

MASTER

**Controller Design for a Quad-Copter
Disturbance Rejection in Dynamic Wind Regimes**

Albers, T.A.C.J.M.

Award date:
2018

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

MASTER THESIS
SYSTEMS & CONTROL

**Controller Design for a Quad-Copter:
Disturbance Rejection in Dynamic Wind Regimes**

CST 2018.102

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF MECHANICAL ENGINEERING
CONTROL SYSTEMS TECHNOLOGY

Student: T.A.C.J.M. Albers
Thesis supervisor: Dr. D.J. Guerreiro Tomé Antunes

Committee: Dr. Ir. A.G. de Jager
Dr. A. Saccon
Dr. D.J. Guerreiro Tomé Antunes

Eindhoven, October 26, 2018

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct¹.

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

26-10-2018

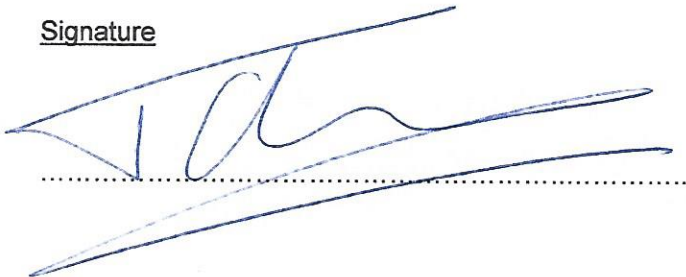
Name

T.A.C.J.M. Albens

ID-number

0264237

Signature



Submit the signed declaration to the student administration of your department.

¹ See: <http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/>

The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU

Abstract

Quad-copters are becoming increasingly more popular and used in many applications. However, currently available quad-copters have limited capabilities in dynamic wind regimes, which limit their applicability to simple tasks, such as crop growth monitoring and package delivery. In this project the design of a testbed to control a fully autonomous drone under wind conditions is carried out and research is conducted to improve the flight behavior of quad-copters in dynamic wind regimes. The developed testbed comprises a TBS quad-copter frame with a Pixhawk as flight management unit, four Tiger Air Gear motor, propeller and ESC sets and a Marvelmind Indoor Positioning System is used to measure the position of the quad-copter during flight. The software designed during this project is based on the PX4 flight stack. However, a new flight control software is developed in Simulink, which provides easy programming and adjusting of the quad-copter flight software. Moreover, drivers are developed in order to integrate the hardware components. A GUI is developed in order to analyze flight data easily. Furthermore, the non-linear dynamic models of a quad-copter are linearized around an equilibrium point and controllers are synthesized. System parameters are estimated based on reconstructing the quad-copter in Solidworks and experiments. Algorithms are developed to calibrate sensors available on the Pixhawk. A madgwick filter and Kalman filter are utilized to filter measurements and obtain unmeasured state estimates. As a result, the testbed is capable of performing flights based on Simulink software and the developed mathematical models.

To enhance the control capabilities in dynamic wind regimes, wind models in the form of Navier-Stokes equations and Dryden Turbulent wind models are analyzed. However, the complexity of the Navier-Stokes equations make the Navier-Stokes equations unsuitable for a model-driven approach to cope with wind regimes. The Dryden Turbulent wind models are based on the low frequency wind components where noise is added to create a stochastic behavior. The simplicity of the Dryden Turbulent wind model makes it suitable for simulation purposes. A solution to cope with dynamic wind regimes is found in the form of disturbance rejection. The novelty of this project is developing controllers based on Internal Model Principle (IMP) methods and Repetitive Control (RC) methods in order to reject wind influences on the quad-copter. The IMP method is based on a model, which rejects a disturbance generated by an exogenous system. The controlled quad-copter is simulated in the hovering state influenced by different wind regimes disturbing the quad-copters' flight. The simulator results show improved flight control during various wind regimes. Noteworthy is the IMP, which even under turbulent wind flows provides an improved flight performance. Although the IMP methods and RC methods show improved flight result in simulation, they are not implemented on the testbed. However, experimental results with the quad-copter hovering are discussed and this project resulted in an easy to program quad-copter, which allows students to conduct their research in future quad-copter projects. Moreover, the experiments conducted in simulation involving IMP method and RC method control structures can be further extended to practical experiments involving a fan, or ideally a wind tunnel.

Preface

This thesis is the final deliverable of my career as a student at the TU/e. Over the past five years, during my pre-master and master I had the privilege to meet and cooperate with extraordinary people from around the world. During my master career at the TU/e I got involved in various challenging projects, from internships abroad involving structural dynamics on NASA launch vehicles to developing 3D ball position estimation for Tech United to developing drone software during my master thesis project. All projects challenged my perseverance and knowledge and my ability to search for creative solutions. All projects helped me develop my personal skills and knowledge in an enjoyable way.

First of all, I would like to thank Duarte Antunes and Alex Andrien, who have both been of great help during my master thesis project. During this master graduation project you provided the freedom to let me develop the project as it is now. The positive attitude during our weekly meetings helped throughout the entire project.

Secondly, I would like to thank Annelore Visker, Amin Talaeizadeh and Chiel Kengen, with which I discussed and talked a lot about the project. Most importantly, with you guys I walked to the canteen to grab a cup of coffee. Those simple walks through Gemini helped me get inspired and look for different approaches for all the challenges this project brought me.

Thirdly, I would like to thank all my friends and family. During my master thesis a lot of long days of hard working behind a laptop were not uncommon. You ensured that during my master thesis project, I still experienced relaxed times.

Lastly, I would like to say special thanks to my girlfriend Carola, which always supported me in times when I needed support. Most importantly, you helped me put everything in perspective in times where I could not find proper solutions to existing problems.

Thank you all for making this thesis possible and helping me through this process.

Tom Albers,

Eindhoven, November 14, 2018

Table of Contents

1	Introduction	1
1.1	Historical background	1
1.2	Motivation and overall aim	1
1.3	Literature survey	2
1.3.1	Controller design for quad-copters	2
1.3.2	Navigation and controller design in wind fields	3
1.3.3	Computational fluid dynamics	4
1.3.4	Quad-copter disturbance rejection	5
1.4	Specific goals, approach and contribution	7
1.4.1	Specific goals	7
1.4.2	Approach	8
1.4.3	Contribution	8
1.5	Organization of the report	9
1.5.1	Part I: Hardware, software and control	9
1.5.2	Part 2: Wind disturbance rejection	10
I	Hardware, Software and Control	11
2	Hardware	12
2.1	Frame	12
2.2	Processing unit	13
2.2.1	Magnetometer	16
2.2.2	Gyroscope	16
2.2.3	Accelerometer	16
2.2.4	Battery sensor	16
2.3	Indoor Positioning System	17
2.4	Motors and electronic speed controllers	19
2.5	Wireless communication	20
2.6	Radio Controller	22
3	Software	24
3.1	Q-ground control and PX4Pro flight stack	24
3.2	Simulink	26
3.2.1	Configuration and timing tools	29
3.2.2	Boot process alternation	30
3.2.3	Quad copter software	31
3.2.4	Ground station software	32
3.3	Configuring remote control	33
3.4	Wireless communication	34
3.5	Indoor Positioning System driver	35
3.5.1	GPGGA message	36
3.5.2	GPRMC message	37
3.5.3	GPVTG message	37
3.5.4	GPZDA message	38
3.6	Matlab Graphical User Interface	38
4	Mathematical model, state estimation and control	40
4.1	Mathematical model	40
4.1.1	Reference frame	40
4.1.2	Non-linear quad-copter model	41
4.1.3	Linearized quad-copter model	42
4.1.4	Motor mixing matrix	42
4.1.5	Propulsion model	43
4.1.6	Sensor Calibration	46
4.2	State estimation	50
4.2.1	Quaternions	50
4.2.2	Madgwick filter	51
4.2.3	Kalman filter	59
4.3	Controller design	61

II	Wind disturbance rejection	64
5	Wind field modeling	65
5.1	Computational Fluid Dynamics	65
5.1.1	Navier-Stokes equations	65
5.1.2	Numerically solving Navier-Stokes equations	67
5.1.3	Boundary conditions	70
5.1.4	Stability analysis	71
5.2	Dryden Turbulent Model	72
6	Disturbance rejection	73
6.1	Continuous time case Internal Model Principle	73
6.1.1	Controller, observer and feedforward design	74
6.1.2	Exogenous System	75
6.1.3	Disturbance rejection	76
6.2	Discrete time Internal Model Principle	78
6.2.1	Controller, observer and feedforward design	78
6.2.2	Exogenous System	79
6.2.3	Disturbance Rejection	80
6.3	Repetitive Control	83
7	Experiments and results	86
7.1	Simulation	86
7.2	Windless conditions	88
7.3	Shear wind	89
7.4	Wind gust	90
7.5	Dryden wind	91
7.6	Dynamical reference	92
7.7	Experimental results	93
8	Conclusion, Remarks and Future work	94
8.1	Conclusion	94
8.1.1	Hardware, Software and Control	94
8.1.2	Wind disturbance rejection	95
8.2	Future work	96
8.2.1	Hardware, Software and Control	96
8.2.2	Wind disturbance rejection	97
	Bibliography	98
	Appendices	102
A	Pixhawk Pinouts	103
B	Tiger Air gear 350 User Manual	105
C	Boot process Pixhawk	107
D	Basic Logged Data Reading	109
E	Setup Raspberry Pi	111
E.1	Configuration Raspberry Pi	111
E.2	Setup Serial 2 Network	111
E.3	Setup Access Point	112
F	Com0Com	117
G	CombyTCP	118

H	GPS Driver Installation	119
H.1	Driver Header file	121
H.2	Driver file	122
H.3	GPS Module	132
I	Marvelmind Configuration	138
I.1	Modem Configuration	138
I.2	Hedgehog Configuration	139
I.3	Beacon Configuration	142
J	Measurement and least squares fit results motors	145
K	Ellipsoid fit	149
L	Inverse thrust mapping	152
M	Solidworks quad-copter model	154
N	Wind conditions and Disturbances	156
O	Position error in windless conditions	157
P	Position error in shear wind conditions	158
Q	Position error in gust wind conditions	159
R	Position error in Dryden wind conditions	160
S	Position error tracking challenging reference	161

List of Figures

2.1	Team Black Sheep Discovery quadcopter frame with DJI F450 Flamewheel arms.	12
2.2	Pixhawk board layout.	13
2.3	3DR Power Module	17
2.4	Marvelmind Indoor Positioning System.	17
2.5	Square pyramid structure representing stationary beacons and hedgehog installation.	18
2.6	Pixhawk hedgehog connection	19
2.7	Tiger Motor Air Gear 350 multi rotor driving equipment set.	19
2.8	Principles of a BLDC motor	20
2.9	3D Robotics Telemetry transmitter/receiver.	21
2.10	Raspberry Pi 3 Model B V1.2	21
2.11	Raspberry Pi GPIO.	21
2.12	FrSky Receiver module.	22
2.13	The FrSky Taranis X9D Plus Radio Control Unit.	23
3.1	PX4Pro firmware architecture.	26
3.2	Boot process Pixhawk flight stack.	30
3.3	Simulink Software block diagram.	31
3.4	Simulink Ground Station Software block diagram.	32
3.5	Communication block diagram from Pixhawk to pc.	35
3.6	Matlab GUI for configuration, evaluation and visualization of the flight data.	38
4.1	Schematic representation of the quad-copter.	40
4.2	Power, current and thrust measurements for motor one with four different propellers.	44
4.3	Least squares fit of second and third order polynomials, including the error.	45
4.4	Sensor calibration, from ellipsoid to orb.	46
4.5	Fitting ellipsoids on measurement data to obtain offset, scaling and rotation parameters.	47
4.6	Magnetometer compensation for motors magnetic fields.	48
4.7	Gyroscope calibration results.	49
4.8	Measurement data and angle estimation quad-copter utilizing gyroscope measurements.	52
4.9	Measurement data and angle estimation quad-copter utilizing gyroscope and accelerometer measurements.	53
4.10	Inclination of Earths magnetic field.	53
4.11	Measurement data and angle estimation quad-copter utilizing gyroscope, accelerometer and magnetometer measurements.	54
4.12	Obtained dip angle θ_{dip} and magnitude $\ m\ _2$ from magnetometer measurements.	55
4.13	Stationary state accelerometer and gyroscope measurements.	57
4.14	Stationary state gyroscope for x -axis only.	57
4.15	Stationary state accelerometer for x -axis only.	58
4.16	Madgwick Adaptive Filter algorithm flow.	58
4.17	Sensor measurements in steady-state conditions and corresponding PDF.	61
5.1	Finite Element Discretization.	68
5.2	Finite Volume Discretization.	69
5.3	Simple body in a flow	70
6.1	Block diagram of a Internal Model Principle controller.	73
6.2	Block diagram of a Repetitive Control Filter.	83
6.3	Examples of Repetitive Control filters.	84
6.4	Examples of Repetitive Control filter $N = 4$	84
7.1	Wind conditions and resulting disturbance force applied to the quad-copter.	87
7.2	Euclidean error in windless conditions.	88
7.3	Euclidean error in shear wind conditions.	89
7.4	Euclidean error in wind gust conditions.	90
7.5	Euclidean error in Dryden wind conditions.	91
7.6	Euclidean error in dynamical conditions.	92
7.7	Results of experiments on testbed with a feedback controller.	93
7.8	Marvelmind sampling delay.	93
B.1	User manual Tiger Air Gear 350 page 1-2.	105
B.2	User manual Tiger Air Gear 350 page 3-4.	105

B.3	User manual Tiger Air Gear 350 page 5-6.	106
F.1	Overview of the com0com software.	117
G.1	Overview of the combyTCP software.	118
J.1	Measurement and least squares fit results motor 1.	145
J.2	Measurement and least squares fit results motor 2.	146
J.3	Measurement and least squares fit results motor 3.	147
J.4	Measurement and least squares fit results motor 4.	148
M.1	TBS Quad-copter reconstructed in Solidworks.	154
N.1	Wind conditions in simulation.	156
N.2	Disturbance moments and forces resulting from wind in simulation.	156
O.1	System dynamics for various controllers in hovering mode without wind.	157
P.1	System dynamics for various controllers in hovering mode in shear wind conditions.	158
Q.1	System dynamics for various controllers in hovering mode in wind gust conditions.	159
R.1	System dynamics for various controllers in hovering mode in Dryden wind conditions.	160
S.1	System dynamics for various controllers tracking challenging reference without wind.	161

List of Tables

2.1	Distances between motors on the quad-copter in millimeters.	12
2.2	Connecting the hedgehog to the Pixhawk.	19
2.3	Connecting the Raspberry Pi to the Pixhawk.	22
2.4	Connecting the FrSky X8R radio receiver to the Pixhawk.	22
3.1	RC Joystick scaling values.	31
3.2	Bitwise Exclusive Or operation.	32
3.3	Remote Control mixer page.	33
3.4	Remote Control switches functionality.	33
3.5	GPGGA message format.	36
3.6	GPRMC message format.	37
3.7	GPVTG message Format.	37
3.8	GPZDA message format.	38
4.1	Motor distances to center of mass.	43
4.2	Motor distances to center of mass.	43
4.3	Least square fit of PWM to thrust utilizing a second order polynomial.	45
4.4	Least square fit of PWM to thrust utilizing a third order polynomial.	45
4.5	Calibration results for magnetometer and accelerometer.	47
4.6	Current calibration results for magnetometer.	48
4.7	Calibration results for gyroscope.	49
4.8	Mean values μ and standard deviations σ of the accelerometer and GPS measurements.	61
4.9	System parameters of the quad-copter.	61
4.10	Control gains controller design quad-copter.	63
6.1	Various signals with their characteristic polynomial, exogenous system matrix representation and eigenvalues.	76
6.2	Various signals with their characteristic polynomial, exogenous system matrix representation and eigenvalues represented in discrete time.	80
7.1	List of parameters used in simulation.	86
7.2	Windless RMSE results.	88
7.3	Shear wind RMSE results.	89
7.4	Wind gust RMSE results.	90
7.5	Dryden wind RMSE results.	91
7.6	Dynamical flight RMSE results.	92
A.1	Pixhawk pin numbers and signal.	104
I.1	Modem configuration Table 1.	138
I.2	Modem configuration Table 2.	139
I.3	Hedgehog configuration Table 1.	139
I.4	Hedgehog configuration Table 2.	140
I.5	Hedgehog configuration Table 3.	141
I.6	Beacons configuration Table 1.	142
I.7	Beacons configuration Table 2.	143
I.8	Beacons configuration Table 3.	144

Listings

3.1	GPGGA example message format.	36
3.2	GPRMC example message format.	37
3.3	GPVTG example message format.	37
3.4	GPZDA example message format.	38
C.1	Pixhawk boot alternation file.	107
D.1	Example Matlab code for reading and plotting the .bin file.	109
D.2	Matlab function to read complete .bin file data.	109
D.3	Matlab function get element type.	109
E.1	Ser2Net installation script.	111
E.2	Raspberry Pi Access Point configuration script.	112
H.1	C make file to compile the GPS driver.	119
H.2	GPS driver type definition declared in the header file.	120
H.3	Marvelmind driver header file.	121
H.4	Marvelmind driver file.	122
H.5	Including Marvelmind header to gps file.	132
H.6	Additional baudrates added to the gps file.	132
H.7	Configuration sequence of the GPS driver.	133
H.8	Debugging information about GPS drivers made available in NSH terminal.	133
H.9	GPS driver loop.	134
H.10	NSH Terminal driver status update request.	135
H.11	Request GPS command information.	135
H.12	NSH Terminal Marvelmind command.	136
K.1	Matlab code for fitting an ellipsoid on data. Obtained from [1].	149
M.1	Mass properties results quad-copter in Solidworks.	155

Chapter 1

Introduction

1.1 Historical background

The origin of Unmanned Aerial Vehicle (UAV) can be traced back to the Military and Government sector, according to a news article from Business Insider [2]. Over the past few years, however, UAVs found their application in the civilian and commercial sector. For example, what was initially a hobby project around 2006 for F. Wang, grew to an 8-billion dollar valued company in 2015 [3] named Da-Jiang Innovations (DJI). Nowadays, the market value for UAV powered solutions is estimated at 127 billion US dollar, according to an article of PricewaterhouseCooper (PwC) [4]. This includes UAV applications in a wide variety of fields of interest, such as infrastructure, agriculture, transportation, security, media and entertainment, insurance, telecommunication, and mining [4],[5].

A specific class of UAVs are the so called multi-rotor drones. The drones are categorized based on the number of rotors varying from three to eight and in some cases even more. The multi-rotor drones have helicopter like behavior, in the sense that the drone is able of vertical take-off and of hovering at fixed positions. This makes the multi-rotor drones ideal for all the tasks referred above and [4]. This master graduate project focuses on four rotor drones. Throughout the remainder of this report, a multi-rotor drone will be referred to as drone or quad-copter.

1.2 Motivation and overall aim

UAVs have an increasingly more important role in nowadays society, such as monitoring dangerous situations and package delivery. New applications and services are becoming available that benefit from deploying UAVs in all weather conditions. These situations demand the ability of flying in extreme wind conditions. However, UAVs are small flying objects. Therefore, UAVs are sensitive to wind gusts and turbulent fluctuations. This does not only limit to global difficult wind conditions, but also to local fast changing winds, such as in urban environments, where wind can change after every building or tree. Apart from urban environments, more situations exists where UAVs are influenced by fast changing winds. An example is inspecting offshore turbines where fast wind changes can be caused by the rotating turbine blades and vortices generated by sunlight heating the sea. This motivates this thesis where the influence of wind disturbances on quad-copters is investigated.

The overall goal of this thesis is to conduct a study on how to compensate for wind disturbances for quad-copters. Both model driven and data-driven approaches are researched, and experimental results are desired. Therefore, an important part of this thesis is setting up the experimental testbed. In particular assembling a quad-copter, integrating the indoor GPS and generating wind. The following section surveys research performed relevant to this thesis: quad-copters, quad-copters controller design, quad-copter control in wind flows, airflow modeling, disturbance rejection specifically on quad-copter and disturbance rejection in general.

1.3 Literature survey

Due to the nature of this thesis, a literature study is conducted varying over a range of topics. Firstly, literature about controller design for quad-copters is discussed. Additionally, the proposed solutions for controlling a quad-copter in wind fields are also addressed. Furthermore, work done on airflow analysis on quad-copters is treated varying from wind tunnel experiments to software simulations. A broader view will be given on airflow analysis by treating computational fluid dynamics and wind field modeling. Apart from airflow analysis, wind can also be seen as a disturbance acting on the quad-copter. Therefore, general disturbance rejection methods will also be considered. In fact, disturbance rejection solutions relying on the Internal Model Principle (IMP), Repetitive Control (RC) and Iterative Learning Control (ILC) and their applicability to quad-copter disturbance rejection methods will be discussed.

1.3.1 Controller design for quad-copters

In [6] a controller is proposed based on the well known Proportional Integral Derivative (PID) structure. First, a non-linear model is developed from which a linear system is extracted. After obtaining a linear model, the PID controller is proposed and evaluated in simulation. As a result, a linear model is obtained for a quad-copter and the linear PID controller stabilizes the quad-copter in flight. Although the article shows proper tracking results, the results were only validated in simulation. Yet the simplicity makes the PID controller a suitable controller.

In [7] a linear controller is proposed based on the Ziegler-Nichols rules. First a feedback linearization is applied to the angular motion of the quad-copter. After feedback linearization, three linear controllers were synthesized and compared for performance. The first controller is a Proportional Derivative (PD) controller tuned by making use of the Ziegler-Nichols rules. The second controller is a PID controller, also tuned making use of the Ziegler-Nichols rules. The third controller is a PD controller tuned by using a genetic algorithm. The paper tests the robust performance of all controllers in simulation by adding zero-mean white-noise. As a result the linear PD controller tuned based on the Ziegler-Nichols rules shows improved performance over the PID controller tuned based on Ziegler-Nichols rules and the PD controller is tuned by a genetic algorithm.

In [8], a comparison between a sliding mode controller, a backstepping controller and a traditional PID controller is made. Stability for the backstepping controller and the sliding mode controller is guaranteed by using Lyapunov equations. All three controllers are tested in a simulation where the performance is measured and analyzed. Both the backstepping controller and switching mode controller show improved performance over the PID controller in case of angle control and height control. However, position control of the switching mode controller shows average results compared to the PID and backstepping controller. Each controller shows the ability to control the linear translations. However, it is recommended to carry out further research.

The article [9] compares quad-copter controller design and in particular a backstepping controller design with a sliding mode controller design. The sliding mode controller shows average results, due to the switching nature of the controller. The switching nature of the sliding mode controller introduces high frequency, low amplitude vibrations, which results in drift. The backstepping controller provides better performance during the experiments. Although results seem promising, the controllers are partially tested in simulation and partially on a test-bench, which is fixed to the world such that the quad-copter can not fly away unexpectedly.

In [10], a backstepping-like feedback linearization method is proposed to control and stabilize a quad-copter. Stability is guaranteed by using Lyapunov stability theorem. The proposed controllers are divided into three sub-controllers which control the quad-copters angles, the height of the quad-copter or the position of the quad-copter. In simulation, the controllers are tested against other non-linear controllers and a PID controller. During simulation the rise time, max overshoot and settling time of all controllers are compared. On the experimental setup, all controllers are evaluated in executing two different tasks. First a takeoff is commanded resulting in hovering in a fixed position. The second task is taking off and following a reference. As a result, the paper has tested their newly non-linear proposed controller in both a simulator and an experimental setup against other controllers. The newly proposed non-linear controller shows satisfactory results. However, testing conditions might have slight changes, which could have effected sensors and controller performances.

In [11], an angle and height backstepping controller is proposed. Lyapunov equations are used to prove stability of the quad-copter. However, additional stability analysis is conducted in case angles reach a specific state in which the height becomes uncontrollable. Furthermore, position control is not considered in [11]. Additionally, an experimental testbed was proposed and the controllers were both tested in simulation and on the testbed. As a result, controller design succeeded to control the quad-copter on the experimental setup. The controller is capable of letting the quad-copter take-off and bring in hovering mode.

This concludes the literature study on controller design for quad-copters. During the controller literature study various controllers have been discussed ranging from, PID, backstepping, sliding mode and non-linear controllers. However, all these papers do not treat the effect of wind on the hardware and the effect on the controllers. The following section will discuss solutions provided in the literature focused on controller design in wind fields.

1.3.2 Navigation and controller design in wind fields

In the previous section, literature focused on controller design is discussed. However, varying wind fields are rarely treated in the literature. Therefore, this section will be focused on work already done on navigation and controller design in the presence of wind fields.

In [12], a navigation and control architecture for multi-rotors in urban wind fields is proposed. The article uses Lyapunov equations to estimate drag coefficients of the drone in a windless environment. The navigation calculates a trajectory, accounting for wind. The trajectory is fed into a backstepping controller which controls the drones on position level. The research shows the wind can be estimated. The estimator tracks the simulated wind closely. However, the procedure is only performed in simulation and complex wind regimes, such as turbulence, is not taken into account.

In [13], navigation in urban environments with changing winds is discussed. This report emphasizes reference generating at which an optimal control problem is presented. The optimal control limits the distance between the drone and generated reference. The article combines a pursuit algorithm with a line-of-sight guidance law in which position and angle errors are kept close to zero. The algorithm is tested in simulation under influence of windless conditions and slowly changing wind. As a result, a way point navigation system using a line-of-sight guidance law is developed to keep the quad-copter close to the trajectory under windy conditions.

In [14], a drone is equipped with a backstepping controller. The position of the drone is obtained by using an external vision system. By using integrators, the drone is navigated to the desired reference under the influence of airflow. The wind is generated by a mechanical fan. In contrast to this report, an integrator is used to compensate for a constant unknown wind disturbance.

Another solution to compensate for wind is by designing an observer, which uses the internal sensors to estimate wind gusts. In [15] an observer based on Global Positioning System (GPS) velocities data and accelerations measured by the IMU were used to estimate the wind gust. The solution shows good results in estimating wind gusts in simulation, but is not tested on an experimental setup. However, a frozen Dryden turbulence model is used to simulate wind gusts. Despite the accuracy obtained in the simulation, the algorithm was only evaluated in a Monte Carlo simulation. Furthermore, the observer was primarily proposed for a flying wing and the ability of observing wind based on the sensors could result in energy harvesting from the wind.

In [16] the wind is estimated by using a recursive Bayesian filter. The whole system uses input/output feedback linearization controller, which estimates a parametric model of the wind field. Furthermore, a Dryden turbulence model is used to simulate wind applied to the quad-copter. Moreover, the research incorporates aerodynamic effects on the vehicle, such as blade flapping and drag as well. Experiments are conducted in simulation. The used recursive Bayesian filter also uses wind measurements in all directions by mounting Pitot tubes. The velocity estimate of the Pitot tubes is compared to the velocity estimates from a GPS and IMU, which results in wind estimations.

Various solutions have been researched on wind field estimation on drones. Most proposed solutions are limited to simulations and make use of simple control strategies such as using integrators. However, under influence of complex wind fields recursive Bayesian filters are proposed. In the section below, a literature study is carried out on Computational Fluid Dynamics, which can be extended to wind field modeling in specific environments to airflow effects on specific hardware parts of the drone.

1.3.3 Computational fluid dynamics

In the previous section, the controller design literature surveyed is studied focused on quad-copters and wind fields. However, most articles only provide results on a simulation basis, where wind is modeled in various different ways. This sections main focus is on Computational Fluid Dynamics, a well-known method for aerodynamic simulations and airflow analysis through complex structured environments. Furthermore, literature about modeling wind regimes will be discussed. Lastly, this section will provide literature on airflow analysis on quad-copters in wind tunnels and in simulations.

Various research has been carried out on airflow modeling [17], [18]. In [17] multiple solutions to Computational Fluid Dynamics (CFD) are explained, including the advantages and disadvantages of each method. Turbulence models used these days and new promising turbulence models are treated as well. In [18] various CFD algorithms are compared, such as Reynolds Averaged Navier-Stokes (RANS), a hybrid RANS, Large-Eddy Simulations (LES) and Detached-Eddy Simulations (DES). All methods are numerically tested and compared. The results show that each model has its own optimal working conditions. Compared to this project, these articles are only limited to indoor airflow, while the UAV is influenced by outdoors wind conditions.

Due to the complexity of CFD algorithms, for simulation purpose it would be beneficial to explore simpler, realistic wind models. The following section treats different ways of modeling the wind compared to real wind.

1.3.3.1 Wind field modeling

The CFD algorithms described above involve complex Navier-Stokes equations and solvers. However, simpler ways of modeling the wind exist and this section will explore literature about other wind models.

In [19] a Computational Fluid Dynamics is designed to simulate wind flow during take-off of a helicopter emergency medical service from a rooftop in Amsterdam. The flow dynamics are simulated by using 2 CFD applications of a flow solver called RANS and the hybrid RANS-LES solver. Furthermore, as turbulence models are the Dryden turbulence models used. The Dryden turbulence models are based on the military graded wind models defined in [20], [21] and [22]. In the simulator, experienced helicopter pilots land the helicopter on a rooftop under turbulent wind conditions. As a result, the experienced pilots rated the turbulence models as "realistic" and sometimes even as "very realistic", despite the simulator being fixed to the ground.

In [23] a model for a wind-gust disturbance is proposed, in order to simulate the pointing accuracy of an antenna and the results are compared against field data. The used method is a so called Davenport Spectrum. The Davenport spectrum can be seen as a white-noise signal filtered through a low frequency bandpass filter. The Davenport wind model is applied to a model in three different manners, wind forces acting on the dish, wind torque acting at the drives and wind acting on the rate input. For all three cases a controller is proposed. Furthermore, the closed-loop pointing accuracy of the antenna is compared over all three controllers. As a result, all three models showed errors that matched field data.

Both [19] and [23] utilize different approaches to model the wind. While [19] uses a CFD including Dryden turbulence models, [23] utilizes a Davenport filter, which acts as a band-pass filter allowing low frequencies to pass through. However, airflow analysis is also conducted on quad-copters. The following section will be focused on airflow analysis applied to drones.

1.3.3.2 Airflow analysis on quad-copters

Besides airflow situations in closed environments, airflow analysis is also performed around a drone. In [24] only the airflow of the rotors is analyzed. The focus is on relating spacing of the rotors to the efficiency of various parts, such as the fuselage and arms. The simulation uses solving 3-Dimensional unstable Navier-Stokes equations, by using a spacial 5th order accurate scheme with double time steps and a Detached-Eddy Simulations (DES) is taken as turbulence model. The article is only limited to simulations and does not involve any type of external airflow.

In 2016 the National Aeronautics and Space Administration (NASA) conducted wind tunnel tests on five different multi-rotor drones and an isolated rotor [25]. The experiment is originally intended to get a better understanding of the performances of multi-rotor drones in general. During the experiment, each drone is in its whole tested on a test rig. After that, the bare frame is tested and only one propeller is tested. The tests are conducted by varying the pitch, yaw, airspeed values and varying the speed of the motors. During the tests, the lift, drag, moments, thrust hover power and isolated rotor electrical efficiency are measured. The contribution of the experiments is to obtain a large data set for various drones and flight conditions, in order to design, analyze and enhance drone modeling and performance. However, during the experiment high frequent vibrations are measured. The vibrations occurred during full frame tests and single propeller tests, which concludes to vibrations resulting from imbalanced propeller motor configurations.

This concludes the literature survey about airflow analysis. In the following section, the wind fields will be treated as a disturbance and the focus of the literature will be on disturbance rejection.

1.3.4 Quad-copter disturbance rejection

Although many solutions are available to estimate or model the wind, effectively the wind disturbance manifests itself as three forces and moments acting on the quad-copter over three different axis. Either an observer needs to be designed for estimating the wind, or the wind should be treated as a disturbance. In case of the latter, a disturbance rejection method can be deployed in the control strategy to reject disturbances applied by the wind on the quad-copter. The following articles are aimed at disturbance rejection on quad-copters in general, not only focused on wind.

In [26] both an PID and backstepping controller is designed for a quad-copter. The PID controller is a conventional controller designed to stabilize the angles of the quad-copter. The backstepping controller contains an additional integral term. Both controllers are evaluated in simulation and on a testbed, which is a gimball fixed to the world at which four propellers control the angles of the gimball. Performance between both controllers is compared based on reference tracking in normal flight, during windy conditions and under parameter uncertainties. Windy conditions are generated by applying an airflow to the testbed by using a wind machine. Furthermore, parameter uncertainty is achieved by adding an additional weight to the quad-copter resulting in a shifted center of gravity. As a result, the integral backstepping controller shows better performance over the PID controller in the sense that the backstepping controller achieves smoother motion and slightly better tracking performance during normal flight. In case of a wind disturbance, both controllers show similar performance. Both controllers stabilize the quad-copter in a robust and efficient manner.

A non-linear controller is designed in [27], while rejecting a constant force disturbance. The solution consists of a backstepping controller, which asymptotically stabilizes the closed-loop system. The constant force disturbance is estimated through the use of a projector operator. Although reference tracking results seem promising, the full state is measured using external hardware, namely a Vicon motion capture system. Furthermore, only a constant force is considered as disturbance.

Although solutions exist specifically designed for quad-copters, the following section treats disturbance rejection in a more general approach, where the main topic is about Internal Model Principle.

1.3.4.1 Internal model principle

In previous section the main focus is on disturbance rejection methods for drones. In this section, Internal Model Principle will be treated. The Internal Model Principle application applies to disturbance rejection for general control systems and it is not focused on quad-copters in particular (although some articles are focused on quad-copters).

Internal Model Principle (IMP) can be used to reject different kinds of signals, such as periodic signals, constant signals or ramps. In [28] an IMP controller is designed in combination with a linear feedback controller and an observer. The IMP controller can be suited with various filter designs which can be used to minimize the tracking error over time. The book, however, only shows the design procedure on a theoretical basis by using for linear time-invariant (LTI) state space system and does not use a quad-copter as a base or provides a practical base at all.

In [29], IMP are used to allow autonomous vertical landing on a deck of a ships, which oscillates in the vertical direction due to high sea states. The results show that an exogenous system can be successfully used to counteract oscillating patterns. However, the control strategy is only applied in vertical direction to counteract a ships deck movement, and not in multiple degrees of freedom.

In [30] an exogenous system is utilized to generate a reference trajectory. The goal of this research is to follow an intruder, which has a specific dynamical movement, which can be generated by an exogenous system. If the quad-copter has arrived at the intruder, it starts to circle around the intruder, which is handled by another exogenous system. The benefit of the control strategy in this research lies in the fact that the controller operates on a priority base, where the stabilizing control law has the highest priority, following the intruder the second highest priority and circling around the intruder has the lowest priority. Furthermore, the IMP controllers are designed in such a way, that they can be placed on top of an already existing controller.

Internal Model Principle allow various filter designs targeting specific frequencies or periodic signals. However, building a memory from previous control inputs can improve current control inputs. This procedure is called Repetitive Control and will be treated in the following section.

1.3.4.2 Repetitive Control

A special case of IMP, is Repetitive Control (RC) [31]. A RC incorporates the control input from previous period to update the current control input. Every time a period still contains an error, the controller tries to improve the control action, such that over time the error is minimized. In [31], the focus lies mainly on servo systems which have repeatable tasks with a known period. Furthermore, the article focuses on proving stability and an increased performance of the servo systems.

In [32], an RC is designed for a system with constraints on the input. The constraints on the input results in unstable behavior, since the controller can not send large values anymore. A model predictive control (MPC) framework is used to design the RC, without the cost of on-line optimization procedures. Furthermore, by using a Fourier analysis of a reference signal or disturbance signal, the repetitive control structure is determined.

RC enables to improve tracking behavior of dynamical systems over time. A more general disturbance rejection method is Iterative Learning Control, which will be treated in the following section.

1.3.4.3 Iterative learning control

An alternative to RC, is Iterative Learning Control (ILC). In [33] it is shown that both RC and ILC can be placed in the same framework. The article compares run-to-run base as well, which is more like an open-loop control design optimization, where the controller is adapted after a batch is finished. Although the framework might suggest the major difference between ILC and RC is either being designed in time domain or in frequency domain, switching between either domains is quite simple. However, the real difference between ILC and RC, is that RC assumes the start of each period is equal to the end of the previous period. The start condition at ILC however is reset after each iteration.

In article [34] the difference between ILC and RC is best shown. Since the tracks on a disk in the hard disk have the tendency to deform during spinning. Therefore, it is difficult to follow the tracks on the disc accurately. This is where ILC comes in. Since each revolution starts at 0 and ends at 2π which is yet again equal to 0, ILC is suitable to track the deformation of the disks since the initial condition of each period can be reset. On the other hand, the needle, which controls the location of the laser is designed in two stages, where mechanical vibrations act on the system and compromise performance. Since RC attacks periodic disturbances, and is not reset after each period, the controller is suitable to improve performance. Although ILC has a proper application for reducing tracking errors, due to the reset nature of ILC, it is not suitable to implement in a quad-copter.

1.4 Specific goals, approach and contribution

As it is described in Section 1.2, the overall aim of this master graduate project is on conducting research on how to compensate for wind disturbances acting on quad-copter. This section describes the specific goals that need to be achieved, the approach to achieve the goals and where this research can contribute to already existing literature.

1.4.1 Specific goals

The overall aim of the thesis is to research solutions to allow drones to fly safely under varying wind regimes. In order to do so, a testbed will be developed to control a quad-copter in windy conditions. The testbed should contain a base including sensors to measure the states of the system. Furthermore, the hardware should provide safety features, such that during experiments, the quad-copter can be operated safely. Additionally, the hardware should allow connecting additional devices and sensors to the need of this and future projects.

Alongside the testbed development, software needs to be designed as well. The software needs to provide a flexible, easy to adjust, base, such that various different control strategies and algorithms can be tested. Furthermore, additional software is needed to analyze flight data after each flight.

Wind can be modeled by using the Navier-Stokes equations as described above. However, the Navier-Stokes equations involve complex mathematical operations. One goal of this thesis is to research the feasibility in using Navier-Stokes equations as a model-driven approach to counteract wind disturbances.

However, a second approach to counteract wind disturbances is to treat them as unmodeled disturbances. This approach treats the wind as a disturbance acting on the drone in the form of three forces and moments. A controller design in the form of Internal Model Principle will be examined. Other suitable disturbance rejection methods, such as Repetitive Control and Iterative Learning Control will be investigated as well.

1.4.2 Approach

The hardware used in this project is mostly provided by the TU/e and consists of a Team Black Sheep quad-copter, which will be equipped with a Pixhawk flight controller board and a Marvelmind Indoor Positioning System. All the hardware related to the drone was already available at the TU/e and fitted the purpose of this project. Therefore, other hardware is not considered during this research.

Furthermore, a complete solution of quad-copter software is available in the form of Q-ground control and the MAVLink communication protocol to communicate in Simulink in real-time with a quad-copter in flight. However, the software in Q-ground control is designed for various different kinds of robots, resulting in a comprehensive software package hard to adjust to the users needs. Therefore, custom software is written, which provides an easier adjustable environment, while still maintaining full control of the quad-copter during flight.

In addition to software development, controllers and observers are synthesized as well. A Madgwick filter is used to fuse the Inertial Measurement Unit measurement together with the magnetometer measurements in order to obtain the full angular state of the quad-copter. The translational positions are measured with the Marvelmind Indoor Positioning System and fused using the accelerometer in a linear Kalman filter. The observed states are used in the controller. A controller is designed based on a traditional PID controller. The synthesized controller stabilizes the linearized system around near-hovering flight modes.

In order to obtain useful information about wind, Navier-Stokes equations can be used to express wind flow around objects. However, Navier-Stokes equations are Partial Differential Equations. In order to make a model-driven approach, the Partial Differential Equation need to be transformed to Ordinary Differential Equation by, for instance, using multiple Pitot tubes located in various evenly spaced locations around a drone. However, due to complexities in Navier-Stokes equations and the lack of large amounts of Pitot tubes, this approach will only be researched on a theoretical basis.

Lastly, a data-driven approach is investigated in terms of Internal Model Principle and Repetitive Control. Both controllers are tested in simulation against a classical feedback PID controller. Simulation consists of multiple scenarios, such as hovering flying modes, reference tracking and hovering under various wind conditions. Desirable, after testing the controllers in simulation are implemented on the testbed and evaluated in actual windy conditions.

1.4.3 Contribution

During this research project, a fully operational drone is developed. The drone consists of a Team Black Sheep [35] base to which Tiger-Motors [36] are mounted. A Marvelmind Indoor Positioning System is connected to the quad-copter in order to measure a three dimensional position. Furthermore, the Radio Control Unit is used to generate a position reference which will be tracked by the quad-copter. Additionally, the Radio Control Unit is used for safety measures during flights, such as arming and dis-arming the quad-copter. The platform will be available for students in the future, which can use the platform for various different research purposes, such as controller design, observer design or way path planning.

In the past a student at TU/e has developed a solution to communicate with quad-copters over MAVLink [37], the solution was too slow to use in a control environment. Therefore, custom software is developed in Simulink. This allows for an easy understandable and adaptable software structure. Furthermore, necessary drivers were developed, in order to let the Marvelmind Indoor Positioning System communicate with the custom software, in order to obtain a 3D-position measurement.

During this research project, a Graphical User Interface is developed to help and understand internally logged signals on the quad-copter. The Graphical User Interface consists of different tools, such that configuration and evaluation of logged data is made easy. The build-in tools consist of sensor calibration tools, Fast-Fourier Transforms of time-varying signals, stored parameters and reference tracking overviews.

In the literature, various solutions are provided in order to reject wind disturbances. However, most literature is only limited to simulation results and rather simple wind models, such as a constant wind

disturbance. Compared to this research, an Internal Model Principle approach is proposed capable of improving reference tracking under complex wind disturbances. The novelty in using this type of controller lies in the fact that Internal Model Principle is as of yet not used on a quad-copter in order to reject wind disturbances. Internal Model Principle is however used to track an intruder by using a quad-copter, under the assumption an intruder can be modeled by an exogenous system [30]. Furthermore, this research even examines Repetitive Control as a solution to counteract wind disturbances. Both controllers are tested safely in a simulator, which contains the quad-copter dynamics and a wind model. As a result, a controller is proposed, which is capable of rejecting the wind. The proposed controller can reject the wind without the need of having external hardware, such as line-of-sight guidance systems. Furthermore, the proposed controller does not need to solve complex Lyapunov functions in order to estimate the drag coefficients in assumable windless conditions. Lastly, the quad-copter is still capable of flying around, compared to testbeds fixed to the real world.

1.5 Organization of the report

The organization of this report is divided in two major parts. Part I is focused on the practical setup. This research makes use of existing hardware at the TU/e. Software is adapted to the hardware and the needs of this master graduate project. Furthermore, a mathematical quad-copter framework is explained, resulting in controller design, observer design and sensor calibration. Part II targets a model-driven approach to handle changing winds. A data-driven approach in the form of disturbance rejection. Lastly, the simulation results are part of the wind disturbance rejection part.

1.5.1 Part I: Hardware, software and control

Chapter 2, Hardware.

In this chapter the hardware used during this project will be discussed. The hardware consist of the quad-copter frame, processing unit, various sensors, motor sets, radio controllers and additional devices. The main focus of the chapter is to show the functionality of each part and how they are connected to each other.

Chapter 3, Software.

This chapter describes the existing software structures and provides information about the important parts of the software. The newly designed Simulink software is explained, which makes use of part of the existing software. Furthermore, basic configuration of the various hardware parts is described as well, such as communication devices and Indoor Positioning System (IPS) drivers. Lastly, the Matlab Graphical User Interface (GUI) especially designed for this project is explained.

Chapter 4, Mathematical model, state estimation and control.

Chapter 4 describes the mathematical model of a quad-copter. Due to the non-linearities of the model, a linear model is first obtained, resulting from standard linearization techniques. Aside from the quad-copter models a propulsion model is considered, which involves measuring the thrust of each propeller. Furthermore, calibration sequences are explained for the most important sensors. Chapter 4 also explains state estimation for a quad-copter in the form of a Madgwick filter and Kalman filter. Lastly, the chapter shows a basic cascaded PID control strategy.

1.5.2 Part 2: Wind disturbance rejection

Chapter 5, Wind field modeling.

In this chapter, relevant literature is described which is based on the quad-copter flight in windy conditions. The chapter treats topics ranging from Computational Fluid Dynamics (CFD), Navier-Stokes equations and their complexities, airflow analysis on quad-copters, navigation and controller design in wind fields, wind field modeling and controller design including disturbance rejection.

Chapter 6, Disturbance rejection.

This chapter explains the basic math for designing Internal Model Principle (IMP) controllers and how to access stability. Furthermore, IMP controllers are divided in continuous time systems and discrete time systems. The procedure to design an IMP contains controller observer and feedforward design, exogenous systems as disturbance generation and disturbance rejection by using a driver model. A special case of IMP called Repetitive Control (RC) is explained as well.

Chapter 7, Experiments and results.

In this chapter five different controllers are tested under various kinds of wind flows. The tested controllers are plain PID feedback controller, IMP controller and various RC filters.

Chapter 8, Conclusion, Remarks and Future work.

The final chapter concludes this research and provides remarks about future improvements on experimental setup side and future work on research side.

Part I

Hardware, Software and Control

Chapter 2

Hardware

In this chapter, a detailed explanation is provided about the hardware used during this master thesis project. The provided hardware was already available at the TU/e. Therefore, no other hardware related to the quad-copter was considered during this project. In Section 2.1 the Team Black Sheep frame will be discussed. A short explanation about the processing unit, its internal sensors and inputs/outputs is provided in Section 2.2. The Indoor Positioning System (IPS) is explained in Section 2.3. Additionally, the motors and Electronic Speed Controller (ESC) are explained in Section 2.4. Furthermore, the communication between the drone and a ground station is explained in detail in Section 2.5. Finally, the Radio Control Unit (RC) and the Radio Control Receiver are explained in Section 2.6.

2.1 Frame

During this project the Team Black Sheep (TBS) Discovery filming rig is used as a base frame. The TBS Discovery is a durable and crash resistant multi-rotor [38]. The frame is built out of a bottom plate, a top plate and multiple spacers which connect the top and bottom plate. Four DJI F450 Flamewheel arms allow the motors to be mounted to the frame. An overview of the frame can be seen in Figure 2.1, where on the left an expanded view of the quad-copter is shown and on the right a top view of the frame is depicted.

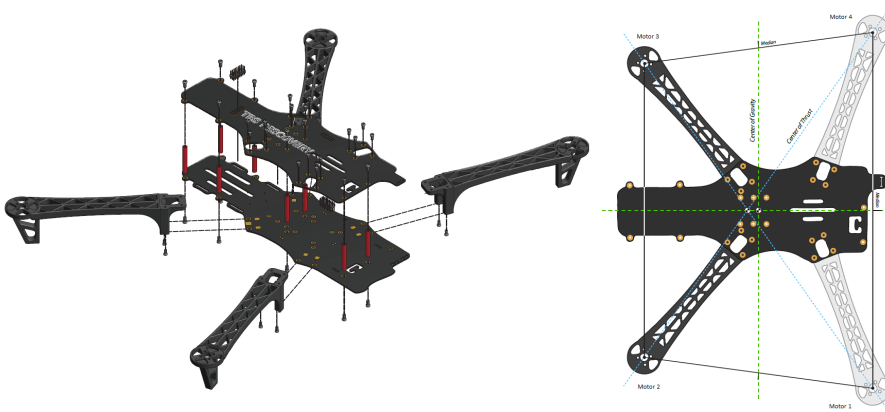


Figure 2.1: Team Black Sheep Discovery frame with DJI F450 Flame Wheel arms. On the left-side, an exploded view of the quad-copter. On the right-side, a top view of the quad-copter. Figure obtained from [38].

Distances motors in [mm]				
	Motor 1	Motor 2	Motor 3	Motor 4
Motor 1	/	274.74	470.37	420.69
Motor 2	274.74	/	346.49	470.37
Motor 3	470.37	346.49	/	274.74
Motor 4	420.69	470.37	274.74	/

Table 2.1: Distances between motors on the quad-copter in millimeters.

The distances between each motor of the TBS quad-copter frame including the Da-Jiang Innovations F450 Flamwheel arms as shown on the right side of Figure 2.1, are listed in Table 2.1. The measured distances between the motors are used to calculate the Center of Thrust (CoT). The distances between

each motor is obtained by reconstructing the quad-copter in Solidworks, a 3D drawing software package, which is provided in Appendix M.

It has to be noted that the median which goes straight through the Center of Gravity (CoG) from the front of the quad-copter to the back, lies in the middle between the left and right side motors. This means, that if the motors on the left side generate an equal amount of thrust compared to the motors on the right side, all generated torques cancel out. However, the median which goes from the left side to the right side of the drone, lies not on the CoG. If motors on the front generate an equal amount of thrust compared to motors on the back, all torques will not cancel out and the quad-copter starts to rotate around the CoG. This concludes that the Center of Thrust (CoT) lies not perfectly on top of the Center of Gravity (CoG). In order to overcome this problem, torques can be distributed to the motors, such that all torques are balanced out and rotations around CoG only exist if requested via the reference. The motor mixing, to balance the requested thrusts and torques, is explained in Section 4.1.4.

2.2 Processing unit

The processing unit used in this configuration, is a Pixhawk 2.4.8. The Pixhawk is developed from APM2 and is released in 2013 by Eidgenössische Technische Hochschule Zürich (ETH Zürich) and 3D Robotics (3DR) [39]. The Pixhawk is a merger between the PX4 Flight Management Unit (PX4-FMU) and the PX4 Input Output unit (PX4-IO). The PX4-FMU was developed for flight management purpose and contains important flight sensors, such as accelerometer and magnetometer. However, the PX4-FMU lacked the connectivity to the motors controlling the propellers. As a result, the PX4-IO was needed to send commands to the motors. Therefore, the merge between the PX4-FMU and the PX4-IO resulted in the Pixhawk containing important flight sensors, as well as connectivity to the motors controlling the propellers.

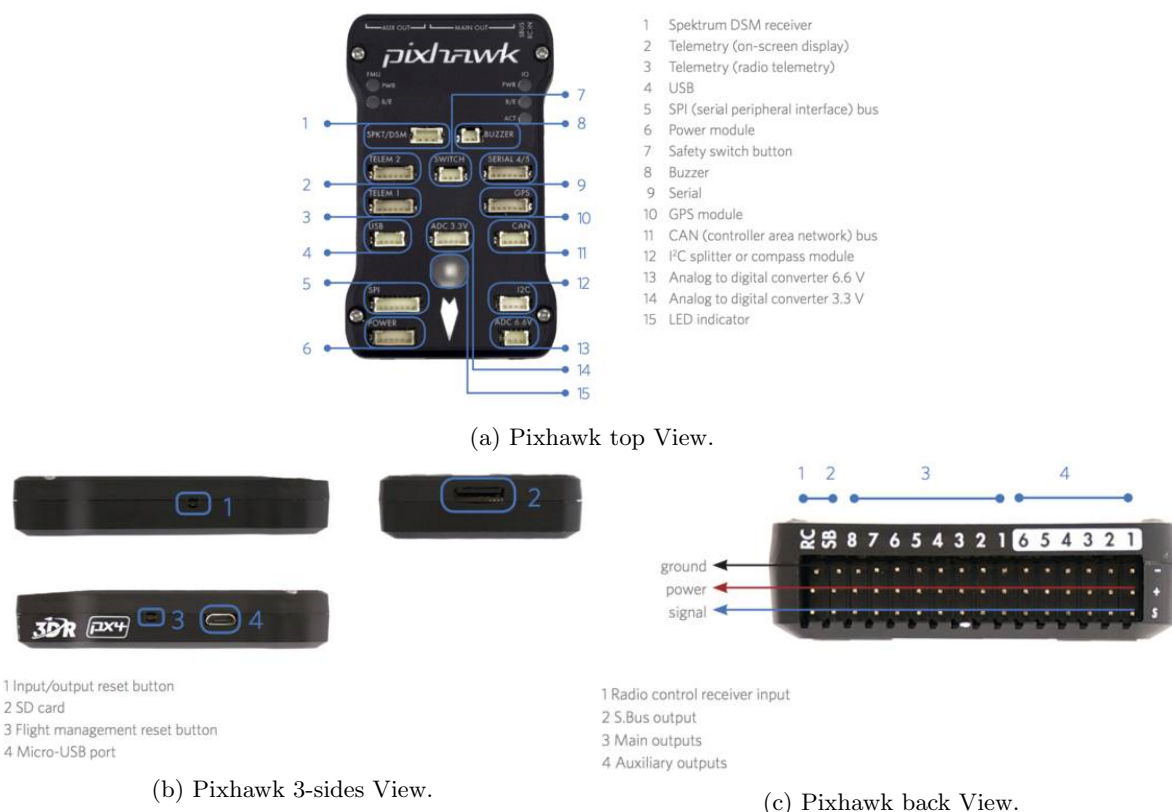


Figure 2.2: Pixhawk layout including indication lights, inputs/outputs, buttons, SD-card slot and micro-USB port. Provided by [40].

The Pixhawk consists of 14 DF-13 connection gates on top, shown in Figure 2.2a. A DF-13 male

connector is a connector which allows to connect a varying number of pins to a DF-13 female connector attached to a wire. The DF-13 connector prevents connecting the cable wrongly, since it only connects in a single way. The DF-13 connector varies in size from two pins for the buzzer to seven pins for the Serial Peripheral Interface (SPI).

SPKT/DSM Connects to a Radio Control Unit (RC) receiver which support Spektrum DSM, DSM2 or DSM-X Satellite.

TELEM1 Connects to a radio data transmitter/receiver to obtain and send data to a ground station, such as a laptop or tablet. TELEM1 supports high power devices up to 1A.

TELEM2 Connects to a radio data transmitter/receiver to obtain and send data to a ground station, such as a laptop or tablet. However, TELEM2 does not support high power devices.

USB Universal Serial Bus (USB) connects to a Future Technology Devices International (FTDI) which is a chipset supporting . Since USB and FTDI is closely related to Universal Asynchronous Receiver/Transmitter (UART) the connector can be used as UART as well. In general, USB is used to connect devices to pc and UART is used to connect hardware to a microcontroller. A UART connection can be connected to a USB port if a bridge and voltage levelers are used. It is however advised not to do so.

SPI Serial Peripheral Interface (SPI) is a synchronic serial datalink between at least two devices in a master slave configuration.

POWER Connects the 3DR power module to the Pixhawk. The power module provides the Pixhawk power and status updates from the battery, such as Voltage and Current measurements.

SWITCH Connects to the safety switch on the quad-copter. Without the safety switch the hardware will not be able to output signals to the motors. The safety switch contains an Light-Emitting Diode (LED), which shows the state of the safety switch.

BUZZER Connects the buzzer to the quad-copter, which allows for audio indications of the system.

SERIAL 4/5 The serial connection port contains two different UART connections to connect various different sensors and or devices.

GPS Global Positioning System (GPS) connects a GPS to the Pixhawk. In this project, the connector will connect the Marvelmind IPS to the Pixhawk.

CAN Controlled Area Network (CAN) connects a CAN device to the Pixhawk.

I2C Inter-Integrated Circuit (I2C) connects multiple I2C devices to the Pixhawk, which allows for bit level communication.

ADC 6.6V Analog Digital Converter (ADC) supplies a 5V power supply to a device and reads out 6.6V analog values.

ADC 3.3V Supplies a 5V power supply to a device and reads out 3.3V analog values. Connector supports up to two devices or analog signals.

The Pixhawk has various communication methods embedded, such as UART, CAN, SPI and I2C. Each communication method allows for different kinds of connections, where CAN and UART supports asynchronous communication (data is send a-periodically). And where CAN and SPI and I2C supports multiple devices on the same connection. However, SPI only operates on bit level, whereas UART, CAN and I2C operate on byte level. In Figure 2.2, a schematic overview is shown of the Pixhawk. The pinlayout of each connection can be found in Appendix A. More information about the various connections can be found at [40].

The Pixhawk contains on top Light-Emitting Diode (LED) which indicate the status of the system. In Figure 2.2a on the top left, indication LEDs for the Flight Management Unit (FMU) are placed and on the right side are the indication LEDs for the Input Output (IO). In the middle of the Pixhawk is the main status LED, which can be programmed and therefore will be explained in other chapters of this report. The indications of the LEDs are:

FMU PWR Indicates the status of the Flight Management Unit power.

FMU B/E Indicates if the processor is in bootloader mode (flashing) or in error mode (solid).

I/O PWR Input/Output power supply status.

I/O B/E Input/Output processor is in bootloader mode (flashing) or in error mode (solid).

ACT Activity, a flashing LED indicating everything is OK.

The Pixhawk contains two buttons, an Secure Digital card (SD-card) and a micro USB connection on the sides and front, Figure 2.2b. The micro USB connection is used to connect the Pixhawk with a computer. This connection provides the ability to load code to the Pixhawk, either custom software or standardized software such as autopilot. During flight the micro USB connection can not be used, since the hardware assumes if it receives power over the micro USB connection it is probably connected to a computer and failsafes will be shut down.

The two buttons are used to reset either the FMU or the IO board. In case of an error, where the FMU B/E LED is on, the FMU reset button located next to the micro USB connector can be pressed to reset the error. In case of an IO error, where the IO B/E is on, the IO reset button can be pressed which is located on the other side than the FMU reset button and the micro USB connector.

The SD-card is located at the front of the Pixhawk. In general, the log files are stored on the SD-card. Later on will be explained that the bootloader can be altered via the SD-card. The FMU can only properly start up if the SD-card is present.

The back of the Pixhawk contains a rack of inputs and outputs, see Figure 2.2c. On the left, the rack contains an RC input to connect the RC receiver to the Pixhawk. The RC works with Pulse Position Modulation (PPM) or Serial Bus (S.Bus). The PPM is an analogue signal like Pulse Width Modulation (PWM) and stacks several PWM like signals onto one wire. S.Bus however is a digitalized signal which can support up to 18 channels using only one cable.

Next to the RC channel is the S.Bus output channel. This port is an output port and can therefore not be used as an RC input. The output channel can be used to send data from the Pixhawk to the receiver and send it back the RC Controller.

Next to the S.Bus output channel are 8 servo pins, which can be connected to 8 different Electronic Speed Controller (ESC). The pins send PWM signals which are transformed by the ESC to desired angular velocity of the propellers.

The last six pins are auxiliary outputs. These outputs operate on PWM signals which can be sent to connected devices capable of handling PWM signals.

The Pixhawk contains internal sensors which are useful during flight. In some cases, such as the accelerometer and gyroscope, the sensors are redundantly installed on the board, such that sensor malfunction during flight will not result in an immediate crash of the quad-copter. The internal sensors are explained more extensively in the following subsections.

