Vuk Krivokapic

Automatic Landning of Multi-Rotor on Moving Platform

Masteroppgave i Kybernetikk og Robotikk
Veileder: Tor Arne Johansen
Co-veileder: Martin Lysvand Sollie
Juni 2019

Vuk Krivokapic

Automatic Landning of Multi-Rotor on Moving Platform

Masteroppgave i Kybernetikk og Robotikk
Veileder: Tor Arne Johansen
Co-veileder: Martin Lysvand Sollie
Juni 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU **Norwegian University of** **Science and Technology**	**Faculty of Information Technology** **and Electrical Engineering** **Department of Engineering Cybernetics**

PROJECT DESCRIPTION SHEET

Name:	Vuk Krivokapic
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Automatisk landing av multirotor på bevegelig plattform
Thesis title (English):	Automatic landing of multi-rotor on moving platform

Thesis Description: One of the most dangerous parts of an offshore autonomous unmanned aerial vehicle (UAV) mission is the landing. To perform a safe landing in rough weather conditions, reliable control and error managing algorithms are crucial. It is also important to know the limits of the system, in order to set necessary weather requirements for a successful mission.

The goal of this project is to develop reliable control algorithms possible to carry out a safe landing in rough weather conditions and test the algorithm performance, both in simulations and in the field. In addition, limits of the developed system has to be tested.

The following tasks should be considered:

1. Perform a literature review on UAV modeling and control system design.
2. Develop control and landing algorithms.
3. Implement a simulation environment for UAV landing on a moving platform in DUNE. Simulate landing sequences for several weather conditions and find weather requirements that has to be fulfilled in order to perform a safe landing. Find minimal size requirements for the landing platform.
4. Build the field testing system by connecting all necessary hardware. Install software needed for the communication.
5. Test the developed and simulated algorithms in the field, using the Real Time kinematics (RTK) positioning. Compare test results with simulation results and check if the requirements from the simulation are match requirements found during field testing.
6. Discuss results and error sources.
7. Conclude your results and suggest future work.

Start date:	2019-01-27
Due date:	2019-06-03

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Tor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor:	PhD Cand. Martin L. Sollie, Dept. of Eng. Cybernetics, NTNU

Abstract

The use of unmanned aerial vehicles (UAV) in autonomous offshore missions has increased drastically in recent years. The UAVs have made it possible to carry out missions in rough weather conditions, without risking human injuries. The main parts of a UAV mission are takeoff, cruise and landing. All three parts are equally important to carry out a successful autonomous mission. This thesis presents the development of a system for autonomous UAV landing based on the software developed by Laboratório de Sistemas e Tecnologia Subaquática (LSTS).

All software and hardware needed for the landing system, and connection between those are presented in the thesis. Mathematical equations of the hexarotor model and environmental disturbances affecting the landing process are derived. High-level velocity control algorithms are developed and used as input to ArduPilot, the low-level control software. Several error handling algorithms are developed, in order to make sure that the landing is carried out efficiently and safely.

The simulation environment is implemented in DUNE: Unified Navigation Environment. Simulations are used to test the performance of developed algorithms, and prepare for the field test. Results of the simulations are presented and discussed in the thesis.

Two field tests are performed. The tests are performed using a 3DR Hexacopter, with DUNE running on a BeagleBone black and ArduPilot running on a PixHawk. All algorithms used for simulations are used in the field as well. Results from the field test are presented, discussed and compared with the simulation results. The vehicle was able to land in the field test environment. Some deviations in the developed algorithms are noticed and a proposal for further development is made.

Sammendrag

Bruk av ubemannede luftfartøyer (UAV) i autonome offshore-oppdrag har økt drastisk de siste årene. UAVene har gjort det mulig å utføre oppdrag i tøffe værforhold uten å risikere menneskeskader. Hoveddelene av et UAV oppdrag består av letting, cruise og landing. Alle tre delene er like viktige i utførelsen av ett autonomt oppdrag. Denne oppgaven presenterer utvikling av et system for autonom UAV landing basert på programvarene utviklet av Laboratório de Sistemas e Tecnologia Subaquática (LSTS).

All programvare og maskinvare som trengs for landingssystemet, og forbindelsen mellom disse presenteres i avhandlingen. Matematiske ligninger av hexarotormodellen og miljøforstyrrelser som påvirker landingsprosessen er avledet. Hastighetsstyringsalgoritmer er utviklet og brukt som input til ArduPilot, programvaren for lavnivåkontroll. Flere feilhåndteringsalgoritmer er utviklet for å sikre at landingen utføres effektivt og trygt.

Simuleringsmiljøet er implementert i DUNE: Unified Navigation Environment. Simuleringer brukes til å teste ytelsen til de utviklede algoritmene, og forberedelsene til forsøkene i feltet. Simuleringsresultatene er presentert og diskutert i avhandlingen.

To feltforsøk er utført. Forsøkene er utført med et 3DR Hexacopter, med DUNE som kjører på en BeagleBone black og ArduPilot som kjører på en PixHawk. Alle algoritmer som brukes i simuleringene, brukes også i feltet. Resultater fra feltforsøkene er presentert, diskutert og sammenlignet med simuleringsresultatene. Luftfartøyet var i stand til å gjennomføre landing i feltforsøkene. Noen avvik i de utviklede algoritmene er loppdaget, og et forslag til videreutvikling er utarbeidet.

Preface

The work presented in this thesis concludes my master degree in Cybernetics and Robotics at Norwegian University of Science and Technology (NTNU). This thesis builds on the specialization project, written by the author the previous semester, where the main focus was building a simulation environment in Matlab.

All hardware needed to carry out the work presented in the thesis is provided by the UAVlab at NTNU. The software used to build the system is developed by the Laboratório de Sistemas e Tecnologia Subaquática (LSTS), Porto. Besides the physical laboratory, there exists a community at the UAVlab sharing previous projects and helping new students to get started with their projects. Being a part of such community helped a lot, as information about LSTS software was hard to find because of the limited use. The physical laboratory was available for use during the entire project.

I would like to thank my co-supervisor Martin L. Sollie for helping me connect all hardware and install the software necessary to make the system work, as well as giving me valuable advice through the entire semester. I would also like to thank my supervisor Tor Arne Johansen for making this project possible, and for providing fast answers to my questions. Artur Piotr Zolich helped me organizing and preparing field tests, which I appreciate. Finally, I would like to thank Pål Kvaløy and Henricus van Rijt for being patient with me and helping me during the field tests.

Vuk Krivokapic *Trondheim, June 2019*

Table of Contents

List of Figures

List of Tables

Abbreviations

AUV	Autonomous Underwater Vehicle
DUNE	DUNE: Unified Navigation Environment
ECEF	Earth-Centered, Earth-Fixed
etc	Et cetra
GLONASS	Globalnaja Navigatsionnaja Sputnikovaja Sistema
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
HIL	Hardware-in-the-loop
I2C	Inter-Integrated Circuit
IMC	Inter-Module Communication
LOS	Line-of-Sight
LQR	Linear Quadratic Regulator
LSTS	Laboratório de Sistemas e Tecnologia Subaquática
MPC	Model Predictive Control
NED	North-East-Down
NTNU	Norwegian University of Science and Technology
PID	Proportional Integral Derivative
PIV	Proportional Integral Velocity
PM Spectrum	Pierson-Moskowitz Spectrum
RTK	Real-Time Kinematic
SIL	Software-in-the-loop
SPI	Serial Peripheral Interface Bus
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Arial Vehicle

Chapter 1

Introduction

1.1 Motivation

In recent years, the use of UAVs has increased significantly, both onshore and offshore. The autonomous vehicles are performing tasks that earlier were difficult and dangerous for human beings. Offshore missions are often marked as difficult and dangerous, as the weather conditions can get rough when the location is far from the coastline. UAVs have increased the weather condition threshold on offshore missions, but the efficiency of the vehicles are far from perfect.

One of the most critical parts of a UAV in offshore missions is the landing. Rough weather conditions imply strong wind, bad visibility, and high waves. Therefore, it is important to design a robust control system that can handle all the difficulties rough weather conditions bring, and carry out a safe landing. Today's control systems, designed for offshore UAV landings, are giving satisfying results, but there is still room for improvements, especially on wind and wave prediction.

Following the increasing usage of the UAVs, NTNU has established its own UAV laboratory. The UAVlab is mainly used by master and Ph.D. students. Many research missions are initiated in order to find solutions to the control problems associated with UAVs. Every student associated with the UAVlab is contributing to a common solution to UAV problems. A huge motivation for this task is to make a contribution to the UAV lab community at NTNU, which can be used to help others solve their problems.

1.2 Specialization Project

This project is a continuation of the specialization project, written the semester before [1]. The specialization project emphasizes modeling of the hexacopter and environmental disturbances. Control and landing algorithms are developed in the project as well. All simulations in the specialization project are done in Matlab. Proportional - Integral - Velocity (PIV) control is used for the horizontal control, while Proportional - Integral - Derivative (PID) control is used for the vertical control. The air force modeling presents a problem in the specialization project, as the roll and pitch response behaves strangely when the constant wind is present.

There were developed three landing algorithms in the specialization project: state machine algorithm, parameter allocation algorithm, and landing timing algorithm. The parameter allocation algorithm requires two sets of parameters, depending on the desired controller aggressiveness level. The state machine and the landing timing algorithms are responsible for the error handling during a landing procedure. The error is measured using a set of requirements, determined beforehand. It is suggested to investigate the algorithms, as they together led to too many aborted landings.

In addition to the high-level controllers, there were developed low-level control algorithms in the specialization project. The algorithms are controlling roll and pitch movement, using PID control. A systematic tuning approach is suggested for the control algorithms, as they are tuned by a trial-error approach.

A list for future work is made at the end of the specialization project:

- Investigate modeling of air forces
- Tune horizontal controllers by using a systematic tuning approach
- Investigate other control strategies
- Do a stability analysis of the whole system
- Improve the landing permission algorithm
- Investigate the possibility of developing a wave estimator
- Adjust landing boundaries

1.3 Related Work

The research work regarding modeling, controller choice and landing algorithms is done in the specialization project [1]. Most of the research for this project is associated with the implementation part, including use of software such as DUNE, Neptus, ArduPilot, and RTKLIB.

A thesis about the automatic landing of a X8 Skywalker flying-wing into a net is written in [2]. The interesting part of the thesis is that it is describing software and hardware that is used in this project. Software and hardware communication is described in detail. The same software is used by [3] and [4]. [3] is using the software for nature research in Arctic areas, while [4] is using the software for search and rescue missions. [4] is having problems with DUNE acting differently on experimental testing compared to simulations, and suggest a deeper investigation of the problem.

An example of ArduPilot software use can be found in [5] and [6]. The software - in - the - loop simulations, which are going to be used in this project, is the reason for ArduPilot use in [5]. The ArduPilot is used as an autopilot for dynamic drone positioning in [6], which is also suggesting the use of RTK GPS together with the ArduPilot.

A detailed description of the RTKLIB, with a performance test, is presented in [7]. The RTKLIB is also investigated in [8], where the RTK GNSS performance is compared to the IMU performance. In addition, [8] uses DUNE for Hardware - in - the - loop testing.

A total system for multirotor landing on a moving platform is developed by [9]. The author is using a vision system to detect the landing platform and PID algorithms for control of the multirotor. An interesting idea of the vehicle flying through different states on the way down to the platform is also introduced in [9] and [10]. A work on autonomous landing on moving platform using Model Predictive Control (MPC) is done by [11].

1.4 Outline

Chapter 2 - System Overview: The chapter gives an overview of software and hardware forming the system, and connection between those.

Chapter 3 - Modeling: Mathematical models of forces and torques acting on the system are presented. Environmental disturbances modeling is also presented in the chapter. Big parts of the chapter are written in the specialization project.

Chapter 4 - Control: Presents development of the control algorithms used in the project.

Chapter 5 - Implementation: This chapter presents implementation of the simulation environment and controllers in DUNE. The landing algorithms are also described in the chapter.

Chapter 6 - Simulation: Simulation results of the system and tuning of the controllers are presented in this chapter.

Chapter 7 - Field Testing: Presents field test results.

Chapter 8 - Discussion: This chapter discusses results from Chapter 6 and Chapter 7.

Chapter 9 - Closing Remarks: Concludes the work and results. A suggestion for future work is also presented in this chapter.

Chapter 2

System Overview

2.1 Software

2.1.1 DUNE

DUNE is a software that is a part of the LSTS toolchain, developed by the University of Porto [12].

The software is designed for communication between different *tasks*. The tasks are different programs with specific functions. Typical functions for a task are: sensor reading, sensor data management, control, maneuvering, mission planning and communication between other tasks [13].

In DUNE, the connection between tasks is done through configuration files. Configuration files include all tasks that are supposed to communicate with each other. The communication happens through IMC messages, described later in this Section. The tasks are dispatching and consuming specific IMC messages. The IMC messages from a specific task are available for all other tasks that are included in the same configuration file.

It is also common to include other configuration files to a configuration file. The result is a hierarchy of several configuration files, which often makes troubleshooting easier. Another benefit of having a hierarchy of configuration files is that it gets easier to remove and add parts of the system, simply by removing and adding the specific configuration file in the hierarchy.

DUNE is running on an onboard computer, both on the UAV and the base station. In addition, it is used for the simulation.

2.1.2 IMC

IMC (Inter-Module Communication) protocol is a protocol designed by LSTS to build reliable real-time communication between the DUNE tasks.

As mentioned in Section 2.1.1, IMC messages are used to send information between DUNE tasks that are included in the same configuration file. All IMC messages available for a system are listed in a .xml file. A developer can easily create new messages to the system by adding the message information to the .xml file.

2.1.3 Neptus

Neptus is a ground station software used to monitor and control vehicles that communicates by IMC messages [14].

The monitoring is performed by utilizing a specific IMC message that sends the exact vehicle position and orientation. Neptus shows the position graphically on a world map pre-chosen by the user (Figure 2.1).

Mission planning is an important part of the Neptus software. The user can click on the different points on the map and plan a desired path for the next mission. Besides, the user can also choose which of the available controllers that are going to be used between the different parts of the mission. It is worth mentioning that the user has full control of a mission from Neptus. By clicking on different buttons presented on the screen, one can switch, edit or abort the mission.

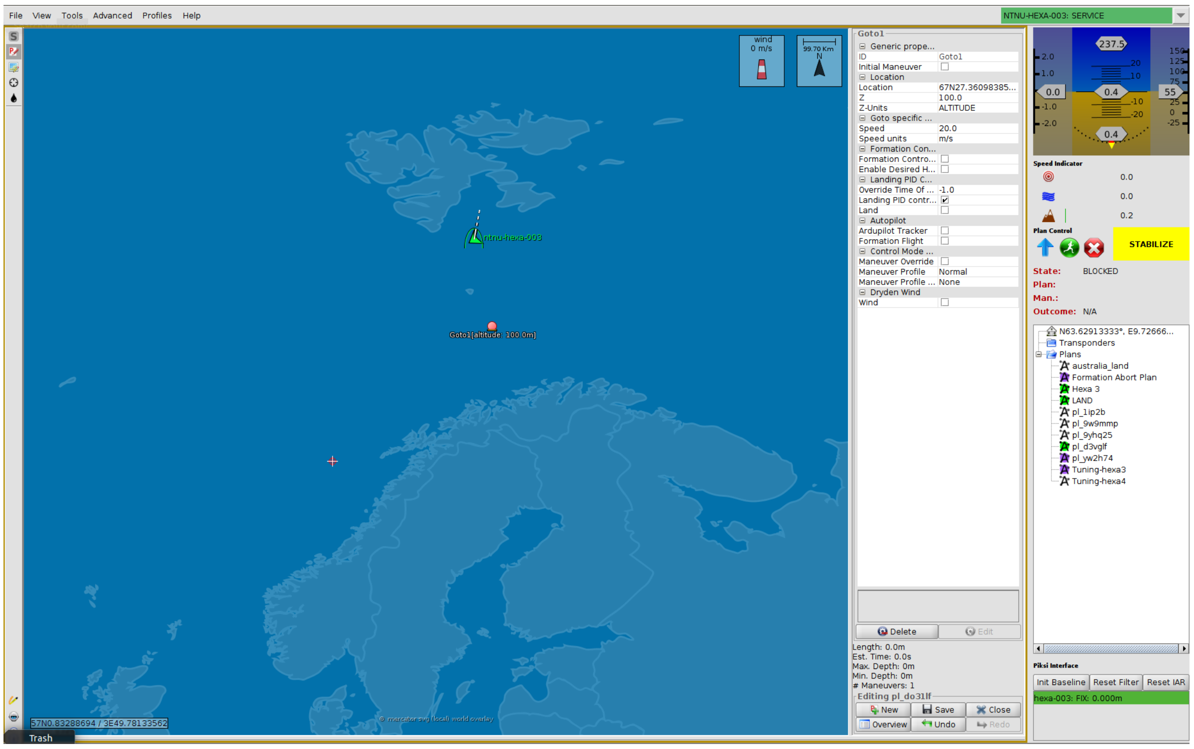


Figure 2.1: Neptus interface

Multiple vehicle control is another possibility that can be carried out in Neptus. It involves control of several vehicles at once, which is used a lot in this project because the UAV is one moving vehicle supposed to land on a platform that can be considered as another moving vehicle[15].

Neptus also provides a mission analysis tool (MRA). The MRA software lists up a log of all active IMC messages during a mission. The IMC messages can further be plotted, which makes the analysis easy. It also provides a possibility of exporting IMC logs to different formats, where Matlab (.mat) is one of the possible formats.

2.1.4 ArduPilot

ArduPilot is an open-source autopilot developed for unmanned vehicles. An onboard computer on the vehicle is running ArduPilot software. The ArduPilot provides reliable real-time communication between the vehicle and the ground station. The software can also be used for simulation.

This project uses ArduPilot, both for simulation and field tests. The main task of the ArduPilot in this project is converting velocity control commands to Euler angles, and actual angular velocities of the motors. An explanation follows later in the thesis [16].

2.1.5 Real-Time Kinematic Positioning

Real-Time Kinematic (RTK) positioning is the position measurement technology used in this project. The technology is using two receivers, one mounted on the base station and the other mounted on the rover. The reason for the use of RTK positioning in this thesis is the precise relative positioning the technology can provide. By differencing receiver measurements, a lot of signal errors are eliminated. Equation 2.1 shows differentiation of receiver measurements. All errors associated with GPS positioning, presented in (2.1), are explained on page 136 in [17].

$$\begin{aligned}
\Delta\phi &= \phi_{rover} - \phi_{base} && \text{(2.1a)}\\
&= \frac{\Delta r}{\lambda} + \Delta N - \cancel{\frac{c}{\lambda}\Delta T_{iono}} + \cancel{\frac{c}{\lambda}\Delta T_{tropo}} - \cancel{\frac{c}{\lambda}\delta t_{sat\ clock}} + \frac{c}{\lambda}\Delta\delta t_{rec\ clock} && \text{(2.1b)}\\
&- \cancel{\Delta\delta\phi_{sat\ bias}} + \Delta\delta\phi_{rec\ bias} + \Delta\delta\phi_{LOS\ dependent} + \Delta\delta\phi_{multipath} + \Delta\omega\\
&= \frac{\Delta r}{\lambda} + \Delta N + \frac{c}{\lambda}\Delta\delta t_{rec\ clock} + \Delta\delta\phi_{rec\ bias} + \Delta\delta\phi_{LOS\ dependent} + \Delta\delta\phi_{multipath} + \Delta\omega && \text{(2.1c)}
\end{aligned}$$

where

$$\begin{aligned}
\phi &= \text{measurement}\\
\lambda &= \text{carrier wavelength}\\
c &= \text{vacuum speed of light}\\
r &= \text{distance to satellite}\\
T_{iono} &= \text{ionospheric delay}\\
T_{tropo} &= \text{tropospheric delay}\\
t_{rec\ clock} &= \text{receiver clock error}\\
t_{sat\ clock} &= \text{satellite clock error}\\
\phi_{sat\ bias}, \phi_{rec\ bias} &= \text{carrier phase delays with respect to code, independent of LOS}\\
\phi_{LOS\ dependent} &= \text{LOS-dependent phase wind up error}\\
\phi_{multipath} &= \text{multipath error}\\
\omega &= \text{tracking noise}
\end{aligned}$$

Further, the RTK subtracts satellite measurements. This is called double differencing and is showed in (2.2). Double differencing removes systematic errors, common to several satellites and receivers.

$$\begin{aligned}
\nabla\Delta\phi &= \Delta\phi_{sat1} - \Delta\phi_{sat2} && \text{(2.2a)}\\
&= \frac{\Delta r}{\lambda} + \nabla\Delta N + \cancel{\frac{c}{\lambda}\nabla\Delta\delta t_{rec\ clock}} + \cancel{\nabla\Delta\delta\phi_{rec\ bias}} + \nabla\Delta\delta\phi_{LOS\ dependent} && \text{(2.2b)}\\
&+ \nabla\Delta\delta\phi_{multipath} + \nabla\Delta\omega\\
&= \frac{\nabla\Delta r}{\lambda} + \nabla\Delta N + \nabla\Delta\omega && \text{(2.2c)}
\end{aligned}$$

Double differencing and correct integer ambiguities eliminates errors, such that the mm/cm procession level is reached. It is important to mention that the precision level only holds for the rover measurements relative to the base, and not rover position in general [18].

2.1.6 RTKLIB

RTKLIB is an open-source program package. It is developed to convert raw GNSS data to a readable position. It provides both real-time processing and post-processing of the GNSS data. The RTKLIB is running on the onboard computer[19].

2.2 Hardware

2.2.1 Hexacopter

The vehicle used in this project is a 3DR Hexacopter, developed and manufactured by 3D Robotics. The UAV frame is made of aluminium, which makes it robust and suitable for development projects. Specifications of the hexacopter can be found in Table 2.1[20].

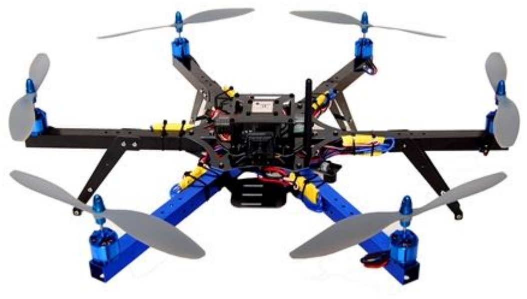


Figure 2.2: Hexa 3DR

Table 2.1: Hexacopter parameters

Frame weight without electrics	1.36 kg
Fully loaded frame weight	2.73 kg
Frame length	0.4 m
Frame width	0.28 m
Frame heigth	0.18 m
Arm length	0.30 m

2.2.2 PixHawk

PixHawk is an open-hardware project provided by 3D robotics. The PixHawk version used in this project is PixHawk 1. The PixHawk is designed to support several autopilot software. In this project, the ArduPilot is running on the PixHawk (Section 2.1.4)[21].

Figure 2.3: PixHawk 1

The PixHawk provides several communication possibilities. It provides support for protocols, such as: SPI, CAN, USB, UART, and I2C. The UART is the communication protocol used mostly in this project. The connection between Beaglebone and PixHawk is going through the UART protocol.

2.2.3 Beaglebone Black

BeagleBone Black is a low-cost, open-source computer produced by the American manufacturer Texas Instruments. It has 512 MB RAM and has a flash memory size of 2 GB. The BeagleBone Black runs Linux Kernel version 3.14 [22]. In this project, the BeagleBone black is used to run DUNE (Section 2.1.1) and RTKLIB (Section 2.1.6) on-board.

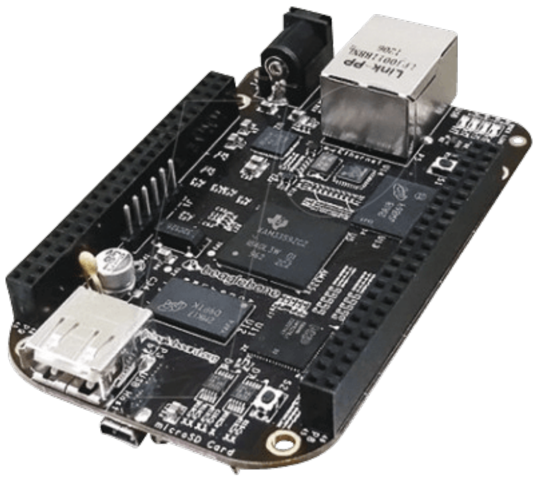


Figure 2.4: BeagleBone black

2.2.4 Cape

A cape for the BeagleBone black is developed at the UAVlab at NTNU using Circuit Studio software[23]. The cape is added to the BeagleBone in order to distribute 5V battery power to other onboard components and provide contacts with lock for I/O.

2.2.5 GNSS Antennas

On-board on the UAV a HX-CH3602A antenna is used. It can receive signals from three different constellations (GPS, GLONASS and BDS). It can only receive L1 frequencies from GPS and GLONASS[24].

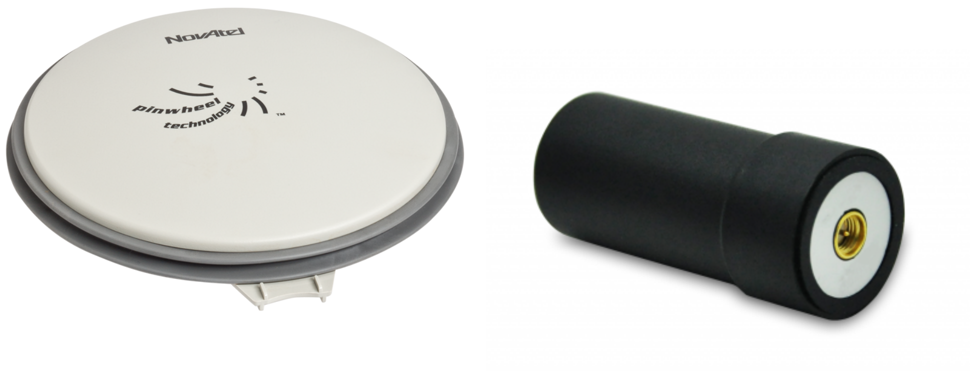


Figure 2.5: Novatel GPS-702-GG and HX-CH3602A

The ground station, described later in this section, uses a Novatel GPS-702-GG antenna. It has better performance than the onboard antenna. The ground station antenna supports dual-frequency GPS and GLONASS signals, which means that it can receive signals from both L1 and L2 frequencies. It also supports Galileo and BeiDou signals.

2.2.6 Rocket

Ubiquti Rocket M5 is used for the radio communication between the vehicle and the ground station. The rockets communicate with a frequency of 5GHz [25]. A user manual follows with the rockets, making the configuration and setup easy.

Figure 2.6: Ubiquti M5 rocket

2.2.7 Ground Station

The ground station is equipped with a BeagleBone black with a cape, router, rocket, and a GNSS antenna. All communication between the user and the UAV goes through the ground station. The router on the ground station is working like a switch, distributing signals between the components. The user PC is directly connected to the router during a mission. The IMC messages, sent in between BeagleBones in the system are also distributed from the router.

