Final project

# Advanced quadrotor flight

***Author:***
Mikael Juhl Kristensen

***Supervisor:***
Henrik Schiøler

AALBORG UNIVERSITY
DENMARK

Department of Electronic Systems
Fredrik Bajers Vej 7
9220 Aalborg Ø
Telephone: 99 40 86 00
http://es.aau.dk

**Title:** Advanced quadrotor flight

**Theme:** Non-linear systems

**Project period:** September 2015 - August 2016

**Project group:** 931/1031

**Group members:**

Mikael Juhl Kristensen

**Supervisor:**

Henrik Schiøler

**Copies:** 3

**Pages:** 60

**Appendices:** 1

**Finished** 12. August 2016

Abstract:

This project is on autonomous quadrotor flight. In relation to the UAWORLD project it is desired to fly indoors using simple sensors. For good estimation of the states a position measurement is needed. A system from GamesOnTrack is in this project used for this. This leaves some problems in terms of measurement of the states, which is solved in this project with different estimators (Extended Kalman Filter and sensor driven Extended Kalman filter). For more applicable flight capability, the quadrotor system should also be able to yaw freely, also while moving. This allows for a payload to be pointed in a desired direction. This problem is handled by good yaw estimation and by generating the error vector in the body reference frame. The error vector could then be used with a Linear Quadratic Regulator (LQR) for flight in three dimensions and with free yaw control. It is also desired that the quadrotor should be tolerant to physical disturbances. Different methods for handling these is therefore investigated. The system was tuned and tested through simulation. An implementation scheme was designed but was unfortunately never implemented on the real system.

# Group Members

Mikael Juhl Kristensen

## Reading guide

To ease the reading of the report an introduction to the syntax used is given.

### Internal references

All references in the report is denoted as for example Figure 6.2. All equations is denoted with parentheses and the equation number as for instance: (3.2).

### Note on notation

To separate vectors and matrices from scalars they are written like this:

| | |
|---|---|
| Matrices : | $\underline{\mathbf{A}}$ |
| Vector : | $\bar{\boldsymbol{v}}$ |
| Identity matrix : | $\underline{\mathbf{1}}$ |

### Reference frames

In this study there is used four reference frames. When something is described in a specific reference frame it is assigned with the following letters.

| | |
|---|---|
| Inertial Reference Frame: | i |
| Body Reference Frame: | b |

The entity in question is then notated as ${}^{\mathrm{b}}X$ in case it is in body reference frame. When an entity has to be rotated from one reference frame to another a rotation matrix is used. These are denoted ${}^{\mathrm{h}}_{\mathrm{i}}\underline{\mathbf{R}}$ where the lower case is where it is from and the upper case is where it is to.

# Abbreviations

| | |
|---|---|
| AAU | Aalborg University |
| BRF | Body Reference Frame |
| EKF | Extended Kalman Filter |
| ESC | Electronic Speed controller |
| GOT | GamesOnTrack |
| IMU | Inertial Measurement Unit |
| IRF | Inertial Reference Frame |
| LQR | Linear Quadratic Regulator |
| PID | Proportion-Integral-Derivative |
| PWM | Pulse Width Modulation |
| RPM | Rotations Per Minute |
| SDK | Software Development Kit |

# Table of Contents

Chapter 1

# Introduction

Quadrotors has recently become a viable platform for many applications, due to the advancements in Inertial Measurement Unit (IMU), Electronic Speed controller (ESC) and batteries precision and price. Aalborg University (AAU) is involved with a project called UAWORLD, which is aimed at indoor quadrotor flight for industrial tasks. There have been several student projects in UAWORLD. Most of these has flown with locked yaw orientation. Which is okay for some applications, but not ideal. This project will not use locked yaw, but will instead show how "forward flight" can be automated, and explain why it is an advantage in some cases. For this type of flight to function, a good estimation of the yaw is needed. Another parallel problem is disturbances. If a quadrotor is to be used for any physical tasks, such as lifting or spraypainting, then it will experience an external force. This project will investigate what effect these can have on a quadrotor, and how they can be handled. [Nielsen 15]

## 1.1 Problem Description

The main problems this project will deal with, is the problem handling disturbances and flying with yaw.

This problem opens up a series of other problems which is related to such operations:

1. Interacting with other physical objects would properly require precise position and orientation (attitude) control. This leads to the problem: How does a quadrotor fly and operate at/between different positions with different orientations.

2. How does a system calculate its states when not all are measured.

3. Most quadrotor models does not take external forces into account, since it is assumed that there will be no physical contact with other objects. This leads to the problem: How can an autonomous controller handle physical disturbances.

## 1.2 Report outline

**Chapter 2. System Overview:** This chapter outlines how the system works in general, both the internal of the quadrotor and the overall setup.

**Chapter 3. Modelling** This chapter describes what tools have been used to model the quadrotor, and outline the modelling process.

**Chapter 4. Controller** This Chapter details how the controller for the quadrotor is designed, and what thoughts has gone into the design choices.

**Chapter 5. Disturbances** As described in the problem formulation, the quadrotor can expect to be disturbed by external forces. This Chapter describes these disturbances, and how they can be: predicted, modelled and handled.

**Chapter 6. Estimation** This Chapter describes how the sensor suit is used to estimate the position and attitude of the quadrotor.

**Chapter 7. Simulation** In this Chapter, a short introduction to the simulation environment is given. The different simulation scenarios is then presented with their results.

**Chapter 8. Implementation** The designs and systems made through out this project is implemented on a real quadrotor. A short run through all the solution is described in this chapter.

**Chapter 9. Closure** Conclusions are drawn based on the simulations and tests results. The project is wrapped up and evaluated. Suggestions for future projects based on experiences are presented.

Chapter 2

# System Overview

Quadrotors are a subgroup of helicopters, signified by having four rotors mounted vertically. The rotors are usually powered by brushless electric motors, which in turn is powered and controlled by ESC. Quadrotors are normally equipped with a series of internal sensors, such as 3D-gyroscopes and accelerometers. These are used to make an estimate of the quadrotors state.

In this project, the quadrotor will be equipped with a disturbance- predictor and modeller. This system will be able to predict when and how a known disturbance is going to enter the system, and how it will affect the system.

In Figure 2.1 an overview of the internals of the system is shown.

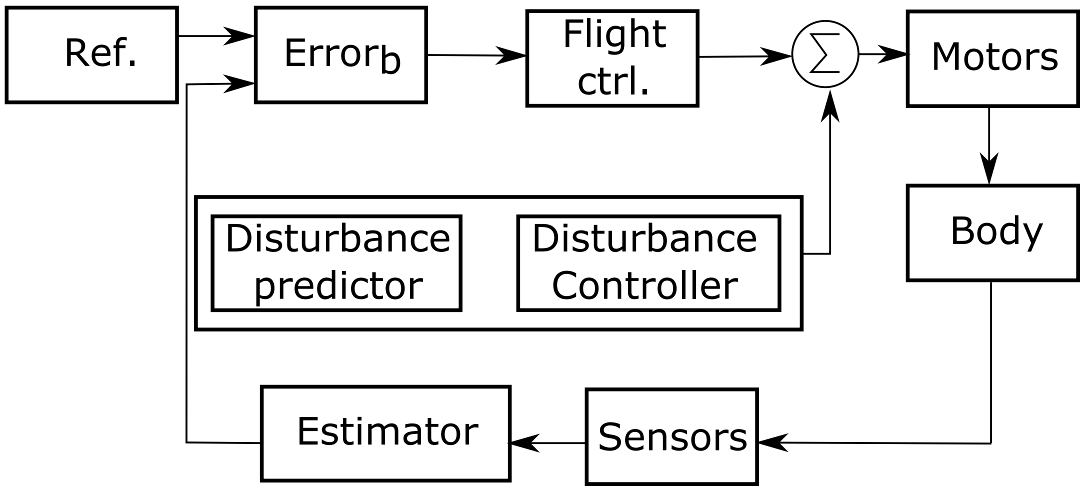


**Figure 2.1:** Overview of the quadrotors internal control system

All indoor quadrotor flight at AAUs Automation and control section is done in its “Vicon lab”. It is an unfurnished room big enough for basic flight tests, equipped with two measuring systems. The first system is a Vicon system. It uses infra-red light and reflectors to measure the position and orientation of the test subject. The second system is a newly aquired system called GamesOnTrack, which uses ultra sound trilateration to measure the position of the test subject. Both systems interface with desktop computer in the adjacent room. The Vicon data is used as true value when evaluating the system, the GamesOnTrack (GOT) data is forwarded to the quadrotor. The test executer should also be able to interface with the quadrotor. A wireless link is therefore introduced. A diagram of the setup can be seen on Figure 2.2

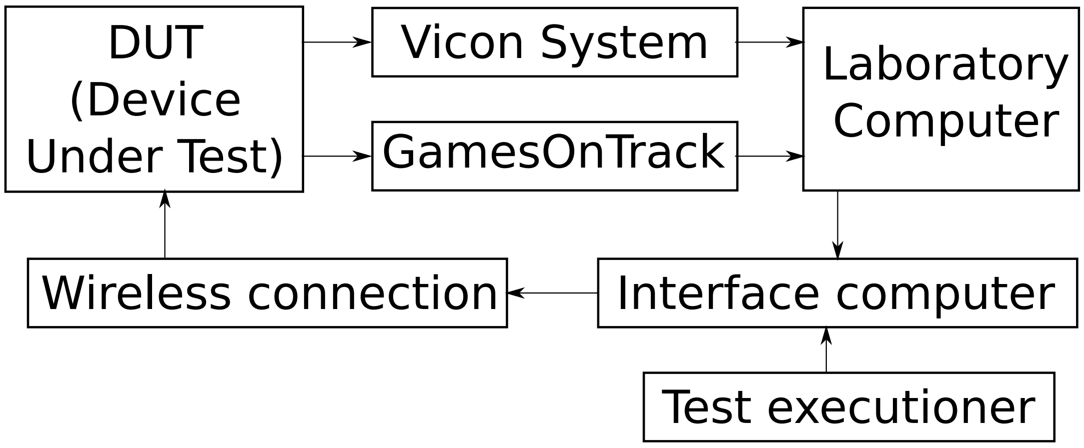


**Figure 2.2:** Diagram of the laboratory setup

CHAPTER 3

# Modelling

*This chapter will describe the quadrotor in mathematical form, such as to allow the design of controllers for autonomous flight*

The models have been extracted in many previous projects, so this chapter will focus on showing the models rather than extrapolating them.

## 3.1 Reference frames

Reference frames are an important tool for modelling systems in relation to each other. Right handed cartesian coordinate system are used. A reference frame is defined by its origin and the direction of two of its axis, as the third is then given.

### 3.1.1 Inertial reference frame

The Inertial Reference Frame (IRF) is set to be the drone laboratory in which test will be executed. With origin in the centre of the room, at ground level. And the x-and y-axis parallel with the walls, such as to match the Vicon system and the GOT. The Z-axis is thereby defined as being positive toward the roof. The coordinate system is notated as "i" $\{{}^i\bar{\boldsymbol{x}},{}^i\bar{\boldsymbol{y}},{}^i\bar{\boldsymbol{z}}\}$

### 3.1.2 Body reference frame

The Body Reference Frame (BRF) is fixed in the quadrotor. The BRF origin is in the centre of mass of the quadrotor. The x-axis is the roll-axis, and is set to be between two of the "arms" of the quadrotor. The y-axis is the pitch axis, and is set to be 90 degrees in horizontal plane from the x-axis. The z-axis is thereby defined to be the vertical axis. The coordinate system is notated as "b" $\{{}^b\bar{\boldsymbol{x}},{}^b\bar{\boldsymbol{y}},{}^b\bar{\boldsymbol{z}}\}$. The coordinate system can be seen on Figure 3.1

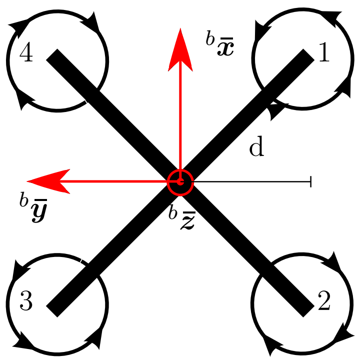

4
1
3
2
${}^b\bar{\boldsymbol{x}}$
${}^b\bar{\boldsymbol{y}}$
${}^b\bar{\boldsymbol{z}}$
d

**Figure 3.1:** The BRF in relation to the quadrotor

## 3.2 Quadrotor parameters

### 3.2.1 Actuator

The quadrotor have four actuators, in the form of the four brushless motors and their attached propellers. The propellers spin in clockwise and anti-clockwise direction in pairs, as illustrated in Figure 3.1. The motor is powered by a three phase signal coming from an ESC, which is controlled by a Pulse Width Modulation (PWM) signal from the micro controller. The relation between the PWM and the Rotations Per Minute (RPM), and the relation between RPM and thrust was found in a previous project. The result of these can be seen in [Thomsen 15]. These measurements are used to find the different gains.

### 3.2.2 Mass

The mass of the quadrotor affects it in several ways. The weight itself results in a force downward (gravity). The distribution of mass throughout the quadrotor affects the change in rotation speed. It is therefore important to have a an estimate of the weight and the distribution of mass. The mass properties were found in [Thomsen 15] to be: The mass of the quadrotor is: 1250grams
The total moment of inertia matrix was found to be:

$$\underline{\mathbf{I}} = \begin{bmatrix} 0.0088 & 0 & 0 \\ 0 & 0.0088 & 0 \\ 0 & 0 & 0.176 \end{bmatrix} \mathrm{kg} \cdot \mathrm{m}^2 \tag{3.1}$$

## 3.3 Kinematic model

*Quaternions is used for describing attitude in this project. The kinematic models describe the derivative of the quaternion, equalling the angular velocity of the attitude*

$q(t)$ is the attitude of the quadrotor at the time $t$, and $q(t + \Delta t)$ is the attitude at the time $t + \Delta t$. $q(t + \Delta t)$ can be seen as the quaternion multiplication of $q(t)$ and $q'(\Delta t)$.

At infinitesimal small rotations $(\Delta\Phi)$ the following approximation can be used:

$$\begin{aligned} \cos\left(\frac{\Delta\Phi}{2}\right) &\approx 1 \\ \sin\left(\frac{\Delta\Phi}{2}\right) &\approx \frac{1}{2}\omega\Delta t \end{aligned} \tag{3.2}$$

$\omega$ is the angular velocity. By using these approximations, the following equation can be found:

$$\bar{\boldsymbol{q}}(t+\Delta t) \approx \left( \underline{\mathbf{1}}_{4x4} + \frac{1}{2}\omega\Delta t \begin{bmatrix} 0 & -u1 & -u2 & -u3 \\ u1 & 0 & -u3 & u2 \\ u2 & u3 & 0 & -u1 \\ u3 & -u2 & u1 & 0 \end{bmatrix} \right) \cdot \bar{\boldsymbol{q}}(t) \tag{3.3}$$

This can be rewritten as:

$$\frac{\mathbf{q}(t+\Delta t) - \mathbf{q}(t)}{\Delta t} = \frac{1}{2}\underline{\boldsymbol{\Omega}}(t)_{4\times 4} \cdot \mathbf{q}(t) \tag{3.4}$$

Where$\underline{\boldsymbol{\Omega}}(t)_{4\times 4}$ is a skew symmetric quaternion multiplication matrix.

$$\underline{\boldsymbol{\Omega}}(t)_{4\times 4} = \begin{bmatrix} 0 & -\omega_1(t) & -\omega_2(t) & -\omega_3(t) \\ \omega_1(t) & 0 & -\omega_3(t) & \omega_2(t) \\ \omega_2(t) & \omega_3(t) & 0 & -\omega_1(t) \\ \omega_3(t) & -\omega_2(t) & \omega_1(t) & 0 \end{bmatrix}, \bar{\boldsymbol{\omega}} = \omega \cdot \bar{\boldsymbol{u}} \tag{3.5}$$

The time derivative of $q(t)$ is then found be letting $\Delta t$ go towards zero, thereby finding the kinematic function:

$$\dot{\mathbf{q}}(t) = \lim_{\Delta t \to 0} \frac{\mathbf{q}(t+\Delta t) - \mathbf{q}(t)}{\Delta t} = \frac{1}{2} \cdot \underline{\boldsymbol{\Omega}}(t)_{4\times 4} \cdot \mathbf{q}(t) \tag{3.6}$$

The time derivative of a rotation matrix ($\dot{[R]}(t)$) is found in a similar matter to be:

$$\underline{\dot{\mathbf{R}}}(t) = -\underline{\boldsymbol{\Omega}}(t)_{3\times 3} \cdot \underline{\mathbf{R}}(t) \tag{3.7}$$

Where:

$$\underline{\boldsymbol{\Omega}}(t)_{3\times 3} = \begin{bmatrix} 0 & -\omega_3(t) & \omega_2(t) \\ \omega_3(t) & 0 & -\omega_1(t) \\ -\omega_2(t) & \omega_1(t) & 0 \end{bmatrix} \tag{3.8}$$

and $\underline{\mathbf{R}}(t)$ is the rotation matrix with the follow property:

$${}^{y}\bar{\boldsymbol{v}} = {}^{y}_{x}\underline{\mathbf{R}}(t)\, {}^{x}\bar{\boldsymbol{v}} \tag{3.9}$$

In both $\dot{q}(t)$ and $\dot{R}(t)$ is it seen that the time derivative of the orientation is dependant on the current orientation, and the angular velocity.

## 3.4 Dynamic model

*This chapter will outline the dynamics present in a quadrotor. The quadrotor is assumed to be rigid.*

The quadrotor has six degrees of freedom, translation in three dimensions and orientation in three dimensions, all containing dynamics. The dynamics are therefore split into rotational- and translation-dynamics.

### 3.4.1 Rotational dynamics

The rotational dynamics describe how the quadrotor reacts to torques, in relation to its current orientation and rotation speed.

Eulers second law states:

$$ {}^{i}\bar{\boldsymbol{\tau}}(t) = {}^{i}\dot{\bar{\boldsymbol{L}}}(t) \tag{3.10}$$

Where ${}^{i}\bar{\boldsymbol{\tau}}(t)$ is the torques acting on the quadrotor, and ${}^{i}\dot{\bar{\boldsymbol{L}}}(t)$ is the change in angular momentum. In order for introducing external disturbances later, the actuator torque is modelled as ${}^{b}\bar{\boldsymbol{\tau}}(t)$ and the disturbance torque is modelled as ${}^{i}\bar{\boldsymbol{\tau}}_{ext}(t)$. The relation between the these two torques is:

$$ {}^{i}\bar{\boldsymbol{\tau}}(t) = {}^{i}\bar{\boldsymbol{\tau}}_{ext}(t) + {}^{i}_{b}\underline{\mathbf{R}}(t)(t) \cdot {}^{b}\bar{\boldsymbol{\tau}}(t) \tag{3.11}$$

Angular momentum of the quadrotors rigid body is defined as:

$$ {}^{i}\bar{\boldsymbol{L}}_{b} = \underline{\mathbf{I}}_{quad} \cdot {}^{i}\bar{\boldsymbol{\omega}}(t) \tag{3.12}$$

Where $\underline{\mathbf{I}}_{quad}$ is the moment of inertia of the quadrotors rigid body, and ${}^{i}\bar{\boldsymbol{\omega}}(t)$ is the rotation speed of the quadrotor body, in the IRF. The propellers and the rotating part of the motor also have an angular momentum. The fact that their centre of rotation is not in the centre of mass of the quadrotor has no effect, it also has no effect that they do not spin in the same direction [Symon 71] . The angular momentum for each propeller is:

$$ {}^{i}\bar{\boldsymbol{L}}_{n} = \underline{\mathbf{I}}_{prop.} \cdot {}^{i}\bar{\boldsymbol{\omega}}_{n}(t) \tag{3.13}$$

Where ${}^{i}\bar{\boldsymbol{\omega}}_{n}(t)$ is the rotation speed of propeller "n". The rotation speed of the propellers are controlled by the quadrotor. The moment of inertia of each propeller is found by assuming uniform mass distribution, with the mass being eight grams and the length being $25cm$.

$$ I = \frac{1}{12} \cdot 0.008kg \cdot (0.25m)^2 = 0.00004166kg \cdot m^2 \tag{3.14}$$

The angular momentums can be summed together:

$$ {}^{i}\bar{\boldsymbol{L}} = {}^{i}\bar{\boldsymbol{L}}_{b} + \sum_{n=1}^{4} {}^{i}\bar{\boldsymbol{L}}_{n} \tag{3.15}$$

The moment of inertia of the propellers are significantly smaller than the body's, but spinning at 4000 RPM, their angular momentum contribution is relevant for the model.