2.2.1 Magnetometer

The Pixhawk contains an onboard magnetometer based on a 14 bit ST Mirco LSM303D chip [41]. The sensor has 3 magnetic field channels, which can be scaled between $\pm 2/\pm 4/\pm 8/\pm 12$ gauss. Configuring the magnetometer is done by the software installed on the Pixhawk, which is explained later on in the report. The magnetometer can be sampled in normal mode at a rate of 100 kHz and in a fast mode at 400 kHz. A temperature sensor measures the environment temperature and corrects the magnetic channels for temperature influences. The normal operating temperature lies in the range between -40°C and $+85^{\circ}\text{C}$.

2.2.2 Gyroscope

The Pixhawk contains an onboard gyroscope based on a 14 bit ST Mirco L3GD20H chip [42]. The sensor has 3 gyroscopic measurement channels, which can be scaled between $\pm 245/\pm 500/\pm 8/\pm 2000^{\circ}\text{C}/\text{sec}$ degrees per second (dps). Configuring the gyroscope is done by the software installed on the Pixhawk, which is explained later on in the report. The gyroscope sampling frequency can be configured between 11.9/23.7/47.3/94.7/189.4/378.8/757.6Hz. A temperature sensor measures the environment temperature and corrects the magnetic channels for temperature influences. The normal operating temperature lies in the range between -40°C and $+85^{\circ}\text{C}$.

The Pixhawk contains a secondary onboard gyroscope based on a 16 bit InvenSense MPU_6000 chip [43]. The sensor has 3-axis gyroscope measurement channels, which can be scaled between $\pm 250/\pm 500/\pm 1000/\pm 2000^{\circ}/\text{sec}$ dps. Configuring the gyroscope is done by the software installed on the Pixhawk, which is explained later on in the report. The gyroscope can be sampled via I2C at 400 kHz or SPI at 1 MHz. In some cases, by configuring the chip properly, the sampling rate at SPI can be set at 20 MHz. A temperature sensor measures the environment temperature and corrects the gyroscope measurements for temperature influences. The normal operating temperature lies in the range between -40°C and $+85^{\circ}\text{C}$.

2.2.3 Accelerometer

The Pixhawk contains an onboard accelerometer based on a 14 bit ST Mirco LSM303D chip [41]. The sensor has 3 linear acceleration channels, which can be scaled between $\pm 2/\pm 4/\pm 6/\pm 8/\pm 16$ gravitational acceleration (G). Configuring the accelerometer is done by the software installed on the pixhawk, which is explained later on in the report. The accelerometer can be sampled in normal mode at a rate of 100 kHz and in a fast mode at 400 kHz. A temperature sensor measures the environment temperature and corrects the acceleration measurements for temperature influences. The normal operating temperature lies in the range between -40°C and $+85^{\circ}\text{C}$.

The Pixhawk contains a secondary onboard accelerometer based on a 16 bit InvenSense MPU_6000 chip [43]. The sensor has 3 linear acceleration channels, which can be scaled between $\pm 2/\pm 4/\pm 8/\pm 16$ G. Configuring the accelerometer is done by the software installed on the Pixhawk, which is explained later on in the report. The accelerometer can be sampled via I2C at 400 kHz or SPI at 1 MHz. In some cases, by configuring the chip properly, the sampling rate at SPI can be set at 20 MHz. A temperature sensor measures the environment temperature and corrects the acceleration measurements for temperature influences. The normal operating temperature lies in the range between -40°C and $+85^{\circ}\text{C}$.

2.2.4 Battery sensor

On the power input connection on the Pixhawk, DF-13 input nr. 6 in Figure 2.2a, a 3DR power module, see Figure 2.3, is connected which provides the Pixhawk with power from the battery. The power module protects the Pixhawk from a brownout which is an unintentionally voltage drop in the power supply which shuts the Pixhawk down during flight. It also provides stable currents of 2.25A at 5.37V to the Pixhawk. Lastly, the power consumption of the quad-copter and the drawn current is measured and communicated with the Pixhawk. This allows the Pixhawk to correct the magnetometer for magnetic disturbances generated by current flow for instance generated by the motors and the ESC.

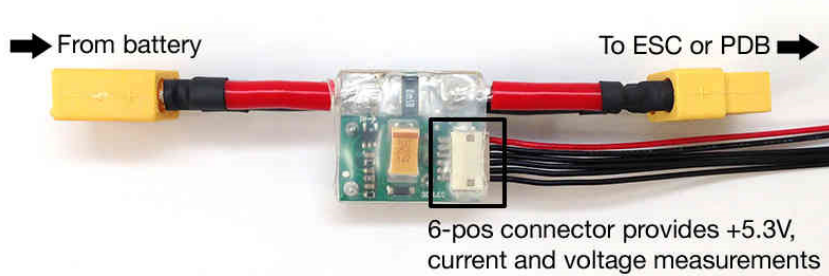


Figure 2.3: 3DR Power module for supplying power to Pixhawk. 3DR Power module includes status information of battery such as voltage and current measurements. Provided by [44].

2.3 Indoor Positioning System

In order to obtain a 3D position of the quad-copter, an Indoor Positioning System (IPS) called Marvelmind is used. The Marvelmind IPS consists of five beacons and a modem, as is shown in Figure 2.4a. Four beacons are used as stationary beacons and one beacon is placed on the quad-copter and configured as a mobile beacon (hedgehog).



(a) Marvelmind kit. Provided by [45].

(b) Marvelmind beacon. Provided by [45].

Figure 2.4: Marvelmind Indoor Positioning System. Setup exists of one modem and five beacons of which one can be setup as a hedgehog. Provided by [45].

Each beacon sends ultrasonic pulses in five different directions. On the right side of Figure 2.4b, all transmitting directions are marked from RX1, RX2, RX3, RX4 and RX5, which covers 360 degrees around the beacon and the top of the beacon. Only the bottom of the beacon is not able to transmit ultrasound pulses. By picking up the pulses and measuring the time of flight, the position of the hedgehog can be trilaterated. However, the system has a few downsides which need to be considered during installation and operation.

First of all, the system operates under the assumption, that the speed of sound through air will not change during operation. The function for the speed of sound, as proposed by [46], can be seen in Equation 2.1.

$$c_{ideal} = \sqrt{\frac{\gamma RT}{M}} \quad (2.1)$$

Where $c_{ideal} \in \mathbb{R}$ is the speed of sound in an ideal gas, $\gamma = \frac{C_p}{C_v} \in \mathbb{R}_+$ is the ratio of specific heats where C_p

is the specific heat at constant pressure and C_v is the specific heat at constant volume. $R = 8314.5 \frac{\text{J}}{\text{mol K}}$ is the molar gas constant, $T \in \mathbb{R}_+$ is the temperature in Kelvin and $M \in \mathbb{R}_+$ is the molar mass. Since most parameters are constant in Equation 2.1, the speed of sound c_{ideal} varies depending on the temperature T . In order to prevent large fluctuations in position estimating, the Marvelmind beacons are setup with temperature sensors to correct for temperature fluctuations. To protect the beacons even more against temperature fluctuations, the stationary beacons should not be placed close to temperature changing objects, such as radiators or air conditioning systems. Furthermore, installing beacons in the vicinity of metal objects might lead to reflections of the ultrasound signal and can therefore be disturbed.

For optimal position measuring performance, the stationary beacons and the hedgehog should be installed and kept during operation in a square or rectangular pyramid like structure, see Figure 2.5. The rectangular base of the pyramids can be seen as the square formed between the beacons on a fixed level. The top of the pyramid represents the hedgehog which is installed on the quad-copter. The stationary beacons can be installed on a fixed height above the ground, or on the ground. In case the stationary beacons are installed on the ground, the hedgehog should be installed on the bottom of the quad-copter facing downwards and in all directions horizontally. In case the stationary beacons are installed at a fixed height, the hedgehog should be installed on top of the drone facing upwards and in all directions horizontally. During flight position measuring is compromised if the hedgehog is close to the same plane as the stationary beacons. The Marvelmind IPS utilizes trilateration to estimate the position of the hedgehog, which is obtaining a position by measuring the distance to the stationary beacons. Therefore, if the hedgehog is close to the plane of the stationary beacons, any small distance measurement error from the hedgehog to a stationary beacon results in large position estimation error according to [45]. Therefore, it is advised to install the beacons on a fixed height instead of the ground. Furthermore, the plane at which the stationary beacons are installed, should never be reached during flight with the hedgehog. A full explanation of installing the stationary beacons and the hedgehog can be found in [45].

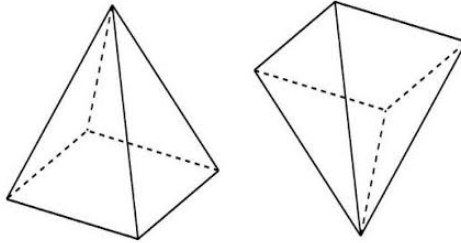
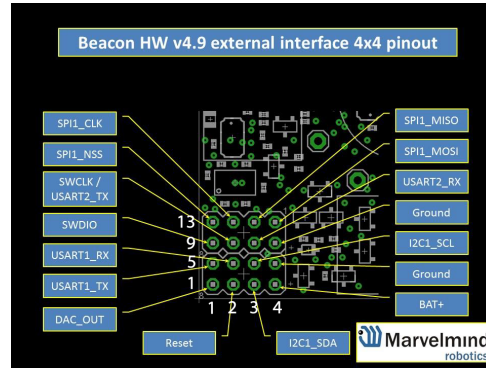


Figure 2.5: Square pyramid structure representing stationary beacons and hedgehog installation.

The Pixhawk and hedgehog are connected via the DF-13 GPS connector and the pins on the hedgehog. The connection is established via UART communication which is supported by both devices. First of all, the ground between both devices can be connected. To let data flow from the hedgehog to the Pixhawk, the transmit line (TX) pin from the hedgehog needs to be connected to the receive line (RX) pin on the Pixhawk. Data flow from the Pixhawk to the hedgehog is established by connecting the TX pin on the Pixhawk with the RX pin on the hedgehog. Figure 2.6a shows which gate and pins on the Pixhawk are used, whereas Figure 2.6b shows which pins are used on the hedgehog. Table 2.2 shows which pins are connected. Since both devices have their own power cell no connection is made between the vcc at the Pixhawk and the Bat+ at the hedgehog. Connecting the Pixhawk and the hedgehog is also explained in [47].



(a) Pixhawk.



(b) hedgehog. Provided by [45].

Figure 2.6: Connecting the hedgehog to the Pixhawk using the pins on the hedgehog and the GPS pins on the Pixhawk.

	Pixhawk GPS		
Pin	6	3	2
Signal	GND	RX	TX
Signal	GND	USART2_TX	USART2_RX
Pin	12	10	11
	hedgehog		

Table 2.2: Connecting the hedgehog to the Pixhawk.

2.4 Motors and electronic speed controllers

The quad-copter is equipped with Tiger Motor Air gear 350 multi rotor driving equipment set, see Figure 2.7. The set consists of:

- 4 Electronic Speed Controller (ESC).
- 2 clockwise (CW) rotating motors.
- 2 counter clockwise (CCW) rotating motors.
- 2 clockwise (CW) propellers.
- 2 counter clockwise (CCW) propellers.



Figure 2.7: Tiger Motor Air Gear 350 multi rotor driving equipment set including four ESC motors and propellers.

The propellers are connected to the motors in a clockwise or counter clockwise fashion depending on the rotation of the motor. If the motor spins clockwise the propeller is attached in counter clockwise rotation which prevents the propeller detaching during flight. In case of the counter clockwise situation, the propeller is attached in clockwise fashion.

The motors are brushless direct current (BLDC) motors which operate on direct current (DC). In general BLDC motors do not require significant maintenance and have an excellent torque weight ratio, which make BLDC motors suitable for the quad-copter propulsion system. A BLDC motor works based on the principle of applying a current to a set of coils, which generate a magnetic field. The current is applied in such a way that the intensity of the magnetic field varies periodically on each fixed motor point. This periodically changing magnetic field is applied to a permanent magnet attached to the rotor which also moves periodically. The process, which generates the proper electric current, is called commutation and is performed by the Electronic Speed Controller (ESC). The working principle and operation of a BLDC motor is depicted in Figure 2.8. By switching the current faster or slower the motor velocity increase or decrease. A more detailed explanation of the principles of a BLDC motor can be found in [48].

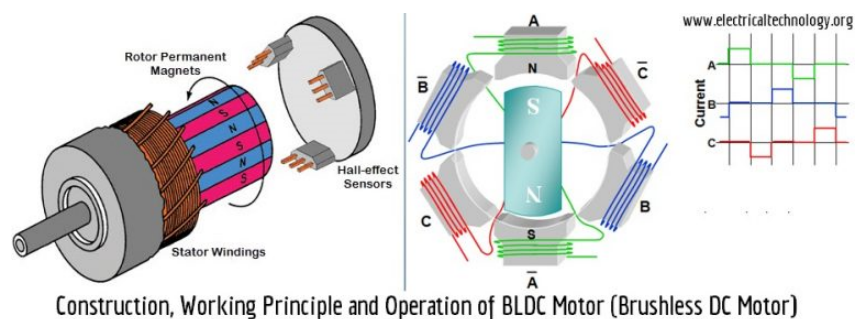


Figure 2.8: Principles of a BLDC motor. Retrieved and adapted from [48].

As just mentioned, the ESC controls the commutating process of the current flowing through the coils. Therefore three gates of the ESC connect to the motor. The desired velocity is provided by the PWM signals obtained from the Pixhawk in the range of +0V and +5V. The ESC, in turn, takes the battery voltage at +15V commutates it in order to bring the motor to the desired velocity. In Appendix B the Tiger Air gear 350 user manual is included for additional information.

2.5 Wireless communication

The communication between the quad-copter and a ground station device is in general provided by a radio telemetry module such as from 3D Robotics (3DR), which is shown in Figure 2.9. The downside of those units is the fact that they are generally equipped with the European 433Mhz [49] band or the American 915Mhz standard. However, the frequency might interfere with the Marvelmind IPS, since the Marvelmind is equipped with 433Mhz for European devices and 915Mhz for the American variant. Therefore standardized telemetry radio units are not used. Moreover, the data transmission from the quad-copter to the ground pc resulted in a ground station pc incapable of accepting user commands anymore.

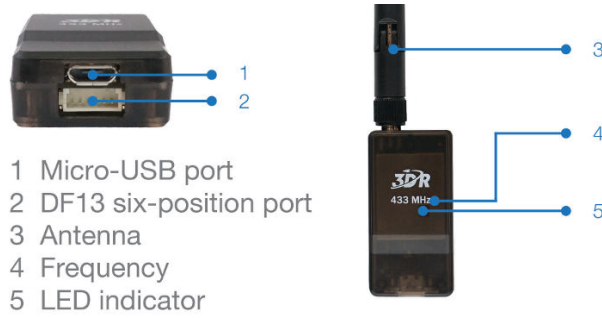


Figure 2.9: 3D Robotics Telemetry transmitter/receiver.

The quad-copter is equipped with a Raspberry Pi 3 model B V1.2 which has a wireless local area network (WLAN). The Raspberry Pi is set up such that it accepts UART communication from the Pixhawk, build a wireless network and send data from to the ground station computer and vice versa. The Raspberry Pi is shown in Figure 2.10.

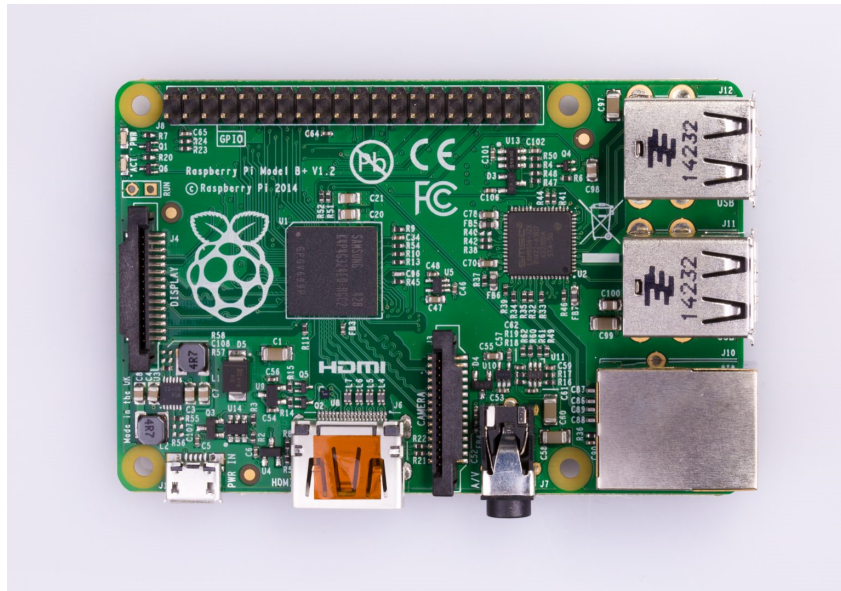


Figure 2.10: Raspberry Pi 3 Model B V1.2

The rack of pins in the top left corner of Figure 2.10 are the header pins which are general purpose input output (GPIO) pins of the Raspberry Pi. Those pins are used to supply the Raspberry Pi at least +4.8V. Besides supplying power to the Raspberry Pi, the GPIO pins are also used to setup an UART connection between the Pixhawk and the Raspberry Pi, over which data is communicated between each device. The pin layout of the Raspberry Pi is shown in Figure 2.11.

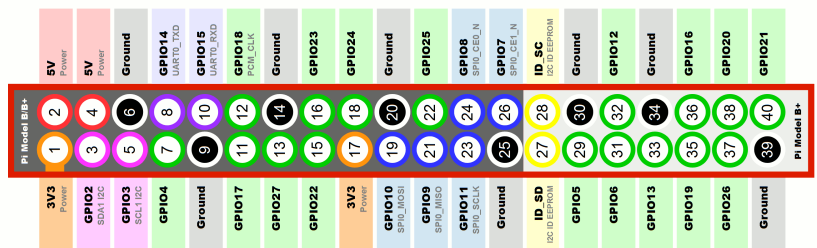


Figure 2.11: Raspberry Pi header pins including general purpose input output pin layout.

The connection between the Raspberry Pi and the Pixhawk is made by connecting pin 2 on TELEM1 TX on the Pixhawk to pin 10 (UART0_RX) on the Raspberry Pi. And by connecting pin 3 RX on the Pixhawk to pin 8 (UART0_TX) on the Raspberry Pi. The full connection is shown in Table 2.3.

	Pixhawk TELEM1			
Pin	1	2	3	6
Signal	Vcc	TX	RX	GND
Signal	+5V	UART0_RX	UART0_TX	GND
Pin	2	10	8	6
	Raspberry Pi GPIO			

Table 2.3: Connecting the Raspberry Pi to the Pixhawk.

2.6 Radio Controller

The last part of the hardware is the Radio Control Unit (RC) and the radio receiver. The RC provides the user to control flight modes and safety procedures available on the quad-copter. The RC is mainly used to arm and dis-arm the quad-copter, and provide a reference commands. The radio receiver is connected to the Pixhawk via the S.Bus port. The S.Bus port is supported by the Pixhawk and provides a digital connection with 16 channels over a single wire. The FrSky X8R receiver module can be seen in Figure 2.12.

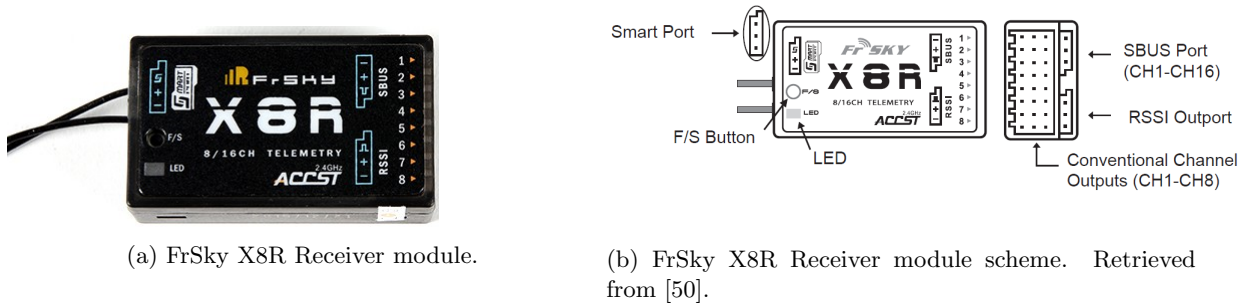


Figure 2.12: FrSky Receiver module.

In Figure 2.12b on the right side the S.Bus port is shown. From top to bottom, the pins are ground, power and signal. The pins are connected to the RC IN on the Pixhawk, see Figure 2.2c, where the top pin is ground, the middle pin is power and the bottom pin is signal. The connection is also shown in Table 2.4. In [50] a more extensive explanation of wiring the FrSky X8R radio receiver to the Pixhawk and its capabilities is provided.

	Pixhawk RC IN		
Pin	-	+	s
Signal	Ground	Power	Signal
Signal	Ground	Power	Signal
Pin	-	+	s
	FrSky X8R Radio receiver		

Table 2.4: Connecting the FrSky X8R radio receiver to the Pixhawk.

The radio receiver is wireless connected to the FrSky Taranis X9D Plus RC. Via the RC various commands can be sent to the quad-copter such as switching flight modes, throttling and applying references to the angles.

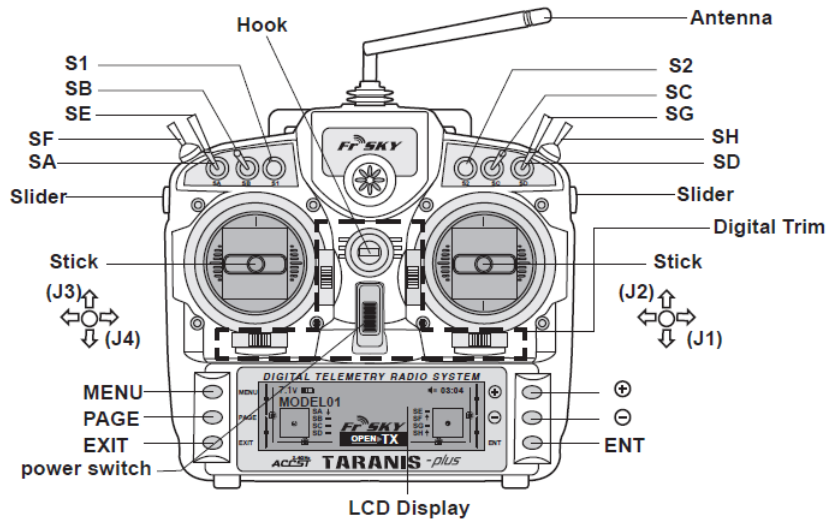


Figure 2.13: The FrSky Taranis X9D Plus Radio Control Unit.

In general, joystick J3 is used as throttle, joystick J4 is used to rotate the quad-copter around the yaw axis, joystick J2 is used to move forward and backward and joystick J1 is used to move left or right. Via the MENU, PAGE, EXIT, +, - and ENT buttons, the controller can be configured via various different menus on the LCD display. Furthermore, switches SA through SH can be configured to put the quad-copter into specific operation modes, such as start logging data, configuration mode or various different controller schemes.

Examples of such controller schemes are normal hover mode, where thrust and the angles are controlled, altitude hold, where joystick J3 controls height, or position hold, where J3 controls height, J4 controls the rotation around the quad-copter, J2 controls in forward movement and J1 control sideways movement. A brief explanation of all the buttons and joysticks can be found in [51]. A better explanation of configuring and operating the FrSky Taranis X9D Plus will be explained in Section 3.3.

Chapter 3

Software

In the previous chapter, the hardware related to this master thesis project is discussed. In this chapter, the software which enables the quad-copter to fly autonomously is discussed. The software to control the quad-copter is based on Q-ground control, Matlab Simulink and custom made software. This chapter describes all the software components and configuration of the software components. In Section 3.1 the modules present in the Q-ground control flight stack are discussed. Section 3.2 discusses the designed software and configuration of the Pixhawk and Simulink. Section 3.3 provides information on configuration of the Radio Control Unit (RC), such that all the functionality of the quad-copter can be accessed via the RC. In Section 3.4, the configuration of the Raspberry Pi and the additional software packages are discussed, such that data can be transmitted between the ground station pc and the quad-copter. Furthermore, Section 3.5 explains the data messages broadcast by the Marvelmind IPS, which are used in the developed driver to obtain position updates. Lastly, Section 3.6 discusses, the Graphical User Interface (GUI), which helps analyzing logged flight data of the quad-copter.

3.1 Q-ground control and PX4Pro flight stack

Q-ground control is a software platform to load and configure firmware on various types of flight machines, such as helicopters, flying wings and quad-copters. Q-ground control loads the so called Px4Pro firmware on the flight controller, in case of this project a Pixhawk described in Section 2.2. An overview of all the software components available in the firmware is shown in Figure 3.1. The software architecture consists of five major different parts:

Storage The storage part contains three different storage modules able of storing data or parameters in various different locations depending on the needs of the software.

Database The first storage module is a database maintained on the SD-card where mission data and flight specific data is stored. Examples of flight specific data are configured flight missions and Geofence, which is a software solution to contain the quad-copter inside a virtual box defined by Global Positioning System (GPS) locations.

Parameters The second storage module stores parameters on the Electrically Erasable Programmable Read-Only Memory (EEPROM) of the Flight Management Unit, which is in this project the Pixhawk. In general, the parameters are set during configuration. After each boot sequence, the parameters are stored in the EEPROM, which provides rapid access to the parameters during flight.

Logging The third storage module stores mostly flight data to the SD-card. If desired, the data can be requested and via Micro Air Vehicle Communication Protocol (MAVLink) and shown on a user device, such as a computer or tablet. Examples are sensor data, or RC inputs.

External Connectivity The firmware has also the ability to communicate with external devices. This allows for parameter tuning during configuration without re-uploading the firmware, or to visualize various signals of the quad-copter during flight. The external communication can be handled by two different modules:

MAVLink is a Micro Air Vehicle Communication Protocol (MAVLink) which allows communication with drones. MAVLink is an efficient communication protocol which uses at least 8 bytes overhead [52].

FastRTPS or Fast Real Time Publish Subscribe (FastRTPS) is a communication protocol which is designed and maintained by Object Management Group (OMG). FastRTPS is a standard widely used in aerospace, defense and Internet of Things (IoT) applications. FastRTPS is also adopted for the Robot Operating System (ROS) robotics toolkit which is widely used for open source robotics projects. FastRTPS can also be used for offboard applications, such as robotics and simulator tools.

Drivers The drivers are a collection of software modules which allow for connectivity between various hardware parts and the main board. Examples of such drivers are for instance GPS and Inertial Measurement Unit (IMU) drivers. Some hardware parts are not used, such as cameras and are therefore not explained.

GPS allow for the configuration and connectivity of various different GPS units. This project makes use of the Marvelmind IPS which needs a custom driver which is explained in Section 3.5.

RC Input configures and processes RC data. Configuration holds information about the hardware connection between the RC and the main board, described in Chapter 2. Furthermore, depending on the communication structure of the RC, the driver converts it to usable data.

IMU Drivers The Inertial Measurement Unit (IMU) driver handles the configuration and data processing of the IMU.

Message Bus The message bus is the center of all software. The message bus is called Ubiquitous Object Request Broker (uORB) which is an asynchronous publish/subscribe messaging Application Programming Interface (API). uORB consists of message structures in which modules, for instance drivers, can publish their data. Every module subscribed to the message can obtain the data from the uORB bus. This procedure is regardless of update frequency and a full visual graph of all topics and connections to subscribers and publishers can be found at [53]. More general information about uORB can be found in [54].

Flight Control The Flight Control part holds the complete control structure as how it is originally designed in Q-ground control. The flight control part hold state machines, state estimators and various different controllers, such as rate control, angle control and position control. The modules included in the flight control part are:

- State Machine
- Autonomous Flight
- Position Controller
- Attitude & Rate Controller
- Output Driver
- Sensors Hub
- Position & Attitude Estimator

All five different parts are shown in Figure 3.1, where the relation between each part is shown. The most important part of the software, as can be seen in the Figure 3.1, is the message bus uORB, which allows for data flow between all the different publishers and subscribers.

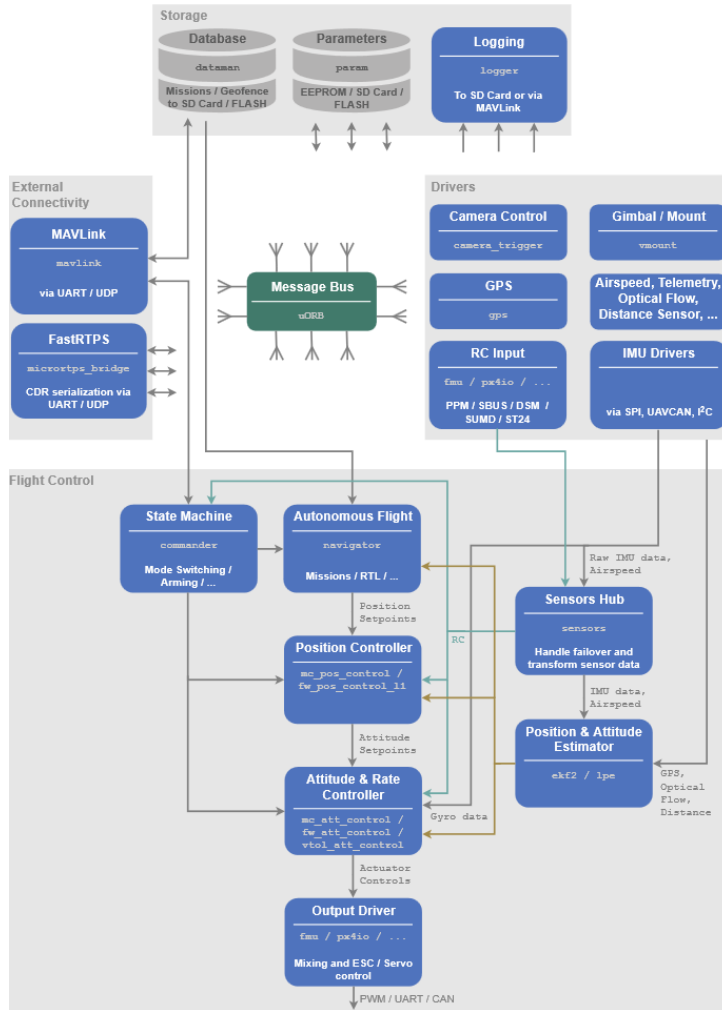


Figure 3.1: PX4Pro firmware architecture. Retrieved from [55].

The software as presented above can be uploaded to the Pixhawk via Q-ground control. The Q-ground control allows for initial sensor calibration and hardware setup for devices connected to the Pixhawk. Detailing the Q-ground control software is out of scope and will not be treated in this thesis. Furthermore, multiple tutorials exist on-line on how to setup and calibrate a quad-copter using Q-ground control.

3.2 Simulink

The flight control part shown in Figure 3.1 is replaced with software designed in Simulink. The most important reason for replacing the flight control part by Simulink custom software is the nature of this project. In fact one of the main objectives is to provide software to control the quad-copter autonomously, and another main objective is to provide algorithms to counteract wind disturbances. To test these algorithms, the flight controller must be modified. Moreover, it is hard to understand exactly what kind of controllers are used in the standard Q-ground control software. As a beneficial side effect, Simulink allows for an easy to understand visual programming scheme, which will ease the future uses of the software.

Before the quad-copter software can be programmed, additional installed software is needed. This can all be found in the manual [56]. Additional software needed consists of the Pixhawk toolchain, CMake and Q-ground control. A full description on how to install the necessary software can be found in the manual [56].

Despite the software packages, Simulink requires multiple toolboxes in order to operate properly. In

the list below, the additional toolboxes are shown. All these toolboxes are necessary, except for the Aerospace Blockset. The Aerospace blockset is used in an example software provided by Matlab. Therefore, it is advised to install the Aerospace blockset, such that those examples can run as well.

- Matlab R2016a/R2016b
- Simulink
- Embedded Coder
- Matlab Coder
- Simulink Coder
- Aerospace Blockset
- Simulink Control Design

After installing various software parts, additional Pixhawk target blocks are provided in the Simulink library. Those blocks allow connectivity between the Simulink software, various Q-ground control software and the hardware. The following blocks are provided by Mathworks Pilot Support Package (PSP)

input_rc This block provides access to the RC signals sent to the quad-copter receiver, which are obtained by the Pixhawk. Various functions can be bounded to the signals, which are explained in Section 3.3. The block provides settings for sample time and tickboxes for selecting specific output signals. The block supports up to 8 channels and various info about the RC connectivity, such as the Received Signal Strength Indicator (RSSI) and failsafes.

PWM_output Is the block that outputs Pulse Width Modulation (PWM) values to the motors. Setting up this block only consists of setting the PWM update rate, which can be set to $\{50, 125, 250, 300, 400\}$ Hz. Furthermore, the block supports up to 8 motors, which means multi-copters with different motor configurations, such as hexa-copters and octa-copters are supported as well. Moreover, one input of the block, called ARM output, is boolean valued and enables/disables all outputted Pulse Width Modulation (PWM) values to the motors.

Speaker_Tune Is a block that outputs tunes to the speaker. In general, the speaker can be used as system indication. The block has 12 pre-defined tunes and custom tunes can be played as well. A trigger input activates the selected tune.

RGB_LED The RGB_LED block manipulates the main Red-Green-Blue (RGB) LED on the Pixhawk. The main usage is system indication. The block can be switched between 15 different colors and 7 different modes.

sensor_combined Is a block that obtains data from most of the sensors, such as magnetometer, accelerometer, gyroscope and barometer. The px4io service needs to be running in order to let the block provide valid signals.

vehicle_attitude The vehicle_attitude block provides signals from the vehicle_attitude uORB topic. The attitude is based on sensor measurements, the selected observer, which can be an extended Kalman Filter or an $SO(3)$ attitude estimator and the sensor calibration. Due to the uncertainties of this block, an attitude observer is designed separately in Simulink and this block is not used.

vehicle_gps The vehicle_gps block provides signals obtained from the vehicle_gps_position topic. GPS modules communicate on the NMEA 0183 protocol, which is a GPS specific message structure to send data from a GPS module to other devices, provided by the National Marine Electronics Association (NMEA). Each GPS module sends specific messages, which are following the NMEA 0183 protocol. Depending on the content of the messages, the uORB topic will be published by the GPS driver and obtained by the Simulink block. The block provides a position timestamp, altitude, longitude, latitude, GPS timestamp, velocity, fix type and number of satellites. However, the availability of the information depends on the installed GPS module.

battery_measure This block monitors the health of the connected battery. If the battery_status uORB topic is published, the block provides information about voltage, low pass filtered voltage, current, electric charge and a timestamp.

binary_logger provides the ability to log data to the SD-card. The block has two inputs. One input is used to provide data to the block, which needs to be stored. The other input is a trigger input used to start and finish logging. If the input is not brought back to zero, the log file is still considered open and data will be corrupted. All data is stored to binary files on the SD-card.

ExamplePrintFcn The ExamplePrintFcn block provides an example of printing data in the PX4 Nuttx console terminal. The Nuttx console terminal is a Real-Time Operating System (RTOS) designed for devices with limited space. However, this function is not always working in conjunction with floating numbers. Furthermore, in order to work, the PX4 serial console shell should be activated via the `rc.txt` file. The Serial console shell can not be active on the main USB gate, since Simulink will otherwise not upload any code. Therefore, the serial console shell should communicate over a different UART connection which can be connected to a FTDI connector. However, this functionality only works when the quad-copter is connected to the pc using a USB cable. Only then, the command prompt of the quad-copter can be entered.

Read ADC Channels The Pixhawk contains three Analog Digital Converter (ADC) connections, which supports analog signals of 3.3V, 3.3V and 6.6V respectively. The Read ADC Channels block provides these signals into Simulink. Sonar sensors could be for instance connected to ADC gate, which provides the quad-copter from additional distance measurements.

uORB Write This block provides the ability to publish data to an existing uORB Topic with properly defined structure elements. In this software version, only a maximum of five structure elements are accessible.

uORB Read / Function-Call Trigger can be used in two different ways. Either this block reads data from a uORB topic continuously, provided the topic exist and is well structured and defined. Or the block is triggered if new data is published on the topic. In this case, the block responds in an asynchronous fashion. The block can only operate in one of those fashions.

Serial The serial block can be used to read serial data form any of the Universal Asynchronous Receiver/Transmitter (UART) ports or the Universal Serial Bus (USB) port available on the Pixhawk. Although this is a direct approach, it is advised to let the drivers handle communication to known devices, such as the GPS and the IMU, since drivers handle configuration of the devices better. The block can be configured as either a send block or a receiving block.

Read uORB Function Trigger Data This block will grab data affiliated with the uORB Topic which is driving the asynchronous subsystem this block is located. The uORB Topic name will automatically be determined based on call-backs.

Software installation on Linux is also provided. Although it is beneficial to run the Pixhawk Simulink toolbox on a Linux distribution since it supports uploading code better than in windows, it is not treated in this report, since it is out of scope. One last remark on the software, a better package is developed during this graduation project, which allows easier installation and it supports newer versions of Matlab. More information can be found at [57].

3.2.1 Configuration and timing tools

Configuring Matlab Simulink, in order to control the quad-copter can be done following Matlabs tutorial [56]. However, some topics deserve additional attention. Most of the settings are applied via the model configuration panel in Simulink. The most important aspects of the code generation can be found in the Subsection "Hardware Implementation" in the "Simulink configuration panel".

One of the most important aspects in real-time software design, is timing. The Matlab Simulink PSP contains a few tools to improve and or analyzing software loop times. Aside from the fixed step size configurable in the solver menu, additional settings can be configured to improve or analyze timing.

Base rate task priority During startup, various threats are spawned, such as the base-rate thread. The base-rate threat tries to let the model run at the configured sample rate. Depending on the priority of the base-rate threat, the model runs on-time or might have issues being finished after the deadline. Increasing the priority results in less processes interrupting the model. However, increasing the priority too high results in other process stop running on the Pixhawk, which might lead to a lack of sensor updates. It is a powerful tool to tune, in order to meet timing requirements, but tuning might lead to no performance at all.

Hard Real-Time constraints Hard real-time constraints can be set in the Hardware Implementation Tab. This setting makes sure deadlines are met in the software. Although this suggests timing improvement, it also means software shuts down after task over-runs occurs to many times. A task over-run means software takes longer to process then the deadline is scheduled. The software will be stopped regardless of the results or the progression and a new iteration is started. The Real-Time constraint helps find a proper sample rate, such that the control loop software is on-time, although it should never be used during flight.

External mode options External mode allows users to examine execution on-line, which facilitates debugging and interactive testing. In external mode, the signal would not only be displayed in Simulink. After a run is completed, the data is logged, such that off-line analysis can be applied. Although this seems to be an excellent form of tuning controllers, it can not be used during flight. External mode relies on a wired connection between the Pixhawk and the PC. Furthermore, the software needs to be started via the play button in Simulink. This means external mode can not rely on a wireless connection. Therefore, external mode is not suitable to use during flight. However, for tuning procedures, which do not involve flight, external mode is an ideal solution to tune parameters.

Rate Transition The rate transition block, provided by Simulink, can be used to lower the sampling rate of particular software parts. The rate transitions spawns new threads, which contains the software located after the rate transition block. Software parts running on different sample rates, and therefore run in different threats, are generally shown in different colors in Simulink after code generation. The most important beneficial property of this block is computational heavy software parts can be placed in slower threats, and therefore reduce limitations on important control loops, which needs to run on higher frequencies.

3.2.2 Boot process alternation

In order to load software designed in Simulink, the bootprocess of the Pixhawk needs to be altered. The bootprocess of the Pixhawk progresses through multiple files for spawning processes. The bootprocess of the Pixhawk starts with the file `rcS` located in `ROMFS/px4fmu_common/init.d` folder. First, the file is reset and all global general parameters are loaded. Examples are disarming the quad-copter, storing specific file locations, and setting up necessary hardware, such as mounting the SD-card.

After loading and configuring global parameters, the bootprocess searches for `rc.txt` file located on the SD-card in the folder `/etc`. If the file exists, the boot process programmed in the `rcS` file will be terminated and the boot process programmed in `rc.txt` continue.

If the `rc.txt` file does not exist, the boot process continues booting following the procedure programmed in `rcS`. The next step is starting basic processes, such as `uORB`, `dataman` and the RGB-LED for system indication and loading hardware specific parameters, such as number of battery cells. If the `config.txt` file is located on the SD-card in the `/etc` folder, parameters can be altered before the boot processes progresses. This allows disabling processes or changing hardware configurations regardless of configuration settings.

After loading the parameters and starting the processes, the file which the boot sequence looks for is `extras.txt`, which is located on the SD-card in the `/etc` folder. The `extras.txt` file allows of starting custom processes. A full overview of the boot process is presented in the block diagram of Figure 3.2. More information about the boot process can be found in [58].

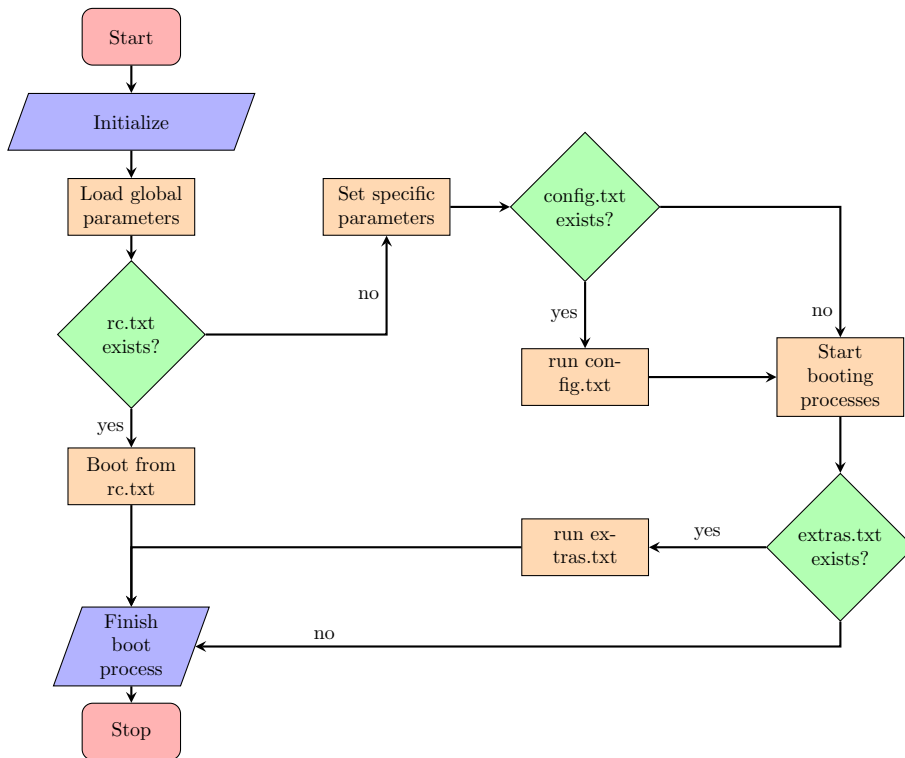


Figure 3.2: Boot process Pixhawk flight stack.

The boot process can be altered or manipulated in two different fashions. The first one is writing a complete custom boot procedure in the `rc.txt` file. The second procedure holds writing commands in the `config.txt` and `extras.txt` files, in order to maintain most of the boot process, but change it where necessary. This project utilizes the first method and a complete custom boot procedure is written. Running Simulink software on the Pixhawk only needs a few specific modules and therefore it is more effective to only load the few needed processes instead of shutting down all other processes. The boot procedure used in this project can be found in Appendix C, which is abbreviated from [56].

3.2.3 Quad copter software

The quad-copter control system design is based on standard control schematics, as provided in Figure 6.1. A state machine, controlling the decision making of the quad-copter, is not present in the control scheme, since those functionalities are controlled via the RC by the user. Furthermore, the control scheme consists of sensors, various filters, controllers, a motor mixer and outputs to the motors. Alongside the control loop is the data processing scheme. In this part of the software, all the important data is captured and, if desired, communicated to the ground station. Furthermore, the quad-copter logs all the data if enabled on the RC.

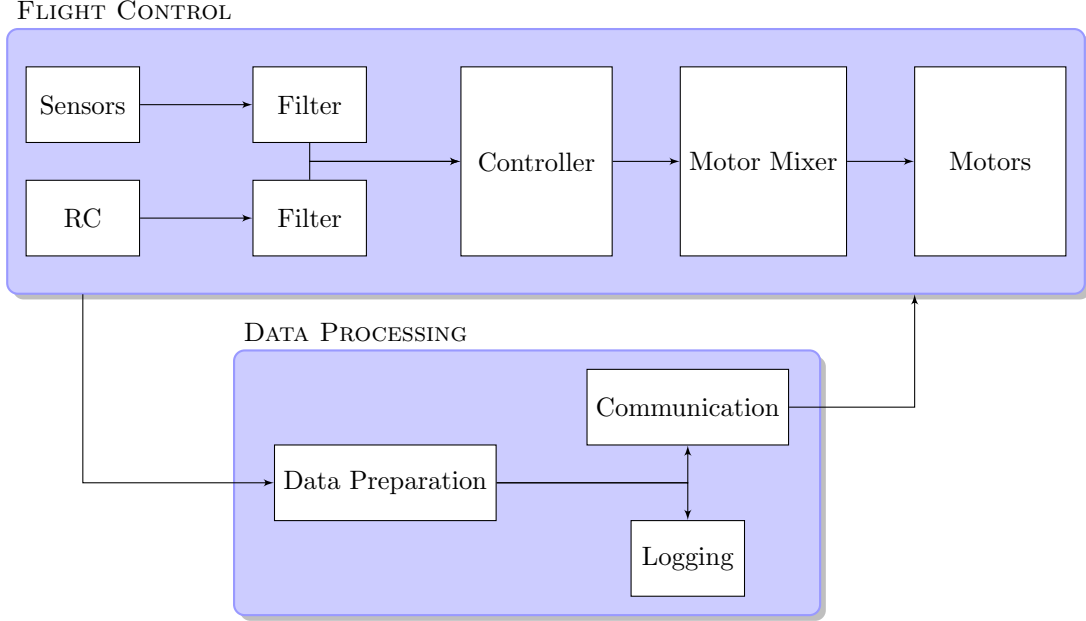


Figure 3.3: Simulink Software block diagram.

The sensor block consists of data measurements of the IMU, magnetometer, battery, GPS and if connected or provided, a sonar for distance measurements. The block RC only contains the RC input block which allows the user to control the quad-copter. Within the RC block, some conditions are set depending on the RC inputs, such that the quad-copter can fly in various different modes. The mode switching of the quad-copter is explained in Section 3.3.

The inputs, both the sensors and the RC block, are filtered. First of all, the RC joysticks are normalized between $\{-1, 1\}$ in case of all joysticks, except for the throttle, which is normalized between $\{0, 1\}$. After normalizing the received values, the signals are amplified to pre-configured values, which can be seen in Table 3.1. In case of the RC block, a low pass filter is used to remove high frequent noise from the joysticks. The filter is set with a cut-off frequency of 10Hz.

Joystick	Roll [rad]	Pitch [rad]	Yaw [rad]	Thrust [N]	X [m]	Y [m]	Z [m]
Min	$-\frac{1}{4}\pi$	$-\frac{1}{4}\pi$	$-\pi$	-0.2760	-1	-1	0
Max	$\frac{1}{4}\pi$	$\frac{1}{4}\pi$	π	-39.8720	1	1	1

Table 3.1: RC Joystick scaling values.

The filters on sensor side consists of removing any offset present in the measurement, and scaling the measurement with calibrated values. Furthermore, a Madgwick filter is used to obtain angles and a Kalman filter is used to filter and estimate the translational states. Both filters will be further explained in Chapter 4, which also explains the sensor calibration.

After filtering, the RC and estimated states are inputted in the control part. Depending on the mode

selected by the user, either the desired angles and thrust obtained from the RC, or desired angles and height, or desired location are set as reference. The control loop also contains a ramp, which after pre-arm mode slowly outputs the control outputs, such that the Electronic Speed Controller (ESC) do not disable. This is mostly important during altitude hold control, in which a large thrust is needed to keep the quad-copter in a fixed height.

The motor mixing distributes the body torques and forces to thrust forces which should be generated by the motors. The motor forces are transformed to PWM values. In case configuration mode is selected, the motor mixing block arranges the thrust provided by the RC to the selected motor. After motor mixing, the inputs are sent to the motors via the motors block. This block accepts PWM values as input to the specific configured motors and a boolean arming command.

Alongside the control loop is the data processing loop. Within this loop the data preparation block gathers most significant signals from the control loop. The gathered data is placed in either a bus, which contains all signals, or a parameter bus, which contains only system parameters. The data can be stored on the SD-card, via the logging block. This only happens, when logging is enabled via the RC. Furthermore, the Communication block sends pre-selected data over UART to the Raspberry Pi and to the ground station. Received information enters the software via this block as well.

3.2.4 Ground station software

The ground station software allows the user to visualize data from the quad-copter in the ground station. The communication flow is shown in Figure 3.4. Data is obtained on a computer via a virtual Communication port (COM port). The virtual COM port is more extensively explained in Section 3.4. Serial Send and Serial Receive block are used to accept data from the COM port port in Simulink.



Figure 3.4: Simulink Ground Station Software block diagram.

Messages to the quad-copter are sent on a sentence based message approach. Each sentence starts with **\$Drone**, such that each start of a sentence can be recognized between all other sentences. Furthermore, data is added to the sentence, based on user requirements. In general this means status updates over specific signals, such as states or battery status, should be used. The sentence is finished with an Astrix followed by a Cyclic Redundancy Check (CRC). The CRC checksum is based on a bitwise Exclusive Or (XOR) operation of the payload of the message.

The bitwise XOR operations starts with the first two bytes in the message, which are the first two bytes after **\$Drone**. Every following two bytes are compared against previous operation. If only one bit is set, the new checksum gets a set bit. If both or none bits are set, the checksum bit is set to zero. This process is repeated until the last two bytes of the payload are reached. Afterwards, the **\$Drone** is placed in front and the Astrix and CRC is placed at the end. An example of two bytes added to the CRC checksum in a XOR fashion is shown in Table 3.2.

Bit	Byte 1			Byte 2		
	1	2	15	16
Previous checksum	0	1	0	1
New Bytes	0	0	1	1
New checksum	0	1	1	0

Table 3.2: Bitwise Exclusive Or operation.

Received data is evaluated via the CRC checksum present at the end. If new data matches the CRC checksum at the end, the data will be further processed and in most cases only visualized. If the CRC checksum does not match, sended data will be discarded. Although the communication is mainly designed to send and receive status data and even though a CRC checksum is used, it is not good

practice to use the communication inside the control loop, due to lost and mis-interpreted messages. More information about the CRC checksum can be found in [59] and [60].

3.3 Configuring remote control

The Radio Control Unit (RC) is used as a state machine, in which the user can select various software routines. Initial configuration of the RC can be found in various documentation, such as [51], [61] and [50], which explains how to bind the RC and the receiver. This documentation also explains how to setup fail safes and receiver number in case of multiple receiver usages. This section only explains configuration after the initial setup.

Various buttons, switches, levers and joysticks are binded to the functionality in the software. The user in this case functions as a state machine and provides the quad-copter with references. An overview of all buttons, switches, levers and radio buttons can be found in Figure 2.13. In the mixer page of the FrSky Taranis X9D Plus RC the inputs can be bounded to specific radio channels. In total, the Simulink block supports up to 8 channels, therefore only 8 inputs can be bounded for functionality. An overview of the binds are shown in Table 3.3.

Mixer Page			
Channel	Weight	Source	Name
1	100	I AIL	Roll
2	100	I RUD	Yaw
3	100	I ELE	Pitch
4	100	I THR	Throttle
5	100	SA	Arm-Config
6	100	SC	F_Modes
7	100	SD	F_Modes
8	100	SB	Optional

Table 3.3: Remote Control mixer page.

Aileron (AIL) controls the roll movement and is bounded by J1. The Rudder (RUD) controls the yaw movement, which is bounded by J4. The Eleron (ELE) controls the pitch movement, which is bounded by J2 and Throttle (THR) controls the throttle which is bounded by J3. Both joysticks are variable in position (up, down, left, right) and moving a lever in a specific direction variates a value on the bounded channel in the Simulink software. Therefore, each lever contains two variable channels, which control two references. The switches SA, SB, SC and SD have three different positions and are used to activate/deactivate programmed functionality. The specific functions of the switches are shown in Table 3.4.

SA	Up	Dis-Arm		SC	SD	SB		
				None	None	None		
	Middle	Arm			SC	SD	SB	
				Up	None	Manual	None	
				Middle	Recording	Semi-Auto	None	
		Down	Recording	Auto	None			
	Down	Config	Auto-recording		SB		SD	
					None	Up	Middle	Down
				SC	Up	M1	M2	None
					Middle	M3	M4	None
Down					None	None	None	

Table 3.4: Remote Control switches functionality.

The switch SA is set as the leading switch, which enables, disables the quad-copter and the functionality of all other switches instantly. The modes of the quad-copter are

Dis-Arm Is a state in which the quad-copter is dis-armed. In this state, the quad-copter is safe to approach and move around by hand although it is advised to dis-arm the quad-copter with the

hardware switch and disconnect the battery. In this state, the motors will not spin, and all functionality of the other switches are disabled.

Arm In armed state, the quad-copter is activated and the motors are clear to spin. In this situation, it is not advised to approach the quad-copter or move it around by hand. Switch SC enables and disables recording data. Switch SC also backups configured parameters. Switch SD manages the following functionality:

Maunal Flight In manual flight, the throttle joysticks controls thrust directly and the roll, pitch and yaw joysticks generate references for the angles of the quad-copter. The activated control scheme is the angle controller.

Semi-auto flight In semi-auto flight, the angles are controlled in the same manner as during manual control. However, the thrust is not controlled by the thrust stick anymore. Instead, the throttle stick is used to generate a desired altitude. The activated control scheme is the altitude-hold controller.

Auto flight In auto flight all joysticks are generating a reference on position level, except for the yaw joystick, which still generates a reference for the yaw angle. The throttle stick relates to the altitude, z-position, the pitch joystick relates to the x-position, which is forward and backward movement. The roll joystick is connected to the y-position, which is a sideways movement. The activated control scheme is the position-hold controller.

Config In config mode, the quad-copter is armed and starts recording automatically. Depending on the state of switches SC and SD, motors 1, 2, 3 or 4 are enabled or disabled. This allows to help find the rotation direction of the motors during installation. Or test motor health separately.

Switch SB is not used in this configuration. Since the Simulink software allows for 8 channels, this switch is open to additional functionality.

3.4 Wireless communication

The wireless communication is performed by the Raspberry Pi 3 model B V1.2. This Raspberry Pi is equipped with a WiFi network card, which can be used as an access point. This handles most of the difficult communication problems, such as packet loss and connectivity.

The data from the Pixhawk is transferred to the Raspberry Pi via a UART connection, which is described in Section 2.5. The data obtained from the UART connection is sent over WiFi to the ground station computer using Serial to Network software (ser2net). ser2net allows serial data to be communicated over a network. On the ground station computer COMbyTCP connects the network port to a virtual COM port, which is provided by COM0COM software. On one side of the virtual COM port a Transmission Control Protocol/Internet Protocol (TCP-IP) connection is established. The other side of the virtual COM port is provided to Simulink ground station software as COM port. The overall connection acts as a COM port, such as any other USB-device, bridged over a network. An overview of data flows through software packages can be seen in Figure 3.5.

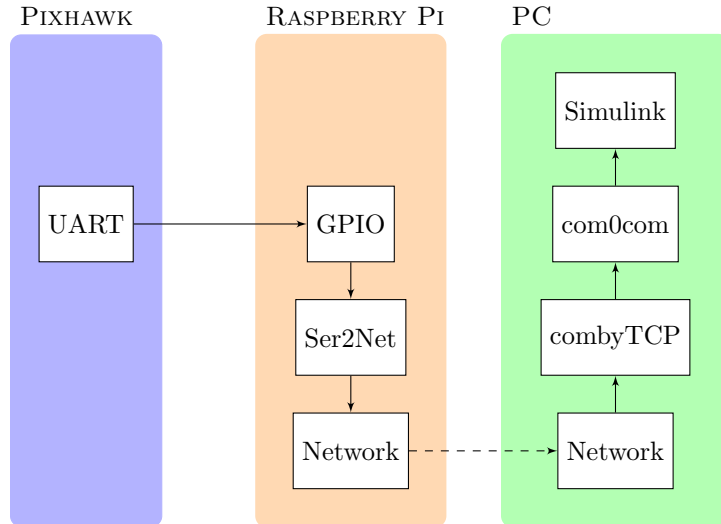


Figure 3.5: Communication block diagram from Pixhawk to pc.

Setting up the Raspberry Pi to transmit data obtained from the general purpose input output (GPIO) pins starts with loading basic Raspbian software. Since multiple tutorials on how to install Raspbian on a Raspberry Pi are available on-line, this part is not covered here. After installation, the SD-card should be resized, such that more space can be used by the software. After that, enabling Secure Shell (SSH) is needed in order to connect to the raspberry Pi via a terminal. Furthermore, enabling serial communication allows data flow over the GPIO pins. Lastly, the IP-addresses are fixed, such that a connection to the Raspberry Pi over a network can always be established. This process, and the commands to do so are provided in Appendix E.1.

After the installation and configuration of Raspbian on the Raspberry Pi, the ser2net software package can be installed. Furthermore, the Raspberry Pi can be transformed into an access point. Configuration and installation of the ser2net software is done via a script provided in Appendix E.2. In Appendix E.3 a script is provided for setting the Raspberry Pi up as an access point. Since the access point software might drop the network connection at all, it is best to first install the ser2net software.

On computer side, the software packages are available on the internet and easy to install. Starting with com0com, which can be found at [62], after unzipping, installing and starting the software, a screen shows which COM port can be setup as virtual COM port. It might take a while before the COM port are available, since the computer installs drivers to let the COM port function. An overview of the software and the configuration can be found in Appendix F. CombyTCP can be downloaded from [63]. The software can be started without installation. On the network side, the IP address of the Raspberry Pi and the port number can be setup. In the COM port properties side, the virtual COM port from the com0com software can be setup. The communication can be further configured according to the settings setup in the ser2net software, such as Baud rate, Parity, number of data bits, number of stop bits and handshake. An overview of the software and sample configuration can be found in Appendix G.

3.5 Indoor Positioning System driver

The Marvelmind mobile beacon (hedgehog) can be configured to communicate locations directly to robots. In order to let the Marvelmind hedgehog communicate with the robot, either a driver for the Marvelmind protocol is needed, or a driver which accepts the generalized GPS protocol, called NMEA 0183. The protocol is maintained by the National Marine Electronics Association (NMEA). The protocol is designed to let various different devices communicate on a sentenced based communication, which only allows American Standard Code for Information Interchange (ASCII) characters [64].

The communication protocol is designed for various different devices of which GPS is one of. The sentences sent over the communication lines follow a specific sentence format. The start is always marked

with an \$-sign followed by two characters, which are called "Talker Identifiers". The talker identifiers differentiate between various devices. All GPS messages are marked with GP, which let the receiver know a GPS messages is send. The talker identifiers are followed by message identifiers, which are three characters, a few examples are treated in [65] and [66]. Furthermore, each message has a fixed format and values are delimited by commas. The end of a sentence is marked with an Astrix followed by the CRC checksum.