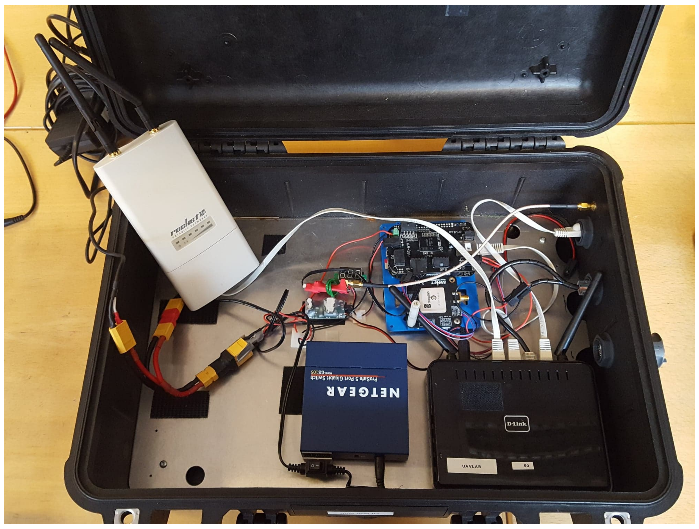


Figure 2.7: Ground Station

2.3 Communication

All components mentioned in Section 2.2 and Section 2.1 constitute a complex system for simulation and control of an UAV. The system can be divided into two parts, the simulation and the field testing system, as they are different on several points.

2.3.1 Simulation System

The main purpose of a simulation is to test different scenarios as close to reality as possible, and avoid damaging hardware. Therefore, the simulation system only deals with the communication between software mentioned in Section 2.1. The UAV simulations are running Software-in-the-Loop (SIL), meaning that a series of software are running together, in a loop, to create a simulation.

DUNE tasks (Section 2.1.1) are constantly running during a UAV simulation. Tasks with different purposes are communicating through IMC messages (2.1.2). There are developed tasks for the UAV control. The tasks that are responsible for the control are sending velocity control commands to the ArduPilot via MAVLink, a protocol developed for the drone communication[26]. ArduPilot transforms the velocity control commands to actual motor rotations and sends the position of the vehicle back to DUNE, also via MAVLink. DUNE tasks convert MAVLink position messages received from the ArduPilot to the IMC messages. The messages are read by Neptus, and the position is then presented graphically.

To get a better understanding, one can follow a specific IMC messages. Control tasks in DUNE are developed to send velocity commands to the tasks that are developed for ArduPilot communication. The commands are sent via *IMC::DesiredVelocity* message. The message consists of four parts: timestamp, velocity in the x-direction of the NED frame (explained later in the report), velocity in the y-direction of the NED frame and velocity in the z-direction of the NED frame. The task transforms *IMC::DesiredVelocity* message to MAVLink message, and sends information to the ArduPilot via TCP protocol. Further, ArduPilot transforms control commands to actual vehicle movement and sends back the vehicle position to the same communication task in DUNE. The DUNE communication task transforms the position message to *IMC::EstimatedState* message, which contains information about vehicles geodetic position. *IMC::EstimatedState* message gets sent back to the control tasks, and used by the controller. It is also transported to Neptus, via several other DUNE transport tasks, and represented graphically.
Figure 2.8 describes the communication system during a simulation.

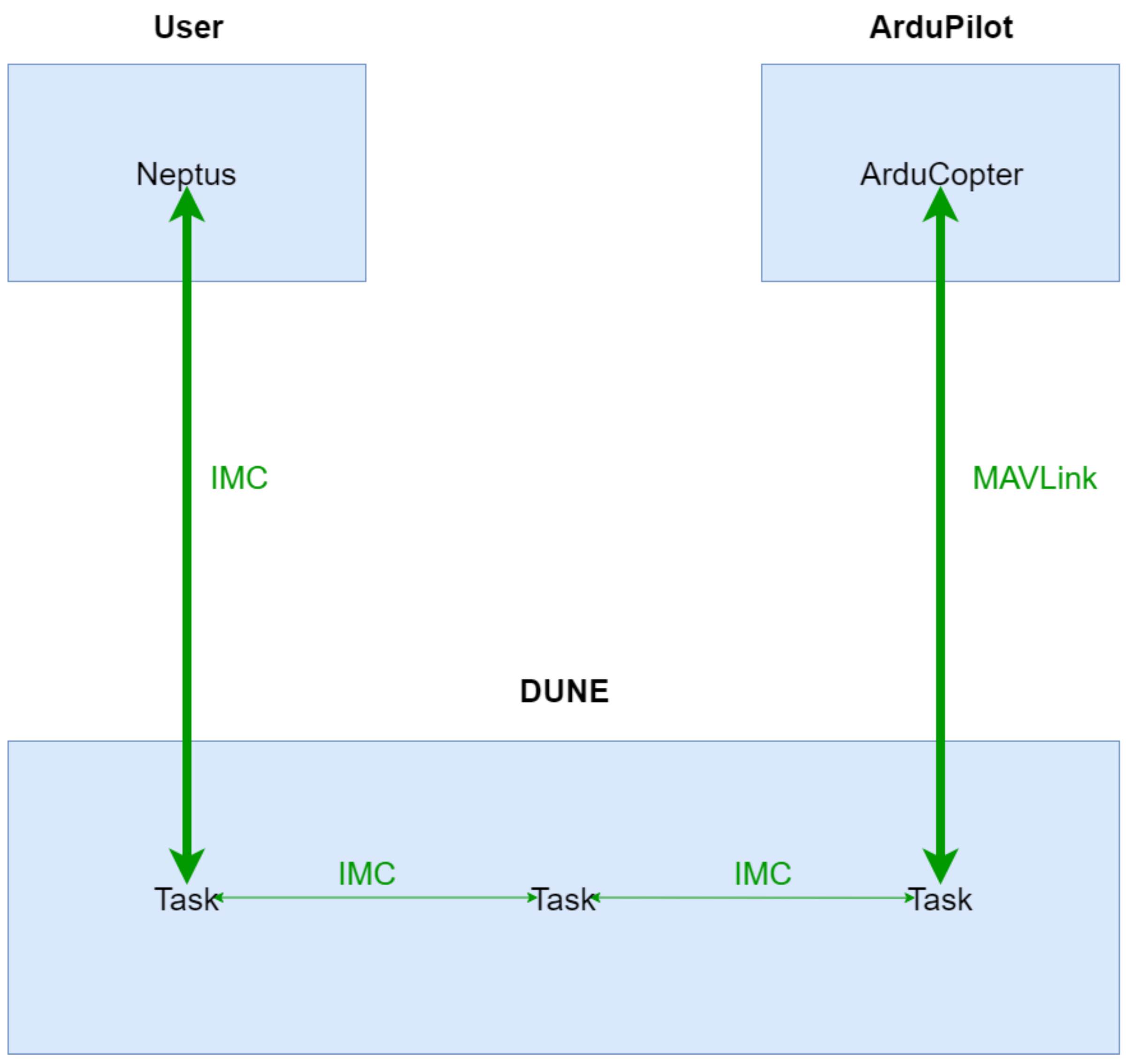


Figure 2.8: Communication during a simulation

It is important to mention that other types of control commands like force and acceleration can be produced in DUNE as well. The reason why velocity commands are chosen in this project is that the original ArduPilot firmware does not support force and acceleration control commands.

2.3.2 Total System

The total communication system is the system that is going to be used in real-life operations. The communication system is more complex than the communication system during a simulation. The reason is that the hardware communication has to be included as well. A communication system containing several hardware components are vulnerable to errors.

The raw GNSS measurements are transformed to position data using RTKLIB (Section 2.1.6), which is installed on the BeagleBone black (Section 2.2.3), both on the vehicle and the base station. The reason why raw GNSS measurements are sent to both vehicle and the base station is the working principle of the RTK positioning, that is described later in the report.

DUNE is running on both BeagleBones in the system. The control system, pre-tested in simulations, is transferred to the BeagleBone black board on the vehicle. Specific DUNE tasks on the board are transforming GNSS position measurements to the *IMC::EstimatedState* message used further in the DUNE system, as described in Section 2.3.1. The control system running on-board is consuming *IMC::EstimatedState* messages and calculating new velocity control commands, dispatched as *IMC::DesiredVelocity* messages (Section 2.3.1). The *IMC::DesiredVelocity* are consumed by DUNE tasks responsible for the communication with the ArduPilot, and transformed to MAVLink messages. Further, MAVLink messages are sent to the ArduPilot, that is running on a PixHawk onboard, via UART Protocol. The ArduPilot is then controlling the UAV motors through the PixHawk.

At any time, the UAV is connected to the ground station via a Rocket radio. The ground station is connected to the user PC, containing Neptus, via ethernet. Neptus reads *IMC::EstimatedState* messages from the BeagleBone black on the ground station, and shows the position graphically. Missions can also be planned on Neptus, on the user PC, and sent back to the UAV.

The whole communication system for field testing is shown in Figure 2.2.

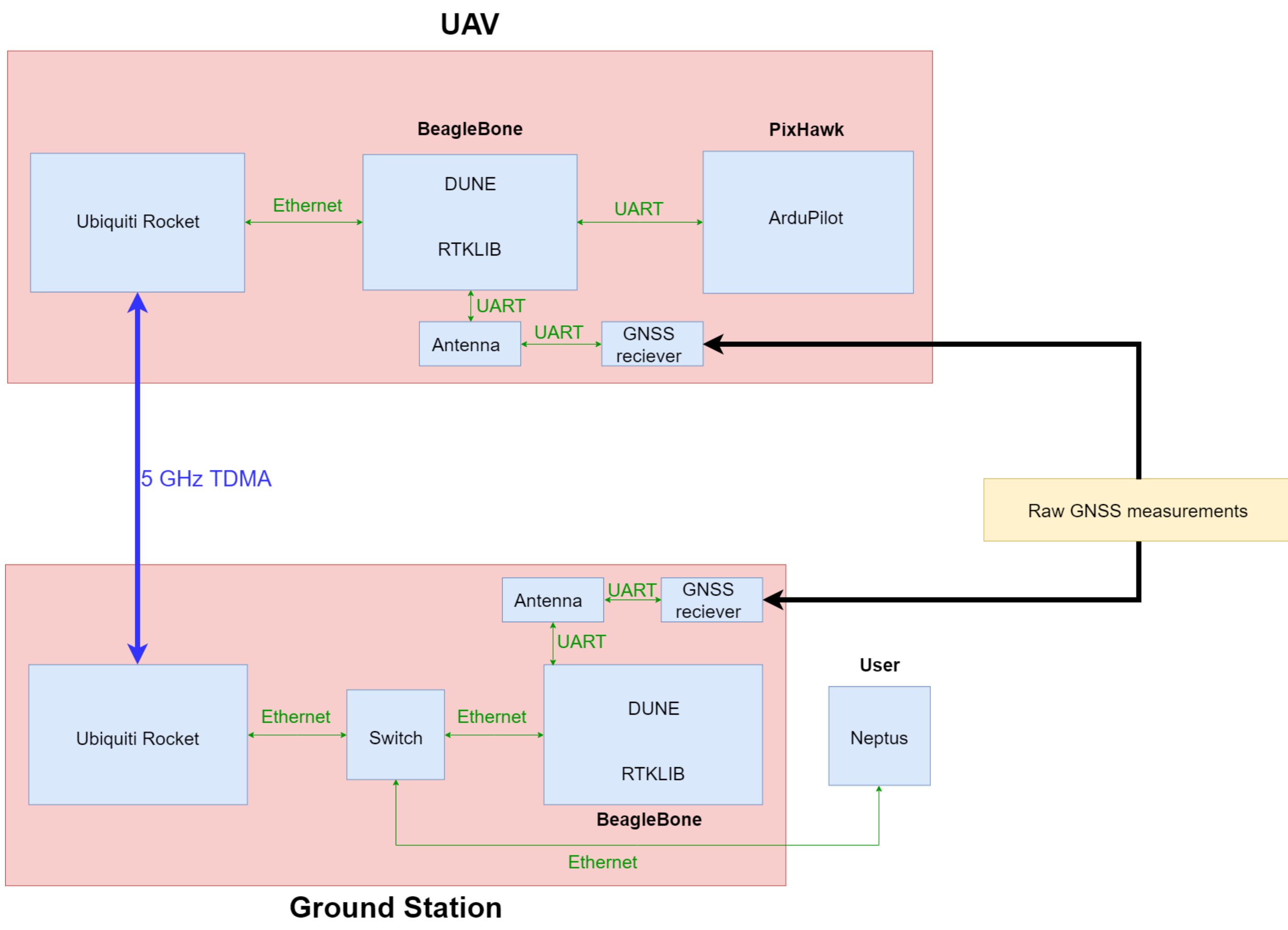


Figure 2.9: Hardware communication

Chapter 3

Modeling

3.1 Theory

3.1.1 Coordinate Frames

A coordinate system consist of two or three axes, depended on the dimension of the coordinate system. The position of an object within a coordinate system can be described by coordinates of that specific system. A vehicle is often described as an own frame, called the vehicle body-frame. In the vehiclew body-frame the x-axis points out the front of the vehicle, the z-axis points straight up, whiler the y-axis completes the right-hand coordinate system. Environmental forces acting on the vehicle are often described relative to the vehicle body frame, like in [17], while vehicle movement is described relative to another frame. For example, the movement of a long-distance airplane can be described relative Earth-Centered-Earth-Fixed frame (ECEF). The origin of the ECEF frame is at the Earth center, while the x-axis is pointing at the cross point between zero-meridian and equator. On the other hand, wind forces acting on the airplane are described relative to the airplane body frame. A transformation between frames can be done to describe environmental forces acting on the airplane relative to the ECEF frame.

North-East-Down (NED) is a coordinate frame used for local navigation. The x-axis of the NED frame is pointing north, the z-axis is pointing to the earth center, while the y-axis is completing the right-hand coordinate system. When the NED frame is used for local navigation, which is the case in this project, all forces are described relative to the NED frame. The frame can not be used for global navigation because the relation between north and earth center varies depending on the position on the earth surface. Figure 3.1 illustrates an example of NED frame relative to ECEF frame.

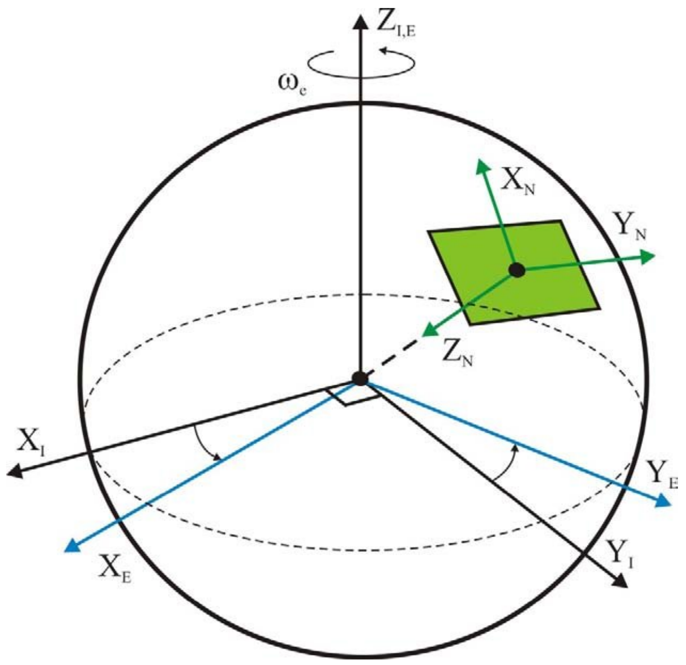


Figure 3.1: Coordinate frames

3.1.2 Rotation

The rotation of a coordinate frame relative to another can be described as a series of rotations of every single axis. Explained, rotation from a given frame A to a given frame B can be described as rotation about x, y and z-axis of frame A relative to frame B. Rotation about coordinate axis can be described with following matrices:

$$\begin{aligned}
\mathbf{R}_x &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \\
\mathbf{R}_y &= \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \\
\mathbf{R}_z &= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.1}$$

Total rotation matrix is,

$$\mathbf{R}_a^b = \mathbf{R}_z \mathbf{R}_y \mathbf{R}_z \tag{3.2}$$

The following rule holds for rotation matrices.

$$\left(\mathbf{R}_a^b\right)^{-1} = \left(\mathbf{R}_a^b\right)^T = \mathbf{R}_b^a \tag{3.3}$$

3.1.3 Transformation

The transformation of a point from one frame to another can be described as a transformation matrix. The transformation matrix contains the total rotation matrix (3.2) and coordinates of the point in the original frame. Equation 3.4 describes transformation between a point from frame A to frame B.

$$\mathbf{T}_a^b = \begin{bmatrix} & \mathrm{x}^a \\ \mathbf{R}_a^b & \mathrm{y}^a \\ & \mathrm{z}^a \\ 000 & 1 \end{bmatrix} \tag{3.4}$$

3.1.4 Pierson-Moskowitz Spectrum

Pierson-Moskowitz Spectrum (PM spectrum) is a model for energy distribution in the sea, developed in 1964. An assumption made for the model is that wind blows steady for a long time over a large area, and that the waves will eventually reach the point of equilibrium with the wind. Model is depended on two parameters, A and B.[27][28]

$$\begin{aligned} A &= 8.1 \cdot 10^{-3} \cdot g^2 \\ B &= 0.74(\frac{g}{V_{19.4}})^4 = \frac{3.11}{H_s^2} \end{aligned} \tag{3.5}$$

where $V_{19.4}$ is wind velocity at height of 19.4 meters, g is the gravity constant and H_s is the wave height. Wave height is depended on sea state, and can be roughly estimated using Table 8.5 in [28].

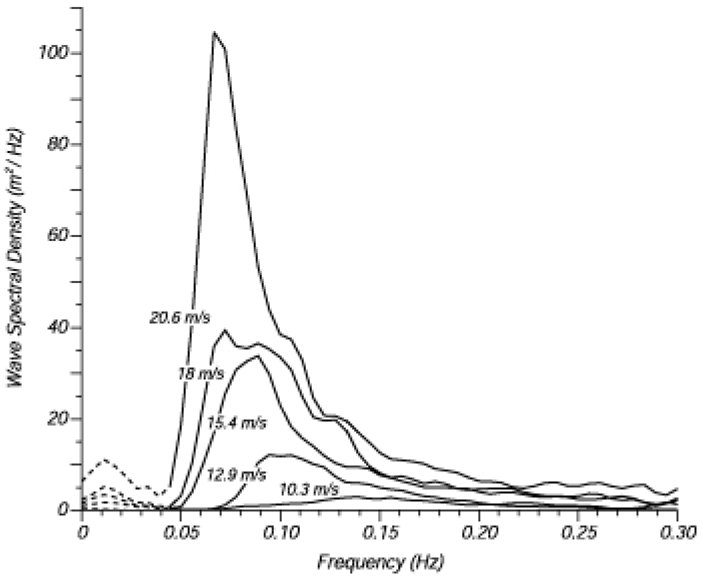


Figure 3.2: PM spectra for different $V_{19.4}$ velocities

Figure 3.2 shows fully developed PM spectra for different $V_{19.4}$ velocities.

3.2 Dynamics

Hexacopter dynamics can be derived from the Euler angles. The reason why this can be done is that the hexacopter is assumed as a symmetrical and rigid body.

3.2.1 Rotation

Rotation from body to NED frame is described by using series of rotations $\psi - \theta - \phi$, also known as yaw-pitch-roll. The result of the series of rotation is following rotation matrix:

$$\begin{aligned}\mathbf{R}_B^N &= \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \sin\theta\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\theta\sin\phi \\ \cos\theta\sin\psi & \sin\psi\sin\theta\sin\phi + \cos\theta\cos\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \sin\phi\cos\theta & \cos\theta\cos\phi \end{bmatrix}\end{aligned} \tag{3.6}$$

By using property from (3.3) it is possible to find a rotation matrix from NED to the body frame.

$$\mathbf{R}_N^B = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \sin\theta\cos\phi & \sin\psi\sin\theta\sin\phi + \cos\theta\cos\psi & \sin\phi\cos\theta \\ \cos\psi\sin\theta\cos\phi + \sin\theta\sin\phi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{3.7}$$

The angular velocity of the body frame relative to the NED frame is described using the same series of rotations.

$$\begin{aligned}\omega_{ib}^b &= \mathbf{R}_{x,-\phi}\mathbf{R}_{y,-\theta}\begin{bmatrix} 0 \\ 0 \\ \dot\psi \end{bmatrix} + \mathbf{R}_{x,-\phi}\begin{bmatrix} 0 \\ \dot\theta \\ 0 \end{bmatrix} + \begin{bmatrix} \dot\phi \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} -\sin\theta\dot\psi + \dot\phi \\ \sin\phi\cos\theta\dot\psi + \cos\phi\dot\theta \\ \cos\phi\cos\theta\dot\psi - \sin\phi\dot\theta \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \sin\theta \\ 0 & \cos\phi & \sin\theta\cos\phi \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}\begin{bmatrix} \dot\phi \\ \dot\theta \\ \dot\psi \end{bmatrix}\end{aligned} \tag{3.8}$$

3.2.2 Forces

All forces acting on the hexacopter can be decomposed in three directions, x,y, and z, of the NED frame. Beginning the force decomposition by decomposing gravity force, which is only acting in the z-direction of the UAV, decomposed in the NED frame.

$$F_{gravity} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{3.9}$$

When the propellers of the hexacopter are rotating, they are creating a force that makes hexacopter fly. That force is called *thrust*. The thrust force is directly depended on the speed of each propeller. Total thrust is the sum of all thrust forces produced by each propeller. Differences in the thrust produced by each propeller yield torque around the vehicle center of mass, resulting in rotation. The thrust force can be described as

$$F_{thrust} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{R}_B^I \sum_{i=1}^{6} b\Omega_i \end{bmatrix} \tag{3.10}$$

where b is the thrust constant in eq. 3.10.

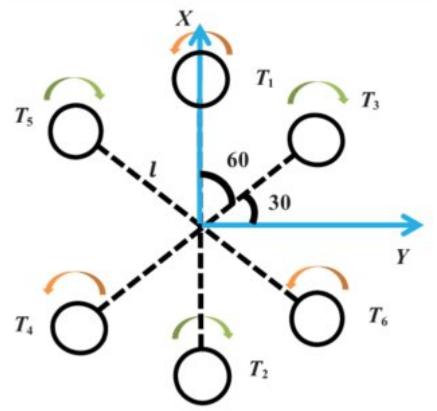


Figure 3.3: Thrust forces and rotation of propellers

The vehicle movement in the air creates drag forces in x, y, and z-direction. A mathematical model presenting the induced drag is,

$$F_{drag} = \begin{bmatrix} \frac{1}{2} C_d \rho V_x^2 A_x \\ \frac{1}{2} C_d \rho V_y^2 A_y \\ \frac{1}{2} C_d \rho V_z^2 A_z \end{bmatrix} \tag{3.11}$$

where C_d is the drag coefficient, found to be 1.05 by considering the shape of the hexacopter [29]. A_x, A_y and A_z are surface areas pointing in x,y and z-direction respectively, while ρ is the air density. V represents the air velocity in a given direction.

3.2.3 Torques

Torques are created by UAV's rotation about its body axis. Moments about the body axis of the UAV are created by adjusting the angular velocity of each propeller independently. Moments in roll, pitch, and yaw can be modeled from the geometrical structure of the hexacopter, and angular velocities of the propellers (Figure 3.3).

$$\tau_\phi = bl\big[-\Omega_2^2+\Omega_5^2+\frac{1}{2}(-\Omega_1^2-\Omega_3^2+\Omega_2^2+\Omega_6^2)\big] \tag{3.12}$$

$$\tau_\theta = bl\frac{\sqrt{3}}{2}(-\Omega_1^2+\Omega_3^2+\Omega_4^2-\Omega_6^2) \tag{3.13}$$

$$\tau_{psi} = d(-\Omega_1^2+\Omega_2^2-\Omega_3^2+\Omega_4^2-\Omega_5^2+\Omega_6^2) \tag{3.14}$$

where b is the thrust constant, l is the distance from propellers to the center of gravity and d is the drag factor.

3.3 Mathematical Model

The mathematical model of the hexacopter can be divided into two parts, translational dynamics and rotational dynamics. Translational dynamics represent hexacopter movement along x,y and z-axis of the inertial frame, while rotational dynamics represent rotation about x,y and z-axis of the body frame. The frame of the landing platform is considered as the NED frame in this project, which makes sense since all controllers, described later in the report, are designed to minimize the error between the inertial and body frame.

3.3.1 Translational dynamics

Translational dynamics are modeled by considering all forces acting on the body, described in Section 3.2.2.

$$ma = \sum F = F_{drag} + F_{thrust} + F_{gravity} \tag{3.15}$$

which gives following translational equations of motion:

$$\ddot{x} = \frac{1}{m}\left[(\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi)(\sum_{i=1}^{6} F_i) + f_{drag_x}\dot{x}\right] \tag{3.16a}$$

$$\ddot{y} = \frac{1}{m}\left[(\cos\phi\sin\psi\sin\theta - \sin\phi\cos\psi)(\sum_{i=1}^{6} F_i) + f_{drag_y}\dot{y}\right] \tag{3.16b}$$

$$\ddot{x} = \frac{1}{m}\left[(\cos\phi\cos\theta)(\sum_{i=1}^{6} F_i) + f_{drag_z}\dot{z}\right] \tag{3.16c}$$

3.3.2 Rotational Dynamics

Rotational dynamics are modeled by considering all moments acting on the body, described in Section 3.2.3.

$$\ddot{\phi} = \frac{1}{J_{xx}}\left[\dot{\theta}\dot{\psi}(J_{yy} - J_{zz}) + bl\Big[-\Omega_2^2 + \Omega_5^2 + \frac{1}{2}(-\Omega_1^2 - \Omega_3^2 + \Omega_2^2 + \Omega_6^2)\Big]\right] \tag{3.17a}$$

$$\ddot{\theta} = \frac{1}{J_{yy}}\left[\dot{\phi}\dot{\psi}(J_{zz} - J_{xx}) + bl\frac{\sqrt{3}}{2}(-\Omega_1^2 + \Omega_3^2 + \Omega_4^2 - \Omega_6^2)\right] \tag{3.17b}$$

$$\ddot{\psi} = \frac{1}{J_{zz}}\left[\dot{\phi}\dot{\theta}(J_{xx} - J_{yy}) + d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2 - \Omega_5^2 + \Omega_6^2)\right] \tag{3.17c}$$

Relation between control inputs and angular velocities of propellers

The system is controlled by four control inputs, u_1, u_2, u_3 and u_4. The first control input is used to control the vertical movement of the hexacopter, while the others are controlling roll, pitch and yaw movements, respectively.

Thrust force acting in the z-direction of the body frame is induced by the angular velocity of all the propellers. The velocity of the vertical movement depends on the force produced by the propellers. Control input u_1 directly controls thrust in body z-axis by controlling angular velocities of the propellers. Roll, pitch, and yaw movement about body axis are obtained by combinations of different angular velocities of the six propellers. Mapping between control inputs u_1, u_2, u_3 and propellers angular velocities are presented in this section.

