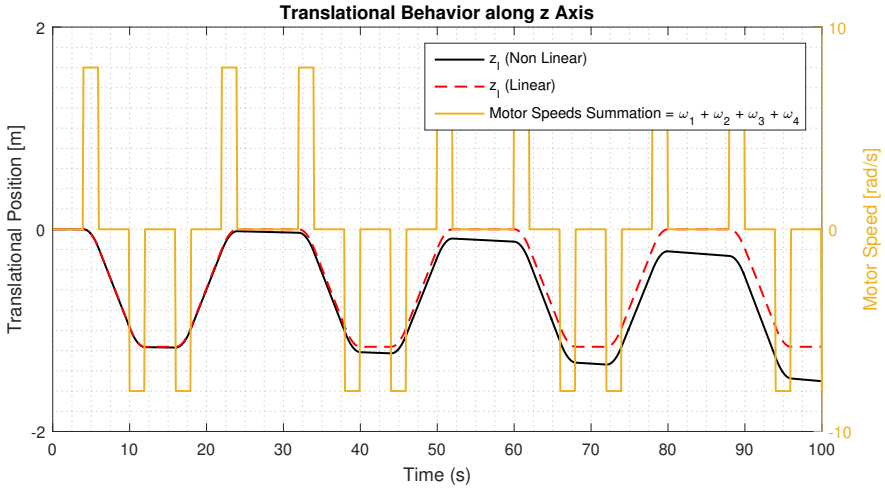

Figure 4.14: Behavior of the non linear and linear models along the z_I axis.

Finally, Figure 4.14 presents the behavior along the z_I axis. The pitch and roll angle are kept to zero and the summation of motor velocities are changed. For a higher sum, a more negative acceleration in the z_I axis is generated as seen in Equation 4.35 and in the simulation through the z_I position. In this case, the linear approximation of the squared velocity term in the equations creates a difference in the output z_I position between the two models that is noticeable in the graph. This is not a problem in the nonlinear model of the x_I and y_I translational models, as these only depend on equilibrium velocities. In the case of x_I and y_I, the velocities are constant squared and not the acting variables squared, as it is for the z_I controller.

In this chapter, the attitude and translational models of the system have been derived. A linear approximation of the obtained equations are used to design the necessary controllers, see chapter 6.

Part II

Design & Implementation

5 | Network

The quadcopter relies on a network between the ground station and the microcontroller, as mentioned in section 2.3. The information transmitted to the microcontroller is the attitude, position, translational velocity and the position reference.

To ensure that the microcontroller can detect packets and disregard incorrect packets received from the computer, a protocol is necessary. As mentioned in section 2.1 the lower level communication is handled by the XBees, and therefore only the transport layer is designed.

The effects of the network in the control system are analyzed using the MATLAB toolbox, TrueTime [22]. The issues considered are the delay from the sensor to the controller and the missed packets. Missed packets are considered to be when the controllers does not utilize the data from a new received packet in each loop.

5.1 Communication Protocol

The transport layer protocol designed must ensure that transmitted packets can be detected and that packets containing errors are not utilized by the microcontroller, as complying with the functional requirements stated in chapter 3.

As real-time features are desired for the quadcopter, the protocol chosen is UDP based. A connection oriented protocol implies a longer packet, retransmissions and a higher computation time, thus increasing the delay. A connectionless protocol is therefore utilized, as it leaves more processor resources available for the controllers.

As the communication is only from the computer to the microcontroller, the protocol does not need a source port or destination addresses, as typically seen in a UDP protocol. Furthermore, it is decided not to change the length of the packet relative to the gathered data. As the length is made constant, only a header is necessary and end sequence is not included. As common in connectionless protocols packet loss is not detected, hereby avoiding implementation of a sequence number in the header of the packet. A checksum, which constitutes the tail of the packet, is considered enough for disregarding corrupted information in the microcontroller.
These considerations simplify the protocol header and tail, compared to UDP, resulting in shorter transmission and computation time. The overall structure of a generated packet is illustrated in Table 5.1.

Header	Data	Checksum

Table 5.1: The structure of the packet.

5.1.1 Header

The header constitutes the start sequence of the packet and it is necessary for the microcontroller to be able to distinguish between transmitted packets and arbitrary data.

If the header is found in the received data, the microcontroller can then extract the information and checksum as the packet is of a fixed length.

The header is set to three bytes, where all the bits are set to ones. This consideration ensures the unlikeliness of finding a header in the middle of a packet. This is because it is highly unlikely for the Vicon system to register two of the transmitted variables, which are placed beside each other in the packet, to be all ones at the same time. The header can be seen in Table 5.1 and Table 5.4.

5.1.2 Data

The variables transmitted are the attitude, roll, pitch and yaw, the position, x_I, y_I and z_I, the position references, $x_{\mathrm{I}_\mathrm{ref}}$, $y_{\mathrm{I}_\mathrm{ref}}$ and $z_{\mathrm{I}_\mathrm{ref}}$, and the calculated translational velocities, $\dot{x}_\mathrm{I}$, $\dot{y}_\mathrm{I}$ and $\dot{z}_\mathrm{I}$.

	Unit	Bits	Range	Resolution	Data	→	Packet	→	Final Data
ϕ	rad	8	±2.55	0.01	0.567...	$\xrightarrow{\times 100}$	57	$\xrightarrow{\times 0.01}$	0.57
θ	rad	8	±2.55	0.01	0.853...	$\xrightarrow{\times 100}$	85	$\xrightarrow{\times 0.01}$	0.85
ψ	rad	8	±2.55	0.01	2.532...	$\xrightarrow{\times 100}$	253	$\xrightarrow{\times 0.01}$	2.53
x_I	mm	9	±5110	10	122.23...	$\xrightarrow{\times 0.1}$	12	$\xrightarrow{\times 0.01}$	120
y_I	mm	9	±5110	10	5020.36...	$\xrightarrow{\times 0.1}$	502	$\xrightarrow{\times 0.01}$	5020
z_I	mm	9	±5110	10	1056.89...	$\xrightarrow{\times 0.1}$	106	$\xrightarrow{\times 10}$	1060
$x_{\mathrm{I}_\mathrm{ref}}$	mm	9	±5110	10	104.23...	$\xrightarrow{\times 0.1}$	10	$\xrightarrow{\times 10}$	100
$y_{\mathrm{I}_\mathrm{ref}}$	mm	9	±5110	10	1056.89...	$\xrightarrow{\times 0.1}$	106	$\xrightarrow{\times 10}$	1060
$z_{\mathrm{I}_\mathrm{ref}}$	mm	9	±5110	10	1056.89...	$\xrightarrow{\times 0.1}$	106	$\xrightarrow{\times 10}$	1060
$\dot{x}_\mathrm{I}$	mm s^{-1}	10	±10230	10	10001.89...	$\xrightarrow{\times 0.1}$	1000	$\xrightarrow{\times 10}$	10000
$\dot{y}_\mathrm{I}$	mm s^{-1}	10	±10230	10	5642.96...	$\xrightarrow{\times 0.1}$	564	$\xrightarrow{\times 10}$	5640
$\dot{z}_\mathrm{I}$	mm s^{-1}	10	±10230	10	300.89...	$\xrightarrow{\times 0.1}$	30	$\xrightarrow{\times 10}$	300

Table 5.2: Overview of the data sent in the protocol, the range, the resolution and an example on how it transmitted.

The roll, pitch and yaw are measured in rad and are contained in a variable of nine bits each in the packet. The first bit is utilized for the sign. The attitude received from the Vicon system, where the maximum values that can be obtained are $\pm\pi$ rad, are multiplied by 100 and rounded to convert them in integers. $\pm\pi$ rad are only measured by the Vicon system if the quadcopter does a turn of 180 degrees. If this occurs in the roll and pitch, it is not possible for the quadcopter to recover from being turned upside down. Therefore, as 8 bits are chosen to send the value, the maximum measured value is 255, that is, 2.55 rad. If the yaw gets bigger than 2.55 rad the error which the controller receives is high, and it can still correct the angle, as yaw is always set to zero, see chapter 6.

The actual position coordinates, x_I, y_I and z_I, are measured in milimeters by the Vicon system and are contained in ten bits each. The first bit is the sign bit. By choosing a precision of centimeters, nine bits are sufficient to store the data. With this size, the position values are between $\pm$ 512 yielding a range of more than ten meters. Thus, making it sufficient for the quadcopter as it is operated in the Vicon room, see chapter 2. The position reference utilizes the same units as the actual position and the same amount of bits for its variables.

Each translational velocity, $\dot{x}_\mathrm{I}$, $\dot{y}_\mathrm{I}$ and $\dot{z}_\mathrm{I}$, is contained in 11 bits, with the first bit as a sign bit. Yielding a possibility of transmitting a value between $\pm$ 1024. The maximum velocity is 10 $\mathrm{m} \cdot \mathrm{s}^{-1}$, which is deemed sufficient.

All these data, constitutes 15 bytes as shown in Table 5.4.

5.1.3 Checksum

If the header is found and the next 15 bytes are extracted to be utilized in the controller, it is not certain that an actual full packet is extracted. As explained earlier, a header can be found in the middle of the packet and arbitrary information is utilized. Furthermore it is possible that the packet is sensitive to noise when transmitted from the computer to the microcontroller and the data therefore gets corrupted. It is necessary to check if the packet contains the correct information. This is done by utilizing a checksum.

The length of the checksum needs to be a multiple of the length of the data, which is 15 bytes. A big checksum, i.e. five bytes, reduces the probability of accepting an incorrect packet as more bits are involved in checking the packets. The drawback is that the packet size increases and so does the transmission time. On the other hand, having a small checksum, i.e. one byte, requires performing more computations and the chance of acquiring corrupted packets increases. As a compromise between the two approaches, a length of three bytes is chosen. An example of how the checksum is utilized and implemented can be seen in Table 5.3.

Bit array	Overflow	Byte	Byte	Byte
Part 1		00100001	01000000	01010000
Part 2		00100001	01000001	01010001
Part 3		00101001	01000000	01010000
Part 4		00100011	01001000	01010001
Part 5		10100001	01000010	01010100
Part 1+2+3+4+5	01	00110000	01001100	10010110
Add overflow		00110000	01001100	10010111
Checksum		11001111	10110011	01101000
Check		11111111	11111111	11111111

Table 5.3: An example of how the checksum is utilized and implemented.

Before the computer transmits the packet, the data is split up into five parts. The five parts are then summed. If the result is bigger than what can be represented with three bytes, the two most significant bits (column 2 in Table 5.3) are added to the summation. The checksum is generated by inverting the final summation. The checksum is then placed at the end of the packet, see Table 5.4.

When the microcontroller examines the packet for errors, it utilizes the checksum generated by the computer. The microcontroller divides the packet in the same five parts, adds the overflow bits and then adds the final summation together with the checksum. The packet is considered correct if the outcome is a binary number formed by ones.

It should be noted that the checksum's error handling is not bulletproof, as an error can cancel out another error. An example might be if a bit in part one is changed from false to true and another bit in the same position in part two is change from true to false. Then the checksum is the same and the error is not detected.

The total packet with the header, data and checksum can be seen in Table 5.4. The header is three bytes, the data is 15 bytes and the checksum is three bytes.

	Byte								Byte							
Byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	Header															
2									ϕ							
4		θ							ψ							
6			x													
8	y								z							
10		x_{ref}							y_{ref}							
12			z_{ref}													
14	$\dot{x}$								$\dot{y}$							
16			$\dot{z}$													
18	Checksum															
20																

Table 5.4: The generated packet transmitted from the computer to the microcontroller.

5.2 Delay and Missed Packets

The delay in the system constitutes one of the main network issues in the system. It forces the control design to be more robust and, in most cases, slower. In order to consider the delay in the controller design, the TrueTime simulation toolbox for MATLAB is utilized. This toolbox allows to simulate the control design, the derived model and the wireless network together in order to design a controller that is still stable.

In the simulation, a model for the delay is needed in order to implement it in TrueTime. The chosen approach is to model the delay as the maximum possible delay appearing in the system, so the worst case scenario is considered. Its value is obtained by adding the time required to make the information available in the serial port of the microcontroller and the maximum time elapsed since the data is available until it can be used by the control loop. Figure 5.1 shows the different network induced delays in the system and the maximum delay. It also shows the period of the control task, which is sampling time used when modeling.

The time required to make the data available for the microcontroller is caused by the time for the code to be executed in MATLAB and the delay in transmission, as seen in Figure 5.1. The code in MATLAB retrieves the sensor data and constructs the packet. The value of the delay in MATLAB is found by timing the code. The value obtained is 15 ms.

The transmission time can be found using the transmission speed of the XBee modules. According to the datasheet [14], it is 115.2 kbps. With this speed and transmitting 21 bytes, the transmission delay is 1.46 ms. This means that the recorded data takes 16.46 ms to get to the microcontroller.

Once the information is on the microcontroller, the maximum time elapsed until the controller uses the data is the transmission period, 15 ms, see Figure 5.1. The reason for this is that in the worst possible case, the packet arrives one sending period before the next control loop starts. A longer waiting time is not possible as another packet would arrive before the next control loop.

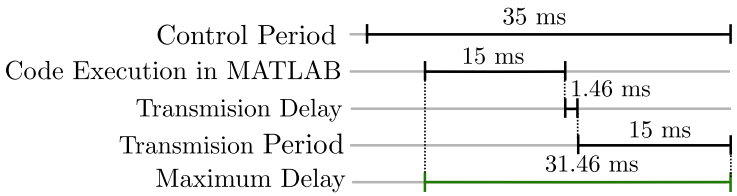

Figure 5.1: Delays in the packets transmitted to the microcontroller and maximum delay.

After all considerations, the value of the maximum delay which is used in the simulation is 31.46 ms.

When considering issues in the network, it is necessary to look into the missed packets. A missed packet can occur if either a new packet is not received between two control loops, making the next control loop use old data, or a received packet is corrupted.

This issue can be included in the simulation by utilizing TrueTime. Here the missed packets can be simulated as a constant probability. To estimate it, tests are conducted in which 10000 packets are transmitted. The amount of packets received correctly and processed in the microcontroller together with the number of control loops executed, provides an estimate of the probability of missing packets. The data obtained with the tests is shown in more detail in section 8.1. The probability of missing a packet is found to be 0.

The protocol has been designed and the main network issues, the delay and missed packets, have been analyzed. This information is used in the design of the controllers.

6 | Control Design

The quadcopter's dynamics have been described with a mathematical model, for both the attitude and the translational behavior, see chapter 4. This model is utilized when designing the different controllers. They are designed to fulfill the functional requirements stated in chapter 3.

Figure 6.1 shows the relation between the different controllers.

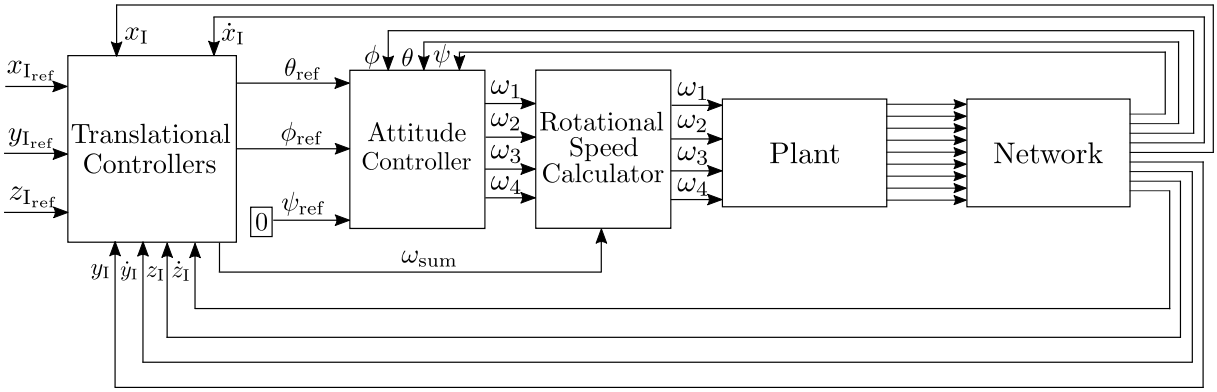

Figure 6.1: Block diagram of the overall control system.