Explaining various different messages is out of scope and is not explained in this report, only the messages which Marvelmind hedgehog is able to send are explained. The full architecture can be found in [64]. The Marvelmind hedgehog is able to broadcast four different NMEA 0183 messages. The GPS NMEA 0183 messages broadcast by the hedgehog are explained in the following subsections.

The designed drivers handles the NMEA 0183 messages broadcasted by the Marvelmind. In Appendix H the installation of the IPS driver is explained. In Appendix I is the configuration provided for the Marvelmind during this project.

3.5.1 GPGGA message

The GGA message is the so called Global Positioning System Fix Data. The message contains most information about the location, quality of the position measurement and timestamps. An example message format is shown in Listing 3.1 and Table 3.5. For controlling the quad-copter on position level, this message contains the most important information. During configuration of the hedgehog, the GGA message needs to be enabled.

1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2																
3	\$GPGGA,	hhmmss.ss,	lllll.ll,	a,	yyyyy.yy,	a,	x,	xx,	x.x,	x.x,	M,	x.x,	M,	x.x,	xxxx*hh	

Listing 3.1: GPGGA example message format.

1	Talker and message identifiers
2	Time (UTC)
3	Latitude
4	N or S (North or South)
5	Longitude
6	E or W (East or West)
7	GPS Quality Indicator
	0 - fix not available
	1 - GPS fix
	2 - Differential GPS fix
8	Number of satellites in view, 00 - 12
9	Horizontal Dilution of precision
10	Antenna Altitude above/below mean-sea-level (geoid)
11	Units of antenna altitude, meters
12	Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
13	Units of geoidal separation, meters
14	Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, null field when DGPS is not used
15	Differential reference station ID, 0000-1023
16	Checksum

Table 3.5: GPGGA message format.

3.5.2 GPRMC message

The RMC message is the so called Recommended Minimum Navigation Information. The message contains most information about the 2D location, velocity over ground, date stamp and information about magnetic variation. An example message format is shown in Listing 3.2 and Table 3.6. Although it is available and contains velocity information, for controlling the quad-copter on position level this message is not necessarily needed since it only contains a 2D position measurement.

1	1	2	3	4	5	6	7	8	9	10	11	12	13
2													
3	\$GPRMC, hhmmss.ss,A, llll.ll, a, yyyyy.yy, a, x.x, x.x, xxxx, x.x, a*hh												

Listing 3.2: GPRMC example message format.

- 1 Talker and message identifiers
- 2 Time (UTC)
- 3 Status, A = data valid, V = data invalid
- 4 Latitude
- 5 N or S (North or South)
- 6 Longitude
- 7 E or W (East or West)
- 8 Speed over ground, knots
- 9 Track made good, degrees true
- 10 Date, ddmmyy
- 11 Magnetic Variation, degrees
- 12 E or W (East or West)
- 13 Checksum

Table 3.6: GPRMC message format.

3.5.3 GPVTG message

The VTG message is the so called Track Made Good and Ground Speed. The message contains information about velocity in knots and kilometers per hour. Since the velocity is expressed in a single value, the direction is included, which can be expressed in degrees and magnetic direction. An example message format is shown in Listing 3.3 and Table 3.7. Although it is available and contains velocity information, for controlling the quad-copter on position level this message is not necessarily needed.

1	1	2	3	4	5	6	7	8	9	10
2										
3	\$GPVTG, x.x, T, x.x, M, x.x, N, x.x, K*hh									

Listing 3.3: GPVTG example message format.

- 1 Talker and message identifiers
- 2 Track Degrees
- 3 T = True
- 4 Track Degrees
- 5 M = Magnetic
- 6 Speed Knots
- 7 N = Knots
- 8 Speed Kilometers Per Hour
- 9 K = Kilometres Per Hour
- 10 Checksum

Table 3.7: GPVTG message Format.

3.5.4 GPZDA message

The ZDA message is the so called Time and Date message. The message contains information about local time zone and date stamp. Since the Marvelmind does not cross timezones during usage, it is not needed to broadcast this message. An example message format is shown in Listing 3.4 and Table 3.8.

1	1	2	3	4	5	6	7	8
2								
3	\$GPZDA, hhmmss.ss, xx, xx, xxxx, xx, xx*hh							

Listing 3.4: GPZDA example message format.

- 1 Talker and message identifiers
- 2 Local zone minutes description, same sign as local hours
- 3 Local zone description, 00 to +/- 13 hours
- 4 Year
- 5 Month, 01 to 12
- 6 Day, 01 to 31
- 7 Time (UTC)
- 8 Checksum

Table 3.8: GPZDA message format.

3.6 Matlab Graphical User Interface

In order to configure, evaluate and visualize the logged data, a Matlab Graphical User Interface (GUI) is designed. The bin-files are loaded using Matlab code presented in Appendix D, which is obtained from [56]. Depending on the pressed buttons, the GUI loads sub windows, in which either data is visualized, or software runs such that specific calibration sequences are performed. The Matlab GUI is shown in Figure 3.6.

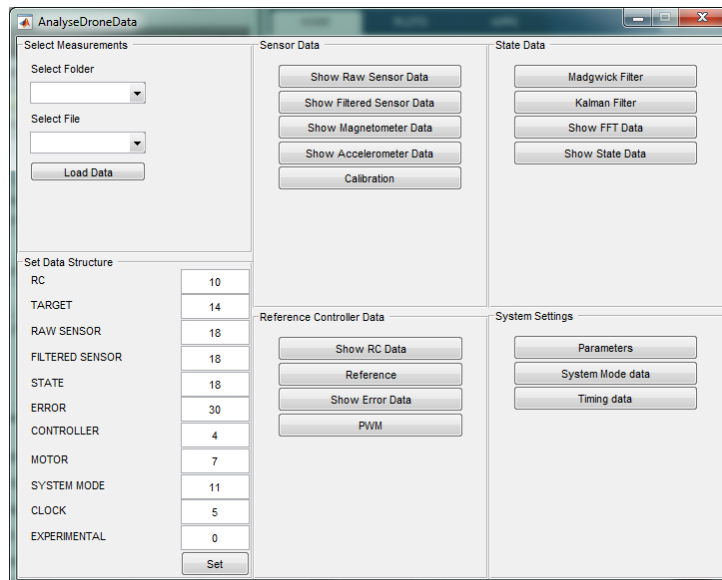


Figure 3.6: Matlab GUI for configuration, evaluation and visualization of the flight data.

The GUI can be divided in six different parts loading data is the first part and setting the data structure is the second part. Both parts are shown on the left hand side of Figure 3.6. The measurement can be placed in folders inside the GUI directory. The GUI looks for three files called `RawSensorData_*.bin`, `SettingsP1_*.bin` and `SettingsP2_*.bin`, where `*` is a wild-card for any kind of name or character set.

If all three files are present, the data can be loaded into the GUI. The structure definition part divides various sizes of vectors from the loaded bin-files. If accidentally the data size changes, it can be adopted here.

The middle top part of the GUI holds the sensor subsection. This section provides options to visualize various sensor signals, starting by raw sensor data, filtered data and 3D data, for instance in case of the magnetometer. Furthermore, a calibration option is present as well, which allows fast calibration based on the loaded data.

The right top part contains options to configure the Madgwick filter and Kalman filter, which are explained in further detail in Subsections 4.2.2 and 4.2.3. Furthermore, state estimation data can be shown as well in plain 2D data or as 3D data in case of positions and angles. In addition, the state data section has the ability to show any kind of signal in Fast-Fourier-Transform (FFT) plot.

The middle bottom section of the GUI contains options regarding references, controllers and outputs. The first options shows all obtain RC inputs, varying from raw roll, pitch, yaw command to Received Signal Strength Indicator (RSSI) and failsafe. The reference option visualizes the references inputted to the control structure. In case of a cascaded control structure, the reference might be obtained from another state. The error data shows the error data inputted to the control structure. This option mainly focuses on what each part of the controller contributes to the control action, such as Proportional Integral Derivative (PID) errors and feedforwards. The last option shows the requested torques and forces generated by the controllers. Furthermore, the resulted PWM values can be visualized. The status of the third order solver provided in Appendix L can be seen as will to debug thrust to PWM transformations.

The last section of the GUI provides system status information. System status information can be found on the right bottom side of the GUI and provides information options for parameters and internal clocks to analyze timing during operation. Lastly, the section provides an option called system mode which shows the flight modes during flight, such as angle control, altitude-hold control or position control.

Chapter 4

Mathematical model, state estimation and control

Previous chapters explained the hardware and software side of this master thesis project. This Chapter provides a mathematical model of the quad-copter and addresses its control. In Section 4.1, the non-linear mathematical model is discussed. The non-linear model will be linearized around near hovering flight modes. Furthermore, Section 4.1 describes the propulsion model and sensor calibration. In Section 4.2, state estimators are discussed. The state estimation algorithms consists of a Madgwick filter for the angular states and a Kalman filter for the translational states. Lastly, in Section 4.3 a controller will be synthesized based on the linearized model near hovering flight modes. The linearized model is discretized. A feedback controller is designed based on pole placement.

4.1 Mathematical model

In various works in the literature, a quad-copter model is already provided [27], [26], [6] and [9]. Therefore, a full first principles (e.g. Lagrange or Newton Euler) derivation of the mathematical quad-copter model will not be explained here. Only key elements, such as frame rotations, are discussed. Figure 4.1 shows a mathematical representation of a quad-copter in which a body fixed frame and earth fixed frame is chosen. Depending on the frame selection both the mathematical model and its dynamics, as the sensor fusion filters and their state estimators vary in structure.

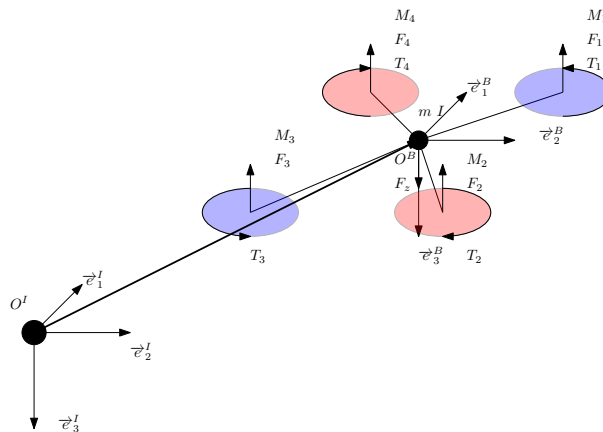


Figure 4.1: Schematic representation of the quad-copter.

4.1.1 Reference frame

The North-East-Down reference frame (NED frame) is the most commonly used rotation frame in aviation. According to this convention, the positive x-axis points north, the positive y-axis points east and the positive z-axis points down. Any rotation around this frame can be described by three independent Euler angles rotating around x , y and z . The rotation around x is described by $R_x(\phi)$, see Equation (4.1), where $\phi \in \mathbb{R}$ is the rotation in radians. The rotation around x is also known as the roll rotation. The rotation around y is given in Equation (4.2), where $\theta \in \mathbb{R}$ is the rotation in radians, also known as the pitch rotation, and a rotation around z is given as Equation (4.3), where $\psi \in \mathbb{R}$ is the rotation in

radians, also known as the yaw rotation.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (4.1)$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (4.2)$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

The full rotation from body fixed frame \vec{e}^B to earth fixed frame \vec{e}^I , as depicted in Figure 4.1, can therefore be expressed as three independent rotations ϕ , θ and ψ around the three independent axes x , y and z . Equation (4.4) describes the rotation in matrix form. The rotation frame is an element of the special orthogonal group $SO(3) = \{R(\lambda) \in \mathbb{R}^{3 \times 3} : R^T(\lambda)R(\lambda) = R(\lambda)R^T(\lambda) = I_3, \det R(\lambda) = 1\}$.

$$\begin{aligned} R(\lambda) &= R_z(\psi)R_y(\theta)R_x(\phi) \\ &= \begin{bmatrix} c\psi c\theta & c\psi s\phi s\theta - c\phi s\psi & s\phi s\psi + c\phi c\psi s\theta \\ c\theta s\psi & c\phi c\psi + s\phi s\psi s\theta & c\phi s\psi s\theta - c\psi s\phi \\ -s\theta & c\theta s\phi & c\phi c\theta \end{bmatrix} \end{aligned} \quad (4.4)$$

where s and c represent sin and cos respectively and ϕ , θ and ψ are the angles around the x , y and z -axis respectively.

4.1.2 Non-linear quad-copter model

The quad-copter model is represented as a non-linear rigid body expressed with Newton-Euler equations. The equations of the quad-copter dynamics are as follows:

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= g\vec{e}_3^I - \frac{1}{m}R(\lambda)T\vec{e}_3^B \\ \dot{\lambda} &= Q(\lambda)\omega \\ \dot{\omega} &= J^{-1}S_{skew}(\omega)J\omega + J^{-1}\tau \end{aligned} \quad (4.5)$$

where $p = [x \ y \ z]^T \in \mathbb{R}^3$ is the position of the center of mass of the quad-copter with respect to the origin of the earth fixed frame O^I , $v \in \mathbb{R}^{3 \times 1}$ is the velocity of the quad-copter, $T \in \mathbb{R}$ is the thrust generated by the quad-copters propellers F_1 , F_2 , F_3 and F_4 in body fixed frame and $g \in \mathbb{R}$ and $m \in \mathbb{R}$ are the gravitational acceleration acting on the quad-copter and the mass of the quad-copter respectively.

The angular velocities are expressed as $\omega = [\omega_x \ \omega_y \ \omega_z]^T \in \mathbb{R}^3$, $R(\lambda) \in SO(3)$ is the rotation matrix as described in Subsection 4.1.1. Furthermore, $\lambda = [\phi \ \theta \ \psi]^T \in \mathbb{R}^{3 \times 1}$ are the rotations angles around the x , y and z -axis from body fixed frame \vec{e}^B to earth fixed frame \vec{e}^I . The torques applied to the quad-copter by the propellers are expressed as $\tau = [\tau_x \ \tau_y \ \tau_z]^T \in \mathbb{R}^3$. $J \in \mathbb{R}^{3 \times 3}$ denotes the inertia matrix. The matrix $S_{skew}(\omega)$ is a skew symmetric matrix given by:

$$S_{skew}(\omega) = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix} \quad (4.6)$$

Lastly, the $Q(\lambda)$ matrix maps angular velocities to angular rate of change, which can be written as follows:

$$Q = \begin{bmatrix} 1 & \sin(\phi) \frac{\sin(\theta)}{\cos(\theta)} & -\cos(\phi) \frac{\sin(\theta)}{\cos(\theta)} \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) \frac{1}{\cos(\theta)} & \cos(\phi) \frac{1}{\cos(\theta)} \end{bmatrix} \quad (4.7)$$

4.1.3 Linearized quad-copter model

For the sake of simplicity, the non-linear quad-copter model proposed in (4.5) is linearized around an equilibrium point referred to as hovering mode. This equilibrium point is characterized by $p = [x \ y \ z]^T \in \mathbb{R}^3$, $v = 0$, $R(\lambda) = I$, which means all velocities are 0, all angles are kept at 0, and all angular velocities are set to 0, $\omega = 0$, assuming the inertia matrix is of a diagonal form $J = \text{diag}(J_{xx}, J_{yy}, J_{zz})$. The linearized decoupled models become:

$$\begin{cases} \dot{\phi} = \omega_x \\ \dot{\omega}_x = \frac{1}{J_{xx}}\tau_x \end{cases} \quad \begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{T}{m}\theta \end{cases} \\ \\ \begin{cases} \dot{\theta} = \omega_y \\ \dot{\omega}_y = \frac{1}{J_{yy}}\tau_y \end{cases} \quad \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \frac{T}{m}\phi \end{cases} \\ \\ \begin{cases} \dot{\psi} = \omega_z \\ \dot{\omega}_z = \frac{1}{J_{zz}}\tau_z \end{cases} \quad \begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = \frac{1}{m}T \end{cases} \end{cases} \quad (4.8)$$

It can be seen, that all the system consists of multiple double integrators, where the input for the x and y dynamics depends on the angles θ and ϕ and the thrust T generated for holding altitude in z direction. This means the quad-copter model can also be written as four linear time-invariant (LTI) decoupled systems as described by:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{T^*}{m}\theta \\ \dot{\theta} = \omega_y \\ \dot{\omega}_y = \frac{1}{J_{yy}}\tau_y \end{cases} \quad \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \frac{T^*}{m}\phi \\ \dot{\phi} = \omega_x \\ \dot{\omega}_x = \frac{1}{J_{xx}}\tau_x \end{cases} \\ \\ \begin{cases} \dot{\psi} = \omega_z \\ \dot{\omega}_z = \frac{1}{J_{zz}}\tau_z \end{cases} \quad \begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = \frac{1}{m}T \end{cases} \end{cases} \quad (4.9)$$

where T^* is the thrust. In near hovering modes the thrust is equal to the gravitational acceleration and the mass $T^* = -gm$ of the quad-copter, to keep it in z -position.

4.1.4 Motor mixing matrix

The motor mixing matrix distributes the requested control forces to the necessary motors. The motor mixing matrix depends on the positioning of the motors in relation to the center of mass. The transformation from control inputs $[f \ \tau_x \ \tau_y \ \tau_z]^T$ to motor inputs $[f_{m1} \ f_{m2} \ f_{m3} \ f_{m4}]^T$ utilizing the motor mixing matrix M_m is described by:

$$\begin{bmatrix} f_{m1} \\ f_{m2} \\ f_{m3} \\ f_{m4} \end{bmatrix} = M_m^{-1} \begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \quad (4.10)$$

where $M_m \in \mathbb{R}^{4 \times 4}$ is the motor mixing matrix, $f \in \mathbb{R}$ is the control thrust, $\tau_x \in \mathbb{R}$ is the roll control moment, $\tau_y \in \mathbb{R}$ is the pitch control moment, $\tau_z \in \mathbb{R}$ is the yaw control moment and $f_{mi} \in \mathbb{R}$ is the motor force for motor i . The motor mixing matrix is obtained by redesigning the quad-copter in Solidworks, see Appendix M. Within Solidworks, distances can be measured from each motor to the center of mass, the results are shown in Table 4.1.

	Distance to center of mass		
	X	Y	Z
	[mm]	[mm]	[mm]
Motor 1	165.87	212.64	53.33
Motor 2	109.79	175.86	53.33
Motor 3	109.79	175.86	53.33
Motor 4	165.87	212.64	53.33

Table 4.1: Motor distances to center of mass.

The motor mixing matrix M_m structure is depending on the configuration of the drone. Provided the configuration as is depicted in Figure 4.1, the motor mixing matrix can be described as:

$$M_m = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \\ -R_1 & -R_2 & R_3 & R_4 \\ P_1 & -P_2 & -P_3 & P_4 \\ Y_1 & -Y_2 & Y_3 & -Y_4 \end{bmatrix} \quad (4.11)$$

where T_i is the thrust constant for motor i , R_i is the roll constant for motor i , P_i is the pitch constant for motor i and Y_i is the yaw constant for motor i . Provided the measured distances in Table 4.1, the motor mixing constants can be obtained. The thrust constant is a an equal contribution of all motors. Therefore T_i is set at 1 for all motors. The roll constant R_i , which transforms motor force f_i to roll torque τ_x is obtained by taking the Y distance of the motor to the center of mass in meters. The pitch constant P_i , which transforms motor force f_i to pitch torque τ_y is obtained by taking the X distance of the motor to the center of mass in meters. The yaw constant Y_i , which transforms motor force f_i to yaw torque τ_z is obtained by taking the X - Y distance of the motor to the center of mass in meters. This is can be described by:

$$Y_i = \sqrt{X_{m_i}^2 + Y_{m_i}^2} \quad (4.12)$$

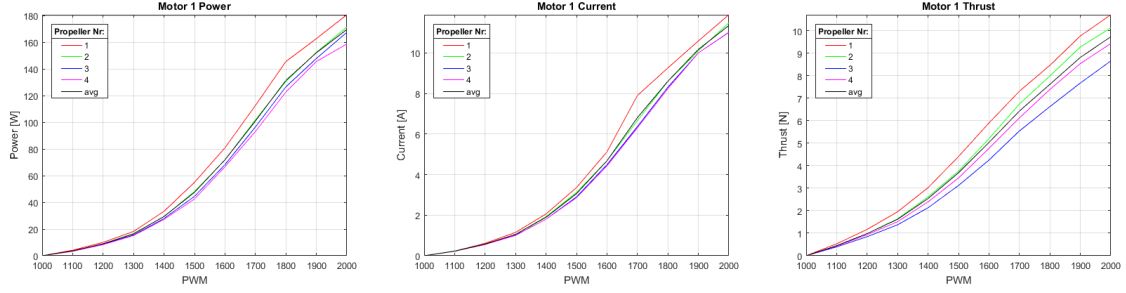
The results of constructing the motor mixing matrix is provided in Table 4.2. Due to inconsistencies in the Solidworks model compared to the real test-bed, the motor mixing matrix is slightly adjusted for improved control.

	Motor mixing constants			
	T	R	P	Y
Motor 1	1	0.212	0.165	0.2686
Motor 2	1	0.175	0.109	0.2062
Motor 3	1	0.175	0.109	0.2062
Motor 4	1	0.212	0.165	0.2686

Table 4.2: Motor distances to center of mass.

4.1.5 Propulsion model

The thrust generated by the propellers can be controlled by sending Pulse Width Modulation (PWM) to the Electronic Speed Controller (ESC). In order to request the desired thrust, a mapping between the thrust and the PWM is experimentally obtained. The ESC accepts PWM values between 1000 and 2000 pulses, which corresponds to a duty-cycle between 50% and 100%. The experiments conducted, increment the PWM with steps of 100 pulses, which results in 11 different thrust measurements. The thrust is measured in terms of grams thrust, which is converted to newtons by multiplying by $\frac{9.81}{1000}$. In the process, the power and current are measured as well. The experiments were conducted with four different propellers per motor. The power, current and thrust measurements are shown in Figure 4.2 for motor 1 for each propeller. In black, is the average of the measurements shown, which is later used to make a least squares fit. Measurement data of motor 2, 3 and 4 are presented in Appendix J.



(a) Power measurement motor 1. (b) Current measurement motor 1. (c) Thrust measurement motor 1.

Figure 4.2: Power, current and thrust measurements for motor one with four different propellers.

In order to analytically obtain a PWM value for a requested thrust a data fit is carried out. In fact, second order and third order polynomials are used and compared to obtain a relation between PWM and thrust. The advantage of utilizing a second order polynomial is the relative easy inverse mapping from desired thrust to PWM. However, a third order polynomial results in a better fit. Therefore, both order polynomials will be compared.

Both polynomials can be expressed as Equation (4.13) where n defines the order of the polynomial. Furthermore, given m measurements, $X \in \mathbb{R}^{m \times n}$ where each column contains PWM measurements $x_i = [\text{pwm}_i^n \ \dots \ \text{pwm}_i^0]$. $\tilde{T} \in \mathbb{R}^{m \times 1}$ is the estimated thrust and $T \in \mathbb{R}^{m \times 1}$ is the measured thrust. The vector $B \in \mathbb{R}^{n \times 1}$ contains the polynomial parameters of the fit. The unknown vector $B = [b_n \ \dots \ b_0]^T$ can be obtained from Equation (4.14), where the parameters are depending on the PWM measurements X and the actual measured thrust T .

$$\tilde{T} = XB \quad (4.13)$$

$$B = (X^T X)^{-1} X^T T \quad (4.14)$$

$$T = [\text{pwm}^n \ \dots \ \text{pwm}^0] \begin{bmatrix} b_n \\ \vdots \\ b_0 \end{bmatrix} \quad (4.15)$$

In order to quantize the error between the least squares fit thrust and the actual measured thrust, an error function is created based on the absolute value of the difference between the measured and estimated thrust, see Equation (4.16), where T is the measured thrust, \tilde{T} is the estimated thrust and E is the absolute value of the difference between the measured and the estimated thrust.

$$E_i = \sqrt{(\tilde{T}_i - T_i)^2} \quad (4.16)$$

$$\bar{E} = \frac{1}{n} \sum_{i=1}^n E_i \quad (4.17)$$

In Figure 4.3 in blue, a second order polynomial and its error function is plotted and in red, a third order polynomial and its error function is plotted over the average measurements of the 4 propellers for motor 1. The black line is the averaged thrust measurement for motor 1. As can be seen in the lower plot, the error for the second order polynomial fit is on average 0.2206 N and for the third order polynomial fit is on average 0.0814 N.

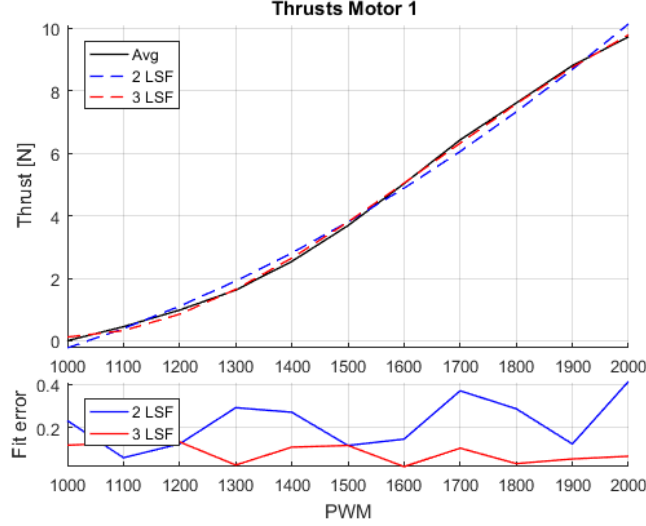


Figure 4.3: Least squares fit of second and third order polynomials, including the error.

Although the second order polynomial approximates the PWM to thrust relation closely, a third order polynomial shows a better fit compared to the second order polynomial. In Table 4.3 are the fitting results shown of the second order polynomial fit. In Table 4.4 are the fitting results shown of the third order polynomial fit. The quality of all fits is shown in column \bar{E} and a third order polynomial fit shows on average 2.5 times better fitting results compared to a second order fit. Therefore, a third order fit will be used on the quad-copter to approximate the thrust, despite the complexity involved in finding an inverse mapping.

Second order polynomial fit constants				
Motor	b_0	b_1	b_2	\bar{E}
M_1	-1.3605	-0.0035	$4.6206e^{-6}$	0.2206
M_2	-2.3054	-0.0023	$4.3465e^{-6}$	0.2334
M_3	-1.1023	-0.0039	$4.7844e^{-6}$	0.2186
M_4	-1.4868	-0.0037	$4.9622e^{-6}$	0.2429

Table 4.3: Results of fitting a second order polynomial fit to all thrust measurements of all four different motors. \bar{E} is the average absolute error of the least squares fit.

Third order polynomial fit constants					
Motor	b_0	b_1	b_2	b_3	\bar{E}
M_1	28.6596	-0.0670	$0.4809e^{-4}$	$-0.9659e^{-8}$	0.0814
M_2	30.4836	-0.0716	$0.5182e^{-4}$	$-1.0550e^{-8}$	0.0790
M_3	28.6231	-0.0668	$0.4782e^{-4}$	$-0.9564e^{-8}$	0.0794
M_4	31.7552	-0.0740	$0.5309e^{-4}$	$-1.0696e^{-8}$	0.0859

Table 4.4: Results of fitting a third order polynomial fit to all thrust measurements of all four different motors. \bar{E} is the average absolute error of the least squares fit.

During flight, a desired thrust, obtained from the controllers is mapped back to the corresponding PWM values. The polynomial to solve is presented in Equation (4.18), where $T_{desired}$ is the desired thrust for the specific motor and pwm is the PWM related to the desired thrust. The full algorithm to convert desired thrust to PWM, can be found in Appendix L.

$$T_{desired} = b_3 \text{pwm}^3 + b_2 \text{pwm}^2 + b_1 \text{pwm} + b_0 \quad (4.18)$$

4.1.6 Sensor Calibration

An important aspect, before sensors can be used is sensor calibration. The quad-copter contains an accelerometer, magnetometer and a gyroscope. Simple calibration methods exist, such as a four parameter calibration and a six parameter calibration [67],[68]. However, this project utilizes fitting an ellipsoid to measurement data. The calibration can be thought of as raw measurement data representing an ellipsoid, offset from the origin. Ideally, the measurements should represent a perfect orb with its center at the origin. By fitting an ellipsoid on the measured ellipsoid, scaling parameters and offset parameters can be obtained, resulting in a transformation from the measured ellipsoid to the desired orb. An example is depicted in Figure 4.4. In Subsection 4.1.6.1 the results of fitting an ellipsoid on the accelerometer and magnetometer data is discussed.

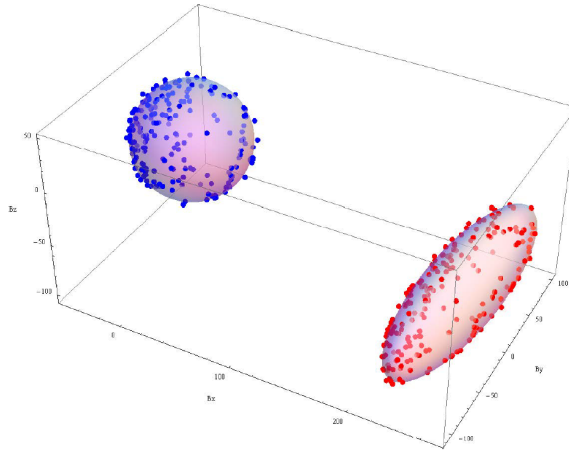


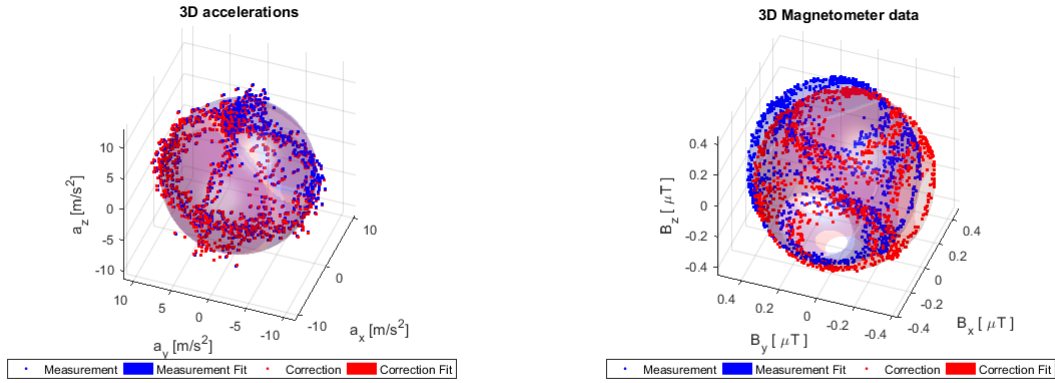
Figure 4.4: Sensor calibration, from ellipsoid to orb. Provided by [68].

Furthermore, the magnetometer needs additional calibration such that disturbances generated by the motors can be rejected. This calibration uses a least squares fit to correct for disturbances, which are related to the current. The calibration procedure is discussed in Subsection 4.1.6.2.

Fitting an ellipsoid is only useful for sensors which measure a reference vector in a specific direction without too much dynamical interference. The reference vector measured by the accelerometer is the gravitational acceleration vector. The reference vector measured by the magnetometer is the earth's magnetic field vector. However, the gyroscope measures angular velocity which only contains system dynamics and no reference vector. Therefore, calibrating the gyroscope is performed by integrating the measurement data and fitting the start and final measured angles to the number of rotations. Calibrating the gyroscope is discussed in Subsection 4.1.6.3.

4.1.6.1 Fitting an ellipsoid

Fitting an ellipsoid is based on [1]. Within this section fitting an ellipsoid is only highlighted. The code used to fit an ellipsoid to measurement data, for instance the accelerometer and magnetometer data, is provided in Appendix K. The Matlab code is obtained from [1]. Measurement data is obtained by rotating the accelerometer or magnetometer around all axis multiple times.



(a) Accelerometer calibration.

(b) Magnetometer calibration.

Figure 4.5: Fitting ellipsoids on measurement data to obtain offset, scaling and rotation parameters.

In Figure 4.5, calibration results of the magnetometer and accelerometer are shown. The results of the Matlab function are a vector "center", a vector "radii" and a matrix "evecs". The offset V is obtained as the "center" vector result obtained from the Matlab function. Measured data is corrected by removing the offset and scaling the measurements as provided by Equation (4.19). In Equation (4.19), $Y \in \mathbb{R}^{3 \times 1}$ contains uncalibrated measurement data from the accelerometer or the magnetometer, $W \in \mathbb{R}^{3 \times 3}$ contains the scaling parameters, $V \in \mathbb{R}^{3 \times 1}$ contains the offset and $\tilde{Y} \in \mathbb{R}^{3 \times 1}$ is the calibrated sensor measurement.

$$\begin{aligned}
 \tilde{Y} &= W(Y - V) \\
 W &= (\text{radii}) (\text{evecs}) (\text{radii})^{-1} (\text{evecs})^T \\
 V &= \text{center}
 \end{aligned} \tag{4.19}$$

The parameters a_i obtained through the ellipsoid fit method are provided in Table 4.5.

	Accelerometer			Magnetometer		
	a_x $\frac{m}{s^2}$	a_y $\frac{m}{s^2}$	a_z $\frac{m}{s^2}$	M_x μT	M_y μT	M_z μT
Scaling W_i	1.0152	-0.0044	-0.0051	1.0278	-0.0553	-0.0198
	-0.0044	1.0023	0.0161	-0.0553	0.9976	0.0167
	-0.0051	0.0161	0.9836	-0.0198	0.0167	0.9828
Offset V_i	0.1221	0.0028	-0.0298	0.0436	0.0715	0.0063

Table 4.5: Calibration results for magnetometer and accelerometer.

4.1.6.2 Current corrections

By fitting an ellipsoid to the measurement data, the accelerometer data can be calibrated. However, the magnetometer is only partially calibrated. Since the magnetometers principles works on earth magnetic field, it also picks up magnetic disturbances around it. One of the major disturbances are the quad-copter motors. By utilizing current measurements, which is related to the generated torques by the motors, the magnetic field disturbance generated by the motors can be compensated.

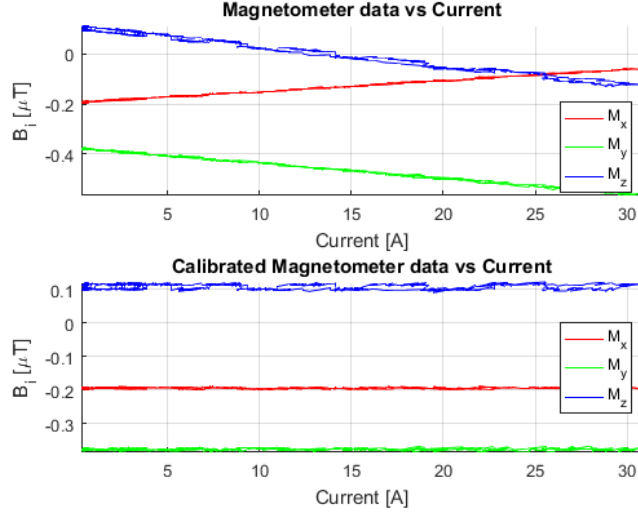


Figure 4.6: Magnetometer compensation for motors magnetic fields.

Figure 4.6 shows the results of calibrating the magnetometer under influence of disturbances generated by the rotating motors. During the calibration the quad-copter is mounted to a fixed object, such that it can not move in any direction. While the quad-copters thrust is slowly increased, current measurements and magnetometer measurements are taken. Since the magnetometer measurements do not contain any of the dynamics of the quad-copter, the change in values can be fully addressed to magnetic field disturbances generated by the rotating motors.

In order to compensate for magnetic disturbances generated by the motors, a first order polynomial is fitted to the magnetometer measurements by utilizing a least squares fit procedure. This relates a change in magnetometer measurement to the measured current. The first order polynomial is of the form:

$$m_{i,j} = a_i c_j + b_i \quad (4.20)$$

Where $m_{i,j}$ is a single magnetometer measurement of a single axis, c_j is a single current measurement. Furthermore, a_i and b_i are the parameters relating the current to the magnetometer measurement. The same relation can also be written in matrix form:

$$\begin{bmatrix} m_{i,1} \\ \vdots \\ m_{i,n} \end{bmatrix} = \begin{bmatrix} c_1 & 1 \\ \vdots & \vdots \\ c_n & 1 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} \quad (4.21)$$

$$M_i = CB_i$$

Where $M_i \in \mathbb{R}^{n \times 1}$ are the obtained magnetometer measurements for axis i , $B_i \in \mathbb{R}^{2 \times 1}$ contains the first order polynomial parameters for axis i and $C \in \mathbb{R}^{n \times 2}$ contains the current measurements of orders 1 and 0, where order 0 results in a column filled with ones. Provided Equation (4.21), the parameters B can be obtained utilizing a least squares fit:

$$B_i = (C^T C)^{-1} C^T M_i \quad (4.22)$$

The parameters a_i and b_i are obtained through the least squares fit method are provided in Table 4.6.

		Magnetometer axis		
		M_x	M_y	M_z
Rate	$\frac{\mu\text{T}}{\text{A}}$	0.0044	-0.0061	-0.0077
Offset	μT	-0.1945	-0.3763	0.1067

Table 4.6: Current calibration results for magnetometer.

The magnetometer corrections $\tilde{M}_{i,j}$ are calculated as in Equation (4.23), where $M_{i,j}$ is the magnetometer measurement of axis i , C_j is the current measurement and a_i is the obtained rate parameter, the obtained offset b_i is not used in correcting the magnetometer measurements. Results of the corrections are shown in the bottom part of Figure 4.6.

$$\tilde{M}_{i,j} = M_{i,j} - a_i C_j \quad (4.23)$$

4.1.6.3 Gyroscope

The gyroscope is calibrated by rotating the sensor multiple times around its axis. By integrating the sensor data, angles are obtained, which should match the number of rotations. By evaluating the angle results of the gyroscope with the actual rotated angles, the magnitude of each gyroscope axis can be calibrated. Since small fluctuations in angle rotations leads to wrongly calibrated axis, it is better practice to rotate the gyroscope multiple rounds. In Figure 4.7, the gyroscope is rotated 10 times around each axis. The integrated angles are evaluated over this movement, which means each rotation starts at 0 radians and ends at $10 \times 2\pi$ radians.

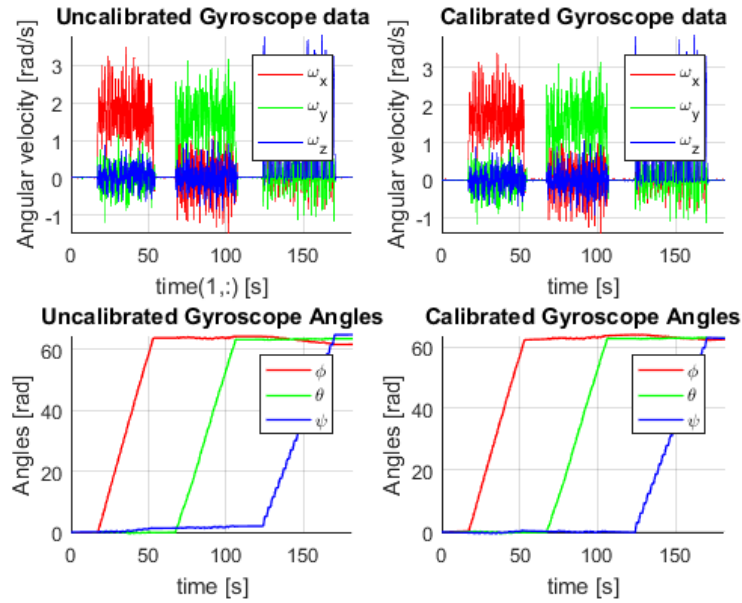


Figure 4.7: Gyroscope calibration results.

Gyroscope axis			
	ω_x $\frac{\text{rad}}{\text{s}}$	ω_y $\frac{\text{rad}}{\text{s}}$	ω_z $\frac{\text{rad}}{\text{s}}$
Bias	-0.0174	0.0010	0.0172
Amplitude	0.9627	0.9934	1.0174

Table 4.7: Calibration results for gyroscope.

In Table 4.7 the calibration results can be seen. The bias is obtained by averaging the measurements before dynamics start. Although the bias is presented here as a parameter, during pre-arm conditions, the bias will be calculated as the average of gyroscope measurements for an axis over a fixed time interval. Furthermore, the amplitude is close to zero, which leads to the conclusion that the gyroscope is already calibrated closely to the actual angular velocities. Furthermore, temperature and environment influences the measurement results of the gyroscope, therefore it is advised to recalibrate the gyroscope on a regular basis.

4.2 State estimation

Various filters can be used to estimate the rotational states of a quad-copter, such as a complementary filter [69], various Kalman filters, Mahoney filter [70] or a Madgwick filter [71]. The nature of the Tech United field results in magnetic disturbances due to the presence of ferromagnetic materials, such as concrete reinforcement present in the floor and the drone cage existing of metal pipes. Therefore, the Madgwick filter is chosen since it is able to reject magnetic disturbed measurements obtained by the magnetometer.

4.2.1 Quaternions

The core of the filter propagates the attitude estimation based on a quaternion representation. Angles can be expressed in Euler/Tait-Bryant angles, quaternions, or rotation matrices, where quaternions and rotation matrices do not contain singularity problems. In control structures quaternions and Euler representations are easier to understand. The Madgwick filter utilizes quaternions. Most of the mathematics related to quaternions can be found in [72]. This section provides only a brief introduction to quaternion based math.

A quaternion is represented as:

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ e_x \sin\left(\frac{\theta}{2}\right) \\ e_y \cos\left(\frac{\theta}{2}\right) \\ e_z \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \quad (4.24)$$

where $\theta \in [0, 2\pi]$ is an angle and $[e_x \ e_y \ e_z]^T$ is a vector with unity norm around which is rotated. The adjoint, the norm, the inverse of a quaternion q and a normalized quaternion \hat{q} are defined as:

$$\begin{aligned} \bar{q} &= [q_0 \ -q_1 \ -q_2 \ -q_3]^T \\ \|q\| &= \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \\ q^{-1} &= \frac{\bar{q}}{\|q\|} \\ \hat{q} &= \frac{q}{\|q\|} \end{aligned} \quad (4.25)$$

Moreover, a multiplication between two quaternions q and p is defined as:

$$\begin{aligned} q \cdot p &= \begin{bmatrix} q_0 p_0 - q_{1:3}^T p_{1:3} \\ q_0 p_{1:3} + p_0 q_{1:3} - q_{1:3} \times p_{1:3} \end{bmatrix} \\ &= \begin{bmatrix} q_0 p_0 - q_1 p_1 - q_2 p_2 - q_3 p_3 \\ q_0 p_1 + q_1 p_0 - q_2 p_3 + q_3 p_2 \\ q_0 p_2 + q_2 p_0 - q_3 p_1 + q_1 p_3 \\ q_0 p_3 + q_3 p_0 - q_1 p_2 + q_2 p_1 \end{bmatrix} \end{aligned} \quad (4.26)$$

A rotation of a vector z can then be described as:

$$\begin{bmatrix} 0 \\ z \end{bmatrix} = q^{-1} \cdot \begin{bmatrix} 0 \\ z \end{bmatrix} \cdot q \quad (4.27)$$

The rotation can be seen as performing half the rotation such that the vector is rotated to an intermediate vector. This operation is performed by pre-multiplying with the quaternion. In order to rotate from the intermediate vector to the final vector, the vector is post multiplied by the inverse of the quaternion. To illustrate the procedure, in Equation (4.28) is a vector $z = [1 \ 0 \ 0]^T$ rotated $\theta = \frac{1}{2}\pi$ radians around an axis $[e_x \ e_y \ e_z]^T = [0 \ 0 \ 1]^T$. The intermediate vector is provided in Equation (4.28) as

$\left[\frac{\sqrt{2}}{2} \quad \frac{\sqrt{2}}{2} \quad 0\right]^T$ which is further rotated to the final vector $\tilde{z} = [0 \quad 1 \quad 0]^T$.

$$\begin{aligned}\tilde{z} &= q \cdot \begin{bmatrix} 0 \\ z \end{bmatrix} \cdot q^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 0 \\ \frac{\sqrt{2}}{2} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 0 \\ -\frac{\sqrt{2}}{2} \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 0 \\ -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}\end{aligned}\tag{4.28}$$

The quaternions can be further extended into matrix representations [72]. This will be out of scope of this research.

Lastly, a quaternion q is related to the cosine direction matrix R . The cosine direction matrix R provides the ability to transform quaternions to Euler angles, where Equation (4.4) relates the cosine direction matrix to the roll ϕ , pitch θ and yaw ψ angles. The quaternion q to cosine direction matrix R is described by:

$$R_q(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}\tag{4.29}$$

4.2.2 Madgwick filter

The Attitude Heading Reference System used in the drone is a Madgwick filter with Magnetic disturbance rejection [71] and [73]. The algorithm utilizes a gradient descent method to fuse accelerometer, magnetometer and gyroscopic data. By integrating the gyroscope data, the bias of the angle grows. The gyroscope measurements $[\omega_x \quad \omega_y \quad \omega_z]^T$ are incorporated in the quaternion derivative by:

$$\begin{aligned}\dot{q}_k &= \frac{1}{2} \hat{q}_{k-1} \cdot S_\omega \\ q_k &= \hat{q}_{k-1} + \dot{q}_k \Delta t\end{aligned}\tag{4.30}$$

where \hat{q}_{k-1} is the quaternion at time $k-1$, \dot{q}_k is the quaternion derivative at time k , q_k is the new quaternion at time k and $S_\omega = [0 \quad \omega_x \quad \omega_y \quad \omega_z]^T$ is the gyroscope measurement vector. At time $k=0$ an initial quaternion is used $\hat{q}_0 = [1 \quad 0 \quad 0 \quad 0]^T$. However, since an addition has taken place, the new quaternion is not of unity norm, therefore, the new quaternion will be normalized to \hat{q}_k . For demonstration purpose, measurements were obtained while rotating the drone around its x , y and z axis from initial conditions to $\frac{1}{2}\pi$ to $-\frac{1}{2}\pi$ and back to initial conditions. The results of integrating the gyroscope measurements are depicted in Figure 4.8. In the lower part of Figure 4.8, quaternions are transformed to Euler angles and as can be seen, drift in all angles ϕ , θ and ψ is present.

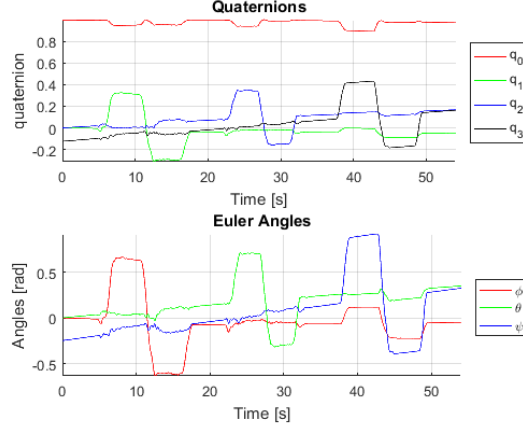


Figure 4.8: Measurement data and angle estimation quad-copter utilizing gyroscope measurements. The quad-copter is rotated from initial conditions to $\frac{1}{2}\pi$ to $-\frac{1}{2}\pi$ and back to initial conditions for each axis separately.

In order to compensate for drift, a gradient descent algorithm is utilized, where accelerometer and magnetometer data corrects for small perturbations in the quaternion derivative. The gradient descent method utilizes a cost function, which needs to be minimized. The cost function f consists of an error between a measured vector S and an expected measurement vector E , which is rotated from earth fixed frame to body frame using the quaternion obtained from previous time step k . In case of the accelerometer, the expected vector $E_{acc} = [0 \ G]^T$ is the gravitational acceleration provided as $G = [0 \ 0 \ 9.81]^T$, which is rotated to body frame by post- and pre-multiplying with previously obtained quaternion \hat{q}_k and its conjugate. The cost function f is described as:

$$f(\hat{q}_k, \hat{E}, \hat{S}_k) = \hat{q}_k^{-1} \cdot \hat{E} \cdot \hat{q}_k - \hat{S}_k \quad (4.31)$$

where $f \in \mathbb{R}^{4 \times 1}$ is the error function, $\hat{S}_k \in \mathbb{R}^{4 \times 1}$ is the normalized sensor measurement, $\hat{q}_k \in \mathbb{R}^{4 \times 1}$ is previous obtained quaternion and $\hat{E} \in \mathbb{R}^{4 \times 1}$ is the measurement vector in earth fixed frame. The cost function to minimize is described by:

$$\min_{\hat{q} \in \mathbb{R}^4} f(\hat{q}_k, \hat{E}, \hat{S}_k) \quad (4.32)$$

where $f(\hat{q}_k, \hat{E}, \hat{S}_k)$ is described by Equation (4.31). In order to solve the optimization problem, a gradient descent algorithm is implemented due to its simple implementation and computability [73]. Therefore, the gradient of the cost function is obtained and is described by:

$$\nabla f(\hat{q}_k, \hat{E}, \hat{S}_k) = J^T(\hat{q}_k, \hat{E}) f(\hat{q}_k, \hat{E}, \hat{S}_k) \quad (4.33)$$

where $J(\hat{q}_k, \hat{E}) = \frac{\partial f}{\partial \hat{q}} \in \mathbb{R}^{4 \times 4}$ is the Jacobian of $f(\hat{q}_k, \hat{E}, \hat{S}_k)$ and $\nabla f(\hat{q}_k, \hat{E}, \hat{S}_k) \in \mathbb{R}^{4 \times 1}$. The gradient of the cost function is incorporated in the quaternion derivative, described by:

$$\dot{\hat{q}} = \frac{1}{2} \hat{q}_{k-1} \cdot S_\omega - \beta \frac{\nabla f}{\|\nabla f\|} \quad (4.34)$$

where $\frac{\nabla f}{\|\nabla f\|}$ is the normalized gradient of the cost function f , β is a parameter tunable by the user such that convergence is faster or slower reached, S_ω is the gyroscope measurement and \hat{q}_{k-1} is previously obtained quaternion. The gradient descent algorithm utilizing the accelerometer can be described by:

$$\hat{q}_k = \begin{bmatrix} \hat{q}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{bmatrix} \quad \hat{S}_{acc,k} = \frac{1}{\sqrt{a_x^2 + a_y^2 + a_z^2}} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \hat{a}_x \\ \hat{a}_y \\ \hat{a}_z \end{bmatrix} \quad \hat{E}_{acc} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (4.35)$$

where the cost function f_{acc} and its Jacobian J_{acc} can be described as:

$$f_{acc}(\hat{q}_k, \hat{E}_{acc}, \hat{S}_{acc,k}) = \begin{bmatrix} 0 \\ 2(q_1q_3 - q_0q_2) - \hat{a}_x \\ 2(q_0q_1 - q_2q_3) - \hat{a}_y \\ 1 - 2(q_1^2 + q_2^2) - \hat{a}_z \end{bmatrix} \quad J_{acc}(\hat{q}_k, \hat{E}_{acc}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -2\hat{q}_2 & 2\hat{q}_3 & -2\hat{q}_0 & 2\hat{q}_1 \\ 2\hat{q}_1 & 2\hat{q}_0 & 2\hat{q}_3 & 2\hat{q}_2 \\ 0 & -4\hat{q}_1 & -4\hat{q}_2 & 0 \end{bmatrix} \quad (4.36)$$

In Figure 4.9 the integrated gyroscope measurements are corrected by using the accelerometer. As depicted in the lower part of Figure 4.9, the drift for angles ϕ and θ is minimized by incorporating accelerometer measurements. However, the yaw ψ is not corrected.

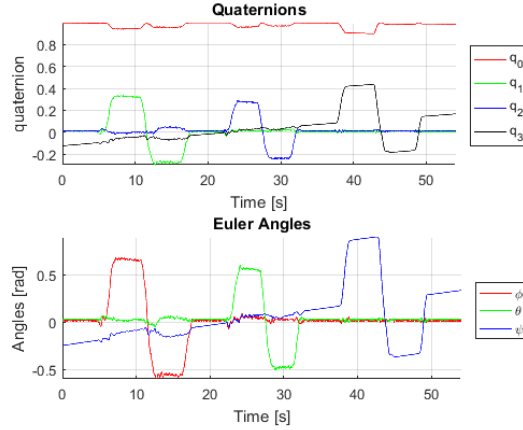


Figure 4.9: Measurement data and angle estimation quad-copter utilizing gyroscope and accelerometer measurements. The quad-copter is rotated from initial conditions to $\frac{1}{2}\pi$ to $-\frac{1}{2}\pi$ and back to initial conditions for each axis separately.

In order to minimize the yaw ψ drift, the gradient descent algorithm is utilized to let the magnetometer correct ψ for drift. In order to let the magnetometer correct ψ , a new cost function is setup, in the form of f_{mag} . However, the expected measurement vector E_{mag} regardless of the inclination of the earths magnetic field. The inclination of the earths magnetic field is defined as the field direction of the magnetic field compared to the earths surface, as depicted in Figure 4.10.

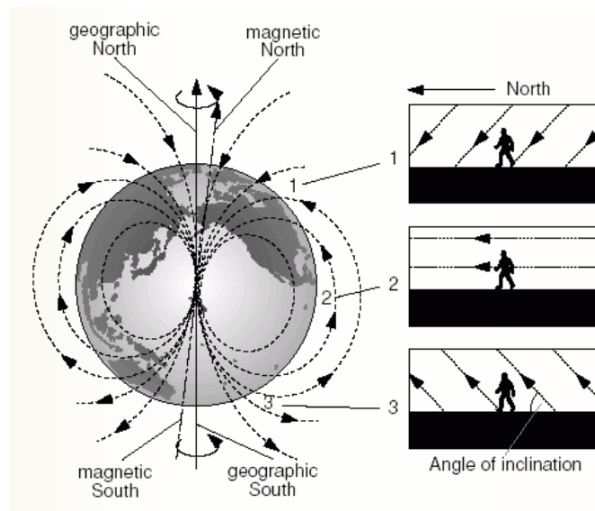


Figure 4.10: Inclination of Earths magnetic field. Provided by [74].

The expected measurement vector E_{mag} is defined, such that it is influenced by the inclination of

earth's magnetic field.

$$\hat{q}_k = \begin{bmatrix} \hat{q}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{bmatrix} \quad \hat{S}_{mag,k} = \frac{1}{\sqrt{m_x^2 + m_y^2 + m_z^2}} \begin{bmatrix} 0 \\ m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{m}_x \\ \hat{m}_y \\ \hat{m}_z \end{bmatrix} \quad (4.37)$$

$$\hat{E}_{mag} = \begin{bmatrix} 0 \\ b_x \\ 0 \\ b_z \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{h_x^2 + h_y^2} \\ 0 \\ h_z \end{bmatrix} \quad h = \begin{bmatrix} 0 \\ h_x \\ h_y \\ h_z \end{bmatrix} = \hat{q}_k \cdot \hat{S}_{mag,k} \cdot \hat{q}_k^{-1}$$

Furthermore, the cost function f_{mag} and its Jacobian J_{mag} can be described as:

$$f_{mag}(\hat{q}_k, \hat{E}_{mag}, \hat{S}_{mag,k}) = \begin{bmatrix} 0 \\ 2b_z(\hat{q}_1\hat{q}_3 - \hat{q}_0\hat{q}_2) - 2b_x(\hat{q}_3^2 + \hat{q}_2^2) + b_x - \hat{m}_x \\ 2b_x(\hat{q}_1\hat{q}_2 - \hat{q}_0\hat{q}_3) + 2b_z(\hat{q}_0\hat{q}_1 + \hat{q}_2\hat{q}_3) - \hat{m}_y \\ 2b_x(\hat{q}_1\hat{q}_3 + \hat{q}_0\hat{q}_2) - 2b_z(\hat{q}_1^2 + \hat{q}_2^2) + b_z - \hat{m}_z \end{bmatrix} \quad (4.38)$$

$$J_{mag}(\hat{q}_k, \hat{E}_{mag}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -2b_z\hat{q}_2 & 2b_z\hat{q}_3 & -4b_x\hat{q}_2 - 2b_z\hat{q}_0 & 2b_z\hat{q}_1 - 4b_x\hat{q}_3 \\ 2b_z\hat{q}_1 - 2b_x\hat{q}_3 & 2b_x\hat{q}_2 + 2b_z\hat{q}_0 & 2b_x\hat{q}_1 + 2b_z\hat{q}_3 & 2b_z\hat{q}_2 - 2b_x\hat{q}_0 \\ 2b_x\hat{q}_2 & 2b_x\hat{q}_3 - 4b_z\hat{q}_1 & 2b_x\hat{q}_0 - 2b_z\hat{q}_2 & 2b_x\hat{q}_1 \end{bmatrix}$$

Provided Equation (4.34) the gradient can be extended with magnetometer contribution. The cost function and its Jacobian which include accelerometer measurements and magnetometer measurement, can be described by:

$$J(\hat{q}_k, \hat{E}_{acc}, \hat{E}_{mag}) = \begin{bmatrix} J_{acc}(\hat{q}_k, \hat{E}_{acc}) \\ J_{mag}(\hat{q}_k, \hat{E}_{mag}) \end{bmatrix} \quad (4.39)$$

$$f(\hat{q}_k, \hat{E}_{acc}, \hat{S}_{acc,k}, \hat{E}_{mag}, \hat{S}_{mag,k}) = \begin{bmatrix} f_{acc}(\hat{q}_k, \hat{E}_{acc}, \hat{S}_{acc,k}) \\ f_{mag}(\hat{q}_k, \hat{E}_{mag}, \hat{S}_{mag,k}) \end{bmatrix}$$

In Figure 4.11 is the magnetometer incorporated utilizing the gradient descent algorithm. As can be seen in the lower part of the Figure 4.11, the ψ drift is removed. However, the ψ is orientated with respect to earth's magnetic north. Therefore, an offset is observed in the ψ angle.

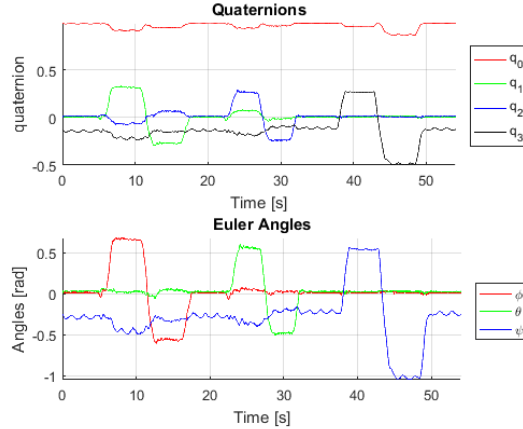


Figure 4.11: Measurement data and angle estimation quad-copter utilizing gyroscope, accelerometer and magnetometer measurements. The quad-copter is rotated from initial conditions to $\frac{1}{2}\pi$ to $-\frac{1}{2}\pi$ and back to initial conditions for each axis separately.