To get external torques into the system, a translation for IRF to BRF is made:

$$ {}^{b}\bar{\boldsymbol{L}}(t) = {}^{b}_{i}\underline{\mathbf{R}}(t) \cdot {}^{i}\bar{\boldsymbol{L}}(t) \tag{3.16}$$

Using the derivative kinematics from 3.7:

$$ {}^{b}\dot{\bar{\boldsymbol{L}}}(t) = {}^{b}_{i}\dot{\underline{\mathbf{R}}}(t) \cdot {}^{i}\bar{\boldsymbol{L}}(t) + {}^{b}_{i}\underline{\mathbf{R}}(t) \cdot {}^{i}\dot{\bar{\boldsymbol{L}}}(t) \tag{3.17}$$

By using equation 3.10 to replace ${}^{i}\dot{\bar{\boldsymbol{L}}}(t)$ with ${}^{i}\bar{\boldsymbol{\tau}}(t)$ the following can be derived:

$$ {}^{b}\dot{\bar{\boldsymbol{L}}}(t) = \underline{\mathbf{I}}_{quad} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = -\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot {}^{b}\bar{\boldsymbol{L}}(t) + {}^{b}_{i}\underline{\mathbf{R}}(t) \cdot {}^{i}\bar{\boldsymbol{\tau}}_{ext}(t) \tag{3.18}$$

It can be seen that the angular acceleration is dependant on the; mass of the quadrotor, the rotation speed of the body and of the propellers, and on the torques affecting the quadrotor.

### 3.4.2 Translation dynamics

The translation model describes the movement of the quadrotor in relation to the applied forces. The model is derived from Newtons second law:

$$\bar{\boldsymbol{F}}_{tot} = m \cdot \bar{\boldsymbol{a}} \tag{3.19}$$

Where $\bar{\boldsymbol{F}}_{tot}$ is the sum of all forces affecting the quadrotor. The two main forces affecting the quadrotor is the gravity and the thrust from the propellers.

Gravity is static, and in IRF only has a component in the z-axis. It can therefore be modelled as:

$$^{b}\bar{\boldsymbol{F}}_{gravity} = {}^{b}_{i}\underline{\mathbf{R}} \cdot (-\,^{i}\bar{\boldsymbol{g}}) \cdot m \tag{3.20}$$

Where $g$ is gravity, and $m$ is the mass of the quadrotor.

All four propellers generate thrust. These can be summed together into one combined thrust in the centre of the quadrotor. In the BRF this combined thrust is only present in the z-axis.

$$^{b}\bar{\boldsymbol{T}}_{total} = {}^{b}\bar{\boldsymbol{T}}_{1} + {}^{b}\bar{\boldsymbol{T}}_{2} + {}^{b}\bar{\boldsymbol{T}}_{3} + {}^{b}\bar{\boldsymbol{T}}_{4} = [0 \quad 0 \quad {}^{b}T_{total}]^{\top} \tag{3.21}$$

By rearranging Newtons second law, the acceleration can be extrapolated.

$$\begin{aligned} ^{b}\bar{\boldsymbol{a}} &= \frac{^{b}\bar{\boldsymbol{F}}_{gravity} + {}^{b}\bar{\boldsymbol{T}}_{total}}{m} \\ ^{i}\bar{\boldsymbol{a}} &= \frac{^{i}_{b}\underline{\mathbf{R}} \cdot {}^{b}\bar{\boldsymbol{T}}_{total}}{m} - [0 \quad 0 \quad g]^{\top} \end{aligned} \tag{3.22}$$

## 3.5 Linearisation

Linear systems are easier to work with, in relation to estimation and control. Very few systems are actually linear by nature, and so would need to be linearised. This is done by looking at a small area of function, and making a linear model of this. The disadvantage of this type of approach comes when the system leaves the linearised area, as the model starts to deviate from the real system then, which in turn could lead to instability or in the case of quadrotors lead to a collision. So before this approach is begun it must first be argued if the system can stay with the area for the desired use.

### 3.5.1 Area of linearisation

Quadrotors normally operate around its hover orientation. This is is the orientation where the z-axis in the BRF and in the IRF are the same, the x- and y-axis does not need to correlate between the two reference frames. This means that the x- and y-axis (the arms of the quadrotor) are horizontal. In this orientation the quadrotor can hang in one position, and move along the z-axis in the IRF. As it is desired to move in three dimensions this is not adequate. The quadrotor accelerates on the x- and y-axis in IRF by pitching and rolling the body. This works because some of the thrust vector normally used to lift the quadrotor is rotated in these axis. How many percentage of the thrust is transferred, is dependant on the amount of degrees pitch/roll, but the amount of acceleration is dependant on the mass of the quadrotor and the thrust in the in this axis. If the pitch/roll is small enough, then it could be acceptable for a linearisation. The amount of thrust in the different axis in percentage can be found by the simple cosinus/sinus relation seen on Figure 3.2

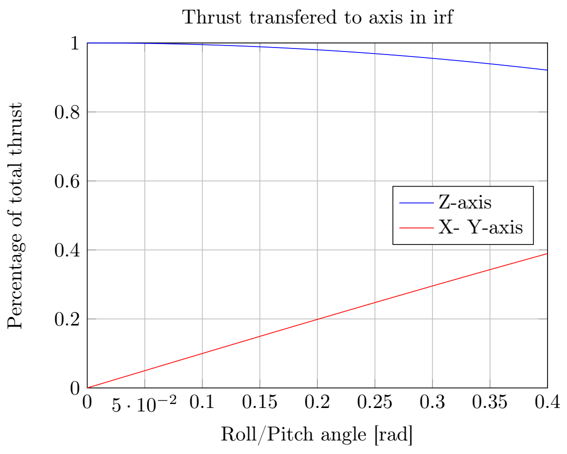


**Figure 3.2:** Graph illustrating the relation between pitch/roll angle

In hover the quadrotor has a given thrust, if this thrust is held at this level while pitching/rolling then it will result in an acceleration in the z-axis (dropping towards the ground). This effect is not desired and should be minimized. A look on this effect over short pitch/roll maneuvers is therefore made in Figure 3.3

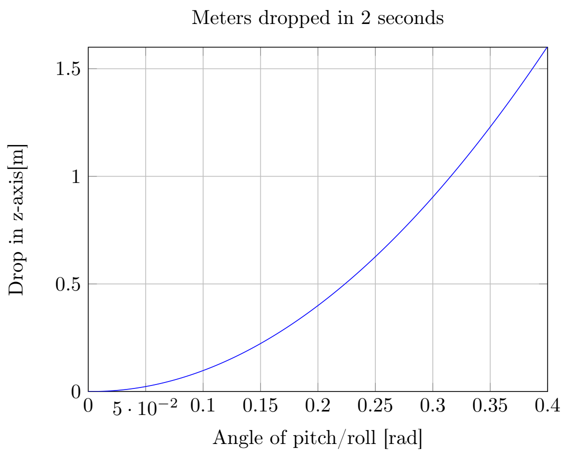


**Figure 3.3:** Graph illustrating the relation between pitch/roll angle and change in height over 2 seconds

Based on Figure 3.2 and 3.3 it is decided to investigate the linearisation point $0^\circ \pm 5^\circ$, as this gives up to 0.1 g acceleration along the x- and y-axis, and only a height loss of 0.1 meter after 2 seconds. When designing the controller it would be relevant to choose a method to limit the pitch and roll within this area.

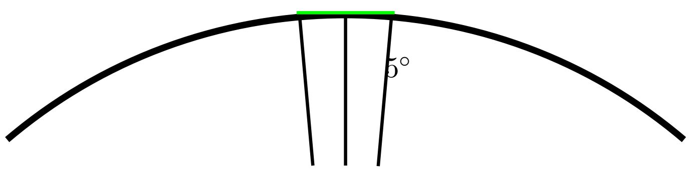


**Figure 3.4:** The green line is the linearisation, the black is the true system

Additionally, the angle of yaw is desired to be constant in the linearisation point, as this simplifies the model. When doing this it is important to note that acceleration of the quadrotor is the same in BRF at any yaw, while speed changes direction in BRF as it is in relation toIRF.

### 3.5.2 Linearisation

With the linearisation point found, a first order Taylor approximation is used to linearise the system.

$$ {}^{b}_{i}\bar{\boldsymbol{q}} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^{\top} \qquad \bar{\boldsymbol{\omega}} = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^{\top} \tag{3.23}$$

With this $\bar{\boldsymbol{\omega}}$,$\underline{\boldsymbol{\Omega}}(t)_{4\times4}$ from equation 3.4 becomes:

$$\underline{\boldsymbol{\Omega}}(t)_{\bar{\boldsymbol{\omega}}} = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix} \tag{3.24}$$

In the linearisation point, there is also no torque from the thrusters, and no external torques.

$$ {}^{b}\bar{\boldsymbol{\tau}}(t) = \bar{\mathbf{0}}, \qquad {}^{i}\bar{\boldsymbol{\tau}}_{ext}(t) = \bar{\mathbf{0}} \tag{3.25}$$

In the linearisation point. When doing pitch, roll or/and yaw, it will move away from the linearisation point.

The first order Taylor approximation is done by inserting into equation 3.6. The linearised rotation speed $\underline{\boldsymbol{\Omega}}(t)_{\bar{\boldsymbol{\omega}}}$ is inserted with the actual quaternion ${}^{b}_{i}q$, and the actual rotation speed matrix $\underline{\boldsymbol{\Omega}}(t)_{4\times4}$ with the linearised quaternion ${}^{b}_{i}\bar{\boldsymbol{q}}$

$$ {}^{b}_{i}\dot{q} = \frac{1}{2}(\underline{\boldsymbol{\Omega}}(t)_{\bar{\boldsymbol{\omega}}} \cdot {}^{b}_{i}q + \underline{\boldsymbol{\Omega}}(t)_{4\times4} \cdot {}^{b}_{i}\bar{\boldsymbol{q}}) = \frac{1}{2} \cdot \begin{bmatrix}0 & \omega_x & \omega_y & \omega_z\end{bmatrix}^{\top} \tag{3.26}$$

The non-linear parts of equation 3.18 is also linearised:

$$ {}^{b}\dot{\bar{\boldsymbol{L}}}(t) = \underline{\mathbf{I}}_{quad} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = -\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot {}^{b}\bar{\boldsymbol{L}}(t)$$

$$\Downarrow$$

$$\underline{\mathbf{I}}_{quad} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) + \sum_{n=1}^{4}(\underline{\mathbf{I}}_{prop.} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}_n(t)) = -\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot (\underline{\mathbf{I}}_{quad} \cdot {}^{b}\bar{\boldsymbol{\omega}}(t) + \sum_{n=1}^{4}(\underline{\mathbf{I}}_{prop.} \cdot {}^{b}\bar{\boldsymbol{\omega}}_n(t)))$$

$$\Downarrow \text{The speed of the rotors do not change in hover}$$

$$\underline{\mathbf{I}}_{quad} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = -\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot (\underline{\mathbf{I}}_{quad} \cdot {}^{b}\bar{\boldsymbol{\omega}}(t) + \sum_{n=1}^{4}(\underline{\mathbf{I}}_{prop.} \cdot {}^{b}\bar{\boldsymbol{\omega}}_n(t)))$$

$$\Downarrow \text{In hover the angular momentum from the propellers equals out}$$

$$\underline{\mathbf{I}}_{quad} \cdot {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = -\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot (\underline{\mathbf{I}}_{quad} \cdot {}^{b}\bar{\boldsymbol{\omega}}(t))$$

$$\Downarrow \text{Move the inertia to the right side}$$

$$ {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = \underline{\mathbf{I}}_{quad}^{-1} \cdot (-\,{}^{b}\underline{\boldsymbol{\Omega}}_{3x3} \cdot (\underline{\mathbf{I}}_{quad} \cdot {}^{b}\bar{\boldsymbol{\omega}}(t))) \tag{3.27}$$

Under the assumption that both body and propellers has diagonal inertia matrices and only spin around the z-axis, the following extrapolation is reached:

$$ {}^{b}\dot{\bar{\boldsymbol{\omega}}}(t) = \begin{bmatrix}\frac{I_y - I_z}{I_x} \cdot \omega_y \cdot \omega_z\\ -\frac{I_x - I_z}{I_y} \cdot \omega_x \cdot \omega_z\\ \frac{I_x - I_y}{I_z} \cdot \omega_x \cdot \omega_y\end{bmatrix} \tag{3.28}$$

In perfect hover the rotation speed of the quadrotor is zero, the linearisation area is slightly bigger, and very small rotation speed might therefore occur. When multiplying two very small numbers the result is close to zero:

$$ {}^{b}\dot{\boldsymbol{\omega}}(t) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.29} $$

Only the linear component then remains:

$$ {}^{b}\dot{\boldsymbol{\omega}}(t) \approx \underline{\mathbf{I}}^{-1} \cdot {}^{b}\bar{\boldsymbol{\tau}}_{ext}(t) \tag{3.30} $$

The translation 3.22 also has to be linearised. In hover all thrust is in the z-axis, and the rotation matrix between BRF and IRF can therefore be simplified. The acceleration is therefore:

$$ {}^{i}\bar{\boldsymbol{a}} = {}^{i}_{b}\underline{\mathbf{R}} \cdot \frac{{}^{b}\bar{\boldsymbol{T}}_{total}}{m} - \begin{bmatrix} 0 & 0 & g \end{bmatrix}^{\top} \approx \begin{bmatrix} 2 \cdot (q_1 \cdot q_3 + q_2 \cdot q_0) \\ 2 \cdot (q_2 \cdot q_3 - q_1 \cdot q_0) \\ -1 - 2 \cdot q_1^2 - 2 \cdot q_2^2 \end{bmatrix} \cdot \frac{{}^{b}\bar{\boldsymbol{T}}_{total}}{m} - \begin{bmatrix} 0 & 0 & g \end{bmatrix}^{\top} \tag{3.31} $$

With the quaternion being:

$$ {}^{b}_{i}\bar{\boldsymbol{q}} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.32} $$

The first order Taylor approximation of equation 3.31 is then:

$$ {}^{i}\bar{\boldsymbol{a}} \approx \begin{bmatrix} 2 \cdot q_2 \\ -2 \cdot q_1 \\ 1 \end{bmatrix} \cdot \frac{{}^{b}\bar{\boldsymbol{T}}_{total}}{m} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.33} $$

The models are now ready to be used in a linear model.

Chapter 4

# Controller

*This chapter will describe how a controller is found based on the model of the quadrotor found in Chapter 3*

As shown on Figure 2.1, the controller will get an estimation and a reference as input. Additionally disturbances can enter the system, and the controller should therefore be robust towards unknown/unpredicted disturbances, and able to handle know/predicted disturbances.

## 4.1 State space

The quadrotor can be described in state space form. The quadrotor itself can be described by its; position, speed, orientation and angular velocity.

$$\begin{aligned} \bar{\boldsymbol{x}}_b &= [q_1 \quad q_2 \quad q_3 \quad \omega_x \quad \omega_y \quad \omega_z \quad x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z}] \\ \text{for short:} \bar{\boldsymbol{x}} &= [\bar{\boldsymbol{q}} \quad \bar{\boldsymbol{\omega}} \quad \bar{\boldsymbol{P}} \quad \dot{\bar{\boldsymbol{P}}}] \end{aligned} \tag{4.1}$$

The quadrotor can essentially only do four things; roll, pitch, yaw and thrust. With these it can change the states of the quadrotor. A state space system can be made such that one controller sets the desired; roll, pitch, yaw and thrust, and the second translates these into motor signals.

### 4.1.1 Body model

With $T$ being the total thrust in the linearisation point, and the input being:

$$\bar{\boldsymbol{u}} = [F_x \quad F_y \quad F_z \quad \tau_x \quad \tau_y \quad \tau_z]^\top \tag{4.2}$$

$$\underline{\mathbf{A}}_b = \begin{bmatrix} 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 2 \cdot \frac{T}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 \cdot \frac{T}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \underline{\mathbf{B}}_b = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_z} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \end{bmatrix}$$

$$\underline{\mathbf{C}}_b = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \underline{\mathbf{D}}_b = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

The output matrix $\underline{\mathbf{C}}$ is chosen based on the sensor availability described in table 6.1.

### 4.1.2 Actuator model

The actuator model translates from; motor signals to; roll, pitch, yaw and thrust. The state of the actuator is the rotation speed "r" of the propellers.

$$\begin{aligned} \bar{\boldsymbol{x}}_a &= [r_1 \quad r_2 \quad r_3 \quad r_4]^\top \\ \underline{\mathbf{A}}_a &= \begin{bmatrix} \frac{1}{t_1} & 0 & 0 & 0 \\ 0 & \frac{1}{t_2} & 0 & 0 \\ 0 & 0 & \frac{1}{t_3} & 0 \\ 0 & 0 & 0 & \frac{1}{t_4} \end{bmatrix}, \quad \underline{\mathbf{B}}_a = \begin{bmatrix} G1 & 0 & 0 & 0 \\ 0 & G1 & 0 & 0 \\ 0 & 0 & G1 & 0 \\ 0 & 0 & 0 & G1 \end{bmatrix} \\ \underline{\mathbf{C}}_a &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ G2 & G2 & G2 & G2 \\ -G2 \cdot d & -G2 \cdot d & G2 \cdot d & G2 \cdot d \\ -G2 \cdot d & G2 \cdot d & G2 \cdot d & -G2 \cdot d \\ -G3 & G3 & -G3 & G3 \end{bmatrix}, \quad \bar{\boldsymbol{Y}} = [T_x \quad T_y \quad T_z \quad \tau_x \quad \tau_y \quad \tau_z]^\top \end{aligned} \tag{4.4}$$

Where:

- $r_{1,2,3,4}$ are the RPM of the propellers
- $t_{1,2,3,4}$ are the time constant of the propellers
- $G1$ is the gain from motor signal to RPM
- $G2$ is the gain from RPM to thrust
- $G3$ is the gain from RPM to torque

- d is the distance shown on Figure 3.1 [m]
- $T_{x,y,z}$ is the thrusts [N]
- $\tau_{x,y,z}$ are the torques in the BRF [Nm]

$G1$, $G2$, $G3$ and $t_{1,2,3,4}$ are found through experiment, found in (skal laves)

### 4.1.3 Combined model

Two state space models have been made, and now need to be combined in one model for later use. The actuator models output goes into the body model. The combined model is:

$$\begin{aligned}\begin{bmatrix}\dot{\bar{\boldsymbol{x}}}_b \\ \dot{\bar{\boldsymbol{x}}}_a\end{bmatrix} &= \begin{bmatrix}\underline{\mathbf{A}}_b & \underline{\mathbf{B}}_b \cdot \underline{\mathbf{C}}_a \\ \underline{\mathbf{0}} & \underline{\mathbf{A}}_a\end{bmatrix} \cdot \begin{bmatrix}\bar{\boldsymbol{x}}_b \\ \bar{\boldsymbol{x}}_a\end{bmatrix} + \begin{bmatrix}\underline{\mathbf{0}} \\ \underline{\mathbf{B}}_a\end{bmatrix} \cdot \bar{\boldsymbol{u}} \\ \begin{bmatrix}\bar{\boldsymbol{y}}_b \\ \bar{\boldsymbol{y}}_a\end{bmatrix} &= \begin{bmatrix}\underline{\mathbf{C}}_b & \underline{\mathbf{0}} \\ \underline{\mathbf{0}} & \underline{\mathbf{C}}_a\end{bmatrix} \cdot \begin{bmatrix}\bar{\boldsymbol{x}}_b \\ \bar{\boldsymbol{x}}_a\end{bmatrix}\end{aligned} \tag{4.5}$$

The observability and controllability can now be assessed. The combined system has 16 states, 12 from the body model and four from the actuator.

