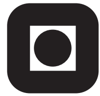


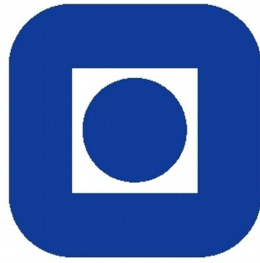
Model, Design and Control of a Quadcopter

Implemented on an Arduino Microcontroller, Using Wireless Communication Linked with a Computer Interface, Utilizing Addative Manufacturing Techniques to Enable Simple Replication

Joakim Brobakk Lehn

Master of Science in Cybernetics and Robotics
Submission date: May 2015
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics



Model, Design and Control of a Quadcopter

Implemented on an Arduino Microcontroller, Using Wireless Communication Linked with a Computer Interface, Utilizing Additive Manufacturing Techniques to Enable Simple Replication

by

Andreas Vikane Hystad
Joakim Brobakk Lehn

May 27, 2015

DEPARTMENT OF ENGINEERING CYBERNETICS
Norwegian University of science and technology

Problem description

The objective is to build, model, design and control a quadcopter sensor platform. The final product should be a good basis platform for future work with regards to advanced dynamical position systems or other applications suitable for a quadcopter platform. User control is to be achieved through the use of a computer application linked with an Arduino microcontroller and a replicable frame is to be developed through project management of an "Experts in Team" group. More specifically, the students have to:

- Obtain fast and accurate attitude and position estimates
- Implement and present a stable solution for the control of both the attitude and altitude
- Devise a path following control scheme and a velocity vector controller
- Develop an user interface for simple commands, that communicates wirelessly with the quadcopter
- Design a quadcopter 3D-model that enables the use of additive manufacturing

Preface

This report is written as a master thesis and builds upon a previous project conducted by the authors. We chose to work with a quadcopter because of our passion for control theory and complex systems, and frankly we find quadcopters extremely cool.

We want to give a special thanks to Sverre Hendseth for his guidance regarding this report and unceasing belief in our capabilities. We would also like to thank the department of engineering cybernetics for the opportunity to conduct this project, and also for providing the means for 3D printing our quadcopter prototype model. Finally we want to thank all the members of the experts in team group who helped design and develop our prototype model; Thomas Rostrup Andersen, Håkon Bråten, Alexander Vognild Burkow, Håkon Leithe, Nils Inge Rugsveen and kristian Stenrød.

A substantial amount of time spent on project management, construction of the quadcopter prototypes, coding the Arduino microcontroller and conducting experiments, are not reflected in this report.

Abstract

The popularity of the quadcopters is increasing as the sensors and control systems are becoming more advanced and less expensive. There are many commercial quadcopters available on the market today, but they are often hard to configure and comprehend. The time required to grasp the existing systems, could be spent designing better solutions. This project aims to use understandable system descriptions and sensor models as a basis to design configurable estimators and controllers, and to build a quadcopter well suited for educational purposes; as well as aiding to more advanced control in the future.

The system consists of several components for necessary sensor input, a radio transmitter, Windows user interface and an Arduino microcontroller. All filtering of signals, estimation of system states, calculation of control inputs and communication handling is done on the microcontroller, while the Windows application allows the user to command various actions. To achieve simple replicability, a 3D model of the frame was developed by an "Experts in Team" group. This provided us with useful experience in project management.

Satisfactory attitude estimates were obtained, a stable attitude controller was deduced and implemented, a user controlled Windows application was successfully developed and a quadcopter frame was created through additive manufacturing.

Abstrakt

Populariteten til kvadrotorer har økt ettersom sensorer og styringssystemer har blit mer avanserte og rimligere. Det er mange kommersielle kvadroterer tilgjengelig på markedet i dag, som ofte har begrensede muligheter for konfigurasjon og som er vanskelige å forstå. Tiden som er nødvendig for å sette seg inn i eksisterende systemer, er tid som kan bli brukt til å utforme bedre løsninger. Dette prosjektet tar sikte på å bygge på en forståelig systembeskrivelse og sensormodell, utforme konfigurerbare estimatorer og kontrollere, samt bygge en kvadrotor model som er godt egnet for undervisningsformål og som kan brukes som et ugangspunkt for mer avansert kontroll i framtiden.

Systemet består av flere komponenter for nødvendig sensor input, en radiosender, Windows brukergrensesnitt og en Arduino mikrokontroller. Filtrering av signaler, estimering av system tilstander, kalkulering av styresignaler og kommunikasjonshåndtering er utført på mikrokontrolleren. Windows applikasjonen gir brukeren muligheten til å sende ulike kommandoer. For å oppnå god replikerbarhet, ble en 3D-modell av rammen utviklet av en "Eksperter i Team" gruppe, noe som gav oss nyttig erfaring som prosjektledere.

Vi har oppnådd tilfredsstillende estimat av vinklene, en stabil kontroller for vinklene ble designet og implementert, en Windows applikasjon ble utviklet og en kvadrotor ramme ble 3D printet.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Fundamental Aspects Required When Designing an Autonomous Quadcopter . . . 2
1.3 Structure of The Report . . . 4

2 Summary of Fall Project and Components **5**
2.1 Introduction . . . 5
2.2 Arduino Due Board . . . 6
2.3 Motors . . . 7
2.4 Propellers . . . 8
2.5 Electric Speed Controller . . . 9
2.6 Inertial Measurement Unit . . . 9
2.7 Magnetometer . . . 10
2.8 Range Sensor . . . 11
2.9 Global Positioning System . . . 12
2.10 Radio . . . 13
2.11 Voltage Measurement . . . 13
2.12 Power Distribution Board . . . 14
2.13 Battery . . . 14
2.14 DC-DC Power Adapter . . . 15

3 Coordinate Systems and Quadcopter Theory **17**
3.1 Coordinate Systems . . . 17
3.1.1 BODY Frame . . . 17
3.1.2 North-East-Down, Earth-Centered-Earth-Fixed and Geodetic Coordinates . . . 17
3.1.3 The Rotation Matrix . . . 18
3.1.4 Rotation Between BODY and NED . . . 19
3.1.5 Transformation Between Geodetic and NED . . . 20
3.2 Quadcopter Dynamics . . . 20
3.2.1 Motor Dynamics . . . 21
3.2.2 Thrust Generated by the Propellers . . . 21
3.2.3 Torques Generated by the Propellers . . . 23
3.2.4 Equations of Motion . . . 24
3.2.5 Kinematic Model . . . 25
3.2.6 Kinetic Model . . . 25

4 Filter Designs **27**
4.1 Theory . . . 27
4.1.1 Low-pass Filter . . . 27
4.1.2 High-pass Filter . . . 27
4.1.3 Complementary Filter . . . 27
4.1.4 Continuous-discrete Extended Kalman Filter . . . 28
4.2 Problem Description . . . 28
4.3 Discrete Implementation . . . 29
4.3.1 Low-pass Filter . . . 29
4.3.2 High-pass Filter . . . 30
4.3.3 Complementary Filter . . . 30
4.3.4 Continuous-discrete Extended Kalman Filter . . . 31
4.4 Discussion . . . 31
4.5 Recommendations and Future Work . . . 32

5 Estimation of Roll and Pitch **33**
5.1 Theory . . . 33
5.1.1 Gravity Vector . . . 33
5.1.2 Accelerometer Measurement Model . . . 33
5.1.3 Gyroscope Measurement Model . . . 33
5.2 Problem Description . . . 34
5.3 Design of Solution . . . 34
5.3.1 Inertial Measurement Unit Sensitivity Range and Scaling . 34
5.3.2 Accelerometer Bias Compensation . . . 35
5.3.3 Gyroscope Bias Compensation . . . 35
5.3.4 Accelerometer Estimate . . . 35
5.3.5 Gyroscope Estimate . . . 36
5.3.6 Complementary Filter Estimate . . . 36
5.3.7 Kalman Filter Implementation . . . 36
5.4 Observations and Results . . . 37
5.5 Discussion and Conclusion . . . 38
5.6 Recommendations and Future Work . . . 39

6 Estimation of Yaw **41**
6.1 Theory . . . 41
6.1.1 Magnetometer measurement model . . . 41
6.1.2 Gyroscope Measurement Model . . . 41
6.1.3 Earth's Magnetic Field . . . 41
6.1.4 Hard and Soft Iron Distortions . . . 43
6.2 Problem Description . . . 44
6.3 Design of Solution . . . 45

6.3.1 Hard and Soft Iron Distortions Compensation 45
6.3.2 Magnetometer Estimate 46
6.3.3 Gyroscope Estimate 47
6.3.4 Combined Magnetometer and Gyroscope 47
6.4 Observations and Results 48
6.4.1 Hard and Soft Iron Calibration 48
6.4.2 Yaw Estimates 49
6.5 Discussion and Conclusion 51
6.6 Recommendations and Future Work 51

7 Estimation of Position 53
7.1 Theory 53
7.1.1 Position Measurement from Global Positioning System Technology 53
7.1.2 Distance Measurement Using Sound 54
7.2 Problem Description 54
7.3 Design of Solution 54
7.3.1 Ultrasonic Sensor Height Estimate 54
7.3.2 GPS Position Estimate 55
7.3.3 Kalman Position Estimate 56
7.4 Observations and Results 57
7.5 Discussion and Conclusion 60
7.6 Recommendations and Future Work 60

8 Control of Attitude 63
8.1 Theory 63
8.1.1 The Linear Quadratic Regulator 63
8.1.2 The Proportional-Integral-Derivative Controller 64
8.1.3 Nonlinear Control Theory 65
8.1.4 Alternative Controllers 66
8.2 Problem Description 67
8.3 Design of Solution 67
8.3.1 The Linear Quadratic Regulator 68
8.3.2 The Proportional-Integral-Derivative Controller 71
8.3.3 Nonlinear Controller 72
8.4 Observations and Results 76
8.4.1 The Linear Quadratic Regulator 76
8.4.2 The Proportional-Integral-Derivative Controller 77
8.5 Discussion and Conclusion 77
8.5.1 The Linear Quadratic Regulator 78
8.5.2 The Proportional-Integral-Derivative Controller 78

8.6 Recommendation and Future Work 79

9 Control of Altitude **81**
9.1 Theory 81
9.2 Problem Description 81
9.3 Design of Solution 81
9.4 Observation and Results 84
9.5 Discussion and Conclusion 85
9.6 Recommendation and Future Work 85

10 Control of Motion **87**
10.1 Theory 87
10.1.1 Dynamic Positioning 87
10.1.2 Waypoints, Paths And Trajectories 87
10.2 Problem Description 87
10.3 Design of Solution 88
10.3.1 Guidance Systems, Trajectory and Path Generation 88
10.3.2 Control of Velocity in North-East-Down Coordinate System 93
10.3.3 Structure of the Control System 95
10.4 Discussion 96
10.5 Recommendation and Future Work 97

11 Prototype Development **99**
11.1 Background 99
11.2 Problem Description 100
11.3 Development of Prototype Version 1.1 101
11.4 Development of Prototype Version 2 102
11.4.1 Design Strategy 102
11.4.2 Robustness Calculations 103
11.4.3 Model and Description of Parts 105
11.5 Observation and Results 111
11.6 Discussion and Conclusion 112
11.7 Recommendation and Future Work 112

12 Windows Application for Quadcopter Control **113**
12.1 Choice of Application Platform 113
12.2 Choice of Programming Language and Framework 114
12.3 Programming Graphical User Interface in C# Using Unity3D . . . 115
12.4 Communication Design and Implementation in C# 117
12.4.1 How to Change COM Port on the Radio in Windows 117
12.4.2 How to Enable Serial Communication in C# 120
12.4.3 Design of Communication Protocol 121

12.5 Functionality Description of the Windows Application 123
12.5.1 Connection Tab 123
12.5.2 Update Tab 124
12.5.3 Manual Control Tab 125

13 Error Handling 129
13.1 Problem Description 129
13.2 Error Detecting and Handling Strategies 129
13.2.1 Loss of Communication 129
13.2.2 Low Battery 129
13.2.3 Loss of Sensors 130
13.2.4 Roll or Pitch Angles Close to Singularity 132
13.3 Recommendation and Future Work 132
13.3.1 Loss Of Communication 132
13.3.2 Loss of Sensors 133
13.3.3 Roll or Pitch Angles Close to Singularity 133

14 Discussion and Conclusion 135

15 Future Work 137

Bibliography 139

A Building Quadcopter Prototype 2 143
A.1 Part List 143
A.2 Preparation of Pre-ordered Parts 144
A.2.1 Motors 144
A.2.2 Electrical Speed Controller 144
A.2.3 Range Sensor 145
A.2.4 Inertial Measurement Unit And Magnetometer 146
A.3 Wiring diagram 148
A.4 Cable List 149
A.5 Mounting Procedure 150
A.5.1 Motors and Arms 150
A.5.2 Electrical Speed Controller and Frame 150
A.5.3 Technical Box 151
A.5.4 Battery Box 151

B Description and Walkthrough of Various Code 153
B.1 Arduino Code 153
B.1.1 Inertial Measurment Unit 153
B.1.2 Infra Red Range Sensor 153

B.1.3 Magnetometer 153
B.1.4 Arduino Main 153
B.1.5 Motor tester 154
B.1.6 Radio 154
B.1.7 Ultrasonic Range Sensor 154
B.1.8 Voltage Reader 154
B.2 Matlab Code 154
B.2.1 Matlab Filtering 154
B.2.2 Read Data over the Radio 154
B.2.3 Read Data over Serial 154
B.2.4 Simulation 155
B.3 Windows Application - Unity 155

List of Figures

2.1 Arduino Due board . . . 7
2.2 Motor used in our quadcopter . . . 8
2.3 Propellers used in our quadcopter . . . 8
2.4 ESC used in our quadcopter . . . 9
2.5 IMU used in our quadcopter . . . 10
2.6 Magnetometer used in our quadcopter for yaw estimation . . . 11
2.7 Range sensor used in our quadcopter . . . 12
2.8 GPS used in our quadcopter . . . 12
2.9 Radio used for communication . . . 13
2.10 Scheme for voltage measurement . . . 14
2.11 Power distribution board . . . 14
2.12 Battery used in our quadcopter . . . 15
3.1 The quadcopter with its BODY coordinate system . . . 17
3.2 Geodetic longitude and latitude definition and relation to NED . . . 18
5.1 A comparison between the different estimates . . . 38
6.1 Total magnetic intensity . . . 42
6.2 Magnetic measurement with no distortions . . . 43
6.3 Magnetic measurement with hard iron distortions . . . 44
6.4 Magnetic measurement with hard and soft iron distortions . . . 44
6.5 Hard and soft iron calibration. Top row: before calibration. Bottom row: after calibration . . . 48
6.6 Yaw estimates . . . 50
7.1 An illustration of how the time difference is measured . . . 54
7.2 Ultrasonic Sensor Concept . . . 55
7.3 A comparison of the distance measurement vs the tilt compensated distance estimate . . . 58
7.4 GPS North and East estimates vs time . . . 59
7.5 GPS estimated North-East position vs actual position . . . 60
8.1 Angle and motor input using LQR . . . 76
8.2 Angle and motor input using PID . . . 77
9.1 Control system design for the altitude . . . 82
9.2 PID control system design for the altitude . . . 83
9.3 Altitude controller . . . 84
10.1 LOS guidance illustration in 2D . . . 89
10.2 PP guidance illustration in 2D . . . 90
10.3 CB guidance illustration in 2D . . . 91
10.4 Straight Lines and Circles path . . . 92
10.5 Control system . . . 95
10.6 Drag illustration . . . 96

11.1 Prototype version 1 house 100
11.2 Prototype version 1 100
11.3 Prototype version 1.1 102
11.4 Top frame 106
11.5 Bottom Frame 107
11.6 Technical box top view 108
11.7 Technical box seen from below 108
11.8 Battery Box 109
11.9 Model of the Arm 110
11.10Model of the Leg 110
11.11Prototype version 2 111
12.1 How to access Windows properties 118
12.2 How to access Device Manager 118
12.3 How to access COM port properties 119
12.4 How to access Advanced port settings 119
12.5 How to change COM port in hardware 120
12.6 Connection tab in Windows application 124
12.7 Configuration Update tab in Windows application 125
12.8 Manual Control tab in Windows application 127
A.1 Prepared motors with termination 144
A.2 A prepared ESC should look like this 145
A.3 Prepared range sensor 146
A.4 IMU and magnetometer with pins 147
A.5 Wiring diagram 148
A.6 The ESC's are palced in the space between the top frame and bottom frame 150
A.7 The resulting quadcopter model 151

List of Tables

2.1 Comparison between the arduino Leonardo and arduino Due board 6
5.1 Sensitivity ranges and scales, accelerometer 34
5.2 Sensitivity ranges and scales, gyroscope 34
8.1 Zieger-Nichols method 64
8.2 PID control gains 79
11.1 Results from load calculations 104
11.2 Capacity for ABS-plus 105
11.3 Approximate time needed for printing the quadcopter 111
A.1 Part list 143
A.2 3D print 143
A.3 Screws and nuts 144
A.4 Wiring schematic 150

List Of Abbreviations

DMP	-	Digital Motion Processor
EKF	-	Extended Kalman filter
ESC	-	Electric Speed Controller
GPS	-	Global Positioning System
IMU	-	Inertial Measurment Unit
IR	-	Infrared
LQR	-	Linear Quadratic Regulator
NED	-	North East Down
RF	-	Radio frequency
PID	-	Proportional-Integral-Derivative (Controller)
RPM	-	Revolutions per Minute
PWM	-	Pulse-Width Modulation
UAV	-	Unmanned Aerial Vehicle
VTOL	-	Vertical Takeoff and Landing

List Of Symbols

$_b$	-	Refered to BODY frame
$_n$	-	Refered to NED frame
ϕ	-	Euler angle roll
θ	-	Euler angle pitch
ψ	-	Euler angle yaw
p	-	Euler angle roll rate
q	-	Euler angle pitch rate
r	-	Euler angle yaw rate

1 Introduction

The Israeli and US military were among the first to recognize the advantage of unmanned aerial vehicles. The research and investment in these machines have been bolstered by the advance of miniaturization, maturing of technologies, more powerful processors and more reliable and cheaper sensors. The miniaturization favored the creation of mini UAV or micro UAV (MAV); weighing less then a kilogram. This have motivated creation of innovative vehicles in the private sector and in universities, some universities in particular turned their attention to the potential of Vertical Takeoff and Landing (VTOL) vehicles. Increasing interest in drones for both commercial and military purposes in modern times, have lead to the development of many commercial "complete packaged" quadcopter solutions being available on the market today.

The quadcopter is a popular drone, mainly because of its unique properties. The major advantages of the quadcopter, is its ability to hover, or stand still in the air, and its VTOL capabilities. This allows the quadcopter to be operated in nearly any environment, such as indoorpleve flying or tight spaces with limited maneuverability.

A conventional helicopter with one main rotor and one tail rotor posses many of the same properties as a quadcopter. However, the quadcopter have no moving parts except for the rotating motors and propellers, while the conventional helicopter require a complex hub to make it possible to rotate the motor axis to induce a translating movement. The quadcopter is also less prone to vibrations and it is more flexible when it comes to the placement of the center of gravity. Due to smaller size of rotors, they can be more easily covered, making it safer to fly indoors.

The typical quadcopter design has, as stated earlier, no moving parts except for the propellers. The motors and their propellers are mounted to the frame and the only way to induce a lateral motion is to tilt the entire frame.

Unlike a conventional helicopter, the quadcopter does not have a tail rotor to control the yaw motion. The quadcopter has four motors where two spins clockwise and two spins counterclockwise. If the pair of clockwise motors are spinning at a different rate than the pair of counterclockwise motors, it will create a moment about the yaw axis.

1.1 Motivation

Quadcopter control is a fundamentally difficult and interesting problem. With six degrees of freedom and only four independent inputs, the quadcopter is underactuated and the resulting dynamics are highly nonlinear. Unlike ground vehicles, aerial vehicles have very little friction to prevent their motion, therefore they must

provide their own damping in order to stop moving and remain stable. These factors make the stabilization of the quadcopter a fascinating control problem.

The span and complexity makes this project an extensive learning platform. Our motivation lies in the design of our own solutions to the many challenges faced during the development of an autonomous quadcopter platform. Time required to grasp the existing commercial quadcopter systems, could be better spent designing our own solution. Another key motivational factor is the learning outcome gained during the execution of this project. Throughout this paper we will therefore evaluate various designs, implement and compare the results in order to conclude which of the solutions that has the best performance. Furthermore, we will gain project management experience by leading an "Experts in Team" group consisting of six students. These students will assist in the development of a prototype that will serve as a test platform, which can be used by future student who wish to work with quadcopters. Our greatest motivation for this project is to learn the many various aspects regarding a project of this size, and to improve our advanced problem solving capabilities.

1.2 Fundamental Aspects Required When Designing an Autonomous Quadcopter

The designing of an autonomous quadcopter is a complicated and comprehensive task. To tackle larger tasks, a "divide and conquer" strategy is often applied. By dividing the project into smaller tasks to be solved independently, the overall complexity is reduced. The project has thus been divided into smaller projects. Following, is a list describing the different "mini" projects and why they are needed as a part of completing the overall design

- *Filter design:* All of our sensors provide raw data, which contains unwanted noise. To reduce the noise levels, there are several different filters which can be applied to the sensor readings. In this section, we will look at a few different options and how to implement them on the Arduino microcontroller.

- *Estimation of Roll and Pitch:* In order to control the quadcopter roll and pitch angles, the angles must be known. None of the sensors measure the angles directly, therefore an estimate has to be obtained from the accelerometer and/or gyroscope sensors. In this section, we propose several estimation schemes for the roll and pitch angles, and compare the resulting estimates. An estimation scheme is chosen based on the results.

- *Estimation of Yaw:* In order to control the quadcopter yaw angle, the yaw angle must be known. Since none of the sensors measure the yaw angle directly, an estimate has to be obtained from the magnetometer and/or

gyroscope sensor. These sensor readings are dependent on the roll and pitch angles, and thus relies on the estimates found in the previous section.

- *Estimation of position:* The height above the ground is the most crucial information needed in order to control the quadcopter. This will allow the quadcopter to be manually controlled, using the desired height and tilt angles as input. If the North and East coordinates are known as well, a complete autonomous control scheme can be implemented. An estimation strategy for height is proposed, using an ultrasonic sensor, and an estimation scheme for position is proposed, using a GPS and an IMU. Solutions are chosen based on the result.

- *Control of attitude:* If the estimates schemes of the attitude obtained from previous sections are accurate, the attitude can be controlled by varying the inputs to the motors. A difference in in the propeller speed on the motors, will create a moment and turn the quadcopter frame. In this section, different control schemes for controlling the attitude is presented, and a solution is chosen based on the results.

- *Control of altitude:* By controlling the attitude, the quadcopter should not drift. Controlling the height above the ground as well, will enable the quadcopter to hover in the air. A control scheme for the altitude will presented, and a solution will be chosen based on the result.

- *Control of motion:* Using a stable attitude and altitude controller as a basis, more advanced control schemes can be devised. Depending on the objective, several guidance systems may apply, and several path generation methods exists. Some guidance systems will be explored along with trajectory generation methods and path generation methods. To control the speed of the quadcopter along the desired velocity given by the guidance systems, a speed controller will be devised.

- *Development of a prototype:* In this section, we will go through the development of a working prototype, printed using additive manufacturing techniques.

- *User control interface:* To control the quadcopter, a user control interface needs to be designed. This control interface can be developed on numerous different platforms using several different strategies. Different options will be presented and an application will be created.

- *Error handling:* Several unexpected errors can occur at any time during a flight. A number of possible errors and solutions will be explored.

1.3 Structure of The Report

This project is a considerable task, and needs to be broken down into smaller projects that can be solved independently. We have structured our report by starting with providing the necessary background information, containing the current status when starting the project as well as summarizing the most fundamental results from our project last fall.

After the background chapter, ten "mini" projects will follow, as described in Section 1.2. This is to make the report more straightforward to follow for the reader, and to make it easier to look up parts of interest. We round off the report with a general discussion regarding the entire project, before providing our final conclusions and recommendations for future work.

2 Summary of Fall Project and Components

In this chapter we provide some of the fundamental prospects that were explored during our fall project[31]. We also explore all the mechanical components and sensors used in our quadcopter. The information in this chapter is useful in order to read the rest of this report.

2.1 Introduction

During the last fall, the authors of this master thesis had a project regarding model, design and control of a quadcopter[31]. This report continue with the work the said report started upon.

We divided the task into four main objectives; design, modelling, control and estimation. The sensors needed to observe the system states were identified and implemented using an Arduino Leonardo microcontroller board. The kinematic and kinetic quadcopter system dynamics were derived, and state space equations for position, attitude, velocities and angular rates were presented. A control law were explored for a subspace of the system state with the aim of controlling the attitude, by minimizing a quadratic cost function. Furthermore a height controller were deduced, and combined with the attitude controller. Tuning were performed by simulating the system dynamics without successfully stabilizing the attitude or altitude in practice.

In order to derive the system model we required knowledge of coordinate systems, Euler angles, rotation matrices ad transformation matrices. The dynamic model were derived using classical kinematic laws, and by summarizing forces and moments for the quadcopter. The model were necessary in order to simulate the controllers for the aircraft, as well as to explore the different state estimators.

By accurately estimating the attitude, it is possible to ignore most of the system and measurement noise, which leads to smoother control. To accomplish this task, a Continuous-Discrete Extended Kalman filter were presented, and applied to some sensor measurements. By merging different sensor measurements, we could achieve better estimates, redundancy and drift compensation.

We built a prototype and managed to ascend from the ground by applying only approximately 35% of the maximal PWM. The remaining motor force available will be useful for rapid stabilizing and satisfactory control of the quadcopter. The motors in combination with the Electronic speed controller had a very short time response compared to other systems.

The quadcopter model, controller, and filters were simulated to allow for controller assessment and tuning on the computer. In the simulator we managed to achieve our objective of controlling the quadcopter attitude and altitude. We were unable to test the complete controller on our prototype, as the prototype

crashed in an early test procedure, and we were unable to complete the construction of a new one within our time limit. In addition, a few of the fundamental aspects required in order to control a quadcopter were implemented successfully on the microcontroller. Only some sensor specific filters meet the requirements we currently have for our quadcopter.

The various sensors and components used in our quadcopter platform were explored and evaluated, and will be briefly summarized here. Some changes have been made regarding the Arduino microcontroller and range sensor in addition to including a DC-DC adapter.

2.2 Arduino Due Board

During our project this fall[31] we estimated that an arduino leonardo would be sufficient for our purposes. We stated that we require a board with sufficient computer power and memory to be able to compute extended Kalman filter as well as the optimal control input, while at the same time process input from the various sensors. During our development we discovered that the arduino leonardo lacked the necessary SRAM as well as flash memory, and we also started to run out of pins. For this reason we needed to upgrade to a more powerful board, in order to ease the development, to allow future upgrades to the software, and to be able to use the current code without any significant change.

	Arduino Leonardo	Arduino Due
Operating voltage	5V	3.3V
Input voltage (recommended)	7-12V	7-12V
Input voltage (limits)	6-20V	6-16V
Digital I/O pins	20	54
PWM channels	7	12
Analog input channels	12	12
Flash memory	32 kb	512 kb
SRAM	2.5 kb	96 kb
Clock speed	16 MHz	84 MHz

Table 2.1: Comparison between the arduino Leonardo and arduino Due board

In table 2.1 we have compared the arduino Leonardo[2] and the arduino Due[1] board specifications, and we can see that the Arduino Due have far superior specifications considering the number of Digital I/O pins, flash memory, SRAM and clock speed. The only limitation regarding the Arduino Due comapared to the Arduino Leonardo is that it has a smaller limit for upper voltage, but our battery limitations are at 12.6 voltage, and will therefore not be a problem.

Based on our requirements the Arduino Due is a good and suitable choice, it passes all requirements with clear margins, it is not expensive and it is deemed as a reliable board. The programming language used in Arduino is c, with a huge number of official libraries that can be used in the code.

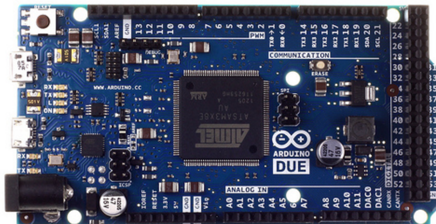


Figure 2.1: Arduino Due board

2.3 Motors

We require high quality reliable motors with rapid response in order to control the quadcopter. If one or several of the motors at some point during a flight experience any problems it would be devastating for the quadcopter, and can at worst endanger the quadcopter itself, property and people. Furthermore it is important that the motors are powerful enough to be able to lift the quadcopter and perform various aerial movements. We also require the motors to have a fast response in order to ensure a more stable flight. Finally we require that the motors are close to vibration free, as any vibration will cause noise in our IMU measurements.

Based on these criterias we decided to acquire the SunnySky Angel A2212 KV800 Brushless Motor G638. It is a brushless motor designed for remote controlled airplanes as well as quadcopters, and are considered to be highly reliable. Sunnysky have long experience with motors for RC airplanes and quadcopters, and their motors are known for being vibration free. According to the specifications[8], each motor can give a thrust of 820 grams at 136 watt, based on our ESC (section 2.5) and propellers (section 2.4), which means that our quadcopter could theoretically fly at approximately 35 percent capacity. This is more than enough to fulfill our requirements, ant it follows that we could perform quick movements if necessary, which will make the control sequence more simple.

Figure 2.2: Motor used in our quadcopter

2.4 Propellers

The requirements for the propellers are less strict than those for the motors. We require light propellers with size and lift potential such that the quadcopter can hover at less then 50 % of the motor capacity. It is also preferable if the propeller can survive soft bumps. For our quadcopter we choose plastic 10X4.5 propellers (254mmx114mm) with their light weight. This is a standard propeller used by many quadcopters. The total length of the propeller is 254mm while the pitch is 114mm.

Figure 2.3: Propellers used in our quadcopter

2.5 Electric Speed Controller

An electric speed controller (ESC) is an electric circuit with the purpose to vary an electric motor's speed[20]. We require that the ESC is fast and reliable for the same reasons stated for the motors in section 2.3. We choose the SS series 18-20A ESC developed by Hobbyking which comes with a limited range of programming functions and are designed to be plug-n-play. This ESC is developed for airplanes as well as multicopters and are designed to be stable as well as reliable and fast.

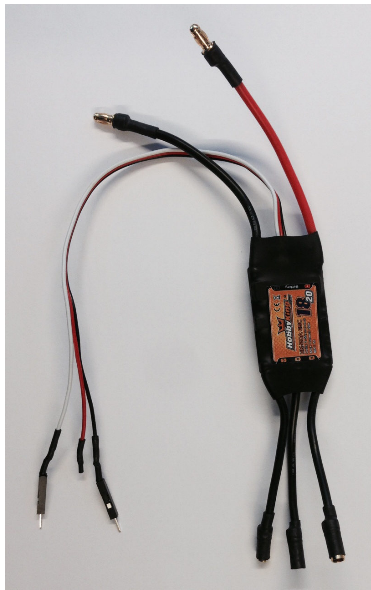


Figure 2.4: ESC used in our quadcopter

2.6 Inertial Measurement Unit

Precision and accuracy is important when it comes to Accelerometer and gyroscope measurement. We require a 3-axis accelerometer and gyroscope that provides reliable and accurate data. It is also an advantage if they can be on the same chip. For this reason we went with the MPU-600, which is a small, thin, ultralow power, 3-axis accelerometer and gyroscope. The device is very accurate, as it contains 16-bis analog to digital conversion hardware for each channel[9]. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. The sensor has a "Digital motion processor" which can be programmed with firmware and is able to do

complex calculations with the sensor values.

Figure 2.5: IMU used in our quadcopter

2.7 Magnetometer

As described in our falls project[31], we know that we can have accurate measurements of both roll and pitch using accelerometer and gyroscope. We also get an estimate for yaw based on our velocity given by the GPS. This is however less accurate for our quadcopter, which will mostly hover at zero speed. The GPS estimates for yaw will be inaccurate when hovering. We can use the data acquired from the IMU to detect any drift in yaw angle, but this estimation is useless if we want to turn the quadcopter towards a specific yaw angle.

For this reason we needed to include a compass. We require the compass to be accurate and reliable, but we accept that it can be magnetic based since we operate far from the magnetic poles. We choose the *MHC5883L* Triple axis compass magnetometer designed for Arduino. This compass will provide suitable measurements for yaw, and can easily be implemented on our Arduino board.

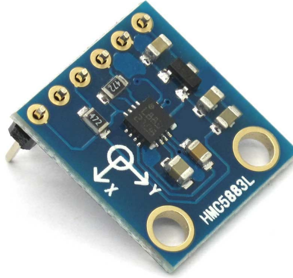


Figure 2.6: Magnetometer used in our quadcopter for yaw estimation

2.8 Range Sensor

In order to get more accurate measurement of our hovering height we decided to also include a range sensor to be used in combination with the GPS measurements. This provides us redundancy as well are more accurate measurements in the sensors working area. Accurate measurements are crucial when the quadcopter hover at low heights as well as landing. We require a range sensor that gives accurate measurements for heights between 20-100 cm. For that reason we choose the *HC-SR04*, which you can see in figure 2.7. This is a simple ultrasonic sensor developed for arduino that can be used to measure distance by sending out a ultrasonic sound wave and detecting the return of said sound wave. According to the sensor data sheet[14] the working area is 2-400 cm, which satisfies our requirements for hovering and landing. During our fall project[31] we used an IR sensor with a narrower working area, and in addition the IR sensor output is quite nonlinear and is difficult to translate directly to distance. Both of these aspects have been improved with this change of range sensor.