The translational controller, along x_{I}, y_{I} and z_{I}, are designed with classical control methods. These controllers receive $x_{\mathrm{I}_{\mathrm{ref}}}$, $y_{\mathrm{I}_{\mathrm{ref}}}$ and $z_{\mathrm{I}_{\mathrm{ref}}}$ positions. The translational controllers for x_{I} and y_{I} constitutes an outer loop with the attitude controller as the inner loop. The z_{I} controller, separated from attitude controller, handles the amount of thrust applied by the motors, by changing the required sum of their rotational speeds. The attitude controller is designed using a state space approach. Note that the reference for yaw is set to zero so the quadcopter is able to move along x_{I} and y_{I} by changing θ and ϕ independently.

The rotational speed calculator receives the speeds required by the attitude controller and the sum of speeds requested by the z_{I} controller. It combines the five inputs into the four motor rotational speeds applied to the system.

The last block in Figure 6.1 is the network. This is necessary to consider when designing controllers utilized with a network distributed system. As the distributed network can include delay and missed packets, also described in chapter 5.

This chapter starts by describing the attitude controller followed by the design of the translational controller. In the design process, simulations including the network effects and the model are depicted. The simulations also consider the sampling frequency, which is 28 Hz as described in section 7.3.

6.1 Attitude Controller

The attitude response of the quadcopter constitutes a coupled behavior, as all the three Euler angles, ϕ, θ and ψ, are affected by the same four motor velocities, see Figure 6.1. A state space approach is therefore utilized, as this method is convenient when the system has multiple input and multiple sensed outputs [23].

The system behavior is represented by means of the linearized model equations. These linearized equations can be represented in state space form, shown in Equation 6.1 and 6.2.

$$\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u} \tag{6.1}$$

$$\mathbf{y} = \mathbf{C}\,\mathbf{x} + \mathbf{D}\,\mathbf{u} \tag{6.2}$$

Where:

x	are the states of the system
u	are the inputs of the system
y	are the outputs of the system
A	is the system matrix
B	is the input matrix
C	is the output matrix
D	is the direct transmission term

A block diagram of the state space description can be seen in Figure 6.2.

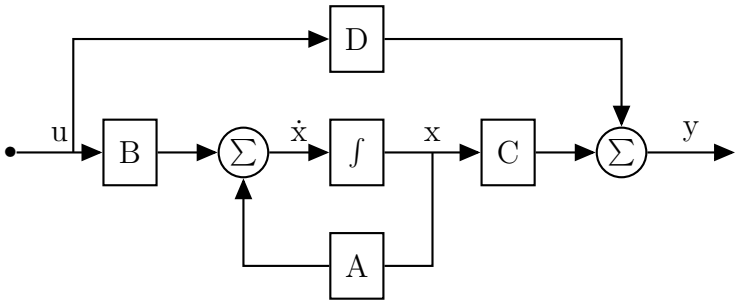

Figure 6.2: Block diagram of the state space representation of the system.

The state vector, **x**, contains six elements, as there are three attitude second order differential equations, see the linearized Equation 4.30, 4.31 and 4.32. As for mechanical systems, the states are the angular velocities and positions of the system [24].

The variables in the linearized equations are represented without the symbol Δ, even though they refer to changes around the linearization point.

The size of the input vector, **u**, corresponds to the number of inputs, which are the four motor velocities. The output, **y**, contains the three angles, ϕ, θ and ψ. The generated vectors can be seen in Equation 6.3.

$$\mathbf{x} = \begin{bmatrix} \phi \\ \theta \\ \psi \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{6.3}$$

The specific matrices for the description of the angular behavior are obtained from the linearized equations of the system. The size of the input matrix, **B**, is $n \times m$, where n is the order of the system and m is the number of inputs. The size of the system matrix, **A**, is an $n \times n$ and the output matrix, **C**, is an $1 \times n$ matrix. The matrices generated from the linear attitude equations are showed embedded in the state space representation in Equation 6.4 and 6.5.

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -\frac{2\ \mathrm{k_{th}}\ \mathrm{L}\ \overline{\omega}_2}{\mathrm{J_x}} & 0 & \frac{2\ \mathrm{k_{th}}\ \mathrm{L}\ \overline{\omega}_4}{\mathrm{J_x}} \\ \frac{2\ \mathrm{k_{th}}\ \mathrm{L}\ \overline{\omega}_1}{\mathrm{J_y}} & 0 & -\frac{2\ \mathrm{k_{th}}\ \mathrm{L}\ \overline{\omega}_3}{\mathrm{J_y}} & 0 \\ \frac{2\ k_d\ \overline{\omega}_1}{J_z} & -\frac{2\ k_d\ \overline{\omega}_2}{J_z} & \frac{2\ k_d\ \overline{\omega}_3}{J_z} & -\frac{2\ k_d\ \overline{\omega}_4}{J_z} \end{bmatrix} \mathbf{u} \tag{6.4}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} \tag{6.5}$$

Before designing the controller, it is convenient to check controllability and observability of the system. For doing so, the controllability and observability matrices shown in Equation 6.6 are used.

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A^2B} & \mathbf{A^3B} & \mathbf{A^4B} & \mathbf{A^5B} \end{bmatrix} \qquad \mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \mathbf{CA}^3 \\ \mathbf{CA}^4 \\ \mathbf{CA}^5 \end{bmatrix} \tag{6.6}$$

The rank of these matrices is six, that is, the number of states. This makes the system both controllable and observable. The entries of the matrices can be seen in Appendix F.

The dynamics of the system can be analyzed by looking at the system matrix **A**. The eigenvalues of this matrix represent the location of the system's open loop poles. In this case, the system has six poles located in zero, which means that the system is marginally stable. In order to place those poles at a stable location, a state feedback is used. This is combined with an integral controller to be able to set a desired reference, different from zero, for the roll and pitch and reject input disturbances in the three angles.

For doing the state feedback, the angular velocities are also needed even though they are not measured. The way of obtaining them is using a reduced order observer. This is done even though the angular velocities could be estimated with a numerical differentiation procedure. The observer is more convenient in case of using on board sensors and by using it, that possibility is kept open.

This approach has the possibility of designing the state feedback, integral control and the reduced order observer independently. This holds due to the separation principle [25], which states that the closed loop poles of the system are the combination of the state feedback with integral control poles and the observer poles. This implies that the system has 12 closed loop poles.
A block diagram showing the system overview of the attitude controller is shown in Figure 6.3. Here the state feedback, the integral control and the reduced order observer can be seen.

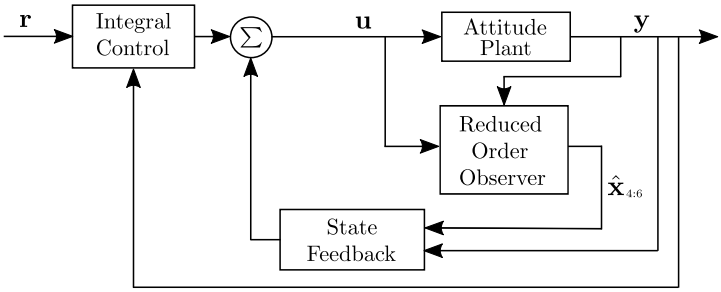

Figure 6.3: Control structure for the system, including the state feedback, the integral control and the reduced order observer.

In the following subsections, the state feedback with the integral control and then the reduced order observer are designed.

6.1.1 State Feedback with Integral Control

The aim of the attitude controller is to be able to track the references for roll, pitch and yaw. Given a state space representation as the one in Equation 6.1 and 6.2, it is possible to design an attitude controller so that this aim is achieved.

In this approach, a state feedback is used to make the system return to the equilibrium position while an integral controller makes it possible to include a new reference and track it while rejecting input disturbances. These two control actions are added together and four velocity values for the motors are transmitted to the Rotational Speed Calculator seen in Figure 6.1. The diagram in Figure 6.4 shows how the two controllers are related. The Figure 6.5 illustrates the highlighted part in Figure 6.4 in more detail, with the actual variables used in the design process.

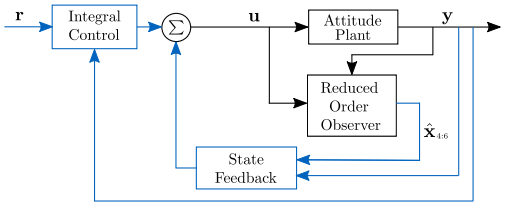

Figure 6.4: Control structure highlighting the state feedback and integral control.

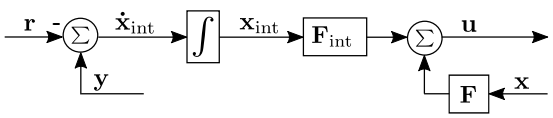

Figure 6.5: The highlighted parts from the left diagram in detail, including all the variables used in the design of the controller.

From Figure 6.5, the feedback law yields Equation 6.7.

$$\mathbf{u} = \mathbf{F}\mathbf{x} + \mathbf{F}_{\text{Int}}\mathbf{x}_{\text{Int}} \tag{6.7}$$

Where:

$\mathbf{F}$	is a 4x6 state feedback matrix
$\mathbf{F}_{\text{Int}}$	is a 3x6 integral feedback matrix

In this equation, $\mathbf{x}_{\text{Int}}$ is given by Equation 6.8, which is equivalent to Equation 6.9, see Figure 6.5.

$$\mathbf{x}_{\text{Int}} = \int_0^t \mathbf{y}(\tau) - \mathbf{r}(\tau)\, d\tau \tag{6.8}$$

$$\dot{\mathbf{x}}_{\text{Int}} = \mathbf{y} - \mathbf{r} \tag{6.9}$$

This last equation is included in the existing state space model, having the system states,

$\mathbf{x}$, and taking $\mathbf{x}_{\text{Int}}$ as new states, yielding the expressions Equation 6.10 and 6.11 [26].

$$\dot{\mathbf{x}}_{\text{e}} = \mathbf{A}_{\text{e}}\mathbf{x}_{\text{e}} + \mathbf{B}_{\text{e}}\mathbf{u} + \begin{bmatrix} \mathbf{0} \\ -\mathbf{I} \end{bmatrix} \mathbf{r} \tag{6.10}$$

$$\mathbf{y} = \mathbf{C}_{\text{e}}\mathbf{x}_{\text{e}} \tag{6.11}$$

where

$$\dot{\mathbf{x}}_{\text{e}} = \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_{\text{Int}} \end{bmatrix} \qquad \mathbf{A}_{\text{e}} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \qquad \mathbf{B}_{\text{e}} = \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{C}_{\text{e}} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \end{bmatrix}$$

The resulting feedback law can be design as a conventional state feedback, where the goal is to choose an appropriate $\mathbf{F}_{\text{e}} = [\mathbf{F}\ \mathbf{F}_{\text{Int}}]$ matrix such that the eigenvalues of $\mathbf{A}_{\text{e}} + \mathbf{B}_{\text{e}}\mathbf{F}_{\text{e}}$ are the new poles of the system.

The Linear Quadratic Regulator (LQR) approach is utilized to design an optimal state feedback for the controller. This approach is utilized together with Bryson's rule, which yields the possibility to prioritize the convergences of the states in the system. Furthermore it does this while being able to set boundaries for the control action applied to the system.

The LQR finds the state feedback that minimizes the cost function Equation 6.12. [26].

$$J = \int_0^\infty \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\ dt \tag{6.12}$$

Where:

$\mathbf{Q}$	is the state-cost matrix
$\mathbf{R}$	is the input-cost matrix

In the cost function, $\mathbf{Q}$ and $\mathbf{R}$ are positive semi-definite and positive definite matrices, respectively. $\mathbf{Q}$ defines the penalties for the error in the states making the cost function value higher for the prioritized states. $\mathbf{R}$ sets the weights for the inputs of the system such that the control action does not exceed its maximum limits. [26]

For the LQR to minimize the cost function, the matrices $\mathbf{Q}$ and $\mathbf{R}$ need to be found. This is done using Bryson's rule as seen in Equation 6.14. The $\mathbf{Q}$ and $\mathbf{R}$ matrices are diagonal matrices. The $\mathbf{Q}$ matrix elements are a function of the maximum acceptable value for the error in the states. The $\mathbf{R}$ elements set limits for the control action in each of the inputs. [27].

$$Q_{ii} = \frac{1}{\text{maximum acceptable value of } [x_i^2]} \tag{6.13}$$

$$R_{ii} = \frac{1}{\text{maximum acceptable value of } [u_i^2]} \tag{6.14}$$

The maximum state error and maximum control action work as soft boundaries that give priorities to the different states and control actions.

The state feedback gain, $\mathbf{F}_e$, can be calculated using the expression seen in Equation 6.15.

$$\mathbf{F}_e = -\mathbf{R}^{-1}\mathbf{B}_e{}^T\mathbf{P} \tag{6.15}$$

Where:

$\mathbf{P}$ is the state-transfer matrix.

The positive definite matrix $\mathbf{P}$, provides, through Equation 6.15, the feedback matrix which minimizes the cost function in Equation 6.12. This can be found using the Algebraic Riccatti equation Equation 6.16. [27]

$$\mathbf{A}e^T\mathbf{P} + \mathbf{P}\mathbf{A}_e - \mathbf{P}\mathbf{B}_e\mathbf{R}^{-1}\mathbf{B}_e^T\mathbf{P} + \mathbf{Q} = \mathbf{0} \tag{6.16}$$

Once $\mathbf{F}_e$ is obtained, it can be split into $\mathbf{F}$ and $\mathbf{F}_{\text{Int}}$ as the first six columns are the state feedback and the last three are the integral control. These matrices can be seen in Appendix F. In this way, the controller can be implemented as shown in Figure 6.5. [26]

6.1.2 Reduced Order Observer

A reduced order observer is designed for estimating the angular velocities. In general, an observer is utilized in a control system if either one of two specific cases occur. If certain states in the system are not measured, an observer can estimate the unmeasured states by the means of some of the measured states. However, it is also possible to use an observer if the measured data gathered from the sensors is affected with a significant amount of noise, as the observer also filters the measurements [28]. The former is the reason for using an observer in the control system.

As described earlier, the attitude model has six states. With the use of the observer, the first three states, $\mathbf{x}_{1:3}$ are equal to the outputs, $\mathbf{y}$, whereas the other three states are estimated, $\hat{\mathbf{x}}_{4:6}$.

In Figure 6.6, a diagram illustrating the setup of the attitude controller is shown. The highlighted lines correspond to the reduced order observer and its inputs and outputs.

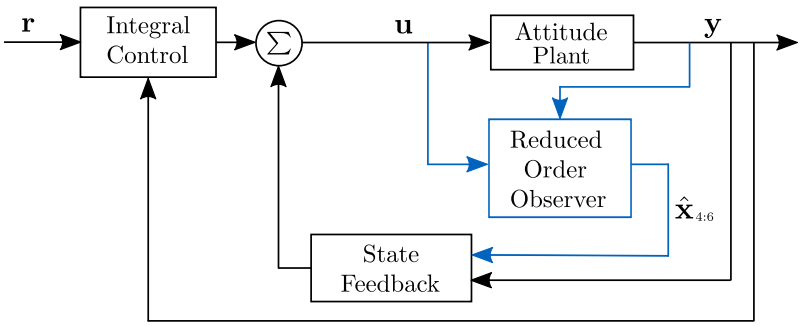

Figure 6.6: An overall block diagram of the attitude controller highlighting the reduced order observer and its corresponding inputs and output.

The matrices that define the system can be separated into submatrices, which are used in the design of the observer [29].