The rank of the controllability matrix is found to be 16. This means that a controller can be designed which can bring all states to zero from any value, via a series of control signals.

The rank of the observability matrix is also found to be 16. This means that while not all states can be measured directly, all states can be observed via the systems outputs.

### 4.1.4 Discretization

The two state space models have been combined into one, but the models is continuous. For the Kalman filter and for the controller as discrete model is needed. The model is discretized with Matlabs “C2D” function

## 4.2 LQR

As outlined in Chapter 2, the system will use an Linear Quadratic Regulator (LQR) controller as its main flight controller. This choice has been made because it gives a good performance, and it has been tested on quadrotors before. The LQR controller will be set to run at 10 Hz, as that will be the speed of the estimator. To design a discrete LQR controller requires a discrete state space model of the system, and a series of tuning parameters. The discrete state space model has been found previous in this chapter.

Before the tuning of an LQR controller can begin, a few decisions has to be made.

1. Reference states
2. Maximum speed required

### 4.2.1 Reference states

The reference states are the states which the user sets, and the controller attempts to achieve. A quadrotor can be flown in many different ways depending on the choice of these. Normally it is said that because the system has four degrees of freedom, up to four reference states can be chosen. Below is a short introduction of some the selections which could be used

#### Attitude - Height/Thrust

This choice is favored by hobby RC flight as it allows easy control from two joysticks. The human operator then acts as a position/velocity controller as (s)he controls the attitude to move the quadrotor around. This mode is also used for First Person Flight, as it is very similar to the controls of an air plane. Directly controlling the attitude also allows the user to do “tricks” such as flips and “loop de loops”.

#### Velocity - yaw

This mode is more for autonomous flight systems, as it can be confusing for manual users to keep track of the direction, especially if the velocity is given in IRF. It has the advantages that quadrotor either does not need to compute the position estimate as it is obsolete, or that it can use the position state as the integral of the reference and thereby use a full Proportion-Integral-Derivative (PID) controller (this effect can also be used in LQR). This type of controller could be used for indoor flight by making a potential field system as in [Thomsen 14], this would keep the system from hitting walls, it requires a good position estimate. If the system has the position state, and uses it in its controller, then a position reference has to be generated based on the velocity reference.

#### Position - yaw

This mode is for autonomous flight, as the user has little control over how the state is achieved. It is often used for waypoint flight, where the user defines a path of locations the quadrotor has to pass through. The user can then use his/her time on controlling the payload, and maybe yaw the quadrotor to get the right attitude for the payload, like a camera filming. This mode is good for indoor flight, but especially with LQR it has a weakness to large changes in the reference, as this will give high velocities, which could be unstable.

#### Choice

For this project the position and yaw state has been chosen as the reference states. This is because the system will test the GOT system which measures the position, and the position control allows for easy safe flight in our laboratory. The yaw state will allow the system to face all directions, which is a core part of this project. To prevent big spikes a limit is put on the error of the position state entering the controller. The velocity will be used as derivative state in the controller, and its reference is set to zero.

### 4.2.2 Maximum speed

For tuning the LQR controller Brysons rule, which is based on maximum allowed deviation from reference. It is therefore relevant to talk what the maximum allowed speed is, as it will act a the derivative of the main state.

The quadrotor will in this project fly in a relatively small room, measuring 5x5 meters. Its functionality is not to fly fast, or get from one point to another at any specific time. On the other hand, as the system is experimental there is an increased change of accidents. It is therefore decided that the maximum speed should be 0.1 meter per second. This is a sufficient speed for testing, while being slow enough for an operator to stop accidents.

### 4.2.3 Generation and tuning

The LQR controller is a matrix. By multiplying this matrix with the state errors the control signals are obtained. The LQR matrix is made by solving the Riccati equation with Matlabs LQR function. The LQR function requires the model, as found earlier, and a $\underline{\mathbf{R}}$ and $\underline{\mathbf{Q}}$ matrix.

The $\underline{\mathbf{R}}$ matrix tunes the control signals, meaning tuning the controllers tendency to use bigger control signals. This is important to control, as the output of the controller should be a number between zero and one indicating the percentage of power on the given motor. The values in the matrix which is used for tuning is in this system the diagonal of the matrix.

The $\underline{\mathbf{Q}}$ matrix tunes the controller in regards to the error states. The basis of tuning here will be based on Brysons rule, which states that the tuning parameter on $\underline{\mathbf{Q}}$s diagonal shall be:

$$Q_{i,j=i} = \frac{1}{\text{Maximum allowed value}^2} \tag{4.6}$$

The LQR controller is tuned and tested in the simulation 7

Chapter 5

# Disturbances

*This chapter will outline what types of disturbances is relevant for a quadrotor, and model these disturbances*

## 5.1 Disturbances

Disturbances are external forces (or lack thereof) acting on the quadrotor. This analysis is based on an indoor quadrotor executing physical tasks.

### 5.1.1 Wind

Normally their is no significant winds indoors, but a drag/gust might occur through a building. For simplicity sake it is assumed that wind will hit the entire quadrotor at once, with equal force. Under this assumption wind will not significantly affect the yaw of a quadrotor. But it could have an effect on roll/pitch, thrust and movement.

Drag or gusts of wind can be difficult to detect, but with a distributed sensor system throughout a building it would be possible. It could therefore be relevant for a quadrotors disturbance controller, to have handling techniques for both unpredicted winds as well as predicted winds.

### 5.1.2 Touch / continous

When interacting with the external hardware or using physical payloads, the quadrotor can experience a continuous force upon its body. These forces can enter the system at different locations, such as under an engine or under the centre of mass. And they can come at different angles.

The: time, location, angle and size can be known or unknown. Locally controlled payloads can be modelled, and all the parameters could thereby the predicted. When interacting with the world the time of interaction can be predicted with a good map and a good position estimation, but the other parameters can be hard to predict precisely. But if the disturbance is from an unknown source, then none of the parameters can be predicted, but they could be estimated and countered.

### 5.1.3 Hit / impulse

Some payloads, tasks or unknown disturbances comes in the form of a impulse, a short/instant force. Some payloads and tasks will enable prediction of all or some of the parameters: time, location, angle and size.

The disturbance controller should be able to take disturbance predictions/measurements and translate these into control signals which counteracts the disturbance. Two approaches where tried to this problem. A Model Predictive controller (MPC) is used as the classic approach, it uses the system model to plan the optimal control signals to the motor in order to achieve the references. A Model Predictive controller would have to replace the LQR controller if used, a different approach is therefore investigated. An experimental controller which is decoupled from the main controller and only focus on removing the disturbance through use of the actuators.

## 5.2 Model Predictive control (MPC)

The MPC is optimal for the length of its control horizon, and by adding disturbance models to the state space, and adding the disturbances as states, the disturbances should be tolerable.

### 5.2.1 Adding disturbance

The disturbances are modeled as first order systems. The resulting equations are:

$$\dot{F}_{dx,y,z} = a_F \cdot F_d + e_F \tag{5.1}$$

$$\dot{\tau}_{dx,y,z} = a_\tau \cdot \tau + e_\tau \tag{5.2}$$

The noise $e_F$ and $e_\tau$ are white noise processes.

The new controller is based upon the a linear discrete state space model, like the one in section 4.1. The dynamics of the disturbances can then be inserted into $\underline{\mathbf{A}}_b$. The disturbance states are then coupled with the system state; The force disturbance is inserted into the speed row as $1/m$, and the torque disturbance is coupled to the rotation speed row via $1/I_{x,y,z}$.

### 5.2.2 Theory

Model predictive controllers are quite different from more normally used controller such as PID and LQR. Normal controllers take an error as input, and based on the inputs calculates an input which should make the error go towards zero. The MPC takes both the reference, feedback(estimation) and control signal as input. The MPC also calculates in changes of the control signal. [Maciejowski 02]

This section will explain the theory behind the model predictive controller.

#### Reference frame

The estimations and references are in the IRF. The MPC uses prediction based on the states, and therefore has to operate in the IRF. The MPC output a control signal in IRF, but the actuators need the control signals in the correct yaw, and the signal vector is therefore rotated by the estimated yaw. The disturbances also need to be rotated into the IRF.

## Gravity

For the MPC to function optimally it needs to account for the Earths gravity. State space models does not allow for insertion of constant. Instead it is added as disturbance force on the z-axis. The size of which is:

$$F_z = m_{quad} \cdot g \tag{5.3}$$

## Prediction

The prediction is based on the current state estimate ($\bar{\boldsymbol{x}}_{k|k}$) and the previous control signal $\bar{\boldsymbol{U}}_{k-1}$.

The states at step $k+1$ can then be calculated as:

$$\hat{\bar{\boldsymbol{x}}}_{k+1|k} = \underline{\mathbf{A}}\bar{\boldsymbol{x}}_{k|k} + \underline{\mathbf{B}}(\Delta\hat{\bar{\boldsymbol{u}}}(k|k) + \bar{\boldsymbol{U}}_{k-1}) \tag{5.4}$$

Where $\Delta\hat{\bar{\boldsymbol{u}}}(k|k$ is the change in the control signal outputted by the controller at the current step. This method can be continued into future time step:

$$\hat{\bar{\boldsymbol{x}}}_{k+2|k} = \underline{\mathbf{A}}^2\bar{\boldsymbol{x}}_{k|k} + \underline{\mathbf{A}}\underline{\mathbf{B}}(\Delta\hat{\bar{\boldsymbol{u}}}(k|k) + \bar{\boldsymbol{U}}_{k-1}) + \underline{\mathbf{B}}(\Delta\hat{\bar{\boldsymbol{u}}}(k+1|k) + \Delta\hat{\bar{\boldsymbol{u}}}(k|k) + \bar{\boldsymbol{U}}_{k-1}) \tag{5.5}$$

The pattern in this method is that the state is propagated by $\underline{\mathbf{A}}$ for each step, and the change in the state from the previous step's input is also propagated by $\underline{\mathbf{A}}$. This pattern repeats all the way up to the control horizon $(k + h_c - 1)$, after this step the control signal does not change. The prediction pattern is continued all the way to the prediction horizon $(k + h_p - 1)$. It is required that the prediction horizon is equal to or greater than the control horizon.

The pattern can be put on matrix form:

$$\begin{aligned}
\bar{\boldsymbol{X}}(k) =& \underline{\mathbf{A}}_p\hat{\bar{\boldsymbol{x}}}(k|k) + \underline{\mathbf{B}}_u\bar{\boldsymbol{u}}(k-1) + \underline{\mathbf{B}}_{\Delta u}\bar{\boldsymbol{\Delta u}}(k) \\
\bar{\boldsymbol{X}}(k) =& \begin{bmatrix} \hat{\bar{\boldsymbol{x}}}(k+1|k) \\ \vdots \\ \hat{\bar{\boldsymbol{x}}}(k+h_c|k) \\ \vdots \\ \hat{\bar{\boldsymbol{x}}}(k+h_p|k) \end{bmatrix} \\
\underline{\mathbf{A}}_p s =& \begin{bmatrix} \underline{\mathbf{A}} \\ \vdots \\ \underline{\mathbf{A}}^{h_c} \\ \vdots \\ \underline{\mathbf{A}}^{h_p} \end{bmatrix} \\
\underline{\mathbf{B}}_u =& \begin{bmatrix} \underline{\mathbf{B}} \\ \underline{\mathbf{A}}\underline{\mathbf{B}} + \underline{\mathbf{B}} \\ \vdots \\ \sum_{i=0}^{h_p} \underline{\mathbf{A}}^i\underline{\mathbf{B}} \end{bmatrix} \\
\underline{\mathbf{B}}_{\Delta u} =& \begin{bmatrix} \underline{\mathbf{B}} & 0 & 0 & \cdots & 0 \\ \underline{\mathbf{A}}\underline{\mathbf{B}} + \underline{\mathbf{B}} & \underline{\mathbf{B}} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \cdots & 0 \\ \sum_{i=0}^{h_p} \underline{\mathbf{A}}^i\underline{\mathbf{B}} & \sum_{i=0}^{h_p-1} \underline{\mathbf{A}}^i\underline{\mathbf{B}} & \cdots & \sum_{i=0}^{h_p-h_c-1} \underline{\mathbf{A}}^i\underline{\mathbf{B}} & \sum_{i=0}^{h_p-h_c} \underline{\mathbf{A}}^i\underline{\mathbf{B}} \end{bmatrix}
\end{aligned} \tag{5.6}$$

## Cost function

As stated earlier the MPC is an optimal solution to the control problem. But optimal is meaningless without a set of costs, the MPC can then solve the problem at the lowest cost. The costs is the tuning parameters of the controller. An initial set of costs is set based on Bryson's rule similar to process used for LQR tuning. Weight is especially put on keeping the attitude and the height correct.

There is also a cost associated with the control signals, these costs is set to keep them from going into saturation.

The state weights in put on the diagonal of $\underline{\mathbf{Q(i)}}$, and the control signal cost on the diagonal of $\underline{\mathbf{R(i)}}$. The "i" indicator indicates which step in the prediction the cost is set for. This allows for different weights along the predicted path. To focus the controller on stability the cost on pitch and roll is increased towards the control horizon.

The cost function is the quadratic equation:

$$V(k) = \sum_{i=H_w}^{H_p} ||\hat{\bar{\boldsymbol{z}}}(k+i|k) - \bar{\boldsymbol{r}}(k+i|k)||^2_{\underline{\mathbf{Q(i)}}} + \sum_{i=0}^{H_c} ||\Delta\hat{\bar{\boldsymbol{u}}}(k+i|k)||^2_{\underline{\mathbf{R(i)}}} \tag{5.7}$$

Where $\bar{\boldsymbol{r}}(k+i|k)$ is the reference vector, and $\hat{\bar{\boldsymbol{z}}}(k+i|k)$ is the state estimation feedback. Since the system has full state feedback from the estimator, the feedback equals the states.

The cost optimization is gonna be made against current control signal throughout the horizon. The expected error in this scenario therefore needs to be generated. Since the reference is user controlled, the controller cannot know what changes in the reference is gonna be, and the reference vector for the horizon is therefore:

$$R(k) = \begin{bmatrix} r(k|k) \\ \vdots \\ r(k|k) \end{bmatrix} \tag{5.8}$$

The predicted error is then:

$$\underline{\mathbf{E(k)}} = R(k) - \underline{\mathbf{A}}_p \hat{vectx}(k|k) - \underline{\mathbf{B}}_u \bar{\boldsymbol{u}}(k-1) \tag{5.9}$$

With the predicted error and full state feedback the cost function can be reformulated as:

$$V(k) = ||\underline{\mathbf{B}}_{\Delta u}\Delta\hat{\bar{\boldsymbol{U}}}(k) - \underline{\mathbf{E(k)}}||^2_{\underline{\mathbf{Q}}} + ||\Delta\hat{\bar{\boldsymbol{U}}}(k)||^2_{\underline{\mathbf{R}}} \tag{5.10}$$

By finding the minimum value of $V(k)$ with $\Delta\hat{\bar{\boldsymbol{U}}}(k)$ as the only adjustable set of variables the optimal solution can be found. The equation can be optimized for computation by rewriting it to:

$$V(k) = const - \Delta\hat{\bar{\boldsymbol{U}}}(k)^\top \underline{\mathbf{g}} + \Delta\hat{\bar{\boldsymbol{U}}}(k)^\top \underline{\mathbf{H}} \Delta\hat{\bar{\boldsymbol{U}}}(k) \tag{5.11}$$

Where:

$$\begin{aligned} \underline{\mathbf{g}} &= 2\underline{\mathbf{B}}_{\Delta u}^\top \underline{\mathbf{Q}}\underline{\mathbf{E(k)}} \\ \underline{\mathbf{H}} &= \underline{\mathbf{B}}_{\Delta u}^\top \underline{\mathbf{Q}}\underline{\mathbf{B}}_{\Delta u} + \underline{\mathbf{R}} \end{aligned} \tag{5.12}$$

## Constraints

There are some constraints in the system which should be kept, and MPC has this functionality built into its design.

**Slew rate (E)** can be set a constraint, it is the maximum change in control signal. This is not set as a constraint for the quadrotor as the optimization of energy use is not a focus in this project.

**Actuator range (F)** is an important constraint. In systems that have limits to their control signals (most systems have), this constraint will allow the MPC to optimize the control signals to levels that can actually be achieved in the system. For this system the limit for the controls signals are; minimum 0.1 and maximum 1.

The constraint need to be setup in a matrix inequality form which can be used in the constrained optimization problem. The optimization problem is given in relation to $\Delta\hat{\boldsymbol{U}}(k)$, and so the constraints will have to be relative to it as well.

**Variables (G)** can also be constrained. For indoor flight of the quadrotor this can be used to keep the quadrotor away from the walls, lowering the speeds and keeping the attitude close to the linearization point. Limit are set on multiple states as shown in the table below

**Table 5.1:** State limits for MPC

| Limits | Minimum | Maximum |
|---|---|---|
| $q1$ | -0.1 | 0.1 |
| $q2$ | -0.1 | 0.1 |
| $q3$ | | |
| $\omega_x$ | | |
| $\omega_y$ | | |
| $\omega_z$ | 0.3 | 0.3 |
| $P_x$ | -10 | 10 |
| $P_y$ | -10 | 10 |
| $P_z$ | 9 | 11 |
| $\dot{P}_x$ | -0.1 | 0.1 |
| $\dot{P}_y$ | -0.1 | 0.1 |
| $\dot{P}_z$ | -0.1 | 0.1 |

The limits of the states are set such that the quadrotor should not hit walls, the floor or the ceiling, while also keeping the system stable.

The limits also needs to be put on matrix inequality relative to $\Delta\hat{\boldsymbol{U}}(k)$.

## Stability

MPC with finite horizon does not guarantee stability, as it does not plan beyond the horizon, and the states might therefore be going towards unstable paths.