4.2.2.1 Magnetic disturbance severity

Given the magnetometer disturbances, the disturbance severity needs to be determined. Two measurements to validate the disturbance severity are (i) the dip angle θ_{dip} of the measurement which is the

inclination of earths magnetic field and (ii) the magnitude $\|m\|_2$ of the measurement. The magnitude is defined as the 2-norm of the magnetometer measurement, as given in Equation (4.40). The dip angle θ_{dip} is the angle between the horizontal frame and the magnetometer vector. First, the angle between the gravitational acceleration vector in earth frame and the magnetometer measurement is obtained. Since gravitational acceleration vector in earth frame is perpendicular to the horizontal frame, the dip angle can be obtained by subtracting the obtained angle from $\frac{1}{2}\pi$. The dip angle and the magnitude are obtained by:

$$\|m\|_2 = \sqrt{m_x^2 + m_y^2 + m_z^2} \quad (4.40)$$

$$\theta_{dip} = \frac{\pi}{2} - \arccos\left(\frac{A(q)h \cdot G}{\|h\|}\right) \quad (4.41)$$

The dip angle θ_{dip} and magnitude $\|m\|_2$ of the magnetometer are depicted in Figure 4.12. The magnetometer measurements are rotated from body fixed frame to earth fixed frame, which is shown in the top part of Figure 4.12.

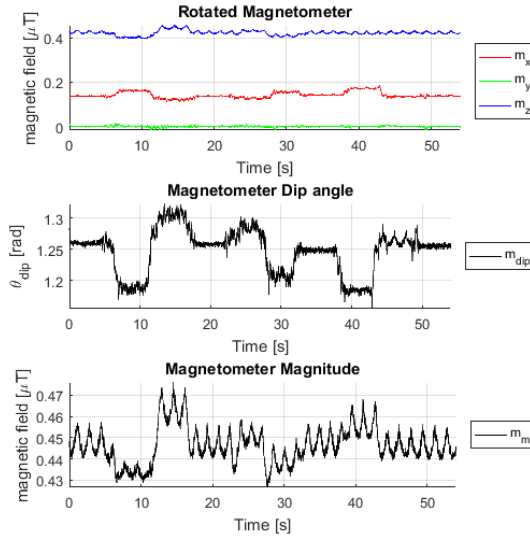


Figure 4.12: Obtained dip angle θ_{dip} and magnitude $\|m\|_2$ from magnetometer measurements.

where G is the gravitational acceleration vector, and $A(q)h$ is the magnetometer measurement rotated to earth frame. The magnetic disturbance based on magnitude $\lambda_1 \in [0, 1]$ quality is described by (4.42), where $m_0 \in \mathbb{R}$ is the expected magnetometer magnitude without any disturbance, which can be obtained during calibration, see Section 4.1.6.1.

$$\lambda_1 = \frac{|\|m\|_2 - m_0|}{m_0} \quad \text{if } \lambda_1 > 1, \lambda_1 = 1 \quad (4.42)$$

The dip angle quality $\lambda_2 \in [0, 1]$ is expressed by (4.43). The expected dip angle $\theta_0 \in \mathbb{R}$ is subtracted from the measured dip angle $\theta_{dip} \in \mathbb{R}$ and scaled by a tunable value $th_{dip} \in \mathbb{R}$. Furthermore, the expected dip angle θ_0 can be obtained by transforming magnetometer measurements from body fixed frame to earth fixed frame and obtaining the dip angle θ_{dip} and where the magnetometer is not influenced by disturbances or drone dynamics. In Figure 4.12, the first 6 seconds of the magnetometer dip angle plot show the expected dip angle $\theta_0 = 0, 4\pi$ radians.

$$\lambda_2 = \frac{|\theta_{dip} - \theta_0|}{th_{dip}} \quad \text{if } \lambda_2 > 1, \lambda_2 = 1 \quad (4.43)$$

Both λ_1 and λ_2 are averaged and used as an indication of the magnetic disturbance severity: λ is used to indicate whether the magnetometer data should be used or not.

$$\lambda = \frac{1}{2}\lambda_1 + \frac{1}{2}\lambda_2 \quad (4.44)$$

Since magnetic disturbance severity is obtained in the form of λ , it needs to be incorporated into the quaternion. From previous estimated quaternion \hat{q}_{k-1} a new state can be obtained \hat{q}_k utilizing the gyroscope measurement S_ω , quaternion derivative \dot{q} see Equations (4.34) and the gradient see Equation (4.33). The new state is obtained in two different manners: (i) the magnetometer is included in the gradient of the cost function $\nabla f_{acc,mag}$ which results in \dot{q}_{MIMU} and (ii) the magnetometer is excluded in the gradient of the cost function ∇f_{acc} which results in \dot{q}_{IMU} . Integrating both derivatives leads to $q_{MIMU,k}$ and $q_{IMU,k}$, respectively as is described by:

$$\begin{aligned} \dot{q}_{MIMU} &= \frac{1}{2} \hat{q}_{k-1} \cdot S_\omega - \beta \frac{\nabla f_{acc,mag}}{\|\nabla f_{acc,mag}\|} & \dot{q}_{IMU} &= \frac{1}{2} \hat{q}_{k-1} \cdot S_\omega - \beta \frac{\nabla f_{acc}}{\|\nabla f_{acc}\|} \\ q_{MIMU,k} &= \hat{q}_{k-1} + \dot{q}_{MIMU} \Delta t & q_{IMU,k} &= \hat{q}_{k-1} + \dot{q}_{IMU} \Delta t \end{aligned} \quad (4.45)$$

The current quaternion q_k is obtained by averaging $\hat{q}_{MIMU,k}$ and $\hat{q}_{IMU,k}$, which are normalized quaternions obtained from $q_{MIMU,k}$ and $q_{IMU,k}$. Weighing both quaternions is based on the obtained magnetic disturbance severity λ and is described by:

$$\begin{aligned} \lambda \hat{q}_{IMU,k} + (1 - \lambda) \hat{q}_{MIMU,k} \\ w_1 \hat{q}_{IMU,k} + w_2 \hat{q}_{MIMU,k} \end{aligned} \quad (4.46)$$

Averaging the quaternions is not a straight forward task. However, in [75] a solution is proposed for averaging two quaternions. The solution proposes an optimization problem, which transforms a quaternion into matrix form. The solution to the optimization problem, utilizing orthogonality principle and Frobenius norms and Davenport's q-method results in a matrix M in which the largest eigenvalues corresponds to the averaged quaternion. A full abbreviation of the math can be found in [75]. As a result, averaging two quaternions can be simplified. The eigenvalues of matrix M are defined as $\lambda_\pm = \frac{1}{2}(w_1 + w_2 \pm z)$, where $z = \sqrt{(w_1 - w_2)^2 + 4w_1w_2(\hat{q}_{IMU,k}^T \hat{q}_{MIMU,k})^2}$ and $w_1 = \lambda$ and $w_2 = 1 - \lambda$. The average quaternion is described by:

$$\bar{q}_k = \pm \left[\sqrt{\frac{w_1(w_1 - w_2 + z)}{z(w_1 + w_2 + z)}} \hat{q}_{IMU,k} + \text{sign}(\hat{q}_{IMU,k}^T \hat{q}_{MIMU,k}) \sqrt{\frac{w_2(w_2 - w_1 + z)}{z(w_1 + w_2 + z)}} \hat{q}_{MIMU,k} \right] \quad (4.47)$$

Lastly, the obtained averaged quaternion \bar{q}_k is normalized and becomes the current quaternion \hat{q}_k .

4.2.2.2 Stationary state detection

The final part of the Madgwick filter detects the stationary state. The stationary state is defined as pre-takeoff flight conditions at which the motors are not rotating. Due to the rotating motors and unbalances existing in the propellers and motors, vibrations will be measured by the accelerometer and gyroscope. Figure 4.13 depicts the stationary state between $t = 0$ and $t = 2.3$ and rotating motors after $t = 2.3$ in which the system is no longer in stationary state, but still in steady state conditions. The quad-copters motors start rotating after $t = 2.3$ and vibrations in the sensors are visible in the form of increased noise. The stationary state is implemented to prevent the obtained angles from integrating gyroscope measurements from drifting. The stationary state is detected, if the gyroscope and accelerometer measurements are between threshold values.

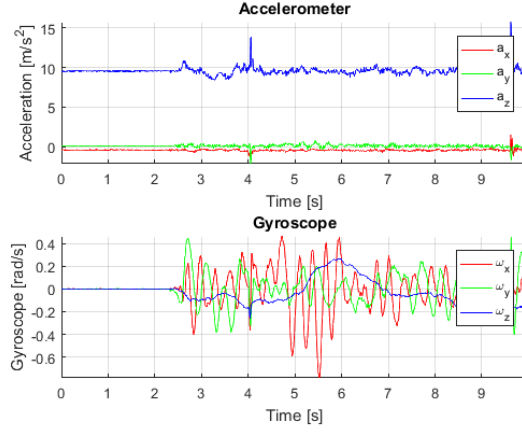


Figure 4.13: Stationary state accelerometer and gyroscope measurements. Between $t = 0$ and $t = 2.3$, the system is in stationary state. After $t = 2.3$

The stationary state detection based on the gyroscope measurements is obtained by taking the absolute value of the gyroscope measurements and determine if they are below the threshold value th_{gyro} . The threshold value th_{gyro} is obtained by obtaining the maximum absolute value of the gyroscope value. The stationary state detection is described by:

$$\begin{aligned}
 |\omega_x| &< th_{gyro} \\
 |\omega_y| &< th_{gyro} \\
 |\omega_z| &< th_{gyro}
 \end{aligned} \tag{4.48}$$

where $th_{gyro} \in \mathbb{R}$ is the threshold value and $\omega \in \mathbb{R}^{3 \times 1}$ are the angular velocities measured by the gyroscope. In Figure 4.14, the gyroscope is in stationary state. By taking $th_{gyro} = 0.005$ radians per second, the whole case is defined as stationary state. In Figure 4.13, after $t = 2.3$ the absolute gyroscope measurements are no longer below the threshold value and the system is no longer in stationary state.

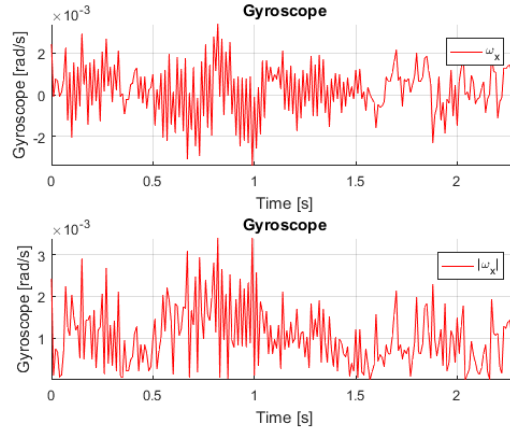


Figure 4.14: Stationary state gyroscope for x -axis only.

The accelerometer stationary state is active when the absolute value of the derivative of accelerometer measurements lies below a threshold value th_a . The threshold value th_a is obtained as the largest absolute derivative value $a_{pp} \in \mathbb{R}$ during stationary state multiplied with a gain $k \in \mathbb{R}$, such that $th_a = k \times a_{pp}$. The stationary state detection based in accelerometer measurements can be described by:

$$\begin{aligned}
 |a_{x,t} - a_{x,t-1}| &< th_a \\
 |a_{y,t} - a_{y,t-1}| &< th_a \\
 |a_{z,t} - a_{z,t-1}| &< th_a
 \end{aligned} \tag{4.49}$$

where $a_{i,t} \in \mathbb{R}$ is current accelerometer measurement for axis i and $a_{i,t-1} \in \mathbb{R}$ is previous accelerometer measurement for axis i , $th_a = k \times a_{pp}$ is the threshold value where $k \in [1.2, 1.5]$ is a tuning value. In Figure 4.15, the accelerometer measurement for axis x is depicted and the absolute value of its derivative in stationary state. The th_a value can be configured to 0.1 and k is set to 1.2. Compared to Figure 4.13, after $t = 2.3$ the absolute value of the derivative of the accelerometer measurements is larger than threshold value th_a . Therefore, the system is no longer in stationary state based on the accelerometer measurements.

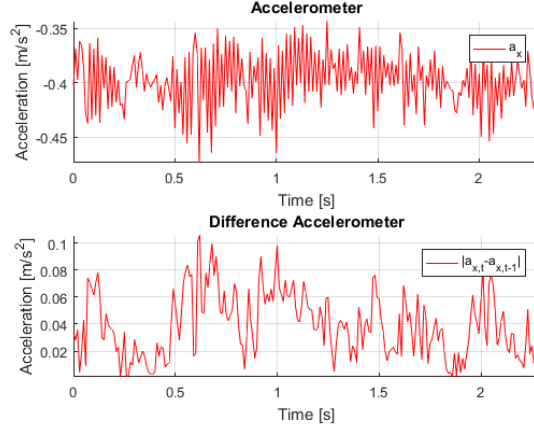


Figure 4.15: Stationary state accelerometer for x -axis only.

The drone is in stationary state if all measurements of both the accelerometer and the gyroscope lie below the specified threshold value, given by Equations (4.48) and (4.49). The algorithm flow can be seen in Figure 4.16, where the first step detects whether the drone is in stationary state or not. The second step determines the magnetic disturbance severity and provides a value $\lambda \in [0, 1]$ whether the IMU algorithm including gyroscope bias should be trusted over the MIMU algorithm including magnetic disturbances, or vice versa.

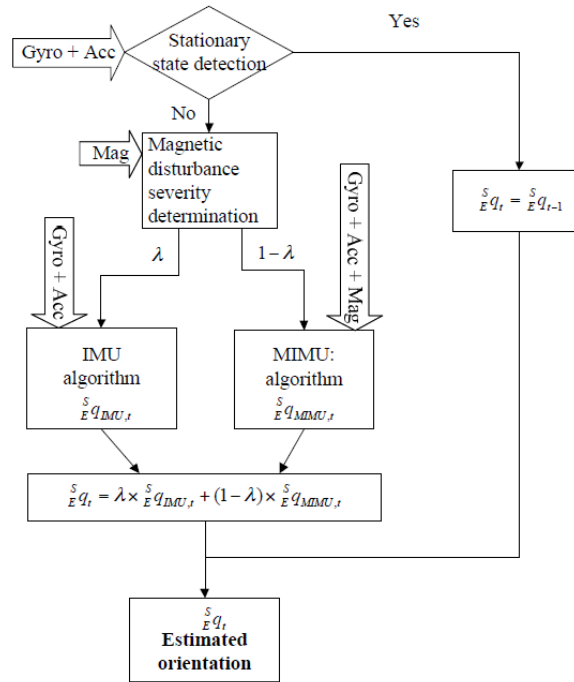


Figure 4.16: Algorithm Flow Madgwick Adaptive Filter. Obtained from [71].

4.2.3 Kalman filter

The drone is equipped with an indoor GPS system from Marvelmind, which has a sampling frequency of 13Hz. Since the control loop runs at 100Hz, a form of extrapolation, filtering or state estimation is needed for the missing data frames. Moreover, in order to use a PID controller, the derivative of the translational states are needed. The most commonly used filter, which estimates states based on multiple sensors, is a Kalman filter.

A Kalman filter is based on the principles that a new state originates from past states, given Equation (4.50). Where $X_t \in \mathbb{R}^n$ are the n states of the system at time t . $A \in \mathbb{R}^{n \times n}$ is the state transition matrix, which links prior states X_{t-1} to the new state X_t . $B \in \mathbb{R}^{n \times m}$ which lets m number of inputs $u_t \in \mathbb{R}^{m \times 1}$ influence n states of the system. A Kalman filter assumes Gaussian white process noise w_t in the system model, which is a zero mean signal, has a normal distribution and a standard deviation σ_w .

$$X_t = Ax_{t-1} + Bu_t + w_t \quad (4.50)$$

Besides the linear model, a Kalman filter utilizes a measurement model, given by Equation (4.51), to further improve the estimated state, where $z_t \in \mathbb{R}^{k \times 1}$ is the measurement vector at time t and $C \in \mathbb{R}^{k \times n}$ is the measurement matrix of the system. A Kalman filter assumes the measurements are disturbed by a Gaussian white noise disturbance $v_t \in \mathbb{R}$ which is a zero mean signal, has a normal distribution and a standard deviation σ_v .

$$z_t = Cx_t + v_t \quad (4.51)$$

The first step in the Kalman filter process is state prediction. Given the previous state $\hat{X}_{k-1|k-1}$ and the control input u_k , the next expected state given the model matrices A and B is provided by Equation (4.52). The covariance matrix $P_{k-1|k-1} \in \mathbb{R}^{n \times n}$ is propagated following Equation (4.53), where A is the state transition matrix and $Q \in \mathbb{R}^{n \times n}$ is the covariance matrix of the process noise w_t .

$$\hat{X}_{k|k-1} = A\hat{X}_{k-1|k-1} + Bu_k \quad (4.52)$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \quad (4.53)$$

The second step in the Kalman filtering process, is obtaining the Kalman gain, which is a measure to shift trust between the model prediction and the sensor measurements, provided the propagated covariance matrix $P_{k|k-1}$, the measurement noise covariance $R \in \mathbb{R}^{k \times k}$ and the measurement matrix C . Where $K \in \mathbb{R}^{n \times k}$ is the Kalman gain.

$$K = P_{k|k-1}C^T (CP_{k|k-1}C^T + R)^{-1} \quad (4.54)$$

The last step of the Kalman filter, is updating the predicted state estimate with a measurement update. Equation (4.55) updates the state estimate $\hat{X}_{k|k}$, using the error between the measurements z and the expected measurement $C\hat{X}_{k|k-1}$ multiplied with the Kalman gain K , where I is an $n \times n$ identity matrix. The covariance $P_{k|k-1}$ is updated, given Equation (4.56). After the update, the estimated state update $\hat{X}_{k|k}$ and covariance update $P_{k|k}$ are stored and are used in the next iteration of the Kalman filter as $\hat{X}_{k-1|k-1}$ and $P_{k-1|k-1}$.

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K \left(z - C\hat{X}_{k|k-1} \right) \quad (4.55)$$

$$P_{k|k} = (I - KC) P_{k|k-1} \quad (4.56)$$

The estimated states of the quad-copter are the translation states in the x , y and z directions. The Kalman filter merges the GPS measurements and the accelerometer data. The relation between the GPS measurements and the accelerometer data, is a double integrator. Prior to the filter, the accelerometer data is transformed to inertial frame, and the gravity component is removed. Therefore, the estimated states $\hat{X}_{k|k}$ can be expressed as the translational discrete states x_k , y_k and z_k and their derivatives \dot{x}_k , \dot{y}_k and \dot{z}_k . The state vector is expressed in Equation (4.57).

$$\hat{X}_{k|k} = [x_k \quad y_k \quad z_k \quad \dot{x}_k \quad \dot{y}_k \quad \dot{z}_k]^T \quad (4.57)$$

The discrete double integrator state space model, regarding the state vector $\hat{X}_{k|k}$, is given by:

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (4.58)$$

The input values of the state space model are given by the accelerometer $u = [a_x \ a_y \ a_z]^T$. Due to the model, the accelerometer data will be used to estimate the velocity and alter the GPS measurements, such that a position update will be provided every 0.01 seconds.

The last part, in order to setup the Kalman filter, is to define covariance matrices. Note that, the accelerometer data is incorporated through the input matrix B . Due to the nature of the B matrix, as given in Equation (4.58), the accelerations are related to the velocities and the positions. Which means that the variance present in the accelerometer data is applied to the model through the B matrix. Therefore, the process covariance matrix Q can be obtained by obtaining the standard deviation of the accelerometer signal and the B matrix, given in Equation (4.59). Where σ_{a_x} is the standard deviation of a_x measurement, σ_{a_y} for a_y and σ_{a_z} for a_z .

$$Q = \begin{bmatrix} \sigma_{a_x}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{a_y}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{a_z}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{a_x}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{a_y}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{a_z}^2 \end{bmatrix} BB^T \quad (4.59)$$

Since the GPS measurements are merged into the Kalman filter via the measurement matrix C , the measurement covariance matrix R is given by Equation (4.60). σ_{GPS_x} is the standard deviation of p_x GPS measurement, σ_{GPS_y} for p_y and σ_{GPS_z} for p_z . The standard deviation can be obtained by measuring the signal in steady state and obtaining the standard deviation.

$$R = \begin{bmatrix} \sigma_{GPS_x}^2 & 0 & 0 \\ 0 & \sigma_{GPS_y}^2 & 0 \\ 0 & 0 & \sigma_{GPS_z}^2 \end{bmatrix} CC^T \quad (4.60)$$

Figure 4.17 show accelerometer measurements and GPS measurements during steady-state conditions. During steady-state conditions, the propellers of the quad-copter were rotating, however, the quad-copter does not take-off. As a result, vibrations due to the unbalanced propellers in the quad-copter are measured at the accelerometer and the Kalman filter takes the vibrations into account. In Figure 4.17a, the GPS measurements are shown and their respective probability density function (PDF). In Figure 4.17b, the accelerometer measurements are shown and their respective PDF. The obtained mean μ values and standard deviations σ are presented in Table 4.8.

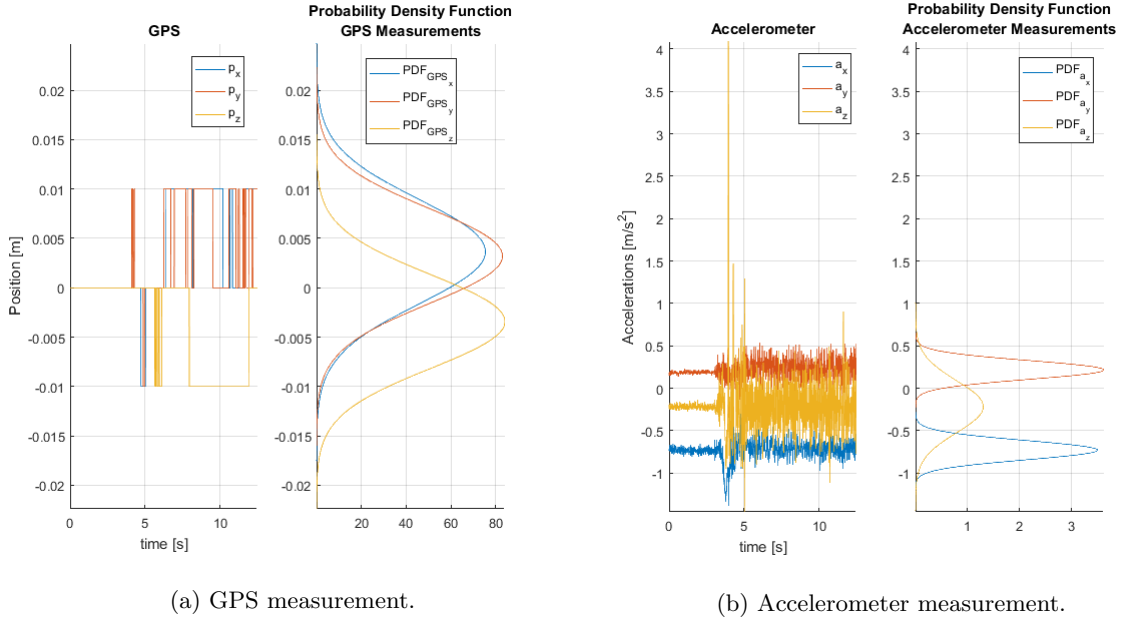


Figure 4.17: Sensor measurements in steady-state conditions and corresponding PDF.

	GPS			Accelerometer		
	X	Y	Z	X	Y	Z
Mean(μ)	0.00366	0.00322	-0.00342	-0.72961	0.21630	-0.21785
Std(σ)	0.00529	0.00481	0.00474	0.11399	0.11016	0.30639

Table 4.8: Mean values μ and standard deviations σ of the accelerometer and GPS measurements.

4.3 Controller design

Controller design is based on the LTI models represented in Equation (4.8). The system models becomes as provided in Equation (4.61) where it is assumed the C-matrix takes the form $C = [1 \ 0]$ for each degree of freedom and the D-matrix is zero.

$$\begin{aligned}
 \begin{bmatrix} \dot{\phi} \\ \dot{\omega}_x \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \omega_x \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J_{xx}} \end{bmatrix} \tau_x & \quad \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{T}{m} \end{bmatrix} \theta \\
 \begin{bmatrix} \dot{\theta} \\ \dot{\omega}_y \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega_y \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J_{yy}} \end{bmatrix} \tau_y & \quad \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{T}{m} \end{bmatrix} \phi \\
 \begin{bmatrix} \dot{\psi} \\ \dot{\omega}_z \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi \\ \omega_z \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J_{zz}} \end{bmatrix} \tau_z & \quad \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} T
 \end{aligned} \tag{4.61}$$

Designing a stabilizing control law is done in three different steps. First a stabilizing control law is designed for the angles. Then a stabilizing control law is designed for altitude control and finally, a stabilizing control law is designed for x and y position control. The system parameters of the quad-copter are shown in Table 4.9, which are obtained by weighing all the lose part, reconstruct the quad-copter in Solidworks and evaluating mass distribution, which can be seen in Appendix M.

Parameter	Value	Parameter	Value
m	1.3660	g	9.81
J_{xx}	0.0304	T^*	-13.4005
J_{yy}	0.0309	f_s	100
J_{zz}	0.0599		

Table 4.9: System parameters of the quad-copter.

The discretized decoupled LTI systems with sampling frequency f_s become:

$$\begin{aligned} \begin{bmatrix} \phi \\ \omega_x \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \phi \\ \omega_x \end{bmatrix}_k + \begin{bmatrix} 0.001645 \\ 0.3289 \end{bmatrix} \tau_x & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} 0.0004905 \\ 0.0981 \end{bmatrix} F_x \\ \begin{bmatrix} \theta \\ \omega_y \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \omega_y \end{bmatrix}_k + \begin{bmatrix} 0.001618 \\ 0.3236 \end{bmatrix} \tau_y & \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_k + \begin{bmatrix} -0.0004905 \\ -0.0981 \end{bmatrix} F_y \\ \begin{bmatrix} \psi \\ \omega_z \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \psi \\ \omega_z \end{bmatrix}_k + \begin{bmatrix} 0.0008347 \\ 0.1669 \end{bmatrix} \tau_z & \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_k + \begin{bmatrix} 0.0000366 \\ 0.007321 \end{bmatrix} T \end{aligned} \quad (4.62)$$

By applying pole placement, the poles of each system can be placed inside the unit disc. The poles are placed close to the edge of the unit disc at $\lambda_i = \{0.95, 0.97\}$. A feedforward steady-state gain is calculated by:

$$M_i = \left(C_i (I - A_i + B_i K_i)^{-1} B_i \right)^{-1} \quad (4.63)$$

The control inputs for each sub system can be written as:

$$\begin{aligned} \tau_x &= -K_\phi \begin{bmatrix} \phi \\ \omega_x \end{bmatrix}_k + M_\phi r_\phi & F_x &= -K_x \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + M_x r_x \\ \tau_y &= -K_\theta \begin{bmatrix} \theta \\ \omega_y \end{bmatrix}_k + M_\theta r_\theta & F_y &= -K_y \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_k + M_y r_y \\ \tau_z &= -K_z \begin{bmatrix} \psi \\ \omega_z \end{bmatrix}_k + M_\psi r_\psi & T &= -K_z \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_k + M_z r_z \end{aligned} \quad (4.64)$$

Furthermore, in case of only angular control, the control inputs τ_x , τ_y and τ_z are applied to the quadcopter. During altitude-hold control, the thrust input T is applied as well to the quadcopter. In case of position control, all inputs are applied to the quadcopter. However, F_x and F_y are uncontrollable due to the under actuated nature of a quadcopter. Therefore, F_x and F_y are used as references for the angles, where $F_x = \theta$ and $F_y = \phi$. Provided system Equations (4.9) for the coupled states, filling in the parameters presented in Table 4.9 and discretizing both models holds:

$$\begin{bmatrix} x_1 \\ x_2 \\ \theta \\ \omega_y \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0.01 & 4.905e^{-4} & 1.635e^{-6} \\ 0 & 1 & 9.81e^{-2} & 4.905e^{-4} \\ 0 & 0 & 1 & 0.01 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \theta \\ \omega_y \end{bmatrix}_k + \begin{bmatrix} 1.323e^{-7} \\ 5.291e^{-5} \\ 1.618e^{-4} \\ 0.3236 \end{bmatrix} \tau_y \quad (4.65)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \phi \\ \omega_x \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0.01 & -4.905e^{-4} & -1.635e^{-6} \\ 0 & 1 & -9.81e^{-2} & -4.905e^{-4} \\ 0 & 0 & 1 & 0.01 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \theta \\ \omega_y \end{bmatrix}_k + \begin{bmatrix} -1.345e^{-7} \\ -5.378e^{-5} \\ 1.645e^{-4} \\ 0.3289 \end{bmatrix} \tau_y \quad (4.66)$$

By filling in the control laws computed before, it can be shown that the cascaded control structure does not compromise stability of the coupled systems, since the eigenvalues still lie inside the unit disc. Therefore, the new control inputs for the coupled systems are:

$$\begin{aligned} \tau_x &= -K_\phi \begin{bmatrix} \phi \\ \omega_x \end{bmatrix}_k + M_\phi \left(-K_y \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_k + M_y r_y \right) &= - \begin{bmatrix} M_\phi K_y & K_\phi \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \phi \\ \omega_x \end{bmatrix}_k + M_\phi M_y r_y \\ \tau_y &= -K_\theta \begin{bmatrix} \theta \\ \omega_y \end{bmatrix}_k + M_\theta \left(-K_x \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + M_x r_x \right) &= - \begin{bmatrix} M_\theta K_x & K_\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \theta \\ \omega_y \end{bmatrix}_k + M_\theta M_x r_x \end{aligned} \quad (4.67)$$

Furthermore, control gains are implemented on the quadcopter. For each degree of freedom, integrators are applied in order to handle model variations. Above all, feedforward terms are added to the control input. The feedforward terms are constants which compensate for gravity in z-direction and handle the weight distribution of the quadcopter. The final control gains can be seen in Table 4.10.

	P	I	D	FF
ϕ	1.00	0.05	0.30	0.00
θ	1.00	0.05	0.30	-0.25
ψ	1.00	0.05	0.45	0.00
X	0.80	0.10	0.60	0.00
Y	0.80	0.10	0.60	0.00
Z	0.80	0.10	0.60	-14.00

Table 4.10: Control gains controller design quad-copter.

Part II

Wind disturbance rejection

Chapter 5

Wind field modeling

Detecting wind regimes in which the quad-copter flies is a complex task. This Chapter explains the basics of wind field modeling. Current flow models for complex objects use Navier-Stokes equations. In Section 5.1 Computational Fluid Dynamics are considered as a model-driven solution for rejecting wind disturbances and as a solution for modeling purposes in simulation. Challenges accompanying Computational Fluid Dynamics, such as numerical solvers, setting up boundary conditions and stability analysis are discussed as well. However, less complex solutions exist to model wind fields. One of the less complex wind field models, named Dryden Turbulence model, is discussed in Section 5.2. Although the Dryden Turbulent model provide realistic wind models, the model is only considered for simulation purposes.

5.1 Computational Fluid Dynamics

In the following subsection, the Navier-Stokes equations will be highlighted to show the necessary functions and dilemmas in simulating a wind flow around object or in contained environments. Although various different examples are provided, which can also be found in literature, the examples are mainly used to highlight the complexity involving in finding solutions to the Navier-Stokes equations. Since the Navier-Stokes equations are complex and highly likely non-linear Partial Differential Equation (PDE), the fact that solutions can only be numerically approximated, results in truncation errors. Furthermore, due to accumulation of the errors, an unstable model might arise from the equations.

5.1.1 Navier-Stokes equations

The governing equations of fluid dynamics consists of three different equations, known as the continuity equation, the momentum equation and the energy conservation equation. The momentum equation is also known as the Navier-Stokes equation, which is a PDE.

The continuity equation is a mass flow conservation of fluids and gases. The continuity equation states that "per unit volume, the sum of all masses flowing in and out per unit time must be equal to the change of mass due to change in density per unit time" [76]. The continuity equation is given by

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) &= 0 \\ \frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} + \frac{\partial (\rho w)}{\partial z} &= 0 \end{aligned} \quad (5.1)$$

where $\rho \in \mathbb{R}$ represent the fluid density, $t \in \mathbb{R}$ is the time, $x \in \mathbb{R}$, $y \in \mathbb{R}$, $z \in \mathbb{R}$ are the 3-dimensional directions, $u \in \mathbb{R}$, $v \in \mathbb{R}$ and $w \in \mathbb{R}$ represents the fluid velocity in x , y and z direction. The momentum equations are also known as the Navier-Stokes equations. A fluid or gas particle reacts on unbalanced applied forces. The net force on a particle will result in an acceleration governing Newton's second law of motion, which states that "the rate of change of momentum of a body is proportional to the unbalanced force acting on it and takes place in the direction of the force" [76]. The moment equations contain a pressure gradient, inertial terms, viscous terms and body force terms. The moment equations are as follows:

$$\begin{aligned} \frac{\partial (\rho u)}{\partial t} + \nabla \cdot (\rho u \vec{V}) &= -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x \\ \frac{\partial (\rho v)}{\partial t} + \nabla \cdot (\rho v \vec{V}) &= -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho f_y \\ \frac{\partial (\rho w)}{\partial t} + \nabla \cdot (\rho w \vec{V}) &= -\frac{\partial p}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} + \rho f_z \end{aligned} \quad (5.2)$$

The last equation is based on energy conservation and is in general derived from thermodynamics, which states that "the rate of change of energy inside the fluid element equals the net flux of heat into the element and the rate of working done on the element due to body and surface forces" [77] and [78]. Essentially, the energy function can be used to apply aerodynamics on objects or in environments where a compressible gas or liquid is flowing. It is given by:

$$\begin{aligned}
\frac{\partial}{\partial t} \left[\rho \left(e + \frac{V^2}{2} \right) \right] + \nabla \cdot \left[\rho \left(e + \frac{V^2}{2} \vec{V} \right) \right] &= \rho \dot{q} + \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) \\
&\quad - \frac{\partial (up)}{\partial x} - \frac{\partial (vp)}{\partial y} - \frac{\partial (wp)}{\partial z} \\
&\quad + \frac{\partial (u\tau_{xx})}{\partial x} + \frac{\partial (u\tau_{yx})}{\partial y} + \frac{\partial (u\tau_{zx})}{\partial z} \\
&\quad + \frac{\partial (v\tau_{xy})}{\partial x} + \frac{\partial (v\tau_{yy})}{\partial y} + \frac{\partial (v\tau_{zy})}{\partial z} \\
&\quad + \frac{\partial (w\tau_{xz})}{\partial x} + \frac{\partial (w\tau_{yz})}{\partial y} + \frac{\partial (w\tau_{zz})}{\partial z} \\
&\quad + \rho \vec{f} \cdot \vec{V}
\end{aligned} \tag{5.3}$$

where

$$\begin{aligned}
\tau_{xx} &= \lambda \nabla \cdot \vec{V} + 2\mu \frac{\partial u}{\partial x} & \tau_{xy} &= \tau_{yx} = \mu \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) \\
\tau_{yy} &= \lambda \nabla \cdot \vec{V} + 2\mu \frac{\partial v}{\partial y} & \tau_{xz} &= \tau_{zx} = \mu \left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \right) \\
\tau_{zz} &= \lambda \nabla \cdot \vec{V} + 2\mu \frac{\partial w}{\partial z} & \tau_{yz} &= \tau_{zy} = \mu \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right)
\end{aligned} \tag{5.4}$$

in all above equations, $\vec{V} = [u \ v \ w]^T$ are the velocities in the $[x \ y \ z]^T$ directions, respectively. t is the time, ρ is the density of the gas or liquid p is the pressure, f_i is the body force in the particular direction. The energy is marked with e and the temperature by T . Furthermore, k , μ and λ are the thermal conductivity, the molecular viscosity and the bulk viscosity. In some literature the bulk viscosity is taken as $\lambda = -\frac{2}{3}\mu$, which is hypothesized by Stokes but is not been definitely confirmed [77].

Although the model includes temperature changes and density changes, for aerodynamics purposes they can be assumed constant over time. In such case the fluid or gas is assumed to be viscous incompressible flow. For such cases, there exist numerical models, which allow to compute the flow of an incompressible flow [79].

By discarding the energy conservation equation, and therefore excluding density changes and temperature changes over time results in a model consisting of only the continuity Equation (5.3) and the Navier-Stokes Equation (5.4). Which means four variables are calculated with only three equations, and a constraint. One way to convert the equations and the constraint to four equations, is by applying the Poisson effect [79], by which the Navier-Stokes equations are differentiated to their specific directions and summed. For simplicity reasons a 2D situation will be considered. Furthermore, the bulk velocity is set to be the negative molecular viscosity $\lambda = -\mu$, the continuity equation and the 2D Navier-Stokes equations becomes:

$$\begin{aligned}
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \\
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial x} + \frac{\mu}{\rho} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f_x \\
\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial y} + \frac{\mu}{\rho} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + f_y
\end{aligned} \tag{5.5}$$

which provides two equations and a constraint for three unknowns, the flow velocities (u, v) and the pressure p . By setting the body forces to zero $f_x = f_y = 0$ and differentiating in the direction, a

relation can be obtained for the pressure and flow velocities in which the continuity constraint can be incorporated.

$$\begin{aligned} & \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial y} \left(\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right) \\ &= \frac{\partial}{\partial x} \left(-\frac{1}{\rho} \frac{\partial p}{\partial x} + \frac{\mu}{\rho} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \right) + \frac{\partial}{\partial y} \left(-\frac{1}{\rho} \frac{\partial p}{\partial y} + \frac{\mu}{\rho} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \right) \end{aligned} \quad (5.6)$$

which can be written as:

$$\begin{aligned} & \left(\frac{\partial}{\partial t} + u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} \right) \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) + \frac{\partial^2 u}{\partial x^2} + 2 \frac{\partial v}{\partial x} \frac{\partial u}{\partial y} + \frac{\partial^2 v}{\partial y^2} \\ &= -\frac{1}{\rho} \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \right) + \frac{\mu}{\rho} \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \end{aligned} \quad (5.7)$$

By using the continuity equation and the already obtained Navier-Stokes equations, a incompressible flow can be modeled as

$$\begin{aligned} \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial x} + \frac{\mu}{\rho} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f_x \\ \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial y} + \frac{\mu}{\rho} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + f_y \\ \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} &= -\rho \left(\frac{\partial^2 u}{\partial x^2} + 2 \frac{\partial v}{\partial x} \frac{\partial u}{\partial y} + \frac{\partial^2 v}{\partial y^2} \right) \end{aligned} \quad (5.8)$$

The Equations in (5.8) can be numerically simulated using finite difference schemes. Those finite difference schemes can also be found in the literature [76], [77], [79] and [80].

5.1.2 Numerically solving Navier-Stokes equations

The Navier-Stokes equations can be solved in various different manners. The most common known methods are Finite Difference Method (FDM), Finite Volume Method (FVM) and Finite Element Method (FEM). Although other methods exist, such as spectral methods, only FDM, FVM and FEM will be highlighted in this section based on [81].

A Finite Difference Method (FDM) is based on a Taylor series of the derivative. From the derivative a forward difference scheme, a central difference scheme and a backward difference scheme can be obtained. Provided the following derivative:

$$\frac{\partial u(x)}{\partial x} = \lim_{\Delta x \rightarrow 0} \frac{u(x + \Delta x) - u(x)}{\Delta x} \quad (5.9)$$

$$\begin{aligned} \text{Forward difference: } \left(\frac{\partial u}{\partial x} \right) &= \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x) \\ \text{Central difference: } \left(\frac{\partial u}{\partial x} \right) &= \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x^2) \\ \text{Backward difference: } \left(\frac{\partial u}{\partial x} \right) &= \frac{u_i - u_{i-1}}{\Delta x} + O(\Delta x) \end{aligned} \quad (5.10)$$

All difference schemes are obtained from [81]. Most importantly to note is the truncation error O written at the end. In case of a central difference scheme, the truncation error is a second order truncation error, where the forward and backward difference only have a first order truncation error. Although the above example is with a first order Taylor series, the FDM can also be designed with higher order difference schemes, which can all be found in [81].

A second method to solve a PDE is called Finite Element Method (FEM). A FEM discretization starts with dividing the domain Ω in elements e_i . Between each element, nodes are in place, which maintain boundary conditions Γ_i .

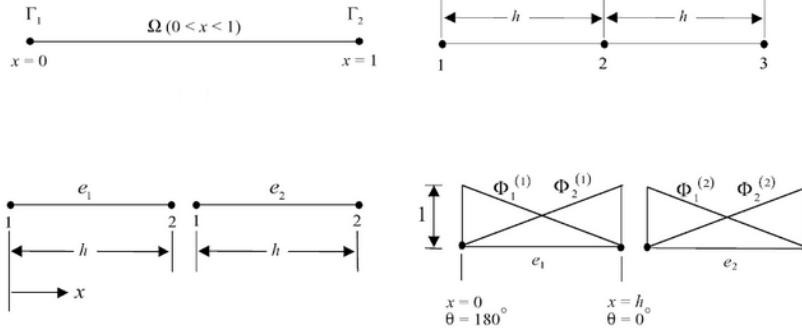


Figure 5.1: Finite element discretization for one-dimensional linear problem with two local elements. (Top left) Given domain (Ω) with boundaries (Γ_1 ($x = 0$), Γ_2 ($x = 1$)). (Top right) Global nodes ($\alpha, \beta = 1, 2, 3$). (Bottom left) Local elements ($N, M = 1, 2$). (Bottom right) Local trial functions. Figure obtained from [81].

Provided the discretization in Figure 5.1, a solution can be obtained by for instance using a standard Galerkin method which assumes that variable $u^{(e)}(x)$ is a linear function in the form of Equation (5.11), which is shown in the bottom right of Figure 5.1. That is,

$$\begin{aligned}
 u^{(e)}(x) &= \alpha_1 + \alpha_2 x \\
 &= \left(1 - \frac{x}{h}\right) u_1^{(e)} + \left(\frac{x}{h}\right) u_2^{(e)} \\
 &= \Phi_N^{(e)}(x) u_N^e \quad (N = 1, 2)
 \end{aligned} \tag{5.11}$$

Furthermore, integrating the solution over the domain sizes, between 0 and h , we obtain:

$$\Phi_N^{(e)} \frac{du}{dx} \Big|_0^h - \left[\int_0^h \frac{d\Phi_N^{(e)}(x)}{dx} \frac{d\Phi_M^{(e)}(x)}{dx} dx \right] u_M^{(e)} - \int_0^h 2\Phi_N^{(e)}(x) dx = 0 \quad (N, M = 1, 2) \tag{5.12}$$

which is known as the variational equation or weak form of the governing. Furthermore, the first term is known as the Niemann boundary condition. A full explanation about the boundary condition can be found in [81]. Mainly the Niemann boundary condition acts as a Dirac delta function, since the boundary condition is 1 if the function is applied at node N , otherwise it is 0. The final scheme is in the form of

$$K_{NM}^{(e)} u_M^{(e)} = F_N^{(e)} + G_N^{(e)} \quad (N, M = 1, 2) \tag{5.13}$$

where $K_{NM}^{(e)}$ is the stiffness matrix, or better called, diffusion or Viscosity matrix. $F_N^{(e)}$ is the source vector and $G_N^{(e)}$ is the Neumann boundary vector. The matrices can be described by:

$$K_{NM}^{(e)} = \int_0^h \frac{d\Phi_N^{(e)}(x)}{dx} \frac{d\Phi_M^{(e)}(x)}{dx} dx = \begin{bmatrix} \int_0^h \frac{d\Phi_1^{(e)}(x)}{dx} \frac{d\Phi_1^{(e)}(x)}{dx} dx & \int_0^h \frac{d\Phi_1^{(e)}(x)}{dx} \frac{d\Phi_2^{(e)}(x)}{dx} dx \\ \int_0^h \frac{d\Phi_2^{(e)}(x)}{dx} \frac{d\Phi_1^{(e)}(x)}{dx} dx & \int_0^h \frac{d\Phi_2^{(e)}(x)}{dx} \frac{d\Phi_2^{(e)}(x)}{dx} dx \end{bmatrix} \tag{5.14}$$

$$= \begin{bmatrix} K_{11}^{(e)} & K_{12}^{(e)} \\ K_{21}^{(e)} & K_{22}^{(e)} \end{bmatrix} = \frac{1}{h} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{5.15}$$

$$F_N^{(e)} = - \int_0^h 2\Phi_N^{(e)} dx = -h \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{5.16}$$

$$G_N^{(e)} = \Phi_N^{(e)} \frac{du}{dx} \Big|_0^h = \Phi_N^{(e)} \frac{du}{dx} \cos(\theta) \tag{5.17}$$

The equations holds for the local element as shown in the bottom right part of Figure 5.1. In order to obtain the final global equations for the global nodes ($\alpha, \beta = 1, 2, 3$), the equations can be summed,

which results in:

$$K_{\alpha\beta} = \begin{bmatrix} K_{11}^{(1)} & K_{12}^{(1)} & 0 \\ K_{21}^{(1)} & K_{22}^{(1)} + K_{11}^{(2)} & K_{12}^{(2)} \\ 0 & K_{21}^{(2)} & K_{22}^{(2)} \end{bmatrix} = \frac{1}{h} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad (5.18)$$

$$F_{\alpha} = \begin{bmatrix} F_1^{(1)} \\ F_2^{(1)} + F_1^{(2)} \\ F_2^{(2)} \end{bmatrix} = -h \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad (5.19)$$

$$G_{\alpha} = \begin{bmatrix} G_1^{(1)} \\ G_2^{(1)} + G_1^{(2)} \\ G_2^{(2)} \end{bmatrix} = \begin{bmatrix} \Phi_1^{*(1)} \\ \Phi_2^{*(1)} + \Phi_1^{*(2)} \\ \Phi_2^{*(2)} \end{bmatrix} \frac{du}{dx} \cos(\theta) \quad (5.20)$$

One last remark about FEM is that if only the elements are analyzed for a first order linear function of $u^{(e)}(x)$, FEM has the same structure as a FDM. Furthermore, FEM in combination with a linear function for $u^{(e)}(x)$ is not suitable for complex non-linear flow mechanics. In that case different function schemes for $u^{(e)}(x)$ need to be selected, which is explained in [81] in more detail.

The last method highlighted is the Finite Volume Method (FVM). The FVM scheme can be derived from both FEM and FDM. The FVM exists of control surfaces and control volumes. The control volumes are related to each other via the control surfaces, as is shown in Figure 5.2.

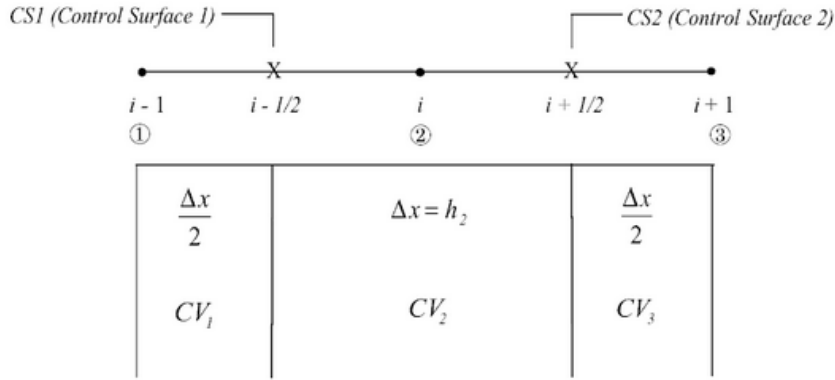


Figure 5.2: Finite Volume approximations. Figure obtained from [81].

The FVM scheme depends on the global form of FEM, which is written as:

$$(\Phi_{\alpha}, R) = \int_0^1 (1) \left(\frac{d^2 u}{dx^2} - 2 \right) dx = 0, \quad 0 < x < 1 \quad (5.21)$$

Integrating the function yields:

$$\sum_{CS1,2} \frac{\Delta u}{\Delta x} - \sum_{CV2} 2\Delta x = 0 \quad (5.22)$$

where the limits are now set by the control surfaces between $i - \frac{1}{2}$ and $i + \frac{1}{2}$, which is explained in [81]. Furthermore, the equation can be written as

$$\frac{u_{i+1} - u_i}{\Delta x} - \frac{u_i - u_{i-1}}{\Delta x} = 2\Delta x \quad (5.23)$$

The equation basically subtracts control surface 1 from control surface 2, in order to obtain control volume 2. By dividing the equation by Δx , the function can be written as the FDM presented earlier.

More information about FDM, FEM and FVM can be found in [81]. The literature further summarizes the differences between FDM, FEM and FVM as:

1. FDM

- (a) Easy to formulate.
- (b) For multidimensional problems, meshes must be structured in either two or three dimensions. Curved meshes must be transformed into orthogonal cartesian coordinates so that finite difference equations can be written in structured cartesian meshes.
- (c) Neumann boundary conditions can only be approximated, not exactly enforced.

2. FEM

- (a) Underlying principles and formulations require a mathematical rigor.
- (b) Complex geometries and unstructured meshes are easily accommodated, no coordinate transformations needed.
- (c) Neumann boundary conditions are enforced exactly.

3. FVM

- (a) Formulations can be based on either FDM or FEM.
- (b) Surface integrals of normal fluxes guarantee the conservation properties throughout the domain.
- (c) Complex geometries and unstructured meshes are easily accommodated, no coordinate transformations needed.

Although some numerical computations can have accurate results, such as FEM, where boundary conditions are enforced compared to FDM, they still are numerical approximations, which do not necessarily guarantee an accurate model with high computation times.

5.1.3 Boundary conditions

Besides a pressure equation and the velocity equations most likely boundary conditions are used to simulate either a closed environment, such as a pipe or a container, or to simulate objects in flow, such as quad-copters. Examples of such boundary conditions can be found in [78]. Figure 5.3 shows a simple body in a flow. For simplicity reasons, the body has only a length in y-direction and has no thickness.

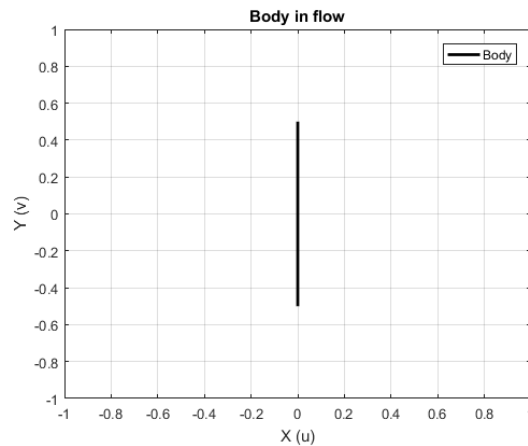


Figure 5.3: Simple body in a flow

Along the body lie the boundary conditions. In general, the proposed boundary conditions are as presented in Equation (5.24). The boundary conditions holds that along the body due to the friction, the flow velocity needs to be zero $v = 0$. Since a flow can not go straight through the body, the velocity in x-direction needs to be zero as well, $u = 0$. Furthermore, the pressure differential at the boundary layer needs to be zero along the body, since it otherwise results in a flow.

$$\begin{aligned} u, v = 0 \text{ at } x = 0 \text{ and } -0.5 \leq y \leq 0.5 \\ \frac{\partial p}{\partial y} = 0 \end{aligned} \tag{5.24}$$

Although the suggested boundary conditions are in a simple form, the conditions become complex if complex body geometries are used. In specialized CFD software packages, high order meshes are used to simulate flow, and for each mesh area around a body, boundary conditions are in place. Furthermore, the suggested boundary conditions are of a simple form, but more complex forms can be used as well [82].

5.1.4 Stability analysis

Another important step in the process of CFD is stability analysis. Since the user sets the grid size and the time step size, the computations might get many rounding errors, resulting in a growing error during simulation. An example is provided at [77], where the following model PDE is used and its finite difference scheme.

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} \\ \frac{u_i^{n+1} - u_i^n}{\Delta t} &= \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}\end{aligned}\quad (5.25)$$

Since the finite difference scheme is used in a Taylor's series, a truncation error is present, since all higher order terms of the Taylor's series are truncated.

$$\frac{\partial u}{\partial t} - \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\partial^2 u}{\partial x^2} - \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} + \left[- \left(\frac{\partial^2 u}{\partial t^2} \right)_i \frac{\Delta t}{2} + \left(\frac{\partial^4 u}{\partial x^4} \right)_i \frac{(\Delta x)^2}{12} + \dots \right] \quad (5.26)$$

where the truncation error is provided by:

$$O \left[\Delta t, (\Delta x)^2 \right] = \left[- \left(\frac{\partial^2 u}{\partial t^2} \right)_i \frac{\Delta t}{2} + \left(\frac{\partial^4 u}{\partial x^4} \right)_i \frac{(\Delta x)^2}{12} + \dots \right] \quad (5.27)$$

As can be seen, all the terms in the truncation error depends on Δt and Δx . Which means if an accurate model, with minimal error is desired, and therefore the truncation error should vanish, both deltas need to be set to zero $\Delta t \approx 0$ and $\Delta x \approx 0$. However, this only increases the number of computations running the CFD scheme.

Furthermore, during simulation the stability of the difference equation needs to be considered. Due to all rounding errors, which are created since an approximation is made of the PDE, it might occur the error will blow up during simulation. The round off error is provided in [77] and is defined as the difference between the "numerical solution from a real computer with finite accuracy" and the "exact solution of the difference equation", which is given as $\epsilon = N - D$. A full explanation of obtaining the error analysis can be found in [77] and it is assumed the error satisfies the difference equation.

$$\frac{\epsilon_i^{n+1} - \epsilon_i^n}{\Delta t} = \frac{\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n}{(\Delta x)^2} \quad (5.28)$$

Stability is obtained, if the error becomes smaller during the progression of the solution from step n to step $n + 1$. If the error grows during progression to the solution, then the numerical solution is unstable. This means the following conditions needs to be satisfied:

$$\left| \frac{\epsilon_i^{n+1}}{\epsilon_i^n} \right| \leq 1 \quad (5.29)$$

Since the assumption is made that a Fourier series can approximate the errors along the x-axis and the time-wise variation can be expressed in an exponential in t [77].

$$\epsilon(x, t) = e^{at} \sum_m e^{ik_m x} \quad (5.30)$$

This results in that the grow rate of the error can be expressed as:

$$\left| \frac{\epsilon_i^{n+1}}{\epsilon_i^n} \right| = |e^{a\Delta t}| = \left| 1 - \frac{4\Delta t}{(\Delta x)^2} \sin^2 \left(\frac{k_m \Delta x}{2} \right) \right| \leq 1 \quad (5.31)$$

This equation results in two different conditions, which both need to be met.

$$0 \leq \frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2} \quad (5.32)$$

The whole procedure of this stability analysis example can be found in [77]. This concludes that caution is needed in CFD analysis. Although it is a powerful tool to analyze for instance wind flows around an object, it is sensitive to minor changes in parameters and they might result in complete different solutions.

5.2 Dryden Turbulent Model

As is described in Section 1.3.3.1, articles [15], [16] and [19] make use of a frozen Dryden turbulence model, which contains a mean value for constant wind flow, a set of sinusoids for predictable wind changes and a set random processes to simulate turbulence. The Dryden turbulence model can be expressed as

$$w_{(\cdot)} = w_{(\cdot),0} + \sum_{n=1}^N a_{(\cdot),n} \sin(\Omega_{(\cdot),n}s + \phi_{(\cdot),n}) \quad (5.33)$$

where w is the wind and (\cdot) denotes a direction component, such as x, y, z , $w_{(\cdot),0}$ is the constant element of the wind. The choice of coefficient $a_{(\cdot),n}$ defines the power spectral density of the wind. $\Omega_{(\cdot),n}$ is the frequency of the wind, s and $\phi_{(\cdot),n}$ simulates the random process of the wind. The power spectral densities are defined as $\Phi_u(\Omega)$ and $\Phi_w(\Omega)$, which are the x and y directions of the wind and the z direction of the wind, respectively.

$$\begin{aligned} \Phi_u(\Omega) &= \sigma_u^2 \frac{2L_u}{\pi} \frac{1}{1 + (L_u\Omega)^2} \\ \Phi_w(\Omega) &= \sigma_w^2 \frac{L_w}{\pi} \frac{1 + 3(L_w\Omega)^2}{(1 + (L_w\Omega)^2)^2} \end{aligned} \quad (5.34)$$

where σ_u and σ_w represents the turbulence intensities, L_u and L_w represents the turbulence scale lengths in x, y direction and z direction respectively. Furthermore, the Frozen Dryden Turbulence model is based on military specifications of wind models, [20], [21] and [22]. For low altitudes, below the 1000 feet, the scale lengths and the disturbance intensities are defined as:

$$\begin{aligned} \frac{L_u}{L_w} &= \frac{1}{(0.177 + 0.000823h)^{1.2}} \\ \frac{\sigma_u}{\sigma_w} &= \frac{1}{(0.177 + 0.000823h)^{0.4}} \end{aligned} \quad (5.35)$$

where $L_w = h$ is the vertical scale in feet, and the turbulence can be set at $\sigma = 0.1w_h$ where w_h is the wind speed at height h . The amplitude is given as:

$$a_n = \sqrt{\Delta\Omega_n \Phi(\Omega_n)} \quad (5.36)$$

Chapter 6

Disturbance rejection

In this Chapter disturbance rejection solutions will be discussed and in particular solutions based on the Internal Model Principle (IMP) are considered. The described Internal Model Principle (IMP) controller design is based on [28]. In Section 6.1 the continuous time case will be considered.

The disturbance can be modeled as an exogenous system to which an IMP is designed to reject the disturbance. In Section 6.2 the design procedure is explained for discrete time systems. Lastly, in Section 6.3 a repetitive controller, which can be seen as a special case of an internal model principle based controller is synthesized. The repetitive control uses a memory of the control input, in order to minimize repetitive tracking errors.

6.1 Continuous time case Internal Model Principle

Firstly, the continuous time case will be considered for a linear time-invariant (LTI) system. For convenience, this section will make use of a general system taking the following form:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \\ y_c &= C_c x \end{aligned} \tag{6.1}$$

where $x \in \mathbb{R}^m$ is the state and $u \in \mathbb{R}^n$ is the input of the system, $y \in \mathbb{R}^o$ are the measurements of the system and $y_c \in \mathbb{R}^p$ are the measured states, which needs to track a specified desired reference. Furthermore, $A \in \mathbb{R}^{m \times m}$ is the state transition matrix, $B \in \mathbb{R}^{m \times n}$ is the input matrix, $C \in \mathbb{R}^{o \times m}$ the measurement matrix and $C_c \in \mathbb{R}^{p \times m}$ is a measurement matrix only selecting specific measurement or a combination of them.