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \end{bmatrix}$$

As the system is observable, there exists a matrix $\mathbf{L}_{obs}$ such that Equation 6.17 is stable.

$$\mathbf{A_{22}} + \mathbf{L_{obs}}\mathbf{A_{12}} \tag{6.17}$$

The observer ensures that the estimate, $\hat{x}_{4:6}$, converges to $x_{4:6}$ with a rate yielded by the eigenvalues of Equation 6.17 [29].

$$\dot{\hat{\mathbf{x}}}_{\mathbf{4:6}} + \mathbf{L}_{\text{obs}}\dot{y} = (\mathbf{A}_{22} + \mathbf{L}_{\text{obs}}\mathbf{A}_{12})\hat{\mathbf{x}}_{4:6} + (\mathbf{A}_{21} + \mathbf{L}_{\text{obs}}\mathbf{A}_{11})\mathbf{y} + (\mathbf{B}_2 + \mathbf{L}_{\text{obs}}\mathbf{B}_1)\mathbf{u} \tag{6.18}$$

To calculate an appropriate observer gain, L_{obs}, it is necessary to evaluate the reliability of the utilized sensor and the model, while also taking into account the sampling rate and system delay. This is done through the different simulations shown in subsubsection 6.1.2.

This estimation is also shown in the form of a block diagram, Figure 6.7.

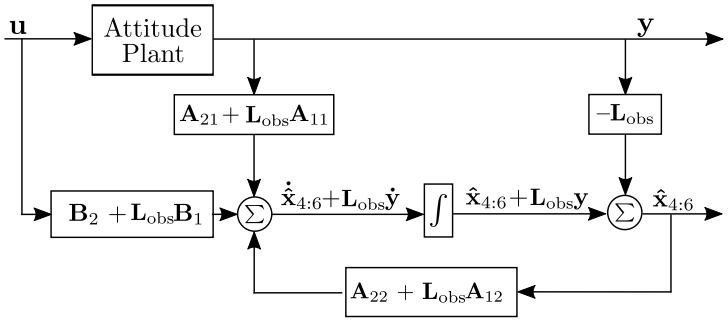

Figure 6.7: A diagram showing the reduced order observer and how it is implemented.

Attitude Controller Simulation

The design for the state feedback and the observer are provided. The final attitude response, observer estimation and step response can be analyzed and discussed. The controller and observer gains are changed in the different simulations to illustrate why the final design is chosen.

In Figure 6.8, the simulated angle response of the final attitude controller is shown. Initial conditions are given to the three angles. These are set to 0.2 rad for the pitch, -0.3 rad for the roll and -0.2 rad for the yaw. This shows how the controller stabilizes the attitude around 0 rad.

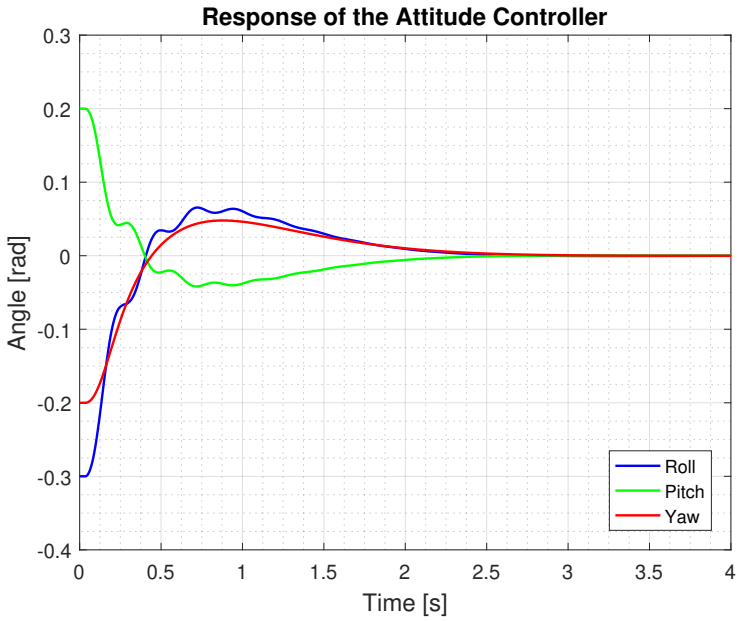

Figure 6.8: System response of the attitude controller, with initial conditions of 0.2 rad for the pitch, -0.3 rad for the roll and -0.2 rad for the yaw. The weighted matrices, $\mathbf{Q_{final}}$ and $\mathbf{R_{final}}$, and the observer gain, $\mathbf{L_{obs}}$, are utilized for the final design and can be found in Appendix F.

It can be seen from Figure 6.8 that it approximately takes 2.5 s for the pitch and roll to settle at 0 rad, where it only takes approximately 1.4-1.5 s for the yaw.

In Figure 6.9, the estimation for the angular velocities, done with the same conditions as in Figure 6.8, is depicted. To be able to evaluate the observer's estimation (red line), it is compared with the actual angular velocity (blue line).

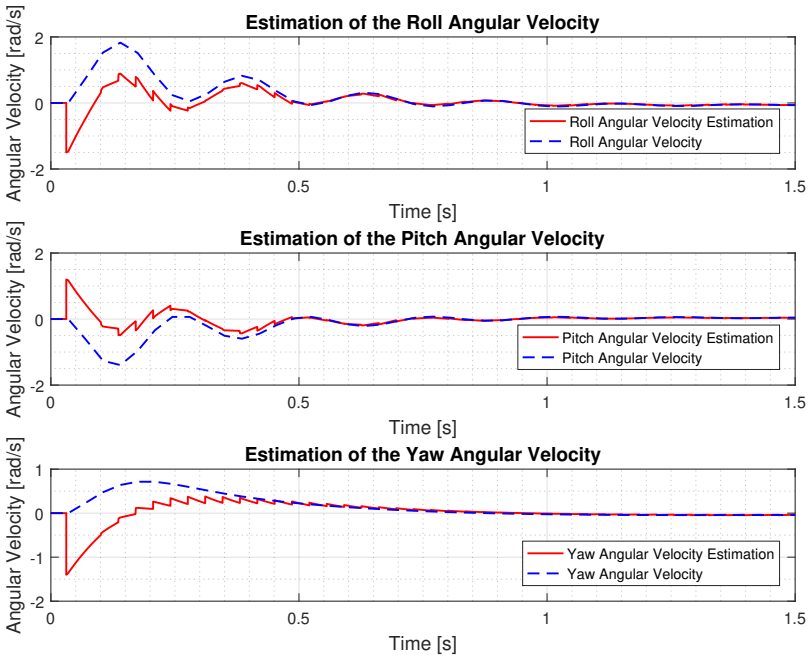

Figure 6.9: The three figures shows the estimation of the angular velocities made by the implemented observer. The simulations initial conditions for roll, pitch and yaw are identical to the simulation in Figure 6.8. The utilized weighted matrices, $\mathbf{Q_{final}}$ and $\mathbf{R_{final}}$, and the observer gain, $\mathbf{L_{obs}}$, can be found in Appendix F.

The estimations' large spikes in the beginning of the responses in Figure 6.9 are due to the initial conditions for the angles. The large gap between the actual and the estimation, is due to multiple factors. The main issue is the convergence of the estimate to the real value as the dynamics of the observer do not allow an instantaneous estimation. Also, the delay and sampling frequency affect the estimation, creating the spikes seen in the plot.

A step response of the roll is shown in Figure 6.10.

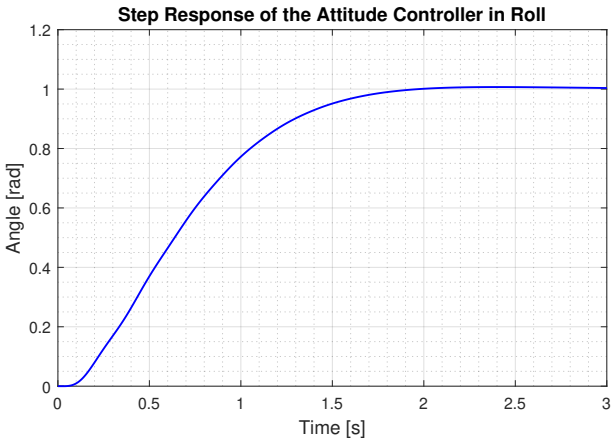

Figure 6.10: An step response of the final system. This illustrates only the behavior of the roll, as the yaw is always set to zero and the pitch would be identical to the roll. The utilized weighted matrices, $\mathbf{Q_{final}}$ and $\mathbf{R_{final}}$, and the observer gain, $\mathbf{L_{obs}}$, can be found in Appendix F.

As the roll and the pitch are implemented identically, the step response of the two are similar. Additionally, the yaw is not illustrated as the reference is set to zero. It is therefore only desired to discover how the roll and pitch track a reference.
The system is subjected to a step input reference of 1 rad. With an error band of 5 %, the settling time is approximately 1.5 s and the overshoot is less than 5 %.

To illustrate the reasoning followed through the design process, two more simulations are shown, where the controller and observer matrices are changed one at a time, so that it is possible to compare the different simulations with the final design and discuss why the final parameters are chosen.

The changed matrices, called $\mathbf{Q_{high}}$ and $\mathbf{R_{high}}$ can be seen in Appendix F. Compared to the final design the allowed state error in the angles is less and the allowed control action is increased. These changes should yield a more reactive system. This can be seen in the angle response shown in Figure 6.11. Note that the initial conditions for the response are the same as for Figure 6.8.

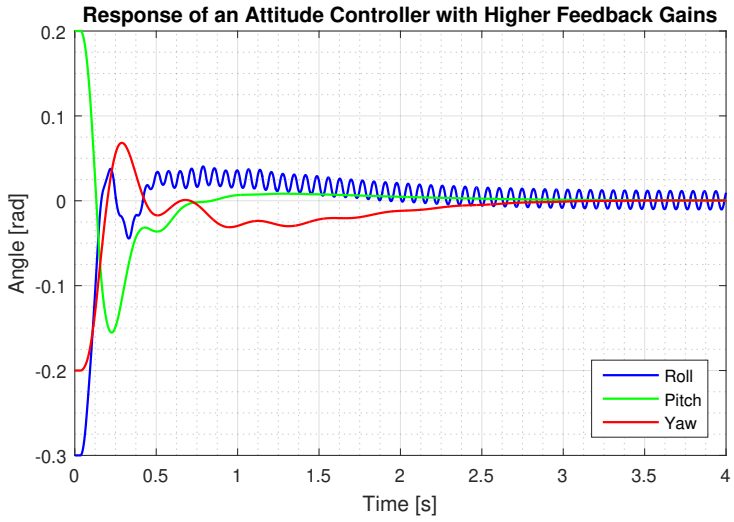

Figure 6.11: Attitude controller response utilizing the weighted matrices, $\mathbf{Q_{high}}$ and $\mathbf{R_{high}}$, and the observer gain, $\mathbf{L_{obs}}$, found in Appendix F. The initial conditions for the angles is set to 0.2 rad for the pitch, -0.3 rad for the roll and -0.2 rad for the yaw.

It can be seen from Figure 6.11 that the roll shows an oscillatory behavior. The network effects in the system and the more reactive controller cause this response.

It is possible to conclude that a higher feedback gain compared to the final utilized gain is not desired for the attitude controller.

The observer gain, $\mathbf{L_{obsH}}$, utilized for the simulation in Figure 6.12 is increased compared to the observer gain utilized in the final design. This results in more oscillations in both the pitch and the roll. Additionally, the attitude needs more time to settle compared to

the response shown in Figure 6.8. It is seen that when increasing the gain the observer is worse at estimating the angular velocity compared to the final design mainly due to the delay and sampling time effect.

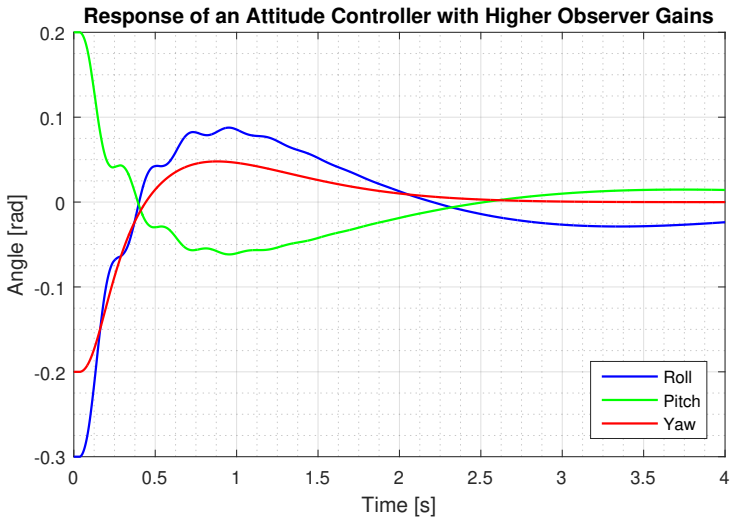

Figure 6.12: Angle response with an increase in the observer gain compared to the final design. The utilized weighted matrices, $\mathbf{Q_{final}}$ and $\mathbf{R_{final}}$, and the observer gain, $\mathbf{L_{obsH}}$ can be found in Appendix F.

From the simulations in Figure 6.11 and 6.12 it is evident that an increase in gain in either the state feedback or in the observer is not desired compared to the final design.

The final attitude controller is therefore a compromise between a fast system, with oscillating behavior, and a slower system without oscillating behavior.

6.2 Translational Controllers

The translational controllers handle the movement of the quadcopter along the inertial frame directions, x_I, y_I and z_I. The overall diagram in Figure 6.1 is rearranged to illustrate the translational controller design structure in Figure 6.13.

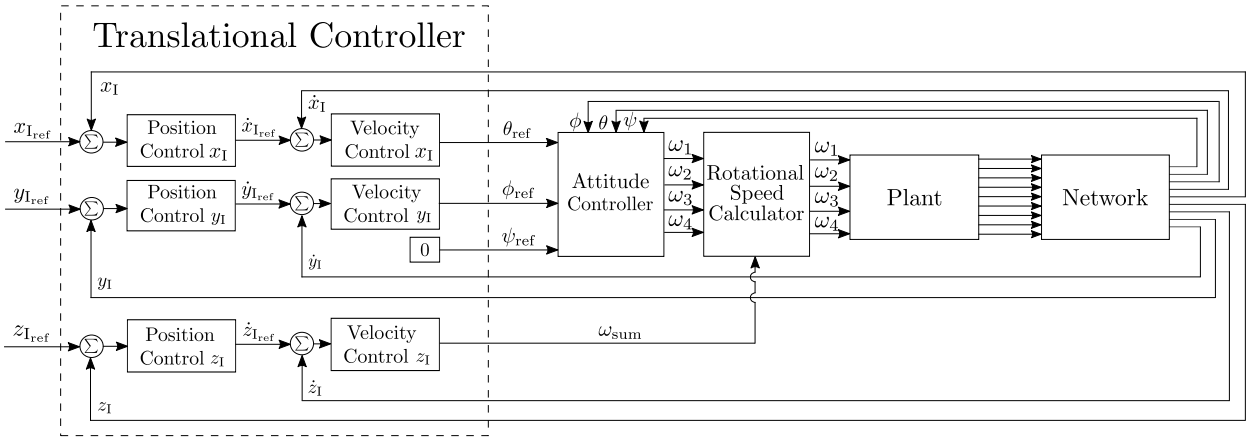

Figure 6.13: The overall translational control design structure.

For each axis, the velocity and position controllers form a cascade control structure, where the position controller sets the reference for the velocity controller. In the figure, it is also possible to see how x_I and y_I controllers share similar properties as both have as output an angle reference for the attitude controller, θ_ref and ϕ_ref, respectively. The z_I velocity controller sets the required sum of motor rotational speeds, ω_sum, such that the vertical position is attained.

In the following sections the design for the mentioned controllers is explained.

6.2.1 Controllers for $\mathbf{x_I}$ and $\mathbf{y_I}$ Directions

The design of the controllers for the translational movement in x_I and y_I directions is done using the transfer functions of the plant.

The linear equations for the translational model, derived in chapter 4, relate these movements with the roll and the pitch that the quadcopter has at each instant.

$$m\ddot{x}_\mathrm{I} = -k_\mathrm{th}(\overline{\omega}_1{}^2 + \overline{\omega}_2{}^2 + \overline{\omega}_3{}^2 + \overline{\omega}_4{}^2)\theta \tag{6.19}$$

$$m\ddot{y}_\mathrm{I} = k_\mathrm{th}(\overline{\omega}_1{}^2 + \overline{\omega}_2{}^2 + \overline{\omega}_3{}^2 + \overline{\omega}_4{}^2)\phi \tag{6.20}$$

Notice that the notation Δ, which correspond to the change from the linearization point, is removed from these equation to avoid excessive notation.