Stability can be guaranteed for controllable system by using infinite prediction horizon. Infinite horizon is not possible to implement in real systems.

By inserting constraints on the states towards the horizon the system goes towards stable. For this particular problem where disturbance enters the system, stability can not be guaranteed.

## Computing control signals

The control signals are computed by finding the minimum value of the cost function, subject to the matrix inequalities made from limits. The problem is a convex minimization problem which can be hard to solve, therefore the CVX Matlab tool is used.

### 5.2.3 Implementation

The MPC need to be implemented into the simulink environment for testing. First a matlab script is made to test out the design and feasibility of the controller. This script can be seen in appendix A. Unfortunately CVX is not available in the simulink environment, and it was therefore inserted with the standard MPC block, and the controller itself was created using the MPC toolbox in matlab. The controller is tuned and tested in simulation. For unexplainable reasons the MPC block could not be configured correctly in time for the finishing of this project, this will be investigated towards the examination.

## 5.3 Experimental controller

This controller is made in a more unorthodox fashion. The controller is designed with the assumption that all disturbances enter the system as a step, and it then attempts to counter them optimally in regard to time and position disturbance.

From the actuator model in section 4.1.2, it can be seen that the actuator of the quadrotor cannot directly make force in the BRF x and y direction. It is therefore decided to make the disturbance controller in two parts; a controller for the thrust and three torques, and a controller for the forces on the x and y direction in BRF. The controllers will be designed with the design philosophy of countering the disturbances as precisely as possible, such that as little disturbance propagates to the states and is corrected by the main flight controller.

The controller controlling thrust and the three torques is fairly simply designed. As there is a model from actuator signal to the four effects, and these consists of a few gains and only a single fast pole, the model can simply be inverted to obtain the controller. The resulting controller is then given as:

$$\begin{bmatrix} k_{thrust} & -k_{RP} & -k_{RP} & -k_{yaw} \\ k_{thrust} & -k_{RP} & k_{RP} & k_{yaw} \\ k_{thrust} & k_{RP} & k_{RP} & -k_{yaw} \\ k_{thrust} & k_{RP} & -k_{RP} & k_{yaw} \end{bmatrix} \cdot \begin{bmatrix} {}^{b}Thrust \\ {}^{b}Torque_x \\ {}^{b}Torque_y \\ {}^{b}Torque_z \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{5.13}$$

Where:

- $U_x$ is the control signal for motor x
- ${}^{b}Thrust$ is the force required to counteract the disturbance in the BRF z-axis
- ${}^{b}Torque$ is the torque required to counteract the disturbance in BRF
- $k_{thrust}$ is the inverse of the actuator model from 4.1.2 for force along the BRF z-axis
- $k_{RP}$ is the inverse of the actuator model from 4.1.2 for Roll and Pitch in BRF
- $k_{yaw}$ is the inverse of the actuator model from 4.1.2 for yaw in BRF

The pole in the propeller gives a small error in the disturbance counter. To counteract this effect, a filter is inserted. As the filter needs to be discrete, the Direct form 1 of an IRR filter is chosen. The filter should have two properties; 1. It should have DC-gain of one 2. The integral of the output from the actuator should be the same as the integral of the input on the disturbance controller. Stepping a pole results in an asymptotic curve equal to:

$$y(t) = a \cdot (1 - e^{(-t/\tau)}) \tag{5.14}$$

Where “a” is the step size. The integral of this function can never be as big as the integral of the step itself. It can thereby be deduced that the filter at least /optimally is a second order filter. The first order trying to overshoot the step, and the second one dropping it back down to DC Gain.

$$a_0 \cdot (1 - e^{(-(1/30)/\tau)}) - a_1 \cdot (e^{(-(1/30)/\tau)}) = 1 \tag{5.15}$$

$$\int_0^{1/30} a_0 \cdot (1 - e^{(-t/\tau)})\, dt + \int_0^{1/30} a_1 \cdot e^{(-t/\tau)}\, dt = 2/30 \tag{5.16}$$

The two equations can be solved to find the two unknown variables $a_0$ and $a_1$. The direct form 1 filter then becomes:

$$a_0 \cdot X(n) + (-a_0 + (a_0 \cdot (1 - e^{(-(1/30)/\tau)}) - a_1)) \cdot X(n-1) + (1 - (a_0 \cdot (1 - e^{(-(1/30)/\tau)}) - a_1)) \cdot X(n-2) = Y(n) \tag{5.17}$$

This filter is setup in the simulation environment, and tested on the motor model. The resulting signal should have DC gain of one, and have the same integration result as a step.

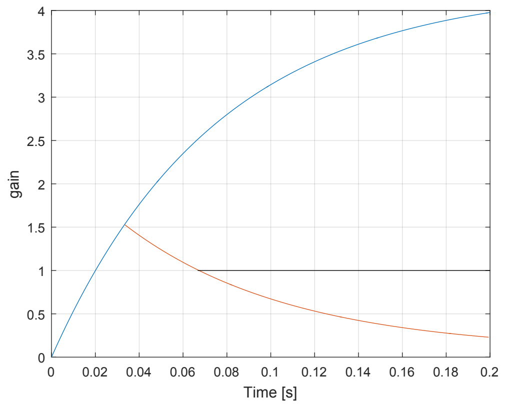


**Figure 5.1:** The poles response to the found filter

The two remaining forces along the x and y-axis in BRF, cannot be induced directly by the actuator. The quadrotor can only generate force in its +Z direction. A system which uses gravity to counteract could also be investigated, but has been ignored for simplicity, it is also not guaranteed to work against all disturbances. The solution is therefore to pitch or roll the quadrotor such that part of the force generated in the +z direction (in BRF) is translated to the x and y-axis of the IRF. The system already has a controller which controls the orientation of the quadrotor; the normal controller. This could therefore potentially be used by changing this controllers error input accordingly.

The normal controller is designed under the linearization assumption that the quadrotor is always close to hover. This same assumption can be used to simplify the design of this disturbance controller. Part of the hover state is a constant thrust, or in better terms a constant force ${}^{b}T_{total}$ along the quadrotors z-axis in its BRF. The force can be translated to the x and y-axis of the IRF by:

$$ {}^{b}F_{x,y} = sin({}^{b}_{h}R) \cdot {}^{b}T_{total} \tag{5.18}$$

The controller should calculate the angle needed, and this is therefore isolated by:

$$ asin({}^{b}F_{x,y}/\,{}^{b}T_{total}) = R_{x,y} \tag{5.19}$$

Where ${}^{b}F_{x,y}$ is the force required to counteract the disturbance. This gives us the angle needed to generate the force needed to counteract the disturbance. To add this angle to the error input of the normal controller, it must first be translated to a quaternion, and then this quaternion can be added to the error quaternion via quaternion multiplication. This then results in the normal controller rotating the quadrotor. The normal controller is an LQR as designed in 4.2. And depending on the tuning of this, it might not give the angling needed for counteracting the disturbance. A gain on the calculated angle (before transforming it into quaternion, to keep unity) is therefore added for tuning. A value for this tuning is found through simulation, and must be confirmed in flight

## 5.4 Modeling

The quadrotor model is split in two parts, the rotational- and the translation dynamics. These respectively take torque and force as input. So if a system in which any of the above mentioned disturbances can be translated into torques and forces can be made, then that would ease the handling of these. Non of the described disturbances are directly torques, they are all forces on one or more points of the quadrotor. By describing the point of disturbance on the quadrotor as ${}^{b}\bar{\boldsymbol{P}}(t) = [x \quad y \quad z]^{\top}$, and the direction and size as ${}^{b}\bar{\boldsymbol{F}}(t) = [F_x \quad F_y \quad F_z]^{\top}$, all disturbances can be described in a uniform fashion. The torque can be calculated as:

$$ {}^{b}\bar{\boldsymbol{\tau}} = {}^{b}\bar{\boldsymbol{P}}(t) \times {}^{b}\bar{\boldsymbol{F}}(t) \tag{5.20}$$

In the case of unknown / unpredicted disturbances, the flight controller should be able to handle these to a certain degree, as they will lead to a change in the states different from the references.

The disturbance controllers ability to counteract disturbances is limited by the actuators. The actuators is in simulation found to be running at 64% in normal hover. This leaves an additional 20.516 Newton in lift in total from the four rotors, equaling about 2 kg.

The worst case for torque disturbances is it being around one of the arms, such that only one pair of rotors can counteract it. This would result in one of the rotors in the pair ramping up towards maximum, and the other ramping down towards minimum RPM. The increasing rotor can deliver an additional 0.834 Nm, and decreasing the opposing rotor equally will contribute an additional 0.4265 Nm, for a total of 1.2605 Nm, which is equal to a $\approx$ 790 gram weight hanging under one of the rotors.

Chapter 6

# Estimation

*For the controller to function, it must have information about both the reference and the current state. In some systems the sensor data is sufficient, But if they are not, then an estimator is necessary*

## 6.1 Sensors

The quadrotor in this project is equipped with a sensor suite consisting of:

- Gyroscope, measures angular velocity $[^\circ/s]$
- Accelerometer, measures acceleration $[m/s^2]$
- Magnetometer, measures the magnetic field $[T]$
- GamesOnTrack, measures position $[m]$

The gyroscope and the accelerometer are the base inertia sensors, they are reliable, reasonable precise sensors. The magnetometer is supposed to measure the Earths magnetic field, which within reasonably distances is static. But when flying indoors or near electric equipment, the magnetic field is quickly altered, and the measurement are then useless. It is therefore important to be aware of such fluctuations. The GamesOnTrack system is a positional system based on ultra sound, which is currently being tested for indoor quadrotor use at AAU, as it is originally for model trains. So far it has been found that high RPM propellers can disturb the system, but the propellers used in this project has not generated this problem.

As mentioned in Section 4.1, the system has a series of states which is desired observed. To get an overview of the association between measurements and these a table has been formed

As seen on the table the speed $\dot{\boldsymbol{P}}$ cannot be measured directly, and should the magnetometer be disturbed by foreign magnetic field then the orientation $\bar{\boldsymbol{q}}$ also becomes unmeasurable. The missing states could be made via integration or differentiation, but these have a tendency to be vulnerable to noise.

**Table 6.1:** Sensor overview

| Sensor \State | $\bar{\boldsymbol{q}}$ | $\bar{\boldsymbol{\omega}}$ | $\bar{\boldsymbol{P}}$ | $\dot{\bar{\boldsymbol{P}}}$ |
|---|---|---|---|---|
| Gyroscope | | M | | |
| Accelerometer | | | | 1/s |
| Magnetometer | I | | | |
| GamesOnTrack | | | M | |

**Table 6.2:** M indicates direct measurement, I denotes indirect and 1/s denotes that the state can be found via integration

## 6.2 Estimators

An estimator uses sensor data, system models and in some cases control signal, to calculate an estimate of the states. There are several types of estimators, with different properties and advantages. The best fitting estimator type should be selected, designed and verified.

### 6.2.1 Particle filter

A particle filter is an estimator.

A particle filter method can be simplified as:

1. Select the current state
2. Generate the possible new state via a uniform distribution
3. Compare the possible new states with measurement data
4. Select the most likely result

The particle filter is well suited for system of a highly stochastic nature, but it can be very computationally heavy if number of possibilities is set high.

### 6.2.2 Kalman filter

Kalman filter is an estimator which uses state space models and sensor data to calculate the states.

The Kalman can be seen as an algorithm.

1. Previous states
2. Predict change, using models and input signals
3. Correct with measurement data
4. Output states

The Kalman filter is a proven filter, and is simple to implement. For some non-linear systems, a linear estimator is not sufficient, in such case non-linear estimator is needed.

#### Extended Kalman filter

The extended Kalman filter is different from the ordinary Kalman filter, in that it uses a non-linear model to predict the states. This means finding the jacobians at each time step for the current states, which is computationally heavy.

**Unscented Kalman filter**

The unscented Kalman filter gives a better performance in highly non-linear systems compared to the extended Kalman filter. This is due to the method the extended Kalman filter propagates the covariance matrix. The unscented Kalman filter uses a different technique to utilize the non-linear model to propagate all parameters. The unscented Kalman filter also avoids the problem of having to find the jacobians at every time step.

While the unscented Kalman filter in some cases gives better performance, it can also be much heavier than the extended Kalman filter [St-Pierre 04]

**Choice**

The Extended Kalman filter is chosen, as it gives a good estimation for all states. Two versions of the Extended Kalman Filter (EKF) is made in order to investigate if there is a performance difference. The first will be a regular EKF, based on the nonlinear model for prediction and using all the sensors for the update step. The second estimator will be a sensor driven EKF, which is mainly driven by the sensors in both the prediction and update step.

The performances of the two estimators will be evaluated through simulations. It is especially of interest to see; ability to correct wrong initial conditions, effect of model parameters imprecisions, and general precision of the estimations.

## 6.3 Sensor driven extended Kalman filter

The prediction step is driven by differential equations based on the gyroscope and Accelerometer. The update step generates a ful state vector based on the magnetometer and the GOT data. The two state vectors are then mixed via a Kalman gain according to their noise levels. The advantage of a sensor driven EKF is that it does not rely on the system model, and the precise measurement of the parameters in the model, such as inertia and propeller gains. It also makes the estimator applicable to any similar system.

The start state of the quadrotor is the linearisation point at the position measured by the GamesOnTrack system and speed is zero.

### 6.3.1 Prediction step

Non-linear functions are used for the predictions, as they should give a better prediction than linearised models. The data from gyroscope and the accelerometer are used in calculating the predictions.

$$\begin{aligned} \hat{\bar{\boldsymbol{x}}}_{k|k-1} &= \hat{\bar{\boldsymbol{x}}}_{k-1|k-1} + \boldsymbol{f}(\bar{\boldsymbol{x}})_{k-1} \\ \underline{\mathbf{P}}_{k|k-1} &= \underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})} \cdot \underline{\mathbf{P}}_{k-1|k-1} \cdot \underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})}^{\top} + \underline{\mathbf{Q}} \end{aligned} \tag{6.1}$$

Where:
$\hat{\bar{\boldsymbol{x}}}_{k|k-1}$ is the estimated state vector at time step "k" based on time step "k-1"
$\boldsymbol{f}(\bar{\boldsymbol{x}})_{k-1}$ is the discrete non-linear system functions at time step "k-1"
$\underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})}$ is the Jacobian of $\bar{\boldsymbol{f}}$
$\underline{\mathbf{P}}_{k|k-1}$ is the covariance matrix of the error in the estimation
$\underline{\mathbf{Q}}$ is the process noise

The estimator has the same states as the model in Chapter 3, and additionaly has the acceleration in IRF; $[\bar{\boldsymbol{q}} \quad \bar{\boldsymbol{\omega}} \quad \bar{\boldsymbol{P}} \quad \dot{\bar{\boldsymbol{P}}} \quad \ddot{\bar{\boldsymbol{P}}}]$.

$\boldsymbol{f}(\bar{\boldsymbol{x}})$ consists of five functions:
$\bar{\boldsymbol{\omega}}_k = {}^i_b\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_{k-1}) \cdot {}^b\bar{\boldsymbol{\omega}}_{imu} + \bar{\boldsymbol{e}}_w$
$\Delta\bar{\boldsymbol{q}}_k = \frac{1}{2}\Omega(\bar{\boldsymbol{\omega}}_k) \cdot \Delta t \cdot \bar{\boldsymbol{q}}_{k-1} + \bar{\boldsymbol{e}}_q$
$\Delta\ddot{\bar{\boldsymbol{P}}}_k = {}^b\ddot{\bar{\boldsymbol{P}}}_{imu} - [0 \quad 0 \quad -g]^\top$
$\Delta\dot{\bar{\boldsymbol{P}}}_k = {}^i_b\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_k) \cdot \ddot{\bar{\boldsymbol{P}}}_{k-1} \cdot \Delta t$
$\Delta\bar{\boldsymbol{P}}_k = \dot{\bar{\boldsymbol{P}}}_k \cdot \Delta t$

${}^i_b\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_{k-1})$ is the rotation matrix between the quadrotors body and the inertial reference frame. The rotation matrix can be constructed from a quaternion according to [Kuipers 02] as:

$$\underline{\mathbf{R}}(\bar{\boldsymbol{q}}) = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix} \tag{6.2}$$

The $\Omega$ function is defined in equation 3.5

For predicting the covariance matrix, the Jacobian of the $\bar{\boldsymbol{f}}$ is needed. The Jacobian is the first partial derivative of a vector function. The Jacobian $\underline{\mathbf{F}}$ of $\bar{\boldsymbol{f}}$ is made by taking MATLABs Jacobian function, times the time-step plus an identity matrix of equal size. In the first run of the estimator, the covariance matrix $\underline{\mathbf{P}}_{k-1|k-1}$ is set to be a diagonal matrix, with the variance of errors as entries.

### 6.3.2 Update step

The Update step is partly dependant on the GOT data, and those calculations can therefore only be run when new GOT data is available, if it is not then results is set to previous result.

$$\begin{aligned} \tilde{y}_k &= \bar{\boldsymbol{z}}_k - \bar{\boldsymbol{h}}(\hat{\bar{\boldsymbol{x}}}_{k|k-1}) \\ \underline{\mathbf{S}}_k &= \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k) \cdot \underline{\mathbf{P}}_{k|k-1} \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)^\top + \underline{\mathbf{R}} \\ \underline{\mathbf{K}}_k &= \underline{\mathbf{P}}_{k|k-1} \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)^\top \cdot \underline{\mathbf{S}}_k^{-1} \\ \hat{\bar{\boldsymbol{x}}}_{k|k} &= \hat{\bar{\boldsymbol{x}}}_{k|k-1} + \underline{\mathbf{K}}_k \cdot \tilde{y}_k \\ \underline{\mathbf{P}}_{k|k} &= (\underline{\mathbf{1}} - \underline{\mathbf{K}}_k \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)) \cdot \underline{\mathbf{P}}_{k|k-1} \end{aligned} \tag{6.3}$$

Where:
$\bar{\boldsymbol{z}}_k$ is a second set of predicted states based on the magnetometer and GOT
$\tilde{\bar{\boldsymbol{y}}}_k$ is the difference between the two predictions
$h(\hat{\bar{\boldsymbol{x}}}_{k|k-1})$ is the observation function, which maps states to sensors
$\underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)$ is the observation matrix, which maps states to sensors at the current state
$\underline{\mathbf{S}}_k$ is the covariance matrix for $\tilde{y}_k$
$\underline{\mathbf{R}}$ is the covariance matrix of the observation noise
$\underline{\mathbf{K}}_k$ is the Kalman gain, which is used to correct the state estimation

$\bar{\boldsymbol{z}}_k$ is made from the magnetometer measurement ${}^b\bar{\boldsymbol{m}}$ and the GOT measurement ${}^b\bar{\boldsymbol{P}}_{got}$ . The magnetometer is used to find the yaw of the quadrotor. In some orientations the pitch and/or roll can also be found from this measurement, but as this is not valid for all orientations and consists of solving a 3D problem it was chosen to not be done. The yaw was found by: Assuming the earths magnetic field ${}^i\bar{\boldsymbol{m}}$ is know in the given location.