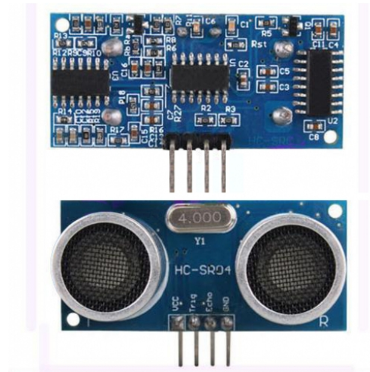


Figure 2.7: Range sensor used in our quadcopter

2.9 Global Positioning System

In order to get some feedback on our estimated position we need a GPS. We require a reliable medium accuracy GPS, it is also preferable if the GPS antenna is small and light. We need a GPS that can provide position data with medium accuracy, and to be able to re-acquire satellite lock fast in the cases where it looses the signal. We choose the 3DR uBlox GPS. This is a high performance GPS designed for multicopters and rovers in particular where GPS accuracy is paramount. It uses a 5 Hz update rate, which is more than fast enough for the low speeds our quadcopter operates in.

Figure 2.8: GPS used in our quadcopter

2.10 Radio

Radio communication is essential for controlling the quadcopter, as well as for tuning when testing the controllers and providing data during flight. It can also serve as a great tool when extending the usage for practical applications. The radio link needs to run on frequencies dedicated for private use in Norway, and is required to have 100 meter range, or more in open terrain. The 3DR Radio set runs on 433 Mhz, which is a standard frequency for private use in Norway. The range in open terrain is more then sufficient and is well suited for our use. The set comes with two antennas, one for the computer on the ground and one for the quadcopter.

Figure 2.9: Radio used for communication

2.11 Voltage Measurement

By measuring the voltage of the battery, we know how much power we have left. This allows us to land the quadcopter safely when the battery power is low, and not damage the LiPo battery by discharging it completely. According to the data sheet for LiPo batteries, a cell is at full power when measuring 4.2 Volt and at 20% power when measuring 3.7 Volt. Since we are using 3 cell batteries, this means that 100% power equals 12.6 Volt and 20% power equals 11.1 Volt across the three cells. To measure the voltage, we simply use an analog input pin on the Arduino board as illustrated in Figure 2.10, where R is chosen as $3M\Omega$ to reduce the power over the resistor.

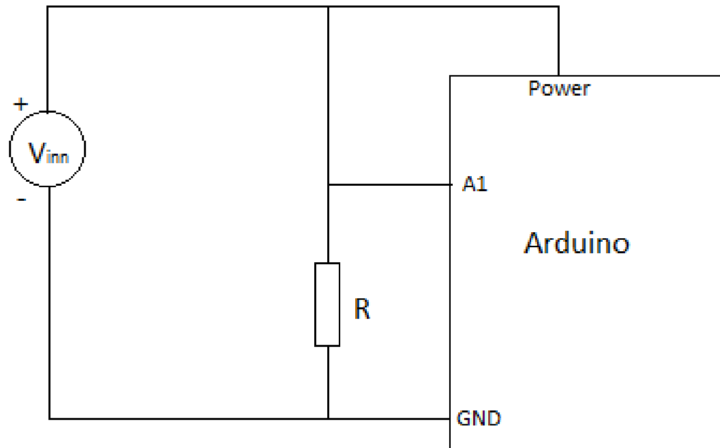


Figure 2.10: Scheme for voltage measurement

2.12 Power Distribution Board

In order to reduce the number of connections directly to the battery we acquired a Power distribution board developed by HobbyKing. This board provides a good solution for power distribution, and the PCB is gold plated for optimal efficiency. This is an easy ”plug and play” solution where we simply connect all four ESC to the board, and connect the board directly to the battery.

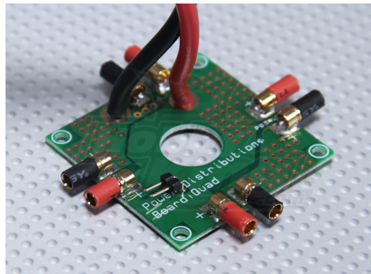


Figure 2.11: Power distribution board

2.13 Battery

The quadcoter motors and sensors are all powered by using a battery pack. We require a battery that stays within the input voltage limits of the microcontroller,

and that the battery provides enough power to be able to sustain a flight for at least 10 minutes. We bought the Turnigy 5000mAh 3S 20C Lipo Pack delivered by HobbyKing. This is a 5000mAh battery which should allow us to have a normal flight for an estimate of 15 minutes, although the battery voltage needs to be checked in software. The battery is quire heavy; 412g [3], and is the tradeoff when choosing such a powerful battery.

Figure 2.12: Battery used in our quadcopter

2.14 DC-DC Power Adapter

During our flight sequences we discovered that the power delivered by the battery contains a significant amount of noise caused by the motors. This caused some of the sensors to malfunction, and deliver faulty data. In order to handle this problem we filter the power through a DC-DC adapter between the battery and the Arduino Due. We discovered this problem at a quite late stage, and currently the only available option were to use an Arduino Leonardo as our DC-DC adapter. Here you simply insert the battery on the input pin on the Arduino Leonardo, and extract the power from the V_{out}, which you in turn insert into the input pin for the Arduino Due.

3 Coordinate Systems and Quadcopter Theory

3.1 Coordinate Systems

3.1.1 BODY Frame

The quadcopter is navigating in a three dimensional space. The BODY coordinate system (denoted 'b') is a moving coordinate frame fixed to the quadcopter, with origin o_b in the middle of the aircraft as seen in figure 3.1. The x-axis is defined to be pointing towards one of the motors, the z-axis pointing out the bottom of the quadcopter and the y-axis complete the right handed orthogonal coordinate system. An illustration of this is shown in Figure 3.1.

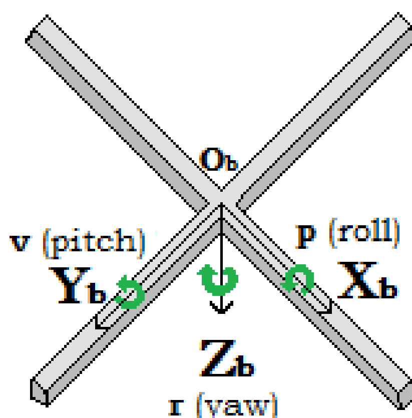


Figure 3.1: The quadcopter with its BODY coordinate system

3.1.2 North-East-Down, Earth-Centered-Earth-Fixed and Geodetic Coordinates

The NED coordinate system is the coordinate system we refer to in our daily life. The x-axis points towards true North, the y-axis towards East, while the z-axis points downwards normal to the Earth's surface. The origin of the NED coordinate system is usually defined as the starting point on the ground where the quadcopter takes off. The position of the quadcopter is defined as the position of the BODY origin with respect to the NED origin.

Presentation of terrestrial position data is often given in terms of the ellipsoidal parameters longitude(l), latitude(μ) and height. This is referred to as Geodetic coordinates.

The ECEF coordinate system is defined to have its origin in the center of the Earth, the x-axis pointing towards the intersection of 0° longitude (Greenwich meridian) and 0° latitude (Equator), the z-axis pointing along Earth's rotational axis (North), and the y-axis complete the right handed orthogonal coordinate system.

Figure 3.2 shows how NED, ECEF and Geodetic coordinates relates to each other.

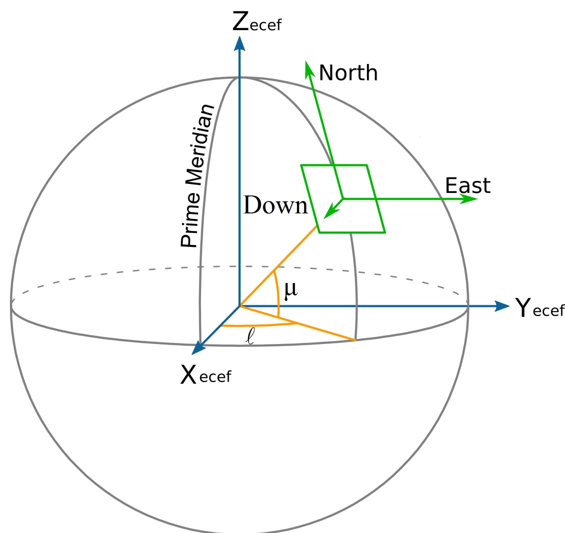


Figure 3.2: Geodetic longitude and latitude definition and relation to NED

3.1.3 The Rotation Matrix

In order to transform vectors between different coordinate systems, we use rotation matrices. A rotation matrix $\mathbb{R}$ which transforms a vector representation in frame b to frame a is denoted $\mathbf{R}_b^a$. Generally a rotation matrix $\mathbf{R}$ is any matrix satisfying

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}, \qquad \det(\mathbf{R}) = 1 \tag{3.1}$$

This implies that $\mathbf{R}$ is orthogonal, and as a consequence the inverse rotation matrix is given by $\mathbf{R}^{-1} = \mathbf{R}^T$. Hence

$$\mathbf{R}_b^a = (\mathbf{R}_a^b)^{-1} = (\mathbf{R}_a^b)^T \tag{3.2}$$

3.1.4 Rotation Between BODY and NED

To transform between the BODY and NED coordinate systems, we need to find the rotation matrix $\mathbf{R}_b^n$.

Euler angles are not commutative, and one must choose a sequence of rotation. In aerospace engineering the convention most widely used is *yaw, pitch, roll* (zyx-convention), noted $\psi,\ \theta,\ \phi$ respectively. This implies that the first rotation corresponds to a yaw rotation, the second a pitch rotation and the last a roll rotation. Where each independent rotation can be described by the following rotation matrices[29]

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.4}$$

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.5}$$

From this we can derive the rotation matrix from BODY to NED as

$$\mathbf{R}_b^n(\mathbf{\Theta_{nb}}) = \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi} \tag{3.6}$$

$$\mathbf{R}_b^n(\mathbf{\Theta_{nb}}) = \begin{bmatrix} c(\psi)c(\theta) & -s(\psi)c(\phi) + c(\psi)s(\theta)s(\phi) & s(\psi)s(\phi) + c(\psi)c(\phi)s(\theta) \\ s(\psi)c(\theta) & c(\psi)c(\phi) + s(\phi)s(\theta)s(\psi) & -c(\psi)s(\phi) + s(\theta)s(\psi)c(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{3.7}$$

where $c(*) = \cos(*)$, $s(*) = \sin(*)$ and $t(*) = \tan(*)$. The calculated matrix gives the convention from BODY to NED. In some cases it can be useful to transform from NED to BODY. From Equation 3.2 we have $\mathbf{R}_n^b = (\mathbf{R}_b^n)^T$, which gives

$$\mathbf{R}_n^b(\mathbf{\Theta_{nb}}) = \begin{bmatrix} c(\psi)c(\theta) & s(\psi)c(\theta) & -s(\theta) \\ -s(\psi)c(\phi) + c(\psi)s(\theta)s(\phi) & c(\psi)c(\theta) + s(\phi)s(\theta)s(\psi) & c(\theta)s(\phi) \\ s(\psi)s(\phi) + c(\psi)c(\phi)s(\theta) & -c(\psi)s(\phi) + s(\theta)s(\psi)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{3.8}$$

3.1.5 Transformation Between Geodetic and NED

In Figure 3.2, we can see how the geodetic coordinates relates to ECEF and NED coordinates. To transform a geodetic coordinate into NED coordinates, we must first transform from geodetic to Eearth Centered Earth Fixed(ECEF) coordinates. Given a geodetic position given as $\hat{\mathbf{p}}_g = [l,\ \mu,\ h]^T$, the ECEF coordinates are given by

$$\hat{\mathbf{p}}^e = \begin{bmatrix} (N+h)\cos(\mu)\cos(l) \\ (N+h)\cos(\mu)\sin(l) \\ (\frac{r_p^2}{r_e^2}N+h)\sin(\mu) \end{bmatrix} \tag{3.9}$$

where N is the radius of curvature in the prime vertical, obtained by

$$N = \frac{r_e^2}{\sqrt{r_e^2\cos^2(\mu) + r_p^2\sin^2(\mu)}} \tag{3.10}$$

and r_e and r_p are the equatorial and polar Earth radii of the ellipsoid, given as

$$r_e = 6378137m \tag{3.11}$$

$$r_p = 6356752m \tag{3.12}$$

The measurement can then be transformed into NED coordinates through the rotation matrix

$$\mathbf{R}_e^n(l,\mu) = \mathbf{R}_{z_e,l}\mathbf{R}_{y_e,(-\mu-\frac{\pi}{2})} = \begin{bmatrix} -\sin(\mu)\cos(l) & -\sin(\mu)\sin(l) & \cos(\mu) \\ -\sin(l) & \cos(l) & 0 \\ -\cos(\mu)\cos(l) & -\cos(\mu)\sin(l) & -\sin(\mu) \end{bmatrix} \tag{3.13}$$

The position in NED coordinates is then given by

$$\hat{\mathbf{p}}^n = \mathbf{R}_e^n(l,\mu)(\hat{\mathbf{p}}^e - \mathbf{p}_{ref}^e) \tag{3.14}$$

where $\mathbf{p}_{ref}^e$ is the ECEF coordinates of the NED origin.

3.2 Quadcopter Dynamics

In order to properly develop a control system for the quadcopter we need an understanding its physical properties. We need to explore the motor dynamics and use energy considerations in order to derive the forces and thrusts that the motors produce.

3.2.1 Motor Dynamics

All motors on the quadcopter are assumed identical, therefore we can analyze a single one without loss of generality. We use electric brushless motors, and the torque produced by these is given by[10]

$$\tau = K_t(I - I_0) \tag{3.15}$$

where τ is the motor torque, I is the input current, I_0 is the current when there is no load on the motor, and K_t is the torque proportionality constant. The voltage across the motor is the sum of the back electromotive force (voltage, or electromotive force, that pushes against the current which induces it [19]) and some resistive loss[10]

$$V = IR_m + K_v\omega \tag{3.16}$$

where V is the voltage drop across the motor, R_m is the motor resistance, ω is the angular velocity of the motor, and K_v is a proportionality constant (indicating the back electromotive force generated per RPM). We can use this description of our motor to calculate the power it consumes.

$$P = IV = \frac{(\tau + K_t I_o)(K_t I_o R_m + \tau R_m + K_t K_v \omega}{K_t^2} \tag{3.17}$$

To obtain a simpler model, we will assume a negligible motor resistance. Then the power becomes proportional to the angular velocity

$$P \approx \frac{(\tau + K_t I_0) K_v \omega}{K_t} \tag{3.18}$$

Furthermore, we can simplify our model by assuming that $K_t I_0 \ll \tau$. This assumption is derived from the fact that I_0 is the current when there is no load, which is rather small[10]. Thus we obtain our final, simplified equation for power

$$P \approx \frac{K_v}{K_t} \tau \omega \tag{3.19}$$

3.2.2 Thrust Generated by the Propellers

The power generated is used to keep the quadcopter aloft. By conservation of energy, we know that the energy the motor produces in a given time period is equal to the force generated on the propeller times the distance that the air it displaces moves ($P \cdot dt = T \cdot dx$). Equivalently the power is equal to the thrust times the air velocity ($P = T\frac{dx}{dt}$)[10]

$$P = T v_h \tag{3.20}$$

We assume that the free stream velocity (v_∞) is zero (the air in the surrounding environment is stationary relative to the quadcopter). Momentum theory gives us the equation for hover velocity as a function of thrust

$$v_h = \sqrt{\frac{T}{2\rho A}} \tag{3.21}$$

where ρ is the density of the surrounding air and A is the area swept out by the rotor. Using the simplified Equation 3.20 for power, we can then write

$$P = \frac{K_v}{K_t}\tau\omega = \frac{K_v K_\tau}{K_t}T\omega = \frac{T^{\frac{3}{2}}}{\sqrt{2\rho A}} \tag{3.22}$$

In the general case, we got that $\tau = \vec{r} \times \vec{F}$, but in this case the torque is proportional to the thrust T by some constant radio K_t determined by the blade configuration parameters[10]. Solving the thrust magnitude T, we obtain that thrust is proportional to the square of angular velocity of the motor, similar to the the results in the falls project [31]

$$T = \left(\frac{K_v K_\tau \sqrt{2\rho A}}{K_t}\omega\right)^2 = k\omega^2 \tag{3.23}$$

where k is some appropriately dimensioned constant. We find that the total thrust of the quadcopter (in the body reference frame) is given by

$$T^b = \sum_{i=1}^{4} T_i = k \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} \omega_i^2 \end{bmatrix} \tag{3.24}$$

If the quadcopter reaches high speeds, the air drag will also have a considerable impact. The drag can then be modeled as

$$F_D = \begin{bmatrix} c_1 \text{sign}(U)(U)^2 \\ c_2 \text{sign}(V)(V)^2 \\ c_3 \text{sign}(W)(W)^2 \end{bmatrix} \tag{3.25}$$

where

$$\text{sign}(a) = \begin{cases} \text{-1} & \text{if } a < 0 \\ 0 & \text{if } a = 0 \\ 1 & \text{if } a > 0 \end{cases} \tag{3.26}$$

3.2.3 Torques Generated by the Propellers

Each rotor contributes some torque about the yaw axis and either the roll or pitch axes. The drag equation from fluid dynamics gives us the frictional force[10]

$$F_d = \frac{1}{2}\rho C_D A v^2 \tag{3.27}$$

where ρ is the surrounding air density, A is the reference area(propeller cross-section, not area swept out by the propeller), and C_D is a dimensionless constant. This implies that the torque due to drag is given by

$$\tau_D = \frac{1}{2}R\rho C_d A v^2 = \frac{1}{2}R\rho C_D A(\omega R)^2 = b\omega^2 \tag{3.28}$$

where ω is the angular velocity of the propeller, R is the radius of the propeller, and b is some appropriately dimensioned constant. We assumed that all the force is applied at the tip of the propeller, which is inaccurate. However, the only result that matters for our purposes is that the drag torque is proportional to the square of the angular velocity. This is in fact the same relationship we found for the thrust. We can then write the complete torque generated about the yaw axis for the motors as[10]

$$\tau_\psi = b\omega^2 + I_M\dot{\omega} \tag{3.29}$$

where I_M is the moment of inertia about the motor yaw axis, $\dot{\omega}$ is the angular acceleration of the propeller, and b is our drag coefficient. When the quadcopter hovers at a constant height we will have $\dot{\omega} \approx 0$, since the propellers will spin with approximately constant thrust, only slightly varied in order to keep a stable attitude. Thus ignoring this term, simplifies the expression to

$$\tau_\psi = (-1)^{i+1} b\omega_i^2 \tag{3.30}$$

where the $(-1)^{i+1}$ term is positive for the i_{th} propeller if the propeller is spinning clockwise and negative if it is spinning counterclockwise. The total torque generated about the yaw axis is given by the sum of all the torques generated by each propeller.

$$\tau_\psi = b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{3.31}$$

The roll and pitch torques are derived from standard mechanics[10]. We can arbitrarily choose the $i = 1$ and $i = 3$ motors to be on the roll axis

$$\tau_\phi = \sum r \times T = L(k\omega_1^2 - k\omega_3^2) = Lk(\omega_1^2 - \omega_3^2) \tag{3.32}$$

Correspondingly, the pitch torque is given by a similar expression

$$\tau_\theta = \sum r \times T = L(k\omega_2^2 - k\omega_4^2) = Lk(\omega_2^2 - \omega_4^2) \tag{3.33}$$

where L is the distance from the center of the quadcopter to any of the four propellers. All together, we can describe the torques in the body reference frame as

$$\tau^b = \begin{bmatrix} Lk(\omega_1^2 - \omega_3^2) \\ Lk(\omega_2^2 - \omega_4^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{3.34}$$

The thrust and torque forces we have derived so far are highly simplified. We have made assumptions and simplifications, and ignored a multitude of advanced effects that contribute to the highly nonlinear dynamics of the quadcopter. We have ignored the rotational drag forces (our rotational velocities are relatively low), blade flapping (deformation of the propeller blades due to high velocities and the fact that the propeller consist of a thin flexible material), surrounding fluid velocities (wind and other disturbances created by both the quadcopter and its environment) to mention a few.

3.2.4 Equations of Motion

In the inertial (NED) frame, the acceleration of the quadcopter is due to thrust generated by the motors, gravity and air drag. We can obtain the thrust vector in the inertial frame by using our rotation matrix R_b^n to map the thrust vector from the body frame to the inertial frame. Thus, the linear motion can be summarized as

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R_b^n T^b + F_D \tag{3.35}$$

where $\vec{x}$ is the position of the quadcopter, g is the acceleration due to the gravity, F_D is the drag force (equation 3.27), and T^b is the thrust vector in the body frame (equation 3.24). While it is convenient to have the linear equations of motion in the inertial frame, the rotational equations of motion are useful to us in the body frame so that we can express rotations about the center of the quadcopter instead of about our inertial center (typical initial position for the quadcopter). We derive the rotational equations of motion from Euler's equations for rigid body dynamics. Expressed in vector form, Euler's equations are written as [10]

$$\mathbf{I}\dot{\omega} + \omega \times (\mathbf{I}\omega) = \tau \tag{3.36}$$

where ω is the angular velocity vector, $\mathbf{I}$ is the inertia matrix, and τ is a vector of external torques. We can rewrite this as

$$\dot{\omega} = \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \mathbf{I}^{-1}(\tau - \omega \times (\mathbf{I}\omega)) \tag{3.37}$$

We can model our quadcopter as two thin uniform rods crossed at the origin with a point mass (motor) at the end of each, and ignore the various components located close to the center of the quadcopter, including the battery. With this simplification in mind, it's clear that the symmetries result in a diagonal inertia matrix of the form

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.38}$$

Using this we obtain our final result for the body frame rotational equations of motion

$$\dot{\omega} = \begin{bmatrix} \tau_\phi I_{xx}^{-1} \\ \tau_\theta I_{yy}^{-1} \\ \tau_\psi I_{zz}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz}-I_{xx}}{I_{xx}} \omega_x \omega_z \\ \frac{I_{xx}-I_{yy}}{I_{xx}} \omega_x \omega_y \end{bmatrix} \tag{3.39}$$

3.2.5 Kinematic Model

The dynamic 6 DOF kinematic model of the quad-copter can be expressed in vector form as [31]

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_\Theta(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.40}$$

$$\Updownarrow$$

$$\begin{bmatrix} \dot{\mathbf{P}}_{b/n}^n \\ \dot{\mathbf{\Theta}}_{nb} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_b^n(\mathbf{\Theta}_{nb}) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{T}_\Theta(\mathbf{\Theta}_{nb}) \end{bmatrix} \begin{bmatrix} \mathbf{v}_{b/n}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix} \tag{3.41}$$

Where the notation $\mathbf{v}_{b/n}^b$ represents velocity of the BODY origin o_b with respect to NED in {b}, and thus $\dot{\mathbf{P}}_{b/n}^n$ is the NED velocity vector for the BODY with respect to NED. $\boldsymbol{\omega}_{b/n}^b = [p, q, r]$ is the body-fixed angular velocity vector, $\mathbf{R}_b^n(\mathbf{\Theta}_{nb})$ is the rotation matrix from BODY to NED, and $\mathbf{\Theta}_{nb} = [\phi, \theta, \psi]$ are the Euler angles from NED to BODY. For easier notation, $\mathbf{\Theta} = \mathbf{\Theta}_{nb}$. Furthermore

$$\mathbf{T}(\mathbf{\Theta}) = \begin{bmatrix} 1 & \tan(\theta)\sin(\phi) & \tan(\theta)\cos(\phi) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{3.42}$$

3.2.6 Kinetic Model

The 6DOF kinetic model of the quad-copter can be expressed in vector form as [31]

$$\mathbf{M}_{RB}^{CG} \begin{bmatrix} \dot{\mathbf{v}}_{g/n}^b \\ \dot{\boldsymbol{\omega}}_{b/n}^v \end{bmatrix} + \mathbf{C}_{RB}^{CG} \begin{bmatrix} \mathbf{v}_{g/n}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_g^b \\ \mathbf{m}_g^b \end{bmatrix} \tag{3.43}$$

or

$$\underbrace{\begin{bmatrix} m\mathbf{I}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{I}_g \end{bmatrix}}_{\mathbf{M}_{RB}^{CG}} \begin{bmatrix} \dot{\mathbf{v}}_{g/n}^b \\ \dot{\boldsymbol{\omega}}_{b/n}^b \end{bmatrix} + \underbrace{\begin{bmatrix} m\mathbf{S}(\boldsymbol{\omega}_{b/n}^b) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & -\mathbf{S}(\mathbf{I}_g\boldsymbol{\omega}_{b/n}^b) \end{bmatrix}}_{\mathbf{C}_{RB}^{CG}} \begin{bmatrix} \mathbf{v}_{g/n}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_g^b \\ \mathbf{m}_g^b \end{bmatrix} \quad (3.44)$$

4 Filter Designs

4.1 Theory

4.1.1 Low-pass Filter

A low-pass filter is a filter that passes signals with a frequency lower than a certain cut-off frequency, and is thus useful for filtering signals with high frequency noise. A simple first order unity gain low-pass filter can be described by the transfer function

$$G_{lp}(s) = \frac{1}{\tau s + 1} \tag{4.1}$$

where τ is the filter time constant. The cut-off frequency is then given by $f_c = \frac{1}{\pi\tau}$

4.1.2 High-pass Filter

A high-pass filter is a filter that passes signals with a frequency above a certain cut-off frequency, and is thus useful for filtering signals with low frequency noise. A first order unity gain high-pass filter transfer function is simply the complement of the transfer function found in Equation 4.1, and is given by

$$G_{hp}(s) = 1 - G_{lp}(s) = \frac{\tau s}{\tau s + 1} \tag{4.2}$$

4.1.3 Complementary Filter

The term "complementary filter" is used when referring to a digital algorithm that "blends" redundant data from different sensors which exhibit noise with different frequency content. The filter refers to the use of two or more transfer functions $\mathbf{G}_i(s)$, $i = 1 \ldots n$ such that $\sum_{i=1}^{n} \mathbf{G}_i(s) = \mathbf{I}$, or $\sum_{i=1}^{n} G_i(s) = 1$ in case of a one-dimensional filter. Typically with a two-input system, one input provides data with low frequency noise, and is thus high-pass filtered, while the other input provides data with high frequency noise, and is thus low-pass filtered. Mathematically, if the low-pass and high-pass filters are complements, the output of the filter is the complete reconstruction of the signal, minus the noise associated with the sensors. The complementary filter applied to a two-input system with signals y_1 and y_2, where y_1 contains high frequency noise and y_2 contains low-frequency noise, is given by

$$Y_{cf} = G_{lp}(s)Y_1 + G_{hp}Y_2 \tag{4.3}$$

4.1.4 Continuous-discrete Extended Kalman Filter

The extended Kalman filter is the nonlinear version of the Kalman filter. The Kalman filter, also known as linear quadratic estimation (LQE), is an algorithm that fuses related measurements observed over time, and produces estimates of unknown variables. The filter assumes that the system dynamics are given by the non-linear system dynamics

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\xi} \\ \mathbf{y}(t_n) &= \mathbf{h}(\mathbf{x}(t_n), \mathbf{u}(t_n)) + \boldsymbol{\eta}(t_n) \end{aligned} \tag{4.4}$$

where $\mathbf{y}(t_n)$, $\mathbf{x}(t_n)$ and $\mathbf{u}(t_n)$ are the nth samples of $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{x}$ respectively. $\boldsymbol{\xi}$ is a vector of zero-mean Gaussian random processes with covariance matrix $\mathbf{Q}$, and $\boldsymbol{\eta}(t_n)$ is a vector of zero-mean Gaussian random variables with covariance $\mathbf{R}$. $\mathbf{R}$ can usually be determined from sensor data, while $\mathbf{Q}$ is generally unknown. Therefore $\mathbf{Q}$ is usually looked upon as a tuning matrix. The Continuous-discrete Extended Kalman filter accounts for sampled sensor readings by propagating the system model and the covariance of the estimation error $\mathbf{P}$, between samples using the equations [28]

$$\begin{aligned} \dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} \\ \mathbf{A} &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\hat{\mathbf{x}}, \mathbf{u}) \\ \dot{\mathbf{P}} &= \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^{\mathbf{T}} + \mathbf{Q} \end{aligned} \tag{4.5}$$

where $\hat{\mathbf{x}}$ is the estimate of $\mathbf{x}$, and then update the estimation for each sensor, i, when a measurement is received using

$$\begin{aligned} \mathbf{C}_i &= \frac{\partial h_i}{\partial \mathbf{x}}(\hat{\mathbf{x}}, \mathbf{u}) \\ \mathbf{L}_i &= \mathbf{P}\mathbf{C}_i^T(\mathbf{R}_i + \mathbf{C}_i\mathbf{P}\mathbf{C}_i^T)^{-1} \\ \mathbf{P} &= (\mathbf{I} - \mathbf{L}_i\mathbf{C}_i)\mathbf{P} \\ \hat{\mathbf{x}} &= \hat{\mathbf{x}} + \mathbf{L}(\mathbf{y}(t_n) - \mathbf{h}(\hat{\mathbf{x}}, \mathbf{u})) \end{aligned} \tag{4.6}$$

More information can be found in Fossen [29]

4.2 Problem Description

To reduce the noise from the sensors readings, filters can be applied to the data. Different sensors possess different noise characteristics, and therefore require different filters. Redundant sensor data can be combined, and knowledge of the system model can be used to obtain even better results. The filter equations

above however, are not well suited for implementation on a microcontroller, as they are continuous or contains matrix operations. We will now explore how these filters can be implemented in C on the Arduino microcontroller.

4.3 Discrete Implementation

4.3.1 Low-pass Filter

Low-pass filtering a one-dimensional signal using the transfer function given in Equation 4.1, yields

$$Y_{lp} = \frac{1}{\tau s + 1} Y \tag{4.7}$$

$$Y_{lp}(\tau s + 1) = Y \tag{4.8}$$

$$\Downarrow \mathcal{L}^{-1}$$

$$\dot{y}_{lp}\tau + y_{lp} = y \tag{4.9}$$

Backwards difference approximation is given by

$$\dot{x} \approx \frac{x[n] - x[n-1]}{h} \tag{4.10}$$

where $x[n]$ is the nth sample of x, and h is the time between samples. Applying backward difference approximation to Equation 4.9, which gives

$$\frac{y_{lp}[n] - y_{lp}[n-1]}{h}\tau + y_{lp}[n] = y[n] \tag{4.11}$$

$$\Updownarrow$$

$$y_{lp}[n] = \frac{\tau}{\tau + h} y_{lp}[n-1] + \frac{h}{\tau + h} y[n-1] \tag{4.12}$$

where h is the time between samples. This can be simplified by defining

$$a = \frac{\tau}{\tau + h} \tag{4.13}$$

$$\Downarrow$$

$$\frac{h}{\tau + h} = 1 - a \tag{4.14}$$

where a can be set to a constant value, and is considered a tuning parameter. Hence, a simple unity gain low-pass filter in discrete form, is given by

$$y_{lp}[n] = a y_{lp}[n-1] + (1-a) y[n] \tag{4.15}$$

which can be implemented in C-code.

4.3.2 High-pass Filter

High-pass filtering a one-dimensional signal using the transfer function given in Equation 4.2, yields

$$Y_{hp} = \frac{\tau s}{\tau s + 1} Y \tag{4.16}$$

$$Y_{hp}(\tau s + 1) = \tau s Y \tag{4.17}$$

$$\Downarrow \mathcal{L}^{-1}$$

$$\dot{y}_{hp}\tau + y_{hp} = \tau \dot{y} \tag{4.18}$$

Discretization using backward difference approximation, gives

$$\frac{y_{hp}[n] - y_{hp}[n-1]}{h}\tau + y_{hp}[n] = \frac{y[n] - y[n-1]}{h}\tau \tag{4.19}$$

where h is the time between samples. With a defined as in Equation 4.13, the filter takes the form

$$y_{hp}[n] = a y_{hp}[n-1] + a(y[n] - y[n-1]) \tag{4.20}$$

which can be implemented in C-code.

4.3.3 Complementary Filter

The complementary filter Equation 4.3 can be written

$$Y_{cf} = \frac{1}{\tau s + 1} Y_1 + \frac{\tau s}{\tau s + 1} Y_2 \tag{4.21}$$

$$Y_{cf}(\tau s + 1) = Y_1 + \tau s Y_2 \tag{4.22}$$

$$\Downarrow \mathcal{L}^{-1}$$

$$\dot{y}_{cf}\tau + y_{cf} = y_1 + \tau \dot{y}_2 \tag{4.23}$$

Discretization using backward difference approximation yields

$$\frac{y_{cf}[n] - y_{cf}[n-1]}{h}\tau + y_{cf}[n] = y_1[n] + \frac{y_2[n] - y_2[n-1]}{h}\tau \tag{4.24}$$

where h is the time between samples. With a defined as in Equation 4.13, the filter takes the form

$$y_{cf}[n] = a y_{cf}[n-1] + (1-a) y_1[n] + a(y_2[n] - y_2[n-1]) \tag{4.25}$$

which can be implemented in C-code.

4.3.4 Continuous-discrete Extended Kalman Filter

To allow simple usage of the filter, we decided to turn the filter Equations 4.5 and 4.6 into a C library. By making the functions general with respect to the number of states and measurements, and making the state and measurement propagation matrices interchangeable, the filter can be applied to a wide variety of systems. Since there is no built-in matrix support in C, an open source matrix math library made by Charlie Matlack [7] enabled us to do the necessary matrix operations. Below is a pseudo code example of how the library can be used

```
Define functions:

void f_func(float* x_hat, float* u, float h){
    propagate x_hat
}
void df_dx(float* A, float* x_hat, float* u){
    A = df/dx(x_hat,u)
}
void h_func(float* H, float* x_hat, float* u){
    H = h(x_hat,u)
}
void dh_dx(float* C, float* x_hat, float* u){
    C = dh/dx(x_hat,u)
}

void Android loop(){
    n = number of states
    m = number of measurements
initialize:
    x_hat[n][1] = initial state guess
    P[n][n]     = initial P matrix
    Q[n][n]     = system states covariance matrix
    R[m][m]     = measurement covariance matrix
loop:
    h = loop time
    u = current system input
    call Kalman.kalmanPredict(*f_func, *df_dx , (float*) x_hat, (float*) u, (
        float*) P, (float*) Q, h, n);

    if (new sensor measurements) {
        z[m][1] = new sensor measurements
        h2 = time between sensor measurements
        call Kalman.kalmanMeasurement(*h_func, *dh_dx, (float*) x_hat, (float
            *) u, (float*) z, (float*) P, (float*) R, n, m);
    }
}
```

4.4 Discussion

Comparing the filter equations; the high-pass, low-pass and complementary filters are all very simple to implement. The Kalman filter matrices also grow exponentially with the number of states and the number of measurements. However, the Kalman filter takes advantage of the system model, and therefore should in

theory result in better results.