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b & b & b \\ -\frac{bl}{2} & -bl & -\frac{bl}{2} & \frac{bl}{2} & bl & -\frac{bl}{2} \\ -\frac{bl\sqrt{3}}{2} & 0 & \frac{bl\sqrt{3}}{2} & \frac{bl\sqrt{3}}{2} & 0 & -\frac{bl\sqrt{3}}{2} \\ -d & d & -d & d & -d & d \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \\ \Omega_5^2 \\ \Omega_6^2 \end{bmatrix} \tag{3.18}$$

Equation 3.18 presents control inputs as functions of the propeller velocities. By taking the pseudo-inverse of the 4x6 matrix in (3.18), it is possible to find propeller velocities as function of control inputs.

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \\ \Omega_5^2 \\ \Omega_6^2 \end{bmatrix} = \frac{1}{6bl} \begin{bmatrix} l & 2 & 0 & -\frac{bl}{d} \\ l & 1 & -\sqrt{3} & \frac{bl}{d} \\ l & -1 & -\sqrt{3} & -\frac{bl}{d} \\ l & -2 & 0 & \frac{bl}{d} \\ l & -1 & \sqrt{3} & -\frac{bl}{d} \\ l & 1 & \sqrt{3} & \frac{bl}{d} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.19}$$

Total System Equations

Model for the total system is presented by combining the translational dynamics of the system, the rotational dynamics of the system and control inputs [30].

$$\begin{aligned}
\ddot{\phi} &= \frac{1}{J_{xx}}\left[\dot{\theta}\dot{\psi}(J_{yy}-J_{zz}) + bl\Big[-\Omega_2^2+\Omega_5^2+\frac{1}{2}(-\Omega_1^2-\Omega_3^2+\Omega_2^2+\Omega_6^2)\Big]\right] \\
&= \frac{1}{J_{xx}}\left[\dot{\theta}\dot{\psi}(J_{yy}-J_{zz}) + u_2\right] && \text{(3.20a)} \\
\ddot{\theta} &= \frac{1}{J_{yy}}\left[\dot{\phi}\dot{\psi}(J_{zz}-J_{xx}) + bl\frac{\sqrt{3}}{2}(-\Omega_1^2+\Omega_3^2+\Omega_4^2-\Omega_6^2)\right] \\
&= \frac{1}{J_{yy}}\left[\dot{\phi}\dot{\psi}(J_{zz}-J_{xx}) + u_3\right] && \text{(3.20b)} \\
\ddot{\psi} &= \frac{1}{J_{zz}}\left[\dot{\phi}\dot{\theta}(J_{xx}-J_{yy}) + d(-\Omega_1^2+\Omega_2^2-\Omega_3^2+\Omega_4^2-\Omega_5^2+\Omega_6^2)\right] \\
&= \frac{1}{J_{zz}}\left[\dot{\phi}\dot{\theta}(J_{xx}-J_{yy}) + u_4\right] && \text{(3.20c)} \\
\ddot{x} &= \frac{1}{m}\left[(\cos\phi\cos\psi\sin\theta+\sin\phi\sin\psi)(\sum_{i=1}^{6}F_i) + f_{drag_x}\dot{x}\right] \\
&= \frac{1}{m}\left[(\cos\phi\cos\psi\sin\theta+\sin\phi\sin\psi)u_1 + f_{drag_x}\dot{x}\right] && \text{(3.20d)} \\
\ddot{y} &= \frac{1}{m}\left[(\cos\phi\sin\psi\sin\theta-\sin\phi\cos\psi)(\sum_{i=1}^{6}F_i) + f_{drag_y}\dot{y}\right] \\
&= \frac{1}{m}\left[(\cos\phi\sin\psi\sin\theta-\sin\phi\cos\psi)u_1 + f_{drag_y}\dot{y}\right] && \text{(3.20e)} \\
\ddot{z} &= \frac{1}{m}\left[(\cos\phi\cos\theta)(\sum_{i=1}^{6}F_i) + f_{drag_z}\dot{z}\right] \\
&= \frac{1}{m}\left[(\cos\phi\cos\theta)u_1 + f_{drag_z}\dot{z}\right] && \text{(3.20f)}
\end{aligned}$$

3.4 Wind

It is important to consider wind disturbances while creating a simulation environment. There exist many techniques for modeling wind. Dryden gust wind model is used in this project.

$$V_{total\ wind} = V_{constant\ wind} + V_{wind\ gust} \tag{3.21}$$

Dryden gust model consists of three transfer functions, one for each wind direction (x, y, and z). Input to the transfer functions is white noise, while the output is wind gusts. To make the model realistic, a constant wind has to be added. Transfer functions contain three parameters. V_a is the nominal airspeed of the vehicle, which is typically 2-4 m/s for the hexacopter used in this task. The same nominal airspeed is used in all three directions. Intensities of turbulence in each direction are represented by σ_x, σ_y and σ_z, and their values can be found in Table 4.1 in [17]. L_x, L_y and L_z are representing the spatial wavelengths, their values can also be found in the same table.

Usually, the transfer function for the wind in the x-direction, which is often defined as the forward direction of a vehicle, is a first-order transfer function, while transfer functions in y and z directions are second-order functions. The reason is that the UAV's usually moves with higher velocity in the x-direction. In this project, the x-direction of the vehicle body frame is not necessary the forward direction of the vehicle movement, therefore second-order transfer functions will be used for wind gusts in all three directions.

$$H_x(s) = \sigma_x \sqrt{\frac{3V_a}{L_x}} \left(\frac{s + \frac{V_a}{\sqrt{3}L_x}}{s + \frac{V_a}{L_x}} \right) \tag{3.22a}$$

$$H_y(s) = \sigma_y \sqrt{\frac{3V_a}{L_y}} \left(\frac{s + \frac{V_a}{\sqrt{3}L_y}}{s + \frac{V_a}{L_y}} \right) \tag{3.22b}$$

$$H_z(s) = \sigma_z \sqrt{\frac{3V_a}{L_z}} \left(\frac{s + \frac{V_a}{\sqrt{3}L_z}}{s + \frac{V_a}{L_z}} \right) \tag{3.22c}$$

Wind gusts are given in body frame, according to [17], while constant winds are given in the inertial frame. Since the whole system is modeled relative to the inertial frame, wind gusts have to be transformed from the body to the inertial frame. Transformation is done by multiplying wind vector with rotation matrix from (3.6).

$$V^I_{wind\ gust} = R^I_B \cdot V^B_{wind\ gust} \tag{3.23}$$

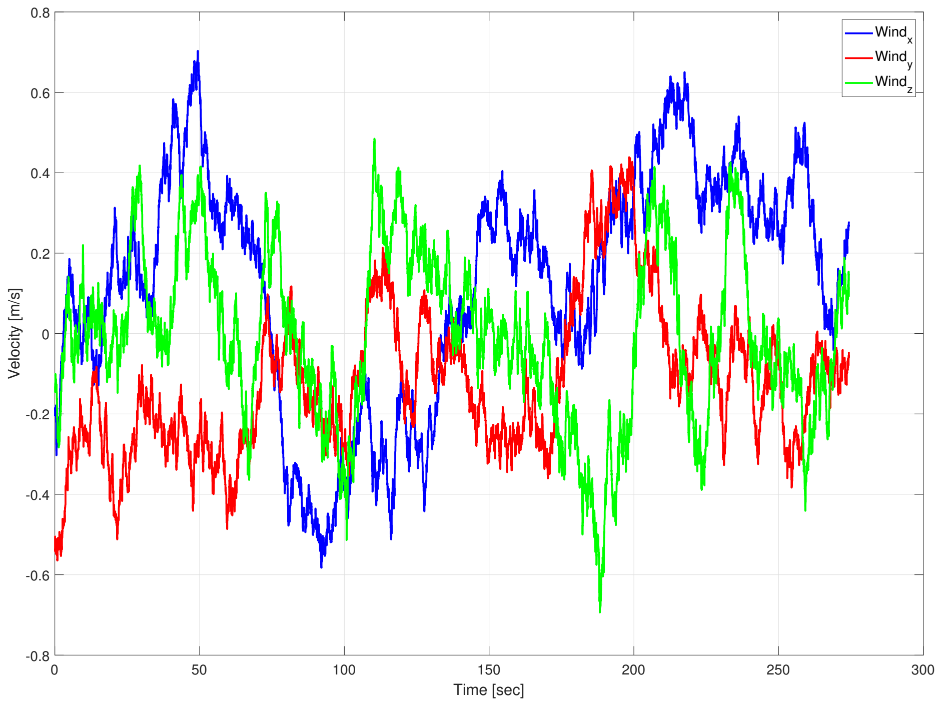


Figure 3.4: Response of the Dryden gust wind model

3.5 Waves

Waves are modeled according to chapter 8.2.6 in [28]. A linear approximation is done during wave modeling. Waves are modeled as second-order transfer functions.

$$h_{wave}(s) = \frac{K_w s}{s^2 + 2\lambda\omega_0 s + \omega_0^2} \tag{3.24}$$

where

$$K_w = 2\lambda\omega_0\sigma$$

Parameters in (3.24) can be found by looking a Pierson-Moskowitz Spectrum (Section 3.1.4). To create a spectra plot, which is later used to compute parameters, $V_{19.4}$ has to be computed. The formula used is,

$$\begin{aligned} H_s &= \frac{2.06}{g^2} V_{19.4}^2 \\ V_{19.4} \approx V_{20} &= g \cdot \sqrt{\frac{H_s}{2.06}} \end{aligned} \tag{3.25}$$

where H_s is average wave height value, depending on chosen sea state. The values of wave height intervals of different sea states are based on Table 8.5 in [28]. Further, wave spectra is created by slightly modifying functions from MSS toolbox.[31]

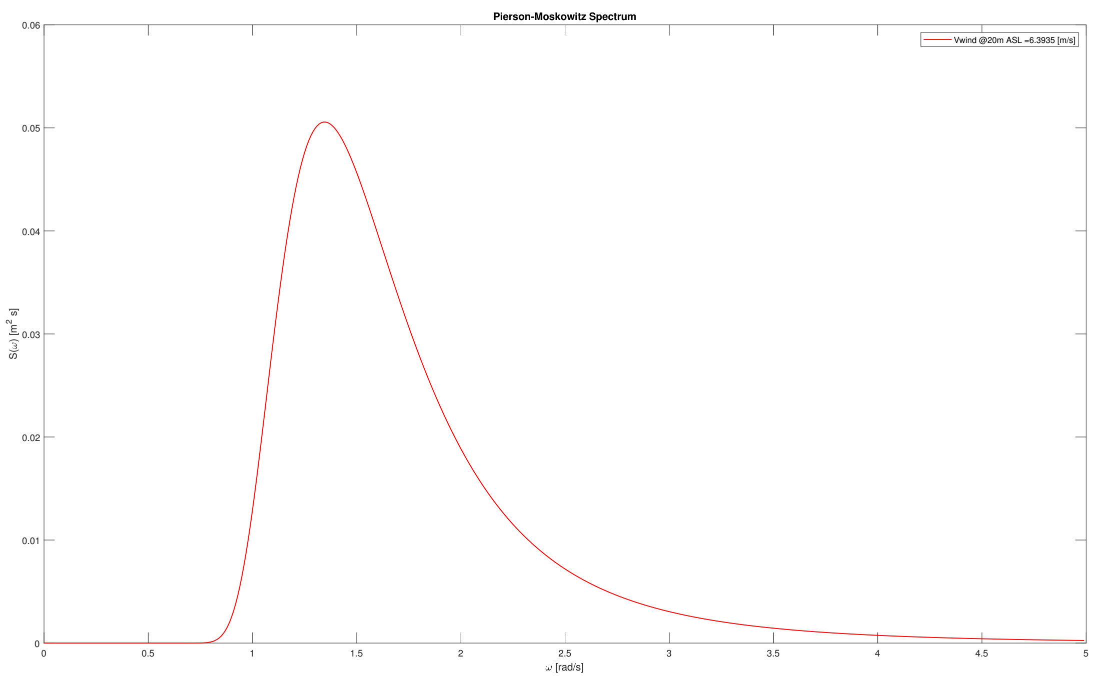


Figure 3.5: PM spectrum for sea state 4

Figure 3.5 shows PM specter for sea state 4. The specter is further used to calculate parameters for (3.24).

$$\begin{aligned} \sigma^2 &= m_0 = S(w_0) = S_{max} \\ \sigma &= \sqrt{S_{max}} \end{aligned} \tag{3.26}$$

Further, λ can be calculated by using Matlab command:

```
    lscurvefit('Slin',0.1,\omega,S)
```

where **Slin** is a function from MSS toolbox.

Table 3.1 shows parameters found for different sea states.

Table 3.1: Parameters for different sea states

Sea State	K_w	λ	ω_0
1	0.00005	0.1	0.0001
2	0.998	0.1	4.99
3	1.1977	0.2604	2.3
4	0.6932	0.2567	1.3501
5	0.4719	0.2564	0.9201
6	0.3583	0.2559	0.7001
7	0.2884	0.2574	0.5601

Wave densities are assumed to be equal in all directions. One transfer function is developed for each direction, but parameters inside functions are equal. A constant value is added in all three directions, representing current.

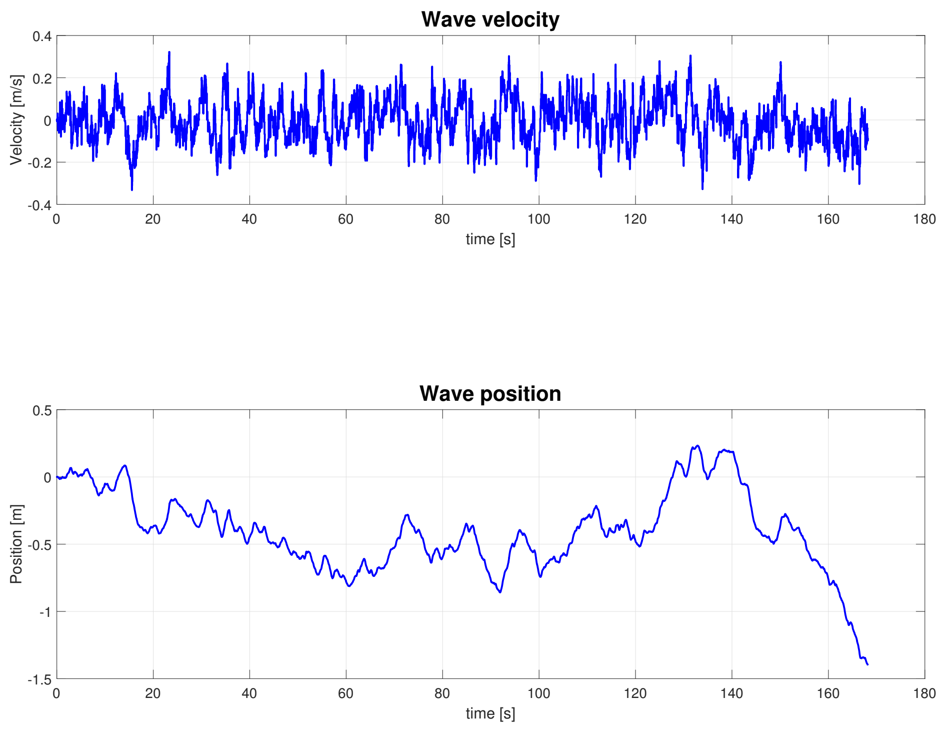


Figure 3.6: Wave response in x-direction for sea state = 4

Figure 3.6 shows response of wave simulation for sea state = 4.

Chapter 4

Control

4.1 Theory

4.1.1 PID

The Proportional Integral Derivative (PID) controller is one of the most applied control algorithms today. In fact, more than 95 % of the process control loops today are using the PID type control [32].
The control algorithm itself is based on the feedback control. The error between the actual value and the desired value is the input of the control algorithm. The error value then goes through three different parts of the control algorithm, that contribute to the output signal (Figure 4.1), each part having its own characteristic. Following three parts constitute the PID algorithm,

Proportional The proportional part of the algorithm multiplies error value with a pre-tuned parameter K_p, trying to remove the error. It is common to only use the proportional part of the PID algorithm as an own controller. Such controllers are called P-controllers.

Integral The working principle of the integral part of the PID algorithm is summing up error values over time in order to remove the stationary offset. The summed up error is multiplied by the pre-tuned parameter K_i. Using an integral part alone in a control algorithm is not that common, but combinations like PI and PID are widely used today.

Derivative The derivative part of the algorithm calculates error slope, based on the derivative of the error value. The derivative of the error value is multiplied with the pre-tuned parameter K_d, in order to remove the error. Common combinations of the control algorithm where the derivative part is used are PD and PID.

$$U = K_p \cdot e(t) K_i \int_0^t e(t)\, dt + K_d \cdot \frac{de(t)}{dt} \tag{4.1}$$

As all three mentioned parts are contributing to the control signal (4.1), it is important to find a balance that gives the desired control signal. The desired balance can be found by tuning K_p,

K_i and K_d parameters. There exist many tuning methods designed to find the perfect parameter values [32]. Two tuning methods worth mentioning are *Ziegler-Nichols* and *Cohen-Coon*, as their use is the most prevalent today. Still, finding the perfect parameter values is a difficult accomplishment, and the process often ends as a trial and error approach.

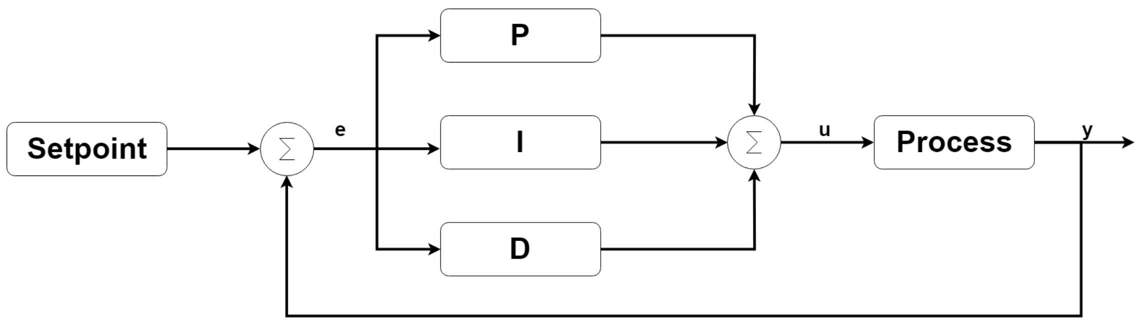


Figure 4.1: PID block diagram

4.1.2 Ziegler-Nichols Tuning Method

The Ziegler-Nichols tuning method is a method used to tune a PID controller (Section 4.1.1). It gives a suggestion of tuning parameters based on the critical gain and the critical period. The critical gain K_c is the lowest proportional gain that gives oscillations that do not get damped out with time. The critical period T_c is the period of the oscillations. After the parameters are found, Table 4.1 is used to find a suggestion to the tuning parameters of the controller. There is often a need for adjustments of the parameters found using the Zeigler-Nichols table.

Table 4.1: Ziegler - Nichols table

Controller type	K_p	K_i	K_d
P	$0.5K_c$		
PI	$0.45K_c$	$0.54K_c/T_c$	
PD	$0.8K_c$		$K_cT_c/10$
PID	$0.6K_c$	$1.2K_c/T_c$	$3K_cT_c/40$
PID - some overshoot	$K_c/3$	$0.66K_c/T_c$	$K_cT_c/10$
PID - no overshoot	$0.5K_c$	$0.4K_c/T_c$	$K_cT_c/15$

Ziegler-Nichols tuning method in steps:

1. Turn off the integral and derivative part of the controller.
2. Increase the proportional gain till the response signal is oscillating with oscillations that do not damp. The proportional gain that gives such oscillations is called critical gain K_c.
3. Measure the period of the oscillations and find T_c.
4. Use Table 4.1 and find a suggestion to controller gains.
5. Adjust the gains.

4.2 Method

4.2.1 Altitude Controller

As mentioned in Section 2.1, controllers are designed to calculate velocity control commands. The altitude PID controller uses altitude error as input, and calculates desired velocity in z-direction of the NED frame (Section 3.1.1). The transformation between the NED and the vehicle body-frame is done by the ArduPilot software (Section 2.1.4). The control algorithm for the altitude PID controller is presented in (4.2).

$$u_z = K_{pz}e_z(t) + K_{dz}\dot{e}(t) + K_{iz}\int_0^T e_z(t)dt \tag{4.2}$$

where

$$e_z(t) = z_d - z$$

Values of tuning parameters K_{pz}, K_{iz} and K_{dz} are shown later in the report.

4.2.2 Horizontal Controllers

In order to perform a successful landing, it is important to hold a satisfying horizontal position relative to the landing target. The horizontal control system is divided into two controllers, one for each horizontal direction, x and y. Like altitude controller (Section 4.2.1), the horizontal controllers also calculates desired control velocity (Section 2.1).

The control design for x and y controller are equal, with variations in the tuning parameters. The control design is also similar to the PID controller from Section 4.1.1, with the difference being the input of the derivative part of the controller.

As the target (landing platform) velocity is known, it will improve the result including the velocity into the control algorithm. Therefore, horizontal controllers are designed to use error between vehicle velocity and desired velocity as input to the derivative part of the PID controller instead of position error derivative used in (4.1) and (4.2).

Also, a velocity feedforward is added to the controllers, with target velocity being the fed signal. The purpose of the velocity feedforward is to get a faster response, especially in direction changes. Equation 4.3 shows the PID algorithm used for the horizontal controllers mathematically.

$$u_x = K_{px}(x_d - x) + K_{ix}\int (x_d - x)dt + K_{dx}(v_{xd} - v_x) + v_x \tag{4.3}$$

From (4.3) it can be noticed that the derivative part of the horizontal controllers actually is a velocity P controller added to the PI position controller (Section 4.1.1). The equation for the y direction controller is equal (4.3) with index being the only difference.

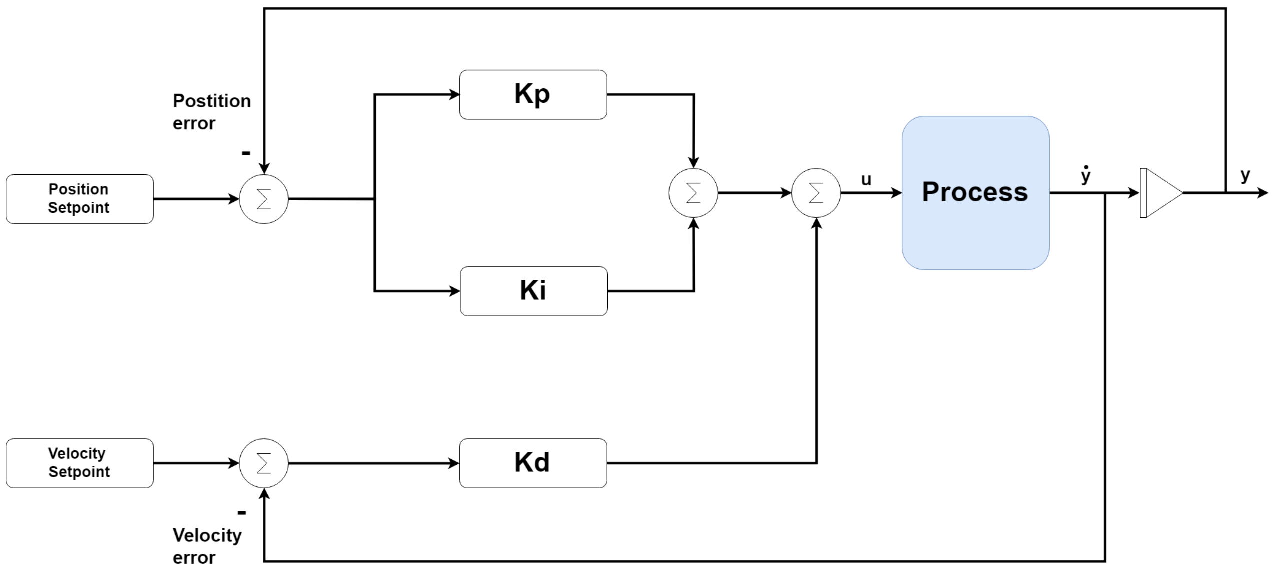


Figure 4.2: PID block diagram using velocity feedback

4.2.3 Filter

To provide a smooth reference input to the altitude controller, a wave filter is designed. The idea with the filter is to only provide the significant changes in wave height to the altitude controller, and damp put the small changes. Damping out small waves will lead to the altitude controller reference without many oscillations.

The filter is designed as a first-order low-pass filter, because the large waves have a lower frequency than the small waves. A first-order low-pass filter is passing frequencies lower than a specific frequency, and damping frequencies higher than the same specific frequency. The specific frequency is called the cutoff frequency. To find the cutoff frequency, frequency specters from all sea states described in Section 3.5 are used. The frequency specters are presented in Figure 4.3. The frequency spectras are computed using the Fourier transform function in Matlab. It is worth mentioning that the frequency specters are obtained by looking at wave frequencies. Heave frequencies for a ship will look slightly different, as the ship will not follow every wave movement. The problem is discussed later in the report.

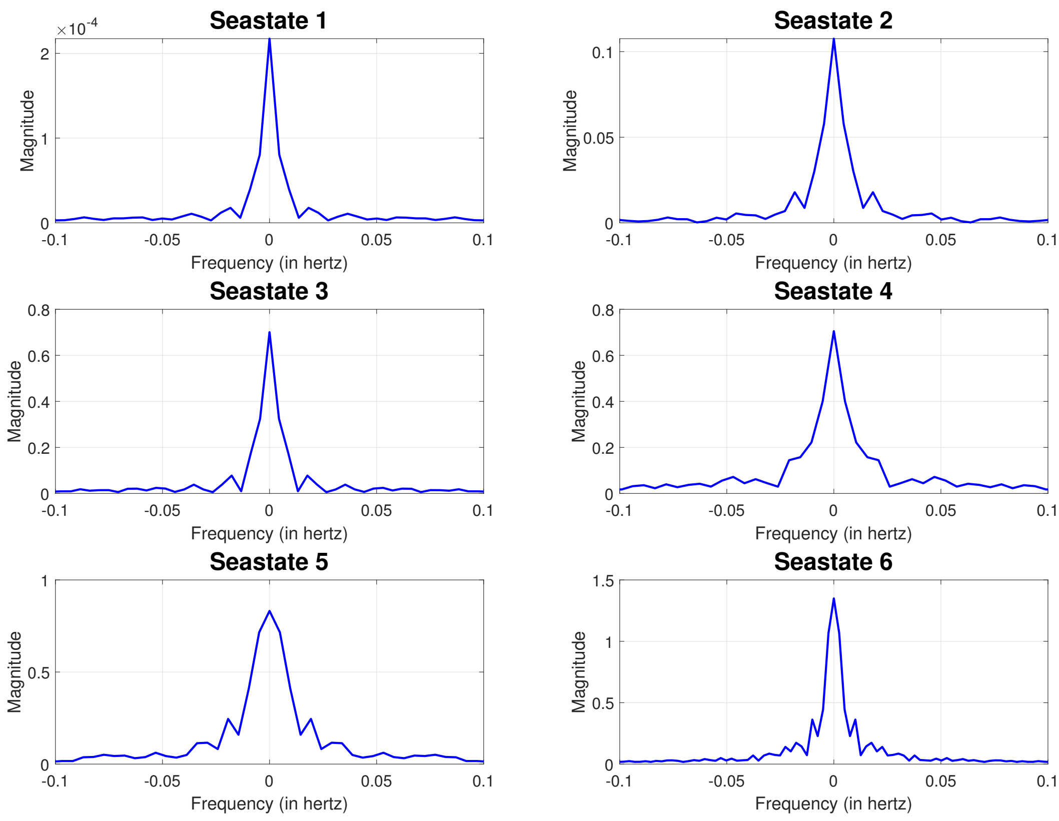


Figure 4.3: Frequency spectras of different sea states

By studying the frequency spectra from Figure 4.3, the desired cutoff frequency is found to be $f = 0.38$ Hz.