Laplace transforming Equation 6.19 and 6.20 yields:

$$m\dot{x}_{\mathrm{I}}s = -k_{\mathrm{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\theta \tag{6.21}$$

$$m\dot{y}_{\mathrm{I}}s = k_{\mathrm{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)\phi \tag{6.22}$$

The inner velocity controller design requires a transfer function from each of the angles to each of the translational velocities. This can be written as:

$$G_{\dot{x}_{\mathrm{I}}} = \frac{\dot{x}_{\mathrm{I}}}{\theta} = \frac{-k_{\mathrm{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)}{ms} = \frac{K_{\dot{x}_{\mathrm{I}}}}{s} \tag{6.23}$$

$$G_{\dot{y}_{\mathrm{I}}} = \frac{\dot{y}_{\mathrm{I}}}{\phi} = \frac{k_{\mathrm{th}}({\overline{\omega}_1}^2 + {\overline{\omega}_2}^2 + {\overline{\omega}_3}^2 + {\overline{\omega}_4}^2)}{ms} = \frac{K_{\dot{y}_{\mathrm{I}}}}{s} \tag{6.24}$$

Where:

$G_{x_{\mathrm{I}}}$	is the plant for the translational velocity in x_{I} direction	[m s^{-1} rad^{-1}]
$G_{y_{\mathrm{I}}}$	is the plant for the translational velocity in y_{I} direction	[m s^{-1} rad^{-1}]

From Equation 6.23 and 6.24 it can be noticed that the two plants are similar but with different sign. The controller design is carried out for the x_{I} translational velocity and applied to both afterwards, as the design is similar.

The transfer functions are formed by an integrator and a gain. This means the systems are marginally stable in open loop. $G_{x_{\mathrm{I}}}$ has a negative gain, which means that it becomes unstable in closed loop if a controller with positive gain is placed. This feature requires a negative controller gain to compensate for the plant.

It is noticeable that, for this design, the variable changed by the controller is the pitch angle reference for the attitude controller. In order to reduce the effect of the dynamics of the inner loop in the design, the closed loop bandwidth of the outer control loop should be lower.

The bandwidth of the inner attitude control is obtained by making a Bode plot approximation from the pitch angle reference to the real pitch angle in the quadcopter. The obtained Bode diagram is shown in Figure 6.14. It shows that the bandwidth of the inner loop of 2 rad s^{-1}. It has been chosen to set the closed loop bandwidth three times lower than the inner loop. Thus yielding a bandwidth for the velocity control loop of 0.7 rad s^{-1}.

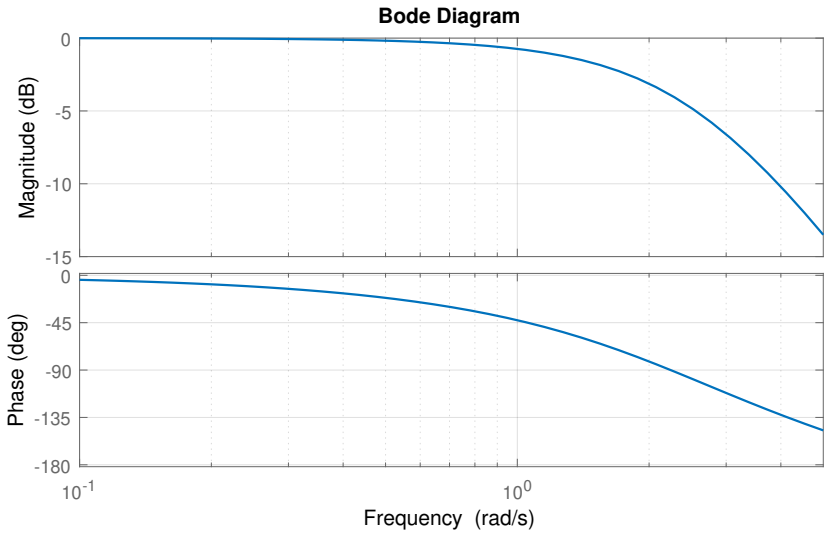

Figure 6.14: Attitude control loop Bode diagram.

As the plant is a type 1 system, since it contains one integrator, there is no closed loop steady state error if no input disturbances occur. Possible causes for input disturbances are wind, different thrust forces generated in the propellers for the same rotational speed and different rotational speeds in the motors when given the same rotational speed commands.

A diagram of a closed loop system with input disturbance can be seen in Figure 6.15.

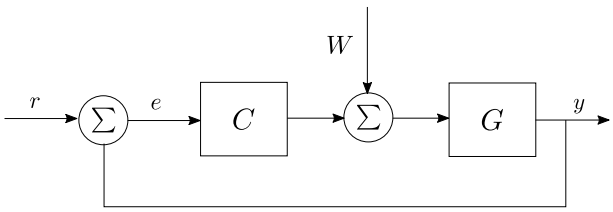

Figure 6.15: Feedback loops including input disturbance.

A transfer function for the closed loop from disturbance to output is expressed as

$$H_{\mathrm{P}} = \frac{G}{1+CG} = \frac{\frac{K}{s}}{1+K_C\frac{K}{s}} = \frac{K}{s+K_CK} \tag{6.25}$$

The DC gain can be expressed as

$$\lim_{s\to 0} H_{\mathrm{P}} = \lim_{s\to 0} \frac{K}{s+K_CK} = \frac{K}{K_CK} \tag{6.26}$$

As an steady state error is present, an integrator is added to the controller.

$$H_{\mathrm{PI}} = \frac{G}{1+CG} = \frac{\frac{K}{s}}{1+\frac{K_C}{s}\frac{K}{s}} = \frac{Ks}{s^2+K_CK} \tag{6.27}$$

The DC gain can thereby be given as:

$$\lim_{s\to 0} H_{\mathrm{PI}} = \lim_{s\to 0} \frac{Ks}{s^2+K_CK} = 0 \tag{6.28}$$

It is thereby revealed from Equation 6.28 that the steady state error is eliminated.

An integrator is therefore included in the control design. Since there already exists an integral term in the plant, the closed loop system becomes marginally unstable since the branches in the root locus are located on the imaginary axis. In order to move them into the left half plane, a zero at -0.05 is added to the controller. Then, the gain is adjusted to reduce the oscillations. The zero placement keeps the required bandwidth and enables the possibility of a non oscillating behavior. The final root locus can be seen in Figure 6.16.

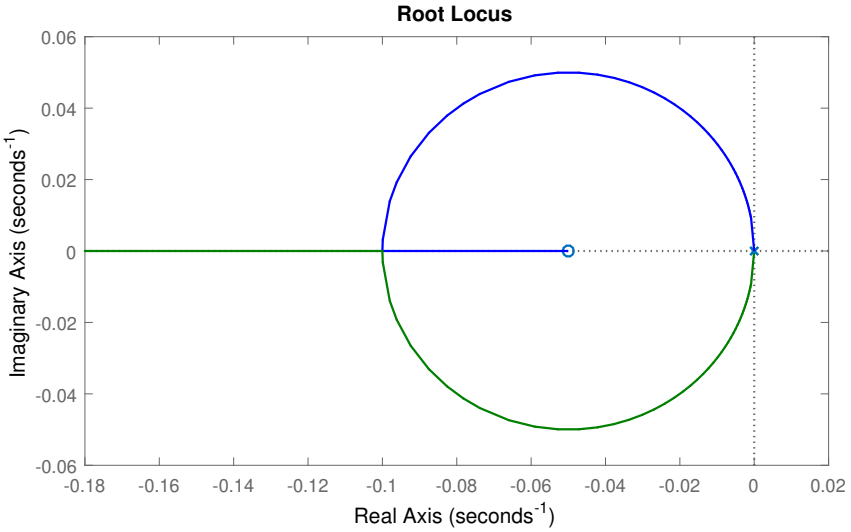

Figure 6.16: Root locus including the plant and the controller for the x_{I} translational velocity.

The Bode plot in Figure 6.17 shows that the final bandwidth of the closed loop system is at the desired 0.7 rad s^{-}1.

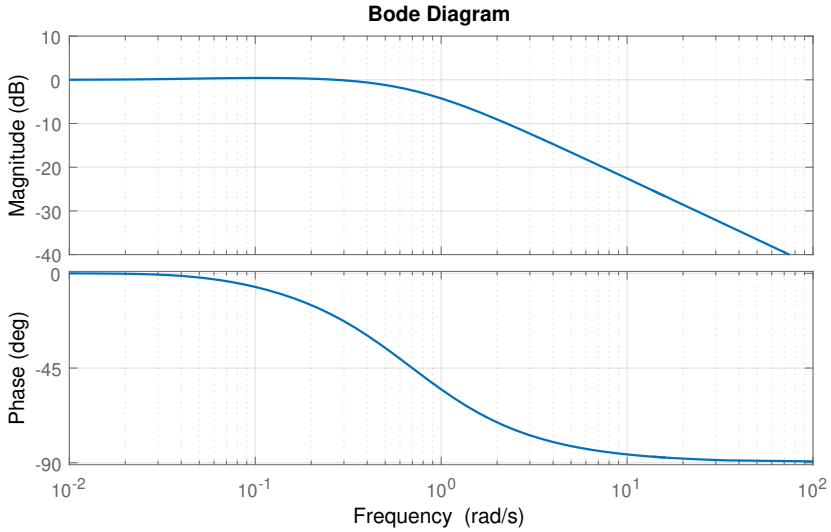

Figure 6.17: Closed loop Bode of the plant and the controller for the x_I translational velocity.

The final expression for the controllers for x_I and x_I translational velocity control is written as Equation 6.29 and 6.30, respectively. Note, that the plant for the x_I controller has already a positive gain and this makes it unnecessary to include a negative sign in its controller.

$$C_{\dot{x}_\mathrm{I}} = -0.0038\frac{1+20s}{s} \tag{6.29}$$

$$C_{\dot{y}_\mathrm{I}} = 0.0038\frac{1+20s}{s} \tag{6.30}$$

Where:

$C_{\dot{x}_\mathrm{I}}$	is the controller for the translational velocity in x_I direction	[rad m^{-1} s]
$C_{\dot{y}_\mathrm{I}}$	is the controller for the translational velocity in y_I direction	[rad m^{-1} s]

Once the velocity controllers are designed, the position controllers for x_I and y_I directions can be designed by using the same procedure. In this case, the position transfer function is just an integrator as seen in Equation 6.31 and 6.32.

$$G_{x_\mathrm{I}} = \frac{x_\mathrm{I}}{\dot{x}_\mathrm{I}} = \frac{1}{s} \tag{6.31}$$

$$G_{y_\mathrm{I}} = \frac{y_\mathrm{I}}{\dot{y}_\mathrm{I}} = \frac{1}{s} \tag{6.32}$$

Where:

G_{x_I}	is the plant for the translational position in x_I direction	[m s m^{-1}]
G_{y_I}	is the plant for the translational position in y_I direction	[m s m^{-1}]

A proportional controller is considered sufficient to control the position of the quadcopter in the x direction as there is an integrator in the plant and the input disturbances are considered in the inner loop.

The proportional gain is chosen so that the bandwidth of the position loop is three times lower than that of the velocity loop. This defines it to be around 0.23 rad s^{-}1. To set this bandwidth, the Bode plot of the plant shown in Figure 6.18 is used. The distance between the gain curve and the 0 dB level at 0.23 rad s^{-}1 sets the gain of the controller to be 0.3.

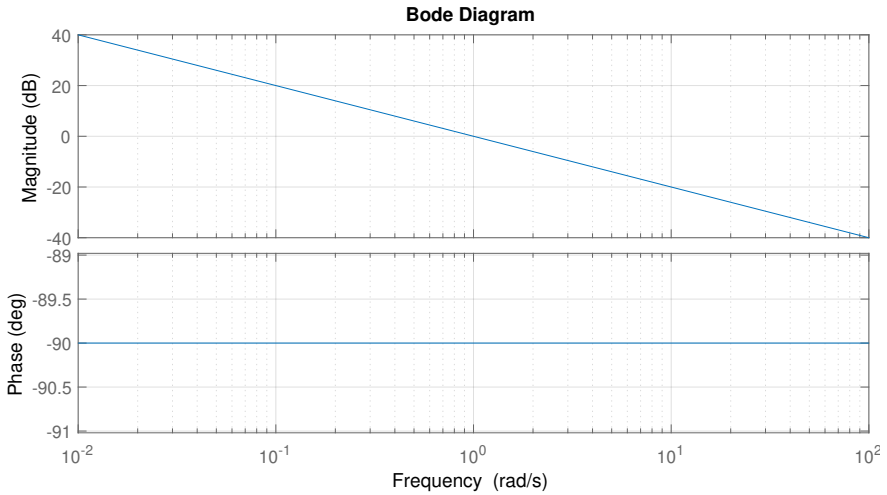

Figure 6.18: Position open loop transfer function Bode diagram for the translational x_I direction.

The final expression for the translational position controllers can be seen in Equation 6.33 and 6.34. In this case, both x_I and y_I controllers are equal.

$$C_{x_\mathrm{I}} = 0.3 \tag{6.33}$$

$$C_{y_\mathrm{I}} = 0.3 \tag{6.34}$$

Where:

C_{x_I}	is the controller for the translational position in x_I direction	[m s^{-1} m^{-1}]
C_{y_I}	is the controller for the translational position in x_I direction	[m s^{-1} m^{-1}]

The translational controllers for the inertial positioning in x_I and x_I have designed. It is thereby possible to simulate the controllers.

Simulation of the x_I and y_I Controllers

Simulations shown in this section include the position and velocity x_I controller, as the y_I is identical. The simulations presented depict the response of the controller subjected to a step input reference signal and a corresponding simulation showing the related control action of the controller.

In Figure 6.19 the translational velocity controller, $\dot{x}_\mathrm{I}$, is subjected to a step input reference signal of $1\,\mathrm{ms}^{-1}$ at time $0\,\mathrm{s}$.

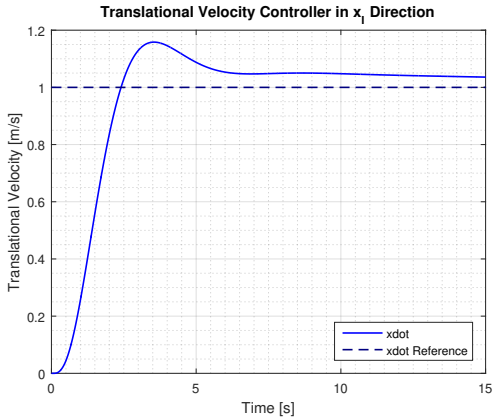

Figure 6.19: Translational velocity controller step response. The control system is subjected to a step input reference signal of $1\,\mathrm{ms}^{-1}$ in the x_I direction at time $0\,\mathrm{s}$.

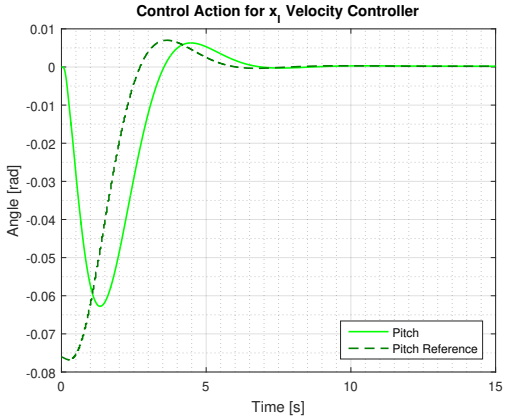

Figure 6.20: Control action required to achieve the velocity response in Figure 6.19, together with the the pitch response of the inner attitude controller.

This yields a settling time of approximately $6\,\mathrm{s}$, when considering an error band of 5 %. The overshoot is 15 %. In Figure 6.20 the corresponding control action is shown.

In Figure 6.21, the translational velocity controllers, x_I, is subjected to a step input reference signal of $1\,\mathrm{ms}^{-1}$ at time $0\,\mathrm{s}$.