4.5 Recommendations and Future Work

To improve the Kalman library, the **P**, **Q** and **R** matrices could be set as private variables, along with necessary initializing functions. Improvements made to the matrix math library could also be made, by taking advantage of the fact that some of the matrices are diagonal.

5 Estimation of Roll and Pitch

An inertial measurement unit is composed of an accelerometer and a gyroscope. The accelerometer measures the accelerations along the x, y and z axes in BODY coordinates, and the gyroscope measures the angular speed around the axes.

5.1 Theory

5.1.1 Gravity Vector

The Earth's gravity vector in NED coordinates can be estimated as

$$\mathbf{g}^n = \begin{bmatrix} 0 \\ 0 \\ 9.81 \end{bmatrix} \tag{5.1}$$

5.1.2 Accelerometer Measurement Model

The measurement model, what the accelerometer measures, for the accelerometer, can be found using Newton's second law, which yields

$$\mathbf{a}^b_{imu} = \frac{1}{m}(\mathbf{F}_{tot} - \mathbf{F}_g) + \boldsymbol{\eta}_{acc} \tag{5.2}$$

where $\mathbf{F}_g = m\mathbf{R}^b_n(\boldsymbol{\Theta})\mathbf{g}^n$, $\mathbf{F}_{tot} = m\dot{\mathbf{v}}^b_{b/n} + m\boldsymbol{\omega}^b_{b/n} \times \mathbf{v}^b_{b/n}$ and $\boldsymbol{\eta}_{acc} = \mathbf{b}^b_{acc} + \mathbf{w}^b_{acc}$. $\mathbf{b}^b_{acc}$ is a sensor bias vector and $\mathbf{w}^b_{acc}$ is a sensor noise vector. Insertion yields

$$\mathbf{a}^b_{imu} = \dot{\mathbf{v}}^b_{b/n} + \boldsymbol{\omega}^b_{b/n} \times \mathbf{v}^b_{b/n} - \mathbf{R}^b_n(\boldsymbol{\Theta})\mathbf{g}^n + \boldsymbol{\eta} \tag{5.3}$$

A simplified measurement model is obtained by approximating $\dot{\mathbf{v}}^b_{b/n} = \mathbf{v}^b_{b/n} \times \boldsymbol{\omega}^b_{b/n} = \mathbf{0}_{3\times 1}$. This simplification necessary as the accelerometer can not differentiate between the different forces acting on it. Expressed in component form this gives

$$\begin{aligned} a_x &= g\sin(\theta) + \eta_x \\ a_y &= -g\cos(\theta)\sin(\phi) + \eta_y \\ a_z &= -g\cos(\theta)\cos(\phi) + \eta_z \end{aligned} \tag{5.4}$$

5.1.3 Gyroscope Measurement Model

The measurement model for the gyroscope when the accelerometer is placed in the center of gravity, is simply found as

$$\boldsymbol{\omega}^b_{imu} = \boldsymbol{\omega}^b_{b/n} + \mathbf{b}^b_{gyro} + \mathbf{w}^b_{gyro} \tag{5.5}$$

5.2 Problem Description

To simplify the readings from the IMU, the data should be scaled to obtain the accelerometer and gyroscope measurements in *m/s* and *rad/s* respectively. Furthermore, accelerometer and gyroscope bias should be estimated and removed. Using the resulting accelerometer and/or gyroscope readings, we want to obtain fast, smooth, and close to bias-free roll and pitch estimates.

5.3 Design of Solution

5.3.1 Inertial Measurement Unit Sensitivity Range and Scaling

The MPU6050 features four different sensitivity ranges for the accelerometer and the gyroscope, with four resulting scale factors. For the accelerometer these are given as

Sensitivity range	Units	Scaling factor
± 2	g	16384
± 4	g	8192
± 8	g	4096
± 16	g	2048

Table 5.1: Sensitivity ranges and scales, accelerometer

and for the gyroscope

Sensitivity range	Units	Scaling factor
± 250	$^\circ$/s	131
± 500	$^\circ$/s	65.5
± 1000	$^\circ$/s	32.8
± 2000	$^\circ$/s	16.4

Table 5.2: Sensitivity ranges and scales, gyroscope

The default sensitivity of the accelerometer and gyroscope are 2g and $250^\circ/s$ respectively, which is suitable for our purpose. Hence to transform the sensor data into units of gravity and units of degrees per second, the accelerometer and gyroscope measurements must be divided by 16384 and 131 respectively. Furthermore, to obtain the acceleration in meters per second, the accelerometer sensor data must be multiplied by 9.81 m/s.

5.3.2 Accelerometer Bias Compensation

The accelerometer bias vector $\mathbf{b}^b_{acc}$ can be estimated by placing the accelerometer such that $\phi \approx \theta \approx 0$. From the simplified measurement model Equations 5.4, an expression for the bias can then be found as

$$\mathbf{a}^b_{acc} = -\mathbf{R}^b_n(\boldsymbol{\Theta})\mathbf{g}^n + \mathbf{b}^b_{acc} + \mathbf{w}^b_{acc} \tag{5.6}$$
$$= -\mathbf{g}^n + \mathbf{b}^b_{acc} + \mathbf{w}^b_{acc} \tag{5.7}$$
$$\Updownarrow \tag{5.8}$$
$$\mathbf{b}^b_{acc} = \mathbf{a}^b_{acc} + \mathbf{g}^n - \mathbf{w}^b_{acc} \tag{5.9}$$

By sampling enough data, and averaging the resulting measurements, the effect of the white noise vector $\mathbf{w}^b_{acc}$ is mitigated, and an estimate of $\mathbf{b}^b_{acc}$ can be obtained.

5.3.3 Gyroscope Bias Compensation

The gyroscope bias vector $\mathbf{b}^b_{gyro}$ can be estimated by placing the IMU on a table such that $\boldsymbol{\omega}^b_{b/n} = \mathbf{0}_{3\times1}$. From the measurement model Equation 6.2, an expression for the bias can then be found as

$$\boldsymbol{\omega}^b_{gyro} = \mathbf{b}^b_{gyro} + \mathbf{w}^b_{gyro} \tag{5.10}$$
$$\mathbf{b}^b_{gyro} = \boldsymbol{\omega}^b_{gyro} - vekw^b_{gyro} \tag{5.11}$$

By sampling enough data, and averaging the resulting measurements, the effect of the white noise vector $\mathbf{w}^b_{gyro}$ can be mitigated, and an estimate of $\mathbf{b}^b_{gyro}$ can be obtained.

5.3.4 Accelerometer Estimate

Using the simplified accelerometer model Equations 5.4 and solving for ϕ and θ gives

$$\phi_{acc} = \tan^{-1}(\frac{a_y}{a_z}) \tag{5.12}$$
$$\theta_{acc} = \sin^{-1}(\frac{a_x}{g}) \tag{5.13}$$

The accelerometer is very sensitive to vibrations and contains a fair amount of high frequency noise. An improvement to the estimate can thus be made by filtering the data using the low-pass filter Equation 4.15, found in Section 4.3.

5.3.5 Gyroscope Estimate

The roll and pitch angles can be estimated from the gyroscope readings as

$$\phi_{gyro} = \int_0^t p(\tau)d\tau \tag{5.14}$$

$$\theta_{gyro} = \int_0^t q(\tau)d\tau \tag{5.15}$$

or discretely, using Euler's method, as

$$\phi_{gyro}[n] = \phi_{gyro}[n-1] + h \cdot p \tag{5.16}$$

$$\theta_{gyro}[n] = \theta_{gyro}[n-1] + h \cdot q \tag{5.17}$$

where h is the time between samples. The gyroscope measurements suffer from low frequency random walk noise, which accumulates when integrated in the angle estimates above. To mitigate some of this effect, the data can be filtered using the high-pass filter Equation 4.20, found in Section 4.3.

5.3.6 Complementary Filter Estimate

By combining the accelerometer and gyroscope estimate through the complementary filter Equation 4.25, where y_1 represents the accelerometer estimate and y_2 represents the gyroscope estimate, we obtain the following roll and pitch estimate equations

$$\phi[n] = a\phi[n-1] + (1-a)\phi_{acc}[n] + a(\phi_{gyro}[n] - \phi_{gyro}[n-1]) \tag{5.18}$$

$$\theta[n] = a\theta[n-1] + (1-a)\theta_{acc}[n] + a(\theta_{gyro}[n] - \theta_{gyro}[n-1]) \tag{5.19}$$

A simpler implementation which directly integrates the gyroscope Equations 5.16 into the complementary filter, is obtained by inserting $\phi_{gyro}[n] = \phi[n] + h \cdot p$, $\phi_{gyro}[n-1] = \phi[n-1]$, $\theta_{gyro}[n] = \theta[n] + h \cdot q$ and $\theta_{gyro}[n-1] = \theta[n-1]$; which yields

$$\phi[n] = a(\phi[n] + h \cdot p) + (1-a)\phi_{acc} \tag{5.20}$$

$$\theta[n] = a(\theta[n] + h \cdot p) + (1-a)\theta_{acc} \tag{5.21}$$

where h is the time between samples, and a is the filter design parameter.

5.3.7 Kalman Filter Implementation

Another solution which combines the accelerometer and gyroscope measurements, can be obtained from the Extended Kalman Filter as described in Section 4.1.4

and 4.3.4. The Kalman filter requires the system propagation functions and the resulting jacobian matrices. From the kinetic Equations 3.41 we have

$$\dot{\phi} = p + q\sin(\phi)\tan(\theta) + r\cos(\phi)\tan(\theta) + \xi_\phi \tag{5.22}$$

$$\dot{\theta} = q\cos(\phi) - r\sin(\phi) + \xi_\theta \tag{5.23}$$

Using the simplified measurement model Equations 5.4 and defining $\mathbf{x} = (\phi, \theta)^T$, $\mathbf{y} = (a_x, a_y, a_z)^t$, $\mathbf{u} = (p, q, r, u, v, w)^T$ and $\boldsymbol{\xi} = (\xi_\phi, \xi_\theta)^T$, yields

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\xi} \tag{5.24}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\eta} \tag{5.25}$$

where

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} p + q\sin(\phi)\tan(\theta) + r\cos(\phi)\tan(\theta) \\ q\cos(\phi) - r\sin(\phi) \end{bmatrix}$$

$$\mathbf{h}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} g\sin(\theta) \\ -g\cos(\theta)\sin(\phi) \\ -g\cos(\theta)\cos(\phi) \end{bmatrix}$$

which yields the Jacobians

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} q\cos(\phi)\tan(\theta) - r\sin(\phi)\tan(\theta) & \frac{q\sin(\phi) - r\cos(\phi)}{\cos^2(\theta)} \\ -q\sin(\phi) - r\cos(\phi) & 0 \end{bmatrix}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \begin{bmatrix} 0 & g\cos(\theta) \\ -g\cos(\phi)\cos(\theta) & g\sin(\phi)\sin(\theta) \\ g\sin(\phi)\cos(\theta) & g\cos(\phi)\sin(\theta) \end{bmatrix}$$

Furthermore, the Kalman filter requires that we define the matrices $\mathbf{x}_0$, $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$. Assuming that we start on a relative flat surface, we set $\mathbf{x}_0 = 0$ and $\mathbf{P} = \mathbf{I}_{3\times 3} \cdot 0.1$. $\mathbf{Q}$ and $\mathbf{R}$ are tuning matrices.

5.4 Observations and Results

A comparison between the accelerometer estimate, the gyroscope estimate, the complementary estimate and the extended Kalman filter estimate, can be seen in Figure 5.1. Here, $\alpha = 0.98$ in the Complementary filter, and we found $Q = \vec{I}_{2\times 2} \cdot 2e^{-5}$ and $\mathbf{R} = \mathbf{I}_{3\times 3} \cdot 0.8$ to give satisfactory results in the Kalman filter. The samples were taken the 11th of April 2015, with two motors running to see how the motor forces would effect the readings.

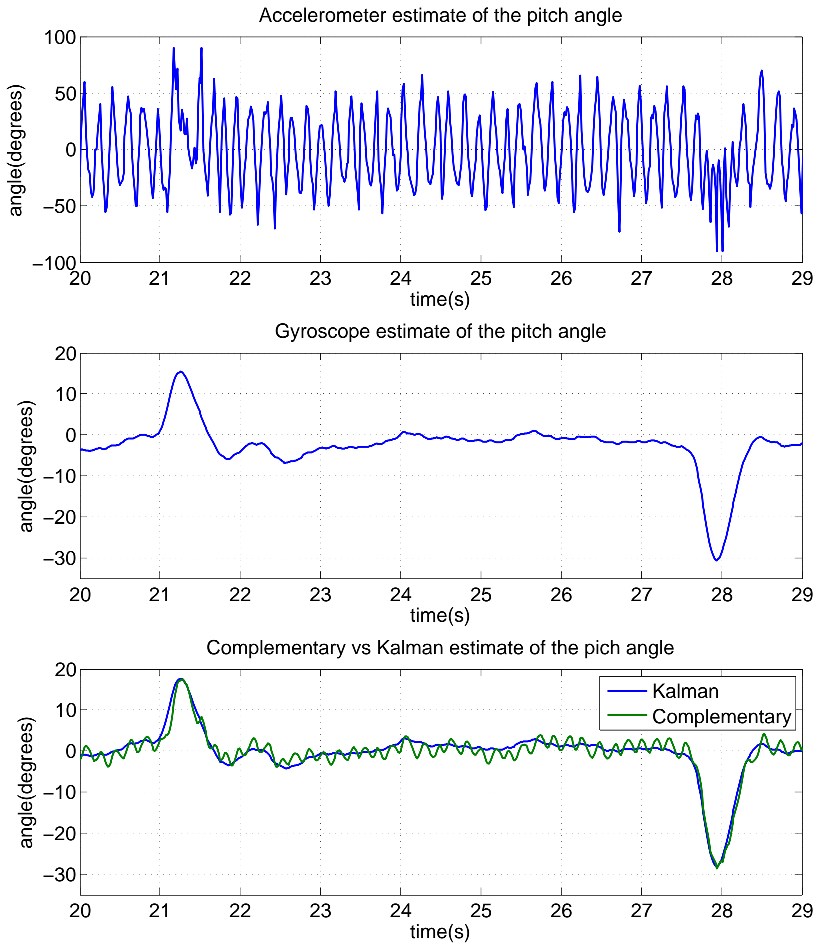


Figure 5.1: A comparison between the different estimates

5.5 Discussion and Conclusion

The accelerometer measures all forces acting on the quadcopter and not only the gravity vector. Hence when the quadcopter is accelerating or turning in any way, this will affect the estimates. Motor vibrations also cause great disturbances in

the estimate, which can be seen in the first graph in Figure 5.1. In the long term, however, the accelerometer measurements can be used to obtain a very rough estimate of the angle.

The gyroscope is great at detecting rapid changes in rotation, but if there is a bias in the measurements, simply integrating the rotation rate will result in an increasing bias error over time, which can be seen when comparing the gyroscope estimate in the middle graph with the estimates in the lowest graph in Figure 5.1.

The Complementary Filter does a decent job at combining the gyroscope and the accelerometer readings, which can be seen in the last graph in Figure 5.1. The estimate is close to bias-free, but is however affected by the noisy accelerometer estimate.

The Extended Kalman Filter estimate is not only smoother than the Complementary Filter estimate, but it appears to react slightly faster to changes in the angle, which can be seen in the last graph in Figure 5.1. When compared to the gyroscope estimate, the smoothness and speed has been maintained, while the bias has been removed, which is exactly the results we had hoped for.

In conclusion, if computational power does not pose an issue, the Kalman filter estimate is the best option.

5.6 Recommendations and Future Work

If the quadcopter is to perform acrobatic maneuvers, we would recommend using quaternions instead of Euler angles to describe the attitude of the quadcopter. The quaternion approach is harder to understand and therefore harder to implement, but the quaternion model does not have singularities at $\theta = \pm 90°$ and it may also yield somewhat faster computations.

6 Estimation of Yaw

6.1 Theory

A magnetometer is a sensor that measures the strength of the local magnetic field. The HMC5883L measures the magnetic strength in three axes, which can be be used to achieve tilt compensated yaw estimation. The local magnetic field is dependent on the Earth's magnetic field and any magnetic fields created by nearby objects.

6.1.1 Magnetometer measurement model

The magnetometer measurement model can be expressed as

$$\mathbf{m}^b_{mag} = \mathbf{D}^{-1}\left(\mathbf{R}^b_n(\mathbf{\Theta})\mathbf{m}^n + \mathbf{b}^b_{mag}\right) + \mathbf{w}^b_{mag} \tag{6.1}$$

where $\mathbf{R}^b_n(\mathbf{\Theta})$ is the rotation matrix from NED to BODY with respect to the attitude $\mathbf{\Theta}$, $\mathbf{b}^b_{mag}$ is the magnetometer bias from hard iron distortions, $\mathbf{w}^b_{mag}$ is a measurement noise vector and $\mathbf{D}$ depends on soft iron distortions. The vector $\mathbf{m}^n$ is the output from the magnetometer in NED coordinates when all angles are zero, and the x-axis of the NED coordinate system is pointing towards magnetic north.

6.1.2 Gyroscope Measurement Model

The measurement model for the gyroscope when the accelerometer is placed in the center of gravity, is simply found as

$$\boldsymbol{\omega}^b_{imu} = \boldsymbol{\omega}^b_{b/n} + \mathbf{b}^b_{gyro} + \mathbf{w}^b_{gyro} \tag{6.2}$$

6.1.3 Earth's Magnetic Field

The Earth's magnetic field is a magnetic dipole with ends near the geographic North- and South- Poles. The magnetic strength varies across the earth, along with the inclination of the magnetic directional vector. For most practical purposes, the magnetic field can be considered constant with respect to time.

The magnetic north is not located at the true north pole, but is "split up" into two different centers close to the true north. Which of the two magnetic centers influence the magnetometer the most, is dependent on where you are located on Earth. This can be seen in Figure 6.1, where the *magnetometer north* will be pointing upwards, normal on the intensity contour lines. In this figure the inclination of the north vector can also be seen. In Trondheim, the declination angle is approximately 2°.

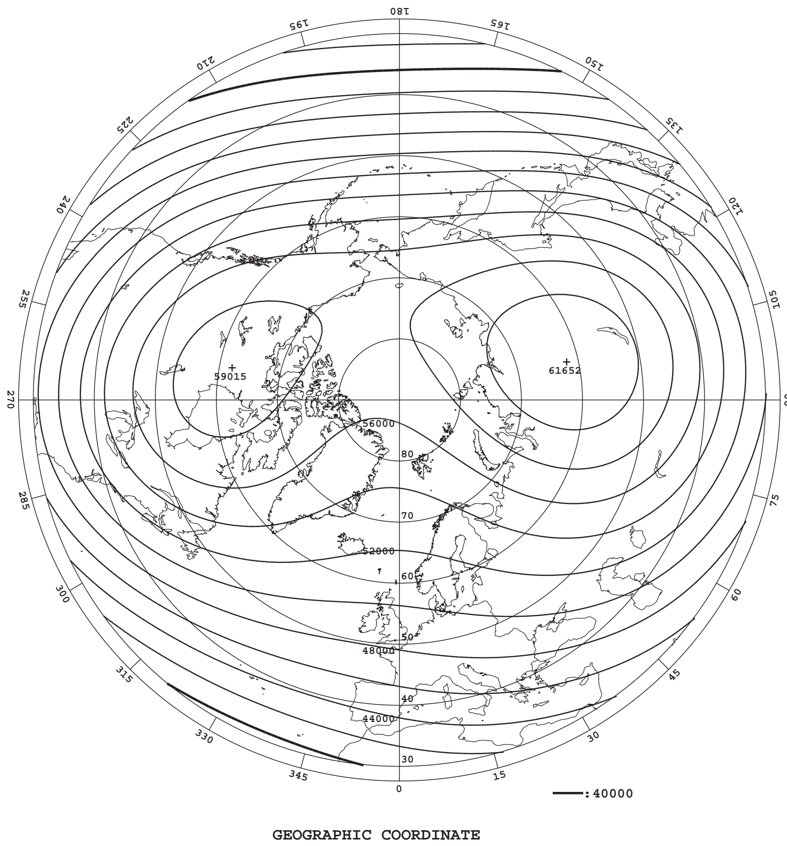


Figure 6.1: Total magnetic intensity

6.1.4 Hard and Soft Iron Distortions

The magnetic measurements will be subjected to mainly two types of distortions; hard iron and soft iron. Hard iron distortions are caused by nearby objects with magnetic fields, which create somewhat constant bias terms on the magnetometer readings. Soft iron distortions are considered alterations in the existing magnetic fields, stretching the magnetic field relative to the sensor. Hard iron distortions contributes the most to the total sensor error. To visualize the two different distortions and their effect on the magnetometer output, we illustrate the different effects using 2D plots of the m_x and m_y outputs.

With no distortions in the magnetic field, a plot of m_x versus m_y will simply be a circle with a center at the origin and radius equal to the intensity of the magnetic field in the x-y plane. This is illustrated in Figure 6.2.

Hard iron distortions causes the center of the circle to change. An illustration of the effects caused by hard iron distortions can be seen in Figure 6.3.

Soft iron distortions causes the circle to stretch into an ellipse. Soft iron distortions have no effect on the center of the ellipse. An illustration of the effects caused by a combination of hard and soft iron distortions can be seen in Figure 6.4.

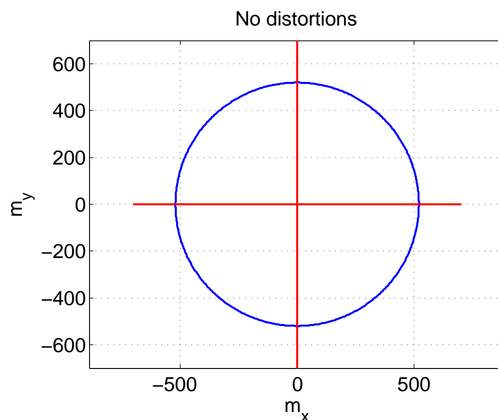


Figure 6.2: Magnetic measurement with no distortions

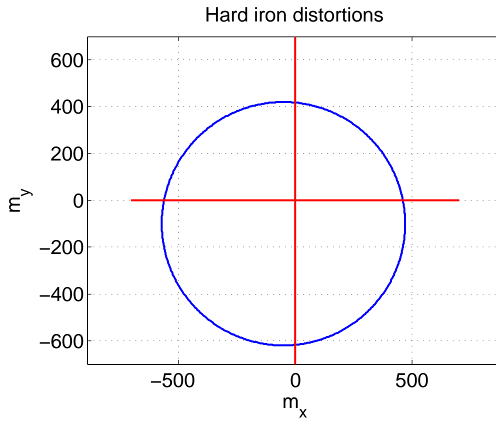


Figure 6.3: Magnetic measurement with hard iron distortions

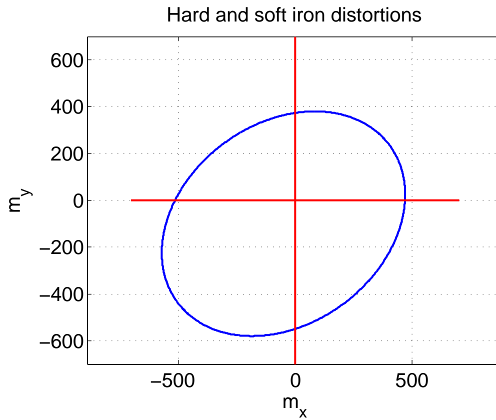


Figure 6.4: Magnetic measurement with hard and soft iron distortions

6.2 Problem Description

The hard and soft iron distortions should be removed from the magnetometer measurements. Furthermore, the magnetometer sensor readings are dependent on the roll and pitch angles of the quadcopter. The readings should therefore be tilt compensated. Using the resulting magnetometer readings and the gyroscope readings, we want to obtain a fast, smooth and close to bias free yaw angle estimate.

6.3 Design of Solution

6.3.1 Hard and Soft Iron Distortions Compensation

Hard iron distortions can be estimated by sampling data from almost all attitudes and taking the maximum and minimum from each measurement axis before taking the average, resulting in

$$b_x = \frac{1}{2} \cdot \left(\min_i(\mathbf{m}^b_{mag}(1,i)) + \max_i(\mathbf{m}^b_{mag}(1,i)) \right) \tag{6.3}$$

$$b_y = \frac{1}{2} \cdot \left(\min_i(\mathbf{m}^b_{mag}(2,i)) + \max_i(\mathbf{m}^b_{mag}(2,i)) \right) \tag{6.4}$$

$$b_z = \frac{1}{2} \cdot \left(\min_i(\mathbf{m}^b_{mag}(3,i)) + \max_i(\mathbf{m}^b_{mag}(3,i)) \right) \tag{6.5}$$

The soft iron distortions are not as easily found. Instead of determining all the elements in the soft iron distortion matrix, we can obtain an estimate by approximating $\mathbf{D}$ as a diagonal matrix. The diagonal elements can then be estimated as

$$d_{11} = \frac{\bar{d}}{\bar{d}_x} \tag{6.6}$$

$$d_{22} = \frac{\bar{d}}{\bar{d}_y} \tag{6.7}$$

$$d_{33} = \frac{\bar{d}}{\bar{d}_z} \tag{6.8}$$

where $\bar{d}$ is the average distance from the center to the data points representing the circle, and $\bar{d}_i$ is half the distance between the maximum and minimum sensor values on the i axis. These can be estimated as

$$\bar{d}_x = \frac{1}{2} \cdot (\max_i(\mathbf{m}^b_{mag}(1,i)) - \min_i(\mathbf{m}^b_{mag}(1,i))) \tag{6.9}$$

$$\bar{d}_y = \frac{1}{2} \cdot (\max_i(\mathbf{m}^b_{mag}(2,i)) - \min_i(\mathbf{m}^b_{mag}(2,i))) \tag{6.10}$$

$$\bar{d}_z = \frac{1}{2} \cdot (\max_i(\mathbf{m}^b_{mag}(3,i)) - \min_i(\mathbf{m}^b_{mag}(3,i))) \tag{6.11}$$

$$\bar{d} = \frac{\bar{d}_x + \bar{d}_y + \bar{d}_z}{3} \tag{6.12}$$

By defining $\mathbf{m}$ as the output from the magnetometer with hard and soft iron distortions removed, we have

$$\mathbf{m} = \mathbf{D}(\mathbf{m}^b_{mag} - \mathbf{b}^b_{mag}) \tag{6.13}$$

By combining Equation 6.1 and 6.13, we then get a measurement model for the calibrated magnetometer output as

$$\mathbf{m} = \mathbf{R}_n^b(\mathbf{\Theta})\mathbf{m}^n + \mathbf{w}_{mag}^b \tag{6.14}$$

6.3.2 Magnetometer Estimate

Given zero roll and pitch angles, the magnetic heading angle ψ_m satisfies the equation

$$\tan(\psi_m) = \frac{m_y}{m_x} \tag{6.15}$$

When roll and pitch are known, we can transform the magnetometer readings to the horizontal plane, satisfying the equation above. This can be done by multiplying the magnetometer readings with the rotation matrices for roll and pitch, giving

$$\begin{bmatrix} m_x^h \\ m_y^h \\ m_z^h \end{bmatrix} = \mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi} \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} \tag{6.16}$$

$$\Updownarrow \tag{6.17}$$

$$\begin{bmatrix} m_x^h \\ m_y^h \\ m_z^h \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} \tag{6.18}$$

We can then find ψ_m using Equation 6.15. The sign of the arguments m_x^h and m_y^h must be taken into account, giving

$$\psi_m = \begin{cases} 180^\circ - \frac{180^\circ}{\pi}\arctan(\frac{m_y^h}{m_x^h}) & \text{if} \quad m_x^h < 0 \\ -\frac{180^\circ}{\pi}\arctan(\frac{m_y^h}{m_x^h}) & \text{if} \quad m_x^h > 0,\, m_y^h < 0 \\ 360 - \frac{180^\circ}{\pi}\arctan(\frac{m_y^h}{m_x^h}) & \text{if} \quad m_x^h > 0,\, m_y^h > 0 \\ 90^\circ & \text{if} \quad m_x^h = 0,\, m_y^h < 0 \\ 270^\circ & \text{if} \quad m_x^h = 0,\, m_y^h > 0 \end{cases} \tag{6.19}$$

To obtain the true North heading, ψ, the declination angle (which is 2° in Trondheim) must be added to the magnetic heading.

6.3.3 Gyroscope Estimate

The yaw angle can be estimated from the gyroscope readings as

$$\psi_{gyro} = \int_0^t r(\tau)d\tau \tag{6.20}$$

$$\tag{6.21}$$

or discretely, using Euler's method, as

$$\psi_{gyro}[n] = \psi_{gyro}[n-1] + h \cdot r \tag{6.22}$$

$$\tag{6.23}$$

where h is the time between samples. The gyroscope measurement suffers from low frequency random walk noise, which accumulates when integrated in the angle estimate. To mitigate some of this effect, the data can be filtered using the high-pass filter Equation 4.20, found in Section 4.3.

6.3.4 Combined Magnetometer and Gyroscope

To obtain more accurate estimates, the magnetometer readings can be merged with the gyroscope readings using the extended Kalman filter as described in Section 4.1.4 and 4.3.4. The Kalman filter requires the system propagation functions and the resulting jacobian matrices. The non-linear propagation model for yaw, ψ, is found from Equation 3.41 as

$$\dot{\psi} = \frac{\sin(\phi)}{\cos(\theta)}q + \frac{\cos(\phi)}{\cos(\theta)}r + \xi_\psi \tag{6.24}$$

where ξ_ψ represents a system model disturbance. Defining $x = \psi$, $\mathbf{y} = \mathbf{m}$, $\mathbf{u} = (\phi, \theta, q, r)$ and using the magnetometer measurement model found in Equation 6.14, we have

$$\dot{x} = f(x, \mathbf{u}) + \xi_\psi \tag{6.25}$$

$$\mathbf{y} = \mathbf{h}(x, \mathbf{u}) + \mathbf{w}^b_{mag} \tag{6.26}$$

where

$$f(x, u) = \frac{\sin(\phi)}{\cos(\theta)}q + \frac{\cos(\phi)}{\cos(\theta)}r \tag{6.27}$$

$$\mathbf{h}(x, \mathbf{u}) = \mathbf{R}^b_n\boldsymbol{\Theta})\mathbf{m}^n \tag{6.28}$$

Furthermore, we have the Jacobians

$$\frac{\partial f}{\partial x} = 0 \tag{6.29}$$

$$\frac{\partial \mathbf{h}}{\partial x} = \begin{bmatrix} -\cos(\theta)\sin(\psi)m_1 \\ (-\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\theta)\sin(\psi))m_1 \\ (\sin(\phi)\cos(\psi) - \cos(\phi)\sin(\theta)\sin(\psi))m_1 \end{bmatrix} \tag{6.30}$$

Furthermore, the Kalman filter requires that we define the matrices $\mathbf{x}_0$, $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$. Since we do not know the heading when we start up, we set $\mathbf{x}_0 = 0$ and $\mathbf{P} = \mathbf{I}_{3\times3}$. $\mathbf{Q}$ and $\mathbf{R}$ are tuning matrices.

6.4 Observations and Results

6.4.1 Hard and Soft Iron Calibration

To compensate for the hard and soft iron distortions, we had to gather data points from the magnetometer from all possible attitudes. By turning the magnetometer around in space and sending the data over serial to Matlab, we then used the data to estimate the hard and soft iron distortions as described in Section 6.3.1. The resulting output of the magnetometer before and after calibration, can be seen in Figure 6.5.

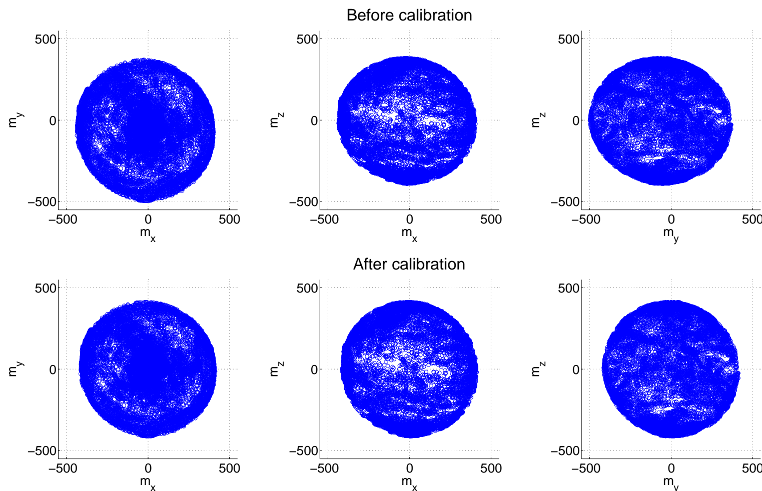
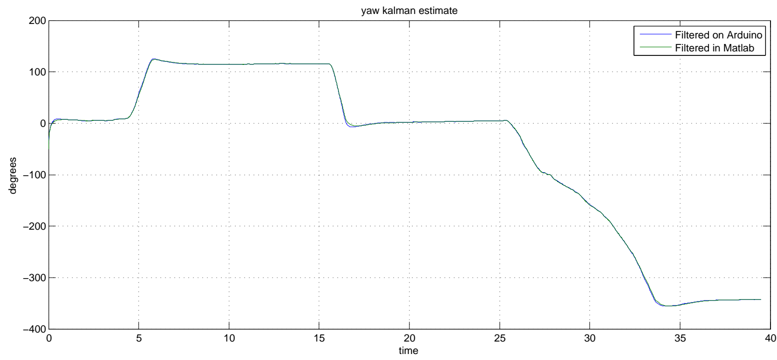


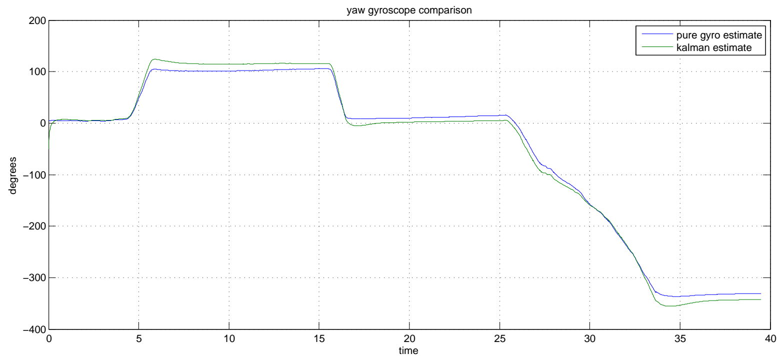
Figure 6.5: Hard and soft iron calibration. Top row: before calibration. Bottom row: after calibration

6.4.2 Yaw Estimates

$\mathbf{R}$ was found by analyzing the variance of the magnetometer output, and was estimated to be $\mathbf{R} = \mathbf{I}_{3\times3} \cdot 0.4$. After some tuning, $\mathbf{Q}$ was set to $\mathbf{Q} = 0.0002 \cdot \mathbf{I}_{3\times3}$.