In the following subsections an observer is designed for the unmeasured states and a controller combining feedback and feedforward is designed, which will bring the states measured in y_c to the desired reference in steady state. Furthermore, a disturbance is applied to the system generated by an exogenous system. Although a reference can be generated as an exogenous system in order to improve tracking performance, it is not treated in this project. A disturbance rejection based on IMP is used to cope with the applied disturbances. Throughout Section 6.1 and Section 6.2, the control architecture depicted in Figure 6.1 is used.

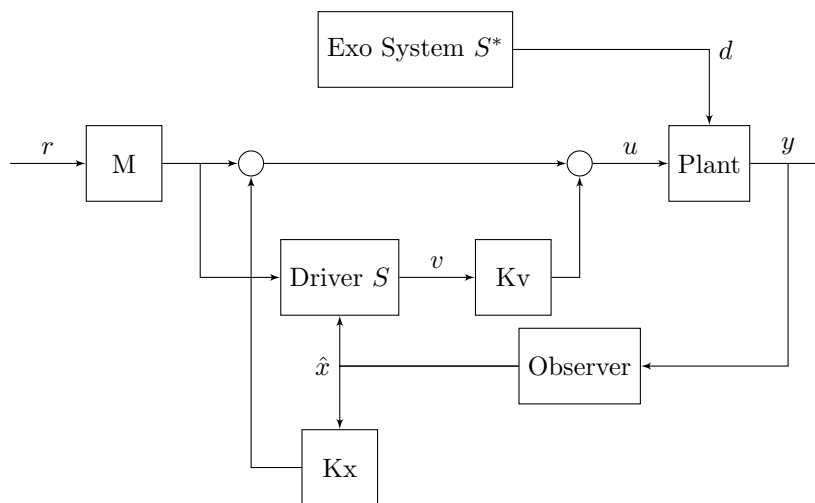


Figure 6.1: Block diagram of an Internal Model Principle controller.

6.1.1 Controller, observer and feedforward design

Provided a continuous time LTI system, described by (6.1) a stabilizing control law is designed to stabilize the system. Various methods can be used to stabilize the system, in this case pole placement is used. The stabilizing control law is described by:

$$u = -Kx \quad (6.2)$$

Provided the system matrices $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{m \times n}$, m poles are placed in the left half plane. Furthermore, it is assumed that (A, B) is controllable, then $K \in \mathbb{R}^{n \times m}$, is the state feedback law, which makes $A - BK$ Hurwitz.

Since not all the states are measured, a full state observer is designed to estimate the remainder states, which are not measured. The estimated states replace the actual states in the control law. The observer is provided in (6.3).

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + Bu + L(y - \hat{y}) \\ \hat{y} &= C\hat{x} \end{aligned} \quad (6.3)$$

The error $(x - \hat{x})$ converges to zero if the poles of $(A - LC)$ are in the open left half plane and the matrix is therefore is Hurwitz. This can be achieved by designing L using for instance pole placement. It is however assumed (A, C) is observable. The error dynamics of the observer are shown in Equation (6.4), which is a combination of the general system (6.1) and the proposed observer (6.3).

$$\dot{x} - \dot{\hat{x}} = (A - LC)(x - \hat{x}) \quad (6.4)$$

In order to design a feedforward which converges the steady state error to the reference $\lim_{t \rightarrow \infty} y_c = r$. An augmented system is used to incorporate the plant model, Equation (6.1), and the observer, Equation (6.3). The control law provided in (6.2) is extended by using the observed state and the reference and presented in equation (6.5).

$$u = -K\hat{x} + Mr \quad (6.5)$$

The augmented model, including the general state space system, the new feedback law and observer becomes:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{\hat{x}} \end{bmatrix} &= \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix} Mr \\ y &= \begin{bmatrix} C & O_{o \times m} \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \end{bmatrix} \\ y_c &= \begin{bmatrix} C_c & O_{o \times m} \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \end{bmatrix} \end{aligned} \quad (6.6)$$

The steady state response can be obtained by making $\begin{bmatrix} \dot{x} \\ \dot{\hat{x}} \end{bmatrix}^T = 0$, which means the system has reached a state in which it will stay if no disturbances are applied or inputs changed. The steady state response can be written as:

$$y_c = \begin{bmatrix} C_c & O_{o \times m} \end{bmatrix} \left(- \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \right)^{-1} \begin{bmatrix} B \\ B \end{bmatrix} Mr \quad (6.7)$$

A steady state is defined as Equation (6.8), which means over time, if the reference r does not change, the output y_c converges to the reference.

$$\lim_{t \rightarrow \infty} y_c = r \text{ for } \dot{r} = 0 \quad (6.8)$$

This condition holds, if $y_c = Ir$ in Equation (6.7). Therefore, M should be chosen as in Equation (6.9), which makes the steady state response shown in Equation (6.7) an identity matrix.

$$M = \left(\begin{bmatrix} C_c & O_{o \times m} \end{bmatrix} \left(- \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \right)^{-1} \begin{bmatrix} B \\ B \end{bmatrix} \right)^{-1} \quad (6.9)$$

The inversion of M only holds, when the matrix is full rank. The matrix is full rank if the system (C_c, A, B) has no invariant zero at $s = 0$. In order to determine if the system (C_c, A, B) contains an invariant zero, the Rosenbrock's system matrix can be used. The Rosenbrock's system matrix is described by:

$$P(s) = \begin{bmatrix} sI - A & B \\ -C & D \end{bmatrix} \quad (6.10)$$

If the matrix $P(s)$ loses rank if $s = 0$, then the system contains an invariant zero and the matrix M can not be inverted. Therefore, a standing assumption is that (6.10) is invertible when $s = 0$.

Provided the general system from Equation (6.1), the control law in Equation (6.5) and the observer in Equation (6.3), the LTI system and observer are stabilized. The designed observer is used in the full state feedback law. The steady state response of the system converges to the reference provided the designed feed forward.

Equation (6.11) is a transformed system from the one presented in Equation (6.6). The transformation provides an opportunity to prove stability of the system, since all poles of the system depends on $(A - BK)$ and $(A - LC)$ matrices. Since both matrices are Hurwitz, the overall system is Hurwitz.

$$\begin{bmatrix} \dot{x} \\ \dot{\hat{x}} - \hat{x} \end{bmatrix} = \begin{bmatrix} A - BK & BK \\ O_{m \times m} & A - LC \end{bmatrix} \begin{bmatrix} x \\ x - \hat{x} \end{bmatrix} + \begin{bmatrix} BM \\ O_{m \times o} \end{bmatrix} r \quad (6.11)$$

6.1.2 Exogenous System

Consider a linear time invariant system described by the following equations:

$$\begin{aligned} \dot{x} &= Ax + Bu + B_d d \\ y &= Cx + D_{cd} d \end{aligned} \quad (6.12)$$

The system originates from the general system provided in Equation (6.1), where $d \in \mathbb{R}^q$ is the unmeasured disturbances acting on the system respectively. Furthermore, $B_d \in \mathbb{R}^{m \times q}$ is the input matrix for disturbances and $D_{cd} \in \mathbb{R}^{o \times q}$ is the disturbance feed through matrix.

The disturbance is driven by a state space system provided in Equation (6.13). This type of disturbance is said to be generated by an exogenous system.

$$\begin{aligned} \dot{v}^* &= S^* v^* \\ d &= H^* v^* \end{aligned} \quad (6.13)$$

$v^* \in \mathbb{R}^r$ and $d \in \mathbb{R}^q$ are the internal uncontrollable state and the disturbance respectively. The matrices $S^* \in \mathbb{R}^{r \times r}$ and $H^* \in \mathbb{R}^{q \times r}$ are the transition matrix and the matrix which combines the internal states to generate a set of disturbance signals. The unknown in the exogenous system is the initial condition v_0^* which is used as an impulse to the system.

Depending on the complexity of the disturbance the matrix S^* can consist of various different functions or a combination of the functions, such as a step function, ramp function or a sinusoidal. A small collection of signals are shown in Table 6.1 and it has to be noted that in general, the real part of the eigenvalues of the exogenous system lie around zero.

Signal form	Characteristic Polynomial	Disturbance transition matrix	Disturbance measurement matrix	Eigenvalues
	$\det(sI - S^*)$	S^*	H^*	λ_i
step	s	0	1	$\lambda_i \in \{0\}$
ramp	s^2	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{0, 0\}$
parabola	s^3	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{0, 0, 0\}$
sine	$s^2 + \omega_0^2$	$\begin{bmatrix} 0 & \omega_0 \\ -\omega_0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{0 \pm \omega_0 i\}$
sine with non-zero mean	$s(s^2 + \omega_0^2)$	$\begin{bmatrix} 0 & \omega_0 & 0 \\ -\omega_0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}^T$	$\lambda_i \in \{0 \pm \omega_0 i, 0\}$
exponential increase	$s - \alpha$ for $\alpha > 0$	α	1	$\lambda_i \in \{\alpha\}$

Table 6.1: Various signals with their characteristic polynomial, exogenous system matrix representation and eigenvalues.

Provided the general system from Equation (6.6), which includes the observer, the feedback law and the feedforward law from Equations (6.3), (6.5) and (6.9) respectively the model can be extended with the disturbance model provided Equation (6.13), which results in the augmented system:

$$\begin{aligned}
\begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{x}} \\ \dot{v}^* \end{bmatrix} &= \begin{bmatrix} A & -BK & B_d H^* \\ LC & A - BK - LC & O_{m \times r} \\ O_{r \times m} & O_{r \times m} & S^* \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \\ v^* \end{bmatrix} + \begin{bmatrix} B \\ B \\ O_{r \times n} \end{bmatrix} Mr \\
y &= [C \quad O_{p \times m} \quad O_{p \times r}] \begin{bmatrix} x \\ \hat{x} \\ v^* \end{bmatrix} \\
y_c &= [C_c \quad O_{p \times m} \quad O_{p \times r}] \begin{bmatrix} x \\ \hat{x} \\ v^* \end{bmatrix}
\end{aligned} \tag{6.14}$$

6.1.3 Disturbance rejection

Provided system (6.14) a disturbance rejection control law can be designed to reject the disturbance applied to the system. The proposed driver model is provided in Equation (6.15). The proposed driver model can be seen as a disturbance observer, which takes the output and reference into account.

$$\dot{v} = Sv + B_\epsilon y - B_\sigma Mr \tag{6.15}$$

In the driver model, $v \in \mathbb{R}^s$, $y \in \mathbb{R}^o$ and $r \in \mathbb{R}^p$ are the internal model state, the measurements of the plant and the reference, respectively. The matrix $S \in \mathbb{R}^{s \times s}$ can be chosen freely. Some examples are presented in Table 6.1 and ideally, the disturbance model S^* , which can be modeled as an exogenous system described in Section 6.1.2, should be included in the driver model S . Since the Internal Model Principle copes with disturbances, S can also be used to remove any model uncertainties. Therefore, the disturbance S^* is not necessarily equal to S , since S is an estimation of the disturbance. Furthermore, the error of the system is not used to drive the disturbance rejection. Instead, the measurements and the reference are fed separately into the driver model by $B_\epsilon \in \mathbb{R}^{s \times o}$ and $B_\sigma \in \mathbb{R}^{s \times n}$. B_ϵ can be chosen up to some freedom, whereas B_σ is designed during the stability analysis. The augmented system can be described by:

$$\begin{aligned}
\dot{x}_{aug} &= A_{aug} x_{aug} + B_{aug} u + B_{r,aug} r \\
y_{aug} &= C_{aug} x_{aug}
\end{aligned} \tag{6.16}$$

where system (6.1) is combined with the driven system (6.15) where:

$$\begin{aligned} x_{aug} = \begin{bmatrix} x \\ v \end{bmatrix} \quad A_{aug} = \begin{bmatrix} A & 0 \\ B_\epsilon C_c & S \end{bmatrix} \quad B_{aug} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad B_{r,aug} = \begin{bmatrix} 0 \\ -B_\sigma M \end{bmatrix} \\ y_{aug} = \begin{bmatrix} y \\ v \end{bmatrix} \quad C_{aug} = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \end{aligned} \quad (6.17)$$

In order to stabilize the augmented system, a control law is proposed, described by:

$$u = -K_{aug}x_{aug} + Mr \quad (6.18)$$

where $K_{aug} = [K_x \quad K_v] \in \mathbb{R}^{n \times m+s}$ is the stabilizing feedback law. The control input can be directly implemented in the system (6.1). By rewriting the control input as a function of the reference $Mr = u + K_x x + K_v v$, the control input can be implemented via the reference in the driver system (6.15). Applying the control law to the plant model and the driver model yields:

$$\begin{aligned} \dot{x} &= (A - BK_x)x + B(Mr - K_v v) \\ \dot{v} &= (S - B_\sigma K_v)v + B_\epsilon Cx - B_\sigma(u + K_x x) \end{aligned} \quad (6.19)$$

An error model between the plant state and the driver state is proposed in the form of $(\Sigma x - v)$. The matrix Σ is later specified and provides the ability to generate an error between the driver model states v and the plant states x . The derivative of the error can be described as:

$$\begin{aligned} \Sigma \dot{x} - \dot{v} &= \Sigma(A - BK_x)x + \Sigma B(Mr - K_v v) \\ &\quad - (S - B_\sigma K_v)v - B_\epsilon Cx + B_\sigma(u + K_x x) \end{aligned} \quad (6.20)$$

By utilizing $Mr = u + K_x x + K_v v$, the error dynamics can be described by:

$$\begin{aligned} \Sigma \dot{x} - \dot{v} &= (S - B_\sigma K_v)(\Sigma x - v) \\ &\quad + (\Sigma(A - BK_x) - (S - B_\sigma K_v)\Sigma - B_\epsilon C)x \\ &\quad + (\Sigma B + B_\sigma)(K_x x + u) \end{aligned} \quad (6.21)$$

Stability of the error dynamics can be made dependable on the dynamics of $(S - B_\sigma K_v)$ if the following conditions holds:

$$\Sigma(A - BK_x) - (S - B_\sigma K_v)\Sigma = B_\epsilon C \quad (6.22)$$

$$B_\sigma = -\Sigma B \quad (6.23)$$

where the error dynamics becomes:

$$\Sigma \dot{x} - \dot{v} = (S - B_\sigma K_v)(\Sigma x - v) \quad (6.24)$$

which can be stabilized by finding a control gain K_v which makes the matrix $S - B_\sigma K_v$ Hurwitz. Given relations (6.22), (6.23) and provided that $K = K_x + K_v \Sigma$, Equation (6.22) can be simplified to

$$\Sigma(A - BK) - S\Sigma = B_\epsilon C \quad (6.25)$$

where $(A - BK)$ is a Hurwitz matrix, where the control gain K is designed in Subsection 6.1.1. Designing a stabilizing control law (6.18) provided the plant model (6.12) and the driver model (6.15) holds the following steps:

1. Compute a stabilizing control gain K , which makes $(A - BK)$ Hurwitz.
2. Solve $\Sigma(A - BK) - S\Sigma = B_\epsilon C$ for $\Sigma \neq 0$.
3. Obtain $B_\sigma = -\Sigma B$.
4. Compute a stabilizing control gain K_v , which makes $(S - B_\sigma K_v)$ Hurwitz.
5. Compute the control gain for the plant $K_x = K - K_v \Sigma$.
6. Compute a feed forward gain M which lets the steady state converge to the reference over time.

After designing a stabilizing control law the augmented system including the plant model, the driver model and the stabilizing control law can be written as:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} A - BK_x & -BK_v \\ B_e C & S \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} B \\ -B_\sigma \end{bmatrix} Mr(t) \quad (6.26)$$

Provided a state transformation to $[x(t) \quad \Sigma x(t) - v(t)]^T$ and the functions presented in the design process of the controller the augmented system transform to the system presented in Equation (6.27). Stability can be proven, since the eigenvalues of the system only depends on $(A - BK)(S - B_\sigma K_v)$ which both have eigenvalues in the left plane, since both are designed to be Hurwitz.

$$\begin{bmatrix} \dot{x}(t) \\ \Sigma \dot{x}(t) - \dot{v}(t) \end{bmatrix} = \begin{bmatrix} A - BK & BK_v \\ 0 & S - B_\sigma K_v \end{bmatrix} \begin{bmatrix} x(t) \\ \Sigma x(t) - v(t) \end{bmatrix} + \begin{bmatrix} BM \\ 0 \end{bmatrix} r(t) \quad (6.27)$$

6.2 Discrete time Internal Model Principle

The discrete time case IMP, holds the same procedure as the continuous time case, except that all matrices should be designed to be Schur instead of Hurwitz. Before all the steps are explained, the system should be discretized, which is done in a Zero-order-hold (ZOH) fashion for an m by m state space model $(A, B, C, 0)$ with a sampling frequency f_s for which h_s is the sampling time. The discretized matrices are obtained by applying the ZOH approach and applying a Taylor expansion which results in the discretized system.

$$\begin{aligned} A &= e^{Ah_s} = \sum_{i=1}^m A^{i-1} h_s^{i-1} \\ B &= \int_0^{h_s} e^{As} B ds = \sum_{i=1}^m \frac{1}{i!} A^{i-1} B h_s^i \end{aligned} \quad (6.28)$$

Only the system matrices C and D remain unchanged. The result of the discretized general system is described by:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \\ y_{c_k} &= C_c x_k \end{aligned} \quad (6.29)$$

The following subsections follow the same design procedures as the continuous time, except for some slight nuances.

6.2.1 Controller, observer and feedforward design

Provided the general state space system (6.1), which is discretized following the ZOH approach in Equation (6.28) and resulted in the state space system presented in Equation (6.29), a controller, observer and feedforward needs to be designed. Furthermore, it is assumed that (A, B) is controllable and (A, C) observable.

A stabilizing control law can be designed based on pole placement. The control law is based on Equation (6.30), which makes the matrix $(A - BK)$ Schur, which means all eigenvalues lie inside the unit disc.

$$u_k = -Kx_k \quad (6.30)$$

Since not all states are measured, a full state observer is designed to estimate the remainder states. The estimated states are used in the control law. The observer is provided in Equation (6.31).

$$\begin{aligned} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + L(y_k - \hat{y}_k) \\ \hat{y}_k &= C\hat{x}_k \end{aligned} \quad (6.31)$$

The error $(x_k - \hat{x}_k)$ converges over time to zero if the poles of $(A - LC)$ are placed inside the unit disc and therefore make the matrix Schur. This can be achieved by designing L using for instance pole

placement. The error dynamics of the observer are shown in Equation (6.32), which is a combination of the general discretized system (6.29) and the proposed observer (6.31).

$$x_{k+1} - \hat{x}_{k+1} = (A - LC)(x_k - \hat{x}_k) \quad (6.32)$$

In order to design a feedforward which converges the steady state error to the reference, $\lim_{t \rightarrow \infty} y_{ck} = r_k$, an augmented system is constructed based on the general discretized plant model (6.29), and the observer (6.31). The control law (6.30) utilizes the estimated state obtained from the observer and the reference with an additional feedforward term. The control law is described by:

$$u_k = -K\hat{x}_k + Mr_k \quad (6.33)$$

The steady state response can be obtained if the new state is equal to previous state,

$([x \ \hat{x}]_{k+1} = [x \ \hat{x}]_k = [x \ \hat{x}]_{ss})$. The steady state is measured via the C_c matrix, therefore the steady state response can be described by:

$$y_{ck} = [C_c \ O_{o \times m}] \begin{bmatrix} x \\ \hat{x} \end{bmatrix}_{ss} = [C_c \ O_{o \times m}] \left(I - \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \right)^{-1} \begin{bmatrix} B \\ B \end{bmatrix} Mr_k \quad (6.34)$$

In order to let the steady state response converge to the reference over time, the steady state response should become equal to $y_{ck} = r_k$ which is achieved if M is designed properly. M can be obtained by:

$$M = \left([C_c \ O_{o \times m}] \left(I - \begin{bmatrix} A & -BK \\ LC & A - BK - LC \end{bmatrix} \right)^{-1} \begin{bmatrix} B \\ B \end{bmatrix} \right)^{-1} \quad (6.35)$$

The inversion of M only holds, when the matrix is full rank. The matrix is full rank if the system (C_c, A, B) has no invariant zero at $z = 1$. In order to determine if the system (C_c, A, B) contains an invariant zero, the Rosenbrock's system matrix can be used. The Rosenbrock's system matrix is described by:

$$P(z) = \begin{bmatrix} zI - A & B \\ -C & D \end{bmatrix} \quad (6.36)$$

If the matrix $P(z)$ loses rank if $z = 1$, then the system contains an invariant zero and the matrix M can not be obtained.

The augmented discrete system, which includes the plant, observer, control gain and feedforward can be written as

$$\begin{aligned} \begin{bmatrix} x \\ x - \hat{x} \end{bmatrix}_{k+1} &= \begin{bmatrix} A - BK & BK \\ O_{m \times m} & A - LC \end{bmatrix} \begin{bmatrix} x \\ x - \hat{x} \end{bmatrix}_k + \begin{bmatrix} B \\ O_{m \times n} \end{bmatrix} Mr_k \\ y_k &= [C \ O_{o \times m}] \begin{bmatrix} x \\ x - \hat{x} \end{bmatrix}_k \\ y_{ck} &= [C_c \ O_{p \times m}] \begin{bmatrix} x \\ x - \hat{x} \end{bmatrix}_k \end{aligned} \quad (6.37)$$

6.2.2 Exogenous System

Consider a LTI discretized system described by the following equations:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + B_d d_k \\ y_k &= Cx_k + D_d d_k \end{aligned} \quad (6.38)$$

The system originates from the general discretized system provided in Equation (6.29) and is extended with a disturbance, $d \in \mathbb{R}^q$ is the unmeasured disturbances applied to the system respectively. Furthermore, $B_d \in \mathbb{R}^{m \times q}$ is the input matrix for disturbances and $D_{cd} \in \mathbb{R}^{o \times q}$ is the disturbance feed through matrix.

The disturbance is driven by a discrete exogenous state space system described by:

$$\begin{aligned} v_{k+1}^* &= S^* v_k^* \\ d_k &= H^* v_k^* \end{aligned} \quad (6.39)$$

where $v^* \in \mathbb{R}^r$ and $d \in \mathbb{R}^q$ are the internal uncontrollable state and the disturbance respectively. The matrices $S^* \in \mathbb{R}^{r \times r}$ and $H^* \in \mathbb{R}^{q \times r}$ are the transition matrix and the matrix which combines the internal states to generate a set of disturbance signals. The unknown in the exogenous system is the initial condition v_0^* which is used as an impulse to the exogenous system.

Depending on the desired complexity of the disturbance, matrix S^* can be filled with various different functions or a combination of the functions, such as a step, ramp or sinusoidal function. A small collection of signals are shown in Table 6.2 and it has to be noted that in general, the eigenvalues of the disturbance transition matrix S^* lie some where on the edge of the unit disc, except for the exponential increase.

Signal form	Characteristic Polynomial $\det(zI - S^*)$	Disturbance transition matrix S^*	Disturbance measurement matrix H	Eigenvalues λ_i
step	$z - 1$	$\begin{bmatrix} 1 \end{bmatrix}$	1	$\lambda_i \in \{1\}$
ramp	$(z - 1)^2$	$\begin{bmatrix} 1 & h_s \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{1, 1\}$
parabola	$(z - 1)^3$	$\begin{bmatrix} 1 & h_s & h_s^2 \\ 0 & 1 & h_s \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{1, 1, 1\}$
sine	$z^2 - 2 \cos(\omega_0 h_s) z + 1$	$\begin{bmatrix} 0 & 1 \\ -1 & 2 \cos(\omega_0 h_s) \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T$	$\lambda_i \in \{a \pm bi\}$
sine with non-zero mean	$(z - 1)(z^2 - 2 \cos(\omega_0 h_s) z + 1)$	$\begin{bmatrix} 1 - h_s^2 w_0^2 & h_s \omega_0 & 0 \\ -h_s \omega_0 & 1 - h_s^2 w_0^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}^T$	$\lambda_i \in \{a \pm bi, 1\}$
exponential increase	$z - \exp^{\alpha h_s}$ for $\alpha > 0$	$[\alpha]$	1	$\lambda_i \in \{\alpha\}$

Table 6.2: Various signals with their characteristic polynomial, exogenous system matrix representation and eigenvalues represented in discrete time.

The general discretized state space system including observer, stabilizing control law and feedforward (6.37) and the exogenous disturbance system (6.39) can be combined to one augmented system. The augmented system is described as:

$$\begin{aligned}
 \begin{bmatrix} x \\ x - \hat{x} \\ v^* \end{bmatrix}_{k+1} &= \begin{bmatrix} A - BK & BK & B_d H^* \\ O_{m \times m} & A - LC & B_d H^* \\ O_{r \times m} & O_{r \times m} & S^* \end{bmatrix} \begin{bmatrix} x \\ x - \hat{x} \\ v^* \end{bmatrix}_k + \begin{bmatrix} B \\ O_{m \times n} \\ O_{r \times n} \end{bmatrix} M r_k \\
 y_k &= [C \quad O_{o \times m} \quad O_{o \times r}] \begin{bmatrix} x \\ x - \hat{x} \\ v^* \end{bmatrix}_k \quad y_{c k} = [C_c \quad O_{p \times m} \quad O_{p \times r}] \begin{bmatrix} x \\ x - \hat{x} \\ v^* \end{bmatrix}_k
 \end{aligned} \tag{6.40}$$

Noteworthy, the state transition matrix is upper triangular, which concludes that stability is determined by the poles on the diagonal of the matrix. Since $(A - BK)$ and $(A - LC)$ are Schur designed earlier in Section 6.2.1, the stability only depends on the disturbance transition matrix S^* . In most cases, the poles of the matrix lie on the edge of the unit disc as can be seen in Table 6.2 and have a multiplicity larger than 1. Due to the multiplicity larger than 1 and in case of the exponential increase where the pole can be even outside the unit disc, stability is compromised, due to the disturbance applied to the system utilizing the exogenous system.

6.2.3 Disturbance Rejection

Designing a stabilizing control law with disturbance rejection for the discrete time case, holds the same procedure as the continuous time case. Therefore, only a recap will be given, except at places where the discrete time case differs from the continuous time case.

Given system (6.40), a disturbance rejection control law can be designed to reject the disturbance applied to the system. The proposed driver model is provided in Equation (6.41). The proposed driver model can be seen as a disturbance observer, which takes the feedback law and reference into account and tries to recreate the disturbance in its states.

$$v_{k+1} = Sv_k + B_\epsilon y_k - B_\sigma Mr_k \quad (6.41)$$

In the driver model, $v \in \mathbb{R}^s$, $y \in \mathbb{R}^o$ and $r \in \mathbb{R}^p$ are the internal model state, the measurements of the plant and the reference, respectively. The matrix $S \in \mathbb{R}^{s \times s}$ can be designed manually. Some examples are presented in Table 6.2 and ideally at least $S^* \in S$. Since the IMP copes with disturbances, S can also be used to remove any model uncertainties. Therefore, the disturbance S^* is not necessarily equal to S , since S is an estimation of the disturbance. Furthermore, the error of the system is not used to drive the disturbance rejection. Instead, the measurements and the reference are fed separately into the driver model by $B_\epsilon \in \mathbb{R}^{s \times o}$ and $B_\sigma \in \mathbb{R}^{s \times n}$, such that frequencies applied to the system via the reference, will not be rejected. B_ϵ can be chosen up to some freedom, whereas B_σ is designed during the stability analysis.

$$\begin{aligned} x_{aug,k+1} &= A_{aug}x_{aug,k} + B_{aug}u_k + B_{r,aug}r_k \\ y_{aug,k} &= C_{aug}x_{aug,k} \end{aligned} \quad (6.42)$$

where system (6.29) is combined with the driven system (6.41) where:

$$\begin{aligned} x_{aug,k} &= \begin{bmatrix} x_k \\ v_k \end{bmatrix} \quad A_{aug} = \begin{bmatrix} A & 0 \\ B_\epsilon C_c & S \end{bmatrix} \quad B_{aug} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad B_{r,aug} = \begin{bmatrix} 0 \\ -B_\sigma M \end{bmatrix} \\ y_{aug} &= \begin{bmatrix} y \\ v \end{bmatrix} \quad C_{aug} = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \end{aligned} \quad (6.43)$$

A control law for the augmented system is proposed in the form of:

$$u_k = -K_{aug}x_{aug,k} + Mr_k \quad (6.44)$$

where $K_{aug} = [K_x \quad K_v] \in \mathbb{R}^{n \times m+s}$ is the stabilizing feedback law. By implementing the control law in the general system (6.29) and rewriting the control law to the reference $Mr_k = u_k + K_x x_k + K_v v_k$, such that the reference can be implemented in the driver model (6.41) results in:

$$\begin{aligned} x_{k+1} &= (A - BK_x)x_k + B(-K_v v_k + Mr_k) \\ v_{k+1} &= (S - B_\sigma K_v)v_k + B_\epsilon C x_k - B_\sigma (u_k + K_x x_k) \end{aligned} \quad (6.45)$$

An error model can be reconstructed in the form of $\Sigma x_k - v_k$. The matrix Σ is specified later on. The matrix Σ relates the states of the driver model v_k to the system states x_k in the form of the proposed error. By taking into account $Mr_k = u_k + K_x x_k + K_v v_k$, the error dynamics between the system states x_k and the driver model states v_k can be described by:

$$\begin{aligned} \Sigma x_{k+1} - v_{k+1} &= (S - B_\sigma K_v)(\Sigma x_k - v_k) \\ &\quad + (\Sigma(A - BK_x) - (S - B_\sigma K_v)\Sigma - B_\epsilon C)x_k \\ &\quad + (\Sigma B + B_\sigma)(u_k + K_x x_k) \end{aligned} \quad (6.46)$$

The stability of the error dynamics can be made dependable on the dynamics of $(S - B_\sigma K_v)$ if the following conditions holds:

$$\begin{aligned} \Sigma(A - BK_x) - (S - B_\sigma K_v)\Sigma &= B_\epsilon C \\ B_\sigma &= -\Sigma B \end{aligned} \quad (6.47)$$

where the error dynamics become:

$$\Sigma x_{k+1} - v_{k+1} = (S - B_\sigma K_v)(\Sigma x_k - v_k) \quad (6.48)$$

The augmented system including the error dynamics $\Sigma x_k - v_k$ can be described as:

$$\begin{bmatrix} x_{k+1} \\ \Sigma x_{k+1} - v_{k+1} \end{bmatrix} = \begin{bmatrix} A - B(K_x + K_v \Sigma) & BK_v \\ 0 & S - B_\sigma K_v \end{bmatrix} \begin{bmatrix} x_k \\ \Sigma x_k - v_k \end{bmatrix} + \begin{bmatrix} BM \\ 0 \end{bmatrix} r_k \quad (6.49)$$

Due to the triangular structure of the augmented system, it can be concluded that the eigenvalues of the system only depend on $A - B(K_x + K_v\Sigma)$ and $S - B_\sigma K_v$. This concludes, provided the control gain (6.30), is related in the form of $K = K_x + K_v\Sigma$. Taking the control gain relation into account, the relations proposed in Equation (6.47) can be simplified to:

$$\begin{aligned}\Sigma(A - BK) - S\Sigma &= B_\epsilon C \\ B_\sigma &= -\Sigma B\end{aligned}\tag{6.50}$$

Therefore, designing a stabilizing control law (6.44) provided the plant model (6.38) including IMP provided in the driver model (6.41) holds the following steps:

1. Compute a stabilizing control gain K , which makes $(A - BK)$ Schur.
2. Solve $\Sigma(A - BK) - S\Sigma = B_\epsilon C$ for $\Sigma \neq 0$.
3. Obtain $B_\sigma = -\Sigma B$.
4. Compute a stabilizing control gain K_v , which makes $(S - B_\sigma K_v)$ Schur.
5. Compute the control gain for the plant $K_x = K - K_v\Sigma$.
6. Compute a feed forward gain M which lets the steady state converge to the reference over time.

After designing a stabilizing control law the augmented system including the plant model, the driver model and the stabilizing control law can be written as:

$$\begin{bmatrix} x \\ v \end{bmatrix}_{k+1} = \begin{bmatrix} A - BK_x & -BK_v \\ B_\epsilon C & S \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix}_k + \begin{bmatrix} B \\ -B_\sigma \end{bmatrix} Mr_k\tag{6.51}$$

Provided a state transformation to $[x \ \Sigma x - v]_k^T$ and the functions presented in the design process of the controller the augmented system transform to the system presented in Equation (6.52). Stability can be proven, since the determinant of the system only depends on $(A - BK)$ and $(S - B_\sigma K_v)$ which both have eigenvalues designed inside or on the edge of the unit disc, since both are designed to be Schur.

$$\begin{bmatrix} x \\ \Sigma x - v \end{bmatrix}_{k+1} = \begin{bmatrix} A - BK & BK_v \\ O_{m \times m} & S - B_\sigma K_v \end{bmatrix} \begin{bmatrix} x \\ \Sigma x - v \end{bmatrix}_k + \begin{bmatrix} B \\ O_{m \times n} \end{bmatrix} Mr_k\tag{6.52}$$

Furthermore, by including the combining the disturbed plant model, the disturbance, the observer, the control law, and the driver model, an augmented system can be realized, presented in Equation (6.53).

$$\begin{bmatrix} x \\ x - \hat{x} \\ v^* \\ \Sigma x - v \end{bmatrix}_{k+1} = \begin{bmatrix} A - BK & BK_x & B_d H^* & BK_v \\ O_{m \times m} & A - LC & B_d H^* & O_{m \times m} \\ O_{r \times m} & O_{r \times m} & S^* & O_{r \times s} \\ O_{s \times m} & -B_\sigma K_x & \Sigma B_d H^* & S - B_\sigma K_v \end{bmatrix} \begin{bmatrix} x \\ x - \hat{x} \\ v^* \\ \Sigma x - v \end{bmatrix}_k + \begin{bmatrix} B \\ O_{m \times n} \\ O_{r \times n} \\ O_{s \times n} \end{bmatrix} Mr_k\tag{6.53}$$

$$y_k = [C \ O_{o \times m} \ O_{o \times r} \ O_{o \times s}] \begin{bmatrix} x \\ x - \hat{x} \\ v^* \\ \Sigma x - v \end{bmatrix}_k \quad y_{ck} = [C_c \ O_{o \times m} \ O_{o \times r} \ O_{o \times s}] \begin{bmatrix} x \\ x - \hat{x} \\ v^* \\ \Sigma x - v \end{bmatrix}_k$$

From the augmented system can be concluded:

1. All states are effected by the disturbance.
2. The eigenvalues are depending on the designed control gain $(A - BK)$, the designed observer $(A - LC)$, the applied disturbance (S^*) and the designed controller for the driver model $(S - B_\sigma K_v)$.
3. The state observer model, the disturbance and the disturbance observer do not depend on the reference.

6.3 Repetitive Control

A special case of IMP is Repetitive Control (RC). RC is a method which is able to learn and counteract periodic disturbance signals, such as tracking a reference or rejecting a periodic disturbance. The RC filter is shown in Figure 6.2 where the z^{-1} is a delay operator of one measurement. The value N is the number of delays incorporated in the filter and typically taken as the period of the signal which needs to be learned.

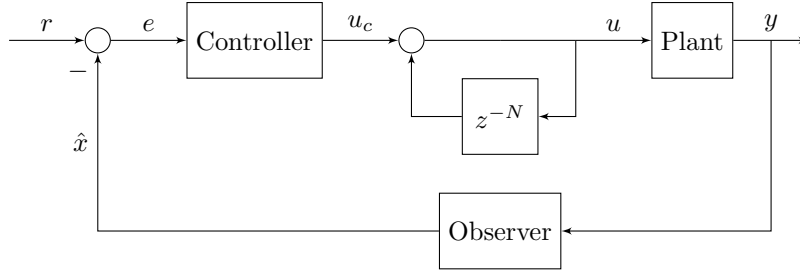


Figure 6.2: Block diagram of a Repetitive Control Filter.

The RC filter can be written as either a transfer function or an state space model. The transfer function is presented in Equation (6.54). The RC filter can be seen as a memory of the system, where it stores the N last inputs to the system. Every cycle, a new control law is computed based on the controller design and the results of previous control inputs, especially the control input of N times back.

$$u = \frac{1}{1 - z^{-N}} u_c \quad (6.54)$$

The state space format is written in Equation (6.55), where x , u_c and u are the internal states, the control input provided by the controller and the control input to the plant respectively. The matrix $A \in \mathbb{R}^{N \times N}$ holds the memory of previous inputs. The B-matrix $B \in \mathbb{R}^{N \times 1}$ is the input matrix, which stores control inputs from the controller in the memory of the filter and keeps them N cycles. The measurement matrix $C \in \mathbb{R}^{1 \times N}$ takes the control input from the controller and the input generated N cycles ago and combines them to the new plant input.

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ \dots \\ x_{k+1-N} \end{bmatrix} &= A \begin{bmatrix} x_k \\ \dots \\ x_{k-N} \end{bmatrix} + B u_c \\ u &= C \begin{bmatrix} x_k \\ \dots \\ x_{k-N} \end{bmatrix} \\ A &= \begin{bmatrix} O_{N-1 \times 1} & I_{N-1 \times N-1} \\ 1 & O_{1 \times N-1} \end{bmatrix} \quad B = \begin{bmatrix} O_{N-1 \times 1} \\ 1 \end{bmatrix} \\ C &= [-1 \quad O_{1 \times N-1}] \end{aligned} \quad (6.55)$$

Since the A matrix of the RC filter is filled mostly with either ones, zeros or minus ones, the eigenvalues lie on the edge of the unit disc, as can be seen in Figure 6.3. The figure illustrates RC filters with length $N \in \{1, 4, 9, 19\}$, where all poles lie on the edge of the unit disc.

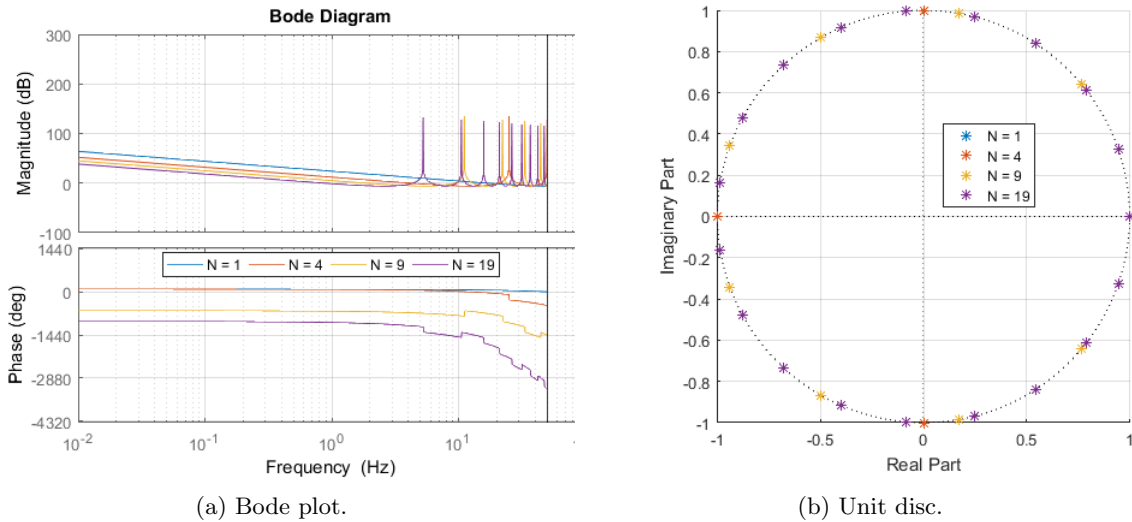


Figure 6.3: Examples of RC filters. On the left are the bode plots shown for RC filters sizes 1, 4, 9 and 19 respectively. On the right, the poles of the filters are shown.

The bode plot in Figure 6.3a only shows half the frequency range of the bode plot, which corresponds to half the unit disc on the right side of Figure 6.3b. The main reason for this is, the bode plot is drawn from low frequencies, in Figure 6.3 from 0.01Hz up to the Nyquist frequency, which is 50Hz for a sampling rate of 100Hz. This corresponds to half the unit disc.

As an example, lets take a filter with $N = 4$, shown in Figure 6.4, which has poles at $\lambda_i \in \{1, i, -1, -i\}$. The poles are distributed evenly around the unit disc, which corresponds to $\{0\pi, \frac{1}{2}\pi, \pi, 1\frac{1}{2}\pi\}$. Mapping the locations of the poles to the frequency band, the poles are placed at $\{0, 25, 50, 75\}$ Hz. Provided the bode plot in Figure 6.4a, changes in the transfer function are indeed at the location of those frequencies. Only the 75Hz is not shown, since it lies past the Nyquist frequency of 50Hz.

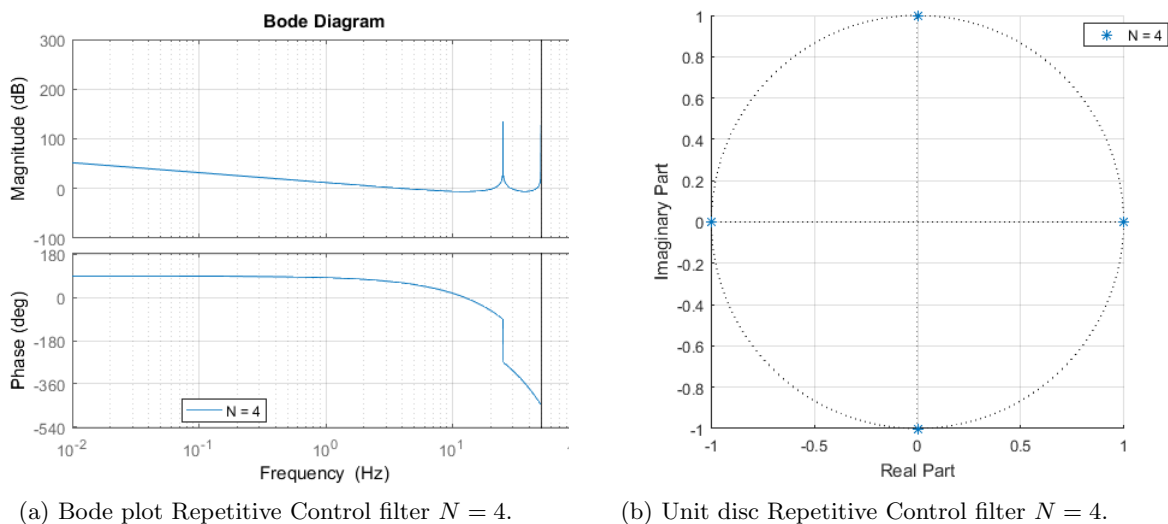


Figure 6.4: Example of RC filters with size 4. On the left is the bode plot and on the right, the poles are shown.

Where the poles are located around the unit disc and how they correspond to the transfer function is important for designing a filter which improves specific frequencies. A signal with a specific frequency can be learned by multiple filters. As an example, lets assume a frequency of $f_f = 50$ Hz needs to be

learned. This means the filter should contain a pole at $\lambda = -1 + 0i$. The lowest filter which can be used is $N = 2$, but every multiple of $N = 2$ has a pole at $\lambda = -1 + 0i$. Furthermore, the lowest filter value N , which should filter the frequency f_f for a given sampling frequency f_s can be calculated as

$$N = \frac{f_s}{f_f} \quad (6.56)$$

If the RC filter is supposed to filter multiple frequencies, the greatest common divider is leading the filter design. Which means, if two different frequencies need to be filtered, for instance 15Hz and 10Hz, N is calculated following Equation (6.56) for $f_f = 5$, since 5 is the greatest common divider of both frequencies.

Furthermore, an RC filter is in some cases designed with various other filters, such as low pass filters, see [83]. Where a $Q(s)$ and a $L(s)$ filters are added to the RC, such that stability is guaranteed. Not only additional filters can be applied, but higher order RC filters can be used as well. In [84] a higher order RC design is proposed. For the scope of this thesis, those topics are not explained.

Provided a general discretized system and a state space RC filter of order N , which are both shown in Equation (6.57), where the plant is defined by (A_p, B_p, C_p) and the RC by (A_{rc}, B_{rc}, C_{rc}) respectively.

$$\begin{aligned} x_{k+1} &= A_p x_k + B_p u_k & \tilde{x}_{k+1} &= A_{rc} \tilde{x}_k + B_{rc} \tilde{u}_k \\ y_k &= C_p x_k & \tilde{y}_k &= C_{rc} \tilde{x}_k \end{aligned} \quad (6.57)$$

Both system can be augmented to one system by taking $u_k = \tilde{y}_k$ and combining the states into one system.

$$\begin{aligned} \begin{bmatrix} x \\ \tilde{x} \end{bmatrix}_{k+1} &= \begin{bmatrix} A_p & B_p C_{rc} \\ O & A_{rc} \end{bmatrix} \begin{bmatrix} x \\ \tilde{x} \end{bmatrix}_k + \begin{bmatrix} O \\ B_{rc} \end{bmatrix} \tilde{u}_k \\ y_k &= [C_p \quad O] \begin{bmatrix} x \\ \tilde{x} \end{bmatrix}_k \end{aligned} \quad (6.58)$$

From the augmented system, a controller, observer and if necessary a feedforward can be designed by utilizing pole placement. This procedure for discrete time is explained in Section 6.2.1. The proposed controller and observer design make sure the matrices in Equation (6.59) are both Schur. Furthermore, it is assumed the augmented system is controllable and observable.

$$\left(\begin{bmatrix} A_p & B_p C_{rc} \\ O & A_{rc} \end{bmatrix} - L [C_p \quad O] \right) \quad \left(\begin{bmatrix} A_p & B_p C_{rc} \\ O & A_{rc} \end{bmatrix} - \begin{bmatrix} O \\ B_{rc} \end{bmatrix} K \right) \quad (6.59)$$

Chapter 7

Experiments and results

In this chapter, the results of the simulations and experiments are discussed. In Section 7.1 the general simulator parameters are discussed, such as drone weight and inertias, wind models, evaluated controllers and a measure for evaluating reference tracking. The sections following Section 7.1 describes the results of hovering performance under various wind conditions. In Section 7.2, windless conditions will be considered. In Section 7.3, shear wind will be discussed. In Section 7.4, a wind gust will be considered. In Section 7.5, Dryden Turbulent wind will be considered. In Section 7.6, however, reference tracking of a dynamical reference in windless conditions will be discussed.

7.1 Simulation

In this section, the parameters for the simulator are discussed. In Table 7.1 a list of parameters used during simulation is shown. These parameters are kept constant between each experiment. The drag coefficients are set at 1.05 in all directions, which is equal to the dynamic properties of a box. Furthermore, the aerodynamic surface is calculated as the surface of a box with the same size as the quad-copter. The measures of the quad-copter are 461.44mm in width by 480.64mm in length and 93.85mm in height.

Parameter	Value	Units	Description
m	1.4	kg	Mass of the quad-copter
f_s	100	Hz	Sampling frequency
ρ_{AIR}	1.2041	kgm^{-3}	Density of air at 20 degrees
G	$[0 \ 0 \ 9.81]^T$	ms^{-2}	Gravity vector
J	$\begin{bmatrix} 0.0304 & 0 & 0 \\ 0 & 0.0309 & 0 \\ 0 & 0 & 0.0599 \end{bmatrix}$	kgm^2	Moment of Inertia
A	$\begin{bmatrix} 0.0433 & 0 & 0 \\ 0 & 0.0451 & 0 \\ 0 & 0 & 0.2218 \end{bmatrix}$	m^2	Aerodynamic surface
C_D	$\begin{bmatrix} 1.05 & 0 & 0 \\ 0 & 1.05 & 0 \\ 0 & 0 & 1.05 \end{bmatrix}$	–	Drag coefficient

Table 7.1: List of parameters used in simulation.

The used controllers are a simple feedback-feedforward controller with full state observer as described in Section 6.2. The poles of the system are placed at the edge of the unit disc. The placed poles for the feedback control gains λ_{K_i} and the observer gains λ_{L_i} are described by:

$$\begin{aligned}
 \lambda_{Kx} &= [0.90 \ 0.93 \ 0.96 \ 0.99] & \lambda_{Lx} &= [0.90 \ 0.93 \ 0.96 \ 0.99] \\
 \lambda_{Ky} &= [0.90 \ 0.93 \ 0.96 \ 0.99] & \lambda_{Ly} &= [0.90 \ 0.93 \ 0.96 \ 0.99] \\
 \lambda_{Kz} &= [0.90 \ 0.99] & \lambda_{Lz} &= [0.90 \ 0.99] \\
 \lambda_{K\psi} &= [0.90 \ 0.99] & \lambda_{L\psi} &= [0.90 \ 0.99]
 \end{aligned} \tag{7.1}$$

The Internal Model Principle (IMP) controller is equipped with a ramp function as described in Table 6.2 in Section 6.2.2. Furthermore, the IMP controller is stacked on top of an already existing controller, which is the feedback, feedforward controller and observer with poles as provided in Equation (7.1). The poles of the IMP controller itself are placed at:

$$\begin{aligned}\lambda_{Sx} &= [0.90 \quad 0.99] \\ \lambda_{Sy} &= [0.90 \quad 0.99] \\ \lambda_{Sz} &= [0.90 \quad 0.99] \\ \lambda_{S\psi} &= [0.90 \quad 0.99]\end{aligned}\tag{7.2}$$

The Repetitive Control (RC) filter can unfortunately not be designed with a simple feedback, feedforward controller with observer, due to near unobservable states. Instead, the repetitive controller is designed by using a linear-quadratic-gaussian (LQG) controller, which consists of a Kalman filter and a linear-quadratic-regulator (LQR). Three different RC filter sizes are tested, namely $N = \{1, 11, 21\}$. The cost matrices for the LQG controller are in all cases kept at identity.

Furthermore, the simulation consists of four different situations, in which the reference is kept constant. First of all, a base line is set in which no wind is present. Next, a constant wind is added as disturbance. In the third simulation, a gust of wind is added to the constant wind and in the last simulation, turbulence is added. The last simulation contains no wind model, but has a reference for four different states to track. The results of the simulations are considered in the next subsections, where each controller is evaluated based on performance during hovering. The various different wind conditions and resulting disturbance force applied to the quad-copter in simulation can be seen in Figure 7.1.

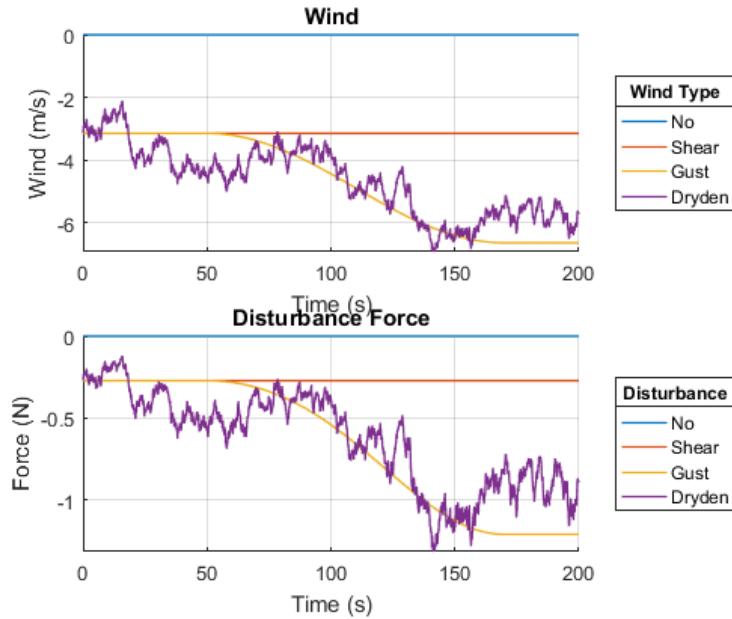


Figure 7.1: Wind conditions and resulting disturbance force applied to the quad-copter. All moments, forces and wind can be seen in Appendix N.

In order to assess the quality of the positioning performance of the quad-copter, the euclidean distance is calculated between the reference and the quad-copter position. Next, the Root Mean Square Error (RMSE) is calculated, which provides a measure over time to assess the positioning accuracy. The error function E and the RMSE values are obtained by:

$$\begin{aligned}E_i &= \sqrt{(r_{xi} - y_{xi})^2 + (r_{yi} - y_{yi})^2 + (r_{zi} - y_{zi})^2} \\ RMSE &= \sqrt{\frac{1}{N} \sum_{i=1}^N E^2}\end{aligned}\tag{7.3}$$

7.2 Windless conditions

In the windless condition case, the quad-copter is sent to a height of 1m and all other states are kept zero. The simulation runs for 200 seconds, however in Figure 7.2, only the first 40 seconds are depicted. The RMSE is calculated over the full 200 seconds. The results of the simulation are shown in Figure 7.2 and Table 7.2.

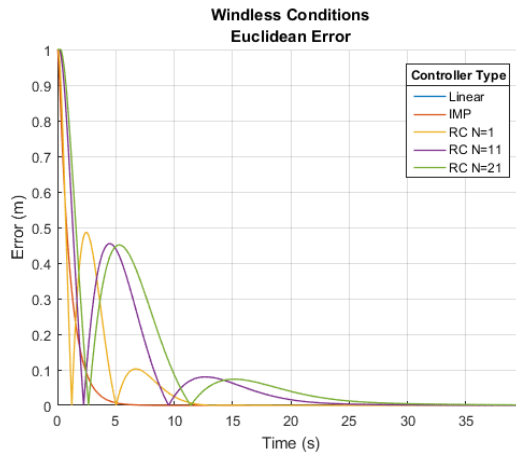


Figure 7.2: Euclidean error of the five different controllers without wind. The quad-copter is sent to 1 meter altitude. Furthermore, the simulation runs over 200 seconds. Only the first 40 seconds are depicted. Additional figures are presented in Appendix O.

Controller	Filter size	RMSE [m]
Linear		0.05447
IMP		0.05447
RC	1	0.06956
RC	11	0.09193
RC	21	0.10066

Table 7.2: Controller performance results in windless conditions.

The results from Table 7.2 and Figure 7.2 show that the linear controller and the IMP controller have similar control performance. Main reason for the comparable performance is due to the fact that the IMP controller is designed such that it rejects disturbances. This simulation, however, does not contain disturbances applied to the quad-copter. Since no disturbance enters the system, the IMP control part is not involved in the control action. Therefore, only the linear control part handles positioning.

However, it is noteworthy that all the RC filters do not show an increased performance compared to the linear controller. However, the RC action takes longer before steady state is reached, as can be seen in Figure 7.2. More results can be seen in Appendix O.

7.3 Shear wind

In shear wind conditions, the quad-copter is sent to a height of 1m and all other states are kept zero. The simulation runs for 200 seconds. Only the first 40 seconds are depicted in Figure 7.3. However, the RMSE is calculated over the full 200 seconds. The results of the simulation are shown in Figure 7.3 and Table 7.3.

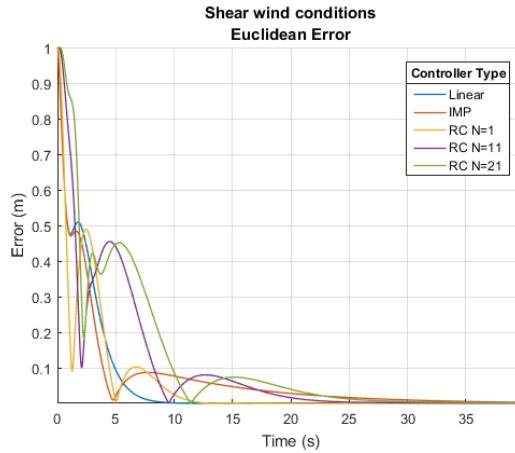


Figure 7.3: Euclidean error of the five different controllers in shear wind conditions. The quad-copter is sent to 1 meter altitude. Furthermore, the simulation runs over 200 seconds, however, for visibility reasons, only the first 40 seconds are shown. Additional figures are presented in Appendix P.

Controller	Filter size	RMSE [m]
Linear		0.07275
IMP		0.07014
RC	1	0.07007
RC	11	0.09588
RC	21	0.11018

Table 7.3: Controller performance results in shear wind conditions.

During the no wind condition, the linear controller and IMP controller showed similar performance. However, during the shear wind experiment, the IMP controller shows a significant improvement over the linear controller. As can be seen in Figure 7.3, as soon as a disturbance is introduced, the IMP controller starts interacting, which results in improvement of the control action.

Furthermore, the smaller RC action also show a significant improvement over the linear controller. Only the larger RC action is worse in the first 10 seconds of the simulation. This can mostly be addressed to the delay character of the RC filter in which the control action send to the plant acts on a delay and needs to settle.

Overall all controllers show an improvement on the linear controller. Although the improvement is only marginal in case of an RC filter with $N = 21$. More results can be seen in Appendix P.

7.4 Wind gust

During the wind gust conditions simulation, the quad-copter is sent to a height of 1m and all other states are kept zero. The simulation runs for 200 seconds, which is also shown in Figure 7.4. The RMSE is calculated over the full 200 seconds. The results of the simulation are shown in Figure 7.4 and Table 7.4. Subfigure 7.4a represents moving the quad-copter to the desired height under shear wind, which is the same simulation conditions as in Section 7.3. In Subfigure 7.4b the error is shown when the wind gust starts disturbing the quad-copter.

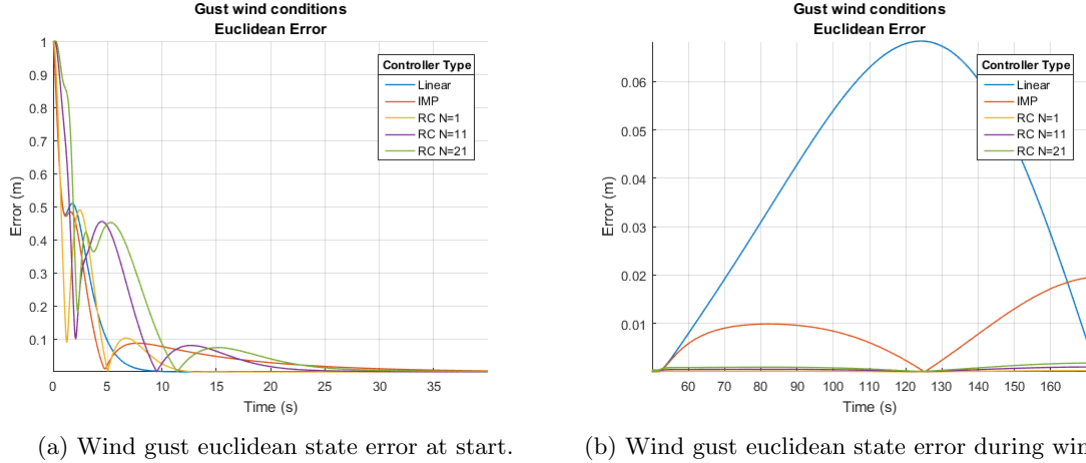


Figure 7.4: Euclidean error of the five different controllers in wind gust conditions. The quad-copter is sent to 1 meter altitude. Additional figures are presented in Appendix Q.