The standard transfer function for a low-pass filter is:

$$H_{filter}(s) = \frac{1}{Ts + 1} \tag{4.4}$$

Inserting the cutoff frequency found by investigating Figure 4.3:

$$H_{filter}(s) = \frac{1}{\frac{1}{f}s + 1} = \frac{1}{2.63s + 1} \tag{4.5}$$

Equation (4.5) implies that all frequencies lower than 0.38Hz will pass through the filter, while higher frequencies will be damped by the filter. A frequency diagram of the filter is presented in Figure 4.4.

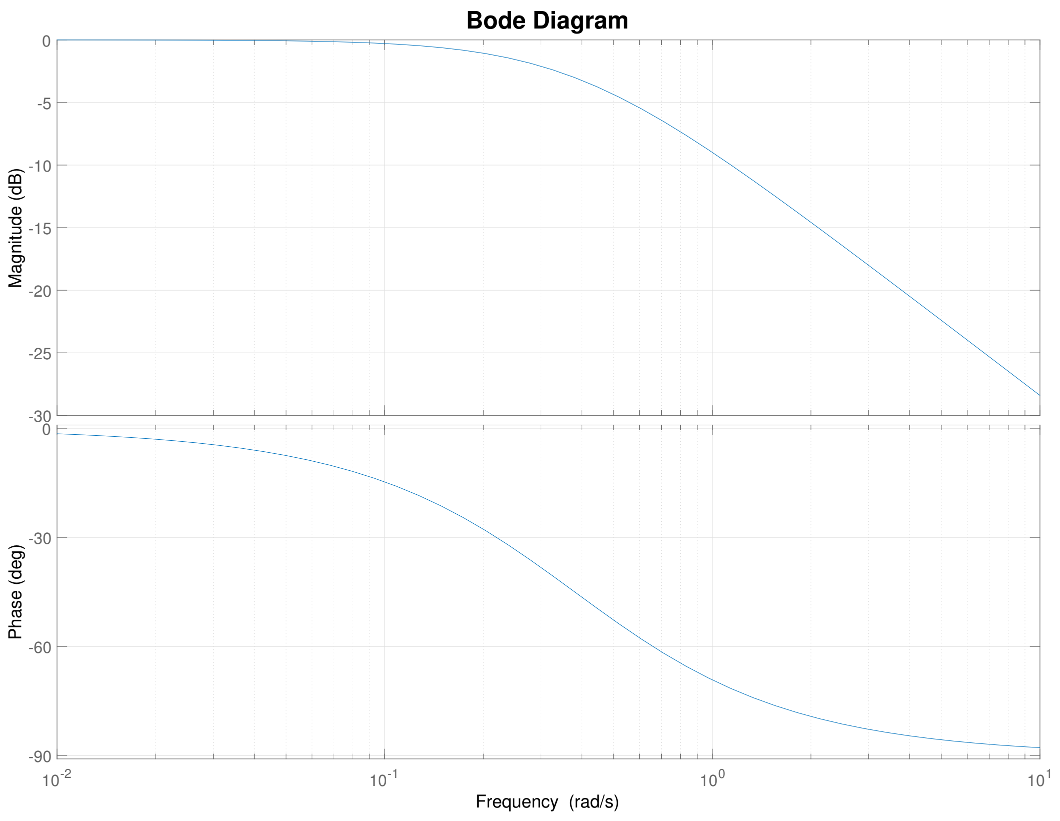


Figure 4.4: Bode diagram of the low-pass filter

Chapter 5

Implementation

The implementation part of this project can be divided into two parts. The first part is set up and configuration of the hardware described in Section 2.2, while the second part is the implementation of the control systems and the simulation environment set up in DUNE.

There exist a wiki web site developed by students associated with the UAV lab at the NTNU [33], with the purpose to help a beginner to prepare their UAV for field testing and set up a simulation environment. The web site contains elaborated set up and configuration guides for every hardware component needed to perform the field tests. The elaborated guides from the web site are used to set up all parts of the system listed up in Section 2.

When it comes to the implementation of the control system and the simulation environment, everything is done by editing and adding tasks to the already existing DUNE system (Section 2.1.1). As the programming language in DUNE is C++, all the implemented code in this project is written in the named language. Of course, the configuration files in DUNE are modified during the implementation, in order to add new tasks to the system (see Section 2.1.1 for explanation).

5.1 Simulation

The whole landing task is about the UAV's ability to track platforms horizontal position, at the same time as it moves safely downwards. To simulate and design algorithms for the landing process, several parts have to be implemented in DUNE.

5.1.1 System

Roughly, the simulation environment for the landing process has to contain:

- UAV movement simulation
- Platform movement simulation
- Wind disturbances acting on the UAV
- Wave disturbances acting on the landing platform
- Communication between the landing platform and the UAV

There exist an additional file package to DUNE, developed by NTNU, that contains simulation packages for several vehicles. Simulation packages for 4 hexacopters are developed. There also exist 4 configuration files, one for each hexacopter, that are including all necessary DUNE tasks to simulate the movement of the hexacopter (Section 2.1.1). The configuration files are named after the pre-assigned number of the given hexacopter, *ntnu-hexa-00x*.

The configuration files are including tasks that are considering all forces and torques acting on the UAV, described in Section 3.2. Therefore, there is no need for implementing new files for the vehicle model.

The landing platform is supposed to be located on a ship. There exist files for ship simulations as well in the named addition package for DUNE. Still, much time at the beginning of the project was spent on getting a better knowledge of the UAV simulation files, while files for ship simulation were overlooked. Besides, a communication system for coordinate flight is developed at the UAVlab, focusing on relative positioning between several hexacopters. It is working in the way that one hexacopter is assigned the master role, and the others are assigned slave roles. UAV assigned the master role gets it's position through ArduPilot and *IMC::EstimatedState* messages, as described in Section 2.3. On the other hand, UAVs assigned the slave role are getting their position relative to the master UAV through *IMC::EstimatedLocalState* message.

For a simulation, the only thing that is necessary to know is the UAV position relative to another object representing the landing platform. The object representing the landing platform does not necessarily have to be a ship, as long it represents ship motion properly. Therefore, simulation files for a hexacopter are utilized in order to simulate the landing platform, meaning that a hexacopter is actually landing on another hexacopter in the simulations. Another reason for using hexacopter files to simulate the landing platform is that the already developed communication system for coordinated flights can be utilized for communication between the "platform" and the UAV. In the communication system, the UAV simulating the landing platform is assigned the master role and the UAV assigned the slave role will search for the zero position relative to the master.

5.1.2 Wind

To make the simulation as realistic as possible, environmental disturbances have to be added to the system. The wind is the disturbance having the most impact on a UAV in the air. The wind disturbance, modeled as described in Section 3.4, is implemented by developing own DUNE task for wind simulation. The task is developed in C++ and included in the configuration file of the UAV.

PM wind model requires Gaussian white noise as input. In the specialization project, there exist own blocks in Simulink for white noise generation. For this project, the random number function from C++ is assumed to be good enough for white noise recreation.

Wind model from Section 3.4 is estimating wind gusts by sending white noise through a transfer function. Transfer functions are easy to implement in Matlab, simply by using already existing transfer function blocks. The implementation procedure is more difficult in C++. To implement the model, the transfer function has to be transformed into a state-space model.
Transfer function from (3.22):

$$
\begin{aligned}
H(s) = \frac{y(s)}{u(s)} &= \sigma\sqrt{\frac{3V}{L}}\frac{s+\frac{V}{L\sqrt{3}}}{(s+\frac{V}{L})^2} && (5.1a)\\
&= K\frac{s+a}{(s+b)^2} && (5.1b)\\
&= K\frac{s+a}{s^2+2bs+b^2} \quad \Big/ \cdot \frac{x(s)}{x(s)} && (5.1c)\\
&= K\frac{(s+a)x(s)}{(s^2+2bs+b^2)x(s)} && (5.1d)
\end{aligned}
$$

where

$$
K = \sigma\sqrt{\frac{3V}{L}} \ , \ a = \frac{V}{L\sqrt{3}} \ , \ b = \frac{V}{L}
$$

Separating numerator and denominator.

$$
\begin{aligned}
y(s) &= K(s+a)x(s) && (5.2)\\
&= Kx(s)\cdot s + Kax(s)
\end{aligned}
$$

Calculating the inverse Laplace of (5.2).

$$
y(t) = K\dot{x} + ax \qquad (5.3)
$$

The same procedure is done for the denominator.

$$u(s) = (s^2 + 2bs + b^2)x(s) \tag{5.4}$$
$$= x(s) \cdot s^2 + 2bx(s) \cdot s + b^2x(s) \tag{5.5}$$

Calculating the inverse Laplace of equation (5.4).

$$u(t) = \ddot{x} + 2b\dot{x} + b^2x \tag{5.6}$$

Using (5.3) and (5.6) and transforming to state space equations where $x_1 = x$ and $x_2 = \dot{x}$.

$$\dot{x}_1 = x_2 \tag{5.7a}$$
$$\dot{x}_2 = u - 2bx_2 - b^2x_1 \tag{5.7b}$$
$$y = Kx_2 + a \tag{5.7c}$$

Signal u is the Gaussian white noise signal, implemented by using a random number generator in C++.

Equation 5.7 is described in continuous time. In order to implement the state-space equations in C++, the equations have to be discretized. There exist many discretization methods, with varying precision and complexity. Forward Euler method, described in Appendix A, is chosen because of its low complexity and satisfying precision for this case.

Discretizing (5.7) by the approach described in Appendix A and getting the discrete state-space equation.

$$x_1[k+1] = x_2[k]h + x_1[k] \tag{5.8}$$
$$x_2[k+1] = h(u - 2bx_2[k] - b^2x_1[k]) + x_2[k] \tag{5.9}$$
$$y[k] = Kx_2[k] + a \tag{5.10}$$

Start values $x_1[0]$ and $x_2[0]$ are assumed to equal zero.

Equation 5.10 is generating estimated wind gust velocities in three directions of the vehicle body frame (Section 3.1.1). The velocities are then transformed to the inertial frame (Section 3.1.1). After transformation to the inertial frame, constant velocities in all three directions are added, to get a realistic wind model.

Then, the complete wind velocity estimate in NED frame is integrated, to get the position in the same frame. The integration is implemented by simply multiplying velocity value with the time difference between steps, and adding the calculated value to the previous integrated value.

```
    position += velocity*dt
```

The position in the inertial frame is further transformed to the geodetic frame. As described in Section 2.1.1, *IMC::EstimatedState* message is sending position information in geodetic coordinate frame. That is the reason why the estimated wind is transformed to the geodetic frame. After the transformation, the wind estimate is dispatched to *IMC::EstimatedWind* message. The message is then consumed by the task responsible for the generation of the *IMC::EstimatedState* message. The wind implementation to the whole system is done simply just by adding content from the *IMC::EstimatedWind* message to the *IMC::EstimatedState* message, and sending it to the whole system.

5.1.3 Waves

Wave disturbances are important to include in order to make a simulation for the landing platform as realistic as possible. The implementation of wave disturbances is done in the same way as the implementation of wind disturbances (Section 5.1.2). That means that an own DUNE task (Section 2.1.1) is developed to estimate wave disturbances. Like the task estimating wind disturbances, task estimating wave disturbances is written in C++.

The principle for modeling wave disturbances is explained in Section 3.5. It is based on a Gaussian white noise going through a transfer function (filter), giving wave disturbances as output. Gaussian white noise is implemented in the same way as described in Section 5.1.2.

First, the transfer function from (3.24) is transformed to state-space equations, in order to implement in C++.

$$H(s) = \frac{y(s)}{u(s)} = \frac{K_w s}{s^2 + 2\lambda\omega_0 s + \omega_0^2} \Big/ \cdot \frac{x(s)}{x(s)} \tag{5.11a}$$

$$= \frac{K_w s x(s)}{(s^2 + 2\lambda\omega_0 s + \omega_0^2)x(s)} \tag{5.11b}$$

where

$$K_w = 2\lambda\omega_0\sigma$$

Separating numerator and denominator in order to perform the Laplace transform.

$$y(s) = K_w s x(s) \tag{5.12}$$

Inverse Laplace of (5.12) gives:

$$y(t) = K_w \dot{x} \tag{5.13}$$

Same procedure for the denominator:

$$\begin{aligned} u(s) &= (s^2 + 2\lambda\omega_0 s + \omega_0^2)x(s) & (5.14) \\ &= s^2 x(s) + 2\lambda\omega_0 s x(s) + \omega_0^2 x(s) & (5.15) \end{aligned}$$

Inverse Laplace of (5.14) gives:

$$u(t) = \ddot{x} + 2\lambda\omega_0 \dot{x} + \omega_0^2 x \tag{5.16}$$

Using (5.13) and (5.16) and transforming to state-space equations where $x_1 = x$ and $x_2 = \dot{x}$.

$$\begin{aligned} \dot{x}_1 &= x_2 & (5.17a) \\ \dot{x}_2 &= u - 2\lambda\omega_0 x_2 - \omega_0^2 x_1 & (5.17b) \\ y &= K_w x_2 & (5.17c) \end{aligned}$$

where u is the white noise signal generated using random number function in C++.

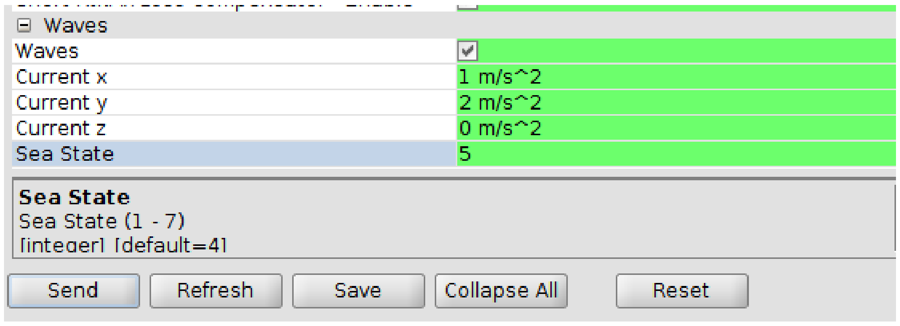
Like (5.7), (5.17) has to be discretized before implementation in C++. Discretizing by the approach described in Appendix A and getting the discrete state-space equation.

$$\begin{aligned} x_1[k+1] &= h x_2[k] + x_1[k] & (5.18) \\ x_2[k+1] &= h(u - 2\lambda\omega_0 x_2[k] - \omega_0^2 x_1[k]) + x_2[k] & (5.19) \\ y[k] &= K_w x_2[k] & (5.20) \end{aligned}$$

Start values are assumed equal zero.

The estimated wave velocities in x,y and z directions are added to *IMC::EstimatedState* messages the same way as wind velocities, shown in Section 5.1.2. Drifting currents are also added, to make the simulation as realistic as possible. One difference from Section 5.1.2 is that wave velocities are modeled in the inertial frame, meaning that they are directly transformed to geodetic coordinate frame according to the approach shown in Section 3.1.1. The wave contribution is transported inside *IMC::EstimatedWave* message, and added to the *IMC::EstimatedState* message of the hexacopter simulating the landing platform.

Table 3.1 shows the parameters of different sea states, depending on how rough weather conditions one wants to simulate. Sea states are implemented as DUNE task arguments. Shortly explained, task arguments are variables that can be changed through task execution. They are also displayed in Neptus (2.1.3), and can be changed during task monitoring. This is useful for simulations that are supposed to test robustness through different weather conditions. Figure 5.1 shows how the sea states can be changed in Neptus.



⊟ Waves	
Waves	✓
Current x	1 m/s^2
Current y	2 m/s^2
Current z	0 m/s^2
Sea State	5

Sea State
Sea State (1 - 7)
[integer] [default=4]

Figure 5.1: Sea state changes in Neptus

5.2 Filter

The filter is implemented as a function at the beginning of the control algorithm task. As the filter function is a transfer function, it has to be transformed into state-space in order to be implemented.

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{Ts+1} \bigg/ \cdot \frac{x(s)}{x(s)} \tag{5.21}$$

$$= \frac{x(s)}{(Ts+1)x(s)} \tag{5.22}$$

Separating denominator and numerator.

$$y(s) = x(s) \tag{5.23}$$

$$y(t) = x(t) \tag{5.24}$$

$$u(s) = (Ts+1)x(s) \tag{5.25}$$

$$= Tsx(s) + x(s) \tag{5.26}$$

The inverse Laplace of (5.25) is:

$$u(t) = T\dot{x} + x \tag{5.27}$$

The state-space equation made from (5.23) and 5.27 is:

$$\dot{x} = \frac{1}{T}(u - x) \tag{5.28}$$

$$y = x \tag{5.29}$$

Discretizing (5.28) in order to implement in C++. Using approach described in Appendix A to get the final, discrete functions.

$$x[k+1] = \frac{h}{T}(u[k] - x[k]) + x[k] \tag{5.30}$$

$$y[k] = x[k] \tag{5.31}$$

5.3 Controllers

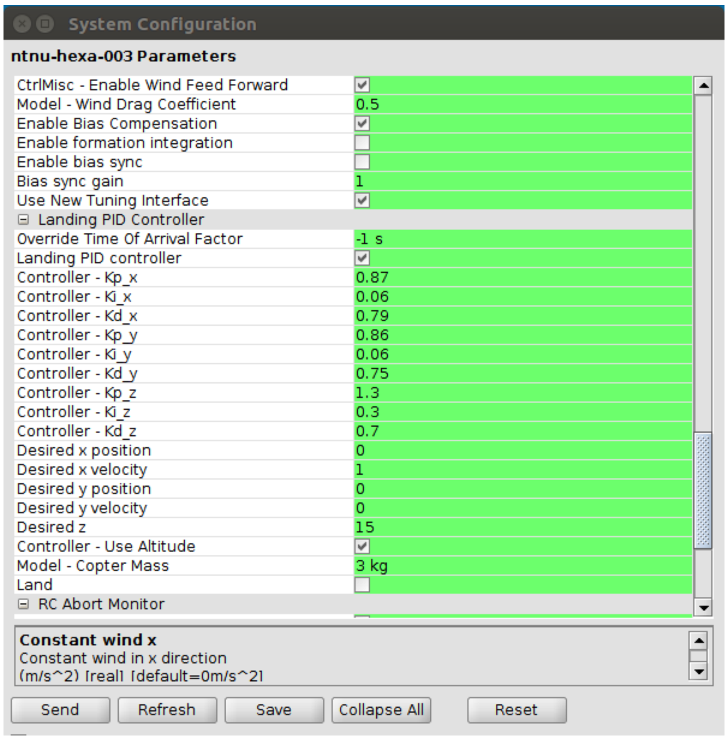
The velocity controllers for all three directions are implemented as one task, and included to the hexacopter configuration file. The psaudocode of the PID controller is presented in Algorithm 1. The controller uses *IMC::EstimatedLocalState* message as the signal feedback. The *IMC::EstimatedLocalState* messages provides geodetic position of the other vehicle in the system, which is considered as the target in simulations (Section 2.3.1). Eigen library in C++ [34] is used to translate the geodetic position to the inertial frame. The translation algorithm requires a reference in order to translate geodetic position to the NED frame. The reference input to the algorithm is the geodetic position of the target, meaning that the position feedback of the PID controller is error in the NED frame relative to the target. Velocity input to the PID algorithm is taken directly from *IMC::EstimatedLocalState*, as it provides the velocity messages directly in the NED frame.

```
Set tuning parameters: $K_p$, $K_i$ and $K_d$ ;
Set saturation values: $sat_{min}$ and $sat_{max}$ ;
dt = current time - time measured previous time step ;
position error = target position - UAV position ;
velocity error = target velocity - UAV velocity ;
control output = $K_p$ · position error + $K_i$ · position error · dt + $K_d$ · velocity error ;
if control output $> sat_{max}$ then
    control output = $sat_{max}$ ;
end
else if control output $< sat_{min}$ then
    control output = $sat_{min}$ ;
end
```

Algorithm 1: PID controller

The control task is designed to first compute the output of the x-controller, second from the y-controller, and the output of the altitude controller at the end. All outputs are dispatched together in the *IMC::DesiredVelocity* message. The message is further consumed by the task that communicates with the ArduPilot, and sent to the ArduPilot through MAVLink.

The tuning parameters and the desired position are implemented as task arguments of the landing controller task. That allows the user to change parameters during the flight, making in-flight tuning process easy. Figure 5.2 shows where the parameters can be changed in Neptus during a flight.



System Configuration

ntnu-hexa-003 Parameters

CtrlMisc - Enable Wind Feed Forward	✓
Model - Wind Drag Coefficient	0.5
Enable Bias Compensation	✓
Enable formation integration	☐
Enable bias sync	☐
Bias sync gain	1
Use New Tuning Interface	✓
⊟ Landing PID Controller	
Override Time Of Arrival Factor	-1 s
Landing PID controller	✓
Controller - Kp_x	0.87
Controller - Ki_x	0.06
Controller - Kd_x	0.79
Controller - Kp_y	0.86
Controller - Ki_y	0.06
Controller - Kd_y	0.75
Controller - Kp_z	1.3
Controller - Ki_z	0.3
Controller - Kd_z	0.7
Desired x position	0
Desired x velocity	1
Desired y position	0
Desired y velocity	0
Desired z	15
Controller - Use Altitude	✓
Model - Copter Mass	3 kg
Land	☐
⊟ RC Abort Monitor	

Constant wind x
Constant wind in x direction
(m/s^2) [real] [default=0m/s^2]

Figure 5.2: Change of tuning parameters in Neptus

5.4 Landing Algorithms

The idea of a state machine is continued from the specialization project. The state machine in this project is simplified compared to the main idea in the specialization project. The main reason for the simplification is that the state machine in the preparation project had too many states and rules, resulting in many landings aborted unnecessary.

5.4.1 Boundaries

The state machine works on the principle that the controller always sees imaginary boundaries. The boundaries are formed by two imaginary cylinders. The large cylinder is extending straight up from the center of the platform, while the small cylinder lays inside the large cylinder, starting 2m above the platform and ending 4m above the platform. The diameter of both cylinders is sat to be 3m, because of the platform size explained in Section 6.3.2. Both cylinders are shown in Figure 5.3.

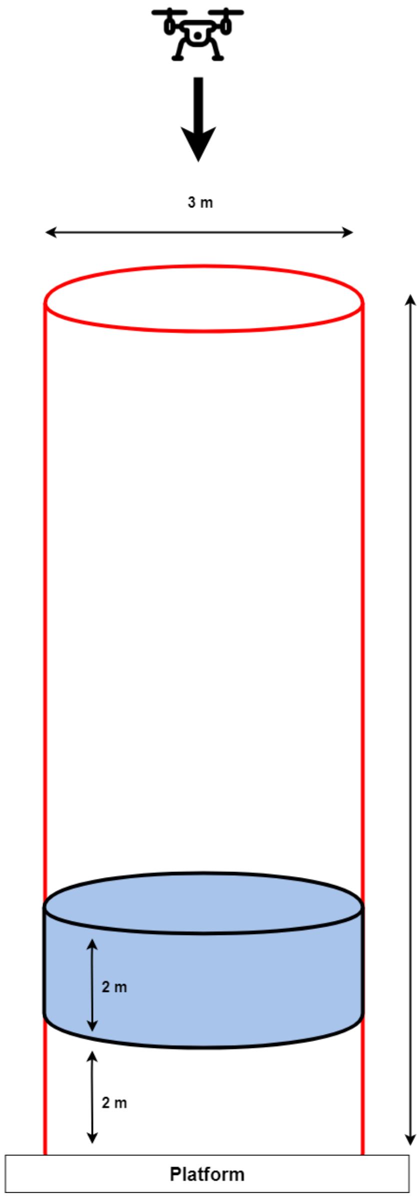


Figure 5.3: Boundaries reprensented by cylinders

The height of the largest cylinder is 15m, which is set to be the hover height. Boundaries of the red cylinder in Figure 5.3 are not strict, and can be violated during a landing, as long as the UAV manages to stay within the boundaries when entering the height of the blue cylinder. On the other hand, the boundaries of the blue cylinder are strict and can not be violated when performing a landing. Violation of the horizontal boundaries of the blue cylinder results in an aborted landing.

5.4.2 Landing Permission

It is important to make sure that the platform is not moving towards the vehicle when the landing is performed. Landing on a platform moving towards the vehicle can cause damages to the vehicle. Such movement is mainly caused by the waves. An algorithm is developed, to make sure that the landing can be performed without risking any damages.

Flow chart of the algorithm can be found in Appendix C.

The algorithm is activated when the vehicle starts the landing operation. The working principle of the algorithm is that it takes four measurements of the platform height. There is one second delay between every measurement. To give landing permission to the vehicle, the algorithm calculates if the second measurement has a value lower than *first measurement + 0.2*. Explained, the algorithm checks if the second measured platform height has moved more than 0.2m over the first measured platform height. If that statement is true, the algorithm waits one second and takes the third measurement. On the other hand, if the statement is false, the algorithm moves back and takes the first measurement again. That means that the algorithm allows platform movement towards the vehicle up to 0.2m/s. A consideration is made, implying that upwards velocities up to 0.2 m/s are harmless for the vehicle. For second platform height measurements lower than the first platform height measurements, the algorithm goes further to the third measurement.

When the algorithm takes the third platform height measurement, the same requirements has to be fulfilled between third and second measurement as for the first and second measurements. If requirements are fulfilled, the algorithm goes further to the fourth measurement. For third height measurements that do not fulfill requirements, the algorithm goes back to the first measurement.

The same requirements are set for the height difference between the fourth and the third measurement. The only difference is that the landing permission is finally given after the fourth measurement. After given permission, the algorithm starts over again. Once requirements between two measurements are not fulfilled, the landing permission is removed.