A comparison between the Extended Kalman Filter, the gyroscope estimate and the magnetometer estimate, can be seen in Figure 6.6.





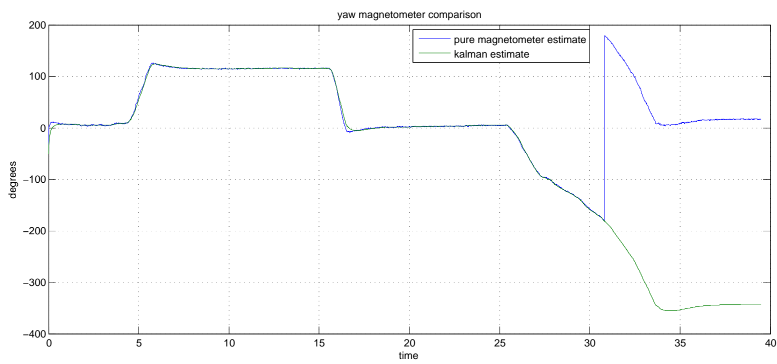


Figure 6.6: Yaw estimates

6.5 Discussion and Conclusion

The gyroscope readings can be used to estimate the yaw angle, as shown in the middle graph of Figure 6.6. However, the estimated angle bias error increases over time, which is not ideal.

The magnetometer estimate yields great yaw estimates as shown in Figure 6.6. The angle jumps when moving between $360°$ and $0°$, however this can be easily handled by the controller through some simple logic.

The Extended Kalman filter estimate which combines the gyroscope readings and the magnetometer readings, has no apparent improvement over the magnetometer estimate. However, the Kalman estimate does provide redundancy should one of the sensors stop working mid flight. A method of detecting sensor loss or sensor fault should then be determined.

In conclusion, if limited computing power poses a problem, the magnetometer readings should be used to estimate yaw, whilst if computing power is not an issue, the Kalman estimate should be used for redundancy. On the Arduino Due, limited computing power does not pose an issue, hence why the Kalman estimate was chosen.

6.6 Recommendations and Future Work

The magnetometer should be calibrated after it has been placed on the quadcopter, since the wires, the micro-controller and the fastening bolts all disturb the measurements. When reading magnetometer data for calibration, avoid moving the magnetometer too close to computers, tables or in general any obstacle that can disturb the measurements.

An alternative way of setting up the propagation equations for the Kalman filter, is to use the rotated magnetometer sensor values as input to the filter, and change the propagation equations for x and y appropriately. This method could then be compared to the method used above, to see if any improvements to computing time or angle estimates are obtained.

7 Estimation of Position

7.1 Theory

7.1.1 Position Measurement from Global Positioning System Technology

Navstar Global Positioning System(GPS) is a dual-use satellite based system that provides positioning and timing data to users worldwide. In the late sixties, the U.S. Air Force and Navy together with the Joint Program Office, laid the foundation for GPS. Even though the first operational satellite was launched in 1978, it was not until 1993 that initial operational capability was declared, and full operational capability was declared in 1994. GPS has been a great success for both military and civil purposes, and can today be found in almost all new cars and smart-phones.

By receiving radio signals from a satellite, the GPS can determine the distance travelled by the radio signal. A specific code sequence is sent from the satellite at time t_1 and is received on the GPS at time t_2. The GPS then generates its own code sequence referenced to the internal clock of the GPS. The generated code sequence is shifted until it's aligned with the incoming code. The time difference can then be determined by how much the generated code had to be shifted. An illustration of this procedure is shown in Figure 7.1. Since the position of the satellite is known, the position of the GPS can be narrowed to a sphere around the satellite with radius $r_1 = (t_2 - t_1)c$, where c is the speed of light. By combining data from at least four satellites, the position of the GPS can then be determined. The pseudorange measurement equation is given by

$$p_i = \sqrt{(\mathbf{p}^n_{s_i} - \mathbf{p}^n)^T(\mathbf{p}^n_{s_i} - \mathbf{p}^n)} + c\tau^* + \epsilon_{p_i} \tag{7.1}$$

where $\mathbf{p}^n_{s_i} \in \mathbb{R}^3$ is the position of satellite i, c is the speed of light, τ^* is the GPS clock bias, and ϵ_{p_i} is measurement noise. The GPS usually gives out a position estimate based on the pseudoranges, which is given in geodetic latitude, longitude and height above mean sea level. This relates to NED coordinates as illustrated in Figure 3.2.

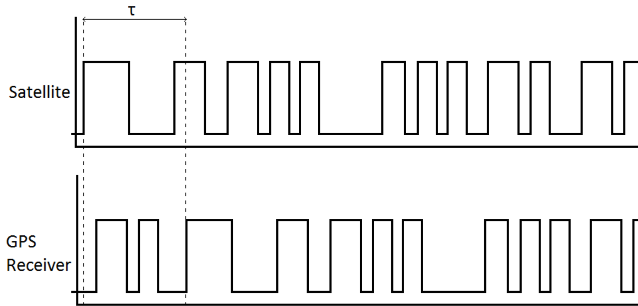


Figure 7.1: An illustration of how the time difference is measured

7.1.2 Distance Measurement Using Sound

A sound source combined with an audio receiver, can be used to estimate distance. By sending out a sound wave and determining the time, t_r it takes for the sound to return. The distance can be estimated as

$$D = \frac{t_r}{2} 340 \tag{7.2}$$

where 340 is approximately the speed of sound in air at 20°.

7.2 Problem Description

Determining the position of the quadcopter can be challenging. The IMU we are using, is quite inexpensive and thus has a lot of noise and uncertainties. The GPS module will only give a position estimate when it has enough satellites locks, and the height estimate can often be unreliable. The ultrasonic sensor will give accurate height estimates, but only when close to the ground.

We want to obtain a smooth and accurate position estimate, using any combination of the available sensors.

7.3 Design of Solution

7.3.1 Ultrasonic Sensor Height Estimate

The HC-SR04 is a simple ultrasonic sensor that can be used to measure distance by sending out a sound wave and detecting the return of said sound wave. An illustration of how this works can be seen in Figure 7.2. The time it takes for the sound to return, can then be used together with the knowledge of sound speed in air, to estimate the distance in the direction the sensor is pointing. By

assuming that the air temperature is around 15 degrees, the speed of sound can be approximated to 340 m/s. A distance estimate can then be found as

$$d_{us} = \frac{t_{end} - t_{start}}{2} 340m/s \tag{7.3}$$

where t_{start} is the time when a sound wave is sent out, and t_{end} is the time when the return of the sound wave is detected. d_{us} is the distance from the sensor to the closest object within the sensors measuring angle. The concept is illustrated in Figure 7.2. As the illustration shows, the distance measurement should equal the height above the ground, and thus should not need to be tilt compensated. To remove any unwanted high frequency noise, the measurement can be filtered using the low-pass filter Equation 4.15.

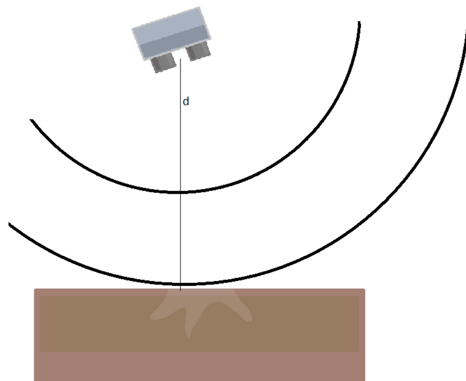


Figure 7.2: Ultrasonic Sensor Concept

7.3.2 GPS Position Estimate

The GPS module provides directly an estimate of the position in Geodetic coordinates. In Section 3.1.5 we find the transformation between Geodetic and NED as

$$\hat{\mathbf{p}}^n = \mathbf{R}_e^n(l, \mu)(\hat{\mathbf{p}}^e - \mathbf{p}_{ref}^e) \tag{7.4}$$

where $\mathbf{p}^e$ is the position estimate in ECEF coordinates and $\mathbf{p}_{ref}^e$ is the ECEF coordinates of the NED origin. The Geodetic to ECEF transformation is given

by

$$\hat{\mathbf{p}}^e = \begin{bmatrix} (N+h)\cos(\mu)\cos(l) \\ (N+h)\cos(\mu)\sin(l) \\ (\frac{r_p^2}{r_e^2}N+h)\sin(\mu) \end{bmatrix} \tag{7.5}$$

as described in Section 3.1.5.

7.3.3 Kalman Position Estimate

Simply using the GPS measurement found through Equation 7.4 to update the position of the quadcopter, will lead to large "jumps" in position, since the measurement is noisy and has a relatively slow update frequency of 5 Hz. This is not ideal when a position controller or a path following controller is active, as this can lead to undesirable movement. To reduce the magnitude of these jumps, and smooth out the position estimate, the Kalman filter, as described in Section 4.1.4 and 4.3.4, can be used. Combining the accelerometer force measurements, the ultrasonic sensor height estimate and the GPS position estimate, should result in better results.

The Kalman filter requires the system propagation functions and the resulting jacobian matrices. From Newton's second law, we have

$$\dot{\mathbf{p}}^n = \mathbf{v}^n \tag{7.6}$$

$$\dot{\mathbf{v}}^n = \frac{1}{m}\mathbf{f}^n \tag{7.7}$$

where we can use the accelerometer to approximate $\mathbf{f}^n$ as

$$\mathbf{f}^n \approx \mathbf{R}_b^n(\mathbf{\Theta})\mathbf{a}_{acc}^b + \mathbf{g}^n + \mathbf{R}_b^n(\mathbf{\Theta})(\mathbf{b}_{acc} + \mathbf{w}_{acc}) \tag{7.8}$$

where $\mathbf{b}_{acc}$ and $\mathbf{w}_{acc}$ is the bias and zero-mean noise of the accelerometer respectively. Since a small bias in the accelerometer measurement will inevitably be integrated and cause large errors in the position estimate, a system model which includes the bias can be used. Defining $\mathbf{x} = [\mathbf{p}^n,\ \mathbf{v}^n,\ \mathbf{b}_{acc}]^T$ yields

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(\mathbf{R}_b^n(\mathbf{\Theta})(\mathbf{a}_{acc}^b + \mathbf{b}_{acc}) + \mathbf{g}^n) \tag{7.9}$$

$$\begin{bmatrix} \mathbf{y}_{gps} \\ \mathbf{y}_{us} \end{bmatrix} = \mathbf{C}\mathbf{x} \tag{7.10}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7.11}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.12}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{7.13}$$

Even though the system equations are linear with respect to the system states, the Extended Kalman Filter, as described in sections 4.1.4 and 4.3.4, can be used to estimate the position, but with the A and C matrices defined as above.

7.4 Observations and Results

Figure 7.3 shows a plot of the ultrasonic distance estimate obtained while varying the pitch angle with the quadcopter placed between two tables such that the roll angle remained constant. During the experiment, the height above the ground remained constant.

A data series was taken from the GPS while walking back and forth in a straight line with the quadcopter. The geodetic coordinates was transformed into NED coordinates using Equation 7.4. A plot of the North coordinate vs time and the East coordinate vs time can be seen in Figure 7.4. Figure 7.5 shows a plot of the North-East GPS estimated position vs the actual position; independent of time.

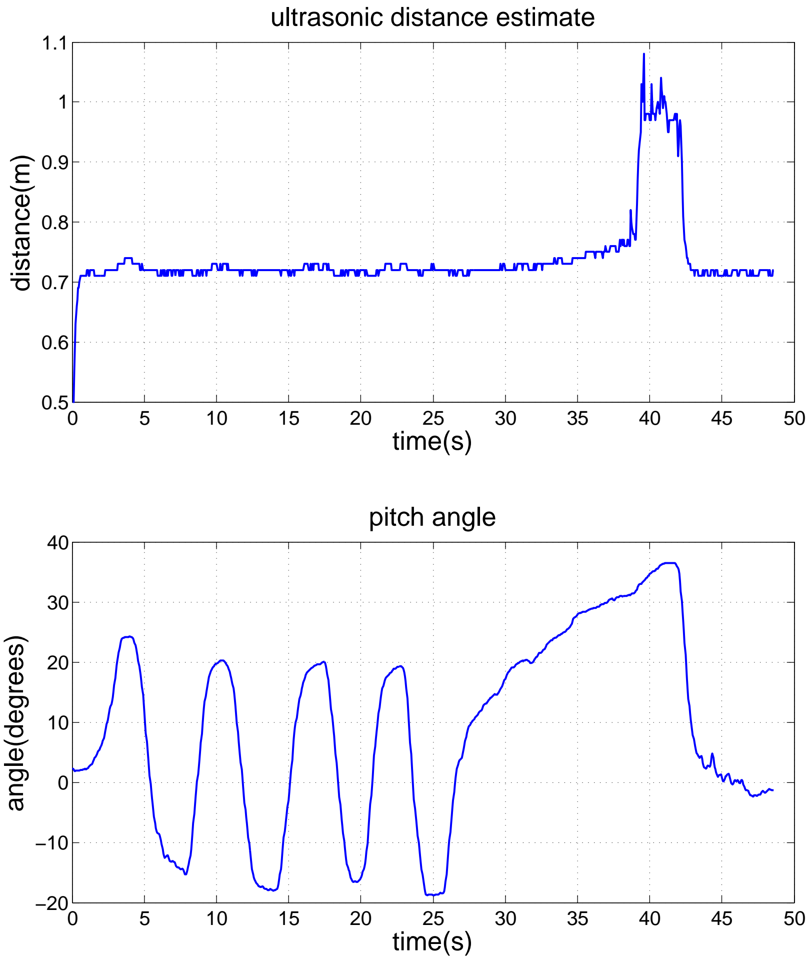
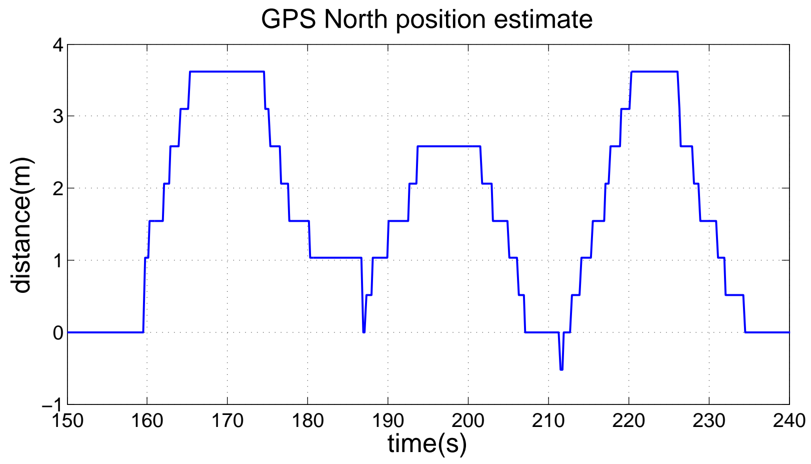


Figure 7.3: A comparison of the distance measurement vs the tilt compensated distance estimate



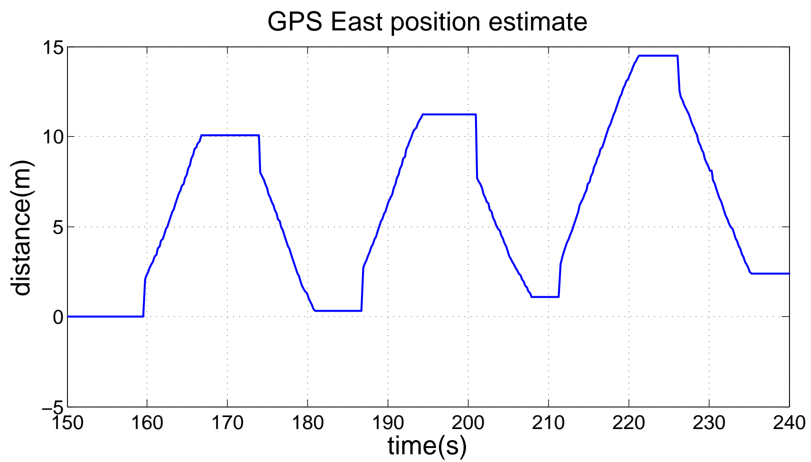


Figure 7.4: GPS North and East estimates vs time

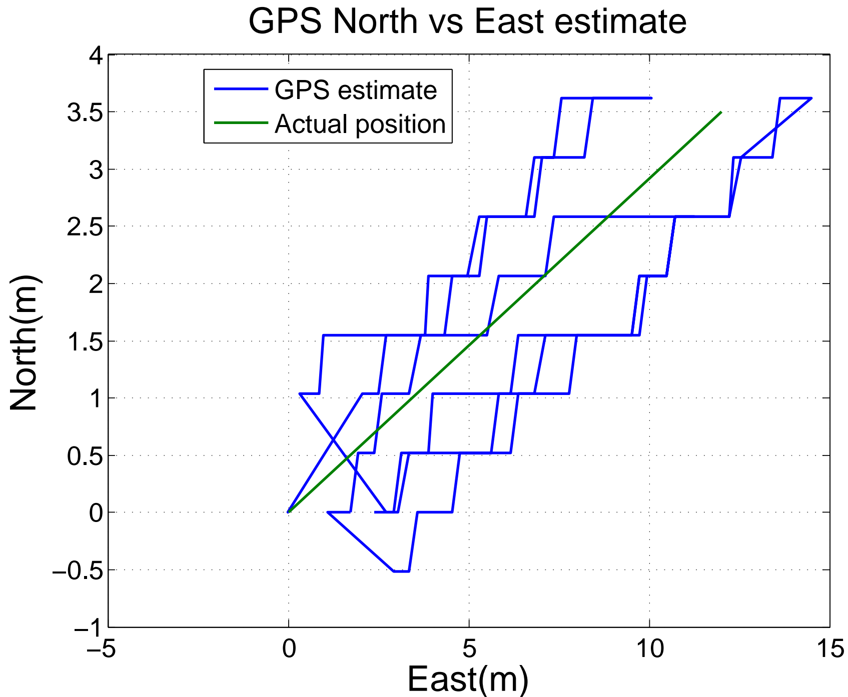


Figure 7.5: GPS estimated North-East position vs actual position

7.5 Discussion and Conclusion

As seen from the figure, the assumption that the distance measurement equals the height above the ground is valid for angles less then approximately 25°. From the specifications of the HC-SR04 sensor, the sensor can measure distances up to 120 cm. Hence, when the quadcopter is below 100 cm and the angle is less then 25°, the height estimate from the ultrasonic sensor should be used.

From Figure 7.5 we see that the position estimate obtained from GPS are noisy and non-smooth. If these estimates are used directly with a position control scheme, this would lead to the quadcopter "hopping" around the desired position.

7.6 Recommendations and Future Work

When flying indoors where the GPS signal is weak or non-existent, we recommend not trying to fly above 100 cm. To improve the height estimate, a differential pressure sensor can be used in addition to the ultrasonic sensor. This will also

allow flying above 100 cm without a GPS satellite lock.

8 Control of Attitude

8.1 Theory

The attitude describes the orientation of the quadcopter. In order to achieve a stable flight, the roll and pitch angles needs to be controlled. Several different control methods can be used to achieve stability, and in this chapter we will explore the various alternatives. The attitude control problem is essentially the same objective as the first quadcopters were trying to achieve. The first ever successful attitude and altitude controlled flight with a quadcopter were achieved by a French engineer, *Etienne Edmond Oemiche* in 1922[31].

8.1.1 The Linear Quadratic Regulator

Given the continuous linear system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{8.1}$$

$$\dot{\mathbf{y}} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{8.2}$$

A LQR solves the optimal control problem of minimizing the cost function

$$\mathbf{J} = \int_0^\infty (\mathbf{x}^T\mathbf{Q}_{lqr}\mathbf{x} + \mathbf{u}^\mathbf{T}\mathbf{R}_{lqr}\mathbf{u})dt \tag{8.3}$$

where $\mathbf{Q}_{lqr}$ and $\mathbf{R}_{lqr}$ are semi-positive definite weighting matrices, which enables us to define the relationship between the input and the response time. The optimal gain matrix $\mathbf{K}_{lqr}$ is given by

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} \tag{8.4}$$

where $\mathbf{P}$ is found by solving the continuous algebraic Riccati equation

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + Q = 0 \tag{8.5}$$

The optimal control gain is found as $\mathbf{u} = -\mathbf{K}_{lqr}\mathbf{e}$, where $e = x - x_{ref}$ is the error between the process variables and the reference signal. The closed loop dynamics using the LQR feedback is given by

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{B}\mathbf{K}_{lqr})\mathbf{e} \tag{8.6}$$

The linear quadratic regulator provides by definition an optimal state-feedback that minimizes the cost function (Equation 8.3), with that in mind the LQR is a clear candidate for implementing attitude control for the quadcopter. Optimal control refers to a class of methods that can be used to incorporate a control design which results in best possible behaviour with respect to the criterias for the system[31].

8.1.2 The Proportional-Integral-Derivative Controller

The Proportional-Integral-Derivative (PID) controller gives the simplest and yet the most efficient solution to various real-world problems. Both the transient and steady-state responses are taken care of with it's three-term functionality (proportional, integral and derivative). Since its invention the popularity of the PID controller has grown tremendously due to the advances in digital technology. The various automatic control systems offers a wide spectrum of choices for control schemes, even though more than 90% of industrial controllers are still implemented based around the PID algorithm [32], particularly at the lowest levels, as no other controllers match the simplicity, clear functionality, applicability, and ease of use offered by the PID controller. The PID control algorithm is given by

$$u(t) = K\Big(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\dot{e}(t)\Big) \tag{8.7}$$

where $e(t) = (y_{sp} - y)$ is the error between a measured process variable and a reference signal, often called a set point, T_i is the integral time and T_d is the derivative time. If a process variable cannot be measured it is common to design an observer that estimates the states by observing the measurable outputs.

Zieger-Nichols Method

In the 1940's, *John G Ziegler* and *Nathaniel B. Nichols* formulated two empirical methods for obtaining controller parameters. The Ziegler-Nichols closed loop tuning method is a heuristic method. Its performed by adjusting the integral and derivative gains to zero, the proportional gain K_p is then increased until it reaches the ultimate gain K_u at which the output of the control loop oscillates with a constant amplitude. K_u and the oscillation period T_u is used to set the Proportional, integral and derivative gains depending on the type of controller used [24];

Zieger-Nichols method			
Control type	K_p	K_i	K_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$1.2K_p/T_u$	-
PD	$0.8K_u$	-	$K_pT_u/8$
Classic PID	$0.6K_u$	$2K_p/T_u$	$K_pT_u/8$
Pessen Integral Rule	$0.7K_u$	$2.5K_p/T_u$	$K_pT_u/20$
Some overshoot	$0.33K_u$	$2K_p/T_u$	$K_pT_u/3$
No overshoot	$0.2K_u$	$2K_p/T_u$	$K_pT_u/3$

Table 8.1: Zieger-Nichols method

8.1.3 Nonlinear Control Theory

Nonlinear control theory is the area of control theory which deals with systems that are nonlinear, time-variant, or both[23]. In our case we know (based on the model derived during our fall project[31]) that the system does not explicitly depend on time, which means that our system is autonomous. There are some key points for why nonlinear control theory is being used[30];

- Few physical systems are truly linear
- The most common method to analyze and design controllers for systems is to start by linearizing around some point, which yields a linear model, and then use linear control techniques. In this process important information about the system can be lost.
- There are systems for which the nonlinearities are important and cannot be ignored, for these systems, nonlinear analysis and design techniques exist and can be used.

Nonlinearity arise naturally in numerous engineering and natural systems, including mechanical and biological systems, aerospace and automotive control, industrial process control and many others[33].

Numerous methods and approaches exist for the analysis and design of nonlinear control systems. Most of the theory and practice focus on feedback control; in feedback control a reference input is compared to the feedback signal, and a control input is produced. Nonlinear models may be classified into smooth and non-smooth nonlinear models. Non-smooth models are often associated with parasitic effects such as dry friction and actuator saturation, and sometimes these effects become significant, and may enter as a constraint in the design, or even require specific compensation techniques.

Alongside general tools and methods from system theory, such as equilibrium points and Lyapunov stability, a number of results and analysis methods apply specifically to feedback systems[33];

- Limit cycles (or sustained oscillations) are common in nonlinear feedback systems, and are usually not desired in control systems. The describing function method investigate the possibility of oscillations by approximating the response of nonlinear elements to sinusoidal inputs of given amplitude and frequency.
- Small gain theorem allows us to establish the input-output stability of a feedback system from properties found in its subsystems.

Control system design in general aims to satisfy certain performance objectives, such as stability, accurate input tracking, disturbance rejection, and robustness. Nonlinear systems are diverse by nature and necessarily they call for a variety of different design approaches.

One design viewpoint is to view the controlled system as an approximately linear one, or linearize the system by appropriate transformation, to which well-established linear control techniques may be applied, e.g. PID control, local linearization and feedback linearization.

Another design method is to approach the design problem directly using nonlinear tools, notably Lyapnov stability and Lyapunov functions, which are examples of robust nonlinear controls[33].

In lyapunov-based design, a stable system is synthesized by first choosing a candidate Lyapunov function V, and then selecting a state-feedback control law that renders the derivate of V negative. The Lyapunov redesign method provides the system with robustness to (bounded) uncertainly in the system dynamics. It starts with a stabilizing control law and Lyapunov function for the nominal system, and adds certain (non-smooth) terms to the control that ensure stability in the face of all admissible uncertainties[33]. While Lyapunov redesign is restricted systems that satisfy a matching condition so that the uncertainty terms enter the state equations at the same point as the control input, the basic approach has been extended to more general situations using recursive or backstepping methods.

Sliding mode control is another robust design approach, also called variable structure control. An appropriate manifold in the state space is fist located on which the system dynamics takes a simple and stable form. This manifold is called the sliding surface or the switching surface. The control law is designed to force trajectories to reach that manifold in finite time, and stay there thereafter. As the basic control law is discontinuous by design around the switching surface, unwanted chattering around that may result and often require some smoothing of the control law[33].

8.1.4 Alternative Controllers

There are many other techniques from control engineering that are applicable to the design of nonlinear systems. They are briefly described below, but not considered suitable for the design of an attitude controller[33].

- *Optimal Control:* The control objective is to minimize a pre-determined cost function. The basic solution tools are dynamic programming and variational methods. The available solutions for nonlinear problems are mostly numeric.

- *Model predictive control:* An approximation approach to optimal control, where the control objective is optimized on-line for a finite time horizon. Due to computational feasibility this method has recently found wide applicability, mainly in industrial process control.

- *Adaptive control:* A general approach to handle uncertainty and possible time variation of the controlled system model. Here the controller parameters are tuned on-line as part of the controller operation, using various estimation and learning techniques.

- *Neural network control:* A particular class of adaptive control systems, where the controller have the form of an artificial neural network.

- *Fuzzy logic control:* Here the controller implements a set of logical (or discrete) rules for synthesizing the control signal based on the observed outputs. Defuzzification and fuzzification procedures are used to obtain a smooth output law from discrete rules.

- *Hybrid dynamic controller:* A controller that exhibits both continuous and discrete dynamic behaviour - a system that can both flow and jump. In general, the state of a hybrid system is defined by the values of the continuous variables and a discrete control mode. The state changes either continuously, according to a flow condition, or discretely according to a control graph. Continuous flow is permitted as long as so-called invariants hold, while discrete transitions can occur as soon as given jump conditions are satisfied[22].

8.2 Problem Description

The attitude controller seeks to control the roll and pitch angles and to stabilize them at zero degrees.

8.3 Design of Solution

The attitude control problem is an unstable system with highly nonlinear dynamics. This is a system which belongs to the class of underactuated mechanical systems, having fewer control inputs than degrees of freedom. This renders the control task challenging when it comes to design, testing, evaluating and comparing the different classical and contemporary control techniques. We have chosen to look more closely at a few of them.

In general, the control problem consists of obtaining dynamic models of the system, and using these models to determine control laws or control strategies to achieve the desired system response and performance. The complexity of

the control algorithms and system model, as well as our objective to guarantee the stability and robustness is a challenging task in real situations for real-time systems.

8.3.1 The Linear Quadratic Regulator

The attitude control problem is a Multiple-Input-Multiple-Output (MIMO) system. When you wish to tune an LQR for a MIMO system you can change individual elements of the $\mathbf{Q}_{lqr}$ matrix. By simply increasing or decreasing the elements along the diagonal, we can change how aggressive the controller should be with respect to the separate process variables.

Implementing LQR as our regulator for attitude control poses some challenges. LQR assumes that all the states of the system are measurable, if not an observer that estimates the states by observing the measurable output is required. Furthermore the LQR requires an analytical model of the system, and in our case it is a nonlinear model, which leads the design of LQR and observer to require a linearized model as well as linear state space equations.

In Dynamical systems, linearization is useful for assessing the local stability of an equilibrium point in a system with nonlinear differential equations. Linearization is an effective method for approximating the output of a function near an equilibrium point, and are therefore very useful in linear control theory. Linear control algorithms are proven to be easier to understand and often easier to implement then nonlinear control algorithms, since many nonlinearities can be ignored or simplified. Various linearization techniques were explored during our fall project[31], as well as the deriving of the following linearized equations.

Linearized Kinematic Equations

From our fall project[31], we see that the system Kinematic model were found to be

$$\begin{bmatrix} \dot{\mathbf{P}}^n_{b/n} \\ \dot{\mathbf{\Theta}}_{nb} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^n_b(\mathbf{\Theta}_{nb}) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{T}(\mathbf{\Theta}_{nb}) \end{bmatrix} \begin{bmatrix} \mathbf{v}^b_{b/n} \\ \omega^b_{b/n} \end{bmatrix} \tag{8.8}$$

After linearization we got the following approximated matrices

$$\mathbf{R}_b^n(\mathbf{\Theta}_{nb} \approx \begin{bmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{bmatrix} \tag{8.9}$$

$$\mathbf{T}(\mathbf{\Theta}) \approx \begin{bmatrix} 1 & 0 & -\theta \\ 0 & 1 & -\phi \\ 0 & \phi & 1 \end{bmatrix} \tag{8.10}$$

$$\dot{\mathbf{P}}_{b/n}^n \approx \begin{bmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \psi & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u - \psi v + \theta w \\ \psi u + v - \phi w \\ -\theta u + \phi v + w \end{bmatrix} \approx \mathbf{v}_{b/n}^b \tag{8.11}$$

$$\dot{\mathbf{\Theta}}_{nb} \approx \begin{bmatrix} 1 & 0 & -\theta \\ 0 & 1 & -\phi \\ 0 & \phi & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} p - \theta r \\ q - \phi r \\ \phi q + r \end{bmatrix} \approx \omega_{b/n}^b \tag{8.12}$$

Linearized Rigid-body Kinetics

From our fall project[31], we see that the system kinetics model were found to be

$$\underbrace{\begin{bmatrix} m\mathbf{I}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{I}_g \end{bmatrix}}_{\mathbf{M}_{RB}^{CG}} \begin{bmatrix} \dot{\mathbf{v}}_{g/n}^b \\ \dot{\boldsymbol{\omega}}_{b/n}^b \end{bmatrix} + \underbrace{\begin{bmatrix} m\mathbf{S}(\boldsymbol{\omega}_{b/n}^b) & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & -\mathbf{S}(\mathbf{I}_g\boldsymbol{\omega}_{b/n}^b) \end{bmatrix}}_{\mathbf{C}_{RB}^{CG}} \begin{bmatrix} \mathbf{v}_{g/n}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_g^b \\ \mathbf{m}_g^b \end{bmatrix} \tag{8.13}$$

The kinetic equations can be approximated as

$$\dot{\mathbf{v}}_{g/n}^b \approx \frac{1}{m}\mathbf{f}_g^b \tag{8.14}$$

$$\dot{\boldsymbol{\omega}}_{b/n}^b \approx \mathbf{I}_g^{-1}\mathbf{m}_g^b \tag{8.15}$$

$$\mathbf{f}_g^b = \mathbf{F}_{gravity} + \mathbf{F}_{motor} \approx \begin{bmatrix} -\theta mg \\ \phi mg \\ mg \end{bmatrix} + \mathbf{F}_{motor} \tag{8.16}$$

Linearized State Space Model

Merging the results from the linearization of the kinetic and kinematic equations and defining $\mathbf{x} = ((\mathbf{p}_{b/n}^n)^T, (\mathbf{\Theta}_{nb})^T, (\mathbf{v}_{g/n}^b)^T, (\boldsymbol{\omega}_{b/n}^b)^T)^T$ yields the following linear state space model[31] :

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{8.17}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{\Upsilon} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{8.18}$$

$$\mathbf{\Upsilon} = \begin{bmatrix} 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8.19}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} \\ K_1 & K_1 & K_1 & K_1 \\ 0 & -K_2 & 0 & K_2 \\ K_2 & 0 & -K_2 & 0 \\ -K_3 & K_3 & -K_3 & K_3 \end{bmatrix} \tag{8.20}$$

Our LQR Attitude Controller

We define $x = (\phi, \theta, \psi, p, q, r)$ and assume full-state feedback, furthermore we have the following linearized subsystem as derived in our fall project[31]

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{8.21}$$

where

$$A = \begin{bmatrix} \mathbf{0}_{3x3} & \mathbf{I}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \tag{8.22}$$

$$B = \begin{bmatrix} \mathbf{0}_{3x1} & \mathbf{0}_{3x1} & \mathbf{0}_{3x1} & \mathbf{0}_{3x1} \\ 0 & -K_2 & 0 & K_2 \\ K_2 & 0 & -K_2 & 0 \\ -K_3 & K_3 & -K_3 & K_3 \end{bmatrix} \tag{8.23}$$

Following the Bryson's weighting rule, our initial design was to choose $\mathbf{Q}_{lqr}$

and $\mathbf{R}_{lqr}$ as

$$\mathbf{Q}_{lqr} = \begin{bmatrix} \frac{1}{x^2_{1,max}} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & \frac{1}{x^2_{6,max}} \end{bmatrix} \tag{8.24}$$

$$\mathbf{R}_{lqr} = \begin{bmatrix} \frac{1}{u^2_{1,max}} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & \frac{1}{u^2_{4,max}} \end{bmatrix} \tag{8.25}$$

where x_{max} and u_{max} are the maximum allowed or desired variations in states and inputs respectively. After some tuning in Matlab simulations, as derived in our fall project[31], we started with;

$$\mathbf{Q}_{lqr} = \text{diag}([800,\ 800,\ 10,\ 10,\ 10,\ 5]) \tag{8.26}$$

$$\mathbf{R}_{lqr} = \text{diag}([0.01,\ 0.01,\ 0.01,\ 0.01]) \tag{8.27}$$

8.3.2 The Proportional-Integral-Derivative Controller

Assuming that the quadcopter is fairly vibration free, its possible to make the quadcopter fly relatively stable using only the proportional gain without having any impressive behavior. If the proportional gain is too low the quadcopter will behave sluggish and use a long time to correct any errors. If the proportional gain coefficient is too high we will get overcorrections from errors (overshoot), and get high frequent oscillations and an unstable quadcopter.