Controller	Filter size	RMSE [m]
Linear		0.08117
IMP		0.07063
RC	1	0.07007
RC	11	0.09588
RC	21	0.11019

Table 7.4: Controller performance results in a wind gust.

The results of a quad-copter during a gust of wind is comparable to a situation where the quad-copter is flying in shear wind. The main reason for the comparable results is that the gust of wind is applied on top of a shear wind, which means the controller is already suppressing the shear wind. Furthermore, a gust of wind only contains dynamics when the gust is applied, as is shown in Figure 7.1. When the gust of wind is applied, it can be treated as a shear wind with a higher wind velocity.

Furthermore, when the wind gust is applied in Figure 7.4b, the linear controller experiences a large euclidean error. All the other controllers are capable of keeping the error rather small. This is also confirmed in Table 7.4. However, the RC filter with size $N = 21$ still maintains a large tracking error, which is comparable during shear wind simulation. This can be fully described to the error dynamics at the start of the simulation, in which the controller maneuvers the quad-copter to 1 meter altitude. Within this moment, the RC filter starts to fill with delayed control input values. Therefore, does not yet apply a proper control action. This is also confirmed in Figure 7.4a in which it is shown that the RC action holds the highest peak at the start of the simulation. More results can be seen in Appendix Q.

7.5 Dryden wind

The last wind simulation experiment consists of the quad-copter flying in Dryden turbulent wind fields. The quad-copter is sent to 1 meter altitude and all other states are kept at zero. The simulation runs for 200 seconds, which is shown in Figure 7.1 and Table 7.5. The RMSE is calculated over the full 200 seconds.

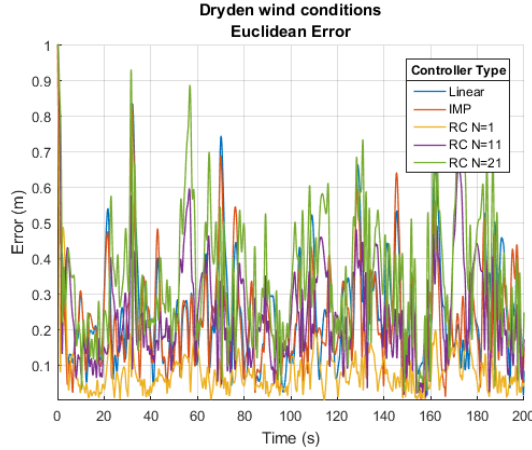


Figure 7.5: Euclidean error of the five different controllers in Dryden wind conditions. The quad-copter is sent to 1 meter altitude..

Controller	Filter size	RMSE [m]
Linear		0.29001
IMP		0.27344
RC	1	0.10781
RC	11	0.27760
RC	21	0.39273

Table 7.5: Controller performance results in Dryden wind conditions. Additional figures are presented in Appendix R.

The results show that Dryden wind conditions, which is a military graded turbulent air flow, results in large Euclidean errors. The results of all controllers vary significantly. Most remarkable are the RC filters of size $N = 11$ and $N = 21$ which are not able to make an improvement compared to the linear controller.

Furthermore, the IMP controller shows the best results. Although the Euclidean error grows slightly compared to the performance during wind gust simulation and shear wind conditions simulation on the IMP controller. The slightly growing error for the IMP controller can be addressed to the stochastic behavior of the Dryden wind flow acting on the quad-copter.

A third remark, the RC filter of size $N = 1$ and the IMP controller had similar performance results during shear wind simulation and the wind gust simulation. However, during Dryden turbulent wind simulation the small RC filter results in a much larger error. The results can be addressed to the Dryden turbulent wind, which behaves rather stochastically then deterministic. Therefore, there are no guarantees on repetitions in the disturbance signal and the RC filter is incapable of generating a proper disturbance rejection control input. Nonetheless, the RC filter with filter size $N = 1$ is still capable of improving the performance compared to the linear controller. More results can be seen in Appendix R.

7.6 Dynamical reference

The Dynamical reference simulation is an experiment in which no wind is applied. However, the reference is changed into a more challenging one. The reference consists of moving to 1 meter altitude and the X and Y references consists of a growing and shrinking circular pattern, which is depicted in Figure 7.6a. The results of the experiment are provided in Figure 7.6b and Table 7.6.

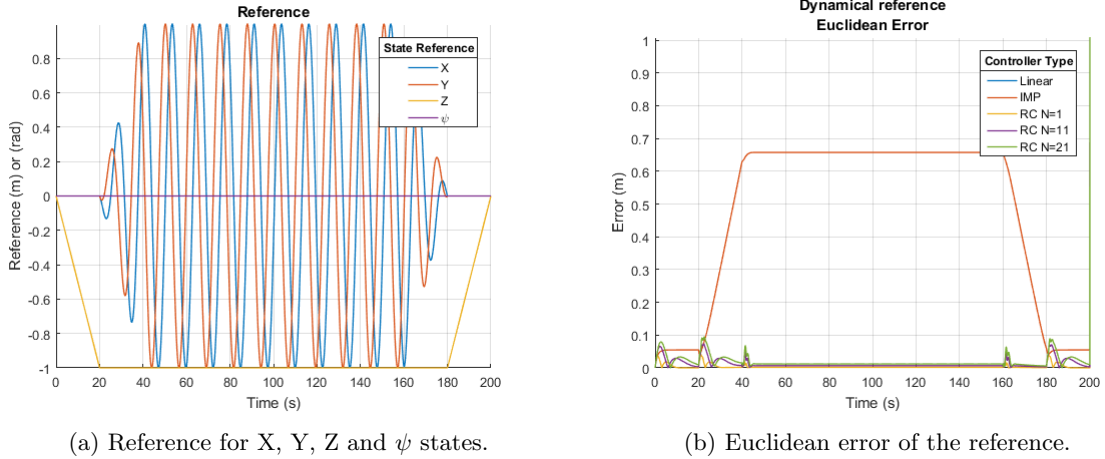


Figure 7.6: Euclidean error of the five different controllers in dynamical conditions. The quad-copter is sent to 1 meter altitude. The X and Y reference consists of a growing and shrinking circular pattern. All other states are kept zero. Additional figures are presented in Appendix S.

Controller	Filter size	RMSE [m]
Linear		0.53746
IMP		0.53746
RC	1	0.01007
RC	11	0.01926
RC	21	0.02585

Table 7.6: Controller performance results in dynamical flight conditions.

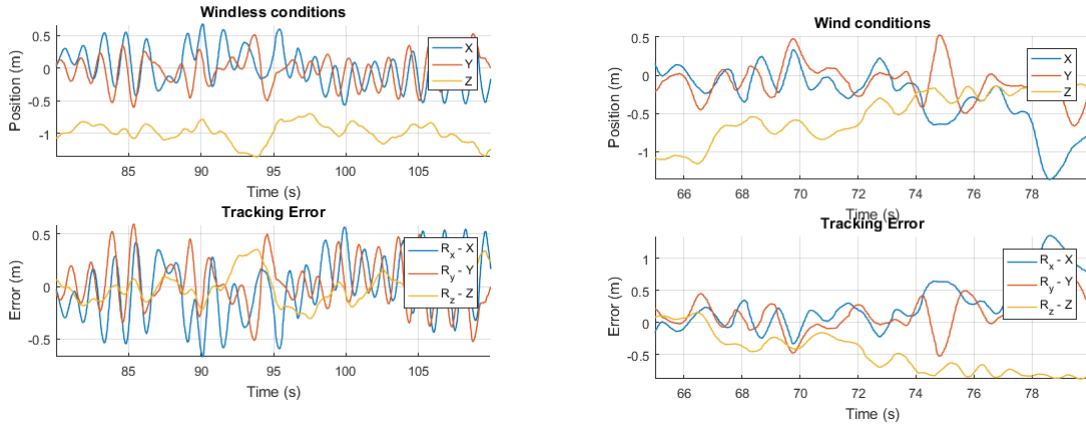
The dynamical reference experiment shows a significant difference between the RC filters and the IMP controller and linear controller. As depicted in Figure 7.6b, the IMP controller and the linear controller have a comparable Euclidean error profile. This can be addressed to the procedure to IMP controller is designed. The IMP controller is designed to only reject disturbance acting on the quad-copter. If no disturbances are applied, the IMP controller does not influence the tracking behavior of the quad-copter.

The RC filters are designed in such a way that they try to minimize the tracking error existing between the reference and the states of the quad-copter. Since a challenging reference is provided to the system, an tracking error is generated due to the lack of the linear controller being capable of tracking the reference properly. Since the error is a result of a deterministic signal, the RC filters can easily learn the proper control input and reduce the error significantly. In contrast to the Dryden turbulent wind simulation, the tracking error had a more stochastic behaviour and could therefore not be properly learned by the RC filters.

Furthermore, all the RC filters show similar RMSE values. The RC filter with filter size $N = 1$ shows improved results compared to the larger filter sizes. However, this can be seen as marginally larger. More results can be seen in Appendix S.

7.7 Experimental results

Experimental results were conducted on the Tech United field. Firstly, the quad-copter is sent to 1 meter altitude and all other states are kept zero. The experiment runs, for as long as the quad-copter can maintain its position in a safe manner. Due to the length of the experiment, the RMSE value can not be calculated, since both experiments have different run times. The implemented controller is a feedback controller as is discussed in Section 4.3. The results of the experiments are depicted in Figure 7.7.



(a) Experimental results hovering in windless conditions. (b) Experimental results hovering in wind conditions.

Figure 7.7: Results of experiments on testbed with a feedback controller.

As can be seen in Figure 7.7, the error of the hovering maneuver in windless conditions is within 0.5 meter of the reference, consistently. During the wind experiments, the quad-copter loses altitude, and larger errors between the reference and the state occur.

However, during the experiments, the dynamics of the quad-copter show vibrations which are not damped. The vibrations are related to the delay present in the Marvelmind. In Figure 7.8 is an experiment shown, where temporarily a sonar is connected to the quad-copter, and is measuring the distance between the quad-copter and the ground. During the experiment, the quad-copter was moved by hand up and down, and as can be seen, the delay in the Marvelmind is approximated at 0.5 seconds. The delayed position measurements are used in the observer to estimate the velocity, which is used in the controller to stabilize the position of the quad-copter. Therefore, the vibrations are present, due to the delay, which can not be damped by a D-gain.

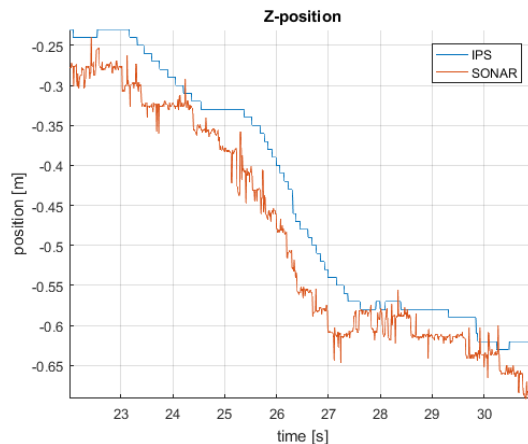


Figure 7.8: Marvelmind sampling delay.

Chapter 8

Conclusion, Remarks and Future work

The main goal of this research was to develop a controller which rejects disturbances applied to a quad-copter in the form of wind in various different complexities. Most commercially available software for quad-copters is either locked, or do not allow for an easy change in software, such that different controllers can not easily be implemented and tested. Therefore, the complete control and software architecture is designed in this thesis, which operates on top of already existing software, allowing for easy and flexible programming. This chapter reflects on the used hardware, designed software and the state estimators and controller design. Furthermore, this chapter reflects also on the wind models, the proposed disturbance rejection algorithms and the simulation results.

8.1 Conclusion

8.1.1 Hardware, Software and Control

Hardware

The hardware used in this master thesis project consists of a Team Black Sheep (TBS) quad-copter equipped with a Pixhawk as Flight Management Unit (FMU). The Pixhawk is a device developed by Eidgenössische Technische Hochschule Zürich (ETH Zürich) for the purpose to control quad-copters. The development of the Pixhawk resulted in a board which contains the necessary sensors to fly the quad-copter on angular control. Sensors included on the Pixhawk are a magnetometer, gyroscope and accelerometer. Furthermore, the Pixhawk is equipped with different commonly used and supported communication channels, such as Serial Peripheral Interface (SPI), Controlled Area Network (CAN) Inter-Integrated Circuit (I2C) and Universal Asynchronous Receiver/Transmitter (UART). Although most sensors are available on the Pixhawk, additional sensors are needed. First of all, measuring battery health during flight, which is achieved with an 3D Robotics (3DR) power module. Secondly, measuring the position of the quad-copter in 3D space, which is achieved by the Marvelmind Beacons and mobile beacon (hedgehog). Both sensors can be connected via the existing communication channels on the Pixhawk. The quad-copter is propelled by four Tiger Motor Air gear 350 rotor driving equipment, which exists of four Electronic Speed Controller (ESC)s, 2 clockwise (CW) rotating motor propellers combination and 2 counter clockwise (CCW) rotating motor propeller combination. The ESC are controlled by the Pixhawk by sending Pulse Width Modulation (PWM) signals, which results in matching Rounds per minute (RPM) on motor side. The wireless communication between the Pixhawk and the ground station computer is maintained by a Raspberry pi, which is connected via the general purpose input output (GPIO) pins and a UART channel. Lastly, a Radio Control Unit (RC) is connected to the Pixhawk, such that the user has control over the quad-copter in terms of sending the quad-copter to desired angular or translational positions. Above all, the RC is used as a failsafe device, which dis-arms the motors in case requested by the user.

Software

Q-ground control is a software platform to control a quad-copter equipped with a Pixhawk. Q-ground control consists of various modules, which each handle separate tasks during flight, such as storage, external connectivity, drivers, message bus and flight control. Some of the modules are replaced with Simulink software, such as the flight control. However, other modules were reused, such as drivers and the Ubiquitous Object Request Broker (uORB) message bus. Only the necessary modules are booted and all other modules are disabled by alternating the boot process. During this master thesis, a complete new software architecture is designed, which contains Simulink Pixhawk toolbox blocks. Furthermore, the software contains the ability to log data locally to the Secure Digital card (SD-card) for after-flight evaluation. Furthermore, the ability to communicate pre-defined data to the ground station was designed

as well. The RC is configured such that it enables the user to dis-arm, arm or put the quad-copter into configuration mode. Furthermore, a selection can be made, such that the quad-copter runs on angle control, altitude-hold or on full position control. During the project, additional software needed to be installed on the Raspberry pi before it could be used as a wireless communication device. Furthermore, the Indoor Positioning System (IPS) needed a driver, before the Pixhawk was able to obtain a position. The driver was developed based on the National Marine Electronics Association (NMEA) 0183 sentence based communication protocol and communication between the hedgehog and the Pixhawk was established. Lastly, for convenience of analyzing flight data afterwards, a Graphical User Interface (GUI) was developed which allows faster analyzing of the flight data.

Mathematical model, state estimation and control

A non-linear mathematical model is obtained via various different literature. The model is designed based on Tait-Bryant angles in a North-East-Down reference frame (NED frame) configuration. For convenience the non-linear model is linearized and decoupled into two 2-Degree of freedom (DOF) and two 4-DOF systems. Furthermore, it was assumed the quad-copter flies in near hovering modes. The propulsion model is obtained via a thrust stand, which allows to create a mapping between commanded Pulse Width Modulation (PWM) values and generated thrust by the propellers. An algorithm for fitting ellipsoids on experimental data is used to calibrate both magnetometer and the accelerometer. Furthermore, the magnetometer was corrected for field distortions generated by the motors. The gyroscope is calibrated by rotating the Pixhawk a known amount of degrees and integrating the gyroscope data allows to match the angles with the rotated degrees. State estimation is performed in two steps. Due to complexity of the environment, such as a metal drone cage and re bar in the ground, a Madgwick filter is used to obtain angles. The Madgwick evaluates the magnetometer data based on dip angle and magnitude and applies the measurement accordingly to the estimator. The translational states are obtained by fusing the IPS position data with the accelerometer data by using a Kalman filter. Lastly a controller is designed based on a Proportional Integral Derivative (PID) controller which stabilizes the quad copter and brings the quad copter to the desired reference. Furthermore, the controller was evaluated over different cascaded control structures and was shown to be stable.

8.1.2 Wind disturbance rejection

Wind field modeling

A Computational Fluid Dynamics (CFD) is considered in the form of Navier-Stokes equations as a model-driven approach to reject wind disturbances. However, a CFD consists of a Partial Differential Equation (PDE), which needs to be solved. Solving the PDE, or in this particular case, the well known Navier-Stokes equations involves numerical methods. A CFD consists of four important parts, which affect the accuracy of wind flow modeling and simulation results. First of all, setting up relations including Navier-Stokes equations, continuity equation and obtaining a relation between air flows and pressure. Although, density, temperature and compressibility of gasses influences the equations as well, they can be assumed constant over time for convenience. Finding a numerical solver, to solve the Navier-Stokes equations, effects the accuracy of the simulation. In general an Finite Difference Method (FDM), Finite Volume Method (FVM) or Finite Element Method (FEM) schemes can be used. Each numerical solver has its own trade-off such as easy implementation but limited complex mesh generation for Finite Difference Method (FDM) to hard mathematical implementations but complex mesh generation for Finite Element Method (FEM). A third important part in CFD is setting up the boundary conditions. The boundary conditions depends on the body to which the flow is applied, as well as the environment in which the material flows. During flight, the boundary conditions are a big unknown, especially how air flows through the environment. Lastly, Computational Fluid Dynamics (CFD) require stability analysis. In some cases, if Computational Fluid Dynamics (CFD) designing is improperly performed, results will not converge to a solution. This excludes the solution of building a Computational Fluid Dynamics (CFD) estimation scheme to estimate the wind during flight on the quad-copter. Although it might be suitable for simulation purposes. However, easier solutions exist to model the wind in simulation, namely Dryden Turbulent models. Due to the simplicity, yet realistic results, a Dryden Turbulent model is used in simulation.

Disturbance rejection

An IMP method based controller is designed during this project. Firstly a feedback controller, feedforward and observer are designed for continuous time. A disturbance can be modeled as an exogenous system, which can have complex internal dynamics. A disturbance rejection system is designed in which only the disturbance is rejected and the reference tracking performance is unchanged. The IMP method based controller design is also repeated for discrete time case. Overall, the IMP controller can be stacked on top of already existing linear controllers. However, perfect knowledge of the plant model is needed to design a control law involving IMP methods, such that only disturbance rejection is achieved. Furthermore, a RC method based controller is designed. By lengthening the filter size, low frequent signals can be rejected. As a result, the RC method improves reference tracking behavior regardless of the presence of a disturbance.

Experiments and results

Five different controllers were evaluated in five different situations. A comparison is made between a linear controller, an IMP controller and three RC filters with varying filter sizes. The first simulation exist of only hovering in windless conditions to lay down a benchmark for all controllers. In the second experiment, shear wind is applied to the quad-copter. A wind gust is applied in the third simulation. Dryden turbulence model is applied in the fourth simulation. The fifth simulation evaluates the effect of each controller on the tracking error when a challenging reference is provided to the quad-copter. The results of the controllers are compared by calculating the Root Mean Square Error (RMSE) of the euclidean distance between the quad-copter and the reference. Results show an increase in performance for the IMP method in all disturbed cases. However, the tracking error is not improved in case of undisturbed conditions, concluding the IMP method is focused on disturbance rejection. Furthermore, RC filters show an increase in reference tracking compared to a linear controller or IMP controller. Furthermore, in windy conditions without turbulence all RC filter show an improvement. However, due to the non-repetitiveness, low RC filter sizes show best performance. During turbulent wind conditions only the small RC filter size achieves slightly better performance compared to a linear controller. Larger sized RC filter show a decrease in tracking performance. Depending on the needs of the system such as improving reference tracking or flying in turbulent wind fields, an IMP controller or an RC filter of size $N = 1$ are the better suited options.

8.2 Future work

8.2.1 Hardware, Software and Control

Based on the first part of the report, where the focus is on hardware, software and controller design, a few notes of improvements can be given. Mainly due to the novelty and at the time poorly documented information, some solutions might be obsolete. First of all the Matlab Pilot Support Package (PSP) is already updated to a newer version. The update might result to easier toolbox implementations. Most importantly, additional functionality could possibly already be available in the PSP.

In the Simulink real-time toolbox, blocks exist which allow for asynchronous, sentenced based communication. The blocks in question, ASCII Encoders/Decoders and First In, First Out (FIFO) Read/Write buffers, might contribute to an improved and flexible communication strategy. The blocks were not used in this project due to a lack of C-code generation abilities. Either a S-function can be written to use the blocks or a TLC-wrapper. Another solution to enhance the communication, is to find a solution to enable using the MAVLink protocol. Currently, the Matlab PSP requires disabling MAVLink, in order to allow code uploading to the Pixhawk. In addition to the communication lines, a Raspberry pi is used to communicate between computer and quad-copter. By providing a WiFi network and letting multiple Raspberry Pi's communicate can enhance and extend the designed software for quad-copter to formation flights or let the quad-copter cooperate other available ground robots.

Since the flight stack designed by Qground-control is documented during this master thesis, parts of the flight stack can be reconsidered in re-usability in combination with the designed Simulink software. At the time of developing software, a lack of Q-ground control documentation resulted in developing a completely newly designed flight stack in Simulink. An important factor in re-using flight modules is the Ubiquitous Object Request Broker (uORB) module. Since more documentation start to appear about uORB, connections between modules can be obtained between Simulink software more directly.

During control structure and observer design, the decision was made to use a Madgwick and a Kalman filter. Reasons for a Madgwick filter is the ability to assess the quality of the magnetometer data. A kalman filter was used since the Indoor Positioning System (IPS) had a low sampling rate. In future work, additional filters can be explored such as extended Kalman filters with quaternions angle representations. In addition to filtering the magnetometer data properly, a hybrid automaton can be used in the form of an extended Kalman filter, where switching sequences is determined based in dip angle and magnitude, such that disturbed magnetometer measurements can be rejected. Additionally, sensor fusion algorithms allowing to incorporate multiple sensors can be considered as well. Sensor fusion algorithms do not need to be limited to various different sensors, using multiple comparable sensors in different locations can result in improvements as well. For instance, magnetometers on the tail and the front of the quad-copter or centered and elevated above the quad-copter.

The Indoor Positioning System (IPS) driver is designed based on the expected sentences broadcast by Marvelmind. This might results in receiving no data at all if the user configures the Marvelmind wrongly. A more practical convenient solution would be to program the driver such that it can configure the Marvelmind hedgehog once the Pixhawk is booted. Furthermore, the delay present at sampling the Marvelmind needs to be reduced as well, in order to perform smoother quad-copter flights.

Furthermore, this master thesis project utilizes mathematical system identification approaches, where some parameters are measured or calculated. A more robust solution, such as estimating a non-linear model from measurement data to obtain a quad-copter model, would be a better approach.

8.2.2 Wind disturbance rejection

The forces and moments acting on the quad-copter are related to the wind via Lord Rayleigh functions. However, a better aerodynamic model, involving complex body geometries can improve the simulator in terms of realistic behavior.

Although an improvement in terms of controller performance is shown, the results are based on a simulator. In order to validate measurement data in a proper way, the same experiments need to be conducted in a wind tunnel, such that the desired wind and environment is controlled. This leads to evaluation over various wind regimes instead of the fixed wind regimes generated by the fan or simulator.

The Internal Model Principle and Repetitive Control methods can be extended to non-linear control of the quad-copter. This master thesis assumed near hovering flight modes, to which a linear model is synthesized. If the non-linear quad-copter model is input/output feedback linearized, the control dynamics still have a linear characteristic, at which an Internal Model Principle or Repetitive Control method can be included in the control structure. Therefore, an Internal Model Principle or Repetitive Control method can be implemented on a quad-copter coping with non-linearities as well.

During the desing process of the proposed controllers using Internal Model Principle and Repetitive Control methods, it is assumed to have perfect knowledge of the quad-copter model. However, the system is modeled based on literature. Furthermore, system parameters are estimated by weighing parts and calculating inertias. In order to implement the Internal Model Principle or Repetitive Control method, a robustness analysis against parameter uncertainty should be made.

The Internal Model Principle controller uses a driven model to compensate for wind flows acting on the quad-copter. In case of a wind tunnel test the IMP states can be compared against the wind data in order to obtain a correlation between both data. If a correlation exist, it might be possible to reconstruct the wind based on the internal states of the quad-copter leading to a flying wind measurement device.

Bibliography

- [1] Y. Petrov, “Ellipsoid fit.” <https://nl.mathworks.com/matlabcentral/fileexchange/24693-ellipsoid-fit>, 2015. Mathworks.
- [2] D. Joshi, “Exploring the latest drone technology for commercial, industrial and military drone uses,” *Tech Insider*, 2017.
- [3] L. CAMILLI, “Emerging technologies, applications, regulations, and market challenges in the consumer aerial drone industry,” in *Conference: San Francisco State College of Business, At San Francisco*, 2015.
- [4] B. Silver, M. Mazur, A. Wisniewski, and A. Babicz, “Welcome to the era of drone-powered solutions: a valuable source of new revenue streams for telecoms operators,” 2017.
- [5] R. Creutzburg, “European activities in civil applications of drones: an overview of remotely piloted aircraft systems (rpas),” in *Mobile Multimedia/Image Processing, Security, and Applications 2015*, vol. 9497, p. 949707, International Society for Optics and Photonics, 2015.
- [6] A. L. Salih, M. Moghavvemi, H. A. Mohamed, and K. S. Gaeid, “Modelling and pid controller design for a quadrotor unmanned air vehicle,” in *Automation Quality and Testing Robotics (AQTR), 2010 IEEE International Conference on*, vol. 1, pp. 1–5, IEEE, 2010.
- [7] Z. He and L. Zhao, “A simple attitude control of quadrotor helicopter based on ziegler-nichols rules for tuning pd parameters,” *The Scientific World Journal*, vol. 2014, 2014.
- [8] V. K. Tripathi, L. Behera, and N. Verma, “Design of sliding mode and backstepping controllers for a quadcopter,” in *Systems Conference (NSC), 2015 39th National*, pp. 1–6, IEEE, 2015.
- [9] S. Bouabdallah and R. Siegwart, “Backstepping and sliding-mode techniques applied to an indoor micro quadrotor,” in *None*, 2005.
- [10] Y.-C. Choi and H.-S. Ahn, “Nonlinear control of quadrotor for point tracking: Actual implementation and experimental tests,” *IEEE/ASME transactions on mechatronics*, vol. 20, no. 3, pp. 1179–1192, 2015.
- [11] N. Jeurgens, “Implementing a simulink controller in an ar. drone 2.0,” Master’s thesis, Eindhoven University of Technology, 2016.
- [12] V. Stepanyan and K. S. Krishnakumar, “Estimation, navigation and control of multi-rotor drones in an urban wind field,” in *AIAA Information Systems-AIAA Infotech@ Aerospace*, p. 0670, 2017.
- [13] M. Kothari, I. Postlethwaite, and D.-W. Gu, “Uav path following in windy urban environments,” *Journal of Intelligent & Robotic Systems*, vol. 74, no. 3-4, pp. 1013–1028, 2014.
- [14] D. Cabecinhas, R. Cunha, and C. Silvestre, “A globally stabilizing path following controller for rotorcraft with wind disturbance rejection,” *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 708–714, 2015.
- [15] J. W. Langelaan, N. Alley, and J. Neidhoefer, “Wind field estimation for small unmanned aerial vehicles,” *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 4, pp. 1016–1030, 2011.
- [16] N. Sydney, B. Smyth, and D. A. Paley, “Dynamic control of autonomous quadrotor flight in an estimated wind field,” in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pp. 3609–3616, IEEE, 2013.
- [17] Z. J. Zhai, Z. Zhang, W. Zhang, and Q. Y. Chen, “Evaluation of various turbulence models in predicting airflow and turbulence in enclosed environments by cfd: Part 1—summary of prevalent turbulence models,” *Hvac&R Research*, vol. 13, no. 6, pp. 853–870, 2007.
- [18] Z. Zhang, W. Zhang, Z. J. Zhai, and Q. Y. Chen, “Evaluation of various turbulence models in predicting airflow and turbulence in enclosed environments by cfd: Part 2—comparison with experimental data from literature,” *Hvac&R Research*, vol. 13, no. 6, pp. 871–886, 2007.
- [19] H. Haverdings, “Helicopter emergency medical service (hems) from a rooftop in amsterdam: A simulation perspective,” 2012.
- [20] Flying Qualities of Piloted Aircraft, “MIL-F-8785C,” in *Department of Defense Handbook*, Washington D.C.: U.S. Department of Defense, 1980.

- [21] Flying Qualities of Piloted Aircraft, “MIL-HDBK-1797,” in *Department of Defense Handbook*, Washington D.C.: U.S. Department of Defense, 1997.
- [22] Flying Qualities of Piloted Aircraft, “MIL-HDBK-1797B,” in *Department of Defense Handbook*, Washington D.C.: U.S. Department of Defense, 2012.
- [23] W. Gawronski, “Modeling wind-gust disturbances for the analysis of antenna pointing accuracy,” *IEEE Antennas and propagation magazine*, vol. 46, no. 1, pp. 50–58, 2004.
- [24] S. Yoon, H. C. Lee, and T. H. Pulliam, “Computational analysis of multi-rotor flows,” in *54th AIAA Aerospace Sciences Meeting*, p. 0812, 2016.
- [25] C. R. Russell, J. Jung, G. Willink, and B. Glasner, “Wind tunnel and hover performance test results for multicopter uas vehicles,” 2016.
- [26] F. Hoffmann, N. Goddemeier, and T. Bertram, “Attitude estimation and control of a quadrocopter,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 1072–1077, IEEE, 2010.
- [27] D. Cabecinhas, R. Cunha, and C. Silvestre, “A nonlinear quadrotor trajectory tracking controller with disturbance rejection,” *Control Engineering Practice*, vol. 26, pp. 1–10, 2014.
- [28] P. Hippe and J. Deutscher, *Design of observer-based compensators: From the time to the frequency domain*. Springer Science & Business Media, 2009.
- [29] L. Marconi, A. Isidori, and A. Serrani, “Autonomous vertical landing on an oscillating platform: an internal-model based approach,” *Automatica*, vol. 38, no. 1, pp. 21–32, 2002.
- [30] G. Casadei, L. Furieri, N. Mimmo, R. Naldi, and L. Marconi, “Internal model-based control for loitering maneuvers of uavs,” in *Control Conference (ECC), 2016 European*, pp. 672–677, IEEE, 2016.
- [31] S. Hara, Y. Yamamoto, T. Omata, and M. Nakano, “Repetitive control system: A new type servo system for periodic exogenous signals,” *IEEE Transactions on automatic control*, vol. 33, no. 7, pp. 659–668, 1988.
- [32] L. Wang, “Tutorial review on repetitive control with anti-windup mechanisms,” *Annual Reviews in Control*, vol. 42, pp. 332–345, 2016.
- [33] Y. Wang, F. Gao, and F. J. Doyle III, “Survey on iterative learning control, repetitive control, and run-to-run control,” *Journal of Process Control*, vol. 19, no. 10, pp. 1589–1600, 2009.
- [34] Y. Chen, K. L. Moore, J. Yu, and T. Zhang, “Iterative learning control and repetitive control in hard disk drive industry—a tutorial,” in *Decision and Control, 2006 45th IEEE Conference on*, pp. 2338–2351, IEEE, 2006.
- [35] Team Black Sheep, “Team Black Sheep Discovery base plates.” <http://team-blacksheep.com/products/product:98>, 2008. [Online; accessed 3-July-2018].
- [36] Tiger-Motor, “Tiger-Motor Air Gear 350 multirotor driving equipment set.” <http://store-en.tmotor.com/goods.php?id=452>, 2008. [Online; accessed 3-July-2018].
- [37] P. Rooijackers, “Design and Control of a Quad-Rotor: Application to Autonomous Drone Refereeing,” Master’s thesis, Eindhoven University of Technology, 2017.
- [38] ivc , *TBS DISCOVERY Quadrocopter. Durable and crash resistant multirotor optimized for dynamic FPV flight*. Team Black Sheep, September 2014. <http://www.team-blacksheep.com/tbs-discovery-manual.pdf>.
- [39] ArduPilot Dev Team, “History of Ardupilot.” <http://ardupilot.org/planner2/docs/common-history-of-ardupilot.html>, 2008. [Online; accessed 4-July-2018].
- [40] PX4 Dev Team, “Pixhawk 1 Flight Controller.” https://docs.px4.io/en/flight_controller/pixhawk.html, 2008. [Online; accessed 3-July-2018].

- [41] ST, “Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer,” November 2013. <https://www.st.com/resource/en/datasheet/lsm303d.pdf>.
- [42] ST, “MEMS motion sensor: three-axis digital output gyroscope,” March 2013. <https://www.st.com/resource/en/datasheet/l3gd20h.pdf>.
- [43] InvenSense Inc., “MPU-6000 and MPU-6050 Product Specification Revision 3.4,” September 2013. https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf.
- [44] Ardupilot, “Common Power Module,” 2016. <http://ardupilot.org/copter/docs/common-3dr-power-module.html>.
- [45] Marvelmind, “Marvelmind Indoor Navigation System Operating manual,” January 2018. https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf.
- [46] G. S. K. Wong, “Speed of sound in standard air,” *The Journal of the Acoustical Society of America*, vol. 79, no. 5, pp. 1359–1366, 1986.
- [47] Marvelmind, “Communication of Pixhawk with Marvelmind mobile beacon,” October 2016. https://marvelmind.com/pics/marvelmind_pixhawk_v2016_10_11a.pdf.
- [48] Electrical Technology, “Brushless dc motor: Construction, working principle and application,” May 2016. <https://www.electricaltechnology.org/2016/05/bl-dc-brushless-dc-motor-construction-working-principle.html>.
- [49] Electronic Communications Committee, “Ecc decision (15)05. the harmonised frequency range 446.0-446.2mhz, technical characteristics, exemption from individual licensing and free carriage and use of analogue and digital pmr 446 applications,” July 2015. <https://www.ecodocdb.dk/download/b8797390-4577/ECCDec1505.pdf>.
- [50] FrSky, *FrSky 2.4GHz ACCST X8R Manual*, July 2017. <https://www.frsky-rc.com/wp-content/uploads/2017/07/Manual/X8R.pdf>.
- [51] FrSky, *FrSky 2.4GHz ACCST Taranis X9D Plus Manual*, Juli 2017. <https://www.frsky-rc.com/wp-content/uploads/2017/07/Manual/X9D%20Plus.pdf>.
- [52] Dronecode, “MAVLink Developer Guide.” <https://mavlink.io/en/>.
- [53] Dronecode, “uORB topics graph.” https://dev.px4.io/en/middleware/uorb_graph.html.
- [54] S. S. Yau and F. Karim, “Context-sensitive object request broker for ubiquitous computing environments,” in *Distributed Computing Systems, 2001. FTDCS 2001. Proceedings. The Eighth IEEE Workshop on Future Trends of*, pp. 34–40, IEEE, 2001.
- [55] Dronecode, “PX4 Architectural Overview.” <https://dev.px4.io/en/concept/architecture.html>.
- [56] Mathworks, “Pixhawk Pilot Support Package (PSP) User Guide.” <http://discuss.px4.io/uploads/default/original/2X/d/d8a49f4c01c834a7d65472408731a80a57560356.pdf>.
- [57] Mathworks, “Pixhawk Support Package.” <https://nl.mathworks.com/hardware-support/pixhawk.html>.
- [58] Dronecode, “System Startup.” https://dev.px4.io/en/advanced/system_startup.html.
- [59] M. Stigge, H. Plötz, W. Müller, and J.-P. Redlich, “Reversing crc—theory and practice,” 2006.
- [60] J. F. Kurose, *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.
- [61] FrSky, *Quickstart Guide for FrSky Taranis with OpenTX*, Juli 2017. <https://www.frsky-rc.com/wp-content/uploads/2017/07/Manual/Quickstart%20Guide%20for%20FrSky%20Taranis%20with%20OpenTX.pdf>.
- [62] vfrolov, “com0com.” <https://sourceforge.net/projects/com0com/>.

- [63] s novelli, “combyTCP.” <https://sourceforge.net/projects/combytcp/>.
- [64] National Marine Electronics Association *et al.*, “The NMEA 0183 Protocol,” URL: <http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf>, 2001.
- [65] R. B. Langley, “NMEA 0183: A GPS receiver interface standard,” *GPS world*, vol. 6, no. 7, pp. 54–57, 1995.
- [66] I. Lita, I. B. Cioc, and D. A. Visan, “A new approach of automobile localization system using GPS and GSM/GPRS transmission,” in *Electronics Technology, 2006. ISSE’06. 29th International Spring Seminar on*, pp. 115–119, IEEE, 2006.
- [67] M. Pedley, “High-precision calibration of a three-axis accelerometer.” http://cache.freescale.com/files/sensors/doc/app_note/AN4399.pdf, 2015. Freescale Semiconductor.
- [68] T. Ozyagcilar, “Calibrating an ecompass in the presence of hard- and soft-iron interference.” <https://www.nxp.com/docs/en/application-note/AN4246.pdf>, 2015. Freescale Semiconductor.
- [69] I. Sa and P. Corke, “System identification, estimation and control for a cost effective open-source quadcopter,” in *Robotics and automation (icra), 2012 ieee international conference on*, pp. 2202–2209, IEEE, 2012.
- [70] R. Mahony, T. Hamel, and J.-M. Pfimlin, “Nonlinear complementary filters on the special orthogonal group,” *IEEE Transactions on automatic control*, vol. 53, no. 5, pp. 1203–1218, 2008.
- [71] B. Fan, Q. Li, C. Wang, and T. Liu, “An adaptive orientation estimation method for magnetic and inertial sensors in the presence of magnetic disturbances,” *Sensors*, vol. 17, no. 5, p. 1161, 2017.
- [72] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” 2006.
- [73] S. O. Madgwick, A. J. Harrison, and R. Vaidyanathan, “Estimation of imu and marg orientation using a gradient descent algorithm,” in *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pp. 1–7, IEEE, 2011.
- [74] State of New South Wales, “North and south.” http://lrrpublic.cli.det.nsw.edu.au/lrrSecure/Sites/Web/Forces_and_fields_creative_commons/7306/other/earths_magnetic_field.html, 2008. [Online; accessed 3-July-2018].
- [75] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman, “Quaternion averaging,” 2007.
- [76] A. Sayma, *Computational fluid dynamics*. Bookboon, 2009.
- [77] J. D. Anderson and J. Wendt, *Computational fluid dynamics*, vol. 206. Springer, 1995.
- [78] J. Anderson, “Fundamentals of aerodynamics, 1991.”
- [79] F. H. Harlow and J. E. Welch, “Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface,” *The physics of fluids*, vol. 8, no. 12, pp. 2182–2189, 1965.
- [80] A. Gronskis and G. Artana, “A simple and efficient direct forcing immersed boundary method combined with a high order compact scheme for simulating flows with moving rigid boundaries,” *Computers & Fluids*, vol. 124, pp. 86–104, 2016.
- [81] T. Chung, *Computational fluid dynamics*. Cambridge university press, 2010.
- [82] B. Blocken, T. Stathopoulos, and J. Carmeliet, “Cfd simulation of the atmospheric boundary layer: wall function problems,” *Atmospheric environment*, vol. 41, no. 2, pp. 238–252, 2007.
- [83] M. Steinbuch, “Repetitive control for systems with uncertain period-time,” *Automatica*, vol. 38, no. 12, pp. 2103–2109, 2002.
- [84] R. Costa-Castelló, G. A. Ramos, J. M. Olm, and M. Steinbuch, “Second-order odd-harmonic repetitive control and its application to active filter control,” in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 6967–6972, IEEE, 2010.

Appendices

Appendix A

Pixhawk Pinouts

	1 (red)	2 (blk)	3 (blk)	Pin Nr.			7 (blk)
				4 (blk)	5 (blk)	6 (blk)	
TELEM1	Signal	VCC	TX	RX	CTS	RTS	GND
TELEM2	Volt	+5V	+3.3V	+3.3V	+3.3V	+3.3V	GND
GPS	Signal	VCC	TX	RX	CAN2 TX	CAN2 RX	GND
	Volt	+5V	+3.3V	+3.3V	+3.3V	+3.3V	GND
Serial 4/5	Signal	VCC	TX(#4)	RX(#4)	TX(#5)	RX(#5)	GND
	Volt	+5V	+3.3V	+3.3V	+3.3V	+3.3V	GND
ADC	Signal	VCC	ADC IN	GND			
6.6V	Volt	+5V	up to +6.6V	GND			
ADC	Signal	VCC	ADC IN	GND	ADC IN	GND	
3.3V	Volt	+5V	up to +3.3V	GND	up to +3.3V	GND	
I2C	Signal	VCC	SCL	SDA	GND		
	Volt	+5V	+3.3(pullups)	+3.3(pullups)	GND		
CAN	Signal	VCC	CAN_H	CAN_L	GND		
	Volt	+5V	+12V	+12V	GND		
SPI	Signal	VCC	SPLSCK	SPL_MISO	SPL_MOSI	!SPL_NSS	!GPIO
	Volt	+5V	+3.3V	+3.3V	+3.3V	+3.3V	+3.3V
Power	Signal	VCC	VCC	Current	Voltage	GND	GND
	Volt	+5V	+5V	up to +3.3V	up to +3.3V	GND	GND
Switch	Signal	VCC	io_led_safety	SAFETY			
	Volt	+3.3V	GND	GND			
Console Port	Signal	+5V	TX	RX	TX	RX	GND
	FTDI				5	4	1
Spektrum DSM Port	Pin	N/C	N/C	N/C	RX(yellow)	TX(orange)	GND(black)
	Signal	1(white)	2(black)	3(red)			
	Volt	Signal	GND	Vcc			
		+3.3V	GND	+3.3V			

Table A.1: Pixhawk pin numbers and signal.

Appendix B

Tiger Air gear 350 User Manual

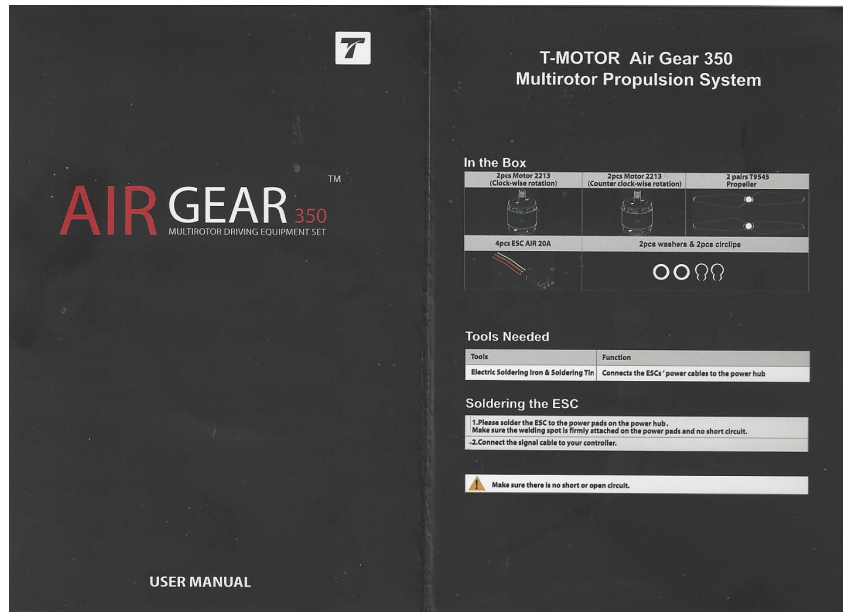


Figure B.1: User manual Tiger Air Gear 350 page 1-2.

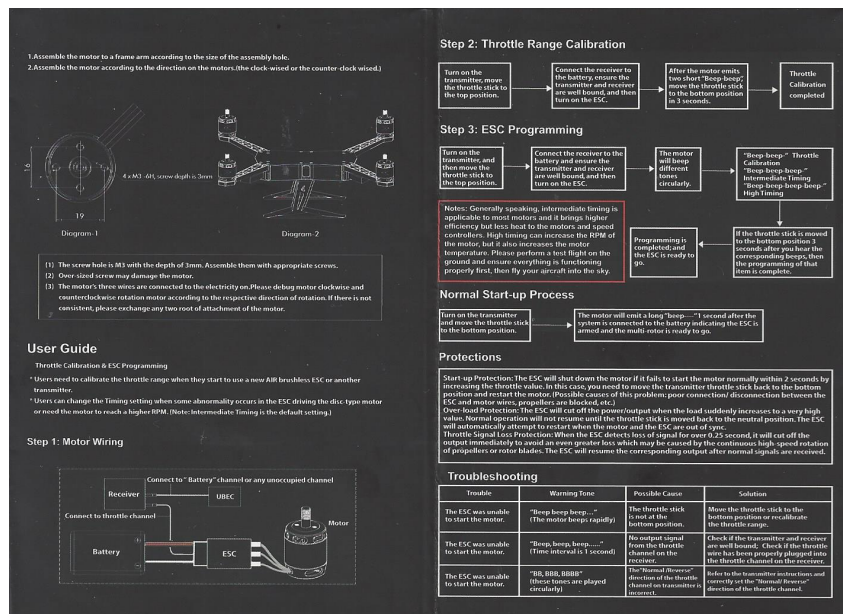


Figure B.2: User manual Tiger Air Gear 350 page 3-4.

Propeller installation

Propeller	CW thread prop (all black bullet holder)	CCW thread prop (black bullet holder with silver ring)
Diagram		
Installation site	Motor (Counter-clockwise rotation)	Motor (Clockwise rotation) (with a keyhole or a laser white dot on the shaft)

1. Tighten the propellers by following the instructions. Propellers are self-tightening during flight.
2. Keep the motor deadlocked in place and remove the propeller according to the instructions.

DO NOT use any thread locker on the threads.

Three different ways for props installation

Self-tightening installation
(Default Setting)

Nut pressure installation

Clamp pressure installation

Specifications:

Working temperature	-5° C~40° C
ESC	
Current	20A
Signal Frequency	600Hz
Voltage	11.1V 14.8V
Battery	3S~4S LiPo
Motor	
Sator Size	22-13mm
KV	920rpm/V
Weight	54g
Propeller	
Diameter/Thread Pitch	9.5x4.5 inch

Disclaimer

Read this disclaimer carefully before using this product. By using this product, you hereby agree to this disclaimer and signify that you have read from Tiger T-MOTOR assumes no liability for damage(s) or injuries incurred directly or indirectly from the use of this product.

Cautions

When flying, the fast rotating propellers will cause serious damage and injury. Therefore, please fly with a high safety awareness.

1. Keep flying multicopter away from obstacles, human beings, high-voltage lines and so on.
2. Do not get close to or touch the working motors and propellers, which will cause serious injury.
3. Make sure there is no short or open circuit.
4. Check that the propellers and the motors are installed correctly and firmly before flight.
5. Check whether all parts of multicopter are in good condition before flight. Do not fly with old or broken parts.
6. Use T-motor spare parts as much as possible.

If you have any problem you can not solve, please contact your dealer or T-MOTOR customer service.

T-MOTOR
WWW.RCTIGERMOTOR.COM

Figure B.3: User manual Tiger Air Gear 350 page 5-6.

Appendix C

Boot process Pixhawk

```
1  usleep 1000
2  uorb start
3
4  usleep 1000
5
6
7  # if this line is active, Simulink will not upload code to the Pixhawk via usb
8  #nshterm /dev/ttyACM0 &
9
10 usleep 1000
11
12
13 px4io start
14 usleep 1000
15
16
17 #commander start
18
19 #usleep 1000
20
21 #mavlink start -d /dev/ttyS1 -b 115200
22
23 #usleep 5000
24
25 #dataman start
26
27 #usleep 1000
28
29 #navigator start
30
31 #usleep 1000
32
33
34 sh /etc/init.d/rc.sensors
35 usleep 1000
36
37
38 #sh /etc/init.d/rc.logging
39
40 #usleep 1000
41
42 #gps start
43 attitude_estimator_q start
44
45 position_estimator_inav start
46
47 usleep 1000
48
49
50 mtd start
51
52 usleep 1000
53
54
55 param load /fs/mtd.params
56
57 usleep 1000
58
59
60 rgbled start
61
62 usleep 1000
```

```
63
64
65 gps start
66 usleep 1000
67
68 px4_simulink_app start
69
70 usleep 1000
71 exit
```

Listing C.1: Pixhawk boot alternation file.

Appendix D

Basic Logged Data Reading

```
1 clear all; clc
2
3 %% Load data
4 Folder = 'SensorCalibration';
5 file = 'Acc';
6
7 sensor_file = ['LoggedData\' Folder '\RawSensorData.' file '.bin']; % Sensor Test
8 [datapts, ~] = readdata(sensor_file); % Sensor tests
9
10 %% Plot Data
11 figure(1); clf; hold on;
12 for i = 1:size(datapts,1)
13     plot(datapts(i,:))
14 end
15 grid on; axis tight;
```

Listing D.1: Example Matlab code for reading and plotting the .bin file.

```
1 %% Load the data into MATLAB from a binary log file
2 % Usage: >> [datapoints, numpoints] = readdata('datafile.log')
3 % Header information format:
4 %     String "MWLOGV##"
5 %     Time/Date 4 bytes (time())
6 %     Number of Signals per record Logged 1 bytes (256 max)
7 %     Data Type of Signals Logged 1 bytes (1-10)
8 %     Number of bytes per record 2 (65535 max)
9 % Plot Data Example: plot([1:numpoints], datapoints(1,:), [1:numpoints], datapoints(2,:))
10 % MathWorks Pilot Engineering 2015
11 % Steve Kuznicki
12 function [datapts, numpts] = readdata(dataFile)
13 %%
14 datapts = 0;
15 numpts = 0;
16
17 if nargin == 0
18     dataFile = 'data.bin';
19 end
20
21 fid = fopen(dataFile, 'r');
22 % load the header information
23 hdrToken = fread(fid, 8, 'char');
24 if strcmp(char(hdrToken), 'MWLOGV', 6) == true
25     logTime = uint32(fread(fid, 1, 'uint32'));
26     numflds = double(fread(fid, 1, 'uint8'));
27     typefld = uint8(fread(fid, 1, 'uint8'));
28     recSize = uint16(fread(fid, 1, 'uint16'));
29     fieldTypeStr = get_elem_type(typefld);
30     datapts = fread(fid, double([numflds, Inf]), fieldTypeStr);
31     fclose(fid);
32     numpts = size(datapts, 2);
33 end
34
35 end
```

Listing D.2: Matlab function to read complete .bin file data.

```
1 %% Load the data into MATLAB from a binary log file
2 % Usage: >> [datapoints, numpoints] = readdata('datafile.log')
3 % Header information format:
```

```

4 %           String "MWLOGV##"
5 %           Time/Date 4 bytes (time())
6 %           Number of Signals per record Logged 1 bytes (256 max)
7 %           Data Type of Signals Logged 1 bytes (1-10)
8 %           Number of bytes per record 2 (65535 max)
9 % Plot Data Example: plot([1:numpoints], datapoints(1,:), [1:numpoints], datapoints(2,:))
10 % MathWorks Pilot Engineering 2015
11 % Steve Kuznicki
12 %% get the element type string
13 function [dtypeStr] = get_elem_type(dtype)
14     switch(dtype)
15     case 1
16         dtypeStr = 'double';
17     case 2
18         dtypeStr = 'single';
19     case 3
20         dtypeStr = 'int32';
21     case 4
22         dtypeStr = 'uint32';
23     case 5
24         dtypeStr = 'int16';
25     case 6
26         dtypeStr = 'uint16';
27     case 7
28         dtypeStr = 'int8';
29     case 8
30         dtypeStr = 'uint8';
31     case 9
32         dtypeStr = 'logical';
33     case 10
34         dtypeStr = 'embedded.fi';
35     end
36 end

```

Listing D.3: Matlab function get element type.

Appendix E

Setup Raspberry Pi

E.1 Configuration Raspberry Pi

The following process is done once the raspberry pi is configured. The commands below, first allow ssh communication. Second expand the filesystem on the SD card. Third, start ssh on boot such that external connectivity is accepted. Fourthly, configure serial communication, such that it can communicate via the GPIO pins. And last, the IP addresses of the raspberry pi are fixed.

```
1 # Startup SSH
2 sudo systemctl enable ssh
3 sudo systemctl start ssh
```

Resizing the SD-card.

```
1 sudo raspi-config
```

Select → 7 *Advanced Options* → A1 *Expand Filesystem* → OK → Finish → Yes.

Startup SSH on boot.

```
1 sudo raspi-config
```

Select → 5 *Interfacing Options* → P2 *SSH* → Yes → Ok → Finish → Yes.

Setup Serial communication.

```
1 sudo raspi-config
```

Select → 5 *Interfacing Options* → P6 *Serial* → No → Yes → Ok → Finish → Yes.

Fix the IP-addresses of the Raspberry pi.

```
1 sudo nano /etc/rc.local
2 sudo ifconfig wlan0 192.168.42.1
3 sudo ifconfig eth0 192.168.178.21
```

E.2 Setup Serial 2 Network

```
1 clear
2 ## This script turns installs and configures ser2net
3 # in order to run this script, run it under sudo
4 # sudo nano Setup_Ser2Net.rc (copy past this file)
5 # sudo chmod +x Setup_Ser2Net.rc
6 # sudo ./Setup_Ser2Net.rc
7
8 # Update stretch and install and Ser2net
9 sudo apt-get update -y
10 sudo apt-get install ser2net -y
11
12 # Configuring USB
13 if grep -Fq "4001:raw:0:/dev/ttyACM0:19200 8DATABITS NONE 1STOPBIT" /etc/ser2net.conf; ...
    then
```

```

14     echo "USB Terminal already configured on ttyACM0 on baudrate 19200 port 4001"
15 else
16     echo "USB Terminal configures at ttyACM0 on baudrate 19200 port 4001"
17     sudo echo "4001:raw:0:/dev/ttyACM0:19200 8DATABITS NONE 1STOPBIT" >> /etc/ser2net.conf
18 fi
19
20 # Configuring GPIO pins
21 if grep -Fq "4002:raw:0:/dev/ttyS0:115200 8DATABITS NONE 1STOPBIT" /etc/ser2net.conf; then
22     echo "USB Terminal already configured on ttyS0 on baudrate 115200 port 4002"
23 else
24     echo "USB Terminal configured at ttyS0 on baudrate 115200 port 4002"
25     sudo echo "4002:raw:0:/dev/ttyS0:115200 8DATABITS NONE 1STOPBIT" >> /etc/ser2net.conf
26 fi

```

Listing E.1: Ser2Net installation script.

Commands for ser2net usage:

Edit configuration file:

```
1 sudo nano /etc/ser2net.conf
```

The configuration is done in the form of:

```
1 <TCP port>:<State>:<Timeout>:<Device>:<Options>
2 4001:raw:0:/dev/ttyACM0:115200 8DATABITS NONE 1STOPBIT
```

Where Options contains information about the serial connection, such as baudrate, number of databits, parity and number of stopbits.

Start service:

```
1 sudo /etc/init.d/ser2net start
```

Stop service:

```
1 sudo /etc/init.d/ser2net stop
```

Restart service:

```
1 sudo /etc/init.d/ser2net restart
```

Check if service is running:

```
1 ps axg | grep [s]er2net
```

E.3 Setup Access Point

```

1 #!/bin/bash
2
3 clear
4
5 ## This script turns a Raspberry pi into an Access Point
6 # in order to run this script, run it under sudo
7 #   sudo nano Setup_AP.rc   (copy past this file)
8 #   sudo chmod +x Setup_AP.rc
9 #   sudo ./Setup_AP.rc
10 # Check if WiFi adapter is available (wlan0)
11 ifconfig -a
12
13 # Update packages and install iptables manager
14 echo ...
    "=====

```

```

15 echo "Updating packages and installing hostapd and iptables manager"
16 echo "Press Yes to all windows in configuration screen"
17 echo ...
18 "-----"
18 sudo apt-get update -y
19 sudo apt-get install hostapd isc-dhcp-server -y
20 sudo apt-get install iptables-persistent -y
21
22 # Set up DHCP server
23 echo ...
24 "-----"
24 echo "Set up DHCP server"
25 echo ...
26 "-----"
26 sudo chown 777 /etc/dhcp/dhcpd.conf
27
28 if grep -Fq "#option domain-name \"example.org\";" /etc/dhcp/dhcpd.conf; then
29     echo "Comment 1 already done in dhcpd.conf."
30 else
31     echo "Comment 1 applied in dhcpd.conf."
32     sudo perl -pi -e 's/option domain-name "example.org";/#option domain-name ...
33         "example.org";/g' /etc/dhcp/dhcpd.conf
34 fi
34 if grep -Fq "#option domain-name-servers ns1.example.org, ns2.example.org;" ...
35     /etc/dhcp/dhcpd.conf; then
36     echo "Comment 2 already done in dhcpd.conf."
37 else
38     echo "Comment 2 applied in dhcpd.conf."
39     sudo perl -pi -e 's/option domain-name-servers ns1.example.org, ...
40         ns2.example.org;/#option domain-name-servers ns1.example.org, ...
41         ns2.example.org;/g' /etc/dhcp/dhcpd.conf
42 fi
40 if grep -Fq "#authoritative;" /etc/dhcp/dhcpd.conf; then
41     echo "Comment 3 removed in dhcpd.conf."
42     sudo perl -pi -e 's/#authoritative;/authoritative;/g' /etc/dhcp/dhcpd.conf
43 else
44     echo "Comment 3 already removed in dhcpd.conf."
45 fi
46
47 # add subnet 192.168.42.0 netmask 255.255.255.0 {
48 if grep -Fqx "subnet 192.168.42.0 netmask 255.255.255.0 {" /etc/dhcp/dhcpd.conf; then
49     echo "Line 1 already exists in dhcpd.conf."
50 else
51     echo "Line 1 added to dhcpd.conf."
52     sudo echo "subnet 192.168.42.0 netmask 255.255.255.0 {" >> /etc/dhcp/dhcpd.conf
53 fi
54 # add range 192.168.42.10 192.168.42.15;
55 if grep -Fqx "range 192.168.42.10 192.168.42.15;" /etc/dhcp/dhcpd.conf; then
56     echo "Line 2 already exists in dhcpd.conf."
57 else
58     echo "Line 2 added to dhcpd.conf."
59     sudo echo "range 192.168.42.10 192.168.42.15;" >> /etc/dhcp/dhcpd.conf
60 fi
61 # add option broadcast-address 192.168.42.255;
62 if grep -Fqx "option broadcast-address 192.168.42.255;" /etc/dhcp/dhcpd.conf; then
63     echo "Line 3 already exists in dhcpd.conf."
64 else
65     echo "Line 3 added to dhcpd.conf."
66     sudo echo "option broadcast-address 192.168.42.255;" >> /etc/dhcp/dhcpd.conf
67 fi
68 # add option routers 192.168.42.1;
69 # add default-lease-time 600;
70 # add max-lease-time 7200;
71 if grep -Fqx "option routers 192.168.42.1;" /etc/dhcp/dhcpd.conf; then
72     echo "Line 4 already exists in dhcpd.conf."
73     echo "Line 5 already exists in dhcpd.conf."
74     echo "Line 6 already exists in dhcpd.conf."
75 else
76     echo "Line 4 added to dhcpd.conf."
77     sudo echo "option routers 192.168.42.1;" >> /etc/dhcp/dhcpd.conf
78     echo "Line 5 added to dhcpd.conf."
79     sudo echo "default-lease-time 600;" >> /etc/dhcp/dhcpd.conf
80     echo "Line 6 added to dhcpd.conf."