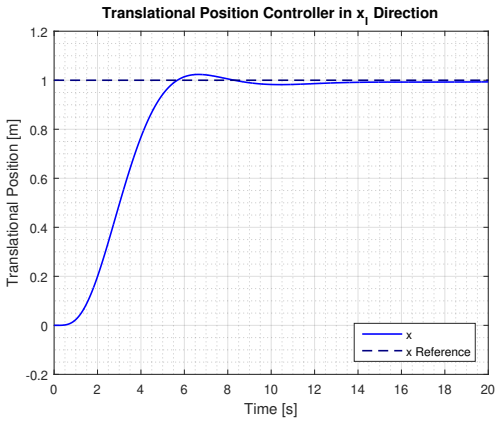

Figure 6.21: Step response of the position controller. The system is subjected to a step input reference signal of $1\,\mathrm{m}$ in the x_I direction, at time $0\,\mathrm{s}$.

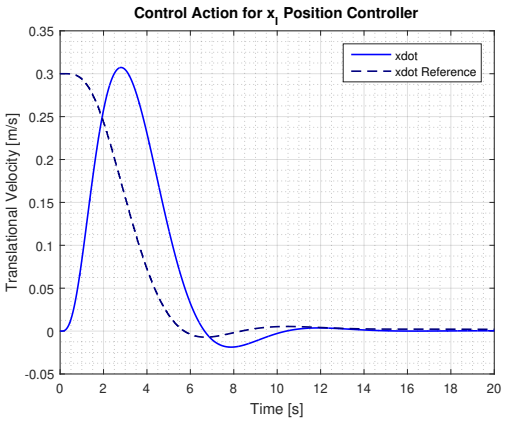

Figure 6.22: Control action required to achieve the position response in Figure 6.21, together with the response of the inner velocity controller.

It yields a settling time of approximately 5 s and a overshoot of 2 %. As the position controllers are designed to have a closed loop bandwidth that is three times lower than the velocity bandwidth, a larger settling time is expected. However, as the overshoot is small compared to the velocity controller, the settling time becomes lower.

The controller for the translational velocity and positioning in x_I and y_I direction have been designed and simulated. The z_I controller is to be designed in the following section.

6.2.2 Controllers for z_I Direction

The controller along the z_I direction is designed using a cascade approach.

The transfer function used to control the velocity is shown in Equation 4.38. It is obtained from the translational linearized block diagram seen in Figure 4.8. The sum of the motor rotational speeds is the input and the velocity in the z_I is the output.

$$G_{\dot{z}_\mathrm{I}} = \frac{\dot{z}_\mathrm{I}}{\omega_\mathrm{sum}} = \frac{\frac{1}{4}(-2k_\mathrm{th})\overline{\omega}_\mathrm{sum}}{ms} \tag{6.35}$$

Where:

G_{z_I}	is the plant for the translational velocity in z_I direction	[m s^{-1} rad^{-1} s]
$\dot{z}_\mathrm{I}$	is the velocity in the z_I direction	[m s^{-1}]
ω_sum	is the sum of motor rotational speeds	[rad s^{-1}]
$\overline{\omega}_\mathrm{sum}$	is the sum of motor rotational speeds in equilibrium	[rad s^{-1}]

This plant has a pole in zero and a negative gain, which means that the root locus on increasing gain drives the system into the right half plane and makes it unstable. This implies a controller with negative gain. As in the case of the other translational controllers, an integral term is added to reduce the effect of input disturbances.

The root locus of the system, which contains two poles in zero, branches along the imaginary axis. To avoid oscillations, the two loci are attracted by the use of a zero, placed on the left real axis as seen in Figure 6.23.

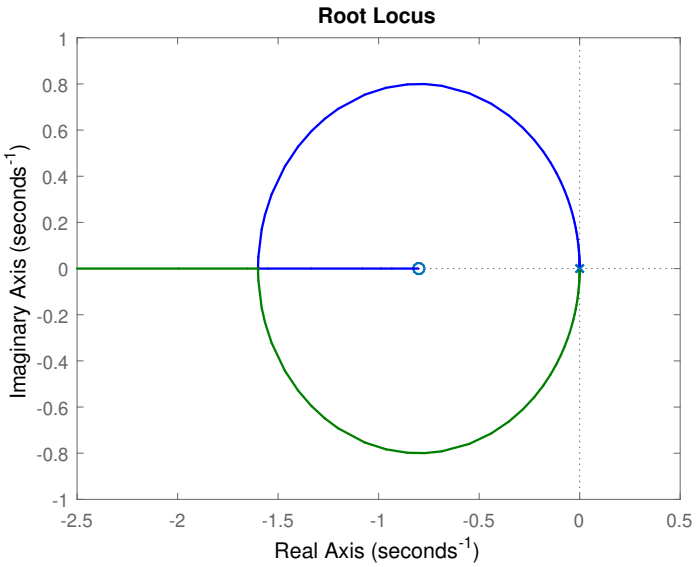

Figure 6.23: Root locus of the system with PI control. The zero is placed in $s = -0.8$

The zero is placed and the gain is scaled to achieve the smallest rise time without reaching the limits of the control action. The zero is placed in -0.8 on the real axis and the gain of the controller is -280.

The final expression for the velocity controller is

$$C_{\dot{x}_\mathrm{I}}(s) = -280\frac{s + 0.8}{s} \tag{6.36}$$

Where:

$C_{\dot{z}_\mathrm{I}}$	is the controller for the translational velocity in z_I direction	[rad s^{-1} m^{-1} s]

The position controller for z_I direction is designed using the same procedure as for x_I and y_I, since the plant is also an integrator.

$$G_{z_\mathrm{I}}(s) = \frac{z_\mathrm{I}(s)}{\dot{z}_\mathrm{I}(s)} = \frac{1}{s} \tag{6.37}$$

Where:

G_{z_I}	is the plant for the translational position in z_I direction	[m s m^{-1}]

The proportional gain is chosen so that the bandwidth of the position control loop is three times lower than that of the velocity loop. This is seen in the close loop Bode of the velocity control loop. This defines it to be 1 rads^{-1}. Since the plant already has this bandwidth, the gain is 1.

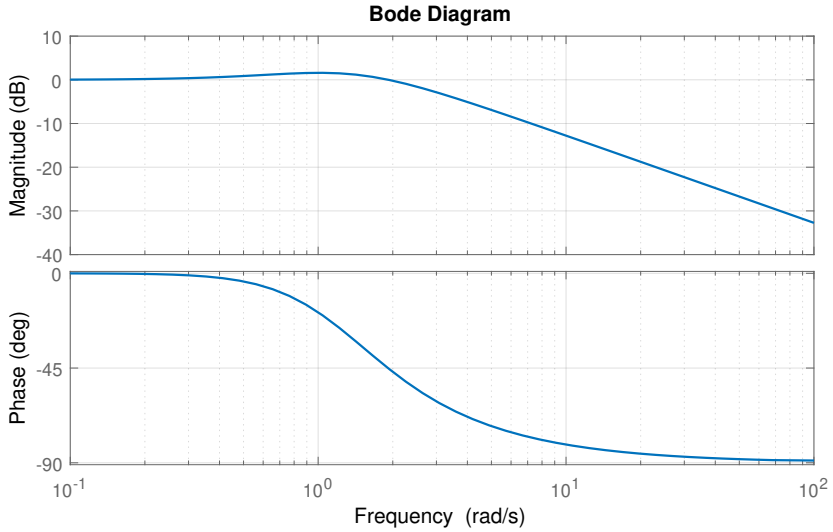

Figure 6.24: Closed loop Bode of the plant and the controller for the z_I translational velocity.

The expression for the z_I position controller is given as Equation 6.38.

$$C_{z_\mathrm{I}}(s) = 1 \tag{6.38}$$

Where:

C_{z_I}	is the controller for the translational position in z_I direction	$[\mathrm{m\ s^{-1}\ m^{-1}}]$

Simulation of the Controllers

The simulations show the behavior of the z_I velocity and position controllers when subjected to a step input reference signal and acting on the non-linear plant.

In Figure 6.25, a step reference of $1\,\mathrm{ms}^{-1}$ is given to the controller. This reveals a rise time of approximately 0.4 s, an overshoot of 16.4 % and a settling time of 2.5 s.

The control action required by the z_I velocity controller is shown in Figure 6.26. This is the sum of rotational speeds in the motors, which is added to the equilibrium speeds to achieve the response in Figure 6.25.

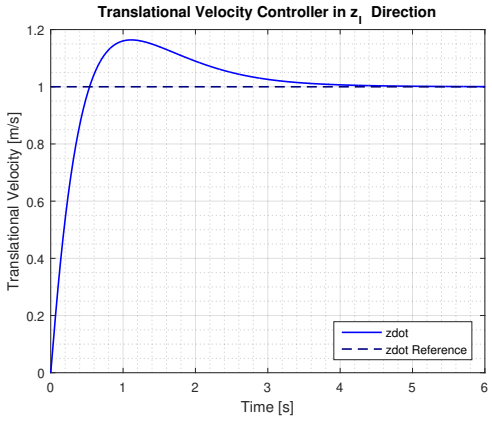

Figure 6.25: Step response of the z_I velocity controller.

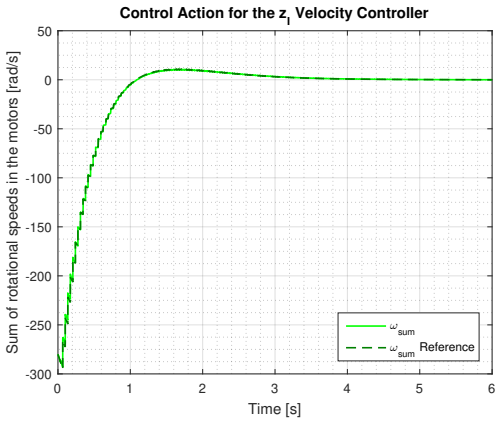

Figure 6.26: Control action required by the z_I velocity controller to achieve the response in Figure 6.25.

The z_I position controller response is seen in Figure 6.27. A step reference of 1 m is imposed on the controller. This reveals a rise time of 1.25 s and a settling time of 1.8 s. In Figure 6.28, the control action imposed by the position controller is shown. This is the translational velocity, which is denoted "zdot Reference" in Figure 6.28. The velocity applied by the velocity controller is "zdot".

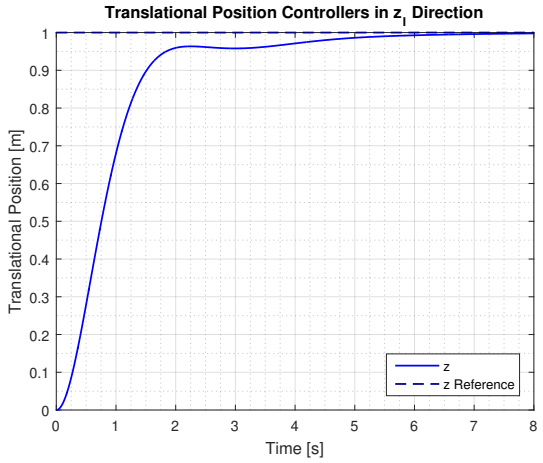

Figure 6.27: Step response of the z_I position controller.

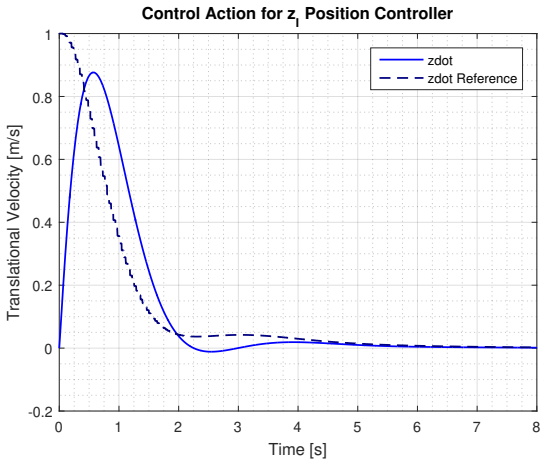

Figure 6.28: Control action imposed by the z_I position controller to track the step reference and the performance of the inner velocity controller seen in Figure 6.27.

7 | Implementation

Throughout the design process, the delay in the network has been analyzed, a protocol has been created for the communication and controllers for the attitude, translational velocity and position have been designed. From this the scheduling, being a critical part of the implementation, can be considered. The scheduling handles the timing of receiving and decoding packets while also servicing the control algorithms. For this task a real time operating system is integrated as the base for implementation in the microcontroller.
To implement the designed controllers, currently described in the continuous domain, they must be transformed to the discrete frequency domain.

In the following section the scheduling is described, followed by a section on implementation of the communication task, the discretization of continuous controllers and a final section describing the implementation of the controllers on the microcontroller.

7.1 Scheduler

Only two tasks are running on the microcontroller, the control algorithm and the communication. The handling of these tasks is however critical. The Vicon system frequency sets a limit for the rate at which the control algorithms can be executed. It is also desirable for the control task to be executed periodically, and at any other time, the microcontroller should listen for new data on the network. Furthermore, the transmission and the control task are not synchronized.
A scheduler can handle priorities and timing of tasks, for this purpose a real time operating system (RTOS) is used, FreeRTOS [30]. The capabilities of this RTOS ranges beyond the needs in this project, but provides the necessary tools, and constitutes a flexible base with room for future development.

In Listing 7.1 the two main tasks are created using `xTaskCreate()` and the task scheduler is started with `vTaskStartScheduler()`.

```
1  int main()
2  {
3      ...
4      // xTaskCreate(FunctionForTask,DebugName,AllocatedStackDepth,Parameters,Priority,↩
           TaskHandle)
5      // Control Task -> Highest priority
6      xTaskCreate(Controllers, "Control", 1000, NULL, configMAX_PRIORITIES - 1, NULL );
7      // Communication Task
8      // xHandle is used for resetting the task if it gets preempted by the Control Task
9      xTaskCreate(Communication, "Com", 1000, NULL, configMAX_PRIORITIES - 2, &xHandle);
10     ...
11     // Scheduler Start
12     vTaskStartScheduler();
13     return 0;
14 }
```

Listing 7.1: Code for initialization, creation of the different tasks, start sequence for the motors and call to the scheduler.

Once the task scheduler is started the program never returns to main again. The scheduler has a configurable tick-rate, set to 1 ms, which it uses for scheduling and timing tasks. The RTOS is set up to run with preemption, such that the higher priority task, `Control`, can preempt the lower priority, `Com`, in order to achieve a periodic execution of the control algorithms. The handle `xHandle` is used to resets the communication task if it is preempted while reading data. Since the XBee always sends data, whenever ready, out on the RX-pin of the microcontroller, data will be lost if the communication is preempted while data is on RX. In Figure 7.1 an example of task occurrences in the microcontroller is shown. In this case, the preemption of the communication task happens at 105 ms.

To make sure that there is always new data available for each execution of the control task, the execution frequency is chosen low enough such that at least one package is received in each period.

At 35 ms in Figure 7.1, the data decoding is preempted by the control task. Since the data is already stored, decoding the package is continued by the scheduler after servicing the control task, and the data will be ready for use at the next control cycle.

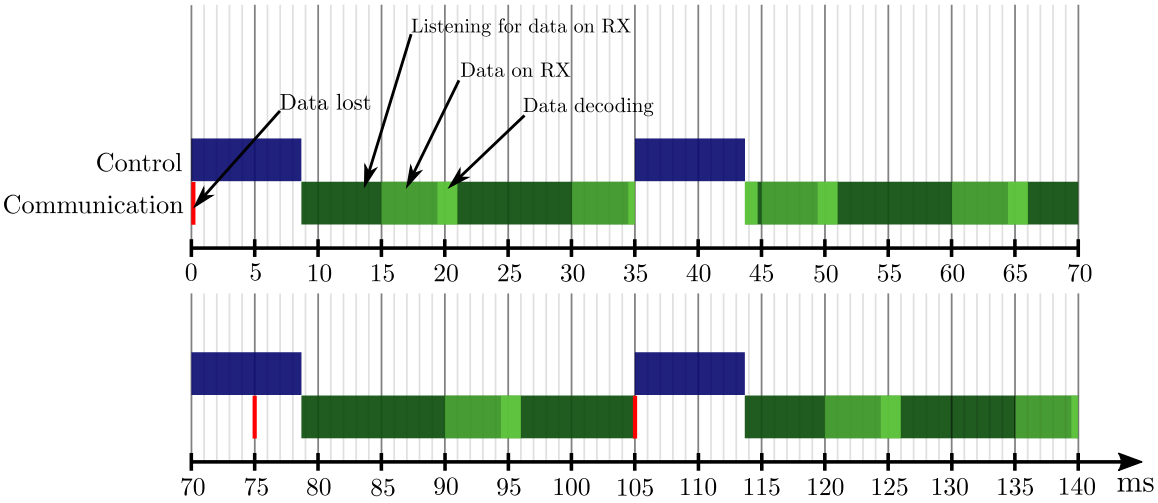

Figure 7.1: An example of task execution schedule. The control task is periodic and the packages can arrive at different instances.