The integral gain coefficient can increase the precision of the angular position, and hence remove steady state errors. The integral gain coefficient is especially useful with irregular wind, and ground effect (turbulence generated by the motors). If the integral gain value gets too high the quadcopter will begin to show signs of slow reaction and a decreased effect of the proportional gain as consequence, it will also start to oscillate with a low frequency.

The derivative gain coefficient is useful in order to prevent overshoot and will be of greater importance in the case of aerobatic flights. It is however possible to ignore the derivative gain value completely and the quadcopter will be able to fly. The derivative gain will change the force applied to correct a position error, when it sees a decrease or increase in the position error, this term will "soften" the movement and can help reduce vibrations.

Tuning Using Zieger-Nichols Method

We used the Zieger-Nichols method in order to get a good starting point for our tuning of the PID controller. We increased the proportional gain, and watched the quadcopter response go from sluggish until we got oscillations and recorded the gain value K_p to be 27, and the oscillation period T_u to be 0.45 seconds. T_u was found by counting the number of oscillations we had during a 10 second flight. From these values we calculated the gains using the classic PID (seen in table 8.1), this yielded $K_p = 22, K_i = 98, K_d = 1$.

8.3.3 Nonlinear Controller

In the previous explored controller options the control algorithm was based on a linearized version of the quadcopter model. The linearization will constraint the control to be valid for certain conditions, a nonlinear controller is suitable to overcome these challenges.

Our nonlinear algorithm is based on the backstepping method integrated with integral and adaptation schemes. It is called the *Adaptive Integral Backstepping controller* (AIBC)[26]. The recursive Lyapunov metodology in the backstepping method will ensure the system stability; the integral part will increase the system robustness against both model uncertainties as well as disturbances, and the adaptation law will help estimate the modelling errors caused by assumptions in simplifying the complexity of the quadcopter model.

This control design has been proposed since it is able to[26];

- Stabilize roll and pitch angles at zero degrees
- Ensure stability of the quadcopter using a nonlinear controller based on the Lyapunov criterion
- Estimate the translation matrix between the Euler angle rates and the inertial orientation angle rates using the adaptation technique

First, let us consider a second order system

$$\ddot{y} = a + bu + A \tag{8.28}$$

where y is the state variable, a represents the body gyroscopic effect, b is constant and represents the inertia, u is the control input, and A is an estimator for model errors as well as variations. We start out control design by defining the position tracking error and its derivative:

$$e_1 = y_d - y_x \tag{8.29}$$

$$\frac{de_1}{dt} = \dot{y}_d - \dot{y}_x \tag{8.30}$$

This definition specifies our control objective, where the recursive approach will drive the tracking error to zero. Lets consider the Lyapunov function V, which is positive definite around the desired position:

$$V = \frac{1}{2}e_1^2 \tag{8.31}$$

$$\dot{V} = e_1(\dot{y}_d - \dot{y}_x) \tag{8.32}$$

If the velocity $\dot{y}_x$ was our control input, we could easily choose $\dot{y}_x$ such that we had exponential convergence for the system, an example is $\dot{y}_x = \dot{y}_d + c_1 e_1$ where c_1 is a positive number that determines the convergence speed for the error[26]. Based on this we can see that the derivative of the Lyapunov function will be semi-negative definite and consequently the error will converge exponentially to zero.

$$\dot{V} = -c_1 e_1^2 \leq 0 \tag{8.33}$$

However, since the velocity $\dot{y}_x$ is only a system variable and not a control input, we can't specify its value as easily as we did in equation 8.32. There is however, possible to choose the desired behavior for $\dot{y}_x$ and consider it as our virtual control input. When performing backstepping design, the desirable dynamic behavior is called the stabilizing function[26]. Furthermore, the integral action will be included by choosing the virtual input as follows:

$$\dot{y}_{xd} = c_1 e_1 + \dot{y}_d + \lambda_1 \chi_1 \tag{8.34}$$

$$\chi_1 = \int_0^t e_1(\tau) d\tau \tag{8.35}$$

Where $\dot{y}_{xd}$ is the desired virtual input. We can ensure the convergence of the tracking error to steady state since we have included the integral action in the backstepping design, despite the presence of disturbances and model uncertainties.

Since $\dot{y}_x$ is not our control input, a dynamic error exists between it and its desired behavior $\dot{y}_{xd}$ [26]. We need to compensate this dynamic error by defining the velocity tracking error and its derivative as follows:

$$e_2 = \dot{y}_{xd} - \dot{y}_x \tag{8.36}$$

$$\frac{de_2}{dt} = c_1(\dot{y}_d - \dot{y}_x) + \ddot{y}_d + \lambda_1 e_1 - \ddot{y}_x \tag{8.37}$$

We can rewrite the derivative of the position error and get:

$$\frac{de_1}{dt} = -c_1 e_1 - \lambda_1 \chi_1 + e_2 \tag{8.38}$$

$$\Downarrow$$

$$\frac{de_2}{dt} = c_1(\dot{y}_d - \dot{y}_x) + \ddot{y}_d + \lambda_1 e_1 - a - bu - A \tag{8.39}$$

By adding the integral action, the augmented Lyapunov function will become [26]:

$$V = \frac{\lambda_2}{2}\chi_1^2 + \frac{1}{2}e_1^2 + \frac{1}{2}e_2^2 \tag{8.40}$$

We want to design an control input such that $\dot{V} \leq 0$

$$u = \frac{1}{b}\Big((1 - c_1^2 + \lambda_1)e_1 + (c_1 + c_2)e_2 - c_1\lambda_1\chi_1 + \ddot{y}_d - a - A\Big) \tag{8.41}$$

$$\Downarrow$$

$$\dot{V} = -c_1 e_1^2 - c_1 e_2^2 \leq 0 \tag{8.42}$$

A problem with this control input is that we do not know the real value of A, we need to replace it with the estimated value $\hat{A}$ and the adaptation law will be used:

$$u = \frac{1}{b}\Big((1 - c_1^2 + \lambda_1)e_1 + (c_1 + c_2)e_2 - c_1\lambda_1\chi_1 + \ddot{y}_d - a - \hat{A}\Big) \tag{8.43}$$

In order to complete the design we need to derive the update law for our estimated parameter $\hat{A}$. Let us define a parameter estimation error signal to be used in equation 8.43 as:

$$\tilde{A} = \hat{A} - A \tag{8.44}$$

If we substitute 8.44 into 8.39 then the derivative of the velocity tracking error is

$$\frac{de_2}{dt} = -c_2 e_2 - e_1 + \tilde{A} \tag{8.45}$$

In order to enhance the estimated parameter error we need to use the Lyapunov function a second time[26].

$$v = \frac{\lambda_1}{w}\chi_1^2 + \frac{1}{2}e_1^2 + \frac{1}{2}e_2^2 + \frac{1}{2\gamma}\tilde{A}^2 \tag{8.46}$$

$$\dot{V} = \lambda_1\chi_1\chi_1^T + e_1\dot{e}_1 + e_2 + \dot{e}_2 + \frac{\tilde{A}}{\gamma}\frac{d\tilde{A}}{dt} \tag{8.47}$$

$$\dot{V} = -c_1 e_1^2 - c_2 e_2^2 + \tilde{A}\Big(e_2 + \frac{1}{\gamma}\frac{d\tilde{A}}{dt}\Big) \tag{8.48}$$

Where γ is a positive design constant that determines the convergence speed of the estimate. In order to render the non-positivity of the Lyapunov derivative in the equation mentioned above, we need to choose the adaptation law for the estimated parameter $\hat{A}$ as:

$$\frac{d\tilde{A}}{dt} = -\gamma e_2 \tag{8.49}$$

$$\tilde{A} = -\gamma\chi_2 \tag{8.50}$$

$$\chi_2 = \int_0^t e_2(\tau)d\tau \tag{8.51}$$

$$\Downarrow$$

$$\dot{V} = -c_1e_1^2 - c_2e_2^2 \leq 0 \tag{8.52}$$

From this we can by using the adaptation law and the integral action derive the control input as following[26]:

$$u = \frac{1}{b}\Big((1 - c_1^2 + \lambda_1)e_1 + (c_1 + c_2)e_2 - c_1\lambda_1\chi_1 + \gamma\chi_2 + \ddot{y}_d - a\Big) \tag{8.53}$$

To get the system equations on the same form as the second order system Equation 8.28, we simply apply Newton's second law to obtain

$$\ddot{\mathbf{p}}^n = \dot{\mathbf{v}}^n \tag{8.54}$$

$$\dot{\mathbf{v}}^n = \frac{\mathbf{f}^n}{m} = \mathbf{R}(\mathbf{\Omega})\mathbf{F}_m + \mathbf{g} \tag{8.55}$$

where

$$\mathbf{F}_m = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^4 F_{m,i} \end{bmatrix} \tag{8.56}$$

Furthermore, by defining the inputs $u_1 = \sum_{i=1}^4 F_{m,i},\ u_2 = \tau_\psi,\ u_3 = \tau_\theta,\ u_4 = \tau_\psi$, this can be inserted into Equation 8.53.

8.4 Observations and Results

8.4.1 The Linear Quadratic Regulator

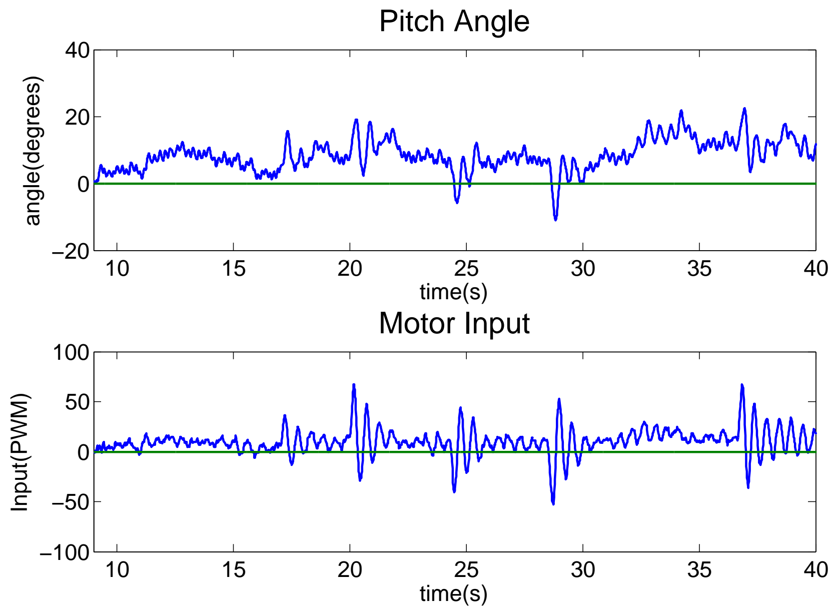


Figure 8.1: Angle and motor input using LQR

8.4.2 The Proportional-Integral-Derivative Controller

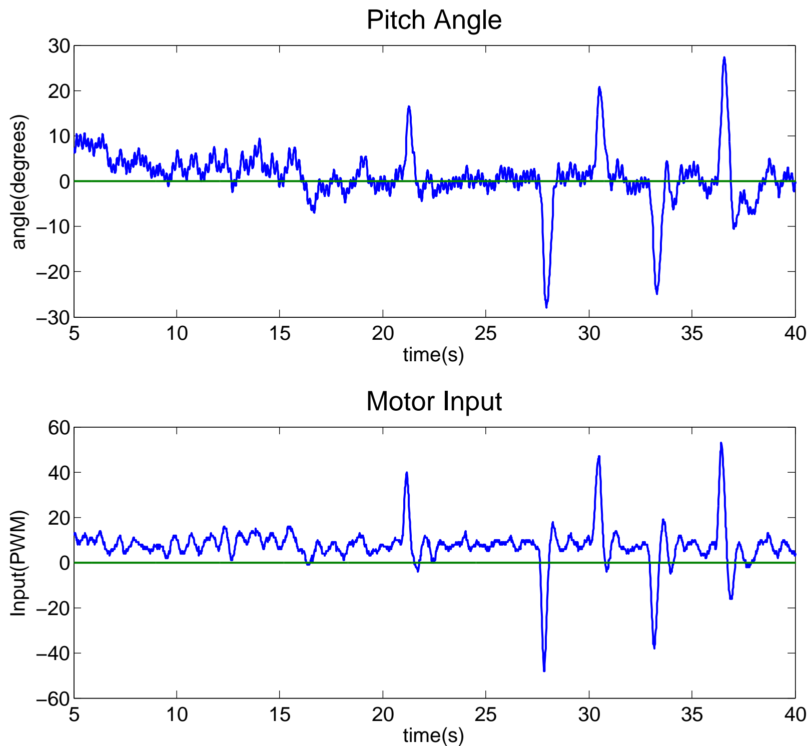


Figure 8.2: Angle and motor input using PID

8.5 Discussion and Conclusion

Throughout this chapter we have explored and discussed mainly three different solutions for the control problem: The Linear Quadratic Regulator, the Proportional-Integral-Derivative Controller and a Nonlinear Controller. The LQR and PID was implemented and tested, and the test results can be seen in section 8.4. We choose not to implement the nonlinear controller because of its complicated nature and the time it would take to implement.

8.5.1 The Linear Quadratic Regulator

After the Linear Quadratic Regulator were implemented, tuned and tested we got the results as can be seen in figure 8.1. We tested the LQR using prototype 1.1 (see section 11.3), where we mounted two propellers on one of the diagonals, and placed the quadcopter with the other diagonal placed on two tables at equal height. Since we have brushed down two of the sides from a square into a cylinder, the quadcopter controllers can be tested fairly realistically, while we can use a serial cable to acquire the data.

Figure 8.1 shows the angle that the controller was supposed to stabilize at 0 degrees, and the input generated on one of the motors based on the angle and angle rate, the other motor will have the same input but with the opposite sign. During this test we occasionally pulled on one of the sides of the quadcopter, in order to test the response with some interference.

The average angle is approximately 7 degrees, and thus this system failed to stabilize the quadcopter at 0 degrees. This is a result from a small differentiation in the weight, that the LQR can not overcome since there is no integral part. The quadcopter shows stability, and counteracts our disturbance with ease.

If we used this controller on the final prototype (section 11.4) we would have a stable quadcopter, but it would drift massively in position. It is in theory possible to control this using position estimation with GPS, but we believe that the quadcopter would drift before returning to the original position and then drift away again. This is not a desirable behavior, and therefore this controller is not suitable as out attitude regulation.

After tuning we ended up with the following Q_{lqr} and R_{lqr} matrices;

$$\mathbf{Q}_{lqr} = \text{diag}([100,\ 100,\ 10,\ 1,\ 1,\ 1]) \tag{8.57}$$

$$\mathbf{R}_{lqr} = \text{diag}([0.01,\ 0.01,\ 0.01,\ 0.01]) \tag{8.58}$$

8.5.2 The Proportional-Integral-Derivative Controller

After the Proportional-Integral-Derivative controller was implemented, tuned and tested we got the results as can be seen in figure 8.2. We tested the controller in the same fashion as the LQR, using prototype 1.1 (see section 11.3), with two motors attached with propellers, while the quadcopter itself were resting across two tables.

Figure 8.2 shows the angle that the PID controller was supposed to stabilize at 0 degrees, and the input generated on one of the motors, based on the angle, angle rate and the integral of the angle. Furthermore we included the same disturbance as with the LQR, where we occasionally pulled with force on one of the sides.

From the results we can see that the integral part of the PID controller helps stabilize the angle around 0 degrees after 15 seconds, and manages to hold the stabilized area at 0 degrees even with the disturbance. The response that the PID controller performed during our disturbance was significantly better than as with the LQR. There is practically no overshoot when it recovers from angles above ±20 degrees and it is stable once again around 0 degrees after just one second.

This control system perform better in every way compared to the LQR, and thus we choose to use the PID controller on prototype 2. Our tests with four propellers and free flight were successful, and the attitude have been stable without exception for more than 5 hours of flight time. However, we were unable to provide any proper plots from this, due to the fact that the radio link have limited amount of messages per second (see section 12.4.3), and there are no uncomplicated way to use a serial cable during flight. A possible solution would be to use a bluetooth antenna, which have a much higher bandwidth than the radio in order to provide the data required for the plots.

The tuning process was a time consuming and thorough process, but it resulted in a satisfactory controller. The performance of the attitude controller was achieved using the control parameters listed in table 8.2. These values are quite different than the suggestions using Zieger-Nichols method, and is another example that there is no certain method for calculating control gains.

Control gain	Zieger-Nichols value	Final value
P	22	70
I	98	10
D	1	15

Table 8.2: PID control gains

8.6 Recommendation and Future Work

In order to ensure proper attitude control it is necessary for the IMU to be calibrated and placed in the center of the quadcopter. If the accelerometer is placed somewhere else than the center, we need to calculate the acceleration and angle velocity using a rotation matrix, which will increase the iteration time and could thus cause poor performance in addition to increased complexity. The accelerometer also need to be calibrated for each time the IMU is moved, due to the difficuly of placing the accelerometer perfectly aligned with the axes of the quadcopter, and even an error of one degree will cause the quadcopter to drift in position. This can also be compensated in flight time using the Windows application, and would be the first place to look if the quadcopter drift.

In the future we would explore the nonlinear controller, even tough it is highly complex and complicated to implement. Simulation results provided in other articles [26] indicates that a proper nonlinear controller have a better performance for the attitude control of a quadcopter, than a tuned PID or LQR. We would estimate that the cycle time would be sligtly higher for the nonlinear controller than for the PID and LQR, which could potentially decrease the efficiency of the nonlinear controller.

9 Control of Altitude

9.1 Theory

The design of an altitude controller was simplified when we separated the attitude and altitude control problems into two separate controller designs. The altitude control problem is unstable; with non-linear dynamics, the force provided by each of the propellers does not share linear dynamics with the thrust, and in addition the force generated depends on the altitude above ground, known as the ground effect.

In order to measure the altitude, we use a range sensor for low heights and GPS position data for altitudes above a meter. The range sensor provides reliable data for ranges up to 1.2 meters, however, any roll and pitch motion performed by the quadcopter will increase the distance the sensor measures towards the ground, and will hence reduce the working area for the range sensor. The GPS provides low precision altitude data, which can be in the range of $\pm$ 2 meters, however, our extended Kalman filter will increase the precision substantially.

9.2 Problem Description

Design and implement an altitude controller, able of stabilizing the altitude to a desired height above ground.

9.3 Design of Solution

Lets look at the dynamics of the system. By defining $\mathbf{p}^n = [N, E, D]$ and applying Newton's second law we get the system model

$$\ddot{\mathbf{p}}^n = \frac{1}{m}\mathbf{f}^b \tag{9.1}$$

where

$$\begin{aligned} \mathbf{f}^b &= \mathbf{g} - \mathbf{R}_b^n(\Omega)\mathbf{F}_m &(9.2)\\ &= \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \mathbf{R}_b^n(\Omega) \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^4 F_{m,i} \end{bmatrix} &(9.3) \end{aligned}$$

We can simplify $\mathbf{F}_m$ by ignoring the difference in motor forces created by the attitude controller, such that $\sum_{i=1}^4 F_{m,i} = 4F_{m,avg}$, where $F_{m,avg}$ represents the average input force to each motor. In the design of an altitude controller this is also a quite realistic simplification, since an increase in motor force due to the

attitude controller on one motor will result in an equal decrease in the opposite motor.

$$\ddot{\mathbf{p}}^n = \frac{1}{m}\begin{bmatrix} -(\sin(\psi)\sin(\phi)+\cos(\psi)\cos(\phi)\sin(\theta))4F_{m,avg} \\ (\cos(\psi)\sin(\phi)-\sin(\theta)\sin(\psi)\cos(\phi))4F_{m,avg} \\ g-\cos(\theta)\cos(\phi)4F_{m,avg} \end{bmatrix} \tag{9.4}$$

From this we can extract that the acceleration along the z-axis can be described as $\ddot{Z} = \frac{g-\cos(\theta)\cos(\phi)4F_{m,avg}}{m}$. The altitude controller will not consider acceleration in the X and Y direction. Using this we can design a controller which will control $F_{m,avg}$, and in turn be able to control the altitude of the quadcopter. A control scheme for this system can be seen in figure 9.1.

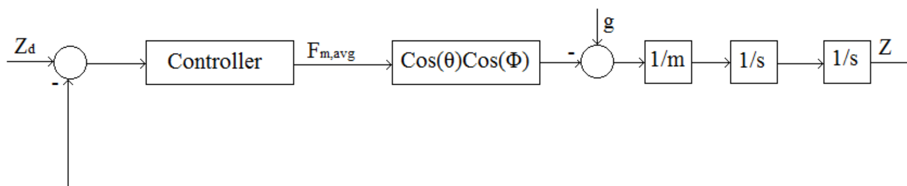


Figure 9.1: Control system design for the altitude

This is a non-linear system, where any roll and pitch motion will reduce the thrust in the Down(NED) direction. A simple solution is to define

$$F_{m,avg} = m\frac{g+F_p}{4} \tag{9.5}$$

where

$$F_p = k_{p,z}\tilde{Z} + k_{i,z}\int\tilde{Z} + k_{d,z}\dot{\tilde{Z}} \tag{9.6}$$

is a PID controller that stabilizes the altitude at $\tilde{Z} = Z_d - Z$, where Z_d is the desired height. It is difficult or impossible to use tuning methods like Zieger-Nichols method for the altitude PID controller, since a practical experiment where we increase the proportional gain until we have oscillations can be dangerous. The PID controller will therefore be tuned slowly in flight. The controller is explained graphically for this system in figure 9.2

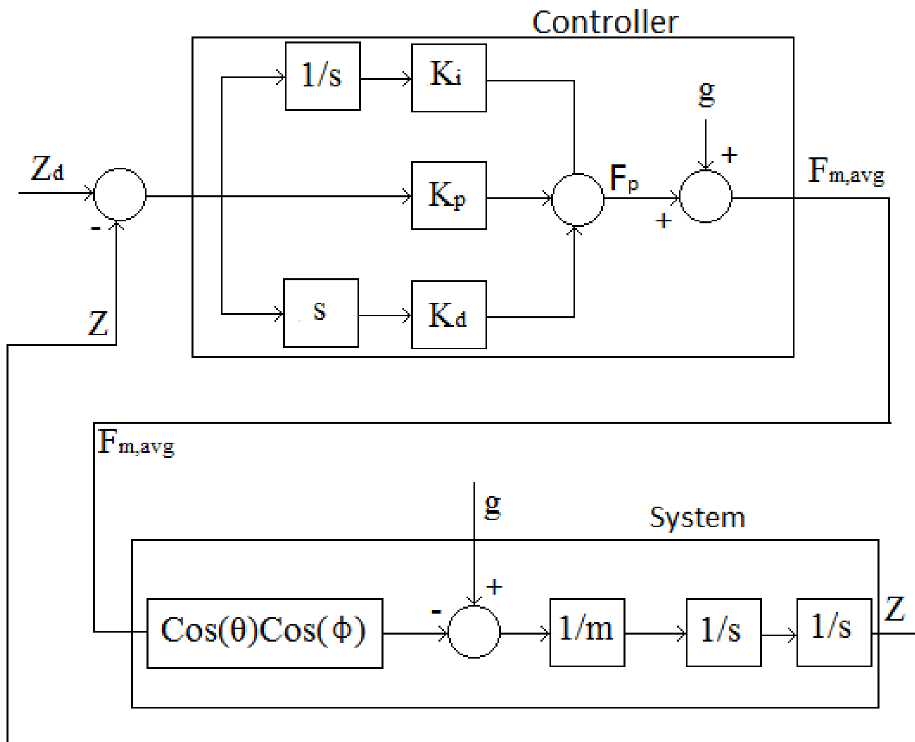


Figure 9.2: PID control system design for the altitude

9.4 Observation and Results

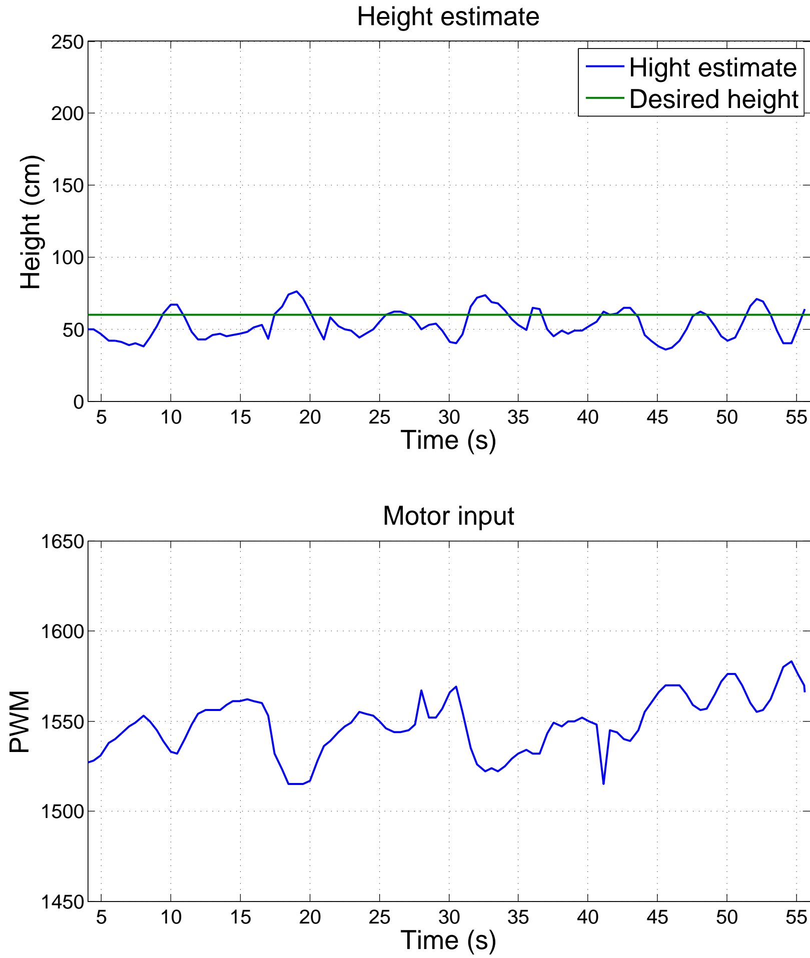


Figure 9.3: Altitude controller

9.5 Discussion and Conclusion

The objective of the altitude controller is to design and implement a controller with stable behavior. In figure 9.3 we see a plot of the estimated height and corresponding motor input generated by the altitude PID controller during a 50 second flight.

The altitude control problem is a highly nonlinear control problem, with challenging behavior. Wind, the surrounding environment and especially the quadcopter itself creates turbulence in the air, which will yield different thrust generated by the propellers even with the same control input. In addition, for small heights (approximately less then one meter), the thrust generated depends on the distance to the ground, where we have greater force and more disturbances the closer we are to the ground.

The altitude controller managed to hold the estimated height within a margin of ±25 cm around the targeted height of 60 cm. Considering the nonlinear terms of the system as well as the unpredictable behavior of the airflow, we consider this altitude controller to be satisfactory. The quadcopter will be able to keep its altitude in a area less then ±25 cm of its targeted height, and the drift within this area is slow.

In our design of the altitude controller, we assumed that the roll and pitch angles are small. A possible improvement to the controller can be made by considering the fact that the actual force applied in the Down(NED) direction is given by

$$F_{down} = \frac{1}{\cos(\phi)\cos(\theta)} F_{m,avg} \tag{9.7}$$

9.6 Recommendation and Future Work

In order to increase the performance of the quadcopter we recommend additional range sensors. The range sensors have various working areas, and by including several range sensors with different working areas, we can ensure proper height estimations for a broader span of altitudes.

A potential improvement to the controller itself would be to consider the equations for the thrust generated by the propellers, as can be seen in section 3.2.2. The thrust generated by each propeller are quadratic to the PWM input, and by including this in the model we may be able to produce better motor inputs, and gain a more stable altitude controller.

10 Control of Motion

10.1 Theory

Controlling the quadcopter position along the three axes in the NED coordinate system is no easy feat. The attitude and the position of the quadcopter must be estimated within reasonable margins, and the attitude must be controlled. Given a stable attitude controller, the quadcopter can then translate along the x- and y- axes by changing the desired attitude angles.

10.1.1 Dynamic Positioning

Dynamic positioning(DP) is a term most widely used together with vessel control for the oil industry as a computer controlled system that automatically maintains the vessel's position and heading even in heavy waves and wind. By combining sensor data, position and attitude can be estimated and used together with a mathematical model of the vessel to determine and apply appropriate thrust in order to maintain the desired position. DP may either be absolute in that the position is locked to a fixed point over the ground, or relative to a moving object.

10.1.2 Waypoints, Paths And Trajectories

The most common way of specifying a desired route of an autonomous quadcopter, is done using Cartesian coordinates (x_k, y_k, z_k) for k=1,...,n. Additionally, other way-point properties such as desired heading can be specified as (ψ_k) for k=1,...,n. This means that the quadcopter should pass through way-point (x_i, y_i, z_i) with heading (ψ_i). In practice, to obtain smoother transitions between way-points, a path can be generated. Path-following is independent of time, hence no restrictions are placed on the temporal propagation along the path. If time execution is critical, a desired trajectory $(x(t), y(t), z(t))$ of desired position in time can be generated. This trajectory can be created using reference models generated by low-pass filters, optimization methods or by simply simulating the quadcopter motion using an adequate model.

The transformation of such way-points to a feasible path or trajectory, is often a non-linear optimization problem for under-actuated crafts, but since the quadcopter can move freely in along all three axes with no limitations other than that of maximum velocity (and potential objects in the way), this can be done quite easily.

10.2 Problem Description

The quadcopter is under-actuated since it can not exert force along its own x- or y-axis. The position can be changed using different strategies by controlling the

attitude and the motor force along the body z-axis. In this section

- Guidance systems for calculating the changes in velocity required to follow paths and trajectories will be explored and discussed.
- To control the speed of the quadcopter along the desired velocity given by the guidance systems, a speed controller is devised.

In this section, position and attitude is assumed known.

10.3 Design of Solution

10.3.1 Guidance Systems, Trajectory and Path Generation

Depending on the objective, several different guidance systems may apply to the situation.

Line of Sight

If the objective is to follow a line between a reference point and a target, Line of Sight (LOS) guidance can be applied. A 2D illustration of the LOS guidance principle using lookahead-based steering is shown in Figure 10.1, where e is the distance from the quadcopter to the current line, s is the distance moved along the current line and α is the angle between $\tilde{\mathbf{p}}_k^n = \mathbf{p}_{k+1}^n - \mathbf{p}_k^n$ and $\tilde{\mathbf{p}}_{k,int}^n = \mathbf{p}^n - \mathbf{p}_k^n$. Furthermore the dot product between two Euclidean vectors are defined as

$$\mathbf{P} \cdot \mathbf{Q} = \cos(\theta)|\mathbf{P}||\mathbf{Q}| \tag{10.1}$$

thus we find α as

$$\alpha = \cos^{-1}\left(\frac{\tilde{\mathbf{p}}_k^n \cdot \tilde{\mathbf{p}}_{k,int}^n}{|\tilde{\mathbf{p}}_k^n||\tilde{\mathbf{p}}_{k,int}^n|}\right) \tag{10.2}$$

Since we now know α, we can find e and s using the trigonometric relations

$$s = |\mathbf{p}^n|\cos(\alpha) \tag{10.3}$$

$$e = |\mathbf{p}^n|\sin(\alpha) \tag{10.4}$$

The desired course vector is then defined as

$$\mathbf{p}_{LOS}^n = \mathbf{p}_k^n + (s + \Delta)(\mathbf{p}_k^n + \tilde{\mathbf{p}}_k^n) - \mathbf{p}^n \tag{10.5}$$

where Δ is a design parameter. The desired velocity vector is defined as

$$\mathbf{v}_{LOS}^n = -\beta\frac{\mathbf{p}_{LOS}^n}{|\mathbf{p}_{LOS}^n|} \tag{10.6}$$

where β is the desired speed towards the target. β can be dependent on the distance from the target such that the desired speed goes towards zero as the quadcopter gets closer to the target.