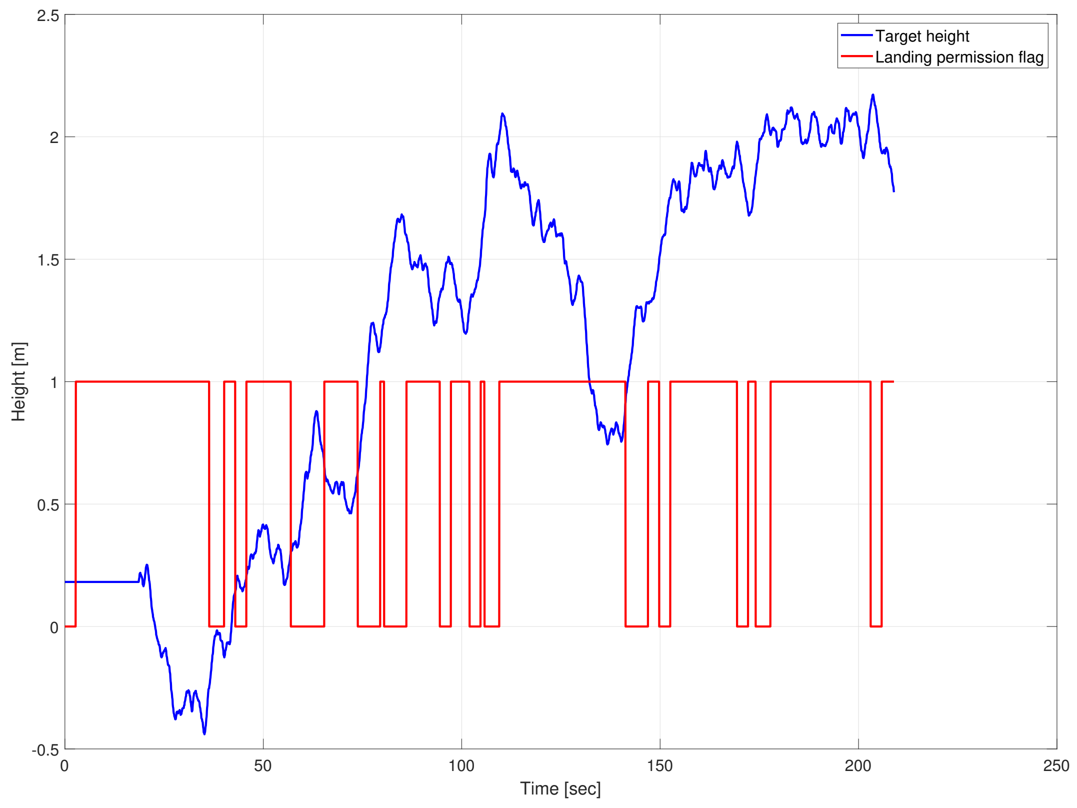


Figure 5.4: Landing permission algorithm performance during a simulation

Figure 5.4 shows landing permissions for a 200 sec simulation on sea state 4. It can be seen that the landing permission is given at reasonable times. Still, there are some points where the target height starts to curve upwards right after the landing permission is given, meaning that there is one second where the landing permission is given and it is dangerous to perform a landing at the same time. The problem will be discussed later in the report.

5.4.3 State Machine

A state machine is developed in order to handle the errors during a landing efficiently, depended on the error size and effect. The state machine is designed to have four different states, that the vehicle has to go through during a landing. Every state has it's requirements for entrance, and own error handling. When an error occurs, the vehicle is sent back to the previous state, depending on the error handling designed for the state the error happened. The states are designed as follows:

State 1 - when inside this state, the vehicle is just waiting for the land command. The altitude controller is holding the position 15m above the target, using the wave filter (Section 6.4). The wave filter is used because it is unnecessary for the vehicle to follow every single wave movement in the z-direction, making it easier for the altitude controller to follow a smooth, filtered reference. The horizontal controllers still have to follow every movement, because of the importance of the vehicle being as close to the target as possible in the horizontal plane. Once the land signal is given by the user, the vehicle starts the landing procedure, and the next state is entered.

State 2 - the state where the first descent happens. The descent velocity in this state is set to 1 m/s. The descent velocity value could possibly be changed to a higher value, but to make the landing as safe and controlled as possible 1 m/s is chosen. State 2 lasts from the moment the landing signal is given until the moment the vehicle enters the blue cylinder in Figure 5.3. Because the vehicle is in State 2 when the altitude relative to the target is higher than 4m, the filtered reference signal for the altitude controller is used. While in State 2, the state machine counts the time the vehicle is inside the horizontal boundaries of the red cylinder in Figure 5.3. If the sate machine counts 3 seconds outside the horizontal boundaries of the red cylinder, the state machine sends the vehicle back to State 1. Before reaching the height of 4m above the target, the vehicle has to stay 3 continuous seconds inside the red cylinder to enter State 3. If the landing flag is turned off by the user, the state machine sends the vehicle back to State 1 automatically.

State 3 - this is the state where the vehicle waits for the land permission signal by the algorithm described in Section 5.4.2. The boundaries of this state are presented by the blue cylinder in Figure 5.3. When entering State 3, the descent velocity changes to 0.3 m/s. Such low descent velocity is chosen to give the land permission algorithm time to send the permission signal. The descent velocity only holds until the altitude of 2m above the target is reached. Once the 2m altitude is reached, the altitude controller turns on, with 2m being the reference altitude. Also, the wave filter is turned off in this state. Being so close to the target requires strict following of every wave movement. If the vehicle violates horizontal boundaries of the blue cylinder in Figure 5.3, the state machine sends the vehicle back to State 3. If the land flag is turned off by the user, the vehicle gets sent back to State 1. To enter the next state, the permission signal is needed from the algorithm described in Section 5.4.

State 4 - when entering this state, there is no way back. The descent velocity is set to 0.1 m/s and the landing is performed.

Figure 5.4 shows how the state machine changes state in different situations.

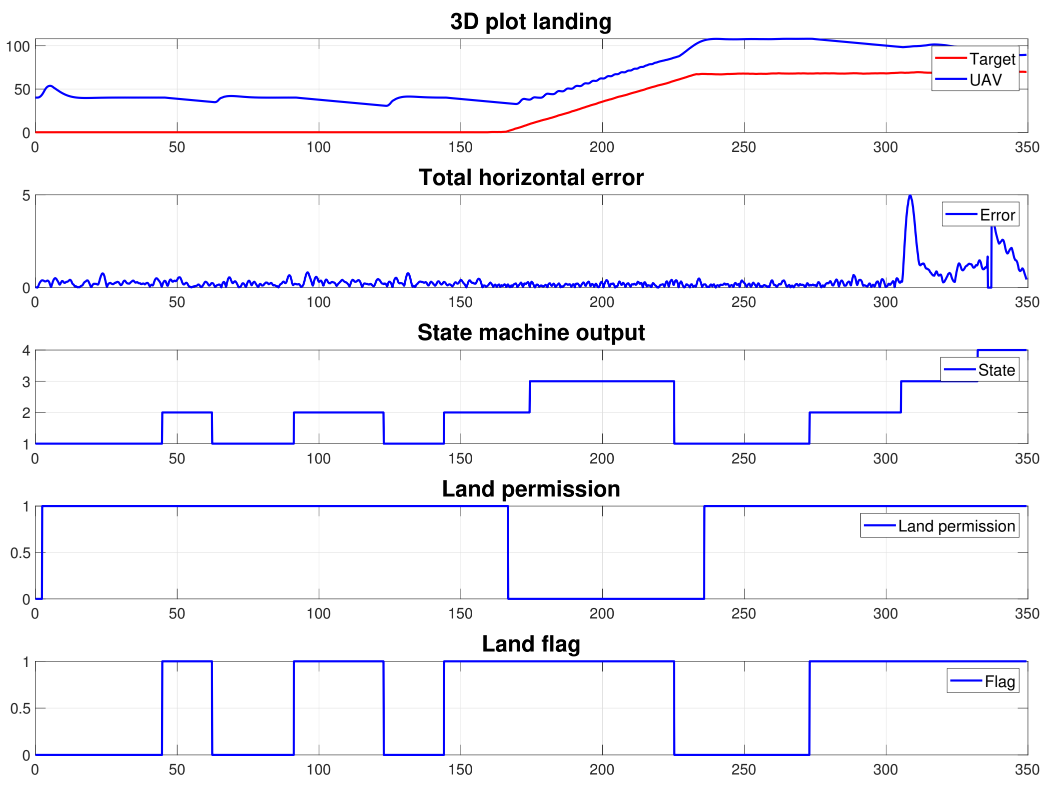


Figure 5.5: State machine performance during a simulation

Chapter 6

Simulation

6.1 Software in the Loop

The SIL simulation means that hardware components in a system are simulated by several software in loop. In the specific case of this project, there exist one task in DUNE for every hardware component on the real-life UAV. Running all existing software together, in a loop, a simulation system of the real-life UAV can be created. The advantage of such a system is the ability to run the tests without being afraid of damaging any components.

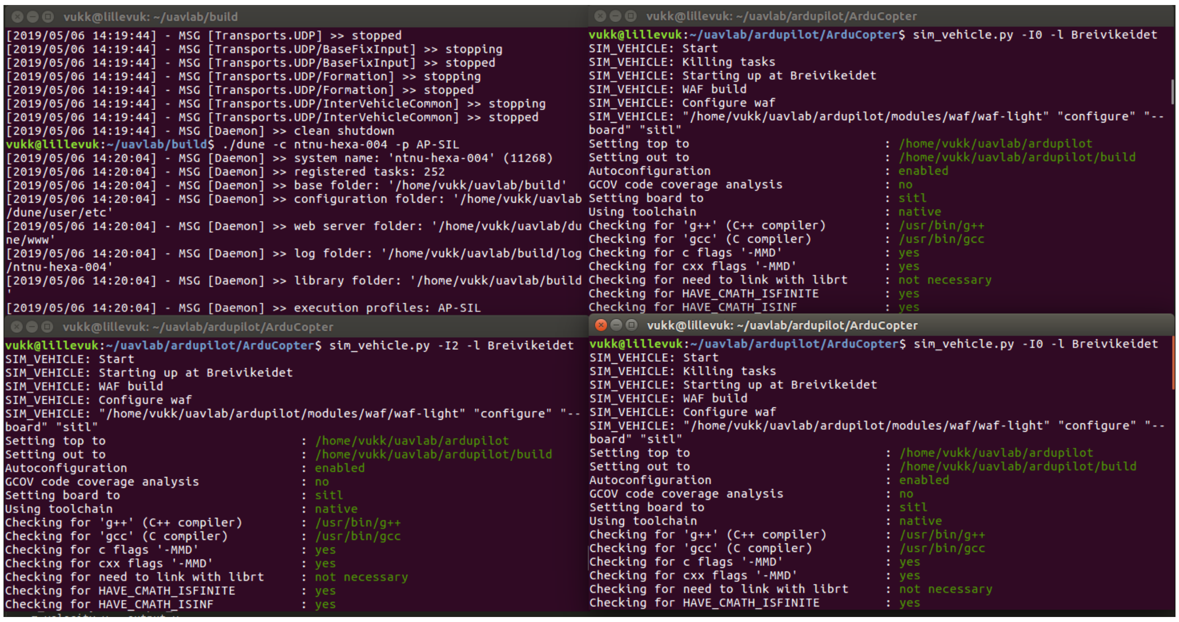
SIL can be achieved by running several configuration files in the Linux terminal. When the configuration files are running, different parts of the systems are sharing information, and a larger system is formed (see Section 2.3). The way the simulator is built in this project requires the execution of four different terminal commands in order to set up a full simulation system.

1. *sim_vehicle.py -I0 -l LOCATION* - this is a command for execution of the ArduPilot simulation software. The ArduPilot simulation software is communicating with DUNE through a TCP connection (Section 2.3). The instance number and the location has to be included in the execution command as well. The instance number is important when several TCP ports are used, which is the case when simulating several vehicles. Every instance is listening to a specific TCP port. The location can be set by writing geodetic coordinates of the desired location.

2. *sim_vehicle.py -I2 -l LOCATION* - same command as above, with the only difference being the instance number. In this project, *ntnu-hexa-004* is communicating with ArduPilot as instance 0, while *ntnu-hexa-003* is communication as instance 2. This is of course just configuration settings, and can be changed any time.

3. *./dune -c ntnu-hexa-003 -p AP-SIL* - execution command for the *ntnu-hexa-003* configuration files. This vehicle is simulating the landing platform (Section 5.1).

4. *./dune -c ntnu-hexa-004 -p AP-SIL* - execution command for the *ntnu-hexa-004* configuration files. This vehicle is simulating the actual UAV in the system.



```
vukk@lillevuk: ~/uavlab/build
[2019/05/06 14:19:44] - MSG [Transports.UDP] >> stopped
[2019/05/06 14:19:44] - MSG [Transports.UDP/BaseFixInput] >> stopping
[2019/05/06 14:19:44] - MSG [Transports.UDP/BaseFixInput] >> stopped
[2019/05/06 14:19:44] - MSG [Transports.UDP/Formation] >> stopping
[2019/05/06 14:19:44] - MSG [Transports.UDP/Formation] >> stopped
[2019/05/06 14:19:44] - MSG [Transports.UDP/InterVehicleCommon] >> stopping
[2019/05/06 14:19:44] - MSG [Transports.UDP/InterVehicleCommon] >> stopped
[2019/05/06 14:19:44] - MSG [Daemon] >> clean shutdown
vukk@lillevuk:~/uavlab/build$ ./dune -c ntnu-hexa-004 -p AP-SIL
[2019/05/06 14:20:04] - MSG [Daemon] >> system name: 'ntnu-hexa-004' (11268)
[2019/05/06 14:20:04] - MSG [Daemon] >> registered tasks: 252
[2019/05/06 14:20:04] - MSG [Daemon] >> base folder: '/home/vukk/uavlab/build'
[2019/05/06 14:20:04] - MSG [Daemon] >> configuration folder: '/home/vukk/uavlab
/dune/user/etc'
[2019/05/06 14:20:04] - MSG [Daemon] >> web server folder: '/home/vukk/uavlab/du
ne/www'
[2019/05/06 14:20:04] - MSG [Daemon] >> log folder: '/home/vukk/uavlab/build/log
/ntnu-hexa-004'
[2019/05/06 14:20:04] - MSG [Daemon] >> library folder: '/home/vukk/uavlab/build
'
[2019/05/06 14:20:04] - MSG [Daemon] >> execution profiles: AP-SIL
```

```
vukk@lillevuk: ~/uavlab/ardupilot/ArduCopter
vukk@lillevuk:~/uavlab/ardupilot/ArduCopter$ sim_vehicle.py -I0 -l Breivikeidet
SIM_VEHICLE: Start
SIM_VEHICLE: Killing tasks
SIM_VEHICLE: Starting up at Breivikeidet
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/vukk/uavlab/ardupilot/modules/waf/waf-light" "configure" "--
board" "sitl"
Setting top to                           : /home/vukk/uavlab/ardupilot
Setting out to                           : /home/vukk/uavlab/ardupilot/build
Autoconfiguration                        : enabled
GCOV code coverage analysis              : no
Setting board to                         : sitl
Using toolchain                          : native
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for c flags '-MMD'              : yes
Checking for cxx flags '-MMD'            : yes
Checking for need to link with librt     : not necessary
Checking for HAVE_CMATH_ISFINITE         : yes
Checking for HAVE_CMATH_ISINF            : yes
```

```
vukk@lillevuk: ~/uavlab/ardupilot/ArduCopter
vukk@lillevuk:~/uavlab/ardupilot/ArduCopter$ sim_vehicle.py -I2 -l Breivikeidet
SIM_VEHICLE: Start
SIM_VEHICLE: Starting up at Breivikeidet
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/vukk/uavlab/ardupilot/modules/waf/waf-light" "configure" "--
board" "sitl"
Setting top to                           : /home/vukk/uavlab/ardupilot
Setting out to                           : /home/vukk/uavlab/ardupilot/build
Autoconfiguration                        : enabled
GCOV code coverage analysis              : no
Setting board to                         : sitl
Using toolchain                          : native
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for c flags '-MMD'              : yes
Checking for cxx flags '-MMD'            : yes
Checking for need to link with librt     : not necessary
Checking for HAVE_CMATH_ISFINITE         : yes
Checking for HAVE_CMATH_ISINF            : yes
```

```
vukk@lillevuk: ~/uavlab/ardupilot/ArduCopter
vukk@lillevuk:~/uavlab/ardupilot/ArduCopter$ sim_vehicle.py -I0 -l Breivikeidet
SIM_VEHICLE: Start
SIM_VEHICLE: Killing tasks
SIM_VEHICLE: Starting up at Breivikeidet
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/vukk/uavlab/ardupilot/modules/waf/waf-light" "configure" "--
board" "sitl"
Setting top to                           : /home/vukk/uavlab/ardupilot
Setting out to                           : /home/vukk/uavlab/ardupilot/build
Autoconfiguration                        : enabled
GCOV code coverage analysis              : no
Setting board to                         : sitl
Using toolchain                          : native
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for c flags '-MMD'              : yes
Checking for cxx flags '-MMD'            : yes
Checking for need to link with librt     : not necessary
Checking for HAVE_CMATH_ISFINITE         : yes
```

Figure 6.1: Four terminals running SIL

6.2 Tuning

6.2.1 Altitude Controller

The altitude controller is tuned using the Ziegler-Nichols tuning method, described in Section 4.1.2. In the first step, both the integral part and the derivative part of the controller are set to zero, to find the critical gain of the proportional part that gives standing oscillations. The critical gain found to give standing oscillations is $K_c = 2.9$. The first subplot in Figure 6.2 shows the response of the standing oscillations. The critical period of the oscillations is found to be $T_k = 5.2$ sec.

Further, Table 4.1 is used to find possible tuning parameters for the altitude controller. The possible parameters are listed in Table 6.2.1.

Table 6.1: Altitude controller parameters found by Ziegler-Nichols tuning approach

	No overshoot	Some overshoot
K_p	0.97	0.58
K_i	0.22	0.37
K_d	1.01	1.51

It is more important having an accurate altitude controller, than having a controller giving fast response. Therefore, tuning variables from Table 6.2.1 giving a response with no overshoot are chosen for further tuning. Still, an overshoot was hard to avoid just by tuning controller parameters. Table 6.2 shows final variables found for the altitude controller. The response of the final tuning variables is represented in the second subplot in Figure 6.2.

Table 6.2: Final tuning parameters altitude controller

Parameter	**Value**
K_p	1.3
K_i	0.3
K_d	0.7

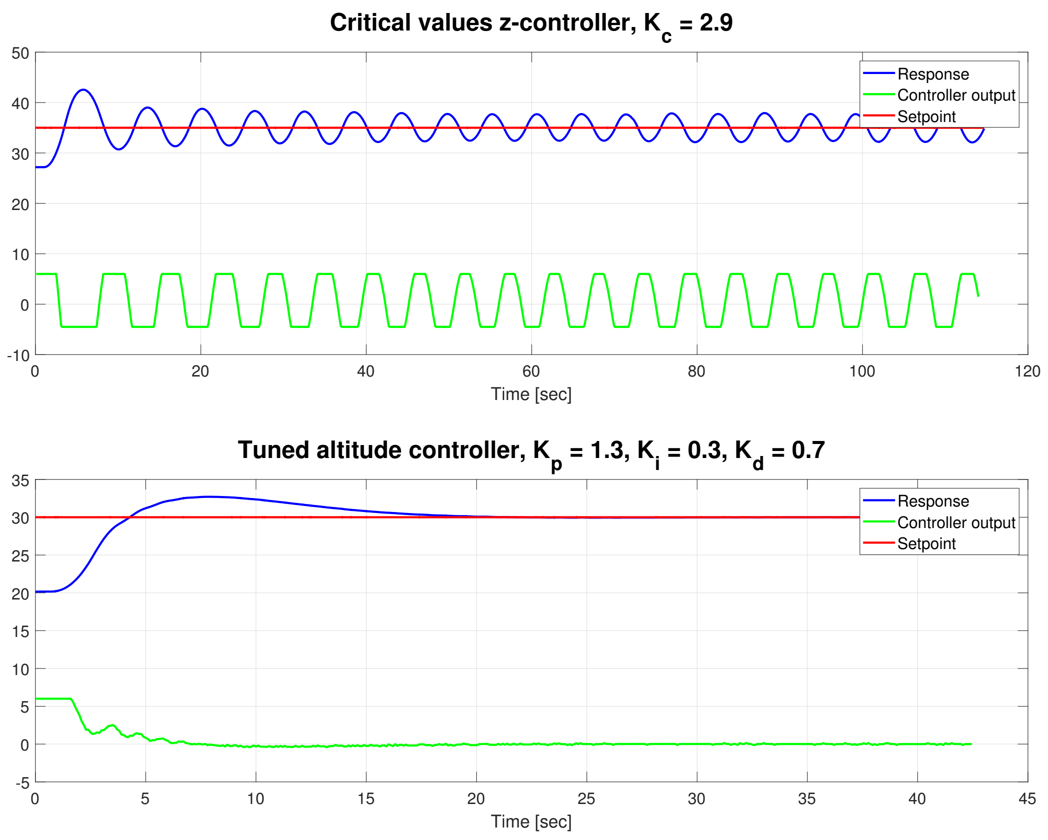


Figure 6.2: Tuning of altitude controller

Overshoot can be noticed in the response represented in Figure 6.2. To remove the overshoot, anti-windup is implemented to the altitude controller. The wind-up, or overshoot, occurs when the integral term of the PID controller (Section 4.1.1) contributes to a higher output than the physical system can provide (Section 6.3). To prevent the wind-up, one can feedback signal to the integrator as soon as it starts to saturate. The anti-windup block diagram is shown in Figure 6.3.

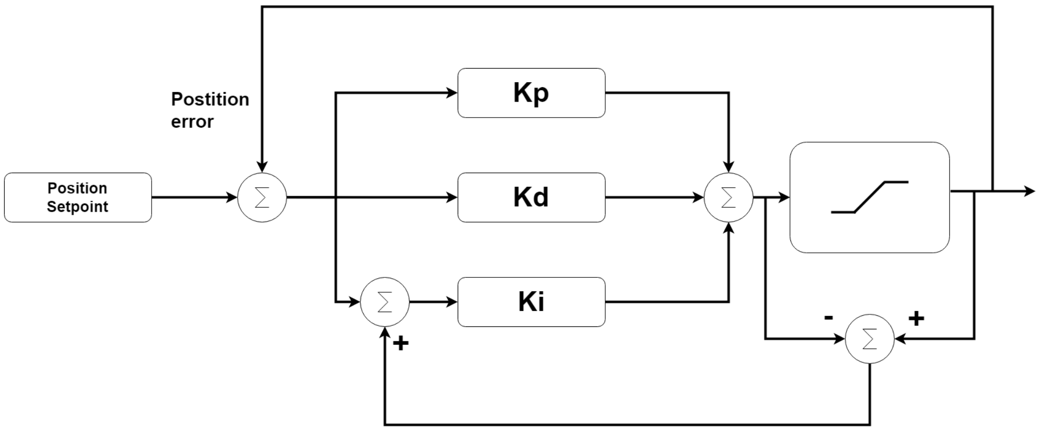


Figure 6.3: Anti wind-up block diagram

The signal feedback is preventing the integrator from providing a higher output than allowed in the system. The important part when implementing anti-windup is knowing the exact limits of the system. Limitations of the system used in this project are investigated, and represented in Table 6.7. The controller response after the anti-windup implementation is represented in Figure 6.4.

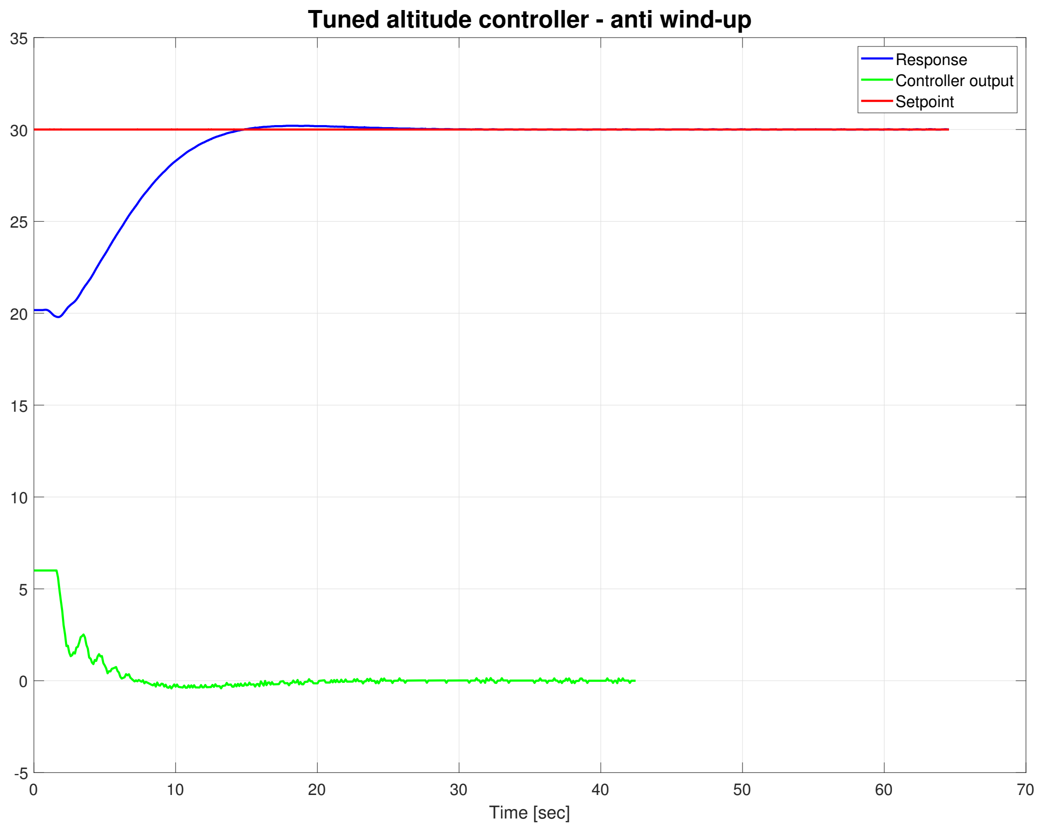


Figure 6.4: Altitude controller with anti-windup implemented

6.2.2 Horizontal Controllers

The horizontal controllers are tuned separately, giving one set of tuning variables for each horizontal direction. Like for the altitude controller, the Ziegler-Nichols method is used to find a suggestion for the values of the tuning parameters. In the specialization project, the horizontal controllers were tuned with two sets of parameters. One set was resulting in an aggressive response, while the other set was resulting in a slow, but a more robust response. The idea was to use the slow parameters while hovering and the aggressive parameters when getting closer to the target. That idea is not used in this project. The reason is too uncertain behavior when switching between controllers. Therefore, the horizontal controllers are only be tuned with one set of parameters each.

For the x-direction controller, the critical oscillations were triggered by the critical gain $K_c = 1.8$. The response is shown in the first subplot of Figure 6.5. The critical period of the critical oscillations is $T_c = 7.2$ sec.