Although the packages are received at instances unrelated to the control task, the packages still arrive periodically, though the period may fluctuate.

7.2 Communication

The communication task is used to receive the data in the microcontroller from the computer. Part of the code used is seen in Listing 7.2.

```
1  void Communication(void *pvParameters)
2  {
3      int pack = 0;
4      while (1)
5      {
6          pack=0;
7          //CheckPackageArrival() returns true if a valid header is detected
8          pack = CheckPackageArrival();
9          if (pack)          // Check if a valid package arrived
10             GetPackage(); // This function decodes the packet.
11     }
12     vTaskDelete(NULL);
13 }
```

Listing 7.2: Code for the comunication task.

It consist of a while loop that is running all the time and checking if a packet has arrived using the function `CheckPackageArrival()`. This function read the bytes coming to the serial port and checks if the header is correct, returning a 0 if that is not the case. If the header is wrong, the if-condition is not fulfilled and the loop starts again until it gets a correct header.

Then, the function `GetPackage()` reads the remaining bytes and uses the checksum to verify that the data has been sent correctly. When the summation of the parts and the checksum gives the correct value, the stored bytes are decoded and the global variables are rewritten with their new values.

7.3 Controllers

To implement the controllers on the microcontroller they must first be transformed into the discrete domain.

This transformation is done through the Tustin (bilinear) approximation in which the s term in the transfer function is substituted as seen in Equation 7.1 [31].

$$s \approx \frac{2}{T}\frac{z-1}{z+1} \tag{7.1}$$

Where:

s	is the Laplace operator
z	is the equivalent of the Laplace operator in discrete domain

T is the sampling period [s]

The Tustin method maps the Laplace stable region into the discrete stable region, that is, the unit circle, as seen in Figure 7.2.

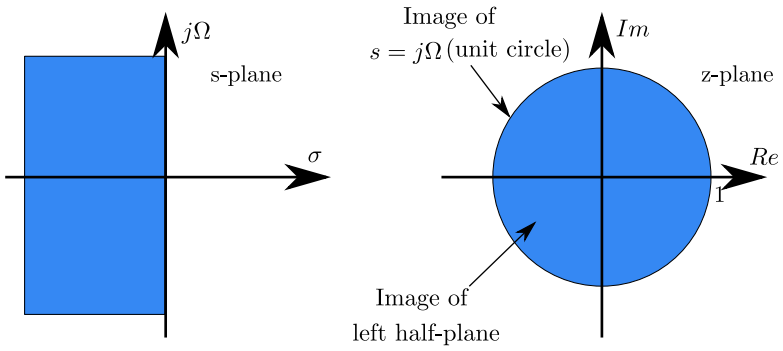

Figure 7.2: The s-domain and the z-domain [32].

The discrete transfer function is then transformed into a difference equation to obtain the expression to be applied in the microcontroller. This is done taking into account that a z^{-1} term implies taking the previous sample of the data.

When discretizing the controller, the sampling time must also be considered. In the system at hand, the sampling frequency is limited by the sensor data (Vicon System) to be 100 Hz as a maximum. To ensure that the discretization does not affect the controller response, the lower sampling limit is set taking into account the bandwidth of the controllers. To do so, the sampling frequency should be between 10 and 20 times higher than the bandwidth of the control loop. The fastest control loop in the system is the attitude controller, which has a bandwidth of 2 rad $\cdot$ s^{-1}, that is, 0.32 Hz. This implies a minimum sampling frequency of 3.2 Hz.

The sampling period is chosen to be 0.035 ms, which is the fastest possible within the schedule as described in section 7.1.

7.3.1 Attitude Controller

The attitude controller is mainly composed of gains formed by the state feedback, the integral and the observer matrices. This makes the discretization easier as there are no s-terms in the controller. There are however several integrators, three of which are in the observer and the remaining three in the integral part of the controller.

The discrete form of an integrator using the Tustin approximation is shown in Equation 7.2. The formula shows the discretized version of the integrator in the integral term

of the attitude control. The first relation in Equation 7.2 is obtained from Figure 6.5.

$$\frac{x_{\text{int}}}{e} = \frac{1}{s} \approx \frac{T}{2}\frac{z+1}{z-1} \tag{7.2}$$

This transformation yields a difference equation as seen in Equation 7.3. It gives the current value of the integral state as a function of the previous integral state, the current and the previous error between the angular reference and the angular data.

$$x_{\text{int}}(k) = x_{\text{int}}(k-1) + \frac{T}{2}e(k) + \frac{T}{2}e(k-1) \tag{7.3}$$

7.3.2 Translational Controllers

The translational position controllers contain only proportional terms, which means that no discretization is needed. However, the velocity controllers also include an integral term and therefore they need to be discretized. This is also done using the Tustin approximation, same as for the attitude controller.

The discrete versions of the PI controllers are

$$\theta_{\text{ref}}(k) = \theta_{\text{ref}}(k-1) - 0.08e_{\dot{x}}(k) + 0.08e_{\dot{x}}(k-1) \tag{7.4}$$
$$\phi_{\text{ref}}(k) = \phi_{\text{ref}}(k-1) + 0.08e_{\dot{y}}(k) - 0.08e_{\dot{y}}(k-1) \tag{7.5}$$
$$\omega_{\text{sum}}(k) = \omega_{\text{sum}}(k-1) - 283.9e_{\dot{z}}(k) + 276.1e_{\dot{z}}(k-1) \tag{7.6}$$

7.3.3 Controller Code

As in the case of the communication, the controllers are included in a task that runs every sampling period, $T = 35$ ms. As seen in Listing 7.3 this is done by storing the time at which the task starts in the variable `xLastWakeTime`. This is then used in the function `vTaskDelayUntil()` to count 35 ms from `xLastWakeTime` before the control task is run again. For each period the control code is executed and the calculated control actions are send to the motor controllers in the form of a PWM signal.

```
1  void Controllers(void *pvParameters)
2  {
3      // Variable to make the task periodic
4      portTickType xLastWakeTime;
5      // xTaskGetTickCount() returns start time of the task
6      xLastWakeTime = xTaskGetTickCount();
7
8      while (1)
9      {
10         Controller();
11         ApplyVelocities();
12         // A delay of 35 ms is set since the task started until the next task
```

```
13          vTaskDelayUntil(&xLastWakeTime, 35);
14      }
15      vTaskDelete(NULL);
16  }
```

Listing 7.3: Code for the controller task.

The implementation of the difference equations inside the function `Controllers()` is seen in Listing 7.4, in which three examples , $\dot{z}_I$ controller, integration in the state space controller and in the observer, are presented.

```
1  ...
2  r_k[1] = -0.08*vel_e_k[0] + 0.08*vel_e_k1[0] + r_k[1]; // x velocity controller
3  ...
4  r_k[0] = 0.08*vel_e_k[1] - 0.08*vel_e_k1[1] + r_k[0]; // y velocity controller
5  ...
6  u_z = -283.9*vel_e_k[2] + 276.1*vel_e_k1[2] + u_z; // z velocity controller
7  ...
8  xint_k[i] = T / 2 * (e_k[i] + e_k1[i]) + xint_k1[i]; // Integral control integrator
9  ...
10 oint_k[i] = T / 2 * (o_k[i] + o_k1[i]) + oint_k1[i]; // Observer integrator
11 ...
```

Listing 7.4: Code for the controllers.

The function `ApplyVelocities()`, seen in Listing 7.3, is in charge of mixing the control action of the attitude controller with the one coming from the z_I controller. Then the duties that correspond to the desired rotational speeds of the motor are calculated using the information of Appendix C and the PWM signal is sent to the motor controllers.

Part III

Test & Closing Statements

8 | Functionality Tests

The capabilities of the design carried out in this project have been tested in order to asses them in relation to the functional requirements stated in chapter 3. For the sake of convenience, these are stated below.

1) The quadcopter should be able to receive its own position and attitude from the Vicon system, through a computer at the ground station.

2) It shall be possible to control the quadcopter's attitude.

3) It shall be possible to control the quadcopter's position in the z axis.

4) It shall be possible to control the quadcopter's position in the x and y axes.

8.1 Communication Test

The designed protocol and the scheduling have been tested following the procedure below.

1. Transmit 10000 packets from MATLAB to the microcontroller.
2. Implement the program in the microcontroller so two counters are enabled when the first of the 10000 packets arrives. One of the counters should count control loops and the other should count correct decoded packets.
3. When the last packet arrives both counters should be stopped and their values should be transmitted back to the computer.
4. The comparison between the number of control loops executed and correctly decoded packets gives an estimate of the probability of missed packets.

Results:

The obtained results are presented in Table 8.1.

Controller loops executed	Packets correctly decoded
4468	4723
4477	4651
4354	4484

Table 8.1: Results from the communication test, showing the number of control loop executed and packets correctly decoded when sending 10000 packets are sent.

It is seen that, out of 10000 packets, less than half of them are correctly decoded in the microcontroller. This happens due to the control task having higher priority than the communication task. When a control loop is being executed, the microcontroller does not listen for new messages. However, missing these packets does not affect the performance as long a new packet arrives in the listening period.

The test reveals that more packages are correctly received and decoded than control loops are executed. This suggests that the vast majority of control loops run with the most recent information and that all packets received in the listening period are received and decoded correctly.

8.2 Control the Quadcopter's Attitude

Three tests have been carried out to test the capabilities of the attitude controller. The first test shows how the controller can stabilize the attitude around equilibrium when starting with initial conditions different from zero in the three angles. The second and the third tests are done in order to test the reference tracking in roll and pitch, respectively.

The procedure followed to test the attitude controller in the quadcopter is the following.

1. Attach the quadcopter to the setup as seen in Appendix D.
2. Program the code for stabilization of the quadcopter in the microcontroller.
3. Start the transmission of attitude data from MATLAB to the quadcopter.
4. Switch on the battery for the quadcopter.
5. Record the attitude data in MATLAB.
6. Program the code for tracking references and repeat repeat from step 3 to 5. This is done for both roll and pitch.

Results:

The results of the first test can be seen in Figure 8.1, where the attitude is stabilized.

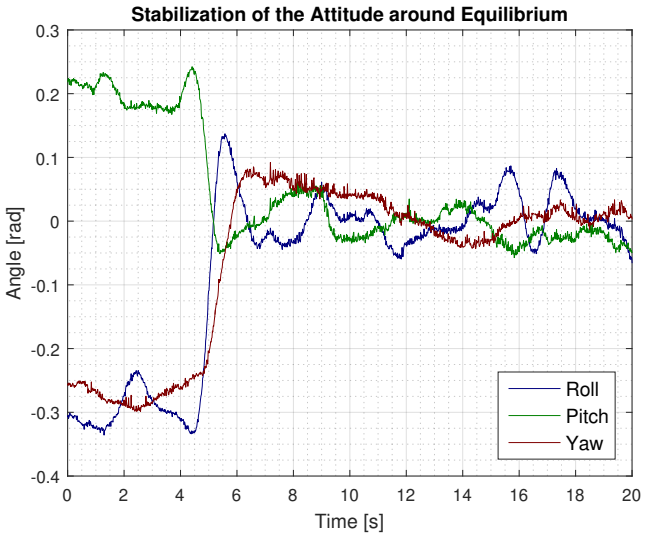

Figure 8.1: Test of the controller ability to stabilize the quadcopter around equilibrium.

The three angles start with an initial condition, -0.2 rad for roll, 0.3 rad for pitch and -0.2 rad for yaw. After approximately five seconds, all the references are set to zero. It can be seen that the roll and pitch angles converge to zero in less than 2 s and the yaw angle in approximately 7 s. From the figure, it can be seen that oscillations of approximately 0.04 rad occur around the stabilization point of 0 rad. These oscillations likely originate from dynamics of the real system not captured by the model, as the utilized parameters for the model could be different from those of the real system. Effects imposed by the test setup are considered in Appendix D.

The response of the controller when tracking a reference in roll is seen in Figure 8.2.

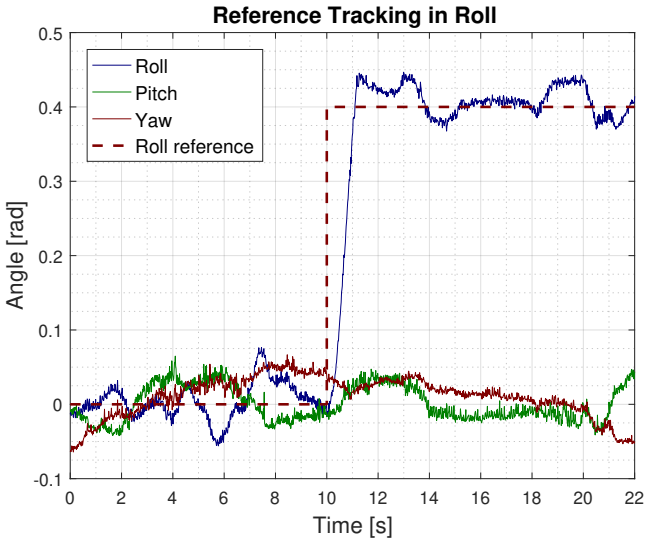

Figure 8.2: Attitude controller response when tracking a pitch angle reference of 0.4 rad.

It can be seen that the reference is tracked and there is no steady state error, there are oscillations of around 0.04 rad also seen in the stabilization of the quadcopter seen in Figure 8.1.

In Figure 8.3 a reference of 0.4 rad in pitch is given to the attitude controller.

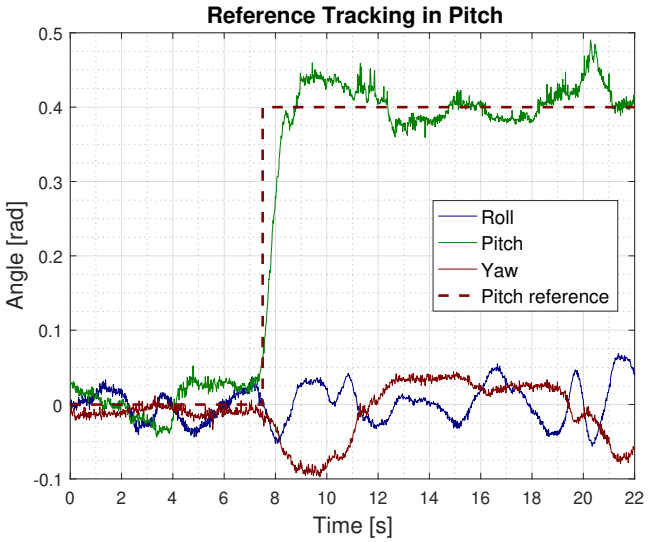

Figure 8.3: Attitude controller response when tracking a pitch angle reference of 0.4 rad.

From the figure, it can be seen that it is possible for the controller to track a change in reference. This is achievable while the roll and yaw angles remain at zero. The oscillations seen around the reference are approximately 0.04 rad. This is consistent with the behavior seen in Figure 8.1.

8.3 Position Control in the z Axis

Experimental tests reveal a non-functional z_{I} controller implementation. However, the design has been tested in simulation, including the network effects, as depicted in section 6.2.

8.4 Position Control in the x and y Axes

As in the case of the previous requirement, experimental tests reveal a non-functional x_{I} and y_{I} controllers implementation. However, it is believed that the designed is valid as it has been verified in simulations as depicted in section 6.2.