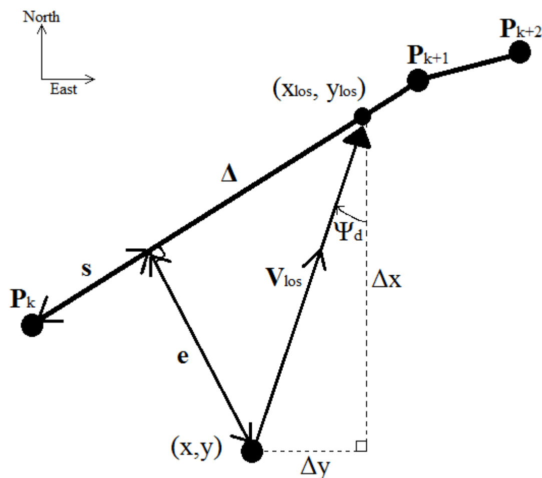


Figure 10.1: LOS guidance illustration in 2D

If the objective is to simply follow a target, a Pure Pursuit (PP) guidance scheme can be applied. The principle is shown in Figure 10.2, where the quadcopter aligns its velocity vector along the vector between the quadcopter and the target. The desired velocity vector is chosen as

$$\mathbf{v}^n_{PP} = -\beta \frac{\tilde{\mathbf{p}}^n}{|\tilde{\mathbf{p}}^n|} \tag{10.7}$$

where $\tilde{\mathbf{p}}^n = \mathbf{p}^n - \mathbf{p}^n_t$, $\mathbf{p}^n$ is the position of the quadcopter, $\mathbf{p}^n_t$ is the position of the target and β is the desired speed towards the target.

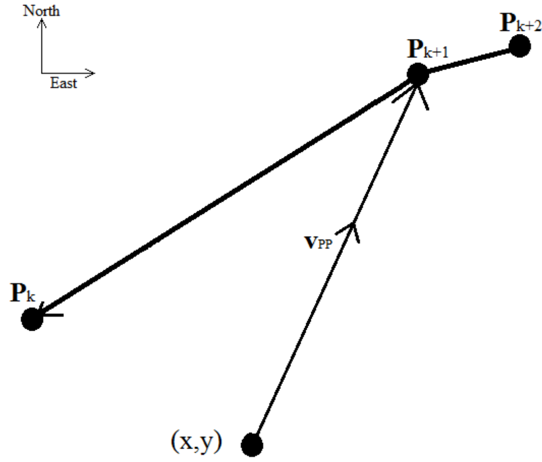


Figure 10.2: PP guidance illustration in 2D

If the objective is to intercept a moving target with known velocity $\mathbf{v}_t^n$, a Constant Bearing (CB) guidance scheme can be applied. The principle is shown in Figure 10.3, where the quadcopter aligns its velocity vector along the sum of the PP vector obtained using the equation above and the speed of the target. More specifically

$$\mathbf{v}_{CB}^n = \mathbf{v}_{LOS}^n + \mathbf{v}_t^n \tag{10.8}$$

This guidance law is commonly found in air-to-air or ground-to-air rockets.

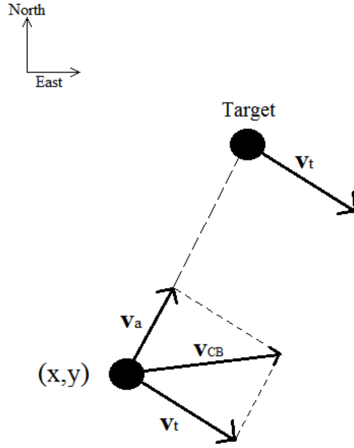


Figure 10.3: CB guidance illustration in 2D

Trajectory tracking can be useful if the objective is to follow a time dependent route or to smooth out linear reference signals. The desired position is then defined as $\mathbf{p}_d^n = (x(t), y(t), z(t))$. If the desired route is infeasible, a reference model trajectory can be obtained using a low-pass filter structure

$$\frac{\mathbf{p}_r^n}{\mathbf{p}_d^n} = \begin{bmatrix} h_{1,LP} \\ h_{2,LP} \\ h_{3,LP} \end{bmatrix} \tag{10.9}$$

where the choice of filter should reflect the system dynamics. The bandwidth of the filters should be chosen lower than the bandwidth of the motion control system, such that the results are feasible. The task of finding suitable filters is non-trivial, but as a starting point the mass-damper-spring system dynamics can be used, such that

$$h_{i,LP} \frac{\omega_{n_i}^2}{s^2 + 2\zeta\omega_{n_i} s + \omega_{n_i}^2} \tag{10.10}$$

where ζ_i are the relative damping ratios and ω_{n_i} are the natural frequencies for the filters.

Path following can be useful for following a route independent of time. Instead of just moving towards the next way-point, a path ensures that the quadcopter is

moving along predefined lines between way-points. The simplest path is the one created by straight lines between the way-points. The guidance law most suitable for following the path created by straight lines is the Line of Sight steering law proposed earlier, where k is incremented when either

$$\mathbf{p}_k^n + s(\mathbf{p}_{k+1}^n - \mathbf{p}_k^n) \geq |(\mathbf{p}_{k+1}^n - \mathbf{p}_k^n)| \tag{10.11}$$

or

$$\mathbf{p}_k^n + (s + \Delta)(\mathbf{p}_{k+1}^n - \mathbf{p}_k^n) \geq |(\mathbf{p}_{k+1}^n - \mathbf{p}_k^n)| \tag{10.12}$$

Other paths that can also be considered, are "straight lines and inscribed circles" paths or paths created by interpolation methods. A straight lines and inscribed circles path is created by defining the radius of each turn between two way-points, as shown in Figure 10.4. Creating paths using interpolation can be done in several ways, but the two most applicable interpolation methods one can use are cubic spline interpolation and the piece-wise cubic Hermite interpolating polynomial. More information about how to create these paths and how to follow them, can be found in the Handbook of Marine Craft Hydrodynamics and Motion Control [29].

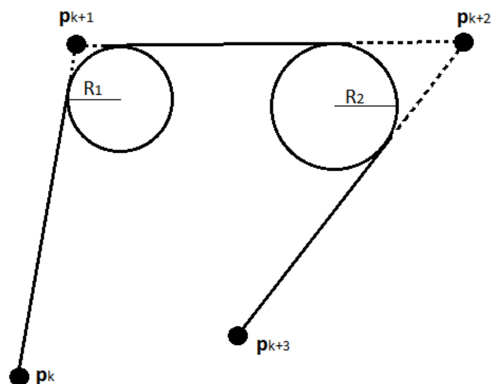


Figure 10.4: Straight Lines and Circles path

10.3.2 Control of Velocity in North-East-Down Coordinate System

For the purpose of executing the guidance schemes found above, a speed controller for the quadcopter should be designed. By defining $\mathbf{v}^n = [U,\ V,\ W]$ and applying Newton's second law we get the system model

$$\dot{\mathbf{v}}^n = \frac{1}{m}\mathbf{f}^b \tag{10.13}$$

where

$$\mathbf{f}^b \approx \mathbf{g} - \mathbf{R}_b^n(\mathbf{\Omega})\mathbf{F}_m - \mathbf{F}_d \tag{10.14}$$

$$= \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \mathbf{R}_b^n(\mathbf{\Omega}) \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^4 F_{m,i} \end{bmatrix} - \begin{bmatrix} c_1 \text{sign}(U)(U)^2 \\ c_2 \text{sign}(V)(V)^2 \\ c_3 \text{sign}(W)(W)^2 \end{bmatrix} \tag{10.15}$$

where $\mathbf{F}_d$ represents drag force and

$$\text{sign}(a) = \begin{cases} \text{-1} & \text{if } a < 0 \\ 0 & \text{if } a = 0 \\ 1 & \text{if } a > 0 \end{cases} \tag{10.16}$$

$\mathbf{F}_m$ can be simplified by ignoring the difference in motor forces created by the attitude controller, such that $\sum_{i=1}^4 F_{m,i} = 4F_{m,avg}$, where $F_{m,avg}$ represents the average input force to each motor. Furthermore, defining $\tilde{\mathbf{v}}^n = \mathbf{v}^n - \mathbf{v}_d^n$, where $\mathbf{v}_d^n$ is the desired speed vector obtained from the guidance law, we have

$$\dot{\tilde{v}}^n = \frac{1}{m} \begin{bmatrix} -(\sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta))4\mathbf{F}_{m,avg} - c_1 \text{sign}(U)(U)^2 \\ (\cos(\psi)\sin(\phi) - \sin(\theta)\sin(\psi)\cos(\phi))4\mathbf{F}_{m,avg} - c_2 \text{sign}(V)(V)^2 \\ g - \cos(\theta)\cos(\phi)4\mathbf{F}_{m,avg} - c_3 \text{sign}(W)(W)^2 \end{bmatrix} \tag{10.17}$$

To find inputs $\phi,\ \theta,\ \psi$ and $F_{m,avg}$ that stabilizes the system, we first define the positive definite Lyapunov functions

$$V_1 = \frac{1}{2}m\tilde{U}^2 \tag{10.18}$$

$$V_2 = \frac{1}{2}m\tilde{V}^2 \tag{10.19}$$

$$V_3 = \frac{1}{2}m\tilde{W}^2 \tag{10.20}$$

Assuming that the attitude controller perfectly controls $\phi,\ \theta$ and ψ, we can set these as we please. For simplicity, we set $\psi = 0$, as this yields less complex control

inputs. From the Lyapunov function, V_1, we find

$$\dot{V}_1 = m\tilde{U}\dot{\tilde{U}} \tag{10.21}$$
$$= \tilde{U}(-(\sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta))4\mathbf{F}_{m,avg} - c_1\text{sign}(U)(U)^2) \tag{10.22}$$

With $\psi = 0$, we find that $\theta = \sin^{-1}\left(\frac{k_{p,1}\tilde{U}}{\cos(\phi)4F_{m,avg}}\right)$ yields

$$\dot{V}_1 = -k_{p,1}\tilde{U}^2 - c_1\tilde{U}\text{sign}(U)(U)^2 \tag{10.23}$$

In practice however, a more linear control input is often preferred. Setting $\theta = \frac{k_{p,1}\tilde{U}}{4F_{m,avg}}$ yields

$$\dot{V}_1 = -\tilde{U}\cos(\phi)\sin\left(\frac{k_{p,1}\tilde{U}}{4F_{m,avg}}\right)4\mathbf{F}_{m,avg} - c_1\tilde{U}\text{sign}(U)(U)^2 \tag{10.24}$$

Ignoring air resistance, setting $-\phi_{max} < \phi < \phi_{max}$ and defining

$$D_1 = \{\tilde{U} \in \mathbb{R} : -\frac{4F_{m,avg}}{k_{p,1}}\theta_{max} < \tilde{U} < \frac{4F_{m,avg}}{k_{p,1}}\theta_{max}\} \tag{10.25}$$

where θ_{max} is a design parameter. We have that $V_1 \geq 0$ and $\dot{V}_1 \leq 0$, $\forall\tilde{U} \in D_1$ and $V_1 = \dot{V}_1 = 0 \Rightarrow \tilde{U} = 0$. Thus $\tilde{U} = 0$ is locally asymptotically stable. Including air resistance the speed will not stabilize at $\tilde{U} = 0$, but rather at a speed where the drag forces equals the forces created by the θ input. To overcome this, a feed forward term canceling the wind or an integrator term could be added trough the input.

From the second Lyapunov function, V_2, we find

$$\dot{V}_2 = m\tilde{V}\dot{\tilde{V}} \tag{10.26}$$
$$= \tilde{V}((\cos(\psi)\sin(\phi) - \sin(\theta)\sin(\psi)\cos(\phi))4\mathbf{F}_{m,avg} - c_2\text{sign}(V)(V)^2) \tag{10.27}$$

With $\psi = 0$, we find that $\phi = \frac{-k_{p,2}\tilde{V}}{4F_{m,avg}}$ yields

$$\tilde{V}(\sin\left(\frac{-k_{p,2}\tilde{V}}{4F_{m,avg}}\right)4\mathbf{F}_{m,avg} - c_2\text{sign}(V)(V)^2) \tag{10.28}$$

Ignoring air resistance and defining

$$D_2 = \{\tilde{V} \in \mathbb{R} : -\frac{4F_{m,avg}}{k_{p,2}}\phi_{max} < \tilde{V} < \frac{4F_{m,avg}}{k_{p,2}}\phi_{max}\} \tag{10.29}$$

where ϕ_{max} is a design parameter. We have that $V_2 \geq 0$ and $\dot{V}_2 \leq 0$, $\forall \tilde{V} \in D_2$ and $V_2 = \dot{V}_2 = 0 \Rightarrow \tilde{V} = 0$. Thus $\tilde{V} = 0$ is locally asymptotically stable. Including air resistance the speed will not stabilize at $\tilde{V} = 0$, but rather at a speed where the drag forces equals the forces created by the ϕ input. To overcome this, a feed forward term canceling the wind or an integrator term could be added trough the input.

From the last Lyapunov function, V_3, we find

$$\dot{V}_3 = m\tilde{W}\dot{\tilde{W}} \tag{10.30}$$

$$= \tilde{W}(g - \cos(\theta)\cos(\phi)4F_{m,avg} - c_3\text{sign}(W)(W)^2) \tag{10.31}$$

Defining $F_{ub} = \frac{g+k_{p,3}\tilde{W}}{4\cos(\theta)\cos(\phi)}$ such that

$$F_{m,avg} = \begin{cases} F_{m,max} & \text{if } F_{m,ub} \geq F_{m,max} \\ F_{m,ub} & \text{if } F_{m,min} < F_{m,ub} < F_{m,max} \\ F_{m,min} & \text{if } F_{m,ub} \leq F_{m,min} \end{cases} \tag{10.32}$$

Furthermore, adding constraints to ϕ and θ such that $-\phi_{max} < \phi < \phi_{max}$ and $-\theta_{max} < \theta < \theta_{max}$, making sure that $F_{m,min} < \frac{g}{4\cos(\theta_{max})\cos(\phi_{max})} < F_{m,max}$, yields

$$\dot{V}_3 = -k_{p,3}\tilde{W}^2 - c_3\tilde{W}\text{sign}(W)(W)^2 \tag{10.33}$$

Ignoring air resistance, we have $V_3 \geq 0$ and $\dot{V}_3 = -k_{p,3}\tilde{W}^2 \leq 0 \,\forall\, \tilde{W}$. Furthermore $V_3 = \dot{V}_3 = 0 \Rightarrow \tilde{W} = 0$ and $V_3 \to \infty \Rightarrow \tilde{W} \to \infty$, proving that $\tilde{W} = 0$ is Globally Exponentially Stable(GES). Including air resistance, the speed is not stable at $\tilde{W} = 0$, but will rather stabilize at an equilibrium where the force created by the input is equal to the drag force. To overcome this, a wind estimate could be fed forward or an integrator term could be added to the control force.

10.3.3 Structure of the Control System

Combining the guidance system and the speed controller yields the following block diagram which illustrates the overall control structure

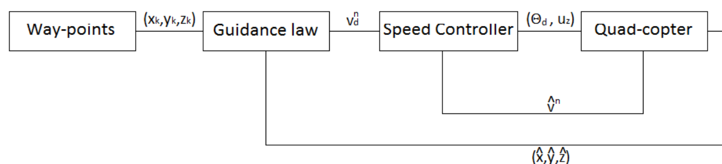


Figure 10.5: Control system

where $(\hat{x}, \hat{y}, \hat{z})$ is the estimated position, and $\hat{v}^n$ is the estimated speed vector. The attitude and height controller is "hidden" inside the quadcopter block.

10.4 Discussion

The control method we chose, to set $\psi = 0$ and use roll and pitch to translate in the desired directions, is just one of many solutions. However, it is the solution yielding the least complex roll and pitch input functions, which is why the method was chosen. As mentioned, the control inputs we found does not make $\tilde{\mathbf{v}}^n = \mathbf{0}_{3\times1}$ a stable equilibrium point because of air drag. However, in practice this will not be an issue. Figure 10.6 shows a 2D illustration on how the air drag will affect the quadcopter following a LOS guidance law with the speed controllers as proposed above. We found this behaviour to be satisfactory, and thus did not find it necessary to add integrator terms or feed forward terms to counteract the drag.

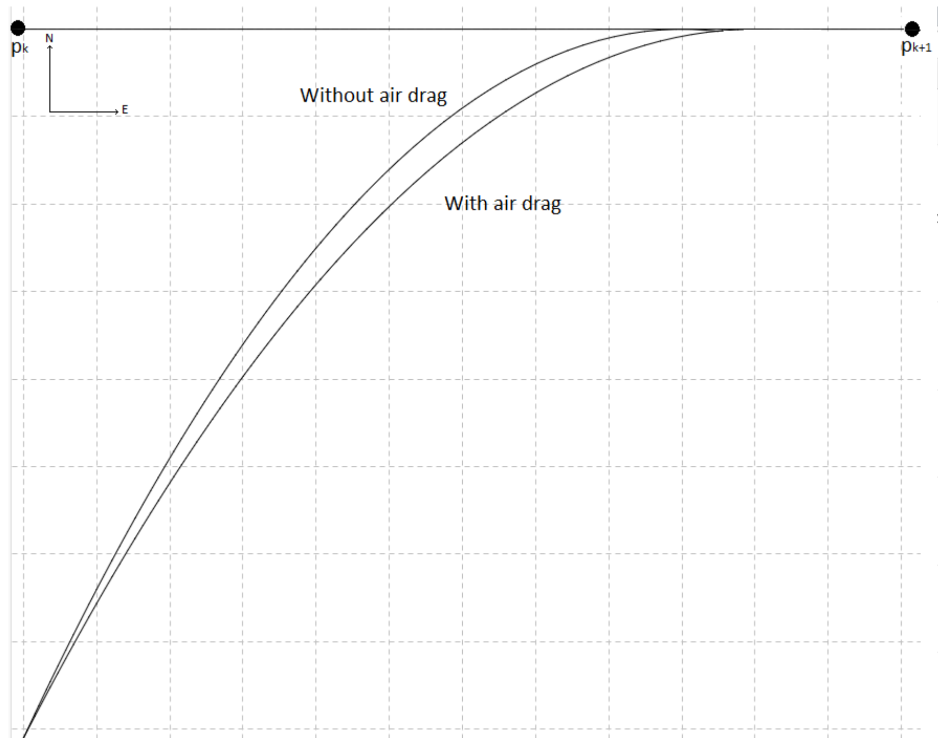


Figure 10.6: Drag illustration

10.5 Recommendation and Future Work

For practical implementation, we recommend that the saturation constraints on ϕ and θ are set low at first, along with small proportional gain constants. If air drag poses a problem, integrator terms or feed forward control could be added.

For future work, straight lines and circular paths could be created along with guidance laws for curved paths. One could also explore the possibility of using interpolation to create even smoother paths.

11 Prototype Development

11.1 Background

Our first prototype, version 1.0 was developed during our fall project and surved as a platform for testing and proof of concept. When building this prototype we had focus on simplicity, weight, price and robustness.

The frame is cut out from wood, this is to reduce weight as well as price. In order to reduce the weight even more we reduced the thickness of the wood between the house and the motors, as well as from the motors to the end. In retrospect we can conclude that the weight loss could not justify the reduction in robustness.

In order to make the quadcopter more robust and to be able to hit objects without breaking the propeller we positioned the motors such that a small piece of the frame is sticking outside the propeller. This will protect the propeller from objects (including the ground) when the quadcopter attitude is stable.

We placed the battery under the center of the quadcopter where it will help with the stability by lowering the center of gravity. This is done by using thin metal plates, which is fastened with screws to the frame, and coupled together with five metal rods. We choose the thin metal plates because of their low weight and robustness.

Directly above the battery house we built another house consisting of the same structure as the battery house, using thin metal plates and four metal rods. This house got three layers, the lower one holds the power distribution board, the Arduino Leonardo is placed in the middle where it may survive a moderate crash. On the top we placed the accelerometer, GPS as well as the radio antenna. See figure 11.1

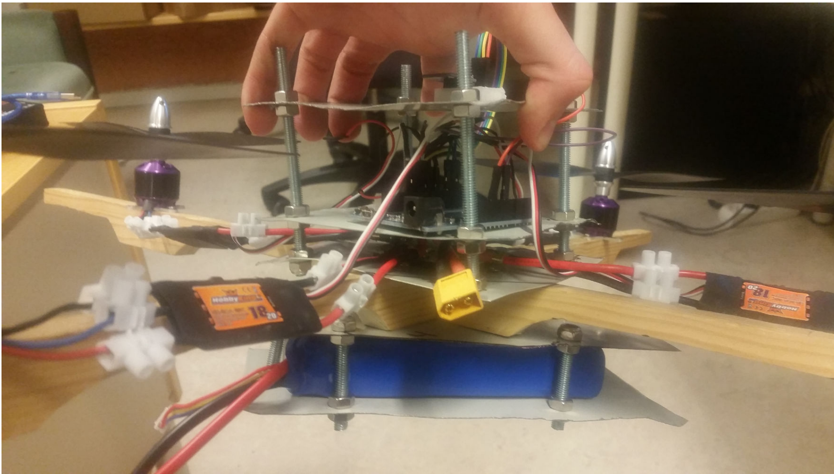


Figure 11.1: Prototype version 1 house

The whole prototype can be viewed in figure 11.2

Figure 11.2: Prototype version 1

11.2 Problem Description

This project requires more than just theoretical work and controller design, it needs a proper quadcopter prototype in order to produce results. Our intention

with this project is to end up with a working quadcopter, where all aspects have been designed locally. This includes everything from software, filters, control systems and also the quadcopter frame.

We want to design a quadcopter model which is easy to reproduce, and is cost effective. Our aim is to produce a model where we can easily exchange damaged parts, and is simple to mount. During development weight and robustness will be key areas of focus alongside price.

11.3 Development of Prototype Version 1.1

During testing of the software with our first prototype we encountered an unforeseen software bug that made the quadcopter crash. The result was that the frame had broken in two pieces and we were forced to develop a new prototype.

We based our new prototype on the design of our first quadcopter, although we made some knowledgeable improvements to the robustness of the aircraft. The main weaknesses were related to the robustness of the frame as well as the thin metal plates.

On the first prototype, the frame had reduced thickness at strategic places to reduce the weight. This however compromised the robustness of the frame to much, and we decided to not reduce the thickness on this model.

The thin metal plates provided a house structure with low weight and could withstand some pressure during crashes, which was proven when the electronic survived the crash that broke the frame. They did however bend easily and made it difficult to perfectly level the IMU. In addition they had sharp edges that could be dangerous for people when the quadcopter were in flight. For those reasons we choose to replace the metal plates with fiberglass. Fiberglass provides several good qualities for our purpose, with its low weight and high robustness.

Another weakness we discovered during our test flights were the lack of legs to support the weight of the quadcopter when its on the ground. Legs would let the quadcopter take off with a stable attitude, and could reduce the impact when landing. This was however not included in this prototype version due to the fact that it would be time consuming to make, and prototype version 2 were already in development. This model will only serve us for a short time.

We placed both of the houses in the same manner as with the first prototype. This were a good design that served its purpose, delivering handy space and protection for sensitive electronics.

In order to have a good system for testing the attitude controller we were developing, we wanted to use a two axis test sequence when tuning. This means that two adjacent propellers would run, while the remaining two propellers would be turned off. In order to create a environment that were as realistic and close to a real flight as possible we needed the quadcopter axes to be able to roll with

small friction and resistance. For this reason we used sandpaper to brush down the edges of the frame until they were round, and shaped like a cylinder.

Figure 11.3: Prototype version 1.1

11.4 Development of Prototype Version 2

The previous prototypes were temporarily models used for controller development as well as proof of concept. The previous models were developed and built in a few days, this prototype however, have been designed thoughtfully over a long period of time.

The prototype version 2 is designed, developed and build by a group of students here at NTNU under our guidance, where we were part of all aspects of the development as project managers. The students (Thomas Rostrup Andersen, Håkon Bråten, Alexander Vognild Burkow, Håkon Leithe, Nils Inge Rugsveen and kristian Stenrød) deserves some credit for the development, design and results that follows in this chapter[27].

This quadcopter are, unlike the previous versions, designed in software and printed on the institutes 3D-printer. This provides flexibility and endless possibilities, while it is at the same time both cheap and robust. It is trivial to produce several identical quadcopters and to replace defect parts.

11.4.1 Design Strategy

By placing all of the propellers equidistant from the center of mass and from each other, we ensure that the force provided by each propeller is equal. This is the

same design principle used in the previous models, and makes the control system for the attitude more simple and not unnecessary complicated. In addition we require that the IMU is as close to the center of mass as possible, which will reduce mathematical operations required in run time.

Based on these criterias it became natural for us and the group to have a central body with equal sized arms attached, where the electronic components could be placed. Furthermore the battery was placed under the center of body as with the previous models due to its weight, this is to optimize the maneuverability of the aircraft.

It is desirable to protect the electronic equipment from both weather as well as impact, the solution were to hide the electronic behind different plastic structures. The arms used to attache motors to the body, are made cylindrical such that the electronics could be hidden within the arms itself. The center of the aircraft consists of a solution to attach the arms, as well as a box beneath the center protecting the battery, and a box above the center protecting the Arduino and other sensitive equipment.

One of the properties we wanted to introduce to the other models were some sort of landing leg that could reduce the shock of impact when landing, as well as to keep the aircraft level on the ground. This feature have been introduced into this prototype by attaching landing legs on all four arms leading to the propellers from the body.

Low weight is one of the key focus areas when designing any aircraft, this is on order to reduce the energy requirements and to increase maneuverability. The greatest weight impact (except for the battery) comes from the amount of plastic used to create the structure. We focused on using the least amount of plastic, while at the same time keeping the robustness of the aircraft at an acceptable level.

In order to make it easy to exchange electronic parts, and have the opportunity to open the quadcopter, it was decided to not use any glue or other permanent fastening methods. All the different plastic components, with the exception of the legs (which are designed to be threaded onto the arms), are fastened using screws and nuts.

11.4.2 Robustness Calculations

In order to achieve a robust quadcopter that omit frequent repairs, we have conducted some control calculations. The calculations control the choice of printer material as well as the design of the different structures in order to ensure that they can handle the force and stress that will be inflicted during flight.

In order to conduct these calculations there have been made some presumptions and simplifications, and for that reason the results are conservative and with a good margin for error. The simplification is to consider the different arms

from one propeller to the other propeller to be one continuous beam. The total force and stress are applied as a point load at the junction of the arms, and spread equally to each arm.

The arms have a octagonal cross-sectional profile, with an internal reinforcing web. All parts have a thickness of 2mm. Anther simplification that have been made is to assume that the octagonal cross-section profile are presumed as a circular cross-section profile. We have calculated five different load scenarios:

- The weight of the quadcopter distributed over the four arms, with only the exterior circular section of the arms to support the weight.
- Upwards acceleration with only the exterior circular section of the arms to support the force provided by the propellers.
- Upwards acceleration with only the internal reinforcing web in the arms to support the force provided by the propellers.
- Upwards acceleration with both exterior circular section and internal reinforcing web in the arms to support the force provided by the propellers.
- Doubled weight and upwards acceleration with both exterior circular section and internal reinforcing web in the arms to support the force provided by the propellers.

These different load scenarios were picked based on previous experience in the "experts in teams" group. As a basic rule, the different parts are tested separately if the cross-sectional profile consists of several elements. The acceleration used in this calculations is scaled to a factor of two compared to the original load.

The calculations can be viewed in detail in [27], and the results are rendered below in table 11.1

Load scenario	Structure part	Result(σ_{uk})
Normal weight, on ground	Exterior circular section	$1,04 N/mm^2$
Normal weight, Upwards acc.	Exterior circular section	$2,08 N/mm^2$
Normal weight, Upwards acc.	Internal reinforcing web	$10,89 N/mm^2$
Normal weight, Upwards acc.	Exterior circular section and internal reinforcing web	$1,75 N/mm^2$
Double weight, Upwards acc.	Exterior circular section and internal reinforcing web	$6,98 N/mm^2$
	Exterior circular section	$8,31 N/mm^2$
	Internal reinforcing web	$43,6 N/mm^2$

Table 11.1: Results from load calculations

From these results we can see the tensile stresses σ_{uk} that will occur underneath the arms of the quadcopter in the different load scenarios. Based on the thickness of our structure and the material used (ABS-plus) there have been calculated how much stress the material can be seduced to before it starts deforming or fracturing [27], the results are listed below in table 11.2:

Capacity	value
Deformation	$6,15N/mm^2$
Fracture	$25,4N/mm^2$

Table 11.2: Capacity for ABS-plus

Using these values we can see which of the load scenarios that are critical. If we get stress above $6,15N/mm^2$, we might risk that the ABS-material becomes elastic and that the arms can bend. If the load stress becomes greater than $25,4N/mm^2$, the ABS-material will get so deformed that we might experience fractions in the structure.

By comparing the calculated tensile stress in table 11.1, with the capacity for the ABS-plus listed in table 11.2, it is possible to determine how robust the quadcopter really is. The situations where only the internal reinforcing web in the arms received the stress, as well as the scenario with double weight and upwards acceleration will cause tensile stress above the capacity of the material. These situations are not likely scenarios, and its important to note that only the case where we have double weight and upwards acceleration and all the stress is carried by the internal reinforcing web in the arms will cause fractions in the material. This is a highly unlikely scenario to encounter.

This structure with the chosen material ABS-plus, will endure any normal flight stress that the quadcopter may experience. In the scenarios where we experience large and unlikely high stress we might see that the arms bends, but we will not experience fractures in the construction from any forces resulting from the quadcopters own weight and acceleration.

11.4.3 Model and Description of Parts

The model were based on an existing quadcopter model named *"Tubular Crossfire 2 quadcopter"* and modified to fit our specifications and desires. The model composes of six different components, in total 12 parts. All of these components can be 3D-printed and fasten with screws and nuts (ISO-standard M3). The procedure for mounting the quadcopter can be viewed in the appendix chapter A.

Top Frame

Figure 11.4 is a graphical representation of the top frame for the center body on the aircraft. The component is the upper part that is used to fasten the arms, and also serves as the part you fasten the electrical components and the technical box. The big holes in the center is used for wires connected with the ESC as well as power cables for the Arduino. The top frame is connected with the bottom frame using 8 M3 screws, where two screws passes through each arm.

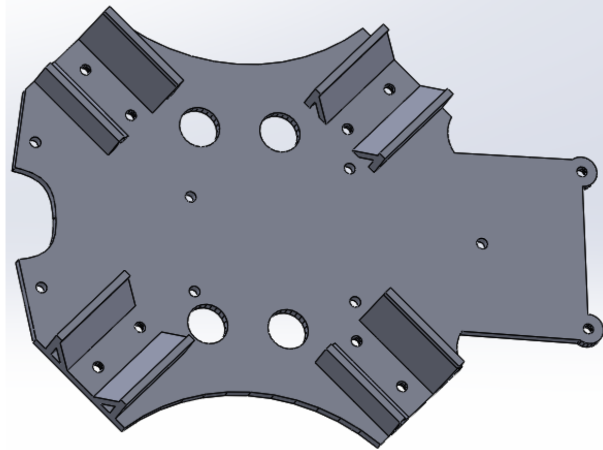


Figure 11.4: Top frame

Bottom Frame

The model of the bottom frame used in the quadcopter can be seen in figure 11.5. This component serves as the lower part of the fastening mechanism for the arms, as well as the part the battery box is attached to. It consist of the same 8 holes used to attach the bottom frame to the top frame, and also comes with holes to fasten the battery box. The square opening is used for the power cable from the battery towards the power distribution board, and is placed right by the opening on the battery box. The reminding holes improves the ventilation, and this reduces the temperature of the components resting both between the frames as well as the components within the technical box.

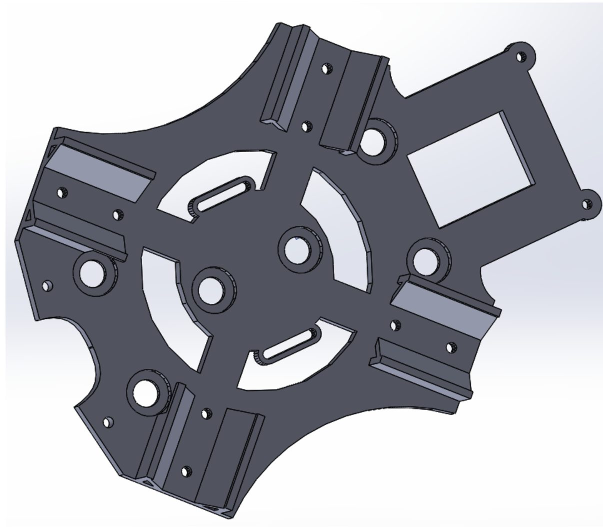


Figure 11.5: Bottom Frame

Technical Box

Figure 11.6 shows the model for the technical box. The technical box is designed to be placed over the electronic components used in the quadcopter, in order to protect against crash as well as to provide basic protection against weather. The box is designed with a small hole for the radio antenna, and also a small shelf where the radio component can be placed, as can be seen in figure 11.7. The box is fastened to the main body using three screws and nuts.