```

```

81     sudo echo "max-lease-time 7200;" >> /etc/dhcp/dhcpd.conf
82 fi
83
84 # add option domain-name "local";
85 if grep -Fxq "option domain-name \"local\";" /etc/dhcp/dhcpd.conf; then
86     echo "Line 7 already exists in dhcpd.conf."
87 else
88     echo "Line 7 added to dhcpd.conf."
89     sudo echo "option domain-name \"local\";" >> /etc/dhcp/dhcpd.conf
90 fi
91 # add option domain-name-servers 8.8.8.8, 8.8.4.4;}
92 if grep -Fxq "option domain-name-servers 8.8.8.8, 8.8.4.4;}" /etc/dhcp/dhcpd.conf; then
93     echo "Line 8 already exists in dhcpd.conf."
94 else
95     echo "Line 8 added to dhcpd.conf."
96     sudo echo "option domain-name-servers 8.8.8.8, 8.8.4.4;}" >> /etc/dhcp/dhcpd.conf
97 fi
98
99 sudo chown 777 /etc/default/isc-dhcp-server
100 perl -pi -e 's/INTERFACES="/INTERFACES="wlan0"/g' /etc/default/isc-dhcp-server
101 perl -pi -e 's/INTERFACESv4="/INTERFACESv4="wlan0"/g' /etc/default/isc-dhcp-server
102 perl -pi -e 's/INTERFACESv6="/INTERFACESv6="wlan0"/g' /etc/default/isc-dhcp-server
103
104 # Set up wlan0 for static IP
105 echo ...
106     "=====
107     "Set up wlan0 for static IP
108     "=====
109 sudo ifdown wlan0
110 sudo chown 777 /etc/network/interfaces
111
112 if grep -xq "auto wlan0" /etc/network/interfaces; then
113     # wlan0 is present in interfaces
114     echo "auto wlan0 is present in interfaces and commented"
115     STARTLINE=$(grep -n 'auto wlan0' /etc/network/interfaces | cut -d : -f 1)
116     ENDLINE=$(cat /etc/network/interfaces | wc -l)
117     sed -i "$STARTLINE,$ENDLINE s/^#/ /" /etc/network/interfaces
118 else
119     # wlan0 is not present in interfaces
120     echo "auto wlan0 is not present in interfaces"
121 fi
122
123 if grep -Fxq "iface wlan0 inet static" /etc/network/interfaces; then
124     echo "Line 1 is present in interfaces."
125 else
126     echo "Line 1 is added in interfaces."
127     echo "iface wlan0 inet static" >> /etc/network/interfaces
128 fi
129
130 if grep -Fxq "address 192.168.42.1" /etc/network/interfaces; then
131     echo "Line 2 is present in interfaces."
132 else
133     echo "Line 2 is added in interfaces."
134     echo "address 192.168.42.1" >> /etc/network/interfaces
135 fi
136
137 if grep -Fxq "netmask 255.255.255.0" /etc/network/interfaces; then
138     echo "Line 3 is present in interfaces."
139 else
140     echo "Line 3 is added in interfaces."
141     echo "netmask 255.255.255.0" >> /etc/network/interfaces
142 fi
143
144 # Assigning a static IP address to the WiFi adapter
145 sudo ifconfig wlan0 192.168.42.1
146
147 # Configure Access Point
148 echo ...
149     "=====
150     "Configure Access Point
151     "=====
152
153 if [ -f /etc/hostapd/hostapd.conf ]; then

```

```

150     echo "File hostapd.conf found and deleted"
151     rm -rf /etc/hostapd/hostapd.conf
152 else
153     echo "File hostapd.conf does not exist"
154 fi
155
156 sudo touch /etc/hostapd/hostapd.conf
157 sudo chown 777 /etc/hostapd/hostapd.conf
158 echo "interface=wlan0" >> /etc/hostapd/hostapd.conf
159 echo "#driver=rtl871xdrv" >> /etc/hostapd/hostapd.conf
160 echo "ssid=TBS" >> /etc/hostapd/hostapd.conf
161 echo "country_code=US" >> /etc/hostapd/hostapd.conf
162 echo "hw_mode=g" >> /etc/hostapd/hostapd.conf
163 echo "channel=6" >> /etc/hostapd/hostapd.conf
164 echo "macaddr_acl=0" >> /etc/hostapd/hostapd.conf
165 echo "auth_algs=1" >> /etc/hostapd/hostapd.conf
166 echo "ignore_broadcast_ssid=0" >> /etc/hostapd/hostapd.conf
167 echo "wpa=2" >> /etc/hostapd/hostapd.conf
168 echo "wpa_passphrase=TBSRaspberry" >> /etc/hostapd/hostapd.conf
169 echo "wpa_key_mgmt=WPA-PSK" >> /etc/hostapd/hostapd.conf
170 echo "wpa_pairwise=CCMP" >> /etc/hostapd/hostapd.conf
171 echo "wpa_group_rekey=86400" >> /etc/hostapd/hostapd.conf
172 echo "ieee80211n=1" >> /etc/hostapd/hostapd.conf
173 echo "wme_enabled=1" >> /etc/hostapd/hostapd.conf
174
175
176 sudo perl -pi -e 's/#DAEMON_CONF=""/DAEMON_CONF="/etc/hostapd/hostapd.conf/g' ...
    /etc/default/hostapd
177 if grep -Fxq "DAEMON_CONF=/etc/hostapd/hostapd.conf" /etc/init.d/hostapd; then
178     echo "Reference already exists in init.d/hostapd"
179 else
180     echo "Reference made in init.d/hostapd"
181     sudo perl -pi -e 's/DAEMON_CONF=/DAEMON_CONF="/etc/hostapd/hostapd.conf/g' ...
        /etc/init.d/hostapd
182 fi
183
184 # Configure Network Address Translation
185 echo ...
186 echo "Configure Network Address Translation"
187 echo ...
188
189 if grep -Fxq "net.ipv4.ip_forward=1" /etc/sysctl.conf; then
190     echo "Line already exists in sysctl.conf"
191 else
192     echo "Line added to sysctl.conf"
193     echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
194 fi
195 sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
196
197 sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
198 sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
199 sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
200
201 # check whats inside the iptables
202 echo "Check whats inside the iptables"
203 sudo iptables -t nat -S
204 sudo iptables -S
205
206 sudo sh -c "iptables-save > /etc/iptables/rules.v4"
207
208 # Test kernel version for addition steps
209 kernel_version=`uname -r`
210 if [ $kernel_version > "4.4" ]; then
211     echo "Kernel version is above 4.5."
212     echo "No additional steps are needed."
213 else
214     echo "Kernel version might be below 4.4.13-v7+."
215     echo "Take additional steps."
216     echo "See: ..."
        https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/install-software"
217 read -n 1 -s

```

```
218 fi
219 rm -rf 4.4
220
221 # First test
222 echo ...
223 echo "===== "
224 echo "FIRST TIME ACCESS POINT IS TESTED"
225 echo ...
226 echo "===== "
227 #sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
228 # Removing WPA-Supplicant
229 echo ...
230 echo "===== "
231 echo "Removing WPA-Supplicant "
232 echo ...
233 echo "===== "
234 # sudo mv /usr/share/dbus-1/system-services/fi.epitest.hostap.WPASupplicant.service -/
235 # sudo reboot
```

Listing E.2: Raspberry Pi Access Point configuration script.

Appendix F

Com0Com

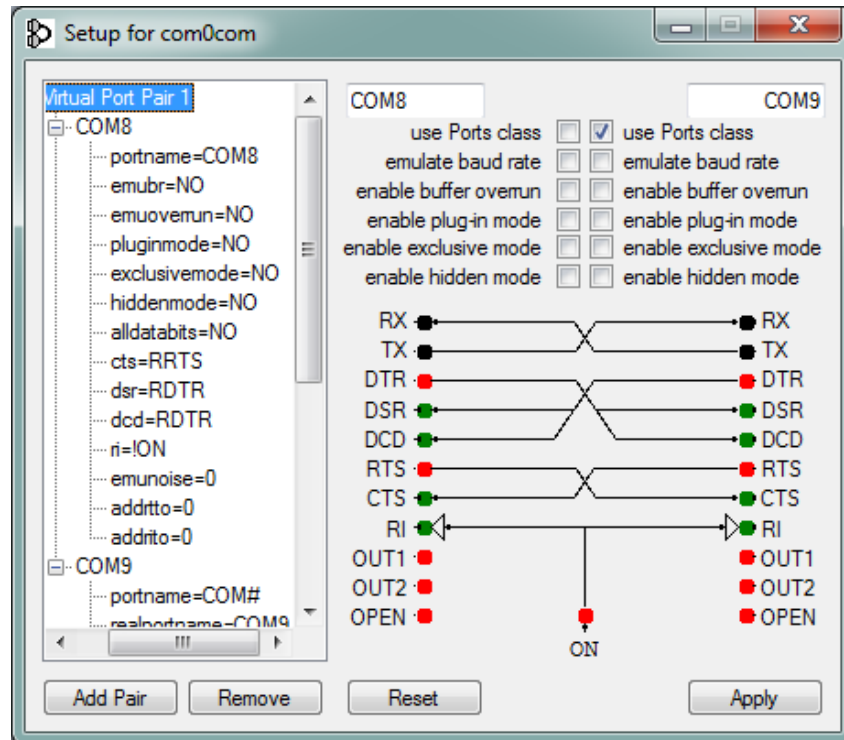


Figure F.1: Overview of the com0com software. In this figure, a simple null-modem configuration is also shown.

Appendix G

CombyTCP

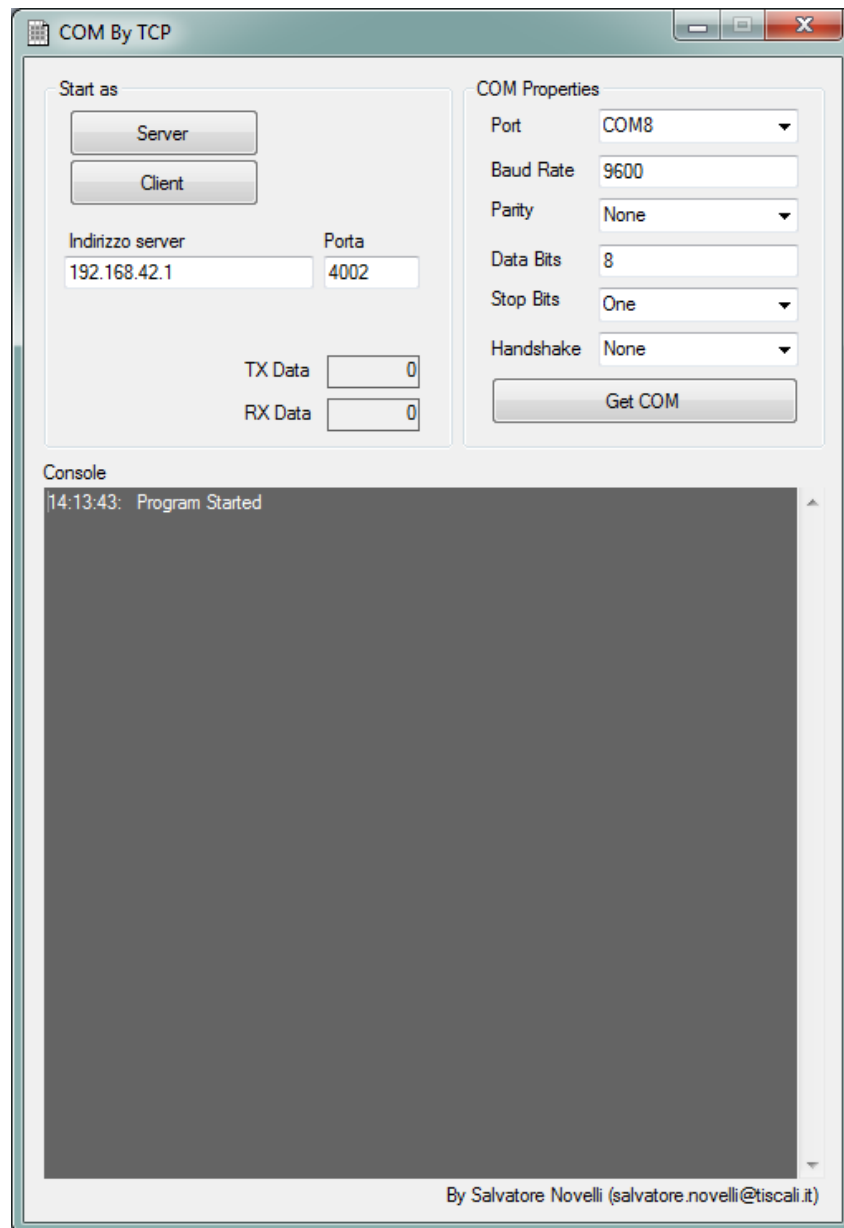


Figure G.1: Overview of the combyTCP software. In this figure, on the server side the raspberry pi settings are shown. On the Com properties side, a configuration for Com communication setup as in ser2net is shown.

Appendix H

GPS Driver Installation

This is a short description of the MarvelMind GPS driver software installation. First of all, a few notes to keep in mind:

- This driver is not suited for commercial use.
- This driver is only designed to accept Marvelmind communication messages.
- Which means that no configuration of the GPS is performed.
- It disables the Ashtech GPS driver and reads the general \$GP GPS protocol messages and posts them to the uORB.

The software for the Pixhawk is located in `../px4`. In most cases located on the windows C-drive. The software located in this directory, is the complete flight stack, which is loaded to the Pixhawk from Simulink. In this software, the driver needs to be installed, to cope with the Marvelmind messages. The first step would be to place the files `marv.h` (section H.1) and `marv.cpp` (section H.2) in the `../px4/Firmware/src/drivers/gps/devices/src` folder.

The driver files need to be included in the compilation process, which is done via CMakeLists file located in `../px4/Firmware/src/drivers/gps`. Below is the new C make file shown where line 46 is added, such that the Marvelmind driver is included in the compilation process.

```
1 #####
2 #
3 #   Copyright (c) 2015 PX4 Development Team. All rights reserved.
4 #
5 # Redistribution and use in source and binary forms, with or without
6 # modification, are permitted provided that the following conditions
7 # are met:
8 #
9 # 1. Redistributions of source code must retain the above copyright
10 # notice, this list of conditions and the following disclaimer.
11 # 2. Redistributions in binary form must reproduce the above copyright
12 # notice, this list of conditions and the following disclaimer in
13 # the documentation and/or other materials provided with the
14 # distribution.
15 # 3. Neither the name PX4 nor the names of its contributors may be
16 # used to endorse or promote products derived from this software
17 # without specific prior written permission.
18 #
19 # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
20 # "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
21 # LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
22 # FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
23 # COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
24 # INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
25 # BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
26 # OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
27 # AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
28 # LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
29 # ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 # POSSIBILITY OF SUCH DAMAGE.
31 #
32 #####
33
34 px4_add_git_submodule(TARGET git_gps_devices PATH "devices")
35
36 px4_add_module(
37     MODULE drivers__gps
38     MAIN gps
```

```

39     STACK_MAIN 1200
40     SRCS
41         gps.cpp
42         devices/src/gps_helper.cpp
43         devices/src/mtk.cpp
44         devices/src/ashtech.cpp
45         devices/src/ubx.cpp
46         devices/src/marv.cpp
47     DEPENDS
48         platforms__common
49         git_gps_devices
50 )

```

Listing H.1: C make file to compile the GPS driver.

Furthermore, the driver header file (`drv_gps.h`) located in `../px4/Firmware/src/drivers` contains definitions of the available drivers. The Marvelmind needs to be included as well in this definition. The added Marvelmind gps definition is shown on line 59 of the header code below.

```

1  /*****
2  *
3  *   Copyright (c) 2013–2017 PX4 Development Team. All rights reserved.
4  *
5  *   Redistribution and use in source and binary forms, with or without
6  *   modification, are permitted provided that the following conditions
7  *   are met:
8  *
9  *   1. Redistributions of source code must retain the above copyright
10 *   notice, this list of conditions and the following disclaimer.
11 *   2. Redistributions in binary form must reproduce the above copyright
12 *   notice, this list of conditions and the following disclaimer in
13 *   the documentation and/or other materials provided with the
14 *   distribution.
15 *   3. Neither the name PX4 nor the names of its contributors may be
16 *   used to endorse or promote products derived from this software
17 *   without specific prior written permission.
18 *
19 *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
20 *   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
21 *   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
22 *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
23 *   COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
24 *   INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
25 *   BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
26 *   OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
27 *   AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
28 *   LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
29 *   ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 *   POSSIBILITY OF SUCH DAMAGE.
31 *
32  *****/
33
34 /**
35  * @file drv_gps.h
36  *
37  * GPS driver interface
38  */
39
40 #pragma once
41
42 #include <stdint.h>
43 #include <sys/ioctl.h>
44
45 #include "board_config.h"
46
47 #include "drv_sensor.h"
48 #include "drv_orb_dev.h"
49
50 #ifndef GPS_DEFAULT_UART_PORT
51 #define GPS_DEFAULT_UART_PORT "/dev/ttyS3"
52 #endif
53

```

```

54 typedef enum {
55     GPS_DRIVER_MODE_NONE = 0,
56     GPS_DRIVER_MODE_UBX,
57     GPS_DRIVER_MODE_MTK,
58     GPS_DRIVER_MODE_ASHTECH,
59     GPS_DRIVER_MODE_MARVELMIND
60 } gps_driver_mode_t;

```

Listing H.2: GPS driver type definition declared in the header file.

A lot of changes needs to be made to the file `gps.cpp` located in `../px4/Firmware/src/drivers/gps`. This will be explained in section H.3.

H.1 Driver Header file

Save the following code as `marv.h` in the `../px4/Firmware/src/drivers/gps/devices/src` directory.

```

1  /*****
2  *
3  *   Copyright (C) 2013. All rights reserved.
4  *   Author: Boriskin Aleksey <a.d.boriskin@gmail.com>
5  *           Kistanov Alexander <akistanov@gramant.ru>
6  *
7  *   Redistribution and use in source and binary forms, with or without
8  *   modification, are permitted provided that the following conditions
9  *   are met:
10 *
11 *   1. Redistributions of source code must retain the above copyright
12 *      notice, this list of conditions and the following disclaimer.
13 *   2. Redistributions in binary form must reproduce the above copyright
14 *      notice, this list of conditions and the following disclaimer in
15 *      the documentation and/or other materials provided with the
16 *      distribution.
17 *   3. Neither the name PX4 nor the names of its contributors may be
18 *      used to endorse or promote products derived from this software
19 *      without specific prior written permission.
20 *
21 *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
22 *   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
23 *   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
24 *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
25 *   COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
26 *   INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
27 *   BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
28 *   OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
29 *   AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
30 *   LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
31 *   ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
32 *   POSSIBILITY OF SUCH DAMAGE.
33 *
34  *****/
35
36 /* @file MARVELMIND protocol definitions */
37
38 #pragma once
39
40 #include "gps_helper.h"
41 #include "../definitions.h"
42
43 #define MARVELMIND_RECV_BUFFER_SIZE 82
44
45 class GPSDriverMarvelmind : public GPSHelper
46 {
47 public:
48     GPSDriverMarvelmind(GPSCallbackPtr callback, void *callback.user,
49                         struct vehicle_gps_position_s *gps_position,
50                         struct satellite_info_s *satellite_info);
51     virtual ~GPSDriverMarvelmind() = default;
52
53     int receive(unsigned timeout);

```

```

54     int configure(unsigned &baud, OutputMode output_mode);
55     int parseChar(uint8_t b);
56
57
58 private:
59     void decodeInit(void);
60     int handleMessage(int len);
61     int receive_config(unsigned timeout);
62     int parseChar_config(uint8_t b);
63
64
65     /** Read int MARVELMIND parameter */
66     int32_t read_int();
67     /** Read float MARVELMIND parameter */
68     double read_float();
69     /** Read char MARVELMIND parameter */
70     char read_char();
71
72     enum marvelmind_decode_state_t {
73         NME_DECODE_UNINIT,
74         NME_DECODE_GOT_SYNC1,
75         NME_DECODE_GOT_ASTRIKS,
76         NME_DECODE_GOT_FIRST_CS_BYTE
77     };
78
79     struct satellite_info_s *_satellite_info {nullptr};
80     struct vehicle_gps_position_s *_gps_position {nullptr};
81     uint64_t _last_timestamp{0};
82     int _marvelmindlog_fd{-1};
83
84     marvelmind_decode_state_t _decode_state{NME_DECODE_UNINIT};
85     marvelmind_decode_state_t _decode_state_config{NME_DECODE_UNINIT};
86     uint8_t _rx_buffer[MARVELMIND_RECV_BUFFER_SIZE] {};
87     uint16_t _rx_buffer_bytes{};
88     bool _got_pashr_pos_message{false}; /**< If we got a PASHR,POS message we will ...
89         ignore GGA messages */
90     bool _parse_error{}; /**< parse error flag */
91     char *_parse_pos{}; /**< parse position */
92 };

```

Listing H.3: Marvelmind driver header file.

H.2 Driver file

Save the following code as `marv.cpp` in the `../px4/Firmware/src/drivers/gps/devices/src` directory.

```

1  /*****
2  *
3  *   Copyright (c) 2012–2016 PX4 Development Team. All rights reserved.
4  *
5  *   Redistribution and use in source and binary forms, with or without
6  *   modification, are permitted provided that the following conditions
7  *   are met:
8  *
9  *   1. Redistributions of source code must retain the above copyright
10 *       notice, this list of conditions and the following disclaimer.
11 *   2. Redistributions in binary form must reproduce the above copyright
12 *       notice, this list of conditions and the following disclaimer in
13 *       the documentation and/or other materials provided with the
14 *       distribution.
15 *   3. Neither the name PX4 nor the names of its contributors may be
16 *       used to endorse or promote products derived from this software
17 *       without specific prior written permission.
18 *
19 *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
20 *   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
21 *   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
22 *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
23 *   COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
24 *   INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,

```

```

25 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
26 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
27 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
28 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
29 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 * POSSIBILITY OF SUCH DAMAGE.
31 *
32 *****/
33
34 #include <stdlib.h>
35 #include <stdio.h>
36 #include <math.h>
37 #include <string.h>
38 #include <ctime>
39
40 #include "marv.h"
41
42 GPSDriverMarvelmind::GPSDriverMarvelmind(GPSCallbackPtr callback, void *callback_user,
43     struct vehicle_gps_position_s *gps_position,
44     struct satellite_info_s *satellite_info) :
45     GPShelper(callback, callback_user),
46     _satellite_info(satellite_info),
47     _gps_position(gps_position)
48 {
49     decodeInit();
50     _decode_state = NME_DECODE_UNINIT;
51     _decode_state_config = NME_DECODE_UNINIT;
52     _rx_buffer_bytes = 0;
53     #define TIMEOUT_5HZ 500
54 }
55
56 /*
57 * All NMEA descriptions are taken from
58 * http://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_MessageOverview.html
59 */
60
61 int GPSDriverMarvelmind::handleMessage(int len)
62 {
63     char *endp;
64
65     if(len < 7){
66         return 0;
67     }
68
69     int uiCalcComma = 0;
70
71     for (int i = 0; i < len; i++){
72         if (_rx_buffer[i] == ',') {uiCalcComma++;}
73     }
74
75     char *bufptr = (char *) (_rx_buffer + 6);
76     int ret = 0;
77
78     //PX4_INFO("Msg: %s", _rx_buffer);
79
80     if ((memcmp(_rx_buffer + 3, "RMC", 3) == 0)) {
81         /*
82          * An example of the GSV message string is:
83
84          * $GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A
85
86          * GPRMC message field
87          * Field Meaning
88          * 0 Message ID $GPRMC
89          * 1 UTC of position fix
90          * 2 Status:
91          * A: active
92          * V: void
93          * 3 Latitude
94          * 4 Longitude
95          * 5 Speed over the ground in knots
96          * 6 Track angle in degrees (True)
97          * 7 Date

```

```

98     8      Magnetic variation in degrees
99     The checksum data, always begins with *
100    */
101
102    double marvelmind_time __attribute__((unused)) = 0.0, lat = 0.0, lon = 0.0, ...
        speed_n = 0.0, magvar = 0.0, track = 0.0;
103    int date __attribute__((unused)) = 0;
104    char ns = '?', ew = '?', ewmag = '?', messtat = '?';
105
106    if (bufptr && *(++bufptr) != ',') { marvelmind_time = strtod(bufptr, ...
        &endp); bufptr = endp; }
107
108    if (bufptr && *(++bufptr) != ',') { messtat = *(bufptr++); }
109
110    if (bufptr && *(++bufptr) != ',') { lat = strtod(bufptr, &endp); bufptr = ...
        endp; }
111
112    if (bufptr && *(++bufptr) != ',') { ns = *(bufptr++); }
113
114    if (bufptr && *(++bufptr) != ',') { lon = strtod(bufptr, &endp); bufptr = ...
        endp; }
115
116    if (bufptr && *(++bufptr) != ',') { ew = *(bufptr++); }
117
118    if (bufptr && *(++bufptr) != ',') { speed_n = strtod(bufptr, &endp); bufptr ...
        = endp; }
119
120    if (bufptr && *(++bufptr) != ',') { track = strtod(bufptr, &endp); bufptr = ...
        endp; }
121
122    if (bufptr && *(++bufptr) != ',') { date = strtol(bufptr, &endp, 10); ...
        bufptr = endp; }
123
124    if (bufptr && *(++bufptr) != ',') { magvar = strtod(bufptr, &endp); bufptr ...
        = endp; }
125
126    if (bufptr && *(++bufptr) != ',') { ewmag = *(bufptr++); }
127
128
129    if (ns == 'S') {
130        lat = -lat;
131    }
132
133    if (ew == 'W') {
134        lon = -lon;
135    }
136
137    if (ewmag == 'W') {
138        magvar = -magvar;
139    }
140
141    if (messtat == 'W') {
142        magvar = magvar;
143    }
144
145
146
147    /* convert from degrees, minutes and seconds, to degrees * 1e7 */
148    _gps_position->lat = static_cast<int>((int)(lat * 0.01) + (lat * 0.01 - ...
        int(lat * 0.01)) * 100.0 / 60.0) * 10000000);
149    _gps_position->lon = static_cast<int>((int)(lon * 0.01) + (lon * 0.01 - ...
        int(lon * 0.01)) * 100.0 / 60.0) * 10000000);
150    _rate_count.lat_lon++;
151    _gps_position->vel_m_s = speed_n * 1.85;
152    _gps_position->cog_rad = static_cast<int>(track * 2 * M_PI / 360.0 * 1e-5);
153    //_gps_position->cog_rad = static_cast<int>(track * double(3.0*M_PI/360.0 * ...
        1e-5f));
154
155
156    /*
157     * Do something smart with:
158     * track
159     * date

```

```

160     * magvar
161     */
162     .gps_position->timestamp_time = gps_absolute_time();
163     ret = 1;
164
165
166 } else if ((memcmp(_rx_buffer + 3, "GGA,", 3) == 0) && (uiCalcComma == 14)){
167     /*
168     Time, position, and fix related data
169     An example of the GBS message string is:
170
171     $GPGGA,172814.0,3723.46587704,N,12202.26957864,W,2,6,1.2,18.893,M-25.669,M,2.0,0031*4F
172
173     Note - The data string exceeds the Marvelmind standard length.
174     GGA message fields
175     Field  Meanings
176     0      Message ID $GPGGA
177     1      UTC of position fix
178     2      Latitude
179     3      Direction of Latitude:
180             N: North
181             S: South
182     4      Longitude
183     5      Direction of longitude:
184             E: East
185             W: West
186     6      GPS Quality indicator:
187             0: Fix not valid
188             1: GPS fix
189             2: Differential GPS fix, PmniSTAR VBS
190             4: Real-Time Kinematic, fixed integers
191             5: Real-Time Kinematic, float integers, OmniSTAR XP/HP or localation RTK
192     7      Number of SVs in use, range from 00 through to 24+
193     8      HDOP
194     9      Orthometric height (MSL reference)
195     10     M: unit of measure for orthometric height is meters
196     11     Geoid separation
197     12     M: geoid separation measured in meters
198     13     Age of differential GPS data record, Type 1 or Type 9. Null field when ...
199             DGPS is not used.
200     14     Reference station ID, range 0000-4095. A null field when any reference ...
201             station ID is selected and no corrections are received.
202
203     15
204     The checksum data, always begins with *
205     Note - If a user-defined geoid model, or an inclined
206     */
207
208     double marvelmind_time __attribute__((unused)) = 0.0, lat = 0.0, lon = 0.0, alt = ...
209             0.0, alt_ellipsoid = 0.0;
210     int num_of_sv __attribute__((unused)) = 0, fix_quality = 0;
211     double hdop __attribute__((unused)) = 99.9;
212     char ns = '?', ew = '?', M = '?';
213
214     if (bufptr && *(++bufptr) != ',') { marvelmind_time = strtod(bufptr, &endp); ...
215             bufptr = endp; }
216
217     if (bufptr && *(++bufptr) != ',') { lat = strtod(bufptr, &endp); bufptr = endp; }
218
219     if (bufptr && *(++bufptr) != ',') { ns = *(bufptr++); }
220
221     if (bufptr && *(++bufptr) != ',') { lon = strtod(bufptr, &endp); bufptr = endp; }
222
223     if (bufptr && *(++bufptr) != ',') { ew = *(bufptr++); }
224
225     if (bufptr && *(++bufptr) != ',') { fix_quality = strtol(bufptr, &endp, 10); ...
226             bufptr = endp; }
227
228     if (bufptr && *(++bufptr) != ',') { num_of_sv = strtol(bufptr, &endp, 10); bufptr ...
229             = endp; }
230
231     if (bufptr && *(++bufptr) != ',') { hdop = strtod(bufptr, &endp); bufptr = endp; }
232
233     if (bufptr && *(++bufptr) != ',') { alt = strtod(bufptr, &endp); bufptr = endp; }

```

```

227
228 if (bufptr && *(++bufptr) != ',') { M = *(bufptr++); }
229
230 if (bufptr && *(++bufptr) != ',') { alt_ellipsoid = strtod(bufptr, &endp); bufptr ...
    = endp; }
231
232 if (bufptr && *(++bufptr) != ',') { M = *(bufptr++); }
233
234 if (ns == 'S') {
235     lat = -lat;
236 }
237
238 if (ew == 'W') {
239     lon = -lon;
240 }
241
242 if (M == 'M') {
243     alt = alt;
244 }
245
246 /* convert from degrees, minutes and seconds, to degrees * 1e7 */
247 _gps_position->lat = static_cast<int>((int(lat * 0.01) + (lat * 0.01 - int(lat * ...
    0.01)) * 100.0 / 60.0) * 10000000);
248 _gps_position->lon = static_cast<int>((int(lon * 0.01) + (lon * 0.01 - int(lon * ...
    0.01)) * 100.0 / 60.0) * 10000000);
249 _gps_position->alt = static_cast<int>(alt * 1000);
250 _rate_count_lat_lon++;
251
252 if (fix_quality <= 0) {
253     _gps_position->fix_type = 0;
254 } else {
255     /*
256      * in this NMEA message float integers (value 5) mode has higher value than ...
257      * fixed integers (value 4), whereas it provides lower quality
258      * and since value 3 is not being used, I "moved" value 5 to 3 add it to ...
259      * _gps_position->fix_type
260      */
261     if (fix_quality == 5) { fix_quality = 3; }
262
263     /*
264      * fix quality 1 means just a normal 3D fix, so I'm subtracting 1 here. This ...
265      * way we'll have 3 for auto, 4 for DGPS, 5 for floats, 6 for fixed.
266      */
267     _gps_position->fix_type = 3 + fix_quality - 1;
268 }
269
270 _gps_position->hdop = hdop;
271 _gps_position->satellites_used = num_of_sv;
272 _gps_position->alt_ellipsoid = alt_ellipsoid;
273 _gps_position->vel_m_s = 0;          /**< GPS ground speed (m/s) */
274 _gps_position->vel_n_m_s = 0;       /**< GPS ground speed in m/s */
275 _gps_position->vel_e_m_s = 0;       /**< GPS ground speed in m/s */
276 _gps_position->vel_d_m_s = 0;       /**< GPS ground speed in m/s */
277 _gps_position->cog_rad = 0;          /**< Course over ground (NOT heading, but ...
    direction of movement) in rad, -PI..PI */
278 _gps_position->vel_ned_valid = true; /**< Flag to indicate if NED speed is ...
    valid */
279 _gps_position->c_variance_rad = 0.1f;
280
281 _gps_position->timestamp_time = gps_absolute_time();
282 ret = 1;
283
284 } else if ((memcmp(_rx_buffer + 3, "VTG,", 3) == 0)){
285     /*
286      * Track made good and ground speed. An example of the GSV message string is:
287      *
288      * $GPVTG,360.0,T,348.7,M,000.0,N,000.0,K*34
289      *
290      * VTG message fields
291      * Field Meaning
292      * 0 Message ID $GPVTG
293      * 1 Track made good (degrees true)
294      * 2 T: track made good is relative to true north

```



```

292     3     Track made good (degrees magnetic)
293     4     M: track made good is relative to magnetic north
294     5     Speed in knots
295     6     N: Speed is measured in knots
296     7     Speed over ground in kilometers/hour (kph)
297     8     K: speed over ground is measured in kph
298     The checksum data, always begins with *
299     */
300
301     /*
302     double speed_n __attribute__((unused)) = 0.0, speed_k = 0.0, track_made_good_t ...
        = 0.0, track_made_good_m = 0.0;
303     char K = '?', N = '?', M = '?', T = '?';
304
305     if (bufptr && *(++bufptr) != ',') { track_made_good_t = strtod(bufptr, &endp); ...
        bufptr = endp; }
306
307     if (bufptr && *(++bufptr) != ',') { T = *(bufptr++); }
308
309     if (bufptr && *(++bufptr) != ',') { track_made_good_m = strtod(bufptr, &endp); ...
        bufptr = endp; }
310
311     if (bufptr && *(++bufptr) != ',') { M = *(bufptr++); }
312
313     if (bufptr && *(++bufptr) != ',') { speed_n = strtod(bufptr, &endp); bufptr = ...
        endp; }
314
315     if (bufptr && *(++bufptr) != ',') { N = *(bufptr++); }
316
317     if (bufptr && *(++bufptr) != ',') { speed_k = strtod(bufptr, &endp); bufptr = ...
        endp; }
318
319     if (bufptr && *(++bufptr) != ',') { K = *(bufptr++); }
320     */
321
322     /*
323     * Do something smart with this data
324     */
325     _gps_position->timestamp_time = gps_absolute_time();
326     ret = 1;
327
328 } else if ((memcmp(_rx_buffer + 3, "ZDA,", 3) == 0) && (uiCalcComma == 6)){
329
330     /*
331     UTC day, month, and year, and local time zone offset
332     An example of the ZDA message string is:
333
334     $GPZDA,172809.456,12,07,1996,00,00+45
335
336     ZDA message fields
337     Field Meaning
338     0 Message ID $GPZDA
339     1 UTC
340     2 Day, ranging between 01 and 31
341     3 Month, ranging between 01 and 12
342     4 Year
343     5 Local time zone offset from GMT, ranging from 00 through 13 hours
344     6 Local time zone offset from GMT, ranging from 00 through 59 minutes
345     7 The checksum data, always begins with *
346     Fields 5 and 6 together yield the total offset. For example, if field 5 is -5 ...
        and field 6 is +15, local time is 5 hours and 15 minutes earlier than GMT.
347     */
348
349     double marvelmind_time = 0.0;
350     int day = 0, month = 0, year = 0, local_time_off_hour __attribute__((unused)) ...
        = 0,
351         local_time_off_min __attribute__((unused)) = 0;
352
353     if (bufptr && *(++bufptr) != ',') { marvelmind_time = strtod(bufptr, &endp); ...
        bufptr = endp; }
354
355     if (bufptr && *(++bufptr) != ',') { day = strtol(bufptr, &endp, 10); bufptr = ...
        endp; }

```

```

356
357     if (bufptr && *(++bufptr) != ',') { month = strtol(bufptr, &endp, 10); bufptr = ...
        = endp; }
358
359     if (bufptr && *(++bufptr) != ',') { year = strtol(bufptr, &endp, 10); bufptr = ...
        endp; }
360
361     if (bufptr && *(++bufptr) != ',') { local_time_off.hour = strtol(bufptr, ...
        &endp, 10); bufptr = endp; }
362
363     if (bufptr && *(++bufptr) != ',') { local_time_off.min = strtol(bufptr, &endp, ...
        10); bufptr = endp; }
364
365     int marvelmind.hour = static_cast<int>(marvelmindtime / 10000);
366     int marvelmind.minute = static_cast<int>((marvelmindtime - marvelmind.hour * ...
        10000) / 100);
367     double marvelmind.sec = static_cast<double>(marvelmindtime - marvelmind.hour ...
        * 10000 - marvelmind.minute * 100);
368
369     /*
370     * Convert to unix timestamp
371     */
372     struct tm timeinfo = {};
373     timeinfo.tm_year = year - 1900;
374     timeinfo.tm_mon = month - 1;
375     timeinfo.tm_mday = day;
376     timeinfo.tm_hour = marvelmind.hour;
377     timeinfo.tm_min = marvelmind.minute;
378     timeinfo.tm_sec = int(marvelmind.sec);
379     timeinfo.tm_isdst = 0;
380
381 #ifndef NO_MKTIME
382     time_t epoch = mktime(&timeinfo);
383
384     if (epoch > GPS_EPOCH_SECS) {
385         uint64_t usecs = static_cast<uint64_t>((marvelmind.sec - ...
            static_cast<uint64_t>(marvelmind.sec))) * 1000000;
386
387         // FMUv2+ boards have a hardware RTC, but GPS helps us to configure it
388         // and control its drift. Since we rely on the HRT for our monotonic
389         // clock, updating it from time to time is safe.
390
391         timespec ts{};
392         ts.tv_sec = epoch;
393         ts.tv_nsec = usecs * 1000;
394
395         setClock(ts);
396
397         _gps.position->time_utc_usec = static_cast<uint64_t>(epoch) * 1000000ULL;
398         _gps.position->time_utc_usec += usecs;
399
400     } else {
401         _gps.position->time_utc_usec = 0;
402     }
403 #else
404     _gps.position->time_utc_usec = 0;
405 #endif
406
407     _last_timestamp_time = gps_absolute_time();
408     _gps.position->timestamp_time = gps_absolute_time();
409     ret = 1;
410
411 }
412 /*
413 if (ret > 0) {
414     _gps.position->timestamp_time_relative = (int32_t)(_last_timestamp_time - ...
        _gps.position->timestamp_time);
415 }
416 */
417 return ret;
418
419 }
420

```

```

421 int GPSDriverMarvelmind::receive_config(unsigned timeout)
422 {
423     {
424         uint8_t buf[GPS_READ_BUFFER_SIZE];
425
426         /* timeout additional to poll */
427         uint64_t time_started = gps_absolute_time();
428
429         int j = 0;
430         ssize_t bytes_count = 0;
431
432         while (true) {
433
434             /* pass received bytes to the packet decoder */
435             while (j < bytes_count) {
436                 int l = 0;
437
438                 //parseChar_config(buf[j]);
439
440                 if ((l = parseChar_config(buf[j])) > 0) {
441                     /* return to configure during configuration or to the gps driver ...
442                        during normal work
443                        * if a packer has arrived */
444                     return 1;
445                 }
446                 j++;
447             }
448
449             /* everything is read */
450             j = bytes_count = 0;
451
452             /* then poll or read for new data */
453             int ret = read(buf, sizeof(buf), timeout * 2);
454
455             if (ret < 0) {
456                 /* something went wrong when polling */
457                 return -1;
458             } else if (ret == 0) {
459                 /* Timeout while polling or just nothing read if reading, let's
460                    * stay here, and use timeout below. */
461             } else if (ret > 0) {
462                 /* if we have new data from GPS, go handle it */
463                 bytes_count = ret;
464                 //return ret;
465             }
466
467             /* in case we get crap from GPS or time out */
468             if (time_started + timeout * 1000 * 2 < gps_absolute_time()) {
469                 return -1;
470             }
471         }
472     }
473 }
474 }
475
476 int GPSDriverMarvelmind::receive(unsigned timeout)
477 {
478     {
479         uint8_t buf[GPS_READ_BUFFER_SIZE];
480
481         /* timeout additional to poll */
482         uint64_t time_started = gps_absolute_time();
483
484         int j = 0;
485         ssize_t bytes_count = 0;
486
487         while (true) {
488             //
489             uint64_t time_started = gps_absolute_time();
490             /* pass received bytes to the packet decoder */
491             while (j < bytes_count) {
492                 int l = 0;

```

```

493         if ((l = parseChar(buf[j])) > 0) {
494             /* return to configure during configuration or to the gps driver ...
                during normal work
                * if a packer has arrived */
495             if (handleMessage(l) > 0) {
496                 return 1;
497             }
498         }
499     }
500
501     j++;
502 }
503
504 /* everything is read */
505 j = bytes_count = 0;
506
507 /* then poll or read for new data */
508 int ret = read(buf, sizeof(buf), timeout * 2);
509
510 if (ret < 0) {
511     /* something went wrong when polling */
512     return -1;
513 } else if (ret == 0) {
514     /* Timeout while polling or just nothing read if reading, let's
515     * stay here, and use timeout below. */
516 } else if (ret > 0) {
517     /* if we have new data from GPS, go handle it */
518     bytes_count = ret;
519 }
520
521 /* in case we get crap from GPS or time out */
522 if (time_started + timeout * 1000 < gps_absolute_time()) {
523     return -1;
524 }
525
526 }
527 }
528 }
529
530 #define HEXDIGIT_CHAR(d) ((char)((d) + ((d) < 0xA) ? '0' : 'A'-0xA))
531
532 int GPSDriverMarvelmind::parseChar(uint8_t b)
533 {
534     int iRet = 0;
535
536     switch (_decode_state) {
537         /* First, look for sync1 */
538         case NME_DECODE_UNINIT:
539             if (b == '$') {
540                 _decode_state = NME_DECODE_GOT_SYNC1;
541                 _rx_buffer_bytes = 0;
542                 _rx_buffer[_rx_buffer_bytes++] = b;
543             }
544             break;
545
546         case NME_DECODE_GOT_SYNC1:
547             if (b == '$') {
548                 _decode_state = NME_DECODE_GOT_SYNC1;
549                 _rx_buffer_bytes = 0;
550             } else if (b == '*') {
551                 _decode_state = NME_DECODE_GOT_ASTRIKS;
552             }
553
554             if (_rx_buffer_bytes >= (sizeof(_rx_buffer) - 5)) {
555                 _decode_state = NME_DECODE_UNINIT;
556                 _rx_buffer_bytes = 0;
557             } else {
558                 _rx_buffer[_rx_buffer_bytes++] = b;
559             }
560             break;
561
562         case NME_DECODE_GOT_ASTRIKS:
563             _rx_buffer[_rx_buffer_bytes++] = b;
564             _decode_state = NME_DECODE_GOT_FIRST_CS_BYTE;

```

```

565     break;
566
567     case NME_DECODE_GOT_FIRST_CS_BYTE:
568         _rx_buffer[_rx_buffer_bytes++] = b;
569         uint8_t checksum = 0;
570         uint8_t *buffer = _rx_buffer + 1;
571         uint8_t *buffend = _rx_buffer + _rx_buffer_bytes - 3;
572
573         for (; buffer < buffend; buffer++) { checksum ^= *buffer; }
574
575         if ((HEXDIGIT_CHAR(checksum >> 4) == *(_rx_buffer + _rx_buffer_bytes - 2)) &&
576             (HEXDIGIT_CHAR(checksum & 0x0F) == *(_rx_buffer + _rx_buffer_bytes - 1))) {
577             iRet = _rx_buffer_bytes;
578         }
579
580         _decode_state = NME_DECODE_UNINIT;
581         _rx_buffer_bytes = 0;
582         break;
583
584     }
585 }
586
587 return iRet;
588 }
589
590 int GPSDriverMarvelmind::parseChar.config(uint8_t b)
591 {
592     int CS_Confirm = 0;
593
594     switch (_decode_state.config) {
595         /* First, look for sync1 */
596         case NME_DECODE_UNINIT:
597             if (b == '$') {
598                 _decode_state.config = NME_DECODE_GOT_SYNC1;
599                 _rx_buffer_bytes = 0;
600                 _rx_buffer[_rx_buffer_bytes++] = b;
601             }
602             break;
603
604         case NME_DECODE_GOT_SYNC1:
605             if (b == '$') {
606                 _decode_state.config = NME_DECODE_GOT_SYNC1;
607                 _rx_buffer_bytes = 0;
608             } else if (b == '*') {
609                 _decode_state.config = NME_DECODE_GOT_ASTRIKS;
610             }
611
612             if (_rx_buffer_bytes >= (sizeof(_rx_buffer) - 5)) {
613                 _decode_state.config = NME_DECODE_UNINIT;
614                 _rx_buffer_bytes = 0;
615             } else {
616                 _rx_buffer[_rx_buffer_bytes++] = b;
617             }
618             break;
619
620         case NME_DECODE_GOT_ASTRIKS:
621             _rx_buffer[_rx_buffer_bytes++] = b;
622             _decode_state.config = NME_DECODE_GOT_FIRST_CS_BYTE;
623             break;
624
625         case NME_DECODE_GOT_FIRST_CS_BYTE:
626             _rx_buffer[_rx_buffer_bytes++] = b;
627             uint8_t checksum = 0;
628             uint8_t *buffer = _rx_buffer + 1;
629             uint8_t *buffend = _rx_buffer + _rx_buffer_bytes - 3;
630
631             for (; buffer < buffend; buffer++) { checksum ^= *buffer; }
632
633             if ((HEXDIGIT_CHAR(checksum >> 4) == *(_rx_buffer + _rx_buffer_bytes - 2)) &&
634                 (HEXDIGIT_CHAR(checksum & 0x0F) == *(_rx_buffer + _rx_buffer_bytes - 1))) {
635                 CS_Confirm = 1;
636             }
637

```

```

638     _decode_state_config = NME_DECODE_UNINIT;
639     _rx_buffer_bytes = 0;
640     break;
641
642
643     }
644     return CS_Confirm;
645 }
646
647
648 void GPSDriverMarvelmind::decodeInit()
649 {
650
651 }
652
653 /*
654  * Marvelmind board configuration script
655  */
656
657 int GPSDriverMarvelmind::configure(unsigned &baudrate, OutputMode output_mode)
658 {
659     if (output_mode != OutputMode::GPS) {
660         GPS_WARN("MARVELMIND: Unsupported Output Mode %i", (int)output_mode);
661         return -1;
662     }
663
664     /* try different baudrates */
665     const unsigned baudrates_to_try[] = {4800, 9600, 19200, 38400, 57600, 115200, 500000};
666
667     for (unsigned int baud_i = 0; baud_i < sizeof(baudrates_to_try) / ...
668         sizeof(baudrates_to_try[0]); baud_i++){
669         baudrate = baudrates_to_try[baud_i];
670         setBaudrate(baudrate);
671         if (GPSDriverMarvelmind::receive_config(TIMEOUT_5HZ) == 1) {
672             return 0;
673         }
674     }
675
676     return -1;
677 }
678 }

```

Listing H.4: Marvelmind driver file.

H.3 GPS Module

The file `gps.cpp` located in `../px4/Firmware/src/drivers/gps` needs a few minor adjustments in order to run the new Marvelmind driver. First of all, the header file needs to be included, which can be done as shown in listing H.5, where lines 85 to 88 are shown with the added Marvelmind header file.

```

1 #include "devices/src/ubx.h"
2 #include "devices/src/mtk.h"
3 #include "devices/src/ashtech.h"
4 #include "devices/src/marv.h"

```

Listing H.5: Including Marvelmind header to gps file.

The Marvelmind supports multiple baudrates. Therefore, different baudrates can be added to the GPS module, such that the Marvelmind driver can also run on those additional baudrates. The lines 478 to 498 are shown in listing H.6 where the additional baudrates are shown.

```

1     switch (baud) {
2     case 4800:     speed = B4800;     break;
3
4     case 9600:     speed = B9600;     break;
5
6     case 19200:    speed = B19200;    break;

```

```

7
8     case 38400: speed = B38400; break;
9
10    case 57600: speed = B57600; break;
11
12    case 115200: speed = B115200; break;
13
14    case 230400: speed = B230400; break;
15
16    case 500000: speed = B500000; break;
17
18    default:
19        PX4_ERR("ERR: unknown baudrate: %d", baud);
20        return -EINVAL;
21    }

```

Listing H.6: Additional baudrates added to the gps file.

Configuration of the GPS drivers take place in the run loop, where the various drivers are tried to get a proper response. This process is a loop, which keeps running until a proper driver is found and the Marvelmind driver needs to be included in the loop. Listing H.7 shows the loop, which can be found at lines 683 to 715, where the Marvelmind driver is included. Lines 26 to 29 are added to the loop.

```

1     switch (_mode) {
2     case GPS_DRIVER_MODE_NONE:
3         _mode = GPS_DRIVER_MODE_UBX;
4
5         /* FALLTHROUGH */
6     case GPS_DRIVER_MODE_UBX: {
7         PX4_INFO("Configure UBX");
8         int32_t param_gps_ubx_dynmodel = 7; // default to 7: airborne with ...
9         <2g acceleration
10        param_get(param_find("GPS_UBX_DYNMODEL"), &param_gps_ubx_dynmodel);
11
12        _helper = new GPSDriverUBX(_interface, &GPS::callback, this, ...
13            &_report_gps_pos, _p_report_sat_info,
14            param_gps_ubx_dynmodel);
15    }
16    break;
17
18    case GPS_DRIVER_MODE_MTK:
19        PX4_INFO("Configure MTK");
20        _helper = new GPSDriverMTK(&GPS::callback, this, &_report_gps_pos);
21    break;
22
23    case GPS_DRIVER_MODE_ASHTECH:
24        PX4_INFO("Configure ASH");
25        _helper = new GPSDriverAshtech(&GPS::callback, this, &_report_gps_pos, ...
26            _p_report_sat_info);
27    break;
28
29    case GPS_DRIVER_MODE_MARVELMIND:
30        PX4_INFO("Configure MAR");
31        _helper = new GPSDriverMarvelmind(&GPS::callback, this, ...
32            &_report_gps_pos, _p_report_sat_info);
33    break;
34
35    default:
36        break;
37    }

```

Listing H.7: Configuration sequence of the GPS driver.

The next part of the code contains debugging information about the GPS drivers. If debugging in the NSH terminal is necessary, the Marvelmind driver can be included. However, for functionality it is not necessary. Listing H.8 shows the debugging lines, which can be found at around 761 to 788. Lines 18 to 20 are included for debugging purposes, if the code is de-commented.

```

1         if (!_healthy) {
2             // Helpful for debugging, but too verbose for normal ops

```

```

3 //                                     const char *mode_str = "unknown";
4 //
5 //                                     switch (_mode) {
6 //                                     case GPS_DRIVER_MODE_UBX:
7 //                                         mode_str = "UBX";
8 //                                         break;
9 //
10 //                                     case GPS_DRIVER_MODE_MTK:
11 //                                         mode_str = "MTK";
12 //                                         break;
13 //
14 //                                     case GPS_DRIVER_MODE_ASHTECH:
15 //                                         mode_str = "ASHTECH";
16 //                                         break;
17 //
18 //                                     case GPS_DRIVER_MODE_MARVELMIND:
19 //                                         mode_str = "MARVELMIND";
20 //                                         break;
21 //
22 //                                     default:
23 //                                         break;
24 //                                     }
25 //
26 //                                     PX4_WARN("module found: %s", mode_str);
27 //                                     _healthy = true;
28 }

```

Listing H.8: Debugging information about GPS drivers made available in NSH terminal.

In the autoloop mode, the software tests every installed driver, in order to find a suitable driver for the GPS. In this loop, the ASHTECH driver needs to be disabled and the Marvelmind needs to be added. This loop tests every driver and if configuration fails, it moves to the next driver. Of none driver is suitable, the loop waits for a while and tries again.

This loop can be found at around lines 798 to 825, which are shown in listing H.9. Line 9 disables the ASHTECH driver and moves from the MTK driver instantly to the Marvelmind driver. If it happens the ASHTECH driver is selected and does not configure properly, it does not select the first driver, the UBX driver, but instead moves to the Marvelmind driver. Therefore line 13 is added and line 14 is commented. Furthermore, lines 17 to 20 adds the Marvelmind driver. If the Marvelmind driver is not responding properly, the loop moves to the UBX driver and waits for a while, before it runs through the loop again.

```

1     if (_mode_auto) {
2         switch (_mode) {
3             case GPS_DRIVER_MODE_UBX:
4                 _mode = GPS_DRIVER_MODE_MTK;
5                 break;
6
7             case GPS_DRIVER_MODE_MTK:
8                 _mode = GPS_DRIVER_MODE_ASHTECH;
9                 _mode = GPS_DRIVER_MODE_MARVELMIND;
10                break;
11
12            case GPS_DRIVER_MODE_ASHTECH:
13                _mode = GPS_DRIVER_MODE_MARVELMIND;
14            //     _mode = GPS_DRIVER_MODE_UBX;
15                break;
16
17            case GPS_DRIVER_MODE_MARVELMIND:
18                _mode = GPS_DRIVER_MODE_UBX;
19                usleep(500000); // tried all possible drivers. Wait a bit before ...
20                               next round
21                break;
22
23            default:
24                break;
25        }
26    } else {
27        usleep(500000);

```


Listing H.9: GPS driver loop.

The function `print_status` shows the status of the GPS unit. This function is in general used in the NSH terminal as driver status update. For better debugging purposes, the Marvelmind command should be included in the status update command. This can be done on lines 878 to 903, which are shown in listing H.10. The lines 19 to 21 are added to show which command can be typed to request status information about the Marvelmind driver.

```

1 //GPS Mode
2 if (.fake-gps) {
3     PX4_INFO("protocol: SIMULATED");
4
5 } else {
6     switch (.mode) {
7     case GPS_DRIVER_MODE_UBX:
8         PX4_INFO("protocol: UBX");
9         break;
10
11     case GPS_DRIVER_MODE_MTK:
12         PX4_INFO("protocol: MTK");
13         break;
14
15     case GPS_DRIVER_MODE_ASHTECH:
16         PX4_INFO("protocol: ASHTECH");
17         break;
18
19     case GPS_DRIVER_MODE_MARVELMIND:
20         PX4_INFO("protocol: MARVELMIND");
21         break;
22
23     default:
24         break;
25     }
26 }
```

Listing H.10: NSH Terminal driver status update request.

The function `print_usage` shows which commands the user can use and are valid. This function is in general used in the NSH terminal as command information. For better debugging purposes and letting the user now the Marvelmind driver is available, the command should be updated. This can be done on lines 964 to 998, which are shown in listing H.11. On line 30 the `|marv` is added to the command, where the `p` commands is extended with `"ubx|mtk|ash|marv"`.

```

1 PRINT_MODULE_DESCRIPTION(
2     R"DESCR_STR(
3     ### Description
4     GPS driver module that handles the communication with the device and publishes the ...
5     position via uORB.
6     It supports multiple protocols (device vendors) and by default automatically selects ...
7     the correct one.
8
9     The module supports a secondary GPS device, specified via '-e' parameter. The position ...
10    will be published
11    on the second uORB topic instance, but it's currently not used by the rest of the ...
12    system (however the
13    data will be logged, so that it can be used for comparisons).
14
15    ### Implementation
16    There is a thread for each device polling for data. The GPS protocol classes are ...
17    implemented with callbacks
18    so that they can be used in other projects as well (eg. QGroundControl uses them too).
19
20    ### Examples
21    For testing it can be useful to fake a GPS signal (it will signal the system that it ...
22    has a valid position):
23
24    $ gps stop
25    $ gps start -f
```

```

19 )DESCR_STR");
20
21 PRINT_MODULE_USAGE_NAME("gps", "driver");
22 PRINT_MODULE_USAGE_COMMAND("start");
23 PRINT_MODULE_USAGE_PARAM_STRING('d', "/dev/ttyS3", "<file:dev>", "GPS device", true);
24 PRINT_MODULE_USAGE_PARAM_STRING('e', nullptr, "<file:dev>", "Optional secondary ...
    GPS device", true);
25
26 PRINT_MODULE_USAGE_PARAM_FLAG('f', "Fake a GPS signal (useful for testing)", true);
27 PRINT_MODULE_USAGE_PARAM_FLAG('s', "Enable publication of satellite info", true);
28
29 PRINT_MODULE_USAGE_PARAM_STRING('i', "uart", "spi|uart", "GPS interface", true);
30 PRINT_MODULE_USAGE_PARAM_STRING('p', nullptr, "ubx|mtk|ash|marv", "GPS Protocol ...
    (default=auto select)", true);
31
32 PRINT_MODULE_USAGE_DEFAULT_COMMANDS();
33
34 return 0;
35 }

```

Listing H.11: Request GPS command information.

The last part executes, the command created previously in listing H.11. In the next piece of code, the "marv" command is defined and a function is applied if desired. The piece of code executing the "p" command in the NSH Terminal can be found around lines 1067 to 1126, which are shown in listing H.12. To add the Marvelmind driver command, lines 42 and 43 are added.

```

1 while ((ch = px4_getopt(argc, argv, "d:e:fsi:p:", &myoptind, &myoptarg)) != EOF) {
2     switch (ch) {
3         case 'd':
4             device_name = myoptarg;
5             break;
6
7         case 'e':
8             device_name_secondary = myoptarg;
9             break;
10
11        case 'f':
12            fake_gps = true;
13            break;
14
15        case 's':
16            enable_sat_info = true;
17            break;
18
19        case 'i':
20            if (!strcmp(myoptarg, "spi")) {
21                interface = GPShelper::Interface::SPI;
22            } else if (!strcmp(myoptarg, "uart")) {
23                interface = GPShelper::Interface::UART;
24            } else {
25                PX4_ERR("unknown interface: %s", myoptarg);
26                error_flag = true;
27            }
28            break;
29
30        case 'p':
31            if (!strcmp(myoptarg, "ubx")) {
32                mode = GPS_DRIVER_MODE_UBX;
33            } else if (!strcmp(myoptarg, "mtk")) {
34                mode = GPS_DRIVER_MODE_MTK;
35            } else if (!strcmp(myoptarg, "ash")) {
36                mode = GPS_DRIVER_MODE_ASHTECH;
37            } else if (!strcmp(myoptarg, "marv")) {
38                mode = GPS_DRIVER_MODE_MARVELMIND;
39            } else {
40
41            }
42
43
44
45

```

```
46         PX4_ERR("unknown interface: %s", myoptarg);
47         error_flag = true;
48     }
49     break;
50
51     case '?':
52         error_flag = true;
53         break;
54
55     default:
56         PX4_WARN("unrecognized flag");
57         error_flag = true;
58         break;
59 }
60 }
```

Listing H.12: NSH Terminal Marvelmind command.