By taking only the measurements from the X- and Y-axis, an angle between the two 2D vectors can be found by:

$$\Phi_z = atan2((^i\bar{\boldsymbol{m}} \cdot {}^b\bar{\boldsymbol{m}}), (^i\bar{\boldsymbol{m}} \times {}^b\bar{\boldsymbol{m}})) \tag{6.4}$$

It is important that both vectors have the same length, they are therefore both divided by their own norm. This is okay because it is the direction of the vectors which is of interest. This method degrades in precision as the pitch and roll increases, this effect is put into the process noise matrix. The ${}^b\bar{\boldsymbol{P}}_{got}$ is a direct measurement of the position state. Taking the first time derivative results in the speed in IRF, $\dot{\bar{\boldsymbol{P}}}$ . Taking the second time derivative of ${}^b\bar{\boldsymbol{P}}_{got}$ result in the acceleration of the quadrotor. The acceleration is directly linked with the pitch/roll and the control signal.

$$a_x = sin(\Phi_y) \cdot U \cdot G1 \cdot G2 \quad \text{and} \quad a_y = sin(\Phi_x) \cdot U \cdot G1 \cdot G2 \tag{6.5}$$

The angle can then be isolated to:

$$asin(\frac{a_x}{U \cdot G1 \cdot G2}) = \Phi_y \quad \text{and} \quad asin(\frac{a_y}{U \cdot G1 \cdot G2}) = \Phi_x \tag{6.6}$$

The newly found yaw, pitch and roll angles are then made into a quaternion. The three angles are also time derived to get an estimate of the rotation speed.

Because rotation quaternions has to be unit, the update of this is modified. The three imaginary parts are used as states, and the real part is made based on these. The kalman gain is multiplied onto the imaginary parts. The quaternion predictions are made such that the real part is always positive. This way the real part can be made by finding the real value between zero and one which makes the quaternion vector have size one. The resulting error quaternion is then quaternion multiplied onto the predicted quaternion.

With this method for doing the update step, the observation function and matrix is 1-to-1, because the update contains a full state vector.

The sensor driven Extended Kalman filter is implemented and tested through simulation in Chapter 7

## 6.4 Standard extended Kalman filter

The standard EKF uses a nonlinear model for generating a full state vector. The states are then compared with the sensor measurements via a sensor model. A Kalman gain is then computed based on the covariance matrices of the prediction and update step, and is finally used to mix the computed states and sensor into the final estimated states vector. As this project uses quaternions some modification will have to be made to the process as quaternions is not linear and commutative.

The procedure for the EKF is:

$$\begin{aligned}
\hat{\bar{\boldsymbol{x}}}_{k|k-1} &= \hat{\bar{\boldsymbol{x}}}_{k-1|k-1} + \boldsymbol{f}(\bar{\boldsymbol{x}})_{k-1} \\
\underline{\mathbf{P}}_{k|k-1} &= \underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})} \cdot \underline{\mathbf{P}}_{k-1|k-1} \cdot \underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})}^\top + \underline{\mathbf{Q}} \\
\tilde{y}_k &= \bar{\boldsymbol{z}}_k - \bar{\boldsymbol{h}}(\hat{\bar{\boldsymbol{x}}}_{k|k-1}) \\
\underline{\mathbf{S}}_k &= \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k) \cdot \underline{\mathbf{P}}_{k|k-1} \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)^\top + \underline{\mathbf{R}} \\
\underline{\mathbf{K}}_k &= \underline{\mathbf{P}}_{k|k-1} \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)^\top \cdot \underline{\mathbf{S}}_k^{-1} \\
\hat{\bar{\boldsymbol{x}}}_{k|k} &= \hat{\bar{\boldsymbol{x}}}_{k|k-1} + \underline{\mathbf{K}}_k \cdot \tilde{y}_k \\
\underline{\mathbf{P}}_{k|k} &= (\underline{\mathbf{1}} - \underline{\mathbf{K}}_k \cdot \underline{\mathbf{H}}(\hat{\bar{\boldsymbol{x}}}_k)) \cdot \underline{\mathbf{P}}_{k|k-1}
\end{aligned} \tag{6.7}$$

$\boldsymbol{f}(\bar{\boldsymbol{x}})$ consists of five nonlinear functions:
$\Delta\bar{\boldsymbol{\omega}}_k = {}^i_b\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_{k-1}) \cdot {}^b\bar{\boldsymbol{\tau}} \cdot \Delta t / I + \bar{\boldsymbol{e}}_w$
$\tau$ is generated by an actuator model which take controls signals as input, it outputs the three torques into $\bar{\boldsymbol{\tau}}$ and also generates the lift $F$ for later. $I$ is the quadrotors inertia, $m$ is the mass.

$\Delta\bar{\boldsymbol{q}}_k = \frac{1}{2}\Omega(\bar{\boldsymbol{\omega}}_k) \cdot \Delta t \cdot \bar{\boldsymbol{q}}_{k-1} + \bar{\boldsymbol{e}}_q$
$\Delta\ddot{\bar{\boldsymbol{P}}}_k = [0; 0; F]^\top / m + ({}^b_i\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_k) \cdot [0 \quad - g]$ $\Delta\dot{\bar{\boldsymbol{P}}}_k = ({}^i_b\underline{\mathbf{R}}(\bar{\boldsymbol{q}}_k) \cdot \ddot{\bar{\boldsymbol{P}}}_k \cdot \Delta t + \bar{\boldsymbol{e}}_a) \cdot \Delta t$
$\Delta\bar{\boldsymbol{P}}_k = \dot{\bar{\boldsymbol{P}}}_k \cdot \Delta t$

The function $\underline{\mathbf{F}(\hat{\bar{\mathbf{x}}}_{\mathbf{k|k-1}})}$ is the jacobian of $\boldsymbol{f}(\bar{\boldsymbol{x}})$ at the current state. The covariance matrix is propagated through this, and process noise is added. The process noise reflects how well the model fits, and can be used to adjust the estimator.

$\tilde{y}_k$ is made by subtracting a transform of the predicted state vector from the sensor measurements. The transform is setup such that it outputs sensor reading at the predicted states. Three of the four sensors fit directly to a state (Gyroscope, GOT and Accelerometer), the magnetometer need a transform.

The magnetometer measures the Earths magnetic field. The magnetic field ($\bar{\boldsymbol{m}}$) is static, meaning the measurement is only dependent on the quadrotor attitude. The predicted measurement can therefore be calculated as:

$$\bar{\boldsymbol{h}}_{\boldsymbol{m}} = {}^b_i\underline{\mathbf{R}}(\hat{\bar{\boldsymbol{q}}}_k) \cdot \bar{\boldsymbol{m}} \tag{6.8}$$

The Kalman gain can now be calculated and put into use. Normally the Kalman gain is simply multiplied onto $\tilde{y}_k$ and the result added to the predicted state, but quaternion require a modification to this process. The three top rows of the Kalman gain matrix results in the three complex parts of a quaternion. And since the quaternion is a unit quaternion (attitude quaternion), the real part ($q0$) can be calculated as:

$$q_0 = \sqrt{(1 - norm([q_1 \quad q_2 \quad q_3]))} \tag{6.9}$$

The resulting quaternion is then added to the predicted quaternion via quaternion multiplication.

This estimator can be tuned by; changing the initial value of $\underline{\mathbf{P}}$ for more or less trust in the initial states of the estimator, or by adjusting the process noise to de- or increase trust in the model prediction. The parameters in the model could also be adjusted for better predictions.

## 6.5 Yaw offset

The estimation can result in an offset in yaw estimation. This can occur if the magnetic field is not as expected. Big yaw errors can result in destabilizing the system. The effect of a yaw offset is investigated through simulation.

Three scenarios is tested; perfect yaw estimation, small deviation and big deviation. In all three scenarios the quadrotors reference is to go to: $x = 2 \quad y = 2 \quad yaw = 45°$ while keeping a steady altitude. The resulting paths can be seen on figure 6.1 below:

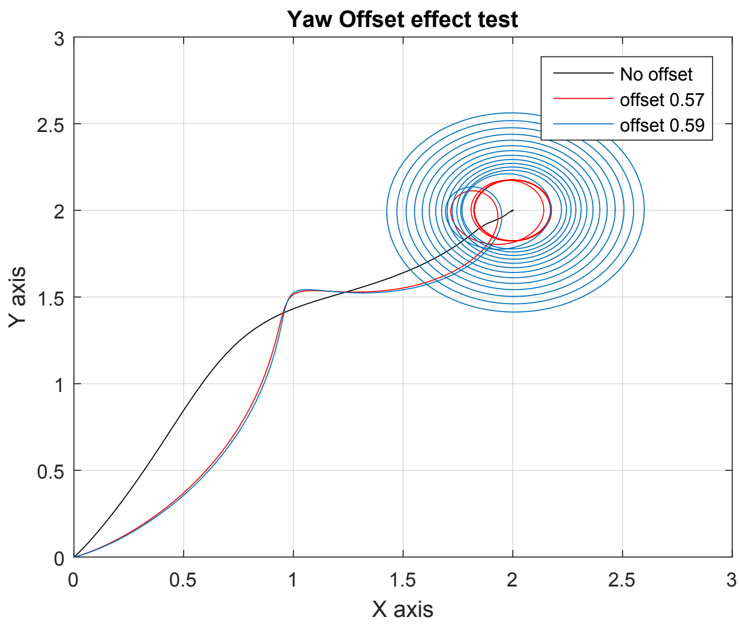


**Figure 6.1:** Data from simulation

It can be seen that small deviation in the yaw estimate leads to circular paths. And a big deviation is seen to make the system uncontrollable. A method of correcting a yaw offset is therefore desired.

First a detection method is needed. It is seen that the path the quadrotor should follow in the x- y-plane is straight line that can be described by a vector in IRF equaling:

$$\bar{\boldsymbol{e}}_{x,y} = \bar{\boldsymbol{r}}_{x,y} - \hat{\bar{\boldsymbol{x}}}_{x,y} \tag{6.10}$$

The speed vector $\hat{\dot{\bar{\boldsymbol{P}}}}_{x,y}$ of the quadrotor in IRF should be parallel to this vector. If the quadrotor is instead flying in a circular path, these vector will be angled on each other. By taking the cross product between these two vectors (normalized), the angle direction can be observed as the z-axis product. A negative angle offset results in a positive output, and positive angle results in a negative output.

The solution to the problem is to subtract the offset from the estimation. As the error is assumed to be a static offset, an integral controller is tried. The output from earlier is integrated over a small tuning constant, and the resulting angle is turned into a quaternion, which can then be added to the estimation. The solution is tested in simulation:

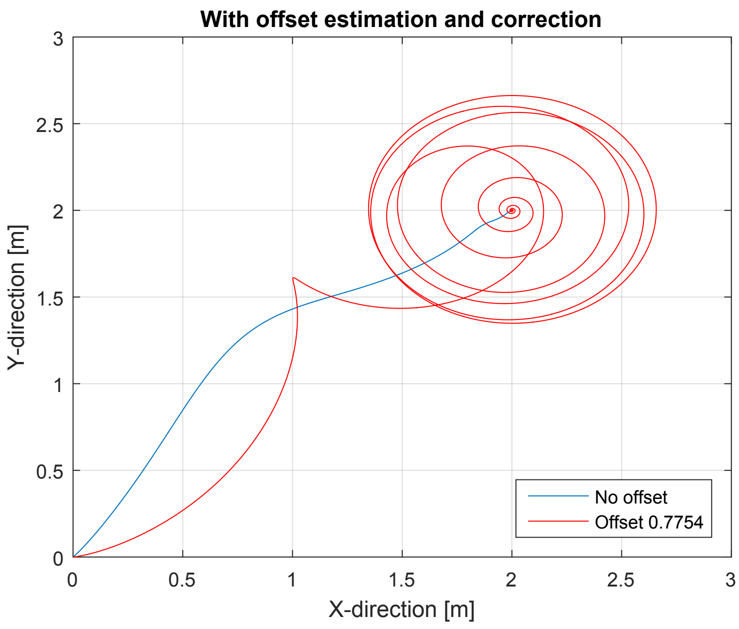


**Figure 6.2:** Data from simulation

It was found that the system could handle much bigger yaw offsets, while still maintaining functionality with no offset, which is important.

### 6.5.1 Conclusion

Based on these simulations, it is highly recommended to implement this type of solution for systems which should go in a straight line towards its target position.

Chapter 7

# Simulation

*Simulation is a common tool in control and automation, as it gives the designer an opportunity to verify the work before implementing anything in the real world. The simulation in this project is based on the nonlinear models found in 3. The simulation is made in MATLABs Simulink environment*

This chapter will briefly describe the setup of the simulation and then go on to outline how it was used for tuning, and finally show the simulated performance of the system

## 7.1 Simulation environment

The simulation environment was split up into several blocks, similar to classic feedback systems diagrams. :Figure:

### 7.1.1 Reference system

The reference system generates the reference states the controller shall attempt to achieve. Normally the height state is set to 10 meters, the quadrotor is given 20 seconds to achieve this height before other states are altered, as it needs to lift of and settle on the height.

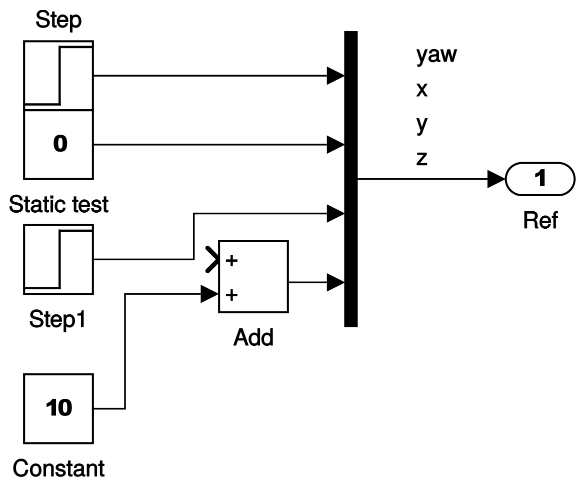


**Figure 7.1:** Reference system in simulation environment

### 7.1.2 Error maker

The error maker system generates the difference between the reference and the feedback. In most systems this is simply a subtraction, but as there are both different reference frames and quaternions involved it gets a bit more complicated.

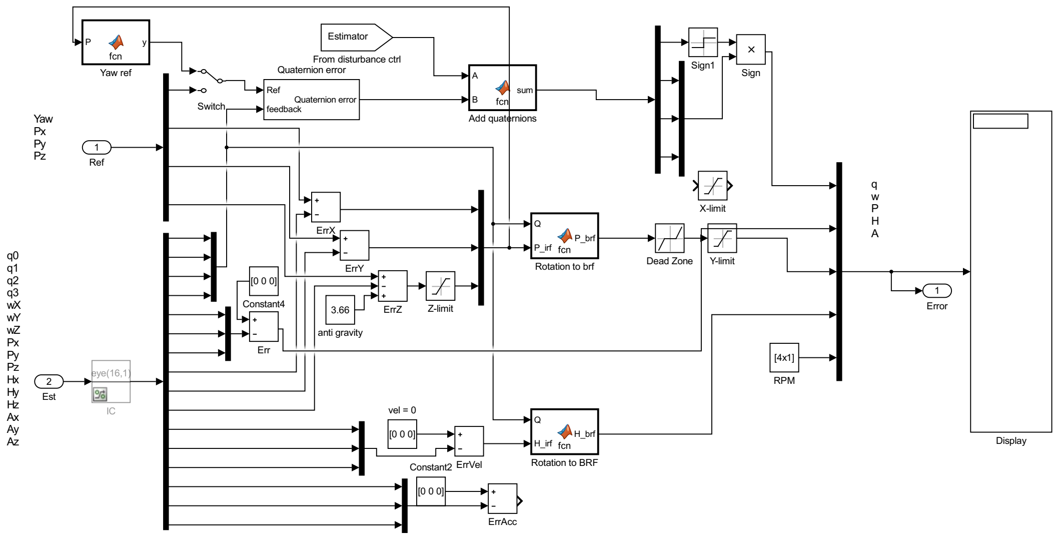


**Figure 7.2:** Error maker system in simulation environment

Included in the generated error is an anti gravity, which should remove most of the steady state error in height. The disturbance controller also gives as part of its control signal a rotation needed to counter disturbances, this is also subtracted in this block.

### 7.1.3 Controller

The controller block mainly contains the LQR controller. The controller is a matrix which is multiplied onto the error vector, which generates four control signals. The control signals should be between zero and one as it represents a percentage of thrust, and the signal is therefore passed through a limiter to prevent more than 100 percentage thrust. In addition this block also contains the disturbance controller.

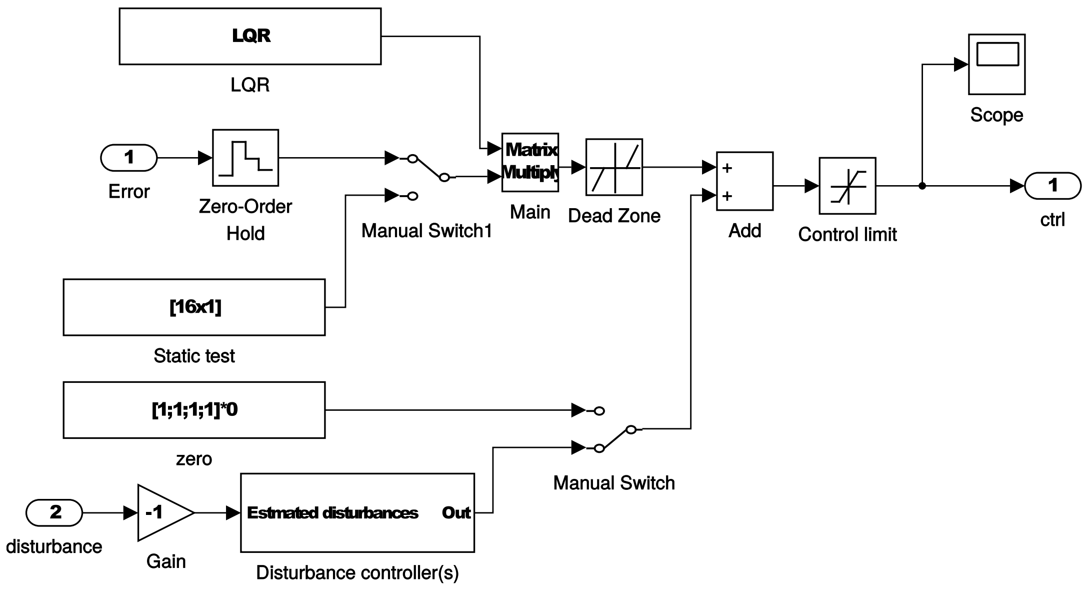


**Figure 7.3:** Control system

### 7.1.4 Motor

The motor block contains both the motor and rotor model, and it mixes the rotor outputs into forces in the BRF. The models are non-linear.