9 | Discussion

The main limiting effects in the control design are the delays and the maximum frequency in which packets can be transmitted to the microcontroller. These two factors make the attitude control loop's bandwidth low. The translational position and velocity controllers are also affected as the bandwidths of these need to be lower than the bandwidth of the attitude loop. This issue can be solved by implementing on board sensors to obtain the attitude of the quadcopter. In this way, the control loop for the attitude controller could run faster and the delays would be smaller, thus allowing a faster response of the system.

From the tests performed, it has been observed that the translational controllers do not perform as expected in the real prototype. The most probable cause for the problem is an implementation error or a hardware issue that is yet to be identified. It has been showed that the design approach resulted in working translational controllers as the simulations yield successful results.

The way of modeling the system is deemed suitable for the purpose of controlling the quadcopter. A more detailed model could yield a more reliable simulation when testing the controllers.

The used microcontroller runs at 16M IPS. This is deemed sufficient but it is possible to find microcontrollers with better performance. A higher execution rate or a device with parallel capabilities such as a FPGA could yield a better performance of the system.

As mentioned in subsection 2.1.2, the motor controllers do not take into account the voltage level of the battery when running. This constitutes a disturbance for the controllers as the rotational speed demanded is not attained by the motors. This issue does not stop the controllers from working, but the performance will be less consistent in sustained flight. It could be solved by measuring and accounting for the battery voltage when setting the duty reference for the motor controllers. Another solution could be to use different motor controllers with continuous battery level compensation.

10 | Conclusion

The focus of this project has been to design a linear control system for the attitude and position of a quadcopter when using an external motion tracking system as sensor.

Since this feature constitutes an networked distributed system, time delays are introduced on the received data, affecting the stability of the quadcopter.

The system's dynamics have been modeled by first principles modeling describing the relation between the rotational speed of the motors and the thrust and drag torques, the attitude behavior and the translational behavior. Based on the model, a linear control system has been designed to control the attitude and position of the quadcopter. The control system has been split into attitude and translational controllers, which have been designed considering the main network effects. The attitude controller has been designed using a state space approach, including state feedback with integral control using LQR and a reduced order observer. The translational control system has been designed by means of classical control methods, yielding a cascaded structure including P and PI controllers.

The results show that the design for both the attitude and the translational behavior is able to control the quadcopter in simulation. Moreover, the implementation and tests of the attitude controller have been carried out on the quadcopter successfully.

Bibliography

[1] *Types of Multicopter*. URL: https://oscarliang.com/types-of-multicopter/ (visited on 08/12/2016).

[2] *Tricopter, Quadcopter, Hexacopter & Octocopter Explained*. URL: https://www.dronersguides.com/what-is-a-quadcopter/ (visited on 08/12/2016).

[3] Daobo Wang Ashfaq Ahmad Mian Mian Ilyas Ahmad. 'Backstepping based Nonlinear Flight Control Strategy for 6 DOF Aerial Robot'. In: *International Conference on Smart Manufacturing Application* (2008).

[4] S. McGilvray A. Tayebi. 'Attitude stabilization of a four-rotor aerial robot'. In: *43rd IEEE Conference on Decision and Control* (2004).

[5] *MT2213-935KV MultiStar Motor and Propeller Combo 10x4.5 CW/CCW*. URL: https://hobbyking.com/en_us/mt2213-935kv-multistar-motor-and-propeller-combo-10x4-5-cw-ccw.html.

[6] *Turnigy Multistar 30 Amp Multi-rotor Brushless ESC 2-4S (US Warehouse)*. 31 Oct. 2016. URL: https://hobbyking.com/en_us/turnigy-multistar-30-amp-blheli-multi-rotor-brushless-esc-2-6s-v2-0.html.

[7] Edgar Reyes. *What is Propeller Pitch?* 8 Mar. 2006. URL: http://www.propellerpages.com/?c=articles&f=2006-03-08_what_is_propeller_pitch (visited on 01/12/2016).

[8] *How to choose Motor and Propeller for Quadcopter*. URL: https://oscarliang.com/quadcopter-motor-propeller/ (visited on 01/12/2016).

[9] *Turnigy 2200mAh 3S 25C Lipo Pack*. URL: https://www.amazon.com/Turnigy-2200mAh-20C-Lipo-Pack/dp/B0072AEY5I/ref=sr_1_3?ie=UTF8&qid=1482153231&sr=8-3&keywords=Turnigy+2200mah+3s+25c+Lipo+Pack (visited on 06/12/2016).

[10] *Turnigy 2200mAh 3S 25C Lipo Pack*. URL: http://www.robosoftsystems.co.in/roboshop/media/catalog/product/cache/2/image/500x500/9df78eab33525d08d6e5fb8d27136e95/t/2/t2200-3-20.jpg (visited on 19/12/2016).

[11] *Vicon DataStream SDK Manual*. Found on Attachment. 1 Jan. 2013.

[12] *Atmel 2549 8-bit AVR Microcontroller ATmega 640 1280 1281 2560 2561 datasheet*. Found on Attachment. 19 Feb. 2016. URL: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf (visited on 06/12/2016).

[13] *Arduino Board Mega 2560 A000067*. URL: https://www.conrad.at/de/arduino-board-mega-2560-a000067-191790.html (visited on 06/12/2016).

[14] *XBee/XBee-PRO RF Modules Product Manual*. 23 Sept. 2009. URL: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf (visited on 04/12/2016).

[15] *XBee 1mW Wire Antenna - Series 1*. URL: http://www.jayconsystems.com/xbee-1mw-wire-antenna-series-1-802-15-4.html (visited on 06/12/2016).

[16] *UartSBee V4*. 22 Aug. 2013. URL: http://wiki.seeedstudio.com/wiki/UartSBee_V4 (visited on 06/12/2016).

[17] *XBee Shield para Arduino*. URL: http://www.filipeflop.com/pd-6b60d-xbee-shield-para-arduino.html?ct=&p=2&s=1 (visited on 06/12/2016).

[18] *Schematic of UartSBee V4.0*. Found on Attachment. 22 Aug. 2013. URL: http://wiki.seeedstudio.com/images/d/d5/UartSBee_V4.0_Sch.png (visited on 06/12/2016).

[19] *Xbee Shield Schematic*. Found on Attachment. URL: https://www.arduino.cc/en/uploads/Main/XbeeShieldSchematic.pdf (visited on 06/12/2016).

[20] H. L. Chan and K. T. Woo. 'Design and Control of Small Quadcopter System with Motor Closed Loop Speed Control'. In: *International Journal of Mechanical Engineering and Robotics Research Vol. 4* (28 Aug. 2015). URL: http://www.ijmerr.com/uploadfile/2015/0921/20150921115356533.pdf (visited on 04/12/2016).

[21] Eric W. Weisstein. *Rotation Matrix*. MathWorld–A Wolfram Web Resource. URL: http://mathworld.wolfram.com/RotationMatrix.html.

[22] Dan Henriksson Anton Cervin et al. 'How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime'. In: *IEEE Control Systems Magazine* (2003).

[23] Abbas Emami-Naeini Gene F. Franklin J. David Powell. 'Feedback Control of Dynamic Systems'. In: 7th Edition. Pearson, 2015. Chap. 7, p. 455.

[24] Abbas Emami-Naeini Gene F. Franklin J. David Powell. 'Feedback Control of Dynamic Systems'. In: 7th Edition. Pearson, 2015. Chap. 7, p. 457.

[25] Jakob Stoustrup edited by Christoffer Sloth. *Multivariable Control, observabiliy, Observers, and observer Based Control*. Found on Attachment. 2016.

[26] Abbas Emami-Naeini Gene F. Franklin J. David Powell. 'Feedback Control of Dynamic Systems'. In: 7th Edition. Pearson, 2015. Chap. 7, pp. 453–585.

[27] Jakob Stoustrup edited by Christoffer Sloth. *Multivariable Control, Introducing reference signals, anti-windup, optimal control*. Found on Attachment. 2016.

[28] Abbas Emami-Naeini Gene F. Franklin J. David Powell. 'Feedback Control of Dynamic Systems'. In: 7th Edition. Pearson, 2015. Chap. 7, p. 515.

[29] Jakob Stoustrup edited by Christoffer Sloth. *Multivariable Control, Reduced Order Observers, Integral Control*. Found on Attachment. 2016.

[30] *FreeRTOS*. URL: http://www.freertos.org/ (visited on 15/12/2016).

[31] Abbas Emami-Naeini Gene F. Franklin J. David Powell. 'Feedback Control of Dynamic Systems'. In: 7th Edition. Pearson, 2015. Chap. 8, pp. 614–633.

[32] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. 2014.

Appendix

A | Thrust Force Coefficient

Name: Group 733
Date: 30/09 - 2016

Purpose

Finding the relation between the rotational speed of the motor and the thrust force generated by the propeller.

Setup

Figure A.1: Setup for the thrust test.

List of Equipment

Instrument	AAU-no.	Type
Tachometer	08246	Shimpo DT-205
Power Supply (11.1 V)	64565	ES 030-5
Processing Unit	-	Arduino Mega
Motor	-	Multistar 2213-935
Motor Speed Controller	-	-
Propeller	-	Turnigy 1045R
Scale	86759	KERN FCB 12K1

Procedure

1. Construct the setup as seen in Figure A.1, the power supply is connected to the motor driver and the Arduino Mega is powered from the computer. One PWM pin and GND pin from the board must be connected to the driver signal cables yellow and brown respectively.
2. Run the program. It should generate a fixed duty PWM signal in the PWM pin.
3. Wait for the speed to stabilize and read the scale value. The thrust force is calculated by multiplying the added mass in kilograms with the gravitational acceleration ($9,81\ \mathrm{m/s^2}$)
4. Measure the rotational speed with the tachometer.

Results

Speed [rpm]	**Speed [rad/s]**	**Added Mass [g]**	**Thrust Force [N]**
2240	234.57	69	0.68
2305	241.38	74	0.73
2445	256.04	85	0.83
2495	261.27	89	0.87
2585	270.70	97	0.95
2665	279.08	99	0.97
2811	294.36	114	1.12
2995	313.63	129	1.27
3195	34.57	150	1.47
3287	344.21	160	1.57
3493	365.78	182	1.79
3609	377.93	195	1.91
3765	394.26	215	2.11
3888	407.15	228	2.23
4060	425.16	250	2.46

Results

The obtained results are shown in Figure A.2. They have been approximated by a parabolic curve by finding a linear relation between the velocity squared and the thrust

force.

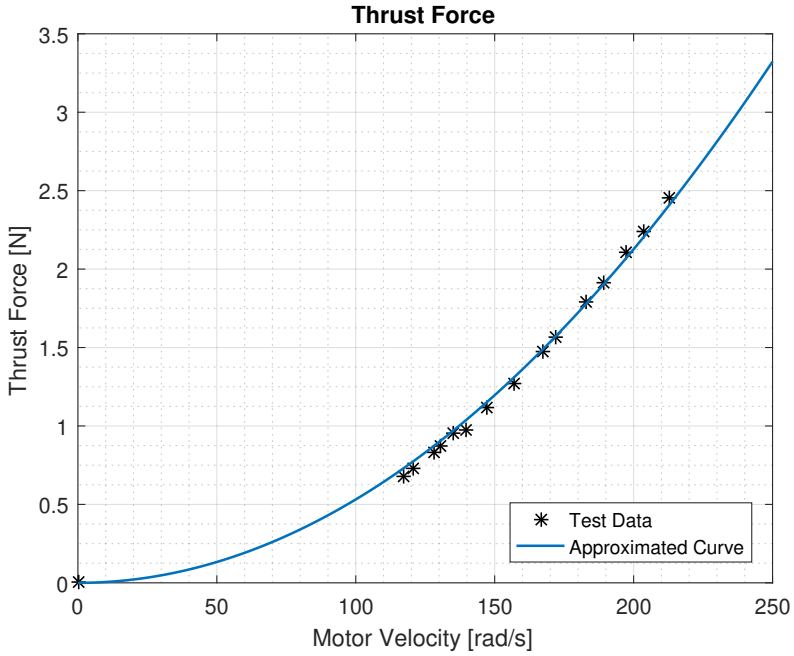

Figure A.2: Data from the thrust test approximated by a parabolic curve.

The resulting coefficient that provides the relation between the thrust force in the propeller and the velocity squared is $1.32922 \cdot 10^{-5} N \cdot s^2 \cdot rad^{-2}$.

B | Drag Torque Coefficient

Name: Group 733
Date: 04/10 - 2016

Purpose

Finding the relation between the rotational speed of the motor and the drag torque generated by the propeller.

Setup

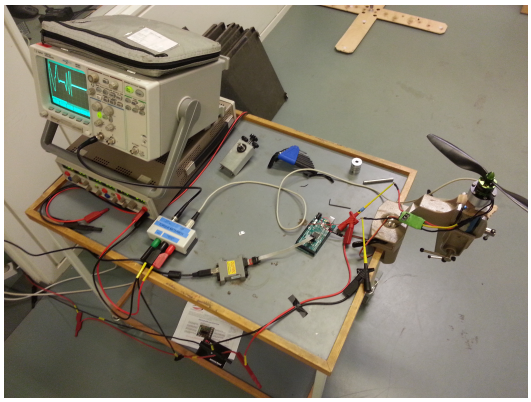

Figure B.1: Setup for the torque test.

List of Equipment

Instrument	AAU-no.	Type
Tachometer	08246	Shimpo DT-205
Power Supply (11.1 V)	64565	ES 030-5
Processing Unit	-	Arduino Mega
Motor	-	Multistar 2213-935
Motor Speed Controller	-	-
Propeller	-	Turnigy 1045R
Torquemeter	-	-
Oscilloscope	61604	Agilent 54621A
Torquemeter Power Supply	61598	HM7042-5

Procedure

1. Construct the setup as seen in *figure B.1*, the power supply is connected to the motor driver and the Arduino Mega is powered from the computer. One PWM pin and GND pin from the board must be connected to the driver signal cables yellow and brown respectively, The torquemeter is connected delivers the measurement as a voltage in the oscilloscope.

2. Run the program. It should generate a fixed duty PWM signal in the PWM pin.

3. Wait for the speed to stabilize and read the torque value as voltage in the oscilloscope. The torque is calculated by considering the the torquemeter specifications. They state that ±5 V are equivalent to ±1 Nm.

4. Measure the rotational speed with the tachometer.

Results

Speed [rpm]	**Speed [rad/s]**	**Torque [mV]**	**Drag Torque[Nm]**
3308	346.41	143.8	0.0288
3416	357.72	153.1	0.0306
3655	382.75	168.8	0.0338
3755	393.22	171.9	0.0344
3916	410.08	206.3	0.0413
4045	423.59	212.5	0.0425
4240	444.02	231.3	0.0463
4310	451.34	240.6	0.0481

The obtained results are shown in *figure B.2*. They have been approximated by a parabolic curve by finding a linear relation between the velocity in rad/s squared and the thrust force.

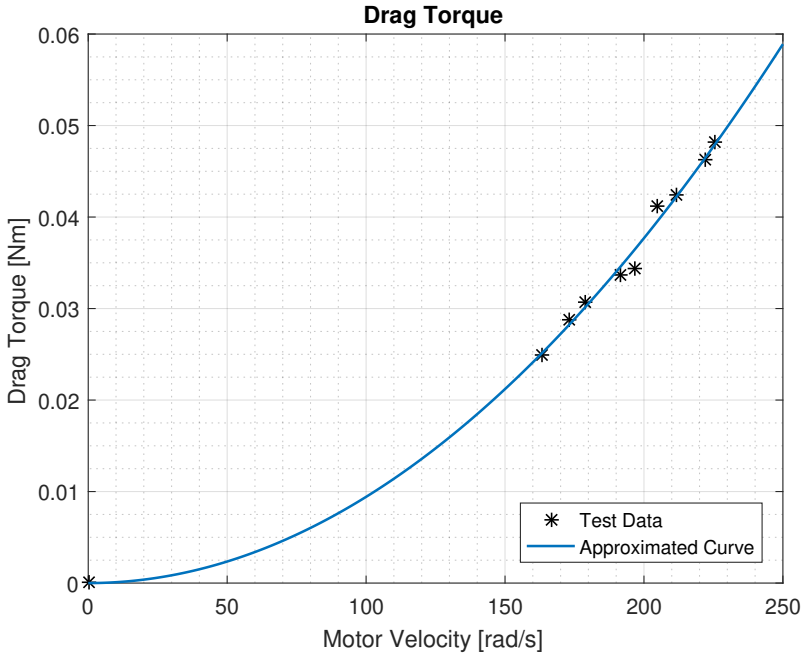

Figure B.2: Data from the torque test approximated by a parabolic curve.