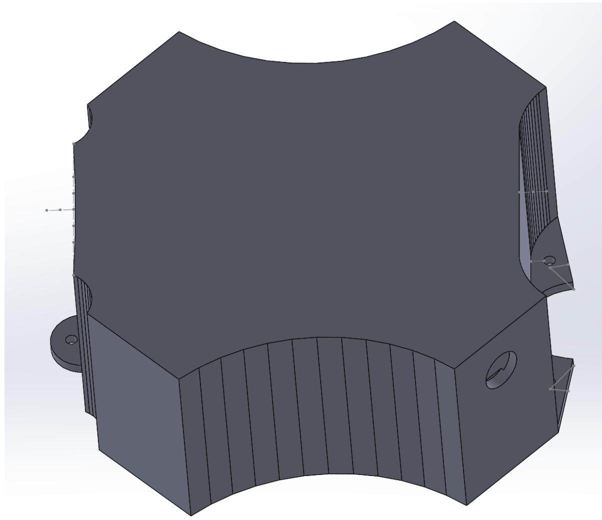


Figure 11.6: Technical box top view

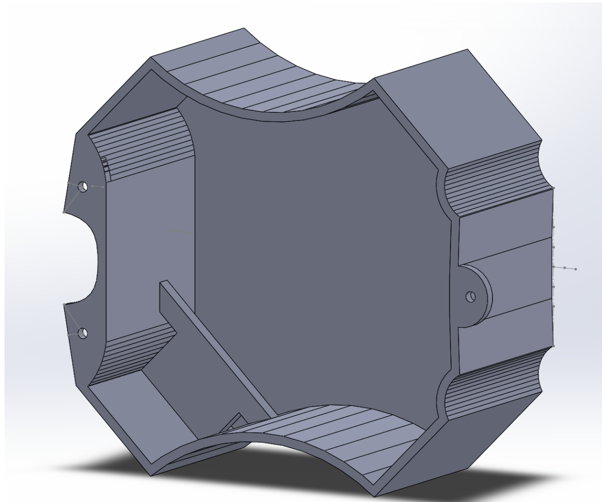


Figure 11.7: Technical box seen from below

Battery Box

Figure 11.8 shows the model of the battery box which carry the battery. Its got one opening on one of the short sides, where the battery can be inserted and replaced with ease. The battery box is designed with four screw holes that allows us to fasten it to the bottom frame. The range sensor can be placed underneath the battery box.

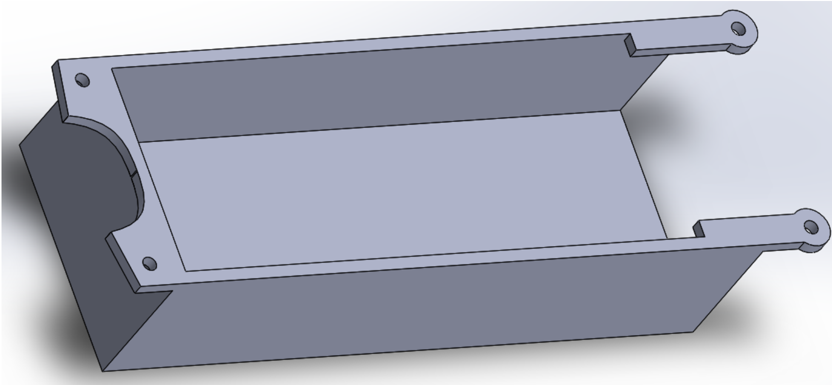


Figure 11.8: Battery Box

Arm

Figure 11.9 shows the design of the arms. On the left side we can see the cylindrical container where the motor can be placed and fastened using two screws from beneath, which are fastened directly into the motor. The wires from the motors are pulled through the cavity in the arm into the central body of the quadcopter, between the top and bottom frame where the ESC and the power distribution board are placed. There are two holes on the opposite side on the arm, this part lies between the top and bottom frame and is fasten using two screws.

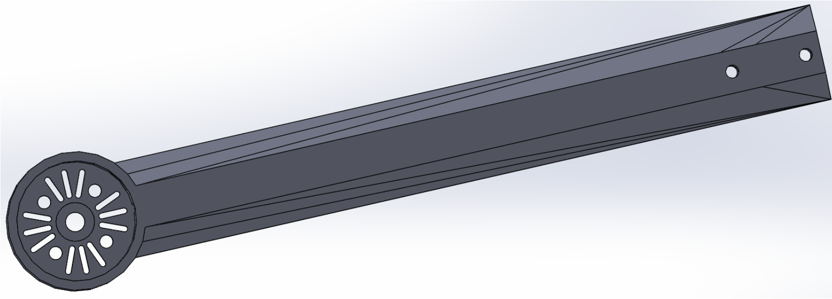


Figure 11.9: Model of the Arm

Leg

Legs are the final component needed in our model, and the model can be viewed in figure 11.10. The legs function is to raise the body of the quadcopter up from the ground, and provide horizontal support. The component is shaped and designed to be threaded onto the arms, and do not not require any extra equipment in order to be mounted. The legs have a slight dampening function for landing, this is accomplished using a thin layer of the material which makes the leg flexible. A square pillar are attached to the button of the legs, and is designed to slide into the rest of the leg and provide support.

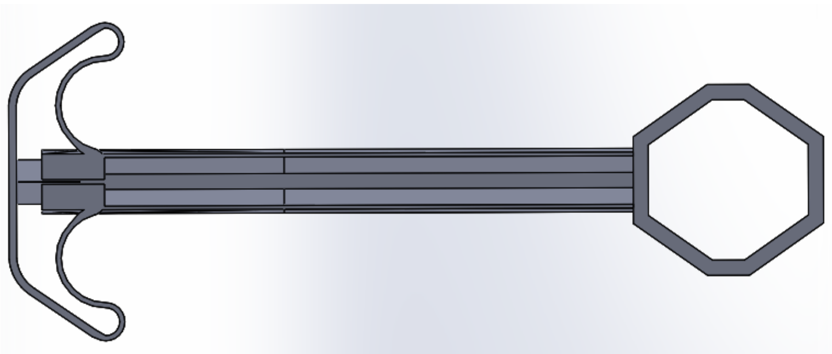


Figure 11.10: Model of the Leg

Printing Time

Table 11.3 indicates how long time it takes to print out each component, depending on the printer as well as how many components are being printed, and their

composition. The total time to print an entire quadcopter is approximately 36 hours when using the 3D-printer owned by department of engineering cybernetics at Norwegian university of science and technology[27].

Component	No.	Time in hours	Total
Top frame	1	3	3
Bottom frame	1	3	3
Technical box	1	7	7
Battery box	1	6	6
Arm	4	3.45	13.4
Leg	4	0.75	3
Total	**12**		**35.4**

Table 11.3: Approximate time needed for printing the quadcopter

11.5 Observation and Results

Figure 11.11: Prototype version 2

11.6 Discussion and Conclusion

The resulting model can be seen in figure 11.11. The production of the quadcopter is cheap due to the fact that we use a 3D printer, but the process is time consuming, which can be seen in table 11.3. The model can easily be printed and replicated by both the institute of engineering cybernetics as well as others.

The model consists of both a technical box as well as arms and legs to protect the electrical components against both impact as well as light protection from weather. The technical box provides enough space for the user to add and exchange electrical parts. The prototype is easily mounted using screws, which enables fast exchange of damaged parts.

One weakness with this model is that the arms are designed to fit our current motor, and therefore it could be problematic to use a motor with different size specifications. However, the motor model used in our quadcopter model exists in a series of different specifications and prices, and is widely used.

With this model, we have fulfilled the criterias stated in chapter 11.2. Testing have proved that the quadcopter is both light enough for a stable flight, as well as robust enough to handle a normal flight procedure. The battery box also provides an easy solution for exchange of the battery, which enables short ground time between longer flights.

11.7 Recommendation and Future Work

The 3D-printing of the different components is a time consuming process, and we therefore recommend to have some spare parts to reduce downtime if the quadcopter is damaged. We especially recommend to print some extra legs, they are the weakest component in this structure.

There are a few weaknesses with the model design which could be improved. The battery box is designed such that it is easy and fast to exchange battery, as can be seen in figure 11.8, but should have some safety mechanism for securing the battery during flight.

One of the weakest parts of the quadcopter is the propellers, which is also a part that often bump into other objects. For both robustness reasons as well as safety, we recommend that a structure is built around the propellers. It could be a separate component that can be mounted on the arm itself, of directly to the motor screw fixings.

12 Windows Application for Quadcopter Control

When altitude, attitude and position control are implemented in the microcontroller it is desirable to control all of these attributes. This can be implemented a number of ways ranging from a remote control, smartphone using bluetooth or Windows application using a radio transmitter.

12.1 Choice of Application Platform

Remote hand-held control is one of the means most commonly used by commercial quadcopters for controlling the aircraft. They have great advantages, the remote hand-held control provides reliable and intuitive control, and is easy to set up. For mass production purposes it is the best choice, but development is time costly for a single quadcopter, and therefore it is a poor choice for us.

Smarphones are common property in Norway, and development for android applications is well documented (Android operating system is incorporated in 76.6 % of today's smartphones [13]). Android applications can be developed in Android studio (alternatively eclipse with some extensions), which provides pre made functionality for android phones. All the functionality within the android applications are programmed in java while the appearance are commonly programmed in XML.

Although there are several different options for wireless communication when using a smartphone, the most obvious choice is to either use Wi-Fi or bluetooth, they are both included in almost all smartphones. Bluetooth is a safe option, it utilized radio signals 1000 times weaker than the standard wireless technologies used by mobile phones, and runs on 2.4GHz RF[4]. Bluetooth provides the option for a secure connection, meaning that once a connection has been made, no-one can listen in and there are also no interference from other Bluetooth devices. The range is normally limited to approximately 10 meters, which is less than desirable when it comes to quadcopter control, but an advantage is that the devices does not need to have line of sight once the connection is established. There exists bluetooth devices that can ensure a range of 90 meters, but these class 1 bluetooth devices[4] are not standard on most smartphones, and thus becomes irrelevant to consider.

Another good choice is to develop an Windows application and provide communication through a radio transmitter. There are numerous of different programming languages that can be used to create a good controller, and documentations for all of these are countless. According to our radio transmitter's data sheets [5] the calculated theoretical radio link distance is a few kilometres is open areas. Based on our experience as well as other user reviews [11] this can be exaggerated, however it can grantee several hundred meters of coverage in open areas.

Comparing the different options we can conclude that a smartphone with bluetooth connection may be the preferred choice for control if the quadcopter is to be used indoors. However our main objective embraces outdoor flying and open environment, and in this situation the radio link will provide far superior range. In addition it is far more convenient to process data and analyze data using a Windows application, since we need to conduct experiments when developing the quadcopter. For those reasons we find it most suitable to develop a Windows application in order to control the quadcopter, using a radio transmitter for communication.

12.2 Choice of Programming Language and Framework

There are numerous of good programming languages as well as programs to choose from when developing a graphical user interface in Windows. In the process of choosing programming language we considered several criterias:

- How easy is it to implement both basic as well as more advanced graphic interface
- How easy is it to implement functions for data analyzation
- How much effort it takes to implement serial communication with the radio transmitter
- Our own level of experience with the language and framework

The most common programming languages used when making graphical user interface applications in Windows are C#, java and python. We have some experience using all of these programming languages, however we only have experience of making a graphical user interfaces with C# and python.

Python does not come bundled with any graphical user interface frameworks (or toolkits), but there exists a huge number of them. We have experience using a framework named WxPython, this provide an easy solution for the creation of both basic and advanced graphics. It is however an event-based solution that can be challenging when implementing quadcopter control using the keyboard. Python provides a huge amount of pre-made functionality including serial communication protocols.

C# is based on C language and part of the .NET languages, C# is intended to be a simple, modern, general-purpose, object-oriented programming language [15]. C# provides the functionality to create elegant graphical designs, and the framework rewards you for exerting proper object oriented techniques with fast code. As with WxPython there exists solutions for serial communication with the radio transmitter, as well as general functionality.

There are many frameworks for graphical user interface development for C#, among the most common we find GOME, KDE, Unity and Xfce[21]. We have some experience using Unity for graphical user interface development, and therefore this became the obvious choice for framework.

Although both the previous discussed options can be argued to be equal options when designing our application, we choose to program our application in C# using the Unity framework. Unity and C# are easy to learn, provides great functionality and is well documented.

12.3 Programming Graphical User Interface in C# Using Unity3D

Unity is a cross-platform game engine used to develop video games for Windows, mac, linux, consoles, mobile devices and websites. Unity contains the tools and functionality to design and develop advanced 3-D games, and thus provides all the functionality needed to create an easy quadcopter control application. In addition to being a powerful cross-platform 3-D engine, Unity provides a user friendly development environment, easy enough for the beginner and powerful enough for the expert. There are numerous of advantages for using unity3D compared to other framework and development environments tools:

- Unity comes in two versions, free and pro. The free version provides most of the functionality and all of the core functionality needed for our project and is available for all persons or companies making less than gross 100.000 $ per year from programs made in Unity3D. The Unity3D pro license costs 1.500 $ and is inexpensive compared with the cost of game engines with the same functionality.
- One of the main advantages of Unity is the portability. The same source code can be used to build projects for multiple platforms with an incredible ease. The latest version allows you to move freely between 21 platforms, including Windows, IOS, linux, android and oculus rift. For our project it could ease the expansion for the quadcopter control to include smartphones as well as oculus rift.
- Unity is an engine that focuses on simplifying the application development workflow, and includes a visual editor. The visual editor allows you to build and modify the project very rapidly. The ability to run the application while simultaneously seeing the properties and locations of all objects in the scene is a powerful and time saving feature.
- Unlike many game engines, unity does not limit object behaviours to built-in modules that come packaged with the engine. Instead unity allows for

powerful behaviour written in any of its languages (JavaScript, C# and Boo). Furthermore, all three languages can be used at the same time within a project to allow people of different technology backgrounds to contribute to a project at the same time [17]. The fact that the languages are used as scripts allows for fast compilation time, quick iteration, and flexibility of design.

- Unity provides simplified object creation, especially for hierarchical object structures and dependencies, which simplifies application code. It contains a mechanism for building instances of objects, which may contain other dependent object instances[16]
- Unity has an interception capability, which allows developers to add functionality to existing components by creating and using handlers that are executed before a method or property call reaches the target component[16].
- It can read configuration information from standard configuration systems, such as XML files (which are used when developing android applications in android studio), and use it to configure the container[16].
- Unity makes no demands on the object class definition, there is no requirements to apply attributes to classes, and there is also no limitation on the class deceleration.
- Unity has an active and supported community, and great forums that can provide assistance and solutions to known problems. The size of this community is enormous, in 2012 53,1% of all mobile developers reported using unity to make games [16].

It is obvious that the Unity3D game engine provides an incredible framework for our application. In order to start with the design and development of our graphical user interface its important to identify the requirements. For our quadcopter control application we require:

- A solution for connecting any radio transmitter through serial cable to the computer, and thus a user interface where we can choose which serial port to communicate with
- Some configuration options, where we can edit the different control gains and also edit the bias for our IMU, which will reduce drift during flight
- Some control scheme where altitude and attitude can be controlled, and where position and yaw angle can be monitored

Our aim it to create a simple and intuitive graphical user interface using a basic design. This will enable new users to quickly grasp the fundamental properties with the application, and thus save time.

12.4 Communication Design and Implementation in C#

The communication between the Windows application and the quadcopter is performed using a radio transmitter connected through a serial cable. In order to achieve this we need to establish a connection with the radio transmitter in our Windows application. In addition there are several concerns that needs attention when developing the communication protocols. Furthermost is the fact that we can loose connections at any time, and we may experience sporadic byte loss.

12.4.1 How to Change COM Port on the Radio in Windows

The first time you insert the serial cable from the radio transmitter, drivers will automatically be installed and the device will be assigned a serial port, also named "COM port" in Windows. What COM port number the radio link will be assigned to depends on which COM ports are available.

There is a known problem using C# and unity when connecting to a COM port number that is 10 or greater. There is a solution for fixing this, where you specify a COM port number greater than 9 using the following syntax: "\\.\COMXX" where XX is the COM port number[25]. This solution however, makes the unity software application buggy and does not work properly on all computers. For this reason we choose to come up with a different solution for the situations where you end up having a COM port greater or equal to 10.

Our solution is to change the assigned COM port provided by Windows. This is a solution that can be performed on any Windows computer, and once its been completed the radio transmitter will be assigned this COM port for the future and therefore this is a one-time effort. There are very few occasions where all COM ports 1-9 are being used by hardware, and in those occasions some of them can usually be assigned a greater COM port.

Following is a step-by-step guide for how to change COM port for the radio in Windows 7. Press start, right click on computer and select properties.

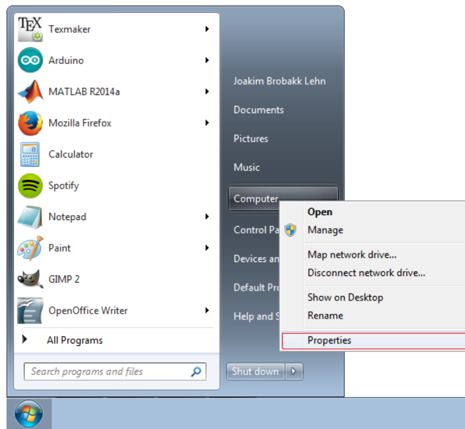


Figure 12.1: How to access Windows properties

In the properties menu, select Device Manager.

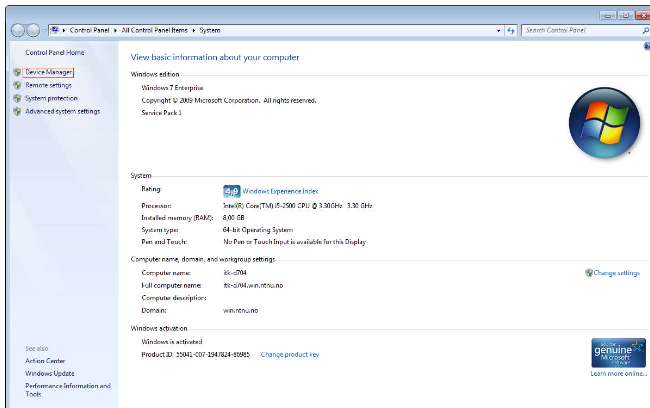


Figure 12.2: How to access Device Manager

Once you have opened the device manager, look for the list item named "ports (COM & LPT)", open this list item, and search through the names of all the connected COM ports. The radio transmitter will be labeled "Silicon Labs CP210x USB to UART Bridge" followed by a COM port number. If the COM port number is 10 or greater you need to proceed with this guide to change the COM port. Right click the radio transmitter item and select properties.

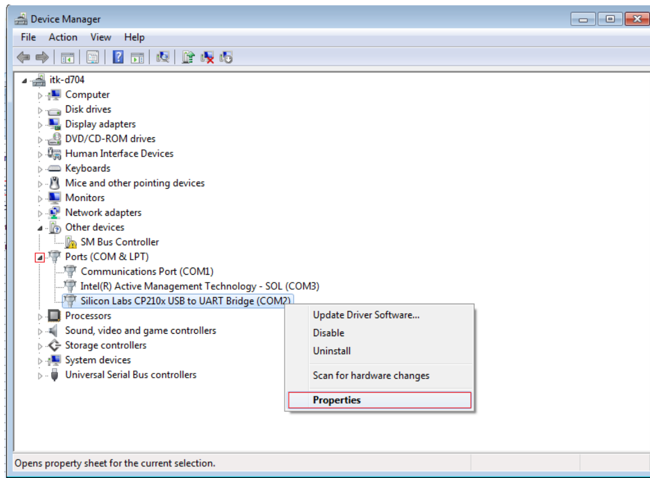


Figure 12.3: How to access COM port properties

Press the "port settings" tab in the upper part of the window, and then press "Advanced..."

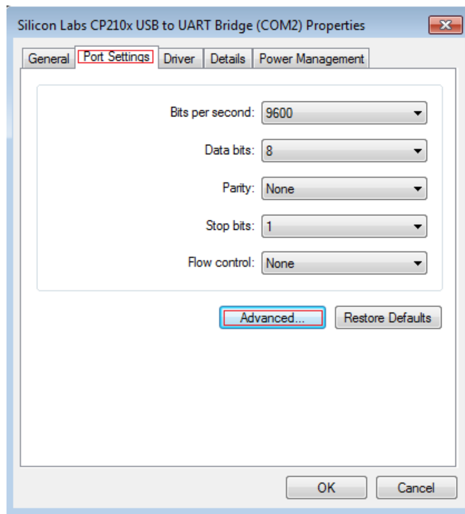


Figure 12.4: How to access Advanced port settings

In the new window that opened you can select a new available COM port in the lower left corner. Simply select one between 1-9 that is not labeled as "(in

use)" and then press "OK" in all the open Windows. You may need to restart your computer in order for this to be updated. If every COM port 1-9 are labeled as "(in use)" you need to go back to the Device Manager and find one COM port 1-9 that you can change to a higher COM port number. This will release a COM port for the radio transmitter.

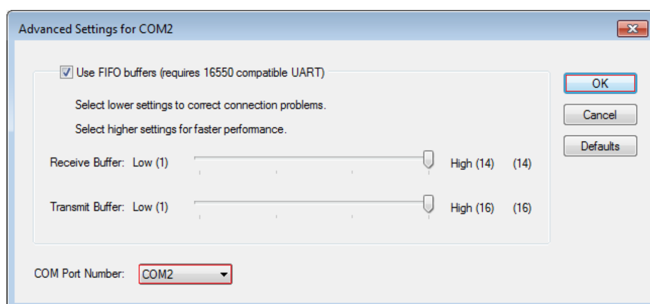


Figure 12.5: How to change COM port in hardware

12.4.2 How to Enable Serial Communication in C#

Serial communication in C# are not among the most challenging tasks, and .NET provides very useful internal classes which can make this kind of communication very simple and efficient [6].

When opening a Serial port for communication you need to know several aspects regarding how the communication works with that particular device:

- Port name: In order to connect to any COM port, you need to know which one you are interested in. This can be done by retrieving a list of all the available serial ports. When you do know which COM port to connect to, the port name will simply be "COMx" where x is the number 1-9.
- Baud rate: Different devices runs on different baud rates, if you input the wrong baud rate than only gibberish will come out. Baud rate is the unit for symbol rate or modulation rate, in symbols per second or pulses per second [18]. Our radio transmitter runs on baud rate 57600.
- Data bits: Sets the standard length of data bits per byte. As default it is 8, which is also what we want for our serial communication.
- Parity: Here you either enable or disable the parity-checking protocol. Parity is an error-checking procedure in which the number of 1s must always be the same—either even or odd—for each group of bits that is transmitted without error. If a parity error occurs on the trailing byte of a stream, an

extra byte will be added to the input buffer with a value of 126[12]. For our case, the parity protocol is set as none.

- Stopbits: Set the standard number of stopbits per byte. Default value is One, and is the same value as the radio transmitter use.

Once a connection has been established, the .NET library provides functionality to send and read from serial port, as well as to check if the port is still connected.

12.4.3 Design of Communication Protocol

Our communication with the quadcopter are enabled by using a radio transmitter, which have many advantages, however one of the main problems with this solution are potential loss of connection and loss of bytes. These are two obstacles that needs to be handled in software on both the quadcopter side as well as the Windows application side.

There are some important limitations to consider when designing a communication protocol, one of the foremost aspects is the time delay imported into the system. The time it takes for the message to be sent from one side and be received on the other side varies, but can take as long as 80 milliseconds. Considering we need to process data on the receiving side as well as send a reply, it may take as much as 200 milliseconds before a reply will be received at the source of the original message. Combine this limitation with the fact that we need to detect and handle both connection loss and byte loss, we need to be careful and thorough when designing a communication protocol.

How We Detect and Handle Loss of Connection

The radio antenna microcontroller does not provide functionality that can present any information on the status of the radio link. The communication protocol needs to ensure weather or not the two antennas are actually still connected.

In our design of the communication protocol, the Windows application is responsible for the control of the connection, and can be considered the "master". The Windows application will send out an "ack" message in the situations where there have been no communication for some period of time, while the quadcopter will respond to any message received with a confirmation message.

In our communication design we created a list (or queue) of messages to be sent from the Windows application, any new message will be appended to the end of this list. Once a message have been sent from the Windows application it will wait for up to 200 millisecond for a confirmation to be received. As soon as we receive a confirmation from the quadcopter, we will delete the first element in the list (the message that was just sent) and transmit the next message in the

queue. If we have not received any confirmation during our 200 millisecond timer delay, we will try to transmit the same message over again.

We will encounter situations where we wont have any relevant information to send for some time, in order to ensure that the radio link connection still holds we have included an "ack" procedure. If there have been no messages transmitted for 250 milliseconds we will add an "ack" message at the start of the message queue to be sent out. This message requires the same confirmation from the quadcopter as any other message.

From the Windows application point of view, we have a stable connection if there have been a successful message transfer during the last second. The connection is considered lost if there have been no successful communication for more than one second. However, this does not change the behavior of the program, the application will still try to transmit the first queued message every 200 millisecond until it receives a confirmation. The application will inform the user that there is a problem with the radio link, and likewise inform the user when the connection is stable.

How We Detect and Handle Loss of Byte

In the cases where we have ensured that we have a stable connection, we still cant guarantee that all the bytes transmitted from one source reaches the other device. There is also a chance that some of the bytes can be corrupted.

In order to handle these challenges we have created this message format; "xxxMESSAGEend" where:

- The first part xxx are three numbers, these three numbers represents the length of the message in bytes, and includes both the message part as well as the end part. The first operation to be performed is to check that the first three bytes are indeed numbers. If they are not numbers, the rest of the message will be ignored since there have clearly already been some byte loss, or in the rare case; a byte corruption. When we have checked that they are in fact numbers, we know the length of the message, and how many bytes we are waiting for.

- The message part speaks for itself, this is the part of the transmission that contains the actual message. The message part starts with a flag that indicates what type of message it contains, which makes the message handling effective and enables us to use short messages.

- Every message ends with the same three characters, namely "end". This is to indicate that this is the end of the message, so there is no chance that two messages can be examined as one message. We use our knowledge about the number of bytes provided at the start of every message in combination

with a known message ending to ensure that we have no byte loss. If the message are the designated length and ends with "end", we will respond with a confirmation message and proceed with the message handling. If the message is to short or ends in some other manner than "end", we will dispense with the message and send a notification that the message received was corrupted.

This message structure along with our solution for detecting and handling the radio link connection have solved both our concerns regarding using a radio transmitter. We still got limitations concerning how many messages we can send ever second, but this has not reduced the functionality of the quadcopter, however, it will constrain our ability to represent detailed flight data in this report. We can also still experience situations where we have corrupted bytes in the message part, which will render the message useless.

12.5 Functionality Description of the Windows Application

12.5.1 Connection Tab

Following is a screenshot showing how the connection page in the Windows application looks. By inputting which COM port the radio link is connected to and its connection baud rate, a connection can be established by simply pressing "Open Com Port".

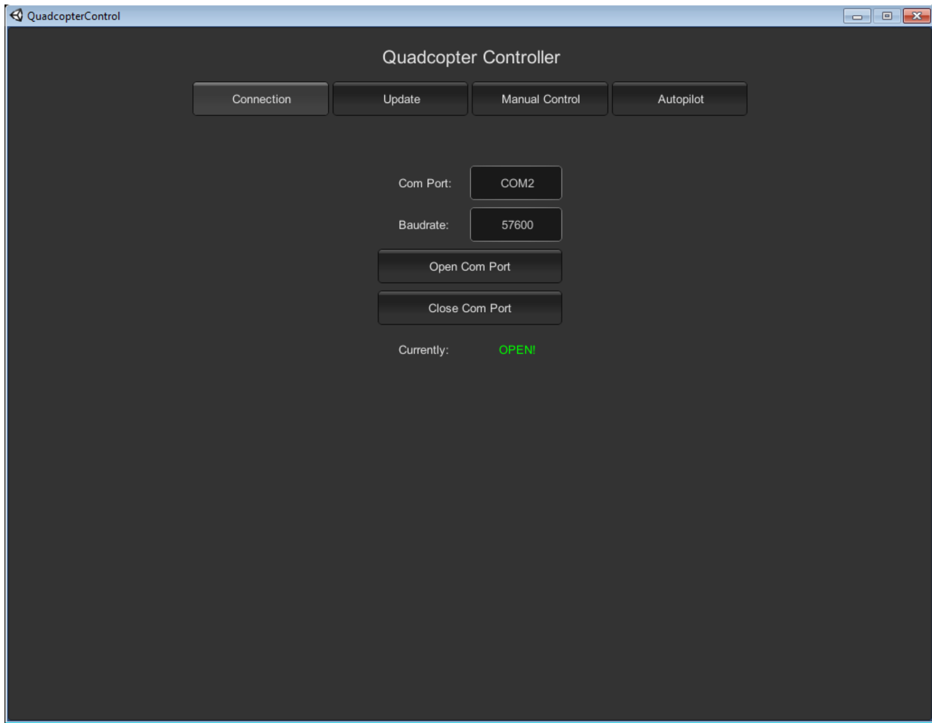


Figure 12.6: Connection tab in Windows application

12.5.2 Update Tab

One of the main aspects when developing a new controller for a system are gain tuning. Tuning is one of the most important parts of the development, and being thorough on this part can mean the difference between good performance and poor performance using the same controller.

We wanted to be able to change all the different gains in the controller, without connecting the quadcopter to the computer with a serial cable, our solution were to include this as a feature in the Windows application.

From the application we can update both the proportional, derivative and integral gains as well as to change the constant gain produced to provide lift. In addition we have included an offset control, which allows us to change the bias in the accelerometer in both roll and pitch. The bias can be changed by either pressing the button or by using the arrow keys on the keyboard. This is a great feature if the user notice that the quadcopter drifts in any direction, which most commonly is due to incorrect leveling of the accelerometer. Drift can also

be caused by uneven weight distribution by for instance the battery, and can be difficult to estimate while the quadcopter is stationed on the ground.

The application will on this tab provide information about the current connection status, current proportional, integral and derivative gains as well as the current roll and pitch bias settings. Pressing the spacebar on the keyboard will immediately stop the quadcopter propellers and is considered a fail-safe solution in case the quadcopter is out of control.

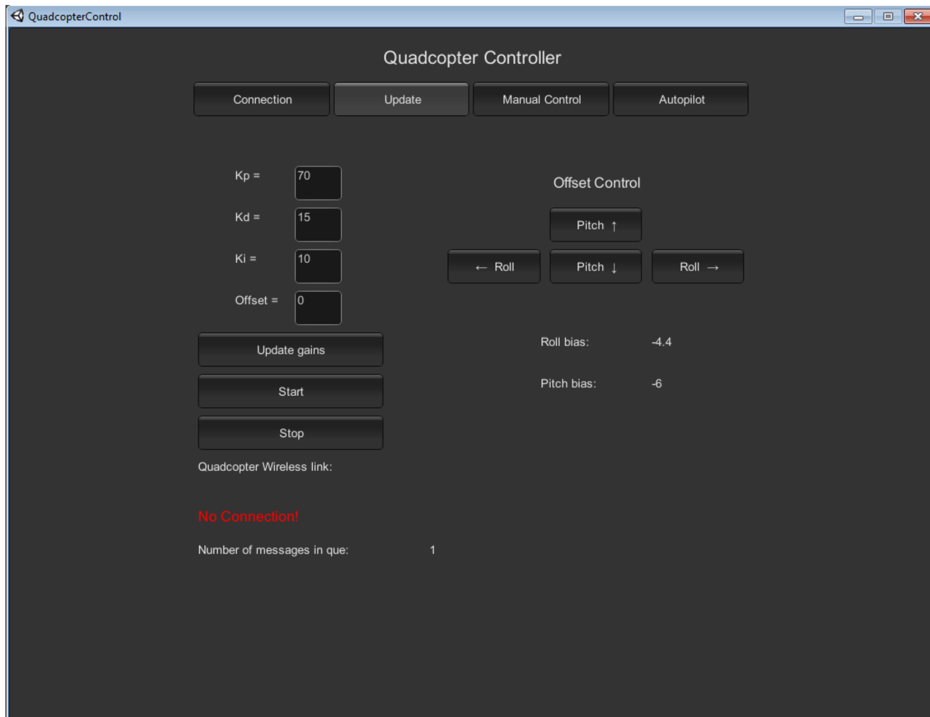


Figure 12.7: Configuration Update tab in Windows application

12.5.3 Manual Control Tab

One of the most basic user control of the quadcopter is to be able to control the angles and altitude of the aircraft. This can only be performed once the attitude and altitude controllers have been proved stable. In order to control said variables we have made a tab called "Manual Control" in the Windows application.

In the Manual Control section all angles can be manipulated, and our goal were to design a system that is intuitive for the user. Any button can either be

pressed using the mouse, or using the keyboard for the respective key.

- Yaw control: The Q and E buttons will change the yaw angle of the quadcopter, Q will make the aircraft turn counter-clock-wise while the E will make the aircraft turn clock-wise. The current yaw estimation (based on the magnetometer in combination with the IMU) are represented as a compass, if the compass needle points at 0 degrees it obviously means that the quadcopter reference points towards north.

- Pitch control: The W and S buttons will change the pitch angle of the quadcopter. This control is intended for changing position and not to change the angle indefinitely, for that reason the change in the pitch angle is constant while the button is pressed down and will be reset to its original angle once released. This is a precaution to prevent the user from making the quadcopter unstable by inducing to large angles.

- Roll control: The A and D buttons will change the roll angle of the quadcopter. This control have the same limitation and behaviour as defined for the pitch control described above.

- Altitude control: The up and down buttons (they can also be enabled using the up and down arrow keys) are designed for the user to control the altitude. Below is a graphical representation of the current motor effort required to conduct the changes and keeping a steady altitude.

There are as described some limitations for what the user might actually perform, but they are all in place to ensure stability and safety for both the aircraft as well as people and property. The last fail-safe solution is to press the spacebar, this will ”immediately” stop the quadcopter propellers.