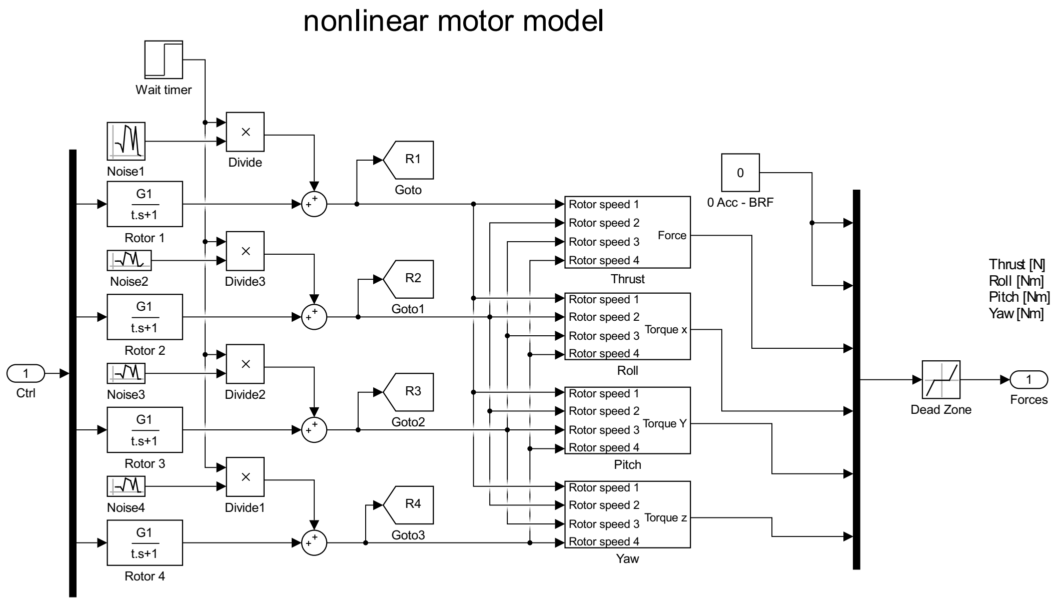


**Figure 7.4:** The motor block translates from control signals to forces and torques

The model cannot generate forces in the x- and y-axis in the BRF, therefore these are set to zero.

### 7.1.5 Body

The body block is the main physics simulator in the simulation. It takes forces and torques as inputs, and output all the states of the quadrotor.

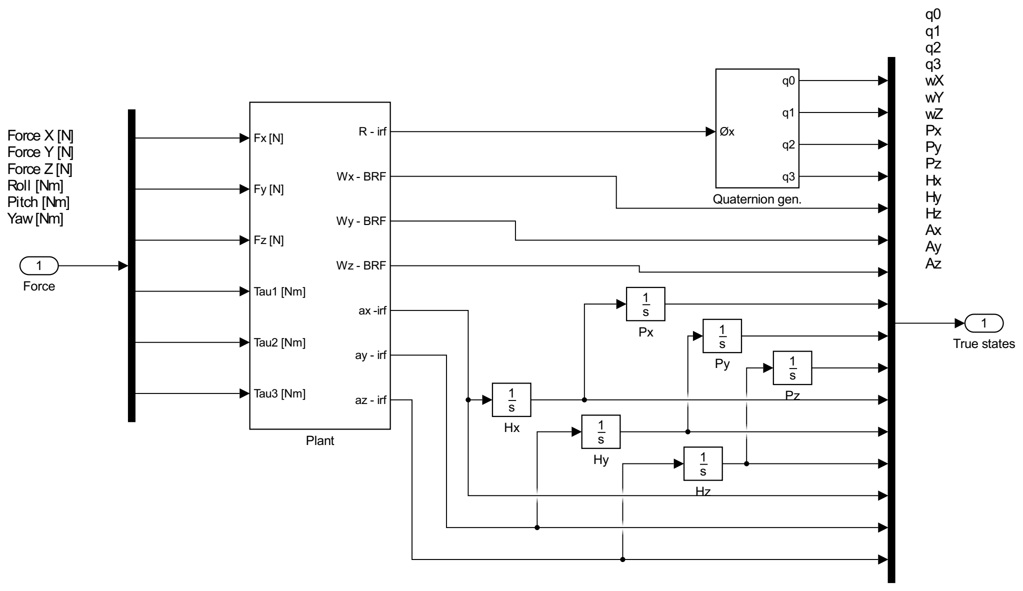
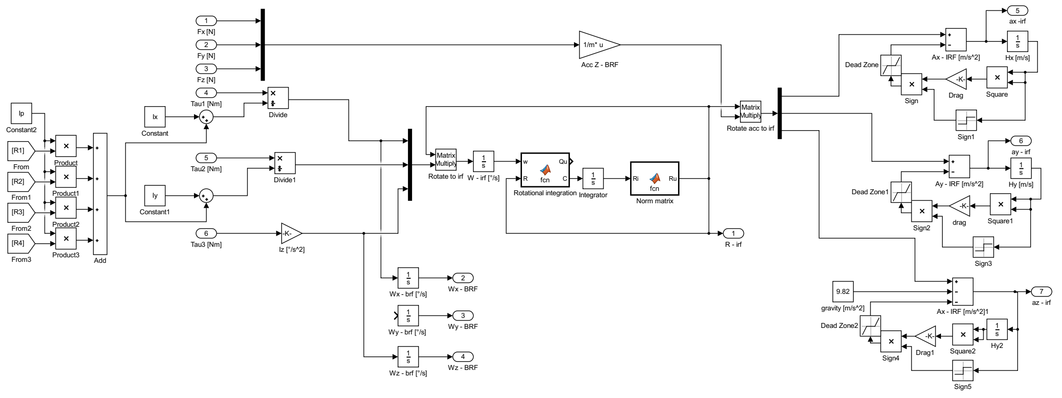


**Figure 7.5:** The body model takes forces and torques as inputs, and outputs the states of the quadrotor

The plant model is non-linear, and contains both gravity and air drag.



### 7.1.6 Sensors

The sensor blocks is meant to simulate the real measurements made by the sensor systems. This means measuring in the correct reference frames, and adding realistic noise to the sensors.

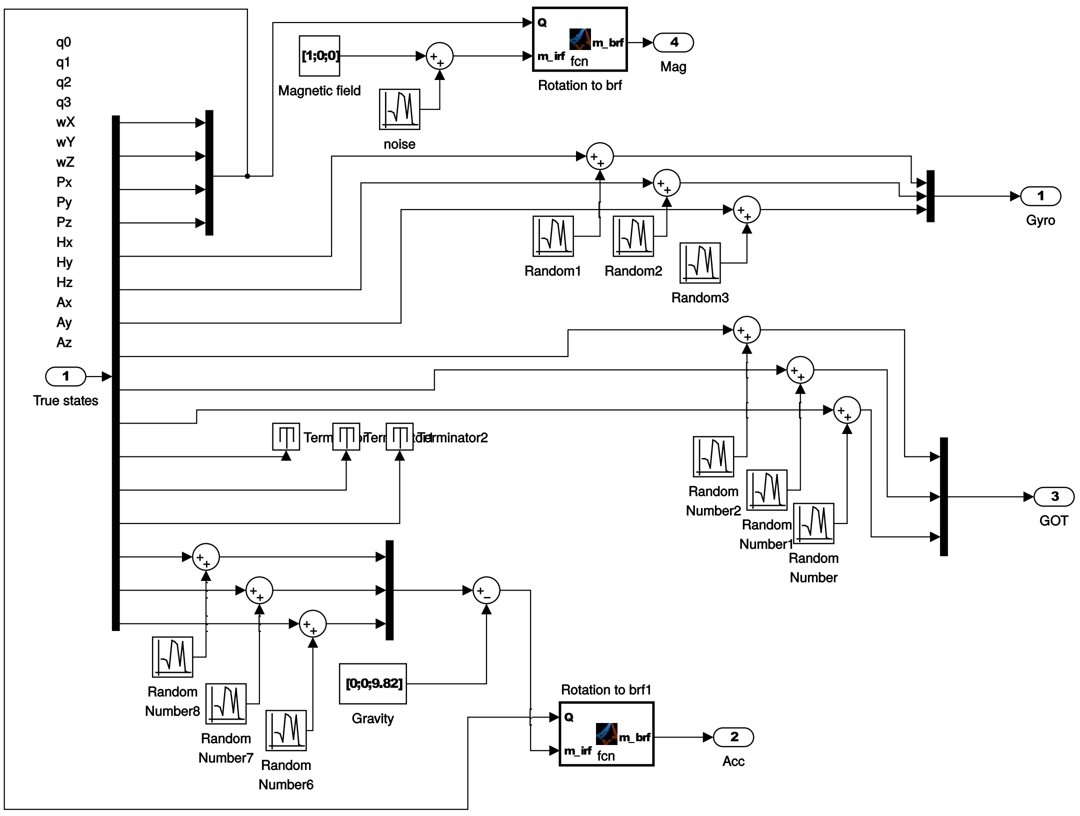


**Figure 7.6:** The sensor block takes the true states and output realistic representation of what a sensor would record

### 7.1.7 Estimator

The estimators, as described in Chapter 6, is an extended Kalman filter and a sensor driven Extended kalman filter. Because they are very complex and filled with matrices, it was built as a Matlab function. The estimator is run at 50 Hz

## 7.2 Tuning

One of the primary uses and reasons why a simulation is made in this project is for pre tuning the system. Simulation is easy and fast to reset between runs. There are three systems which needs tuning in this project, the LQR controller, the estimator and the disturbance controller. Additionally a good estimate of the anti gravity value can be found through simulation. Before any tuning can begin, the environment has to be verified first though.

### 7.2.1 Simulation environment verification

The blocks which are interesting here is the motor and body blocks. It is important for the conclusions of the simulations that these represent the real behaviour of the quadrotor as close as possible. This could be done with real data, sending in control signals and comparing the output states to real measured data. In this project another approach has been used, due to lack of applicable data. The control signals are manually set such that the outputs are easily predictable.

### Motor verification

Each motor can take one input and outputs an RPM. The RPMs are then mixed into the forces and torques the motors combined generate. The verification process will be a series of small tests, with given inputs and expected outputs.

In the first test all the input are set to 70% and then after 5 seconds drop to 60%. The resulting output should be a force upwards, which is lowered after 5 seconds. The result from the test was exactly as expected.

The second test is of the torque generation. From section 4.1.2 it can be seen that the expected result would be that if signal one and three is lower than two and four, then a positive torque on the z-axis should appear (as well as some force upwards). The resulting torque from this test can be seen on Figure 7.7 below.

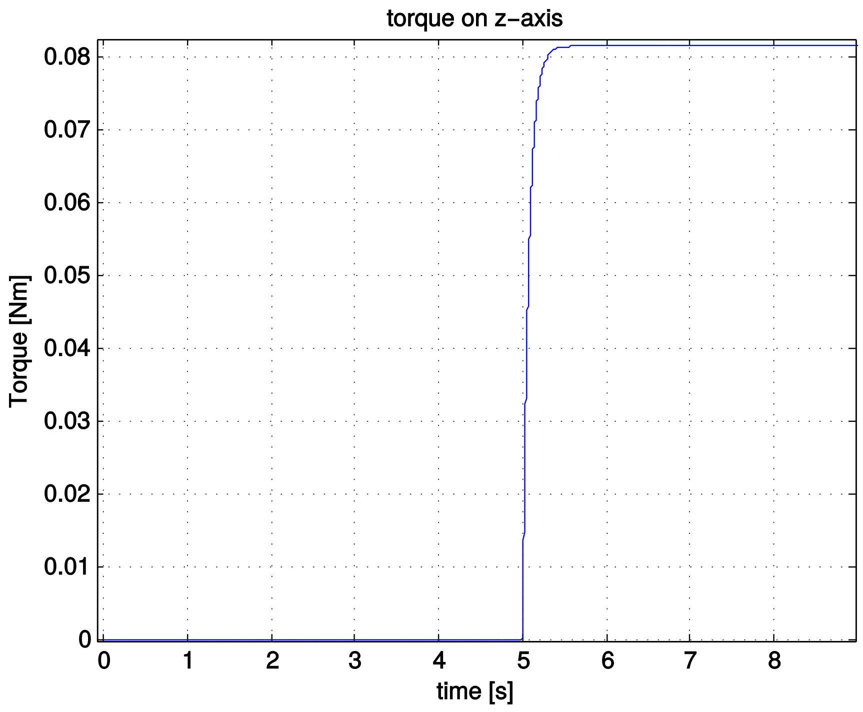


**Figure 7.7:** Data from simulation

Only a torque around the z-axis and force upwards was observed, as expected. The pole in the motor can be seen in the plot as well, and it fits the measured pole.

Lastly the roll and pitch torques are tested. The control signals are again based on section 4.1.2. It will be set to first roll (x-axis) then pitch (y-axis).

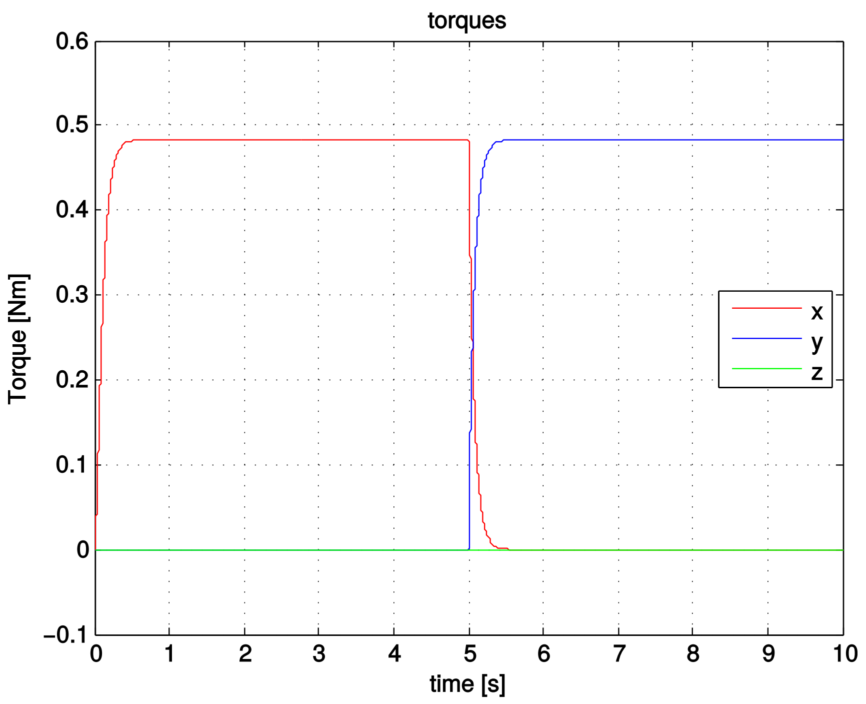


**Figure 7.8:** Data from simulation

The output from the motor block is as expected. It can therefore be concluded that the simulation of the motor block is valid.

### Body verification

The body model take forces and torques as inputs, and output all the states of the system. The verification of this block is especially focused on the rotational effects.

The first test is to check the start orientation, and the rotation of thrust. The body is exerted to a thrust, and then after 10 seconds a torque on the z-axis. The result from this should be upwards acceleration (given thrust is big enough), and when the torque is applied an increasing rotation speed around the z-axis in IRF. The rotation around the z-axis should not affect the upwards acceleration. The acceleration was seen to be constant throughout the test. The rotation was observed by rotating a simple [100] vector by the rotation matrix, resulting in the graph seen below in Figure 7.9

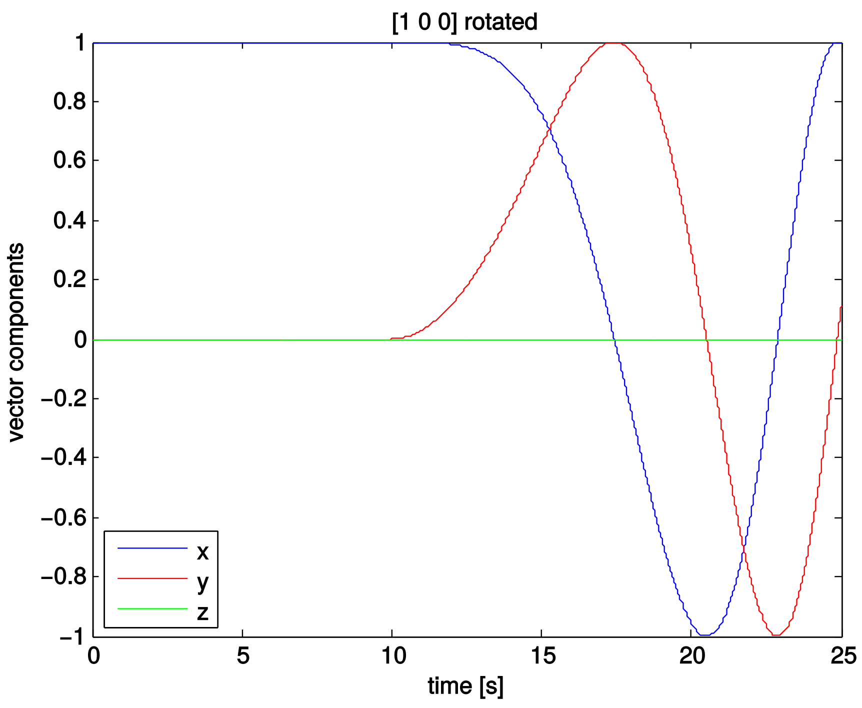


**Figure 7.9:** Data from simulation

The Figure shows a response similar to the expected response.

The second test checks rotation of rotational speed between the two frames. This is done by rotating around the z-axis by 90°, and then rotating around the y-axis (in BRF. The expected result from this is rotation around the x-axis in IRF in the negative direction. The result is again seen on the effect on a [100] vector in Figure 7.10 below

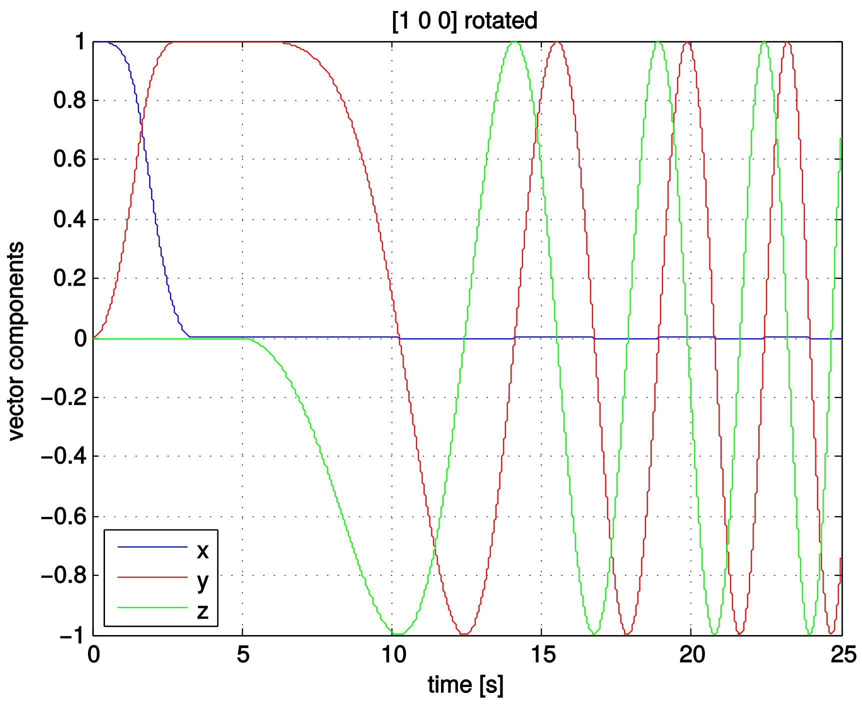


**Figure 7.10:** Data from simulation

It can be seen that the vector is first rotated 90° around the z-axis finishing around 4 seconds. Then the system is rotated negatively around the x-axis, completely as expected.

The rotation of the accelerations was done in a similar manner, and where found to be functioning as intended.

Lastly in the body block, the rotation matrix is translated into a quaternion. The quaternion should describe rotation from IRF to BRF ($^{i}_{b}\bar{\boldsymbol{Q}}$). The same control signals are as used in the second test. Interpreting quaternions with more than one axis in use can be difficult, instead a simple vector is rotated by the quaternion. The resulting vector behaves identical with the result from the second test, and the quaternion generation is thereby correct.