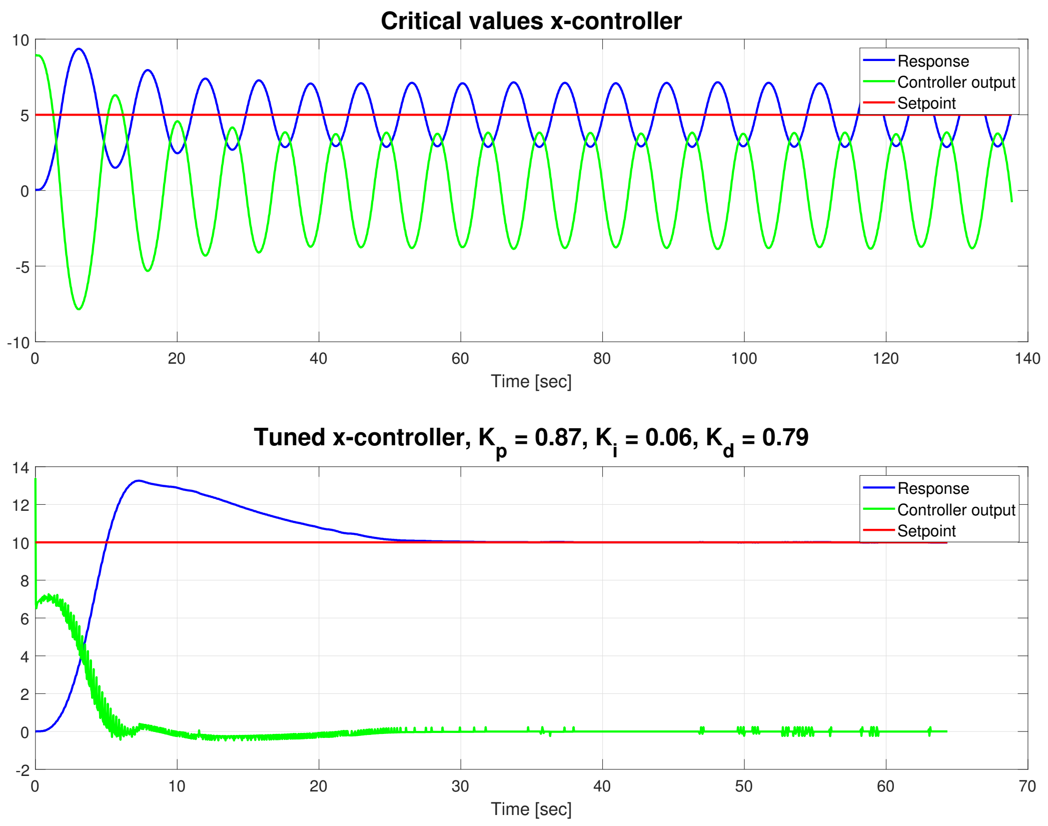


Figure 6.5: Tuning x-direction controller

Table 4.1 is used to find suggestions for the controller parameters.

Table 6.3: X-direction controller parameters found by Ziegler-Nichols tuning approach

	No overshoot	**Some overshoot**
K_p	0.36	1.08
K_i	0.1	0.3
K_d	0.86	0.97

The final tuning parameters are found by a mix between suggestions for the parameters supposed to give a response with some overshoot, and the parameters supposed to give response without overshoot. The mix is chosen because the horizontal controllers have to respond fast to wind gusts and sudden wave occurrences. At the same time, the overshoot has to be limited such that the UAV does not drift far away from the target. The final parameters are listed in Table 6.4.

Table 6.4: Final tuning parameters x-direction controller

Parameter	**Value**
K$_p$	0.87
K$_i$	0.06
K$_d$	0.79

The response of the parameters from Table 6.4 is presented in the second subplot of Figure 6.5. An overshoot can be noticed in the response. The maximal value of the overshoot is 30% above the desired value. It was not possible to implement an anti-windup solution to remove the overshoot, because the controller output never saturates. One has to keep in mind that the step response presented in Figure 6.5 is from 0 to 10 meters. Such large responses will not occur in real life. Still, the overshoot can represent a problem in some cases, and a different tuning approach may be necessary.

The y-direction controller is tuned using the same approach as the x-direction controller. Because of similar equations of motion of the horizontal directions (3.20), tuning parameters were expected to be similar for the horizontal directions. The critical oscillations for the y-direction controller were triggered by the critical gain $K_c = 1.62$. The oscillations are presented in the first subplot of Figure 6.6.

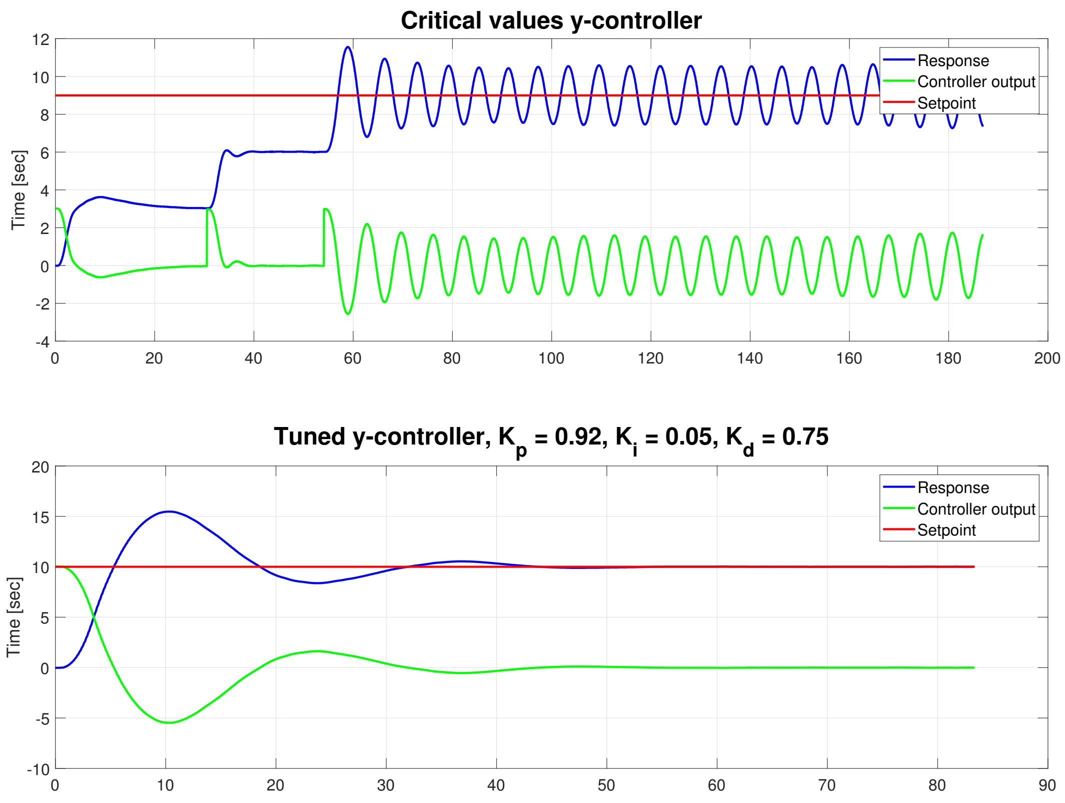


Figure 6.6: Tuning y-direction controller

Suggestions for tuning variables are listed in Table 6.5.

Table 6.5: Y-direction controller parameters found by Ziegler-Nichols tuning approach

	No overshoot	**Some overshoot**
K_p	0.36	1.08
K_i	0.1	0.3
K_d	0.86	0.97

The same requirements as for the x-direction holds for the y-direction. A mix of the suggested variables is used to find the final values, which are listed in Table 6.6.

Table 6.6: Final tuning parameters y-direction controller

Parameter	**Value**
K$_p$	0.92
K$_i$	0.05
K$_d$	0.75

The response of tuning parameters from Table 6.6 is presented in the second subplot of Figure 6.6. Comparing to response of the x-controller (Figure 6.5), the y controller oscillates more, but has a faster response.

6.3 Limitations

6.3.1 Velocities

Velocity limitations of the hexarotor (Section 2.2.1) are investigated by the Norwegian Defence Research Establishment [20] and the parameters found are presented in Table 6.7.

Table 6.7: Velocity limitations of the hexarotor

Max ascent velocity	4 m/s
Max descent velocity	2.5 m/s
Max flight velocity	10 m/s

Horizontal velocity limitations from Table 6.7 are used in this project, while the vertical limitations are decreased, to get better vehicle control during the landing process. The vertical velocity limitations used in this task are presented in Table 6.8.

Table 6.8: Decreased vertical velocity limitations

Max ascent velocity	4 m/s
Max descent velocity	2.5 m/s

6.3.2 Platform Size

One of the parts of the project was to find the necessary size of the landing platform, based on the simulation results of the controllers. The preferred size of the landing platform is as small as possible. In order to find the minimum requirement for the landing platform size, total horizontal error during the simulations is investigated closer. A collection of horizontal errors during simulations is presented in Figure 6.7. Simulations in Figure 6.7 are simulated with 4 m/s constant wind speed, and sea state 4.

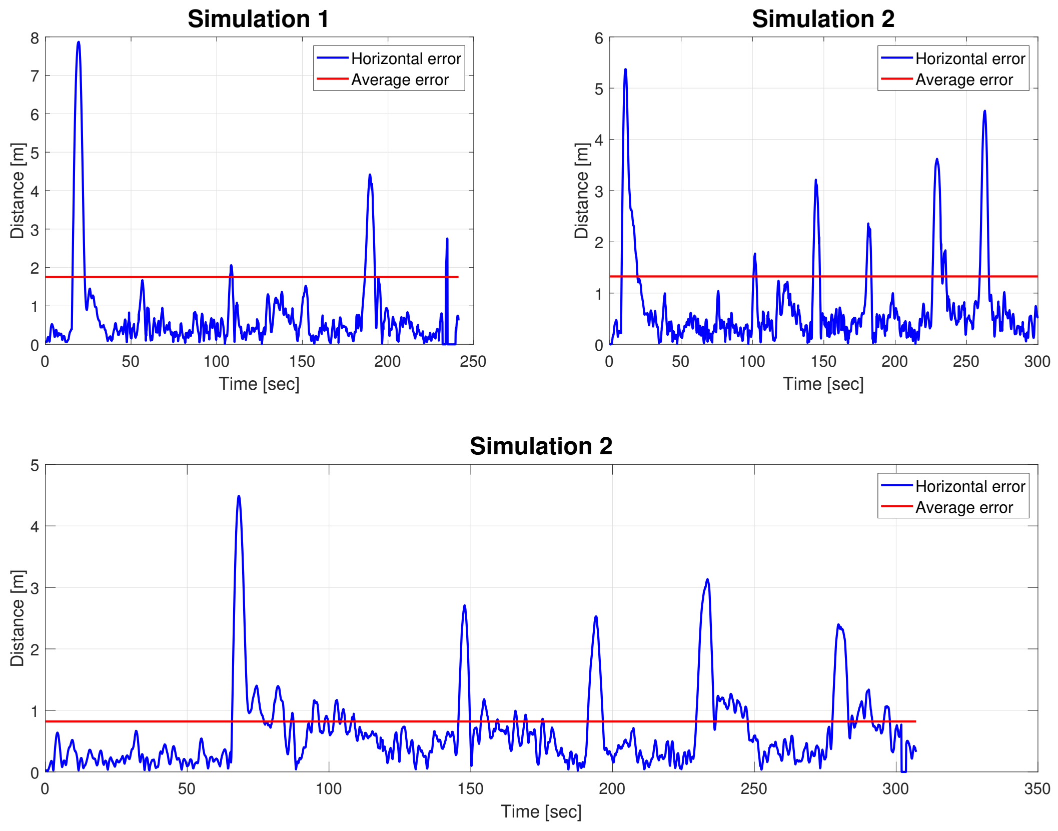


Figure 6.7: Total horizontal error in simulations

The platform radius can not be smaller than the average horizontal error. Studying Figure 6.7, error peaks can be noticed. The peaks occur due to sudden turns of the ship carrying the platform. The path of the ship during the simulations will be presented later in the report.

Average horizontal error of the simulations in Figure 6.7 varies from 0.81m to 1.75m. The figure shows that the average error is highly influenced by the peaks. There are about 50 seconds between each peak, which is enough time to perform a landing. Explained, as long the landing is performed in the periods between error peaks, it is enough to decide platform size based on the horizontal error between peaks. Simulation 1 has the largest average horizontal error. Still, looking at the figure it is noticeable that the error is under the average value most of the time. Based on the simulations, a decision is made that the minimal platform radius should be **1.5m**.

6.3.3 Maximal Wind

It is important to investigate the system behavior when the wind is applied to find constraints for the maximal allowed wind to perform a flight. First, the relation between the roll and pitch response and the wind is investigated. This relation is important to investigate, having in mind that the errors were discovered on this topic in the specialization project. The error was that pitch and roll did not respond to the constant wind in order to keep the desired position. The position was kept the same, without any response in the pitch and roll, which is physically impossible.

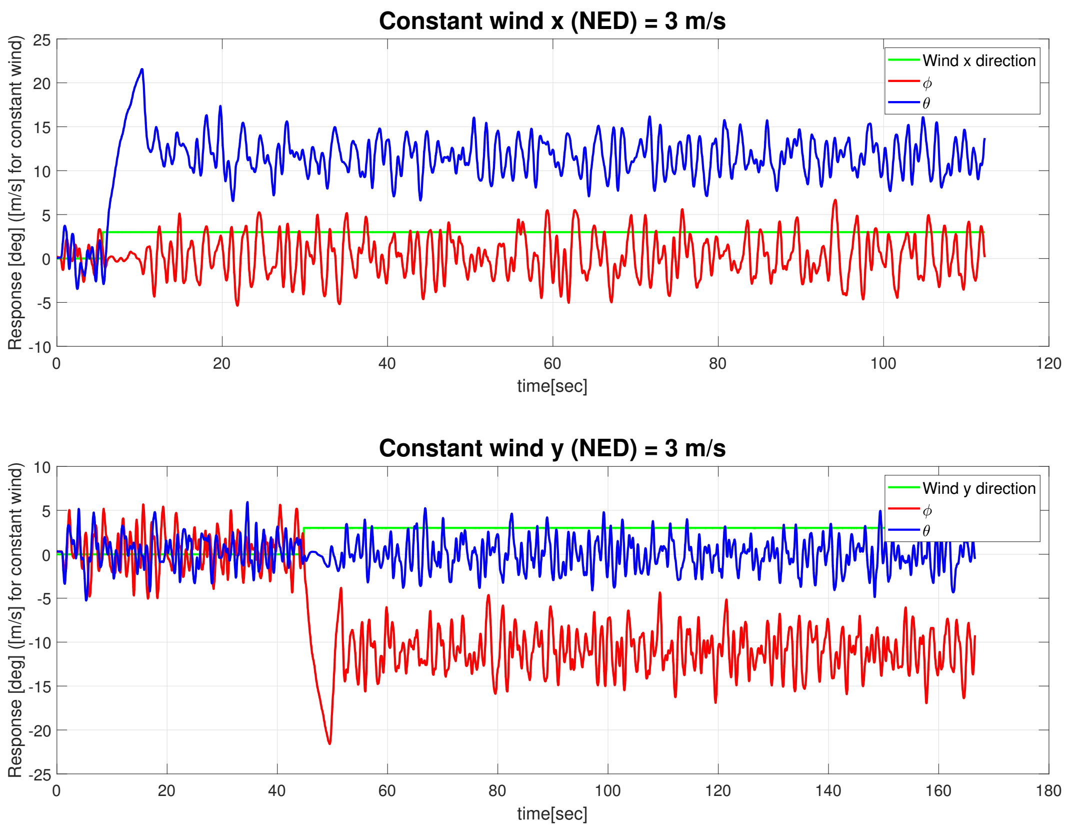


Figure 6.8: Roll and pitch response on the constant wind

Figure 6.8 shows roll and pitch response to constant wind in x and y direction of the NED frame (Section 3.1.1). A response in both roll and pitch can be noticed in the figure. For example, for constant wind in the x-direction of the NED frame, the vehicle responds with a pitch value oscillating slowly about 12 degrees. Responses presented in Figure 6.8 indicates that the roll and pitch response problem is fixed.

The maximal allowed wind is investigated by slowly increasing the constant wind velocity in both horizontal directions. At some point, the UAV has to give up for the wind forces, due to physical limits. The response of slowly increasing the wind is shown in Figure 6.9.

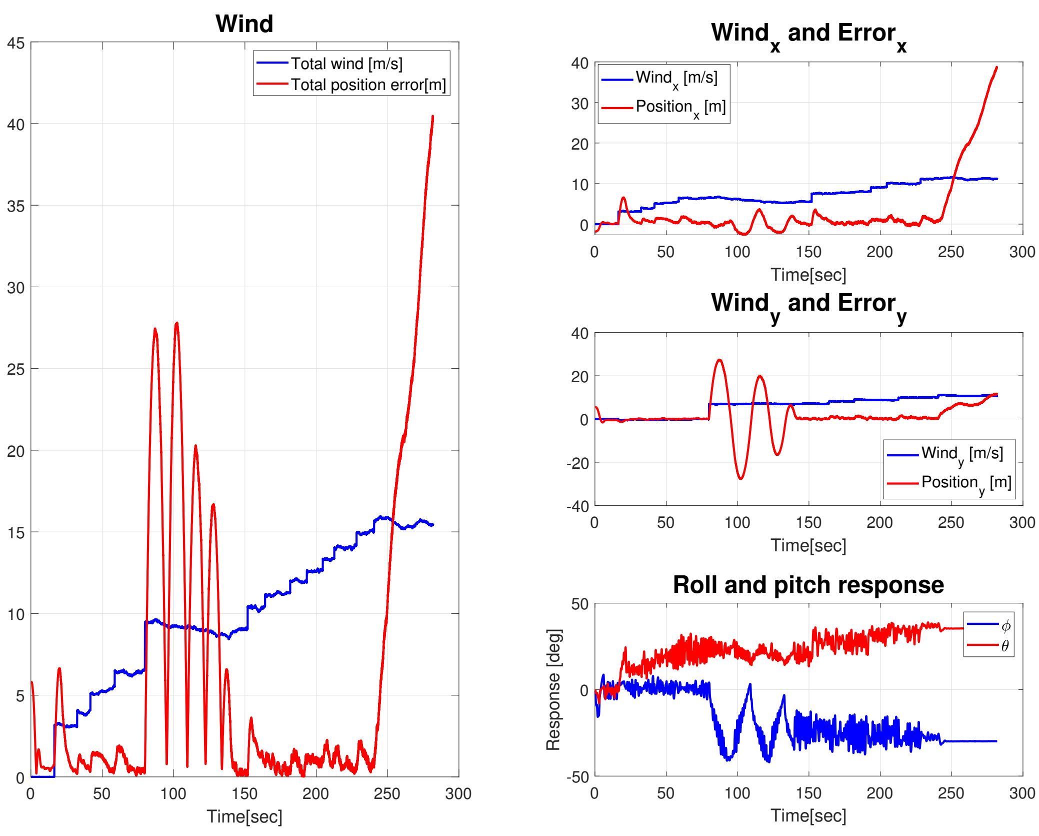


Figure 6.9: Test of maximal wind threshold

Figure 6.3.3 presents results showing that the UAV, with the control system tuned as described in Section 6.2, can handle total horizontal wind up to 15 m/s. According to the Beaufort scale, 15 m/s wind has number 7[35].

Number 7 on the Belfort scale is known as *High wind*, with a characteristic behavior of setting whole trees in motion. The maximal wind advise from the Norwegian Defence Research Establishment is 8 m/s. The total horizontal wind is measured as $wind_{total} = \sqrt{wind_x^2 + wind_y^2}$. The contribution from each direction is 7.5 m/s, which is close to the Norwegian Defence Research Establishment advise.

6.4 Filter

Figure 6.10 shows a simulation of filter performance test. The filter design is described in Section 6.4. The tests considers sea states 2 to 7 from Section 3.5.

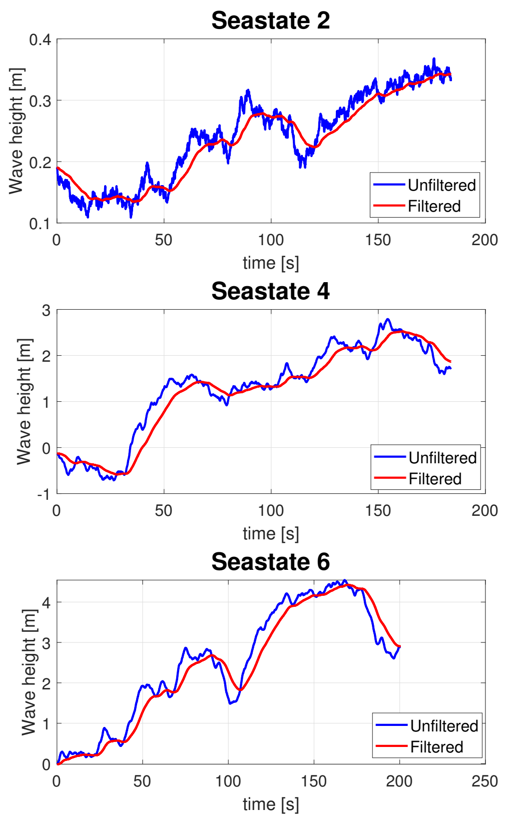


Figure 6.10: Filter performance

Timulation results shows that the low-pass filter performance on damping high frequencies is satisfying. Looking at lower state simulations, it is clear that the filter damps all the high-frequency oscillations, giving a smooth input to the altitude controller. A drawback is the time delay of the filter. The delay is easy to recognize looking at the sea state 7 simulations. The filter time delay is measured to be 1.2 seconds. A solution to the delay problem is to implement another type of filter. Still, the delay is not considered as a big problem, as the filter is only going to be used when the height distance between the vehicle and platform is large.

6.5 Results

The simulations presented in this chapter will start with simulations where the target ship is moving in a straight line, with constant velocity, without any disturbances present. The environmental disturbances are added gradually to the later presented simulation results. The reason is to give the reader an idea in which disturbances are affecting the controller most.

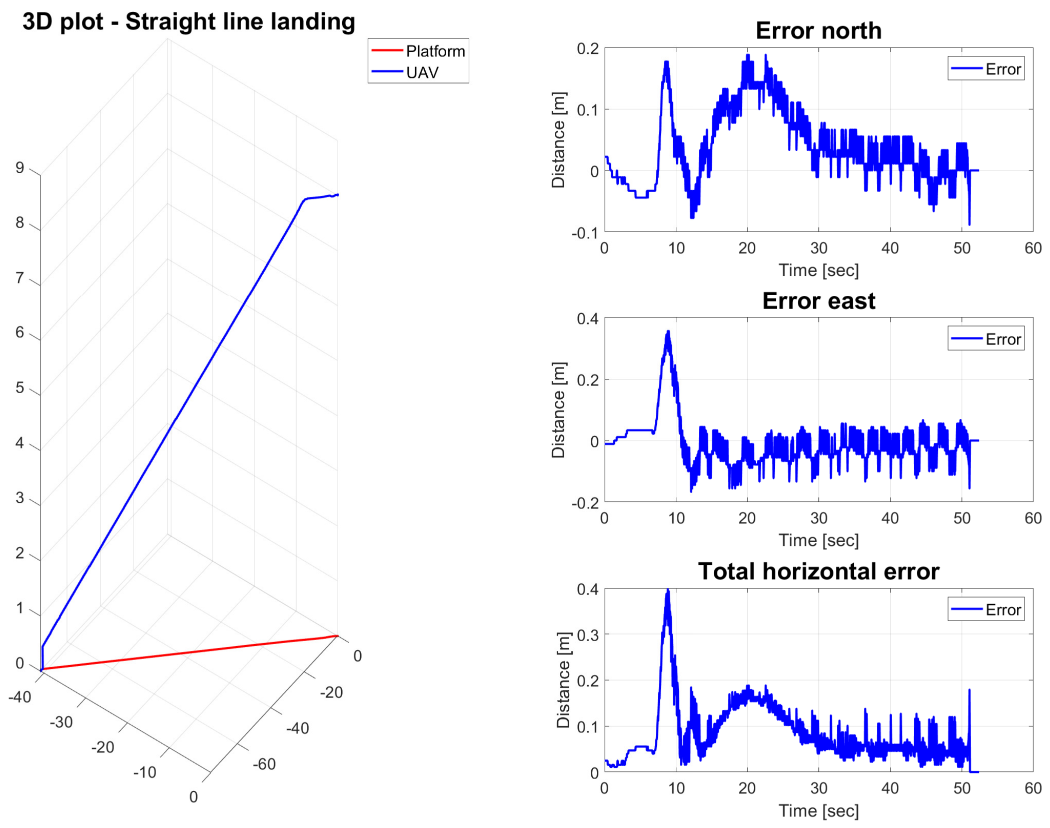


Figure 6.11: Straight line landing

Figure 6.11 shows the results of a straight line landing simulation where the target is moving forward with constant velocity, without any environmental disturbances present. As the plot shows, the landing is performed without any problems. The horizontal error is close to zero, with a noticeable error in the beginning, when the target suddenly starts moving. The result is expected, as there are no disturbances present to challenge the controllers.

For the next simulation results that are going to be presented in the report, the target ship is following the path showed in Figure 6.12.