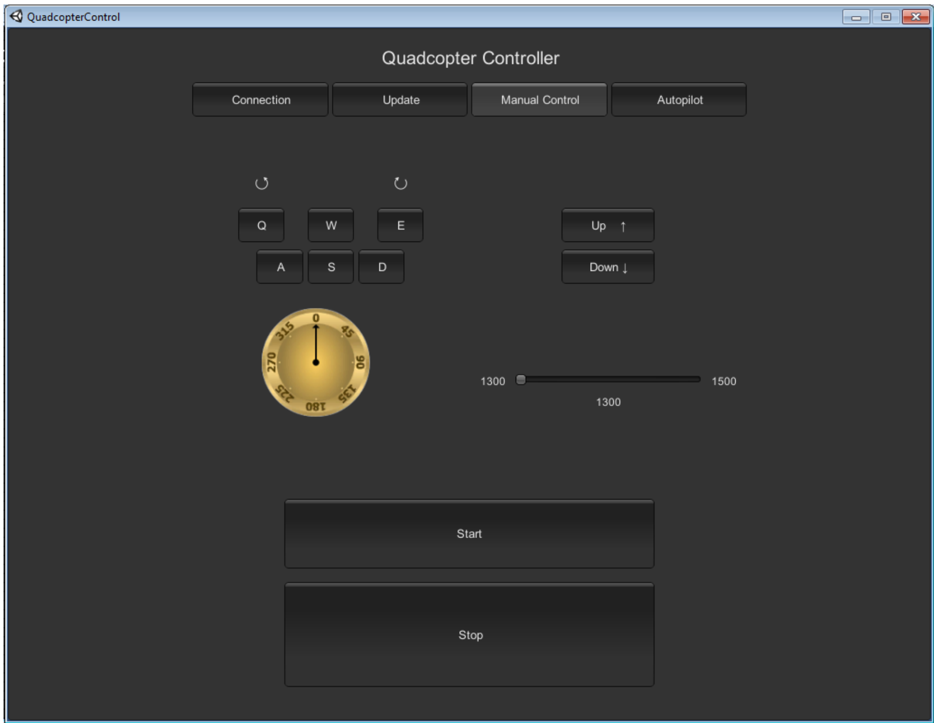


Figure 12.8: Manual Control tab in Windows application

13 Error Handling

The objective to control a quadcopter is a complicated task, composed of many processes and sensor measurements. There are numerous errors that can occur, some are crucial, while others can be handled such that the quadcopter can continue with its flight. In this chapter we will explore some of the errors and our strategies for detecting and handling the errors.

13.1 Problem Description

Explore the various probable errors that may occur, and develop a strategy for detecting and handling said errors.

13.2 Error Detecting and Handling Strategies

13.2.1 Loss of Communication

The quadcopter communicates with the Windows application using a radio transmitter, the radio link can at any time be broken, and this needs to be handled. In section 12.4.3 we discuss how to ensure whether or not we got a stable communication, on the arduino side we have a slightly simplified test for the stability of the communication.

We have designed our Windows application to always send or resend some message at least every 250 milliseconds. based on this we know that we should get at least four messages every second. On the arduino side we deem the radio link as stable if we have received a proper message according to the message protocol (as described in section 12.4.3) during the last 1.5 seconds. In other words we allow the loss of 5 messages in a row before the communication is presumed lost.

The loss of communication is considered a critical error, if the quadcopter experiences some other error there would be no way for stopping it. Therefore, in the case of loss of communication we will stop the motors to prevent danger to people or property.

13.2.2 Low Battery

The quadcopter is powered by a battery, and the voltage is measured as described in section 2.11. The battery is fully charged if each cell voltage is 4.2 Volt, and around 20% if we measure 3.7 Volt. The total voltage for the battery will then be 12.6 Volt at 100 % and 11.1 Volt across the three cells at 20 %. To include a safety margin we will consider any voltage less than 11.3 Volt as "low battery".

In the case of Low battery we will alert the user of the Windows application that the voltage is low. The user should then take action to stop the quadcopter

fairly soon in order to not damage the battery, or risk crashing the quadcopter. At low voltage the user would also experience poorer performance from the quadcopter, this is a result from slightly lower input to each of the motors and therefore we get a more "sluggish" response. An even better solution would be to implement a landing function as described in section 13.3.1, to be called once the battery gets as low as 11.1 Volt as an extra safety precaution.

13.2.3 Loss of Sensors

There is a slim chance that we at any time can loose connection with sensors, for various reasons, but mainly these problems will come from loose wires. Some sensors are more critical with respect to the flight operations than others, but all sensors serves a purpose. The challenge is to detect that we have lost the connection, and develop some strategy to reduce the potential danger to both the quadcopter as well as people and property.

Inertial Measurement Unit

The most critical and necessary sensor are the IMU with its accelerometer and gyroscope measurements. It is impossible to control the quadcopter without frequent updates from this sensor, and a loss of connection will cause the quadcopter to become unstable. The loss of sensor connection can come from loose wires, a breach in the cable or a malfunction in the IMU itself. All of them are unlikely, the IMU and Arduino are attached to the body of the quadcopter and thus do not move with respect to each other, but there is still a slight possibility that this can happen.

There are two ways to detect loss of connection with the IMU. The first one is to use a timeout strategy, if we have not received any data in an unreasonable time period we can argue that it is a high probability that the connection is lost, and either way we do not know the current status of the quadcopter. This strategy will detect problems related to the wires. The other method is to compare the value of the raw data provided by the IMU, to the previous received data. If the data is equal for 5 time samples, there is a high probability that we have an malfunction in the IMU itself, it's even unlikely that we would experience identical data for all axes two measurement in a row.

We only use one IMU on the quadcopter, and there are no simple strategy to safely lang the aircraft without accelerometer and gyroscope measurements. The best strategy in order to reduce potential damage is to stop all four motors and let the quadcopter crash to he ground. This is not a good solution, but the potential danger of letting the quadcopter fly freely are more severe. In order to develop a better strategy a second IMU is required.

There is one last potential danger regarding the IMU, and this concerns a malfunction in the IMU itself. If the IMU generates "random" values for the accelerometer and gyroscope measurements we would have a dangerous situation. This is difficult to discover and handle, but in theory we could compare the raw data to the previous data, and use the motor input data to reason whether or not they are realistic. This is something we have not implemented in the microcontroller.

Range Sensor

The range sensor provides reliable height measurements when hovering at altitudes below 1.2 meters. This prevents the quadcopter from drifting away, and from crashing with the ground. We can encounter some of the same errors as with the IMU; no data, same measurements or random measurements.

We can detect that we receive no data in the same fashion as with the IMU, by using a timeout. The data we receive from the range sensor is the time it takes from sending an ultrasonic pulse until the pulse returns, and is measured in microseconds. We will use the same strategy for discovering sensor malfunction as we did with the IMU, by comparing the last five measurements. If they are identical we assume that the sensor connection is lost. It is significantly easier to determine whether we get reliable data, or if the data is just random in the range sensor than it is to access with the IMU. We can use the model along with the previous sensor output combined with motor input to reason whether the sensor data is reliable or not, and use this to detect if the sensor is malfunctioning.

If we discover that we have lost connection with the range sensor, our strategy is to let the propellers spin slightly slower than what's necessary in order to keep a constant altitude. This will cause the quadcopter to drop fairly slowly towards the ground. It will slow down slightly when its close to the ground due to the increased thrust each propeller generates close to the ground. Using this strategy we ensure that the quadcopter lands with a stable attitude, and in most cases we land so softly that the quadcopter receives no damage.

Magnetometer

We will use the same strategy for detecting loss of connection with the magnetometer as with the range sensor, and we have the same possible errors. If we detect that the connection is lost we will use the accelerometer and gyro measurements in combination with the model for the quadcopter in order to estimate the yaw angle. This estimation is fairly good, and will allow us to continue with the flight. This estimate will detect any yaw rotation and slow it, but we are not able to set the quadcopter facing any specific angle from the Windows application.

GPS

We use the GPS in order to acquire a fairly good position estimate, and use this to prevent any major drift in position. We may experience the same errors as with the magnetometer and range sensor, and detect them in the same manner. The chance that the GPS will provide random measurements are however extremely slim. In addition to these errors we can also loose lock with the satellites, this is an error that the GPS chip will detect itself.

If we experience some error regarding the GPS, we can use the model along with IMU sensor data to estimate the position. Using this we will experience drift in the position estimation, and we recommend that the user lands the aircraft.

13.2.4 Roll or Pitch Angles Close to Singularity

The attitude controller is based on the system model, and our model uses Euler angles to describe the behavior. Euler angles provides an easy and intuitive representation of the model, but the main weakness is that it contains singularities at ± 90 degrees for roll and pitch angles. This will cause the control system to act unstable, and unpredictable.

Our strategy for handling this is to stop the motors if the angles are above ± 70 degrees for either roll or pitch. Our main concern are the singularities, but we also consider the quadcopter unstable, and with small chances of recovery if the roll or pitch angle somehow are above ± 70 degrees. By stopping the motors we will experience a crash landing with an unstable attitude, and the aircraft will in most cases be inflicted significant damage. This is still a better strategy than to let the motors keep spinning, where we risk further damage to people or property.

13.3 Recommendation and Future Work

13.3.1 Loss Of Communication

Our strategy for handling loss of communication is as described in section 13.2.1, to stop the motors in order to prevent danger to people or property. In most cases the quadcopter will function normally when the communication link is broken, and therefore we will cause an unnecessary crash. A better solution is to implement a landing function. This landing function is useful both when we have loss of communication as well as in normal operating mode.

13.3.2 Loss of Sensors

IMU

The IMU is the single most important sensor used in the quadcopter, and the quadcopter can simply not work without one. A possible improvement is to include a backup IMU, to be able to faster detect any errors (including the randome values that we currently have no solution to detect, as described in section 13.2.3) as well as providing a solution when we have an IMU malfunction.

13.3.3 Roll or Pitch Angles Close to Singularity

The main problem regarding high roll or pitch angles is the fact that the model uses Euler angles which contains singularities at ±90 degrees. By using Quaternions we avoid this problem, and can handle these angles, assuming that the control system is able to re-stabilize the quadcopter.

14 Discussion and Conclusion

Throughout this paper, we have developed a quadcopter sensor platform from scratch, including; system modeling, state estimation, control design, communication handling and implementation of a user interface. Furthermore, a quadcopter frame was designed and printed using additive manufacturing techniques. Standalone cheap sensors have been coupled through an Arduino Due microcontroller and different filtering methods have been examined.

Implementation of discrete filtering schemes were devised and implemented. Together with a system model describing the quadcopter motion, the library implementation of the Kalman filter proved useful throughout the project. By defining the system propagation matrices and the resulting jacobians, new estimation schemes were simple to implement on the microcontroller.

Fast and accurate attitude estimates were obtained in Chapter 5 and 6 with the aid the extended Kalman filter. However, the Euler angles used to describe the attitude contains singularities at $\theta = \pm 90°$. A solution (which is harder to understand and implement) would be to use a quaternion system description. This would enable the quadcopter to perform acrobatic maneuvers.

The position estimates provided by the GPS should not be used directly with a control scheme relying on position data, as this would lead to the quadcopter "hopping" around the desired position. When flying close to the ground, the ultrasonic sensor can be used as an alternative method for height estimation in order to avoid ground collisions.

A proportional-integral-derivative controller proved to yield more stable roll and pitch angles when compared to the alternatives and stable flight was achieved. A nonlinear controller could in theory provide better inputs to the system and thus lead to improved performance, but implementation would be time consuming.

To allow user control of the quadcopter, an altitude controller was devised, implemented and proved stable. Together with the attitude controller, this enabled the quadcopter to hover in place. However, the controller is limited by the range of the ultrasonic sensor if the GPS does not have a satellite lock.

Guidance systems calculating required velocity vectors were explored, and a velocity controller to achieve the desired velocity vector was deduced. Implementation of these techniques will allow autonomous flight in the future, were the user simply provides a set of desired way-points.

To enable user configurations and control mid-flight, a Windows application for user control was implemented. Through radio transmission and by utilizing foul proof communication protocols, connection over long distances is assured.

This project has given us valuable experience when it comes to; system modeling, advanced filtering and estimation methods, control algorithms, graphical

user interface development, project management, microcontroller programming and conduction of an extensive project. We are satisfied with our overall achievements and everything we have learned through the execution of this project.

15 Future Work

The quadcopter possesses countless possibilities when it comes to functional capabilities. Some of the expansions we would consider to implement in the future, can be summarized as

- Using quaternions instead of Euler angles to describe the attitude to avoid singularities
- Adding a differential pressure sensor to the quadcopter, enabling height estimation without GPS above the range of the ultrasonic sensor
- Implementation of the velocity controller to enable autonomous way-point control
- Tracking of target equipped with a GPS device
- Adding a camera and develop a target tracking algorithm using imaging techniques

References

[1] Arduino - arduino due. http://www.arduino.cc/en/Main/arduinoBoardDue. Acessed: 2015-05-13.

[2] Arduino - arduino leonardo. http://arduino.cc/en/Main/arduinoBoardLeonardo. Acessed: 2015-05-13.

[3] Battery specifications. http://www.hobbyking.com/hobbyking/store/__9184__Turnigy_5000mAh_3S_20C_Lipo_Pack.html. Acessed: 2015-05-13.

[4] Bluetooth vs rf. http://www.headsets.com/headsets/resources/bluetooth-vs-rf.html. Acessed: 2015-05-13.

[5] Hm-trp series 100mw transciever modules v1.0. http://www.hoperf.com/upload/rf_app/HM-TRP.pdf. Acessed: 2015-05-13.

[6] How to work with c-sharp serial port communication. http://codesamplez.com/programming/serial-port-communication-c-sharp. Acessed: 2015-05-13.

[7] Matrixmath library. http://playground.arduino.cc/Code/MatrixMath. Acessed: 2015-01-25.

[8] Motor specifications. http://www.himodel.com/electric/SUNNYSKY_Angel_Series_A2212-980KV_2-3S_Airplane_Outrunner_Brushless_Motor.html. Acessed: 2014-09-03.

[9] Playground Arduino - mpu-6050 accelerometer + gyro. http://playground.arduino.cc/Main/MPU-6050. Acessed: 2014-09-11.

[10] Quadcopter dynamics and simulation. http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/. Acessed: 2015-05-14.

[11] The real specs and ranges for 3dr radios. http://diydrones.com/forum/topics/the-real-specs-and-ranges-for-3dr-radios. Acessed: 2015-05-13.

[12] Serialport.parity propoerty. https://msdn.microsoft.com/en-us/library/system.io.ports.serialport.parity. Acessed: 2015-05-13.

[13] Smartphone os market share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Acessed: 2015-05-13.

[14] Ultrasonic ranging module specifications. http://www.micropik.com/PDF/HCSR04.pdf. Acessed: 2015-05-13.

[15] What is c-sharp good for. http://forums.codeguru.com/showthread.php?313788-What-is-C-good-for. Acessed: 2015-05-13.

[16] When should i use unity. https://msdn.microsoft.com/en-us/library/ff660859. Acessed: 2015-05-13.

[17] Why you should be using the unity game engine. http://www.informit.com/articles/article.aspx?p=2031153. Acessed: 2015-05-13.

[18] Wikipedia, The Free Encyclopedia - baud. http://en.wikipedia.org/wiki/Baud. Acessed: 2015-05-13.

[19] Wikipedia, The Free Encyclopedia - counter-electromotive force. http://en.wikipedia.org/wiki/Counter-electromotive_force. Acessed: 2015-05-14.

[20] Wikipedia, The Free Encyclopedia - electronic speed control. http://en.wikipedia.org/wiki/Electronic_speed_control. Acessed: 2014-09-11.

[21] Wikipedia, The Free Encyclopedia - graphical user interface. http://en.wikipedia.org/wiki/Graphical_user_interface. Acessed: 2015-05-13.

[22] Wikipedia, The Free Encyclopedia - hybrid system. http://en.wikipedia.org/wiki/Hybrid_system. Acessed: 2015-05-25.

[23] Wikipedia, The Free Encyclopedia - nonlinear control. http://en.wikipedia.org/wiki/Nonlinear_control. Acessed: 2015-05-13.

[24] Wikipedia, The Free Encyclopedia - ziegler-nichols method. http://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method. Acessed: 2015-05-13.

[25] Windows - createfile function. https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx. Acessed: 2015-05-13.

[26] Y. M. Al-Younes, M. A. Al-Jarrah, and A. A. Jhemi. Linear vs. nonlinear control techniques for a quadrotor vehicle. 2010.

[27] T. R. Andersen, H. Braaten, A. V. Burkow, H. Leithe, N. I. Rugsveen, and K. Stenroed. Kvadrokopter: Fra prosjekt til produksjon. 2015.

[28] R. W. Beard and T. W. McLain. *Small unmanned Aircraft Theory and Practice*. Princeton, 2012.

[29] T. I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011.

[30] J. K. Hedrick and A. Girard. Control of nonlinear dynamic systems: Theory and applications. 2010.

[31] A. V. Hystad and J. B. Lehn. Model, design and control of a quadcopter. 2014.

[32] L. B. Prasad, B. Tyagi, and H. O. Gupta. Optimal control of nonlinear inverted pendulum dynamical system with disturbance input using pid controller and lqr. 2011.

[33] N. Shimkin. Nonlinear control systems. 2009.

A Building Quadcopter Prototype 2

Following is a detailed mounting procedure for the quadcopter prototype 2[27]

A.1 Part List

Pos	No.	part number	producent	Description
1	1	A000062	Arduino	Arduino Due
2	4	Angel A2212 2212-980KV	SUNNYSKY	Motor
3	4	1045R	Any	Propel 10x4.5 (254mX114mm)
4	4	HK-SS20A	Hobbyking	ESC -SS series 18-20A
5	1	GY-521	Arduino	IMU – MPU6050
6	1	M2011051101	GeekOnFire	GPS – Arduino shield
7	1	HMC5883L	Any	Magnetometer
8	1	GP2Y0A21	Sharp	Range sensor - IR
9	1	Telemetry Kit 433Mhz	3DRobotics	Radio – 3DR Radio set
10	1	9171000033	HobbyKing	Power distribution board
11	1	T5000.3S.20	HobbyKing	Battery (11.1V-5000mah)
12	1	A2012042808	GeekOnFire	GPS Antenna
13	20	3.5mm Male	Any	Male banana plug
14	12	3.5mm Female	Any	Female banana plug
15	1	XT-60 Male Female Bullet	Any	Battery connector
16	1	Dupont Wire Cable	Any	2.54mm
17	1	Multistar XT60	Any	power distributor

Table A.1: Part list

Pos	**No**	**Drawing file**	**Description**
101	1	Battery_box	Battery box
102	1	Button_plate	Bottom frame
103	1	Top_plate	Top frame
104	1	Tech_box	Technical Box
105	4	Arm	Arms
106	4	Leg	Legs

Table A.2: 3D print

Pre-ordered parts is listed in table A.1. Parts that needs to be 3D-printed is listed in table A.2. The necessary screws and nuts are listed in A.3.

Pos	No.	Component	Proucent	Description
201	15	M3 Nuts	Any	Nuts
202	8	M3x35 screw	Any	Screw to mount Arms
203	8	M3x6 screw	Any	Screw to mount motors
204	7	M3x10 screw	Any	Screw to mount frames

Table A.3: Screws and nuts

A.2 Preparation of Pre-ordered Parts

A.2.1 Motors

The motor (pos 2) is delivered with unterminated cables, they are extended to 22 cm, and needs to be terminated using male banana plugs (pos 13)

Figure A.1: Prepared motors with termination

A.2.2 Electrical Speed Controller

The ESC(pos 4) is delivered with unterminatd cables, In order to prepare them for mounting, you need to:

- Terminate using male banana plug (pos 13) on the battery + and -.
- Terminate using female banana plug (pos 14) on motor A, B and C.

- Terminate using male header pin on black and white signal cable.
- The red signal cable is not in use, and hence does not need to be terminated.

Se figur A.2

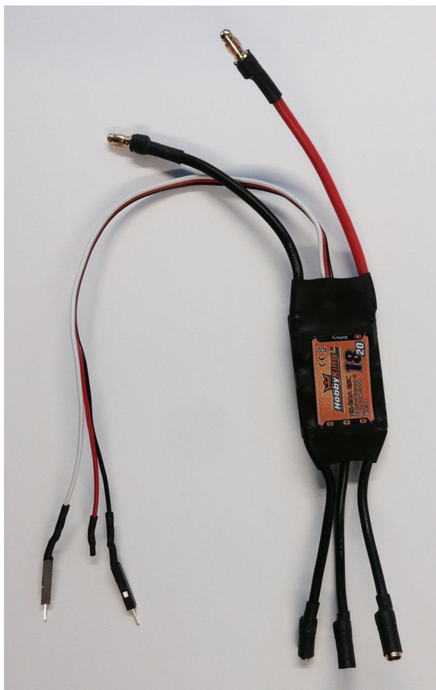


Figure A.2: A prepared ESC should look like this

A.2.3 Range Sensor

Keep the original length and terminate using male hader pin on alle three cables, see figure A.3

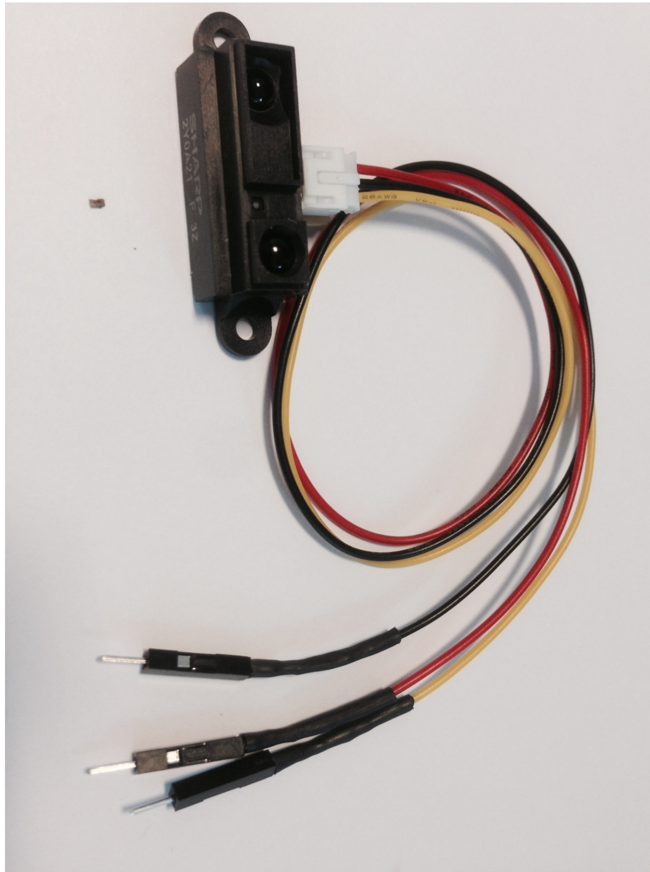


Figure A.3: Prepared range sensor

A.2.4 Inertial Measurement Unit And Magnetometer

The IMU and magnetometer need to be soldered using the accompanying header pins as showns in figure A.4

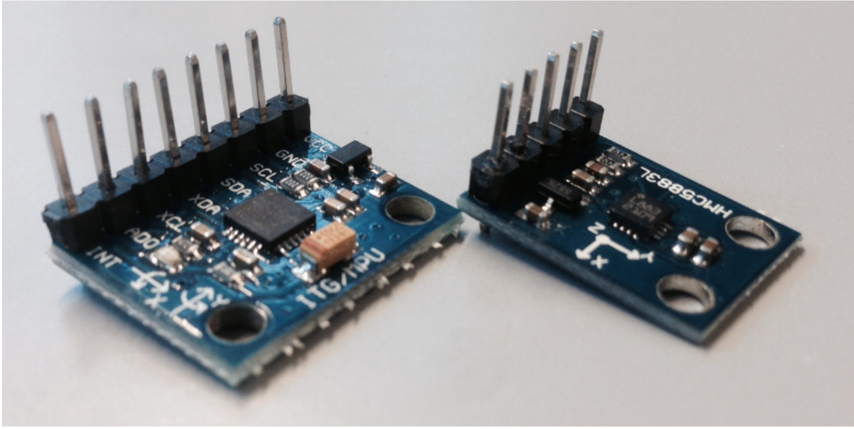


Figure A.4: IMU and magnetometer with pins

A.3 Wiring diagram

A graphical illustration of the wiring's can be seen in figure A.5

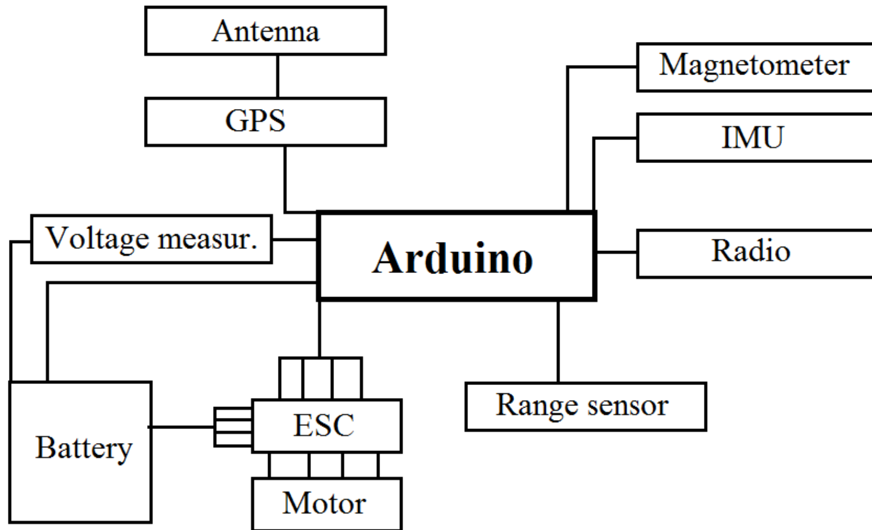


Figure A.5: Wiring diagram

A.4 Cable List

All cables is connected as shown in table A.4

Ref.	Cable	Color	Source	Destination
K1	pos 9	Yellow	Radio	Ard: 0
K2	pos 9	Green	Radio	Ard: 1
K3	pos 9	Red	Radio	Ard: 5V
K4	pos 9	Black	Radio	Ard: GND
K5	pos 12	Coax	Antenna	GPS
K6	pos 4	White	ESC 1	Ard: 9
K7	pos 4	Black	ESC 1	Ard: GND
K8	pos 4	Red	ESC 1	PDB: +
K9	pos 4	Black	ESC 1	PDB: -
K10	pos 4	White	ESC 2	Ard: 10
K11	pos 4	Black	ESC 2	Ard: GND
K12	pos 4	Red	ESC 2	PDB: +
K13	pos 4	Black	ESC 2	PDB: -
K14	pos 4	White	ESC 3	Ard: 11
K15	pos 4	Black	ESC 3	Ard: GND
K16	pos 4	Red	ESC 3	PDB: +
K17	pos 4	Black	ESC 3	PDB: -
K18	pos 4	White	ESC 4	Ard: 12
K19	pos 4	Black	ESC 4	Ard: GND
K20	pos 4	Red	ESC 4	PDB: +
K21	pos 4	Black	ESC 4	PDB: -
K22	pos 8	Yellow	Range sensor	Ard: A0
K23	pos 8	Red	Range sensor	Ard: 5V
K24	pos 8	Black	Range sensor	Ard: GND
K25	pos 8	Red	Voltage meter	Ard: Vin
K26	pos 8	Black	Voltage meter	Ard: GND
K27	pos 8	Yellow	Voltage meter	Ard: A1
K28	pos 11	Black	Battery: -	PDB: -in
K29	pos 12	Red	Battery: +	PDB: +in
K30	pos 16	Any	Compass: SDA	Ard: 2
K31	pos 16	Any	Compass: SDL	Ard: 3
K32	pos 16	Red	Compass: Vin	Ard: 5V
K33	pos 16	Black	Compass: GND	Ard: GND
K34	pos 16	Any	IMU: SDA	Ard: 2
K35	pos 16	Any	IMU: SDL	Ard: 3
K36	pos 16	Red	IMU: Vin	Ard: 5V

K37	pos 16	Black	IMU: GND	Ard: GND

Table A.4: Wiring schematic

A.5 Mounting Procedure

A.5.1 Motors and Arms

The motors (pos 2) are mounted in the arms(pos 105) using two screws(pos 203), the wires are thread through the arm.

A.5.2 Electrical Speed Controller and Frame

The arms is mounted between the top frame(pos 103) and bottom frame(pos 102) using screws(pos 202) and nuts(pos 201). The ESC(pos 4) and power distribution board(pos17) needs to be connected with the motors and placed between the two frames before the arms are mounted. All cables are connected according to table A.4.

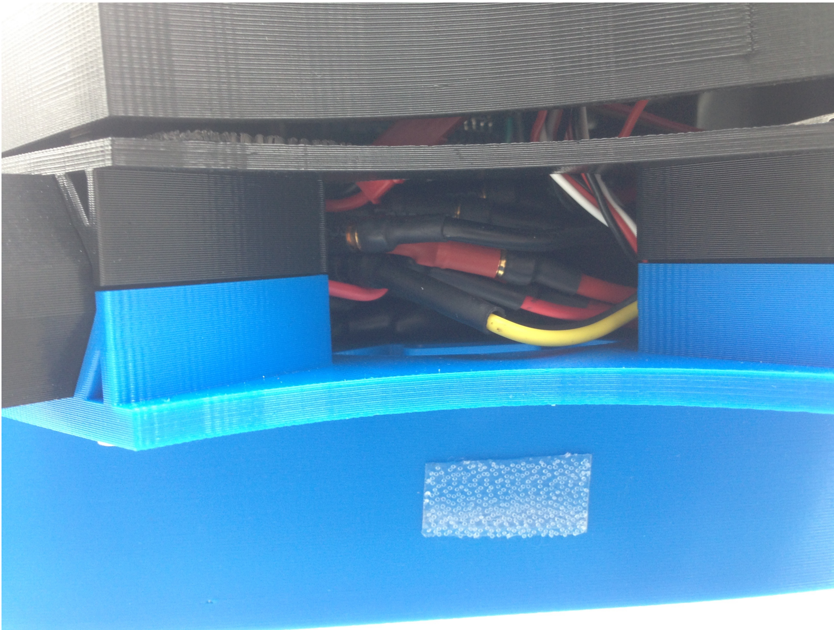


Figure A.6: The ESC's are palced in the space between the top frame and bottom frame

A.5.3 Technical Box

In the technical box we put the remaining electronic components. The Arduino(pos 1), IMU(pos 5), GPS(pos 6), magnetometer(pos 7) and radio(pos 9). The box is mounted using 3 screws(pos 204) and nuts(pos 201).

A.5.4 Battery Box

The battery box(pos 101) is mounted using screws(pos 204) and nuts(pos 201).

Figure A.7: The resulting quadcopter model

B Description and Walkthrough of Various Code

B.1 Arduino Code

B.1.1 Inertial Measurment Unit

Contains code related to the inertial measurement unit. This includes code to

- Extract raw IMU data
- Extract raw IMU DMP data (from the IMU's own filter)
- Extract filtered IMU DMP data
- Acquire raw IMU data, which is then processed in a extended Kalman filter
- Calculate the current IMU offset

B.1.2 Infra Red Range Sensor

Contains code to read the sensor measurements provided b the IR range sensor.

B.1.3 Magnetometer

We have included C code for both raw and Kalman filtered Magnetic measurements provided by the magnetometer.

B.1.4 Arduino Main

The folder titled "ARDUINO MAIN" contains all the C-files required when performing a flight with the quadcopter.

- *ARDUINO MAIN*: Initialize remaining sensors, calculate control gains and motor input. This is the only file that should me altered if we want some different behavior in the quadcopter
- *Filter*: Contains all the functionality for handling the Kalman filtration of all required sensors and system states
- *Motor*: Initializes the electric speed controllers and set the current motor speed to zero
- *Radio*: Contain functionality for validating incoming messages, handle validated messages, and ensure the stability of the radio link

B.1.5 Motor tester

This is a useful script to test that the motors work properly, here you can insert any PWM to set the speed.

B.1.6 Radio

Contains the core functionality for validating messages and controlling the radio link status.

B.1.7 Ultrasonic Range Sensor

Simple code for reading the range measurements provided by the ultrasonic range sensor.

B.1.8 Voltage Reader

A simple script for assessing the overall voltage of the battery.

B.2 Matlab Code

B.2.1 Matlab Filtering

All Matlab code related to various filter techniques are included within this folder. This includes;

- All files regarding filter comparison made during this report
- Files used for Kalman filtering of the position
- Files used for Kalman filtering of roll and pitch angles
- Files used for Kalman filtering of the yaw angle

B.2.2 Read Data over the Radio

Files required for receiving data from the radio directly into Matlab

B.2.3 Read Data over Serial

This folder includes various Matlab scripts for reading data over a serial cable, this includes;

- DMP filtered values from the IMU
- Both DMP filtered values and raw measurements from the IMU

- Range (or height) measurements
- IMU data and corresponding motor input
- IMU data and magnetometer measurements
- Raw magnetometer measurements
- Kalman filtered roll and pitch angles
- θ input
- Kalman filtered yaw angle

B.2.4 Simulation

Simulation of both PID and LQR controllers conducted last fall is included in this file.

B.3 Windows Application - Unity

The folder that contains the entire Windows application code may look complicated, but this is mostly unity files, and we follow the standard unity convention for storing our files. In order to open the project in unity, the most straightforward way is to open the quadcopter control folder, then open the "Assets" folder, and then the "Scenes" folder, then open the "Scene1" file in Unity.

All our source code files can be foul under; Quadcopter control folder -¿ "Assets" -¿ "Scripts", and they will also be valuable when opening the project in Unity. All pictures used in our project are located under; Quadcopter control folder -¿ "Assets" -¿ "Textures" and are also valuable when opening the project in Unity.

There are only two C# source files for our quadcopter control application:

- *GUIController.cs:* The GUIController file contains all the functionality directly connected towards the creation and response from the visual content in the application. Also the functionality for queue control, and "ack" messages with the corresponding timers are located in this file, and it can thus be viewed upon as the "main" file for the project.
- *SerialComm.cs:* This file contains all the functionality directly concerning the serial communication with the radio device. It contains functions for ensuring that the serial connection is still working after the COM port is opened, functionality for sending and reading messages to and from the radio device as well as functions for opening and closing the designated COM port.