In conclusion; the body model functions correctly, and is ready for use.

The conclusion is that this simulation environment is a sufficient representation of the real system, and it can therefore be used for tuning and preliminary testing of the system.

### 7.2.2 LQR tuning

As mentioned in section 4.2, the LQR controller is tuned in the simulation environment. This is done to be more efficient, and prevent unnecessary damage to the real quadrotor. The simulation is set to use the true states as feedback. During the tuning, attention will be put on control signals size (should be between 0.1 and 1), overshoots and settling times, as well as general stability of the system.

First the reference is set to 10 meter height, zero yaw. This equals take-off. The $\underline{\mathbf{Q}}$ and $\underline{\mathbf{R}}$ matrices are tuned such that this is achieved in about 10 seconds without overshoot, and the control signal does not go above one.

Thereafter the quadrotor is stepped in a directional flight. It was seen that big step would make the attitude change too much and crash the quadrotor, so a limit on the position error was instituted, and set to 5 meters. This kept the system from crashing. The controller then had minor adjustments to prevent too much of a height loss when doing movement. The limit on the error does not prevent the system from reaching the reference.

Lastly the yaw was stepped, while there was still an error on the position. Firstly it was confirmed that the position rotates correctly as the quadrotor rotates. Minor tweaks was made to the controller for settling time and overshoot of the yaw rotation

The quadrotor can in simulation take off, fly to a given position while yawing freely, with state feedback.

### 7.2.3 Estimator tuning

The extended Kalman filter can be tuned by setting the noise levels and covariances, and set the initial values of the state error covariance matrices, which in practise translates to defining whether the filter should trust the prediction over sensor measurements.

The filter was tuned initially while the system got the true states as feedback to the controller, and the references made the quadrotor take off and fly around while yawing. This way the true states could be compared with the estimators output on a stable system.

Finally the system was set to use the output of the estimators as feedback to the error maker.

## 7.3 Simulations

The simulation environment is used to verify and measure the performance of the system design. Different scenarios will be set up for the various parts of the system.

### 7.3.1 Experimental disturbance controller

The experimental controller found in Section 9.1.2 is setup up in the environment. For the purpose of removing noise and getting the best view into the controllers performance the state feedback is used for this testing, instead of the estimator.

The scenario used for the test is a movement from hover in both the x- and y-direction from hover, and later a rotation around the z-axis. This scenario is chosen because it uses all the states of the system to achieve the end result, and is thereby vulnerable to all disturbances.

#### Clean run

The first run is without any disturbance, and no action from the disturbance controller. The result from this run will act as a baseline performance for the later runs. The reference they are set to follow is: after 20 seconds go to position $x = -5\mathbf{y} = 5$, after 40 seconds rotate 22.5 degrees. The disturbances are always activated after 25 seconds.

The resulting position and quaternion can be seen on figure 7.11 and 7.12

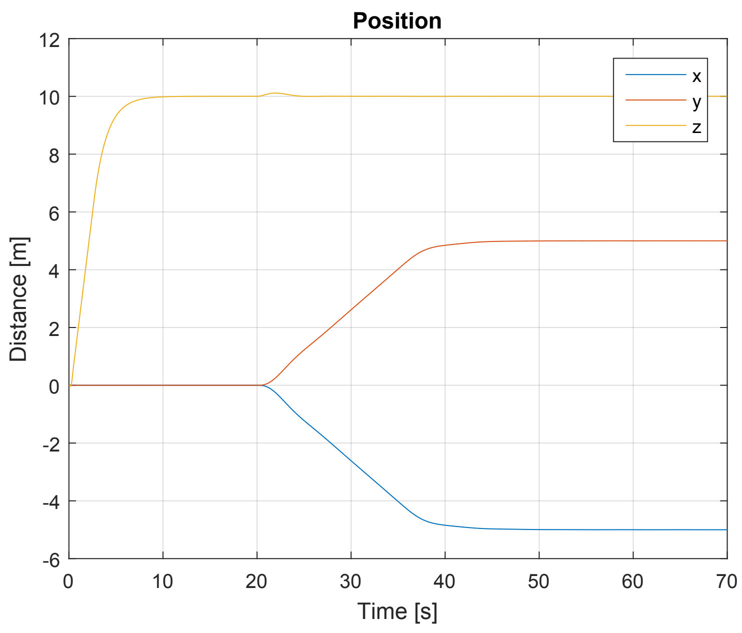


**Figure 7.11:** Data from simulation

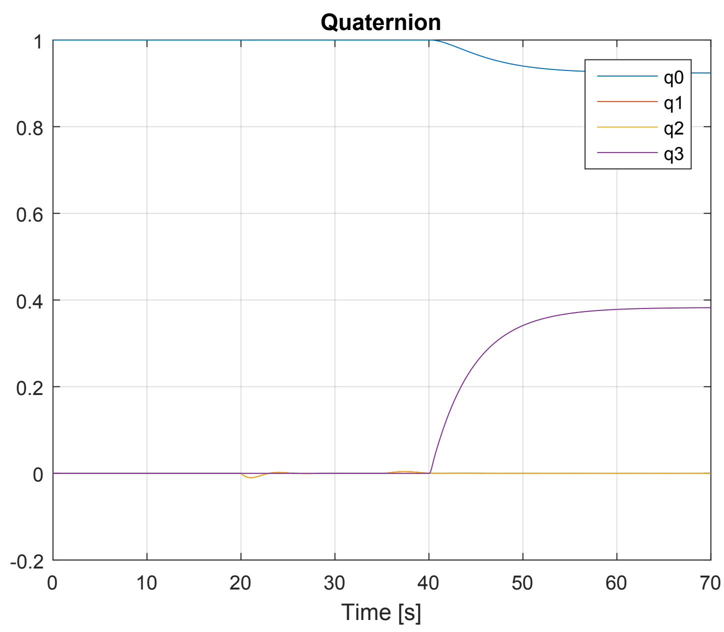


**Figure 7.12:** Data from simulation

It takes 21.65 seconds to reach within 0.1 meter from the x reference, 21.59 for the y reference and 13.47 to reach within 95% of the rotation reference.

### No control with disturbance

A series of runs with disturbances, but with no response from the disturbance controller is made, to establish how the disturbances would normally affect the quadrotor. The size of the disturbance is set such that the controller is expected to have good performance. The results is put into a table below.

**Table 7.1:** Comparison table

| Ref | $P_x$ (0.4N) | $P_y$ (0.4N) | $P_z$ (-10N) | $\tau_x$ (0.4Nm) | $\tau_y$ (0.4Nm) | $\tau_z$ (-0.1Nm) | **Baseline** |
|---|---|---|---|---|---|---|---|
| X | 79.44 s at-4.06 | 32.89 s at -5.65 | 21.6 s | Failed | Failed | 22.61 s | 21.65 s |
| Y | 79.47 s at 5.7 | 31.4 s at 5.66 | 21.53 s | Failed | Failed | 28 s | 21.59 s |
| Yaw | 13.47 s | 13.47 s | 13.54 s | Failed | Failed | Failed | 13.47 s |

In the z-direction test, there was an offset on the z-axis of $-42.56$ cm. In the Z-torque test, the system never achieved a stable attitude on the z-axis.

### Control with disturbance

The disturbance controller is turned on, and the same tests are run again. The results can be seen in the table below

**Table 7.2:** Comparison table

| Ref | $P_x$ (0.4N) | $P_y$ (0.4N) | $P_z$ (-10N) | $\tau_x$ (0.4Nm) | $\tau_y$ (0.4Nm) | $\tau_z$ (-0.1Nm) | **Baseline** |
|---|---|---|---|---|---|---|---|
| X | 92.42 s at-4.64 | 29.79 s at -5.95 | 21.66 s | 22.55 s | 24.99 s | 21.8 s | 21.65 s |
| Y | 97 s at 5.99 | 33.73 s at 5.36 | 21.59 s | 25.44 s | 22.05 s | 21.41 s | 21.59 s |
| Yaw | 13.5 s | 13.5 s | 13.54 s | 13.46 s | 13.46 s | 16.81 s | 13.47 s |

The z-direction test resulted in an offset of +8 cm.

### Conclusion

The experimental controller has now been tested through simulation. The primary result is that the controller makes the system stable for disturbances of these sizes and smaller.

This controller is not good at handling forces along x- and y-direction.

The controller has some loss of performance. Up to 18% slower in achieving the reference.

## 7.3.2 Flight scenario

This simulation aims to show the systems ability to fly using the LQR controller and the estimators.

The reference in all runs is: after 20 seconds go to position $x = -5\mathbf{y} = 5$, after 40 seconds rotate 45 degrees.

### Static Kalman gain

In this run a static Kalman gain is used in the Extended Kalman filter. The true values of the states was recorded, and the position can be seen in figure 7.13

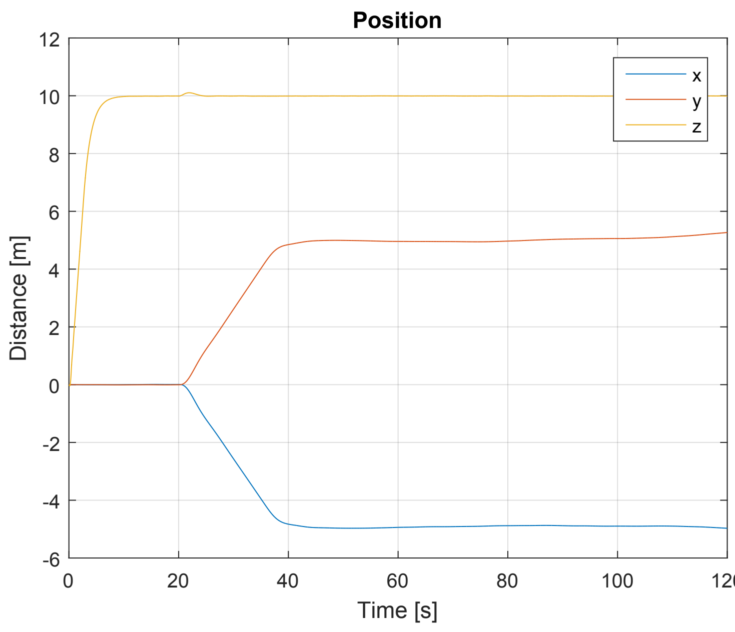


**Figure 7.13:** Data from simulation

In general was the performance good, but beyond the 80 seconds mark it can be seen that the position starts to drift. This is found caused by the offset on the speed estimate which is not corrected properly.

### Extended Kalman filter

In this run the extended Kalman filter was used, the resulting true states can be seen on figure 7.14

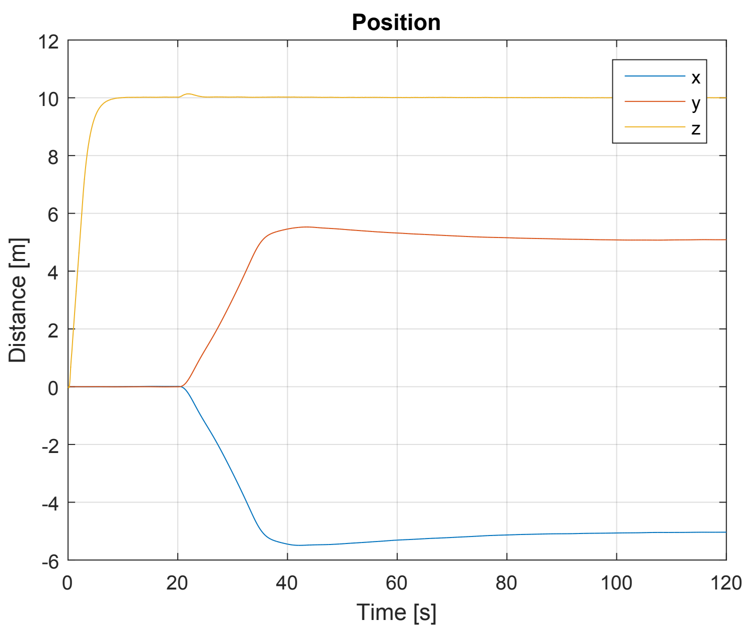


**Figure 7.14:** Data from simulation

The performance is good, and there is no offset on the position. In general this is a good performing estimator. One of thee weaknesses of the extended Kalman filter comes to play when the system deviate from the model. Another run is therefore made with a disturbance on the Z-axis. The resulting positions can be seen on figure 7.15

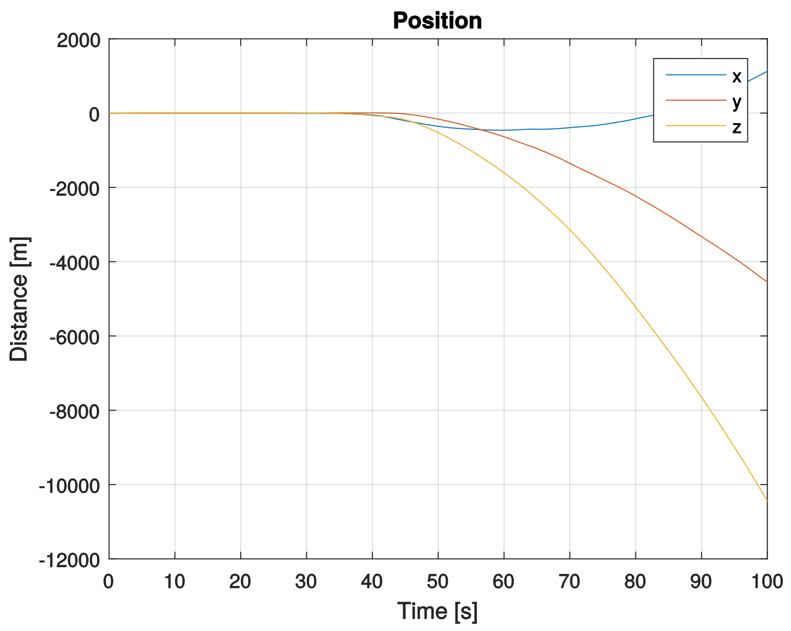


**Figure 7.15:** Data from simulation

As it can be seen the system is not stable in this configuration. This is because the model cannot account for the disturbance.

Including the disturbance model in the extended Kalman filter might solve this problem.

### Sensor driven Extended Kalman filter

Lastly the sensor driven extended Kalman filter is used as estimator. First a normal run is done, and the resulting positions can be seen on figure 7.16

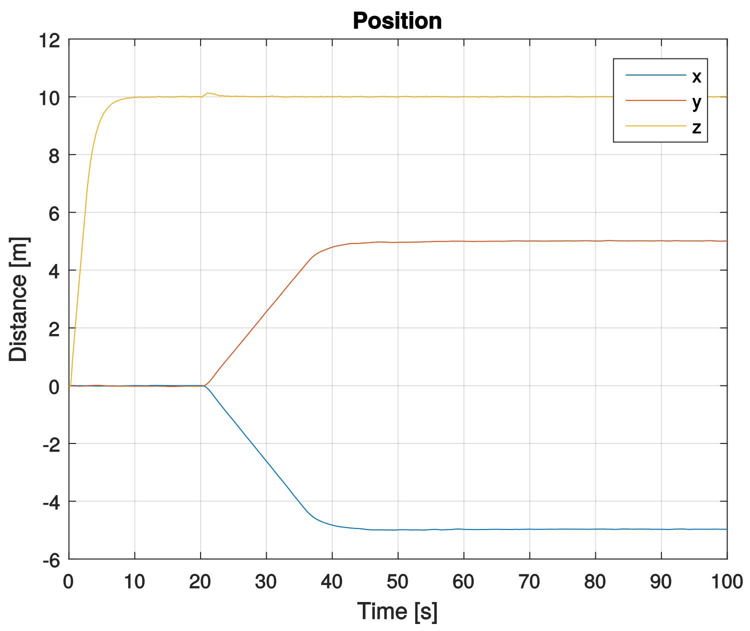


**Figure 7.16:** Data from simulation

The performance is good, and there is no offsets. One of the advantages of the sensor driven extended Kalman filter is that it is not model dependant. It should therefore not be vulnerable to disturbances. A run with a disturbance on the Z-axis is therefore run, the result can be seen on figure 7.17

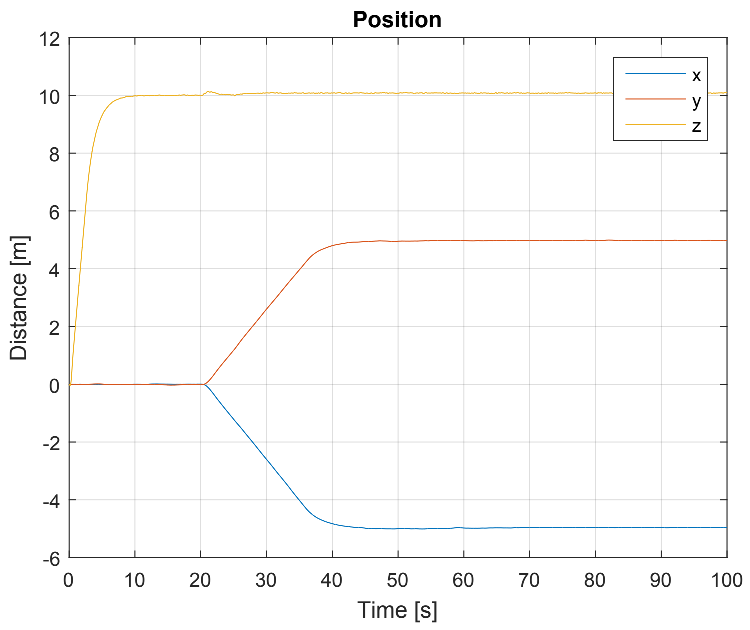


**Figure 7.17:** Data from simulation

The system was stable and kept good performance despite the disturbance.

## Conclusion

The conclusion drawn from the flight scenarios is that the system is stable. With no disturbances the LQR and either EKF or sensor driven EKF is sufficient for flight. When disturbances are introduced either the EKF needs to be approved else the sensor driven EKF is needed.

CHAPTER 8

# Implementation

*The implementation of this project into the real world was unsuccessful. This chapter describes the design of the intended implementation, and what problems this design ran into.*

It was originally the plan to put the estimator and controller(s) on the quadrotors Arduino Mega 2560, but the heavy matrix operations would not be able to run on this. It was therefore decided to run the estimator and controller on a PC, and the uplink the control signals to the quadrotor, while downlinking sensor data to the PC. The system can be seen on Figure 8.1 below.