Appendix I

Marvelmind Configuration

I.1 Modem Configuration

	Modem
Starting beacon trilateration	0
Location update rate	16Hz
Update rate speedup	turbo auto
Maximum speed m/s	5
3D navigation	enabled
Power safe function	disabled
Windows of averaging	10
IMU	
Movement filtering	disabled
Use pairs of beacons	disabled
Analyze signal quality	disabled
Minimum signal quality	10
Track with low signal	blue
High resolution mode (mm)	disabled
Accept new woken devices	disabled
Temperature of air, ?C	23
Radio Frequency Band	433 MHz
Device address	1
Channel	0
Parameters of Radio	
Base frequency	433,400 MHz
Radio Profile	153 Kbps
Device address	1
Hopping mode	None
Channel	0
Modulation	GFSK
Power of TX	10dBm
Channel spacing KHz	49.194
Intermediate frequency (ID) KHz	305
Offset frequency KHz	-53.96
Deviation frequency KHz	152.344
Channel bandwidth KHz	541.667
CCA mode	always
DC blocking filter	enabled
Manchester	disabled
Whitening	enabled
FEC	enabled

Table I.1: Modem configuration Table 1.

Interfaces	
	Modem
UART speed bps	500000
Protocol on UART/USB output	Marvelmind
Geofencing	
Latitude	N0
Longitude	E0
Limitation distance	auto
Stationary beacons visible	enabled
Service areas visible	enabled
Service areas active	enabled
Submap X shift in m	0.00
Submap Y shift in m	0.00
Submap rotation degrees	0.00

Table I.2: Modem configuration Table 2.

I.2 Hedgehog Configuration

Hedgehog 21	
Hedgehog mode	enabled
Radio Frequency Band	433 MHz
Device address	21
Channel	0
Minimum Threshold	-50
IMU	
Ax zero	-10
Ay zero	8
Az zero	-122
Ax K	0.982
Ay K	0.973
Az K	0.982
Parameters of Radio	
Base frequency	433,400 MHz
Radio Profile	153 Kbps
Device address	21
Channel	0
Modulation	GFSK
Power of TX	10dBm
Channel spacing KHz	49.194
Intermediate frequency (ID) KHz	305
Offset frequency KHz	-53.96
Deviation frequency KHz	152.344
Channel bandwidth KHz	541.667
CCA mode	always
DC blocking filter	enabled
Manchester	disabled
Whitening	enabled
FEC	enabled

Table I.3: Hedgehog configuration Table 1.

Ultrasound

Ultrasound	Hedgehog 21
Mode of work	TX+RX normal
Analoge power in sleep	disabled
Power after transmission	not turn off
Transmitter mode	PWM
Frequency Hz	31000
Duty %	50
Number of periods	30
Amplifier limitation (calibrated)	4000
Amplification	AGC
Time gain control	disabled
AGC desired level	-500
AGC hysteresis	130
AGC step dB	3
Mode of treshold	automatic
minimum treshold	-50
Treshold to noise dB	6
Signal detection	by ADC
Periods for detector	5
Min. speed of raise LSB/cm	5,0
Min. over raise for new front	30
Coef. of estimated front quality	8
AGC low treshold, over raise	3
Speed of amplification increase	10
AGC high treshold, over raise	20
Speed of amplification decrease	10
Receive window low m	0
Receive window high m	255
Minimum distance limitation	enabled
Auto measurements when radio gps	enabled
Filter selection	19 KHz
RX1 normal	enabled
RX2 normal	enabled
RX3 normal	enabled
RX4 normal	enabled
RX5 normal	enabled
RX1 frozen	enabled
RX2 frozen	enabled
RX3 frozen	enabled
RX4 frozen	enabled
RX5 frozen	enabled

Table I.4: Hedgehog configuration Table 2.

Interfaces	
	Hedgehog 21
UART speed bps	115200
Protocol on UART/USB output	NMEA0183
NMEA \$GPRMC	disabled
NMEA \$GPGGA	enabled
NMEA \$GPVTG	disabled
NMEA \$GPZDA	disabled
External device	No control
PA15 pin function	SPI slave CS
Raw inertial sensors data	disabled
Processed IMU data	disabled
Raw distance data	disabled
User payload data size	0
Geofencing	
Latitude	N0
Longitude	E0
Misc. settings	
Sleep with external power	60
Hedgehogs pairing	
Sleep with external power	no pairing

Table I.5: Hedgehog configuration Table 3.

I.3 Beacon Configuration

	Beacon 2	Beacon 3	Beacon 13	Beacon 15
Hedgehog mode	disabled	disabled	disabled	disabled
Height	2.37 m	2.37 m	2.37 m	2.37 m
Radio Frequency Band	433 MHz	433 MHz	433 MHz	433 MHz
Device address	2	3	13	15
Channel	0	0	0	0
Minimum Threshold	-50	-50	-50	-50
IMU				
Ax zero	-10	-10	-10	-10
Ay zero	8	8	8	8
Az zero	-122	-122	-122	-122
Ax K	0.982	0.982	0.982	0.982
Ay K	0.973	0.973	0.973	0.973
Az K	0.982	0.982	0.982	0.982
Parameters of Radio				
Base frequency	433,400 MHz	433,400 MHz	433,400 MHz	433,400 MHz
Radio Profile	153 Kbps	153 Kbps	153 Kbps	153 Kbps
Device address	2	3	13	15
Channel	0	0	0	0
Modulation	GFSK	GFSK	GFSK	GFSK
Power of TX	10dBm	10dBm	10dBm	10dBm
Channel spacing KHz	49.194	49.194	49.194	49.194
Intermediate frequency (ID) KHz	305	305	305	305
Offset frequency KHz	0.00	-65.06	-55.54	-55.54
Deviation frequency KHz	152.344	152.344	152.344	152.344
Channel bandwidth KHz	541.667	541.667	541.667	541.667
CCA mode	always	always	always	always
DC blocking filter	enabled	enabled	enabled	enabled
Manchester	disabled	disabled	disabled	disabled
Whitening	enabled	enabled	enabled	enabled
FEC	enabled	enabled	enabled	enabled

Table I.6: Beacons configuration Table 1.

	Ultrasound			
	Beacon 2	Beacon 3	Beacon 13	Beacon 15
Mode of work	TX+RX normal	TX+RX normal	TX+RX normal	TX+RX normal
Analoge power in sleep	disabled	disabled	disabled	disabled
Power after transmission	not turn off	not turn off	not turn off	not turn off
Transmitter mode	PWM	PWM	PWM	PWM
Frequency Hz	31000	31000	31000	31000
Duty %	50	50	50	50
Number of periods	30	30	30	30
Amplifier limitation (calibrated)	4000	4000	4000	4000
Amplification	AGC	AGC	AGC	AGC
Time gain control	disabled	disabled	disabled	disabled
AGC desired level	-500	-500	-500	-500
AGC hysteresis	130	130	130	130
AGC step dB	3	3	3	3
Mode of treshold	automatic	automatic	automatic	automatic
minimum treshold	-50	-50	-50	-50
Treshold to noise dB	6	6	6	6
Signal detection	by ADC	by ADC	by ADC	by ADC
Periods for detector	5	5	5	5
Min. speed of raise LSB/cm	5,0	5,0	5,0	5,0
Min. over raise for new front	30	30	30	30
Coef. of estimated front quality	8	8	8	8
AGC low treshold, over raise	3	3	3	3
Speed of amplification increase	10	10	10	10
AGC high treshold, over raise	20	20	20	20
Speed of amplification decrease	10	10	10	10
Receive window low m	0	0	0	0
Receive window high m	255	255	255	255
Minimum distance limitation	enabled	enabled	enabled	enabled
Auto measurements when radio gps	enabled	enabled	enabled	enabled
Filter selection	19 KHz	19 KHz	19 KHz	19 KHz
RX1 normal	enabled	enabled	enabled	enabled
RX2 normal	enabled	enabled	enabled	enabled
RX3 normal	enabled	enabled	enabled	enabled
RX4 normal	enabled	enabled	enabled	enabled
RX5 normal	enabled	enabled	enabled	enabled
RX1 frozen	enabled	enabled	enabled	enabled
RX2 frozen	enabled	enabled	enabled	enabled
RX3 frozen	enabled	enabled	enabled	enabled
RX4 frozen	enabled	enabled	enabled	enabled
RX5 frozen	enabled	enabled	enabled	enabled

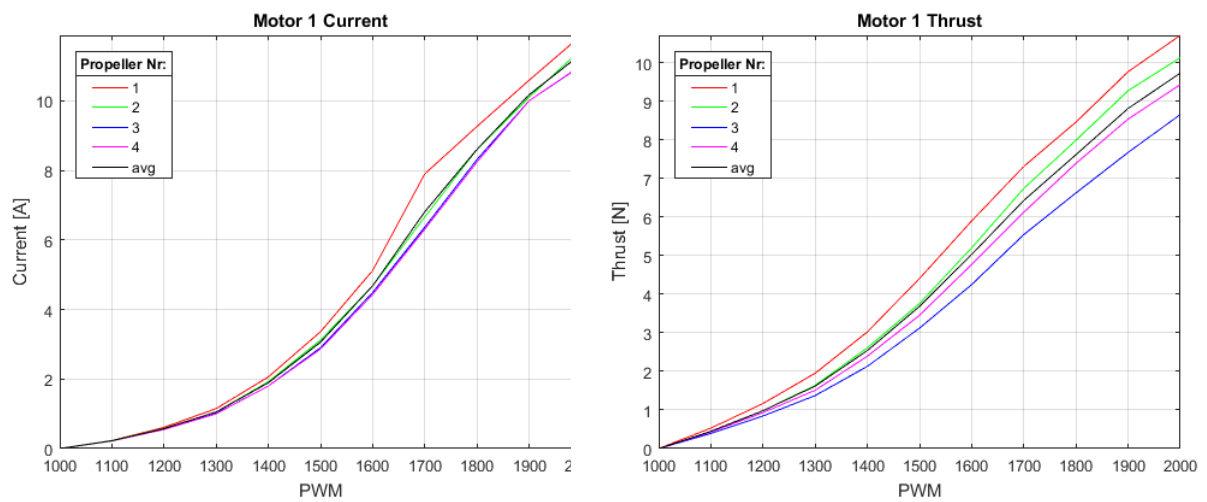
Table I.7: Beacons configuration Table 2.

Interfaces				
	Beacon 2	Beacon 3	Beacon 13	Beacon 15
UART speed bps	500000	500000	500000	500000
Protocol on UART/USB output	Marvelmind	Marvelmind	Marvelmind	Marvelmind
PA15 pin function	SPI slave CS	SPI slave CS	SPI slave CS	SPI slave CS
Raw inertial sensors data	disabled	disabled	disabled	disabled
Processed IMU data	disabled	disabled	disabled	disabled
Geofencing				
Latitude	N0	N0	N0	N0
Longitude	E0	E0	E0	E0
Misc. settings				
Sleep with external power	60	60	60	60
Hedgehogs pairing				
Sleep with external power	no pairing	no pairing	no pairing	no pairing

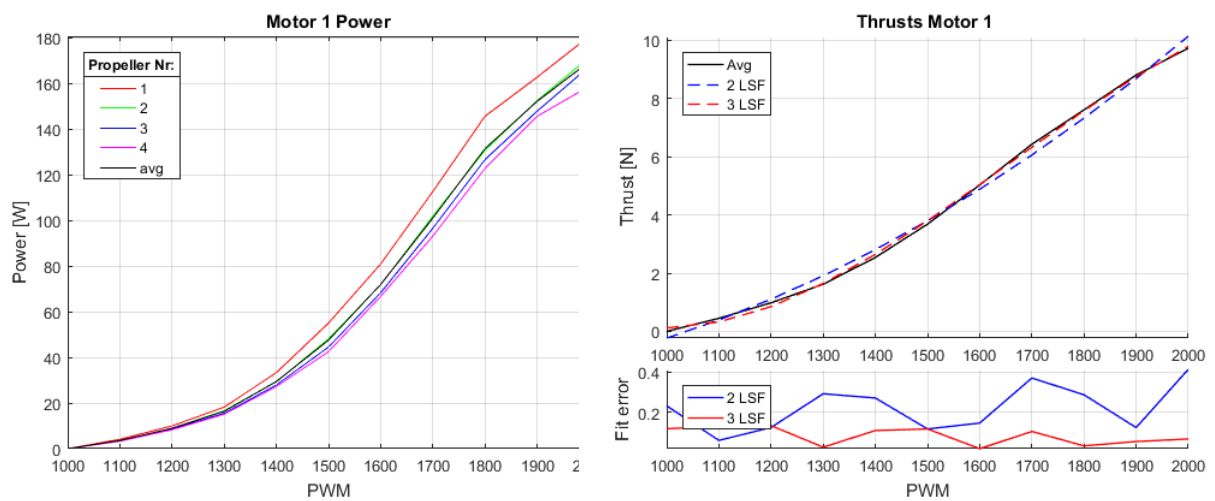
Table I.8: Beacons configuration Table 3.

Appendix J

Measurement and least squares fit results motors

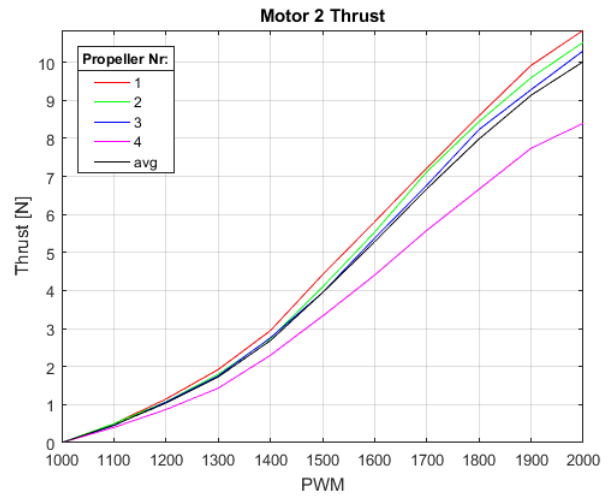
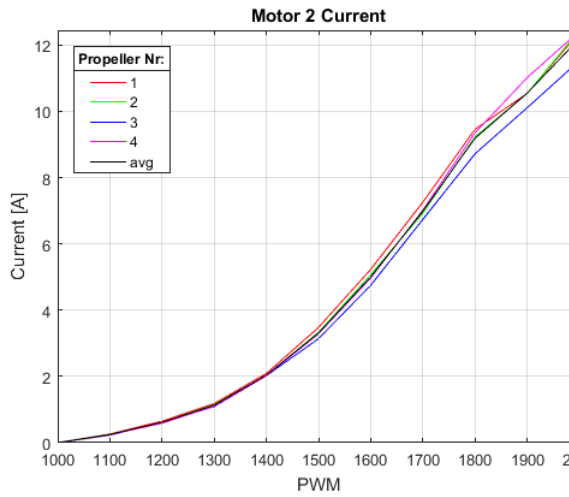


(a) Current measurements motor 1 with 4 different propellers. (b) Thrust measurements motor 1 with 4 different propellers.



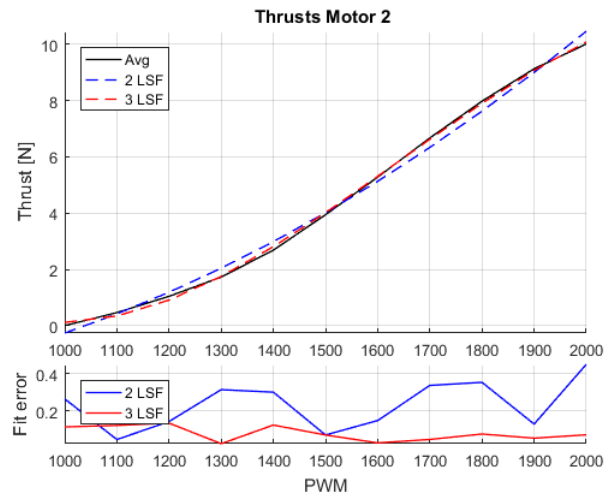
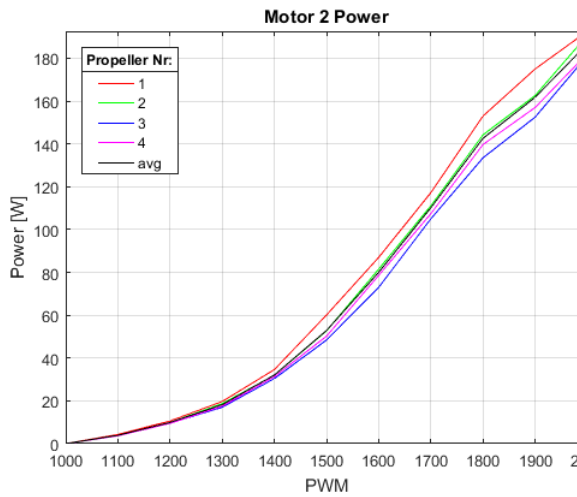
(c) Power measurements motor 1 with 4 different propellers. (d) Least squares fit motor 1 using second and third order polynomial, and their respectively error.

Figure J.1: Measurement and least squares fit results motor 1.



(a) Current measurements motor 2 with 4 different propellers.

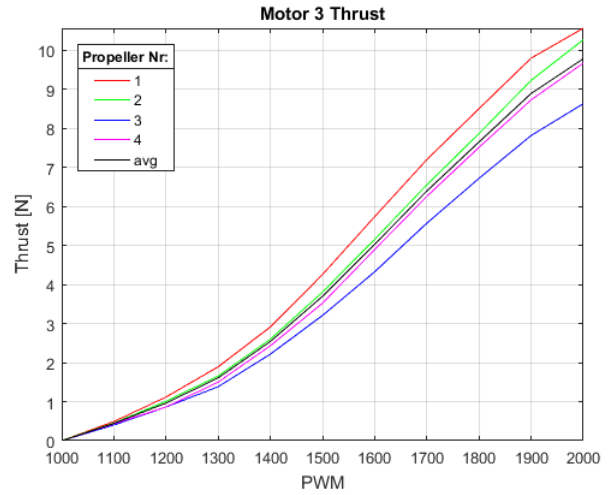
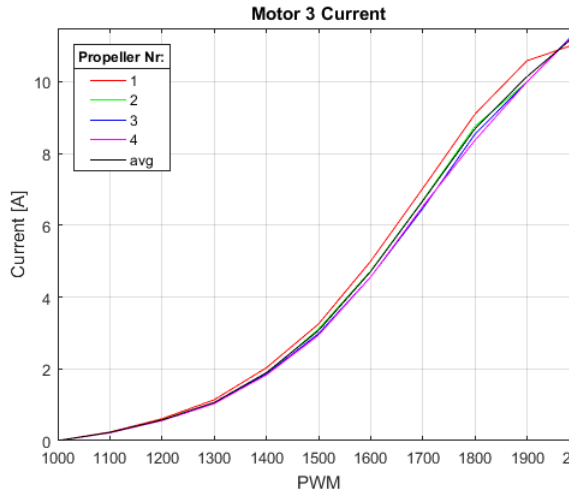
(b) Thrust measurements motor 2 with 4 different propellers.



(c) Power measurements motor 2 with 4 different propellers.

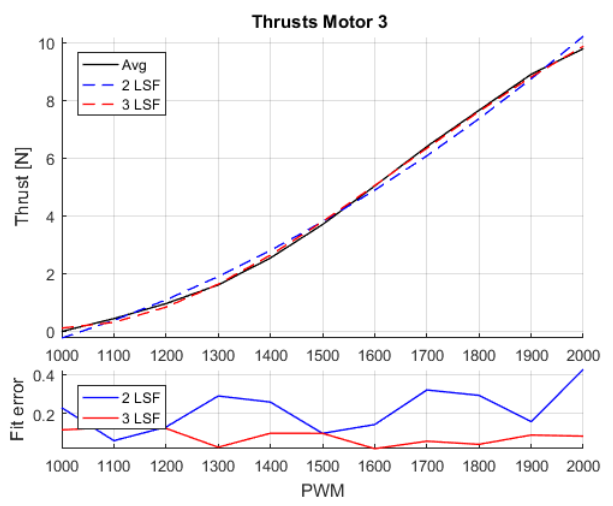
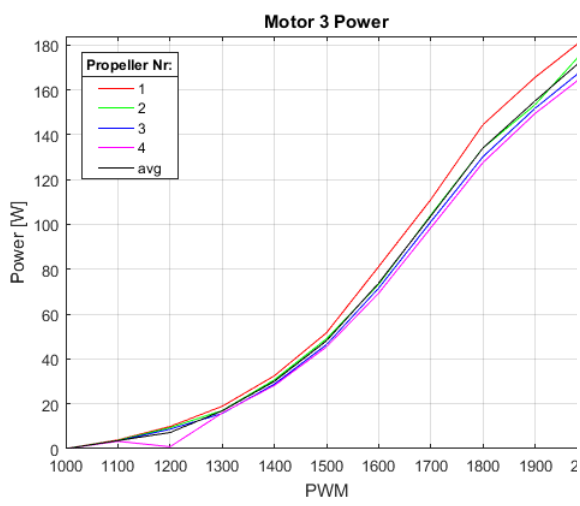
(d) Least squares fit motor 2 using second and third order polynomial, and their respectively error.

Figure J.2: Measurement and least squares fit results motor 2.



(a) Current measurements motor 3 with 4 different propellers.

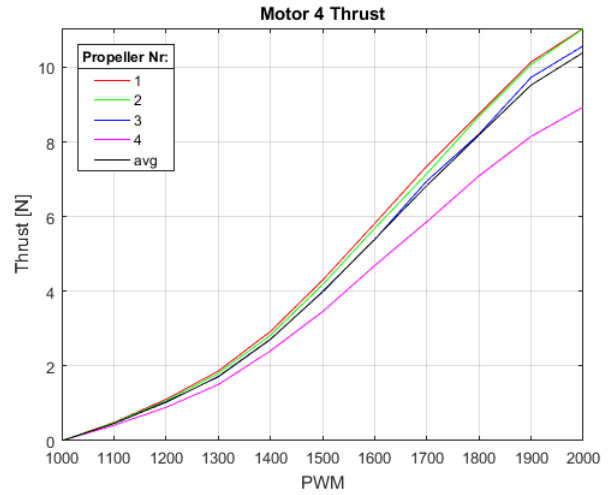
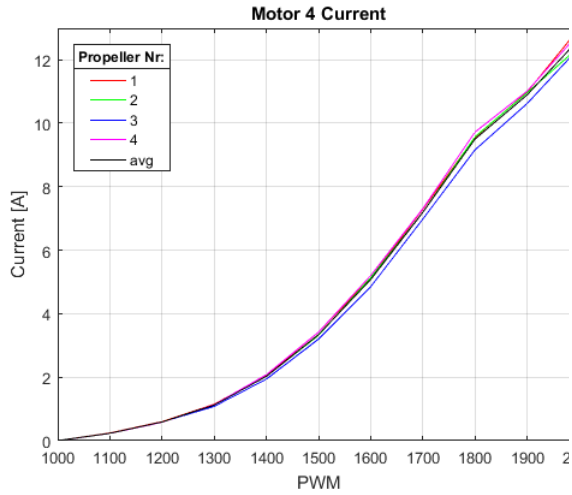
(b) Thrust measurements motor 3 with 4 different propellers.



(c) Power measurements motor 3 with 4 different propellers.

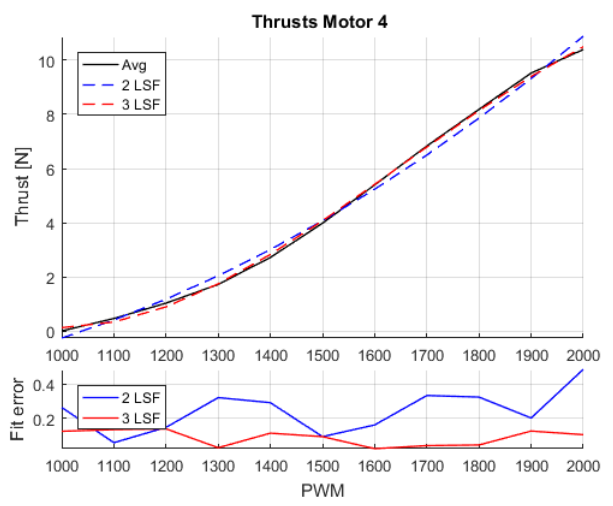
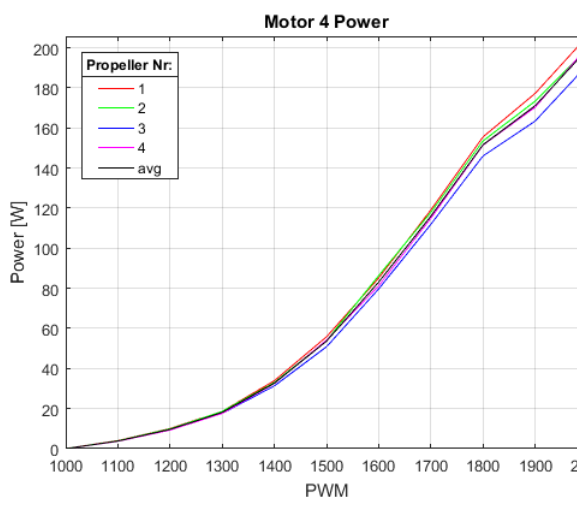
(d) Least squares fit motor 3 using second and third order polynomial, and their respectively error.

Figure J.3: Measurement and least squares fit results motor 3.



(a) Current measurements motor 4 with 4 different propellers.

(b) Thrust measurements motor 4 with 4 different propellers.



(c) Power measurements motor 4 with 4 different propellers.

(d) Least squares fit motor 4 using second and third order polynomial, and their respectively error.

Figure J.4: Measurement and least squares fit results motor 4.

Appendix K

Ellipsoid fit

```
1 function [ center, radii, evecs, v, chi2 ] = ellipsoid.fit.new( X, equals )
2 %
3 % Fit an ellipsoid/sphere/paraboloid/hyperboloid to a set of xyz data points:
4 %
5 % [center, radii, evecs, pars ] = ellipsoid.fit( X )
6 % [center, radii, evecs, pars ] = ellipsoid.fit( [x y z] );
7 % [center, radii, evecs, pars ] = ellipsoid.fit( X, 1 );
8 % [center, radii, evecs, pars ] = ellipsoid.fit( X, 2, 'xz' );
9 % [center, radii, evecs, pars ] = ellipsoid.fit( X, 3 );
10 %
11 % Parameters:
12 % * X, [x y z] - Cartesian data, n x 3 matrix or three n x 1 vectors
13 % * flag      - '' or empty fits an arbitrary ellipsoid (default),
14 %            - 'xy' fits a spheroid with x- and y- radii equal
15 %            - 'xz' fits a spheroid with x- and z- radii equal
16 %            - 'xyz' fits a sphere
17 %            - '0' fits an ellipsoid with its axes aligned along [x y z] axes
18 %            - '0xy' the same with x- and y- radii equal
19 %            - '0xz' the same with x- and z- radii equal
20 %
21 % Output:
22 % * center    - ellipsoid or other conic center coordinates [xc; yc; zc]
23 % * radii     - ellipsoid or other conic radii [a; b; c]
24 % * evecs     - the radii directions as columns of the 3x3 matrix
25 % * v         - the 10 parameters describing the ellipsoid / conic algebraically:
26 %            Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exz + 2Fyz + 2Gx + 2Hy + 2Iz + J = 0
27 % * chi2      - residual sum of squared errors (chi^2), this chi2 is in the
28 %            coordinate frame in which the ellipsoid is a unit sphere.
29 %
30 % Author:
31 % Yury Petrov, Oculus VR
32 % Date:
33 % September, 2015
34 %
35
36 narginchk( 1, 3 ) ; % check input arguments
37 if nargin == 1
38     equals = ''; % no constraints by default
39 end
40
41 if size( X, 2 ) ≠ 3
42     error( 'Input data must have three columns!' );
43 else
44     x = X( :, 1 );
45     y = X( :, 2 );
46     z = X( :, 3 );
47 end
48
49 % need nine or more data points
50 if length( x ) < 9 && strcmp( equals, '' )
51     error( 'Must have at least 9 points to fit a unique ellipsoid' );
52 end
53 if length( x ) < 8 && ( strcmp( equals, 'xy' ) || strcmp( equals, 'xz' ) )
54     error( 'Must have at least 8 points to fit a unique ellipsoid with two equal radii' );
55 end
56 if length( x ) < 6 && strcmp( equals, '0' )
57     error( 'Must have at least 6 points to fit a unique oriented ellipsoid' );
58 end
59 if length( x ) < 5 && ( strcmp( equals, '0xy' ) || strcmp( equals, '0xz' ) )
60     error( 'Must have at least 5 points to fit a unique oriented ellipsoid with two ...
61         equal radii' );
62 end
```

```

62 if length( x ) < 4 && strcmp( equals, 'xyz' );
63     error( 'Must have at least 4 points to fit a unique sphere' );
64 end
65
66 % fit ellipsoid in the form Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exz + 2Fyz + 2Gx +
67 % 2Hy + 2Iz + J = 0 and A + B + C = 3 constraint removing one extra
68 % parameter
69 if strcmp( equals, '' )
70     D = [ x .* x + y .* y - 2 * z .* z, ...
71          x .* x + z .* z - 2 * y .* y, ...
72          2 * x .* y, ...
73          2 * x .* z, ...
74          2 * y .* z, ...
75          2 * x, ...
76          2 * y, ...
77          2 * z, ...
78          1 + 0 * x ]; % ndatapoints x 9 ellipsoid parameters
79 elseif strcmp( equals, 'xy' )
80     D = [ x .* x + y .* y - 2 * z .* z, ...
81          2 * x .* y, ...
82          2 * x .* z, ...
83          2 * y .* z, ...
84          2 * x, ...
85          2 * y, ...
86          2 * z, ...
87          1 + 0 * x ]; % ndatapoints x 8 ellipsoid parameters
88 elseif strcmp( equals, 'xz' )
89     D = [ x .* x + z .* z - 2 * y .* y, ...
90          2 * x .* y, ...
91          2 * x .* z, ...
92          2 * y .* z, ...
93          2 * x, ...
94          2 * y, ...
95          2 * z, ...
96          1 + 0 * x ]; % ndatapoints x 8 ellipsoid parameters
97 % fit ellipsoid in the form Ax^2 + By^2 + Cz^2 + 2Gx + 2Hy + 2Iz = 1
98 elseif strcmp( equals, '0' )
99     D = [ x .* x + y .* y - 2 * z .* z, ...
100          x .* x + z .* z - 2 * y .* y, ...
101          2 * x, ...
102          2 * y, ...
103          2 * z, ...
104          1 + 0 * x ]; % ndatapoints x 6 ellipsoid parameters
105 % fit ellipsoid in the form Ax^2 + By^2 + Cz^2 + 2Gx + 2Hy + 2Iz = 1,
106 % where A = B or B = C or A = C
107 elseif strcmp( equals, '0xy' )
108     D = [ x .* x + y .* y - 2 * z .* z, ...
109          2 * x, ...
110          2 * y, ...
111          2 * z, ...
112          1 + 0 * x ]; % ndatapoints x 5 ellipsoid parameters
113 elseif strcmp( equals, '0xz' )
114     D = [ x .* x + z .* z - 2 * y .* y, ...
115          2 * x, ...
116          2 * y, ...
117          2 * z, ...
118          1 + 0 * x ]; % ndatapoints x 5 ellipsoid parameters
119 % fit sphere in the form A(x^2 + y^2 + z^2) + 2Gx + 2Hy + 2Iz = 1
120 elseif strcmp( equals, 'xyz' )
121     D = [ 2 * x, ...
122          2 * y, ...
123          2 * z, ...
124          1 + 0 * x ]; % ndatapoints x 4 ellipsoid parameters
125 else
126     error( [ 'Unknown parameter value ' equals '!' ] );
127 end
128
129 % solve the normal system of equations
130 d2 = x .* x + y .* y + z .* z; % the RHS of the llsq problem (y's)
131 u = ( D' * D ) \ ( D' * d2 ); % solution to the normal equations
132
133 % find the residual sum of errors

```



```

134 % chi2 = sum( ( 1 - ( D * u ) ./ d2 ).^2 ); % this chi2 is in the coordinate frame in ...
      which the ellipsoid is a unit sphere.
135
136 % find the ellipsoid parameters
137 % convert back to the conventional algebraic form
138 if strcmp( equals, '' )
139     v(1) = u(1) +      u(2) - 1;
140     v(2) = u(1) - 2 * u(2) - 1;
141     v(3) = u(2) - 2 * u(1) - 1;
142     v( 4 : 10 ) = u( 3 : 9 );
143 elseif strcmp( equals, 'xy' )
144     v(1) = u(1) - 1;
145     v(2) = u(1) - 1;
146     v(3) = -2 * u(1) - 1;
147     v( 4 : 10 ) = u( 2 : 8 );
148 elseif strcmp( equals, 'xz' )
149     v(1) = u(1) - 1;
150     v(2) = -2 * u(1) - 1;
151     v(3) = u(1) - 1;
152     v( 4 : 10 ) = u( 2 : 8 );
153 elseif strcmp( equals, '0' )
154     v(1) = u(1) +      u(2) - 1;
155     v(2) = u(1) - 2 * u(2) - 1;
156     v(3) = u(2) - 2 * u(1) - 1;
157     v = [ v(1) v(2) v(3) 0 0 0 u( 3 : 6 )' ];
158 elseif strcmp( equals, '0xy' )
159     v(1) = u(1) - 1;
160     v(2) = u(1) - 1;
161     v(3) = -2 * u(1) - 1;
162     v = [ v(1) v(2) v(3) 0 0 0 u( 2 : 5 )' ];
163 elseif strcmp( equals, '0xz' )
164     v(1) = u(1) - 1;
165     v(2) = -2 * u(1) - 1;
166     v(3) = u(1) - 1;
167     v = [ v(1) v(2) v(3) 0 0 0 u( 2 : 5 )' ];
168 elseif strcmp( equals, 'xyz' )
169     v = [ -1 -1 -1 0 0 0 u( 1 : 4 )' ];
170 end
171 v = v';
172
173 % form the algebraic form of the ellipsoid
174 A = [ v(1) v(4) v(5) v(7); ...
      v(4) v(2) v(6) v(8); ...
      v(5) v(6) v(3) v(9); ...
      v(7) v(8) v(9) v(10) ];
175
176 % find the center of the ellipsoid
177 center = -A( 1:3, 1:3 ) \ v( 7:9 );
178 % form the corresponding translation matrix
179 T = eye( 4 );
180 T( 4, 1:3 ) = center';
181 % translate to the center
182 R = T * A * T';
183 % solve the eigenproblem
184 [ evects, evals ] = eig( R( 1:3, 1:3 ) / -R( 4, 4 ) );
185 radii = sqrt( 1 ./ diag( abs( evals ) ) );
186 sgns = sign( diag( evals ) );
187 radii = radii .* sgns;
188
189 % calculate difference of the fitted points from the actual data normalized by the ...
      conic radii
190 d = [ x - center(1), y - center(2), z - center(3) ]; % shift data to origin
191 d = d * evects; % rotate to cardinal axes of the conic;
192 d = [ d(:,1) / radii(1), d(:,2) / radii(2), d(:,3) / radii(3) ]; % normalize to the ...
      conic radii
193 chi2 = sum( abs( 1 - sum( d.^2 .* repmat( sgns', size( d, 1 ), 1 ), 2 ) ) );
194
195 if abs( v(end) ) > 1e-6
196     v = -v / v(end); % normalize to the more conventional form with constant term = -1
197 else
198     v = -sign( v(end) ) * v;
199 end
200
201 end

```

Listing K.1: Matlab code for fitting an ellipsoid on data. Obtained from [1].

Appendix L

Inverse thrust mapping

Since in section 4.1.5 a third order polynomial is fitted to the measured thrust data to get a mapping between pwm and thrust, an algorithm performing the conversion from thrust to pwm is needed. The obtained third order polynomial is described as equation L.1, which can be written as solving a cubic polynomial of form equation L.2. The procedure described here, is following the discriminant method.

$$T_{desired} = b_3 \text{pwm}^3 + b_2 \text{pwm}^2 + b_1 \text{pwm} + b_0 \quad (\text{L.1})$$

$$0 = \text{pwm}^3 + \frac{b_2}{b_3} \text{pwm}^2 + \frac{b_1}{b_3} \text{pwm} + \frac{b_0 - T_{desired}}{b_3} \quad (\text{L.2})$$

$$ax^3 + bx^2 + cx + d = 0 \quad (\text{L.3})$$

Starting with a much simpler function, given equation L.3. Solution to this equations can come in three different forms, solutions with:

- Only 1 real root and 2 imaginary roots.
- All roots are equal and real.
- All roots are different and real.

The function f , g and h , provided in equation L.4, split the solutions in three different categories described above.

$$\begin{aligned} f &= \frac{3ac - b^2}{3a^2} \\ g &= \frac{2b^3 - 9abc + 27a^2d}{27a^3} \\ h &= \frac{g^2}{4} + \frac{f^3}{27} \end{aligned} \quad (\text{L.4})$$

The categorization depends on the values of h , g and f as follows:

- $h > 0$; Only 1 real root and 2 imaginary roots.
- $h = 0, g = 0$ and $f = 0$; All roots are equal and real.
- $h \leq 0$; All roots are different and real.

To find the solutions to 1 real root and 2 imaginary roots, the following procedure is used:

$$S = \sqrt[3]{-\frac{g}{2} + \sqrt{h}} \quad (\text{L.5})$$

$$U = \sqrt[3]{-\frac{g}{2} - \sqrt{h}} \quad (\text{L.6})$$

$$x_1 = S + U - \frac{b}{3a} \quad (\text{L.7})$$

$$x_2 = -\frac{S + U}{2} - \frac{b}{3a} + i \frac{\sqrt{3}(S - U)}{2} \quad (\text{L.8})$$

$$x_3 = -\frac{S + U}{2} - \frac{b}{3a} - i \frac{\sqrt{3}(S - U)}{2} \quad (\text{L.9})$$

Where $x_1 \in \mathbb{R}$ is the only real solution and $x_2 \in \mathbb{C}$ and $x_3 \in \mathbb{C}$ are the complex solutions to the cubicle polynomial. To find the solutions to all roots are real and equal, the following procedure is used:

$$\begin{aligned}x_1 &= \sqrt[3]{-\frac{d}{a}} \\x_2 &= \sqrt[3]{-\frac{d}{a}} \\x_3 &= \sqrt[3]{-\frac{d}{a}}\end{aligned}\tag{L.10}$$

Where $x_1 = x_2 = x_3 \in \mathbb{R}$. The last procedure shows how to find all different real roots:

$$j = \sqrt[3]{\sqrt{\frac{g^2}{4} - h}}\tag{L.11}$$

$$k = \arccos\left(\frac{-g}{2\sqrt{\frac{g^2}{4} - h}}\right)\tag{L.12}$$

$$x_1 = 2j \cos\left(\frac{k}{3}\right) - \frac{b}{3a}\tag{L.13}$$

$$x_2 = -j \left(\cos\left(\frac{k}{3}\right) + \sqrt{3} \sin\left(\frac{k}{3}\right) \right) - \frac{b}{3a}\tag{L.14}$$

$$x_3 = -j \left(\cos\left(\frac{k}{3}\right) - \sqrt{3} \sin\left(\frac{k}{3}\right) \right) - \frac{b}{3a}\tag{L.15}$$

Where $x_1 \in \mathbb{R}$, $x_2 \in \mathbb{R}$ and $x_3 \in \mathbb{R}$.

Appendix M

Solidworks quad-copter model



Figure M.1: TBS Quad-copter reconstructed in Solidworks.

```

1 Mass properties of TBS_Quad
2 Configuration: Default
3 Coordinate system: — default —
4
5 * Includes the mass properties of one or more hidden components/bodies.
6
7 Mass = 1366.00 grams
8
9 Volume = 4524822.43 cubic millimeters
10
11 Surface area = 795321.72 square millimeters
12
13 Center of mass: ( millimeters )
14 X = 82.97
15 Y = 202.94
16 Z = 295.54
17
18 Principal axes of inertia and principal moments of inertia: ( grams * square ...
millimeters )
19 Taken at the center of mass.
20 Ix = (-0.04, -0.02, 1.00) Px = 30369852.63
21 Iy = ( 1.00, 0.00, 0.04) Py = 30852409.95
22 Iz = ( 0.00, 1.00, 0.02) Pz = 59906882.28
23
24 Moments of inertia: ( grams * square millimeters )
25 Taken at the center of mass and aligned with the output coordinate system.
26 Lxx = 30851658.96 Lxy = -1932.81 Lxz = -19060.68
27 Lyx = -1932.81 Lyy = 59897904.59 Lyz = -514860.89
28 Lzx = -19060.68 Lzy = -514860.89 Lzz = 30379581.31
29
30 Moments of inertia: ( grams * square millimeters )
31 Taken at the output coordinate system.
32 Ixx = 206427013.30 Ixy = 23000234.15 Ixz = 33478597.94
33 Iyx = 23000234.15 Iyy = 188617268.75 Iyz = 81416273.19
34 Izx = 33478597.94 Izy = 81416273.19 Izz = 96044507.67

```

Listing M.1: Mass properties results quad-copter in Solidworks.

Appendix N

Wind conditions and Disturbances

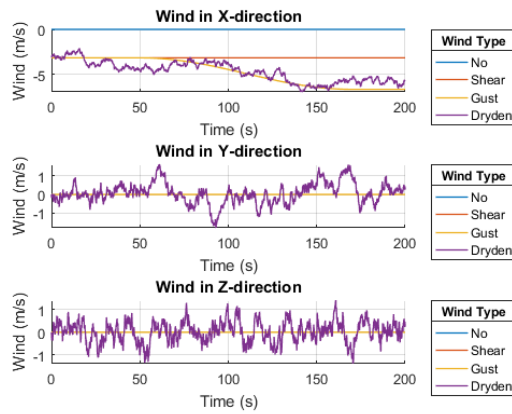


Figure N.1: Wind conditions in simulation.

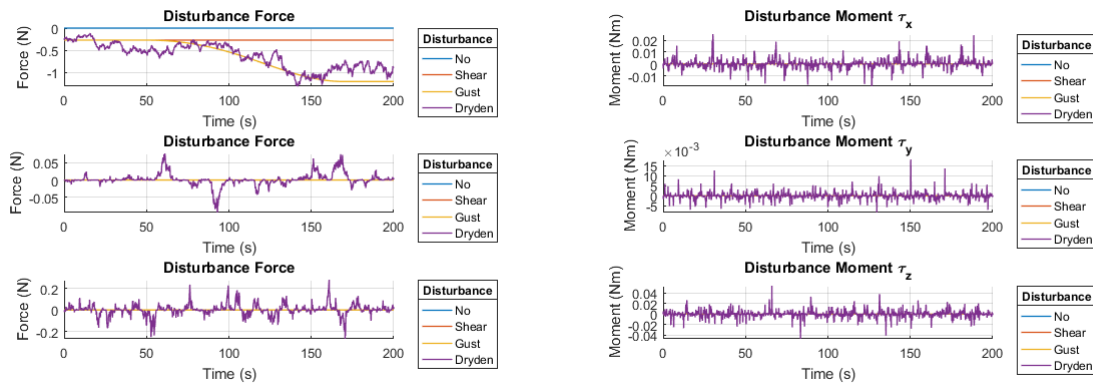


Figure N.2: Disturbance moments and forces resulting from wind in simulation.

Appendix O

Position error in windless conditions

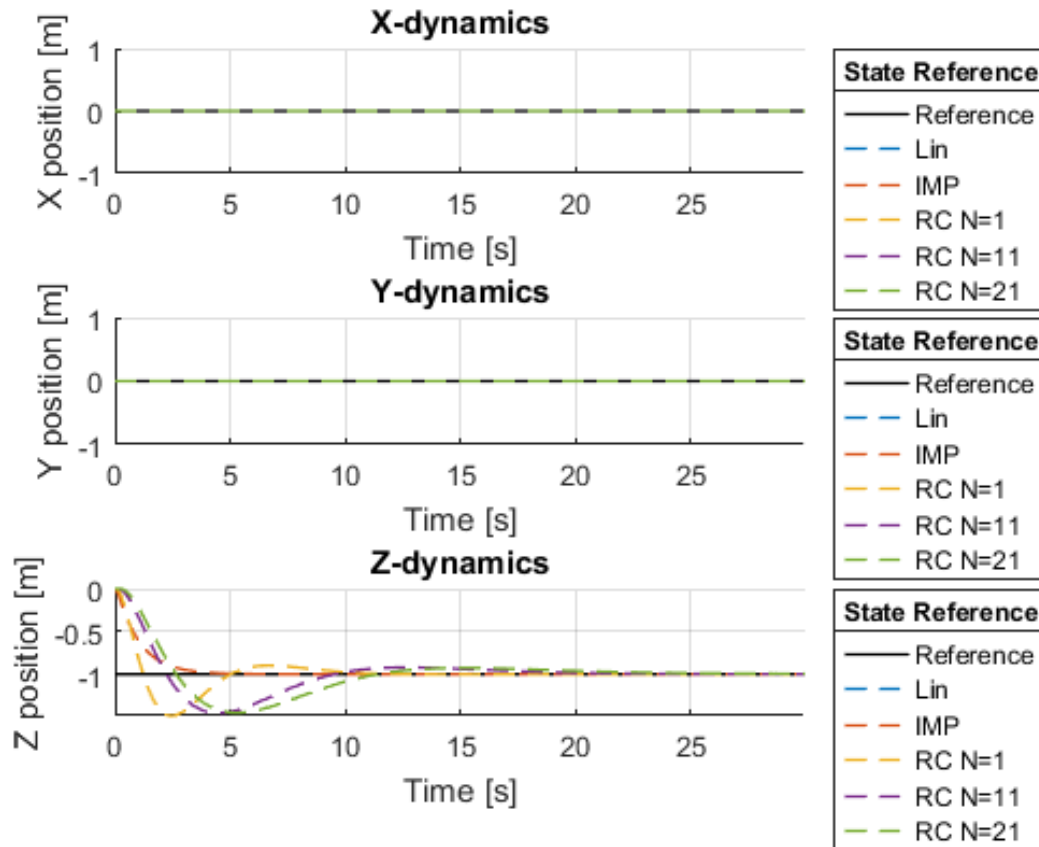


Figure O.1: System dynamics for various controllers in hovering mode without wind.

Appendix P

Position error in shear wind conditions

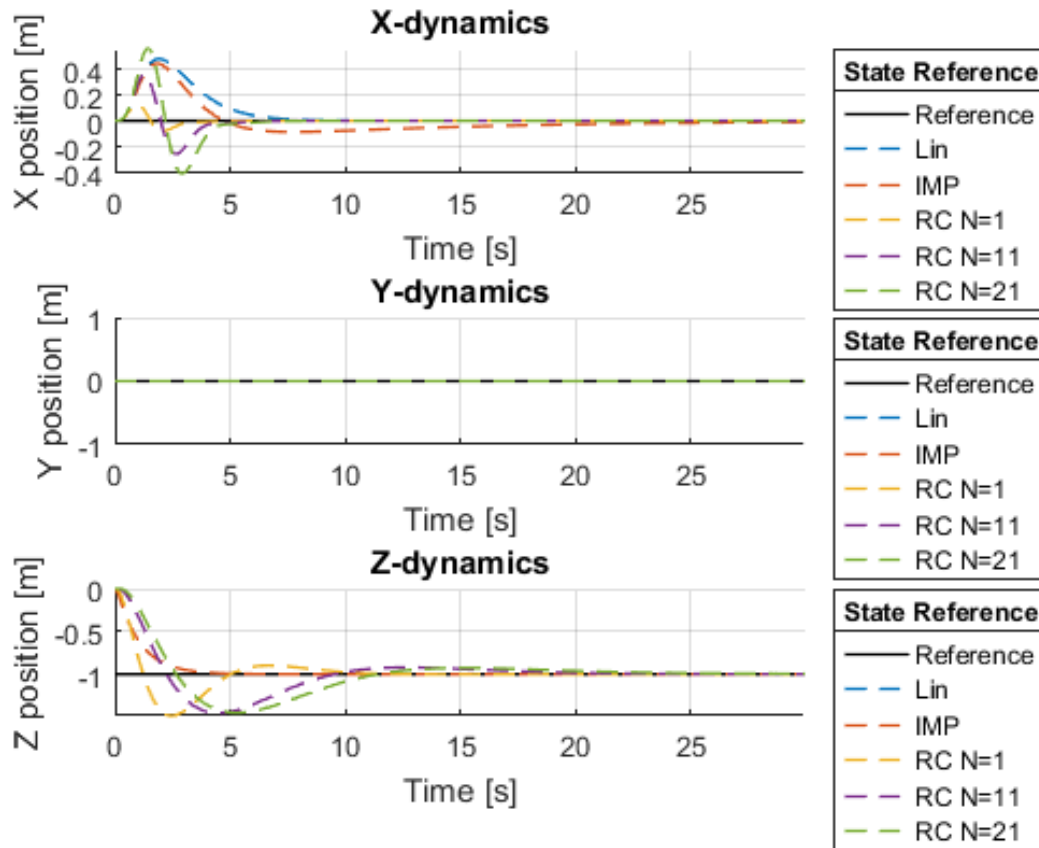


Figure P.1: System dynamics for various controllers in hovering mode in shear wind conditions.

Appendix Q

Position error in gust wind conditions

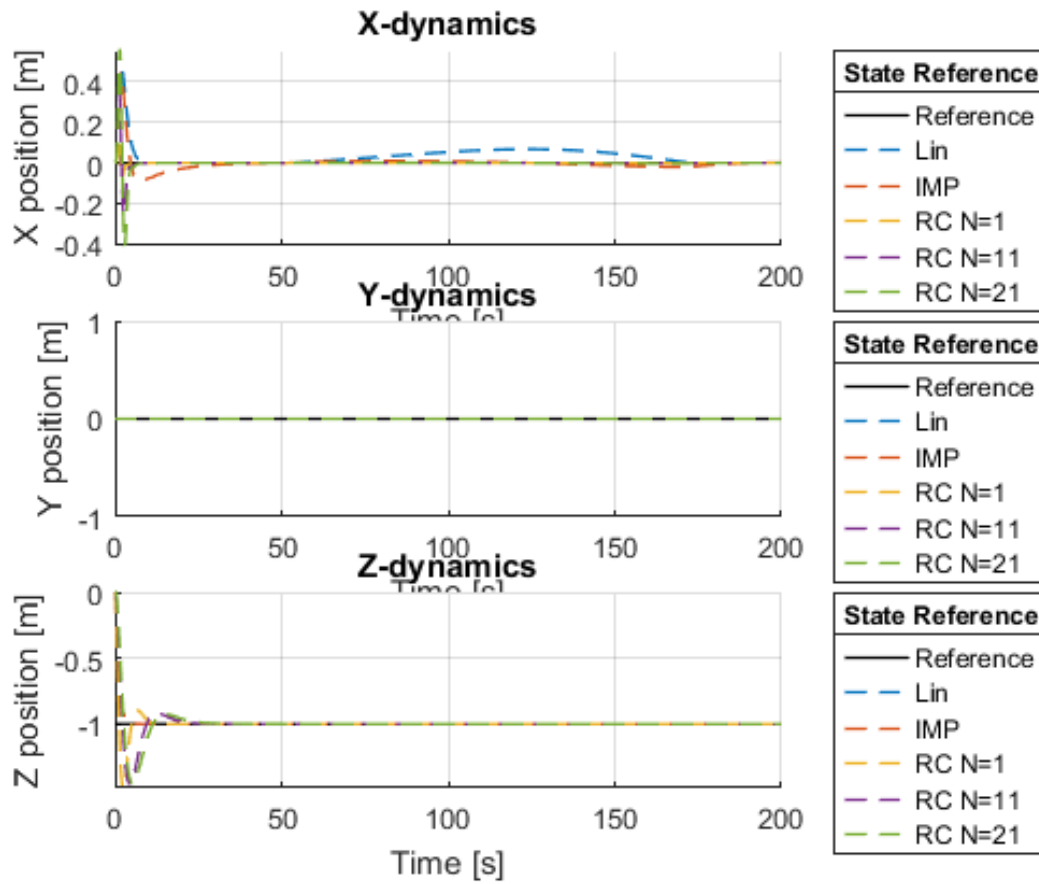


Figure Q.1: System dynamics for various controllers in hovering mode in wind gust conditions.

Appendix R

Position error in Dryden wind conditions

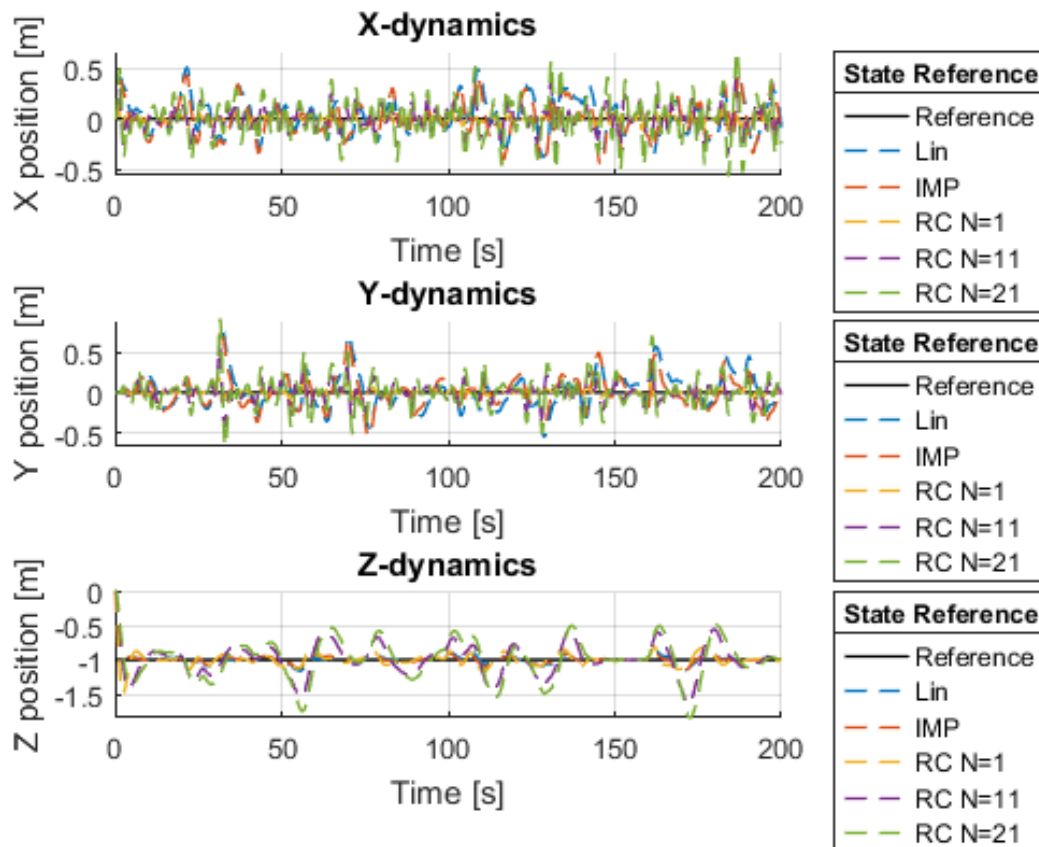


Figure R.1: System dynamics for various controllers in hovering mode in Dryden wind conditions.

Appendix S

Position error tracking challenging reference

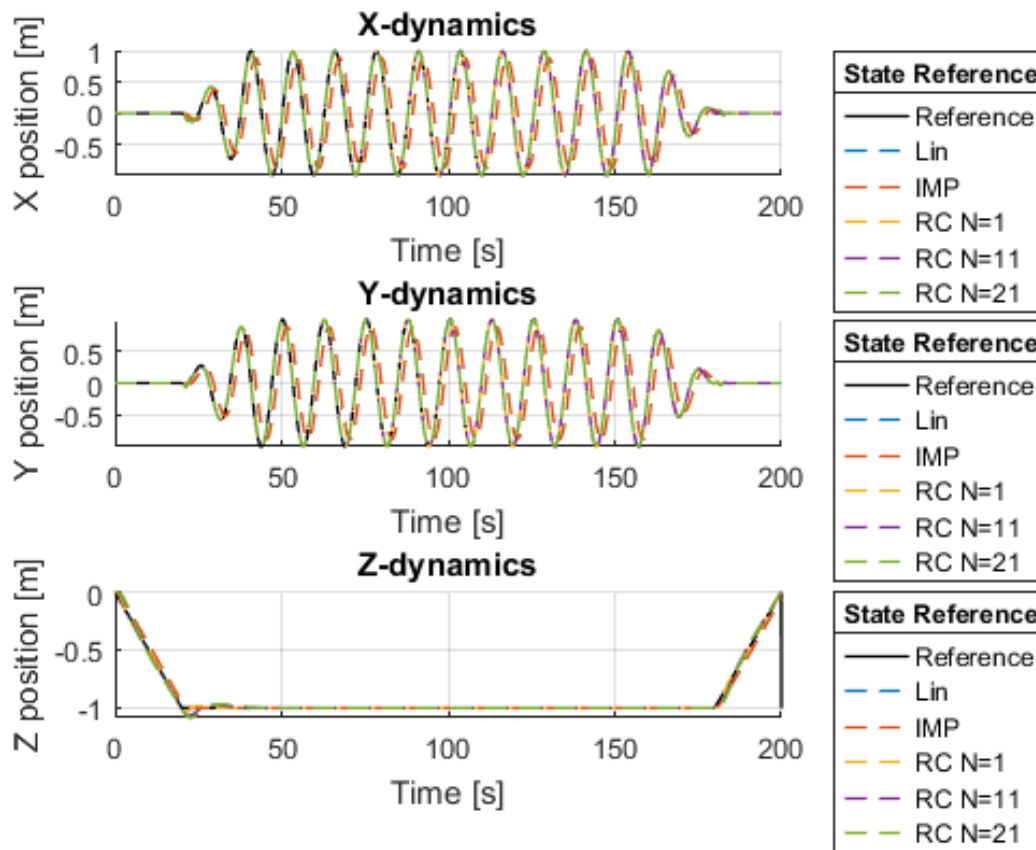


Figure S.1: System dynamics for various controllers tracking challenging reference without wind.