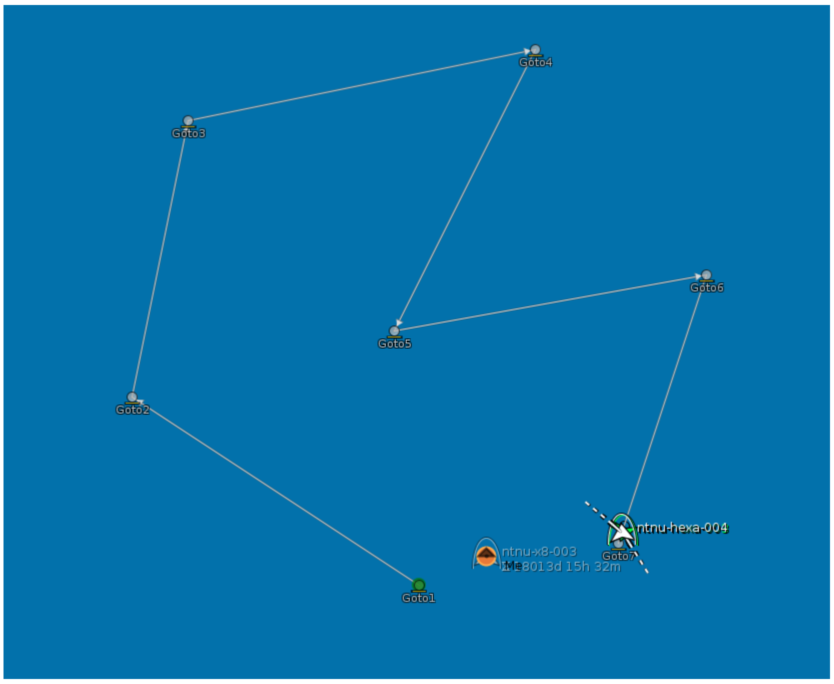


Figure 6.12: Moving path for target vehicle during simulations

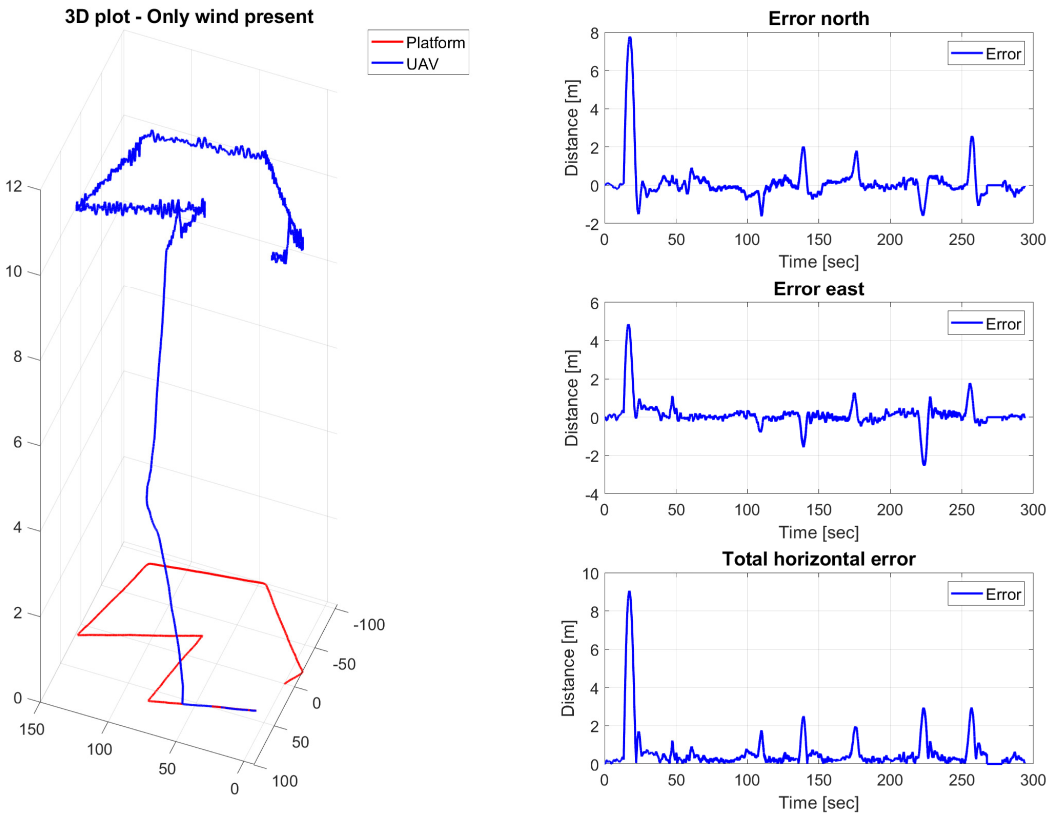


Figure 6.13: Simulation results with only wind present

The results of a simulation with only wind present are shown in Figure 6.13. In addition to wind gusts, 4m/s constant wind is added in both horizontal directions. The large error peaks at the beginning of the simulation is a result of the wind getting turned on suddenly. The horizontal error peaks that occur periodically can be noticed in the plots. The reason is the velocity direction changes of the target (Figure 6.12). Overall, the simulation results are satisfying as the total horizontal error rarely exceeds 1m, which is acceptable considering the landing platform size. It is also worth mentioning that the state machine is not a part of the shown simulation. The focus is on horizontal controllers.

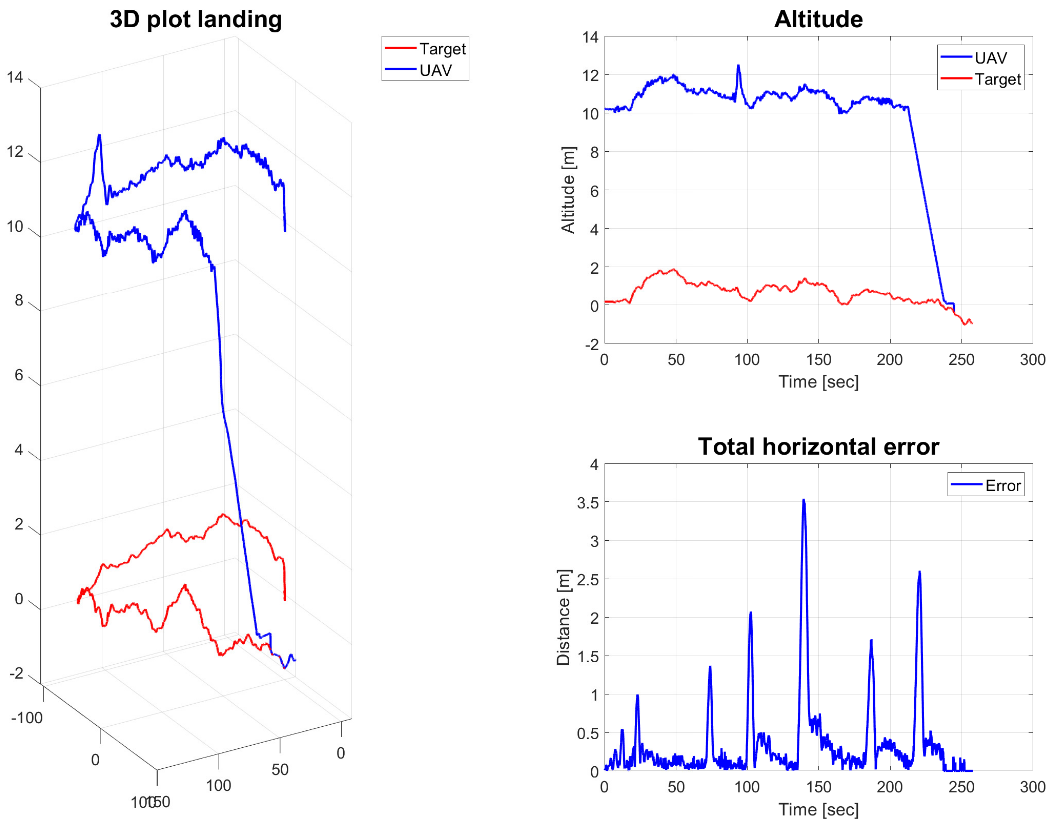


Figure 6.14: Simulation results with only waves present, without wave filter

Figure 6.14 presents the results of a landing approach with only waves present. The wave filter is not used for this simulation. Like in Figure 6.13, horizontal error peaks due to target velocity change can be noticed. There are also many small altitude oscillations. The reason is that the wave filter is not turned on and the vehicle follows every single wave movement.

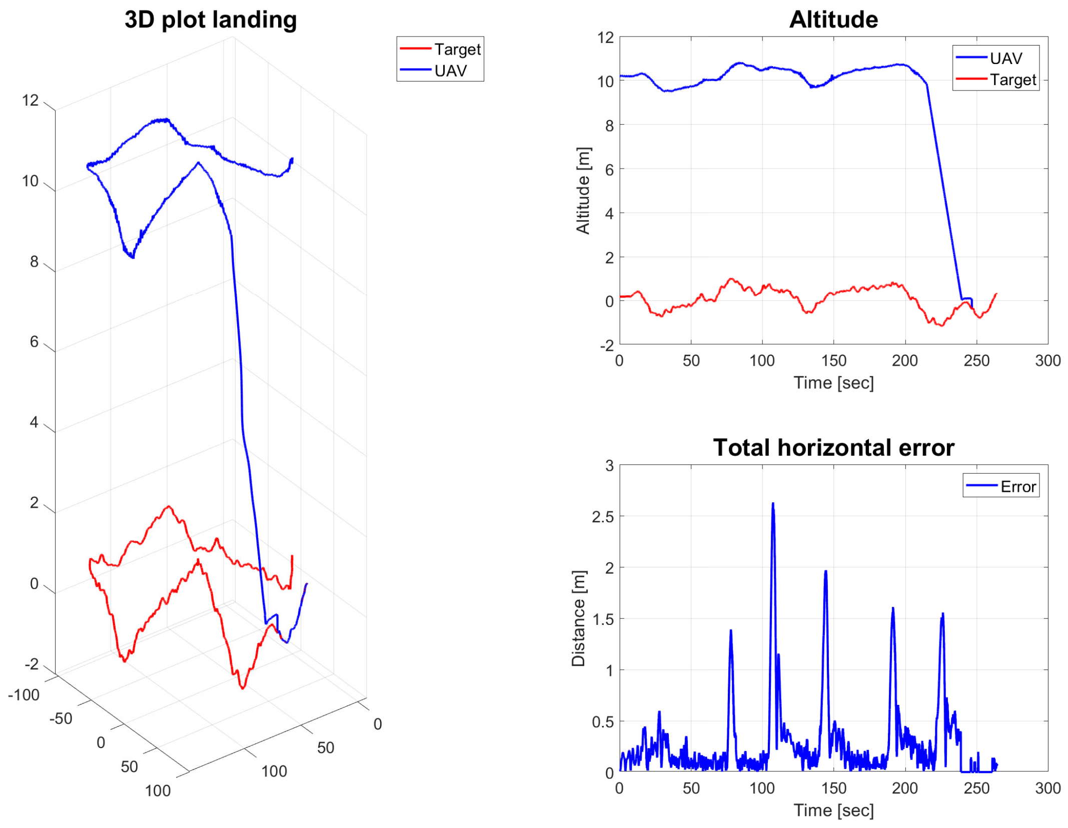


Figure 6.15: Simulation results with only waves present, with wave filter turned on

The simulation results of a landing approach with only wave disturbances present and wave filter turned on are shown in Figure 6.15. Compared to Figure 6.14, the altitude oscillations are removed, and the vehicle only follows the large wave movements. A small time delay between the wave movements and the UAV response can be noticed. The reason for the delay is the time delay of the filter. Still, the delay is acceptable as the filtered reference is only going to be used when the UAV is located high above the target.

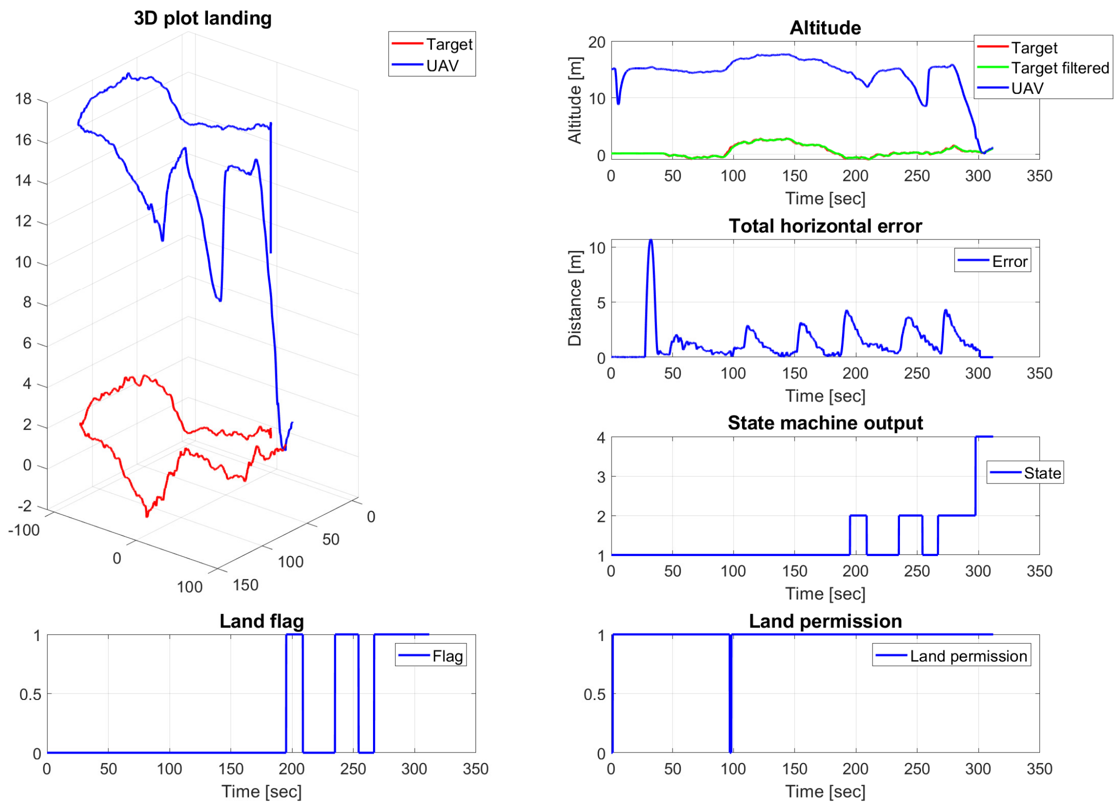


Figure 6.16: Simulation results of the complete system

The full system landing, with all environmental disturbances and state machine implemented is presented in Figure 6.16. The presented result is simulated with a constant wind velocity of 4m/s in both x and y direction, and sea state 4. As in all other simulation results presented in this chapter, horizontal error peaks can be noticed on the points where the target changes the velocity direction (Figure 6.12). The state machine reliability is tested by two manual landing abortions by the user, which can be recognized after 200 and 250 seconds of the altitude subplot in Figure 6.16. The state machine response on manual landing abortions is satisfying, as the vehicle returns quickly and smoothly to the hover altitude.

Analyzing the state machine, the result seems satisfying between State 1 and State 2. The state machine is changing states as expected on users landing commands. About 300 seconds, when the vehicle is located about 4 meters above the target, the state machine goes directly from State 2 to State 4. The reason is that all the counters and the landing permission requirements were fulfilled at the point the State 3 was entered. The state skip is deeper discussed later in the report.

More simulations of the full system are presented in Appendix D.

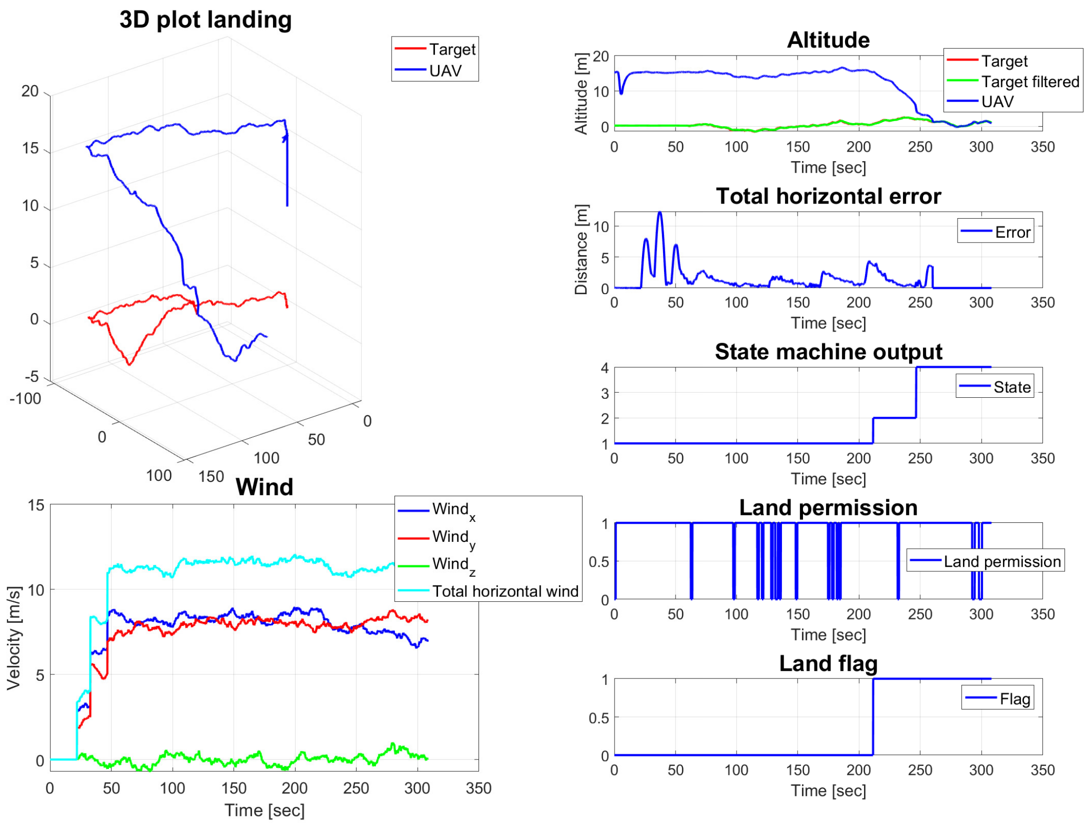


Figure 6.17: Simulation results of the full system i rough weather conditions

Figure 6.17 presents results of a simulation under rough weather conditions. Constant wind in x and y direction is 8 m/s. That makes the total horizontal wind varying about 12 m/s, which is close to the maximal wind (Section 6.9). The sea state simulated was 8 (Section 3.4). As expected, the average horizontal error was larger than the error in previous simulations in this section because of the rougher environmental disturbances. The land permission flag requirements are less satisfied than other simulations presented in this section. The reason is that sea conditions. Sea state 8 involves frequent wave height changes compared to sea state 4, used in the other simulations presented in this section.

Chapter 7

Field Testing

7.1 Preparation

The tests are held at Udduvoll, a small airport close to Trondheim, Norway. As a normal procedure for all tests involving UAVs at NTNU, a mission acceptance form has to be filled out. The form is a formal contract between the student and the university. An example of the Mission Acceptance form can be found in Appendix E. As a part of the preparation, all components have to be proven working. It is extremely important to check the GPS precision, and confirm RTK positioning software accuracy. In addition, a certified pilot has to go through the system and approve that it is ready for testing.

7.2 Setup

Compared to the simulation system (Section 6), small modifications are done for field testing. The idea of using another hexacopter simulating the landing platform (Section 5.1) is used for the field testing also. It is done by installing *ntnu-hexa-004* software to a BeagleBone black (Section 2.2.3). The BeagleBone is connected to the ground station via an Ethernet cable. A GNSS antenna is attached to the GNSS receiver, which is attached to the BeagleBone using a 6m long wire. The software of *ntnu-hexa-003* is installed on the hexarotor's Beaglebone black. The rest of the connections and communication between the UAV and ground are done as described in Figure 2.9. According to the idea from Section 5.1, *ntnu-hexa-004* is assigned the master role in the system, while *ntnu-hexa-003* is assigned the slave role in the system. The reason why the field testing is done by using two vehicles in addition to the ground station is that the ground station frame can be used as a reference frame. Figure 7.1 shows coordinate frames from the field test relative to each other.

One option is to use the ground station as both the landing platform and reference frame. The drawback with that option is that the vertical platform movement is not accurate enough, and the velocity cannot be measured in the NED frame.

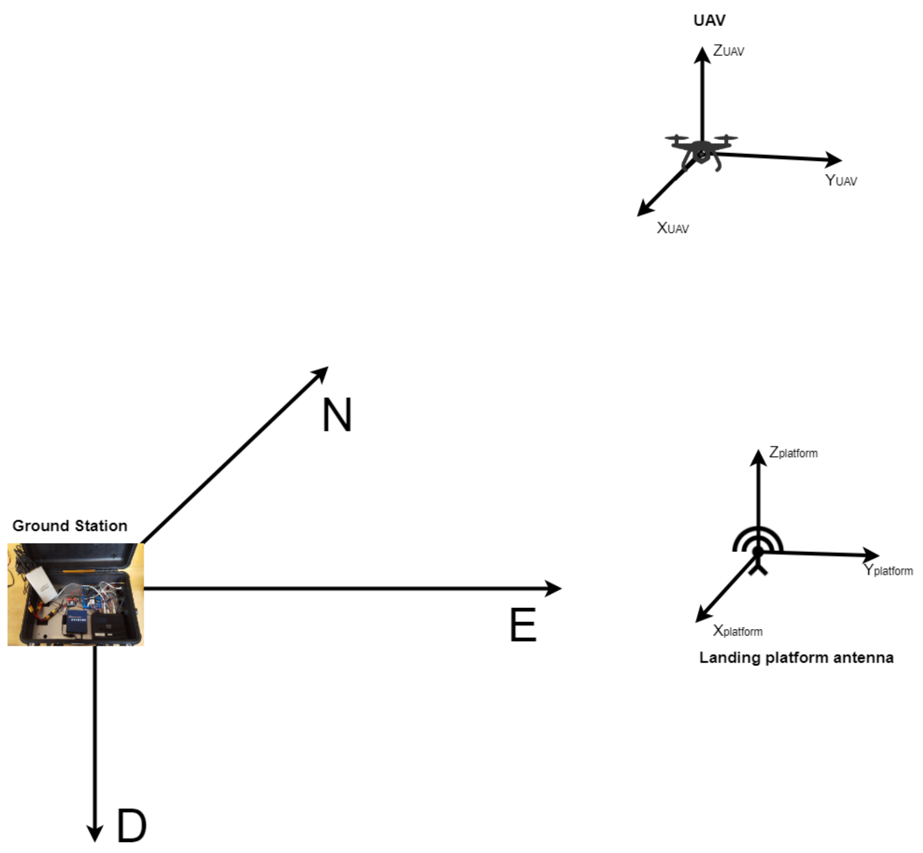


Figure 7.1: Coordinate frame setup for the field test

A third GNSS antenna for the landing platform will provide the landing platform position and the velocity measurements relative to the ground station, using the RTK positioning. Using the RTK positioning between the ground station and the UAV will provide position and velocity measurements of the UAV relative to the ground station as well. As the ground station is a NED frame (Figure 7.1), all measurements given relative to the ground station can be read by the ArduPilot.

Section 5.1 describes, among other things, how *IMC::EstimatedLocalState* messages transports *IMC::EstimatedState* messages from the master vehicle to the slave vehicle. A modification is done in the way that *IMC::EstimatedState* messages are replaced with *IMC::GpsFixRtk* messages, meaning that *IMC::EstimatedLocalState* messages are sending messages of position and velocity relative to the ground station, which can be used as input to the controllers. The controllers are also using *IMC::GpsFixRtk* messages during the field testing, instead of *IMC::EstimatedState* messaged used during the simulation. Again, the reason is because all coordinates has to be given relative to the same frame, which is chosen to be the ground station frame.

For safety reasons, the field tests are not performed over water. The landing platform is imitated by attaching the GNSS antenna at the end of a 6m long fishing rod (Figure 7.2). The fishing rod gets swayed in all directions, imitating waves. The movements are not the perfect copy of actual wave movements, but the imitation is good enough to make an error in all directions, and test the PID controller. The controllers implemented on the UAV are described in Section 5.3.

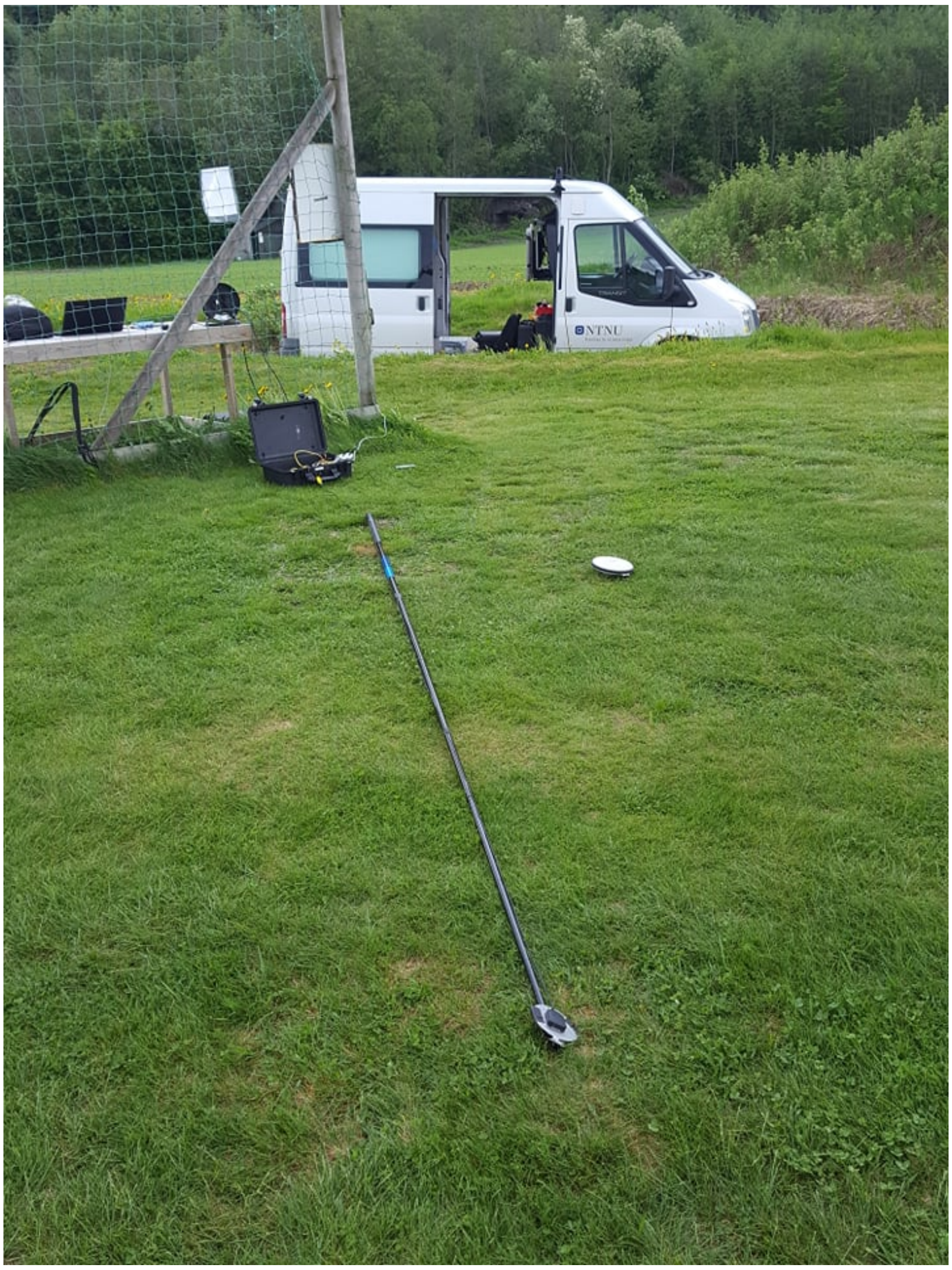


Figure 7.2: Field test setup

7.3 Results

7.3.1 Test 1

The initial test was carried out using parameters from Table 6.4, 6.6 and 6.2. Figure 7.3 shows the UAV position response from the test. The test is performed with stationary target, without any target movement. For the initial test, an offset of 15m in the North direction was included to have a reasonable safety distance between the hexacopter and the landing platform hardware. All controller outputs presented in Figure 7.3 are the velocity outpus form the PID controllers presented in Section 4.