The resulting coefficient that provides the relation between the drag torque in the propeller and the velocity squared is $9.39741 \cdot 10^{-7} Nm \cdot s^2 \cdot rad^{-2}$.

C | Duty-to-Speed Test

Name: Group 733
Date: 5/12 - 2016

Purpose

Finding the relation between the duty cycle sent to the motor controller and the rotational speed of the motors, to be able to select the appropriate duty once the control actions are calculated.

List of Equipment

Instrument	AAU-no.	Type
Tachometer	08246	Shimpo DT-205
Power Supply (11.1 V)	64565	ES 030-5
Processing Unit	-	Arduino Mega 2560
Motor	-	Multistar 2213-935
Motor Speed Controller	-	-
Propeller	-	Turnigy 1045R

Procedure

1. Connect the power supply to the motor controller and the Arduino Mega to the computer though the programmer. One PWM pin and GND pin from the board must be connected to the driver signal cables yellow and brown respectively.
2. Run the program with a fixed PWM duty for each test. The duty can be set from from 128 to 255.
3. Wait for the speed to stabilize and measure it with the tachometer.
4. Repeat with a different duty cycle.

Results

Duty	Motor Speed [rpm]	Motor Speed [rad/s]
160	2210	231.43
165	2554	267.45
170	2820	295.31
175	3107	325.36
180	3381	354.05
185	3740	391.65
190	4035	422.54
195	4345	455.01
200	4620	483.81
205	4882	511.24

Table C.1: Results obtained when applying a duty cycle from 160 to 205 to the motor controllers.

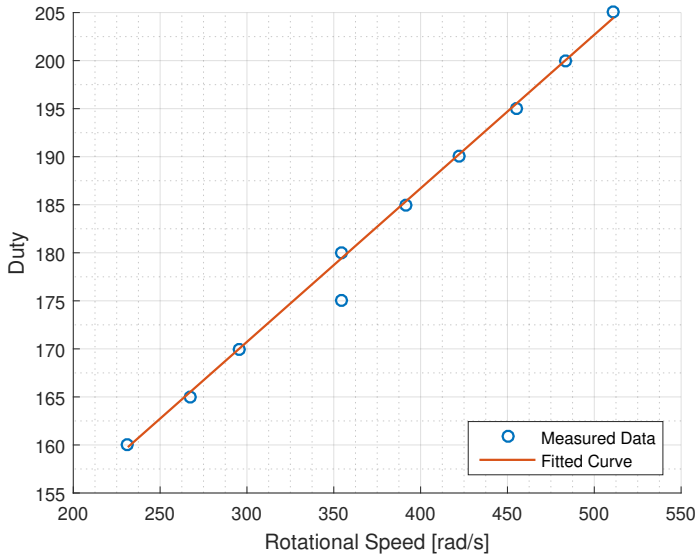

Figure C.1: Linear relation between rotational speed of the motors and duty sent to the ESCs.

The final equation to be able to set the needed duty depending on the required rotational speed of the motor, blue line in Figure C.1, is given by

$$\text{duty}_\text{i} = 0.1598\omega_\text{i} + 122.79 \tag{C.1}$$

D | Attitude Controller Test Setup

This appendix describes the equipment needed and the considerations needed when using the set up utilized to test he attitude controller on the quadcopter.

List of Equipment

Instrument	AAU-no.
Quadcopter	-
Vicon System	75459
Computer with MATLAB	A6703
Attitude quadcopter holder	-
Attitude quadcopter connector	-

Setup

Figure D.1 shows the test set up used when testing the attitude controller.

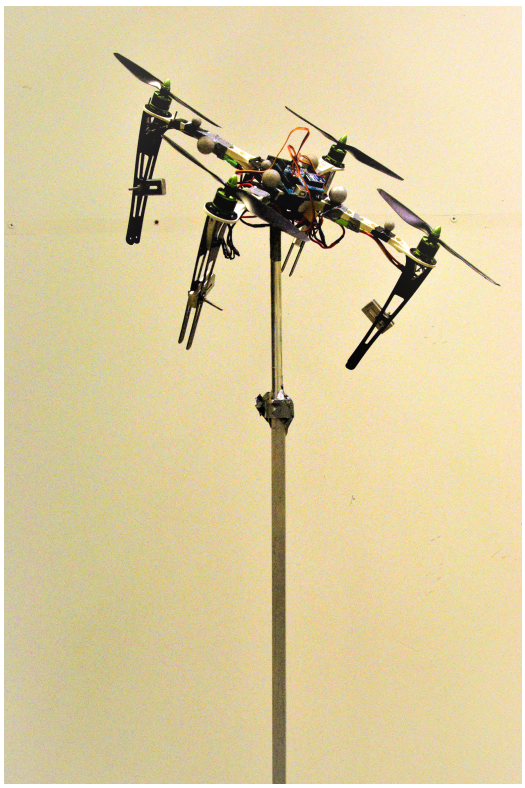

Figure D.1: The test setup utilized to test the attitude controller.

The set up distorts the moments of inertia around the different axes and the location of the center of rotation with respect to the center of mass. The latter is especially critical as the quadcopter behaves like an inverted pendulum. To counteract this effect, the center of mass of the quadcopter must be moved down. This is done placing masses in the four arms of the quadcopter and below the center of rotation.

E | Moments of Inertia Derivation

One set of parameters in the model is the inertia of the quadcopter around its axes, roll, pitch and yaw. There are different approaches to find the inertia, but to get a good starting point the quadcopter is split op in several masses and the inertia is found analytically.

The quadcopter is first decomposed into a set of objects for which the inertia is well defined, see Figure E.1.

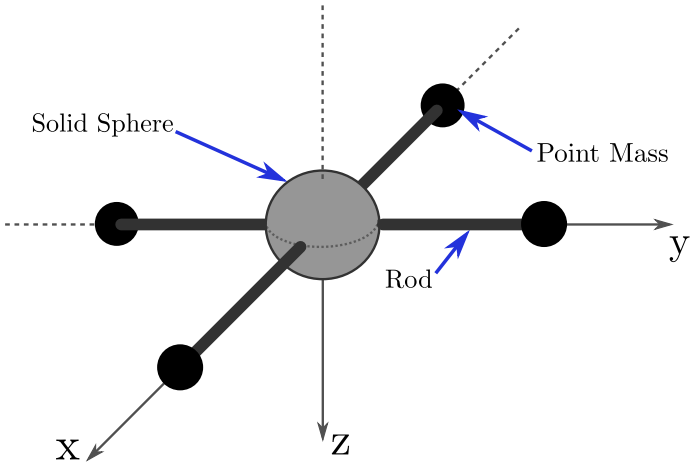

Figure E.1: Decomposition of the quadcopter into objects for which the inertia is well defined.

The inertia must be calculated around axes for which the roll, pitch and yaw angles are defined, that is, the x_B, y_B and z_B axes. Since the quadcopter is controlled in plus configuration, the x_B and y_B axes aligns with the four arms, as seen in Figure E.1.

The objects are analyzed individually around the center of mass, CM, see Figure E.2, after which the inertias are summed for each axis of rotation.

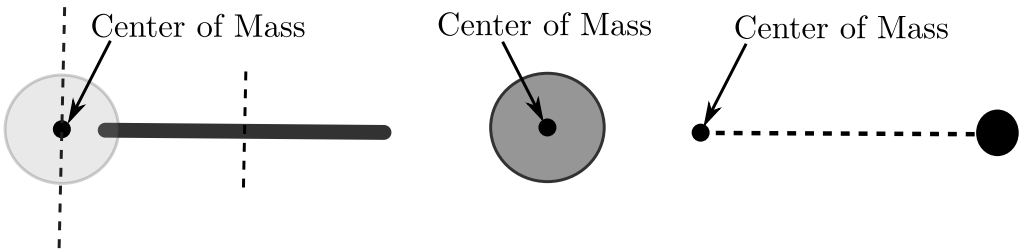

Figure E.2: The masses with respect to the center of mass of the quadcopter and axes of rotation.

To calculate the inertias it is necessary to distribute the mass of the quadcopter between the decomposed objects in Figure E.1. To do this the motors and propellers are weighed and considered to be the point mass, the arm and ESC are weighed and considered to be the rod. Finally the entire quadcopter was weighed and the weight of the other objects subtracted to find the mass of the sphere.

The inertia of the sphere is directly given by

$$I_s = \frac{2}{5} m_s r_s^2 \qquad \left[\mathrm{kg} \cdot \mathrm{m}^2\right] \quad \text{(E.1)}$$

Where:

I_s	is the moment of inertia around the CM of the quadcopter	$[\mathrm{kg} \cdot \mathrm{m}^2]$
m_s	is the mass of the sphere	[kg]
r_s	is the radius of the sphere	[m]

For a point mass the moment of inertia around the CM of the quadcopter is given by

$$I_p = m_p d_p^2 \qquad \left[\mathrm{kg} \cdot \mathrm{m}^2\right] \quad \text{(E.2)}$$

Where:

I_p	is the moment of inertia around the CM of the quadcopter	$[\mathrm{kg} \cdot \mathrm{m}^2]$
m_p	is the mass of the point	[kg]
d_p	is the distance from the point mass to the CM of the quadcopter	[m]

To calculate the moment of inertia of the rod, it is first evaluated around its own center of mass, see Figure E.2. Then by use of the parallel axis theorem the inertia is moved, such that it is described around the center of mass of the quadcopter.

The parallel axis theorem states that any mass with known inertia around an axis can be described around a parallel axis by adding its mass multiplied by the distance between the parallel axes squared.

For the rod this yields the following,

$$I_r = \frac{1}{12} m_r L_r^2 + m_r d_r^2 \qquad \left[\mathrm{kg \cdot m^2}\right] \quad \text{(E.3)}$$

Where:

I_r	is the moment of inertia around CM of the rod	$[\mathrm{kg \cdot m^2}]$
m_r	is the mass of the rod	[kg]
L_r	is the length of the rod	[m]
d_r	is the distance from CM of the rod to the CM of the quadcopter	[m]

The found inertias are summed for each axis of rotation to obtain the final inertias of the quadcopter.

$$I_x = I_s + 2I_p + 2I_r \qquad \left[\mathrm{kg \cdot m^2}\right] \quad \text{(E.4)}$$

$$I_y = I_s + 2I_p + 2I_r \qquad \left[\mathrm{kg \cdot m^2}\right] \quad \text{(E.5)}$$

$$I_z = I_s + 4I_p + 4Ir \qquad \left[\mathrm{kg \cdot m^2}\right] \quad \text{(E.6)}$$

Where:

I_x	is the moment of inertia around the x-axis	$[\mathrm{kg \cdot m^2}]$
I_y	is the moment of inertia around the y-axis	$[\mathrm{kg \cdot m^2}]$
I_z	is the moment of inertia around the z-axis	$[\mathrm{kg \cdot m^2}]$

In Table E.1 the measured quantities are given.

m_s	m_p	m_r	r_s	d_p	L_r	d_r
0.4 kg	0.074 kg	0.075 kg	0.065 m	0.120 m	0.165 m	0.225 m

Table E.1: Measured parameters of the quadcopter.

Inserting the measured quantities in the formulas yields the following,

$$I_s = \frac{2}{5} 0.4 \times 0.065^2 = 0.6760 \times 10^{-3} \quad \mathrm{kg \cdot m^2} \quad \text{(E.7)}$$

$$I_p = 0.074 \times 0.120^2 = 0.0011 \quad \mathrm{kg \cdot m^2} \quad \text{(E.8)}$$

$$I_r = \frac{1}{12} 0.075 \times 0.165^2 + 0.075 \times 0.225^2 = 0.0040 \quad \mathrm{kg \cdot m^2} \quad \text{(E.9)}$$

$$I_x = 0.6760 + 2 \times 0.0011 + 2 \times 0.0040 = 0.0107 \quad \mathrm{kg \cdot m^2} \quad \text{(E.10)}$$

$$I_y = 0.6760 + 2 \times 0.0011 + 2 \times 0.0040 = 0.0107 \quad \mathrm{kg \cdot m^2} \quad \text{(E.11)}$$

$$I_z = 0.6760 + 4 \times 0.0011 + 4 \times 0.0040 = 0.0208 \quad \mathrm{kg \cdot m^2} \quad \text{(E.12)}$$

F | Matrices for State Space Design

In this appendix the exact matrices used in the state space design are presented.

$$\mathbf{A}_{6\times6} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{B}_{6\times4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -0.2396 & 0 & 0.2396 \\ 0.2396 & 0 & -0.2396 & 0 \\ 0.0377 & -0.0377 & 0.0377 & -0.0377 \end{bmatrix}$$

$$\mathbf{C}_{3\times6} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{\mathcal{C}}_{6\times24} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -0.2396 & 0 & 0.2396 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0.2396 & 0 & 0.2396 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0.0377 & -0.0377 & 0.0377 & -0.0377 & 0 & \dots \\ 0 & -0.2396 & 0 & 0.2396 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0.2396 & 0 & -0.2396 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0.0377 & -0.0377 & 0.0377 & -0.0377 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}$$

$$\mathcal{O}_{18\times 6} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{F}_{4\times 6} = \begin{bmatrix} -0.0000 & -165.1476 & -223.3527 & -0.0000 & -44.0357 & -68.5229 \\ 165.1476 & 0.0000 & 223.3527 & 44.0357 & 0.0000 & 68.5229 \\ 0.0000 & 165.1476 & -223.3527 & 0.0000 & 44.0357 & -68.5229 \\ -165.1476 & -0.0000 & 223.3527 & -44.0357 & -0.0000 & 68.5229 \end{bmatrix}$$

$$\mathbf{F}_{\mathbf{Int}4\times 6} = \begin{bmatrix} 0.0000 & -220.9709 & -250.0000 \\ 220.9709 & 0.0000 & 250.0000 \\ -0.0000 & 220.9709 & -250.0000 \\ -220.9709 & -0.0000 & 250.0000 \end{bmatrix}$$

$$\mathbf{A11}_{3\times3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{A12}_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A21}_{3\times3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{A22}_{3\times3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{B1}_{3\times4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{B2}_{3\times4} = \begin{bmatrix} 0 & -0.2396 & 0 & 0.2396 \\ 0.2396 & 0 & -0.2396 & 0 \\ 0.0377 & -0.0377 & 0.0377 & -0.0377 \end{bmatrix}$$

$$\mathbf{L_{obs}} = \begin{bmatrix} -11 & 0 & 0 \\ 0 & -12 & 0 \\ 0 & 0 & -13 \end{bmatrix}$$

$$\mathbf{L_{obsH}} = \begin{bmatrix} -31 & 0 & 0 \\ 0 & -32 & 0 \\ 0 & 0 & -33 \end{bmatrix}$$

$$\mathbf{Q_{final}}_{9\times9} = \begin{bmatrix} 1/0.2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/0.2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/0.1^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/0.5^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/0.5^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/0.3^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/0.08^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/0.08^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/0.05^2 \end{bmatrix}$$

$$\mathbf{R_{final}}_{4\times4} = \begin{bmatrix} 1/25^2 & 0 & 0 & 0 \\ 0 & 1/25^2 & 0 & 0 \\ 0 & 0 & 1/25^2 & 0 \\ 0 & 0 & 0 & 1/25^2 \end{bmatrix}$$

$$\mathbf{Q_{high}}_{9\times9} = \begin{bmatrix} 1/0.1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/0.1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/0.1^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/0.4^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/0.5^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/0.3^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/0.08^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/0.08^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/0.05^2 \end{bmatrix}$$

$$\mathbf{R_{high}}_{4\times4} = \begin{bmatrix} 1/65^2 & 0 & 0 & 0 \\ 0 & 1/65^2 & 0 & 0 \\ 0 & 0 & 1/65^2 & 0 \\ 0 & 0 & 0 & 1/65^2 \end{bmatrix}$$