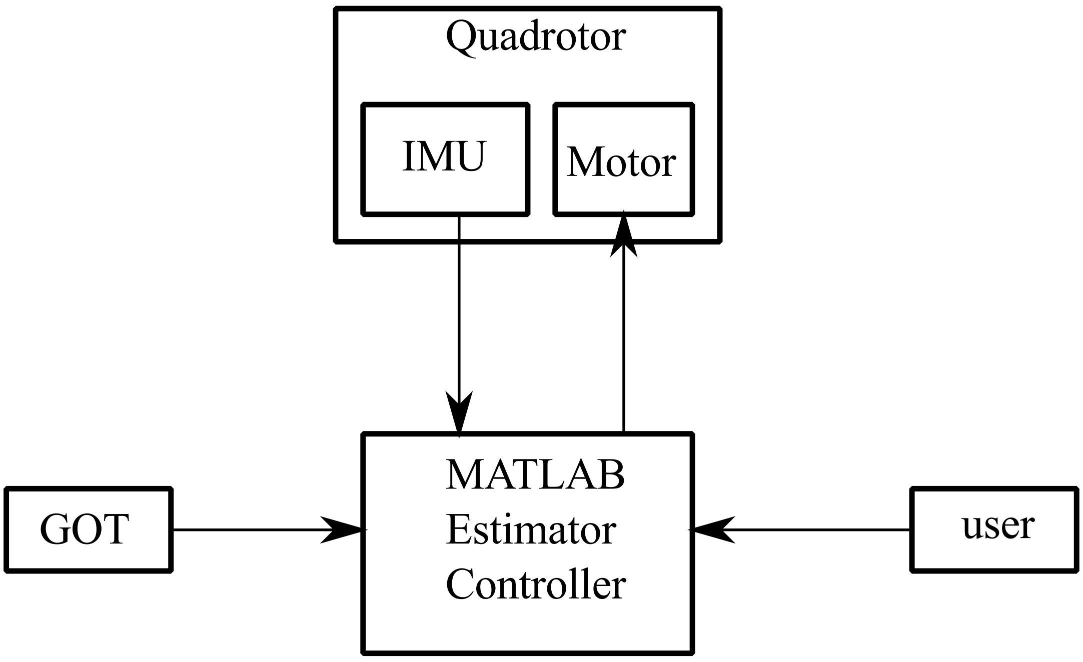


**Figure 8.1:** System overview

## 8.1 System setup

The systems on Figure 8.1 has to be implemented and setup to interact with each other.

### 8.1.1 GamesOnTrack system

The GOT system used in this project is a test version on loan. The system handles the positional measurement of the quadrotor. The quadrotor is equipped with an ultra sonic transmitter, which is timed with the GOT system via a wireless link. The ultra sonic wave is received by several ultra sonic receivers which are placed in a calibrated pattern around the room. The distance between each receiver and the transmitter can then be calculated, and used for trilateration of the position of the transmitter in a 3D environment. This calculation is done in an Software Development Kit (SDK) on a PC. The SDK is edited such that the position is low pass filtered, as the system sometimes generates large errors. And then transmits its results to a MATLAB process.

### 8.1.2 Radio link

The quadrotors on-board Arduino needs to be able to get data to and from the PC, a wireless radio link is therefore setup. The radio is an XBee S1, and is setup to be "transparent", meaning it passes data straight through and to the Arduino and the PC it looks like they are directly connected.

The radio link will be used to pass data and control signals. To simplify the communication handling, a standard package is used for both upload and download. The package consists of 17 bytes.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Start | Ctrl | U1 | U2 | U3 | U4 | mx | my | mz | ax | ay | az | gx | gy | gz | test | End |

**Table 8.1:** The arrangement of data in each package

The radio link ended up being one critical components which prevented a full implementation, as this specific one did not allow faster communication than 10Hz, which is to slow for stable controller.

## 8.2 Arduino

The Arduino Mega 2560 was chosen because it is easy and quick to program. It was the hope that by using this simple platform, future projects would be easier. The Arduino in this project shall take data from the IMU and transmit them to the PC, while receiving motor signals from the PC and translating these to PWM for the motors ESC.

For a control loop to function properly it is important to control the frequency of the control loop, but as the control signals come from the PC, the Arduinos loop should run as fast as possible to induce the smallest amount of delay from the PC to motor speed change.

As a safety feature a communication watchdog was implemented. It tracked how many failed communication attempts had occurred in a row. And when this number ran over a limit it would trigger the stop mode.

#### IMU data

The IMU is mounted directly to the Arduino on a custom shield, made by the UA-world project. It returns the three sensors values as floats. These data are stored in the next downlink data package, if new sensor data is not ready then the value in the package is set to maximum, such that the PC can ignore them.

#### Motor control

The Arduino is directly connected to the ESCs. These are controlled by PWM in a specific interval. The code in the arduino is setup such that when flying the motor are never stopped fully.

When the quadrotor is first initialised, a calibration of the ESCs are needed. This consist of setting the PWM to the minimum value for $1500ms$ and then increasing the value for an additional $1500ms$. This enables the motors, which is confirmed by a beeping noise from each motor.

#### Communication handler

The communication handlers task is to ensure that all incoming commands and data are received and stored, while also sending data packages to the PC.

It ensure the validity of incoming packages by searching incoming data for the start byte of a package, it then checks if the end byte is in the correct position relative to the start byte, and if yes passes the package as being valid. This system could be improved by calculating a checksum and adding it to the package from the source, and then confirming the content of the package.

The PC in the system has a similar communication handler.

## 8.3 Matlab / PC

A PC is setup with matlab for the purpose of collecting and processing data from the GOT system, the quadrotor and for verification purposes data from the VICON system. The PC is also used as user interface, which is also programmed in Matlab.

**Initialization** The Matlab script start by setting variables, starting the user interface and connecting to the GOT, VICON and the Xbee module. The GOT system is connected by starting its SDK, which has been modified to send its output data to a local port, which Matlab can read.

**User input** The user input is made as a Matlab figure with a selection of buttons that can be clicked, and key presses.

The buttons controls the “Ctrl” variable, which decides the flight mode to be in. The modes are:

1. Stop
2. Calibrate
3. Calibrated (hidden)
4. Initialize
5. Fly

Key presses can be used to change the reference, such that the user can control the position and yaw reference.

**Communication handler** The communication handler checks bytes available from the Xbee module, if more than 15 is available, then it checks for start and end bytes being

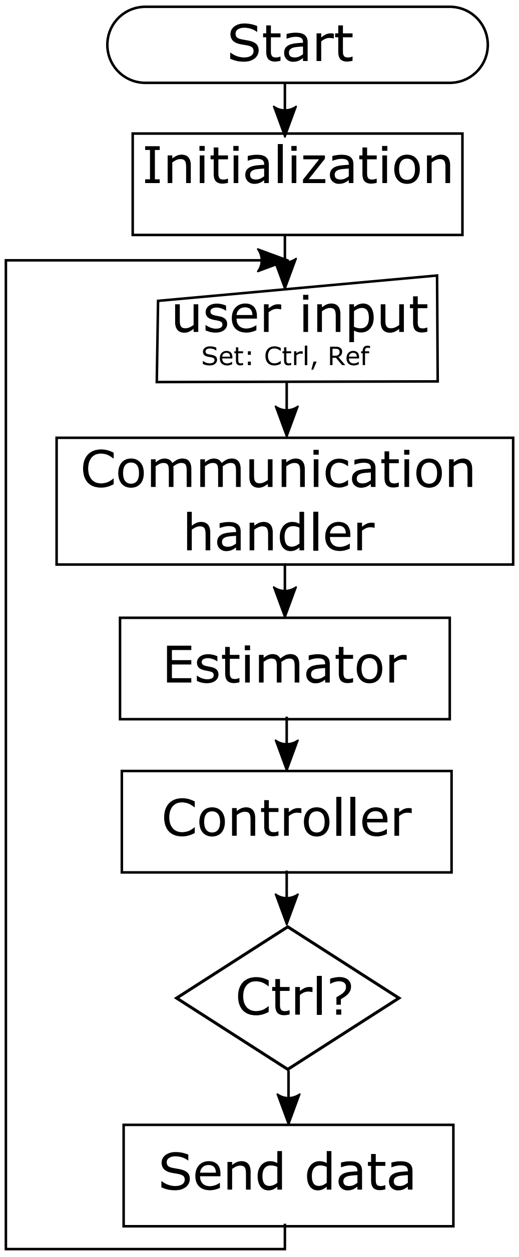


**Figure 8.2:** Matlab flowchart

placed correctly. If the package is setup correct, then it is accepted and copied into local variables. If not then the package is discarded.

**Estimator** The estimator gathers data from the IMU and the GOT system, and applies the estimator described in Chapter 6. This results in a full set of states.

The states are subtracted from the reference through an error maker just like in subsection 7.1.2.

**Controller** The controller is simply a matrix multiplication between the error vector and the LQR matrix.

Depending on the "Ctrl" variable the script either send the control signal made by the controller, or sends the lowest value. This is made for the modes where the quadrotor should not fly.

## 8.4 Conclusion

A lot of time was used on the implementation. In the end it was found that it was not a good option to implement it on an Arduino, as it simply could not run the required operations fast enough.

When coding from scratch it is very important to build in failsafes, as quadrotors can be very dangerous when turned on inside a room with a human. These failsafes cannot depend upon user inputs or wireless links.

It was also found that starting up the motor to their lowest RPM before going to flight mode was a good idea, as the controller is not meant to operate from zero.

Chapter 9

# Closure

*This chapter serves as round up of the project, and it will collect the conclusion on the solutions and problems found throughout the project*

## 9.1 Discussion

Many areas within modern quadrotor flight was investigated throughout this project.

### 9.1.1 LQR controller

The LQR controller was in this project used as the main controller for flight. It is a simple and safe choice, and it is simple to tune. It performed well, achieving stable flight in normal simulation scenarios. Disturbance scenarios showed the weakness of this approach, as seen on table 7.1. The LQR is not optimal for handling disturbances which is unknown to its state space model.

LQR control is a good start choice for future projects, but better solutions should be striven for.

### 9.1.2 Disturbance handling

The experimental controller was an attempt at handling the disturbances separately from the quadrotors states. The controller showed its worth in simulation against disturbances, as seen on table 7.2. It allowed for flight while under disturbances which normally made the system uncontrollable.

More work is needed on verifying the approach, as it is not proven stable or optimal in this project.

The MPC approach was investigated, and seems to be a viable option. This approach needs to be tested to verify this claim. It could also be interesting to see if the performance increases with the use of a non-linear MPC.

### 9.1.3 Estimators

Three estimators was examined in this project; a Static Kalman gain EKF, an EKF and sensor driven EKF. These where simulated in Simulink, with realistic sensor models.

The Static Kalman gain EKF had decent performance, but shoved a tendency to have a deviating speed estimate, which led to a position drift.

The EKF had good performance, giving an estimation of the states which was precise enough that the system could remain stable and reach its reference roughly as fast as with state feedback. It did not however handle disturbances well. Further work could be put into incorporating the disturbances, to solve this problem

The sensor driven EKF had good performance very similar to the EKF. It handled the estimation perfectly even with disturbances. The sensor driven EKF also have the advantage of being easily transferable to other systems. It could be interesting to see if the estimator works as well on other systems.

### 9.1.4 Yaw Estimation

The estimation of yaw is important for precise flight, especially indoors. A system which can fix any estimation errors is therefore highly useful. The solution found improved the flight path dramatically, and was able to reach the target destination with much bigger initial errors, see figure 6.1 and 6.2

A solution like this one should always be present in quadrotors.

### 9.1.5 Simulations

The simulations ended up being the main tool of this project. After having established a good environment, it was used to tune both controllers and estimators.

It would be a good addition if the plant in the simulation environment could be compared to real life data, for verification purposes. The plant model also needs a drag coefficient to enable this effect.

### 9.1.6 Implementation

The implementation was unfortunately not achieved at the time of finish this report, as several problems in the system slowed this process down.

An attempt at implementing some of the solution for real world validation is planned.

### 9.1.7 Problems

In Section 1.1 a series of problems where given.

**How does a quadrotor fly and operate at/between different positions with different orientations.**
By having a good estimate of the states, and keeping track of the reference frames, operation in all yaw orientations can be achieved.

**How does a system calculate its states when not all are measured.**
Using an estimator, like the sensor driven EKF, a good state vector can be achieved.

**How can an autonomous controller handle physical disturbances.**
If the disturbance is known it can be fed into a controller based on the systems models. This could be a model predictive controller or a controller similar to the experimental controller in this project.

# Bibliography

[Kuipers 02] Jack B. Kuipers. Quaternions and rotation sequences. Princeton University Press, 2002.

[Maciejowski 02] J. M. Maciejowski. Predictive control with constraints. 2002.

[Nielsen 15] Carsten Nielsen. *Droner rykker indendørs med dansk teknologi.* `http://www.aau.dk/nyheder/alle-nyheder/vis/droner-rykker-indendoers-med-dansk-teknologi.cid154989`, 2015.

[St-Pierre 04] Mathieu St-Pierre & Denis Gingras. *Comparison between the unscented Kalman filter and the extended Kalman filter for the position estimation module of an integrated navigation information system.* 2004.

[Symon 71] Keith R. Symon. Mechanics, third edition. Addison-Wesley"publishing company, 1971.

[Thomsen 14] Brian Gasberg Thomsen, Jens Nielsen, Mikael Juhl Kristensen & Nikolaj Holm. *Aau Quad Swarm.* Student worksheet compilation, 2014.

[Thomsen 15] Brian Gasberg Thomsen, Jens Nielsen, Mikael Juhl Kristensen & Nikolaj Holm. *Sensor Fault Tolerant Quadrotor Flight in GPS Denied Environments.* Student project, 2015.

Appendix A

# MPC script

*This appendix contains the matlab script needed for MPC, the script was originally intended to be used in simulation, but this was not compatible.*

```
function u = fcn(ref,est)
%#codegen
%Setup
    p=6; %prediction horizon
    c=5; %Control horizon
persistent Ap Bu Bdu g H R Q U
if isempty(Ap)
    U=[0.604 0.604 0.604 0.604]';

    A=[1.0000 0 0 0.0167 0 0 0 0 0 0 0 0 0 0 0 0.0316 0 0 -0.0000 -0.0000 0.0000 0.0000
        0 1.0000 0 0 0.0167 0 0 0 0 0 0 0 0 0 0 0 0.0316 0 -0.0000 0.0000 0.0000 -0.0000
        0 0 1.0000 0 0 0.0167 0 0 0 0 0 0 0 0 0 0 0 0.0158 -0.0000 0.0000 -0.0000 0.0000
        0 0 0 1.0000 0 0 0 0 0 0 0 0 0 0 0 3.7942 0 0 -0.0006 -0.0006 0.0006 0.0006
        0 0 0 0 1.0000 0 0 0 0 0 0 0 0 0 0 0 3.7942 0 -0.0006 0.0006 0.0006 -0.0006
        0 0 0 0 0 1.0000 0 0 0 0 0 0 0 0 0 0 0 1.8971 -0.0001 0.0001 -0.0001 0.0001
        0 0.0046 0 0 0.0000 0 1.0000 0 0 0.0333 0 0 0.0006 0 0 0 0.0000 0 -0.0000 0.0000
    0.0000 -0.0000
        -0.0046 0 0 -0.0000 0 0 0 1.0000 0 0 0.0333 0 0 0.0006 0 -0.0000 0 0 0.0000
    0.0000 -0.0000 -0.0000
        0 0 0 0 0 0 0 0 1.0000 0 0 0.0333 0 0 0.0006 0 0 0 0.0000 0.0000 0.0000 0.0000
        0 0.2778 0 0 0.0023 0 0 0 0 1.0000 0 0 0.0334 0 0 0 0.0029 0 -0.0000 0.0000
    0.0000 -0.0000
        -0.2778 0 0 -0.0023 0 0 0 0 0 0 1.0000 0 0 0.0334 0 -0.0029 0 0 0.0000 0.0000
    -0.0000 -0.0000
        0 0 0 0 0 0 0 0 0 0 0 1.0000 0 0 0.0334 0 0 0 0.0000 0.0000 0.0000 0.0000
        0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1.0033 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.6416 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.6419 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.6423 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.6427];

    B=[-0.0005   -0.0005    0.0005    0.0005
        -0.0005    0.0005    0.0005   -0.0005
        -0.0000    0.0000   -0.0000    0.0000
        -0.0807   -0.0807    0.0807    0.0807
        -0.0807    0.0807    0.0807   -0.0807
        -0.0064    0.0064   -0.0064    0.0064
        -0.0000    0.0000    0.0000   -0.0000
        0.0000    0.0000   -0.0000   -0.0000
        0.0000    0.0000    0.0000    0.0000
        -0.0000    0.0000    0.0000   -0.0000
        0.0000    0.0000   -0.0000   -0.0000
        0.0026    0.0026    0.0026    0.0026
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        189.7558         0         0         0
        0  189.8077         0         0
        0         0  189.8595         0
        0         0         0  189.9111];

    C=eye(19);


    %% System matrices
    Ap=zeros(22*p,22); Bu=zeros(22*p,4);Bdu=zeros(22*p,4*c);
    for (i=1:p)
        Ap(1+(i-1)*22 :22+(i-1)*22,1:22)=[A^i];
    end
    Bu(1:22,1:4)=B;
    for (i=1:p-1)
        Bu((1+i*22):(i*22)+22,1:4) = A*(Bu(1+((i-1)*22):((i-1)*22)+22,1:4))+B;
    end
    j=0;i=0;

    Bdu((1+j*22):(j*22)+22,1+i:4*(i+1))=B;
    for (j=1:p-1)
        Bdu((1+j*22):(j*22)+22,1:4)=(A^(j) *B) + Bdu((1+(j-1)*22):((j-1)*22)+22,1:4);
    end
    for (i=1:c-1)
        Bdu(:,1+(i*4):4*(i+1))=[zeros(22*i,4);Bdu(1:132-22*i,1:4)];
    end

    R=eye(4*c);
    Qi=[90 90 50 20 20 20 30 30 95 5 5 10 0 0 0 0 0 0 0 0 0 0]; %state weights
    Q=diag([Qi,Qi,Qi,Qi,Qi,Qi]);


    H=Bdu' * Q * Bdu + R;

end
%%

e=ref-est;
g=2*Bdu'*Q*e;

%% constraints
Umax=1; Umin=0.1; %Control signal limits
Qmax=0.1; Qmin=-0.1; %Roll/Pitch limits
Wmax=0.3; Wmin=-0.3; %Yaw speed limit
Pmax=10; Pmin=-10; %X-Y limits
Zmax=11; Zmin=9;
Smax=0.1; Smin=-0.1; %Speed limits
Lmin=[Qmin;Qmin;Wmin;Pmin;Pmin;Zmin;Smin;Smin;Smin];
Lmax=[Qmax;Qmax;Wmax;Pmax;Pmax;Zmax;Smax;Smax;Smax];
L=zeros(9,22);L(1,1)=1;L(2,2)=1;L(3,6)=1;L(4,7)=1;
```

```
        L(5,8)=1;L(6,9)=1;L(7,10)=1;L(8,11)=1;L(9,12)=1;

Lm=zeros(9*p,22*p);
for (i=1:p)
    Lm(1+(i-1)*9:9+(i-1)*9,:)=[zeros(9,(i-1)*22),L,zeros(9,(p-i)*22)];
end

F=[eye(4), eye(4), zeros(4,16)
    eye(4), eye(4), eye(4), zeros(4,12)
    eye(4), eye(4), eye(4),eye(4), zeros(4,8)
    eye(4), eye(4), eye(4),eye(4),eye(4), zeros(4,4)
    eye(4), eye(4), eye(4),eye(4),eye(4),eye(4)];
% CVX
n=c*4;
 cvx_begin
     variable dU(n);
     minimize(-dU' * g + dU'*H*dU);
     subject to
         0.1<=( F*   [U; dU]   )<=1; %Control signal limits
         [Lmin;Lmin;Lmin;Lmin;Lmin;Lmin]<Lm*(Ap*est + Bu*U +
    Bdu*dU)<[Lmax;Lmax;Lmax;Lmax;Lmax;Lmax];
 cvx_end
Y= F*[U; dU];

u = y(5:8);
```