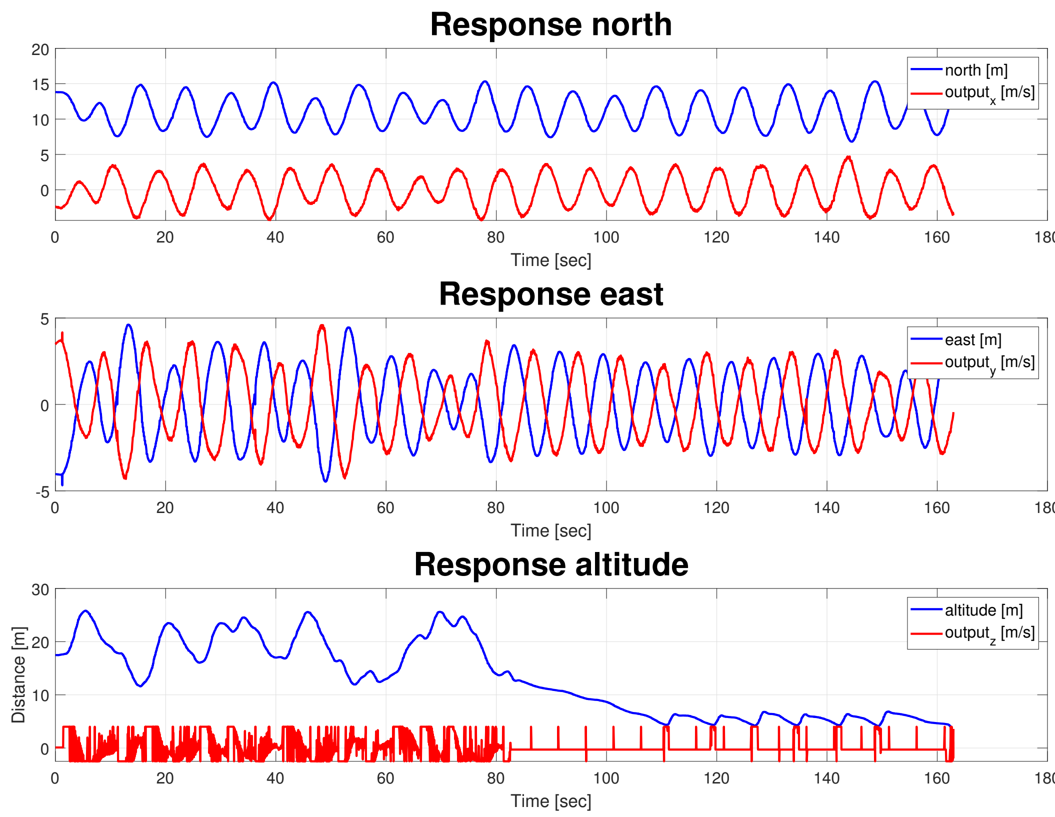


Figure 7.3: Position response using parameters from Section 6.2

It is noticeable that the horizontal responses are dominated by oscillations. The amplitude of the oscillations can be considered as equal, meaning that the system is marginally stable. Looking at the phase between the controller output and the response, the phase difference can be roughly measured to 200°, for both x- and y-direction. A 200° phase difference between the controller output and the response is indicating something in between P and I oscillations, according to [36], meaning that K_p and K_d values used for the test are too high.

The altitude does not consist of any recognizable patterns. Still, it can be noticed that the controller is working aggressively when trying to hold hover height (first 80 seconds). The controller is consequently calculating outputs leading to saturation. Saturation may be an indication that the controller is too aggressive tuned, meaning that the tuning variables are too high. The period from 80 to approximately 110 seconds is the descending period of the vehicle. During the descent period, the desired descent velocity is constant (Section 5.4), which is working well based on teh results from Figure 7.3. The period from 110 to 160 seconds is affected by repeating sequences caused by a state machine stuck between two states because of the horizontal controllers not being able to hold the desired position.

The anti wind-up, implemented as described in Section 4.2.1, is having a large impact on the control output in Figure 7.3. The constantly saturating control output values are trigging the anti-windup, affecting the integral output. Wrong tuning may lead to a loop of anti-windup feedback to the integral controller, causing wrong integral output.

Since the horizontal response from Figure 7.3 indicated too aggressive tuning parameters, a decision was made to try decreasing parameters mid-air to get a better response. The response from the mid-air tuning process is shown in Figure 7.4.

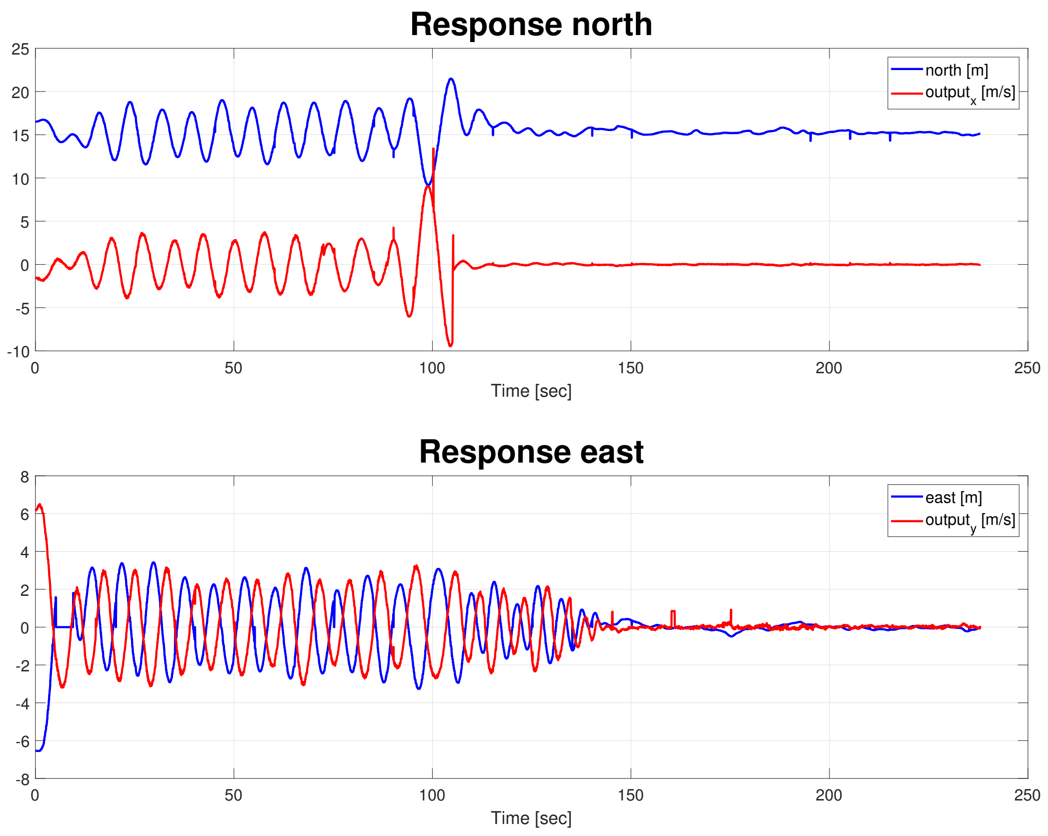


Figure 7.4: Mid-air controller tuning

The tuning in Figure 7.4 is done by trial-error approach, having in mind that the oscillations in Figure 7.3 are caused by too high K_p and K_d parameters. In the beginning, the variables were decreased gradually. After approximately 100 seconds, a decision was made to drastically decrease all variables of the x-controller, ending up with parameters listed in Table 7.1. After approximately 150 seconds, y-controller variables were replaced with parameters from Table

7.1, meaning that equal tuning parameters were assigned to the horizontal controllers. The response with the new parameters gave a satisfying horizontal response, within the landing platform boundaries found in Section 6.3.2.

Table 7.1: Horizontal control parameters found after mid-air tuning

Parameter	**Value**
K_p	0.2
K_i	0.01
K_d	0.2

The response from Figure 7.4 gave an indication that the vertical response could be improved by simply decreasing tuning parameters. The new altitude controller parameters are listed in Table 7.2 and the response using the parameters is presented in Figure 7.5. In addition, the anti wind-up was turned off for the test presented in Figure 7.5.

Table 7.2: New altitude controller parameters

Parameter	**Value**
K_p	0.6
K_i	0.1
K_d	0.3

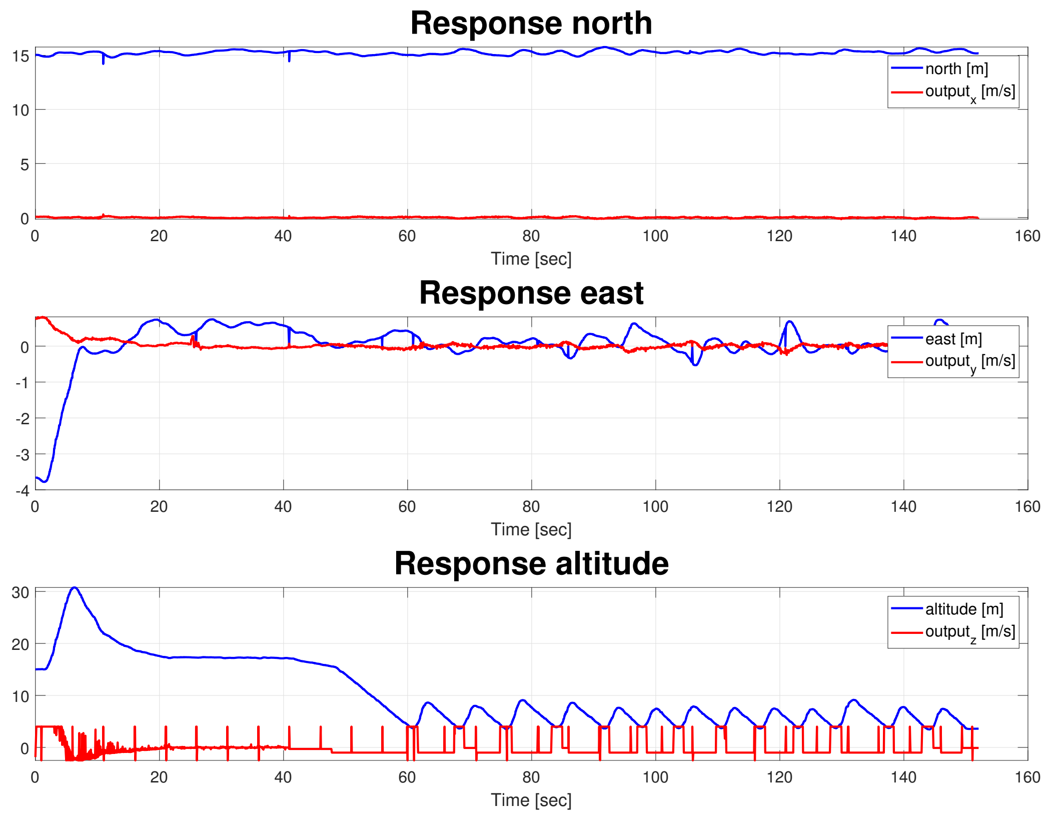


Figure 7.5: Response with parameters from Table 7.1 and Table 7.2

The occurrence of the altitude peak after 10 seconds in Figure 7.5 happens because the anti-windup was switched off during the test. This time, the controller was able to hold the hover height, but the problems started again after entering the 4 meter height. The controller was not able to switch from constant velocity output to position control. Saturation of the altitude controller outputs are also present, indicating that tuning parameters from Table 7.2 are still too aggressive. On the other hand, horizontal responses shown in Figure 7.5 are satisfying.

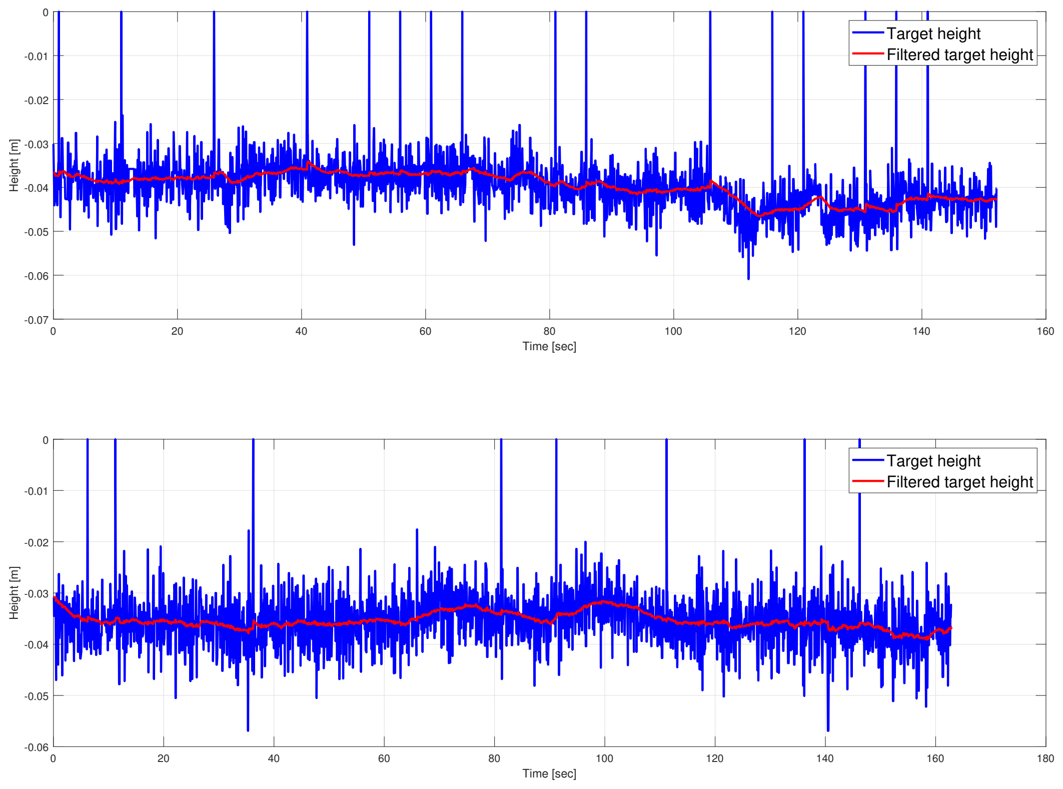


Figure 7.6: Filter performance in test 1

Filtered target height values from tests shown in Figure 7.3 and Figure 7.5 are showed in Figure 7.6. Although the GNSS antenna of the landing platform was standing still during the simulations, some signal noise was produced. The noise amplitude is not affecting the position measurements, as the noise oscillations are no more than 3 cm. Still, the noise is a good test for the low-pass filter, implemented as described in Section 6.4. Figure 7.6 shows that the low-pass filter is able to filter out oscillations, giving a smooth reference input to the altitude controller.

7.3.2 Test 2

The second test was held a few days after the first test. The tuning parameters were decreased drastically before the second test, since the results of the first test indicated too aggressive parameters. During the first test, a set of horizontal control parameters were tuned. The same set of parameters is used for the second test (Table 7.1). The altitude control parameters, listed in Table 7.3, are changed and tuned to give a slow response, because of the experience from the first test. Also, the anti-windup and the derivative part of the altitude controller is removed to make the controller as simple as possible.

Table 7.3: Final altitude controller parameters

Parameter	Value
K_p	0.4
K_i	0.01
K_d	0

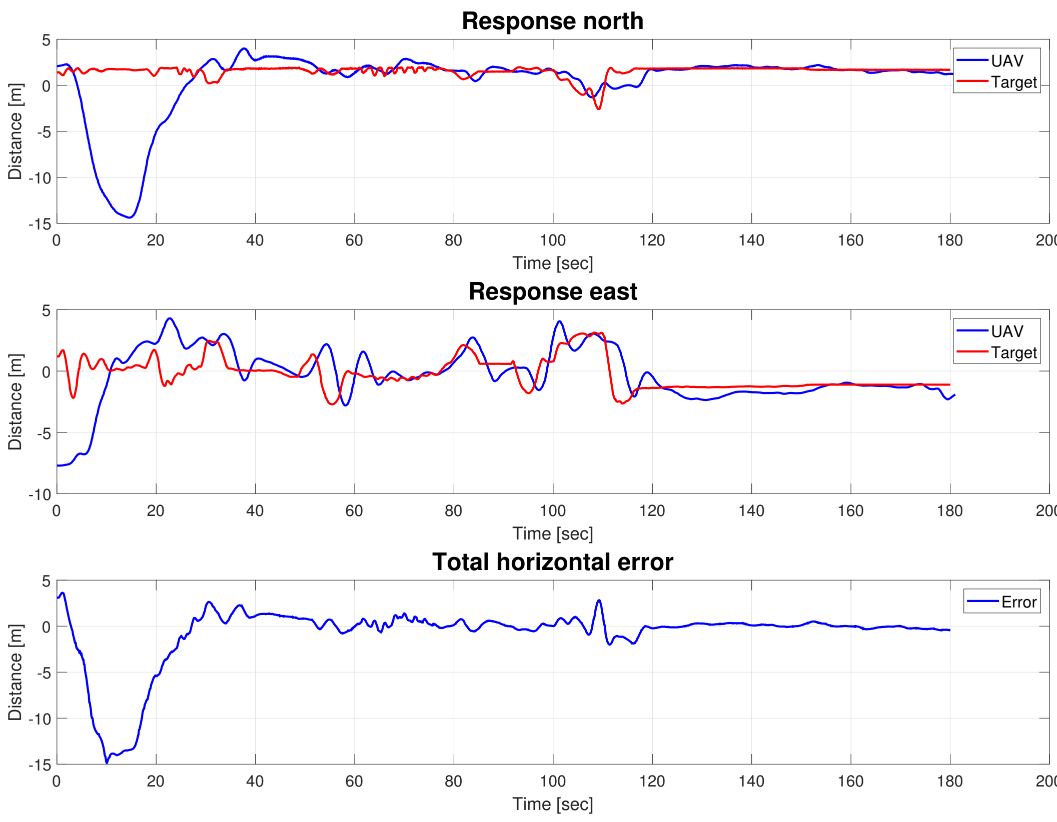


Figure 7.7: Horizontal response in test 2

Figure 7.7 presents the horizontal response from the second test. The fishing rod, representing the target platform (Figure 7.2), was moved through the whole test, in order to induce an error between the target position and the UAV position. It is clear that the horizontal response is slow, as the UAV reaches the target position several seconds after the target itself gets there. Although the horizontal response is slow, the total horizontal error rarely exceeds 3m, which is the minimal landing platform size (Section 6.3.2). The reason for the first 15m large error is that the set-point was changed -15 meters at the beginning of the test, and then changed back to 0m, to test the control systems ability to track larger distances.

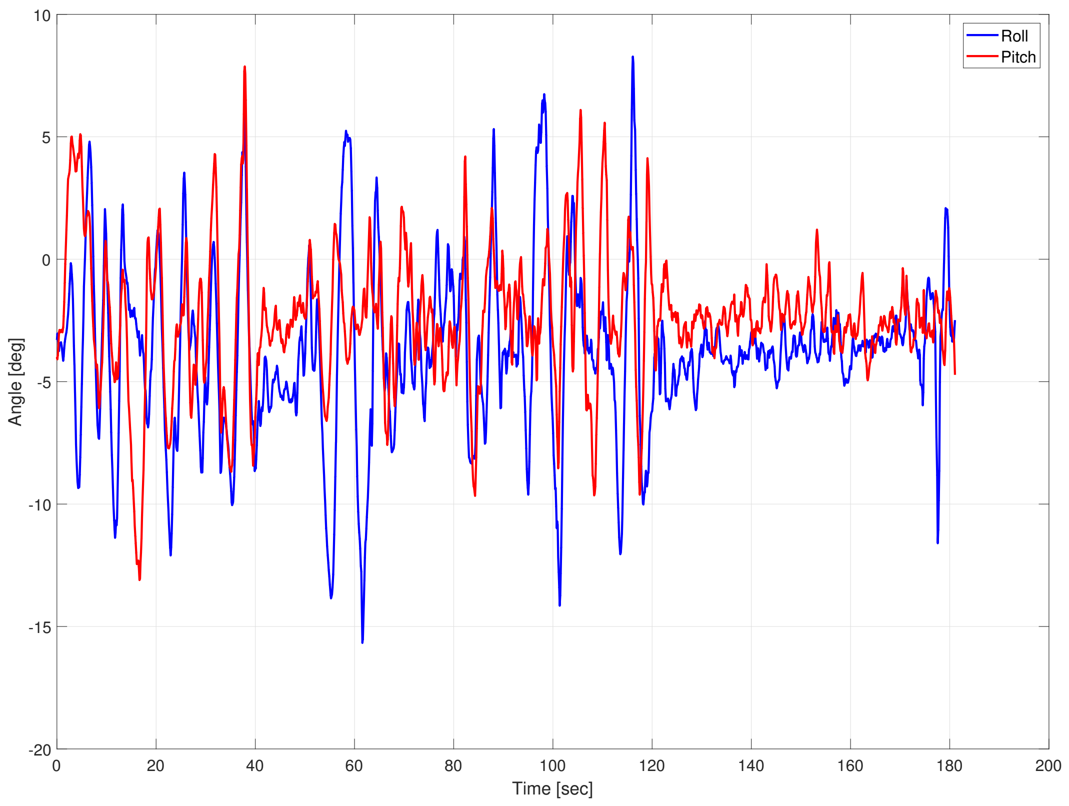


Figure 7.8: Roll and pitch response in test 2

The roll and pitch response from the horizontal response in Figure 7.7 is presented in Figure 7.8. The angle values never exceeds $\pm 15°$, because of the slowly tuned control parameters form Table 7.1. The roll and pitch values are not close to saturation, which is $\pm 45°$[20], meaning that the controller parameters can be tuned more aggressively. Greater roll and pitch response leads to a faster horizontal response, leading to the possible removal of the response delay from Figure 7.7. Still, one has to be aware of the fact that greater roll and pitch response can lead to an overshoot and unstable system.

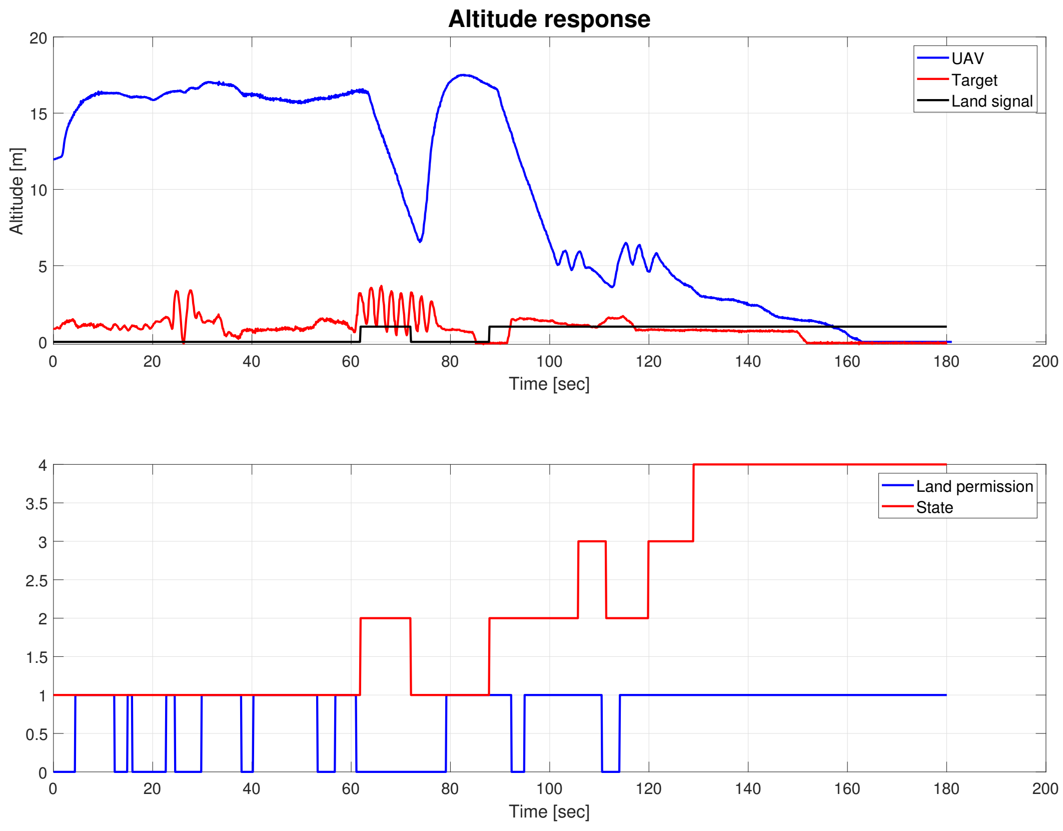


Figure 7.9: Altitude response in test 2

Interaction between the land permission algorithm (Section 5.4), the state machine (Section 5.4) and the altitude response is tested and presented in Figure 7.9. As the first subplot of the figure shows, the landing command is given after 60 seconds, and that aborted after 70 seconds, to checks the vehicle's ability to respond to lading abortion by the user. The oscillations in the target signal are made to test the filter and lading permission algorithm, which are presented later in the report. It is noticeable that the altitude response overshoots after the aborted landing. In addition, it takes almost ten seconds to get from the overshoot peak to the hover height although the peak height is only a few meters above the hover height. The explanation of the slow response lays in the tuning parameters from Table 7.3. For example, $K_p = 0.01$ minimizes the integral contribution, leading to slow diverging to the reference altitude. The overshoot itself could possibly be removed implementing anti-windup form Section 4.2.1, but the results from test 1 tell that the control system should be as simple as possible.

Oscillations in the altitude response occur after 100 seconds, and after 110 seconds in the first subplot. The reason for the first oscillations is that the controller changes from constant descent velocity to PID control, as the state machine, described in Section 5.4, changes from State 2 to State 3. Oscillations are significantly smaller than in the first test (Figure 7.3). Again, the reason for the undesired oscillations lies in bad control parameter tuning. The second set of oscillations, after 110 seconds, is caused by an upward target movement. What happens in the first oscillation is that the controller actually overshoots, trying to follow the upward movement of the target. In addition, the state machine changes from State 2 to State 3, leading to one additional oscillation after 120 seconds.

The interaction itself works as desired. Although there occur few undesired oscillations, the controller actually responds on state changes in the state machine. After 110 seconds in Figure 7.9, the state machine changes from State 3 to State 2 because of the horizontal error from Figure 7.7 exceeds allowed boundaries of the cylinder from Section 5.4.1.

An error regarding the State Machine was experienced during the test. Figure 7.10 shows the response of a test with the horizontal error value manually set to be 4. Such horizontal error value is violating boundaries from Section 5.4.1, meaning that the state machine should not enter State 3. Analyzing Figure 7.10, one can see that some standing oscillations occur in the altitude response. The reason for the oscillations is that the state machine is stuck between state 2 and 1. As the total horizontal error constantly exceeds boundaries, the state machine counts three seconds outside boundaries, and sends the vehicle back to State 1. State 1 sends the vehicle back to State 2, since the landing flag is still turned on by the user. A solution to the problem is discussed later in the report.

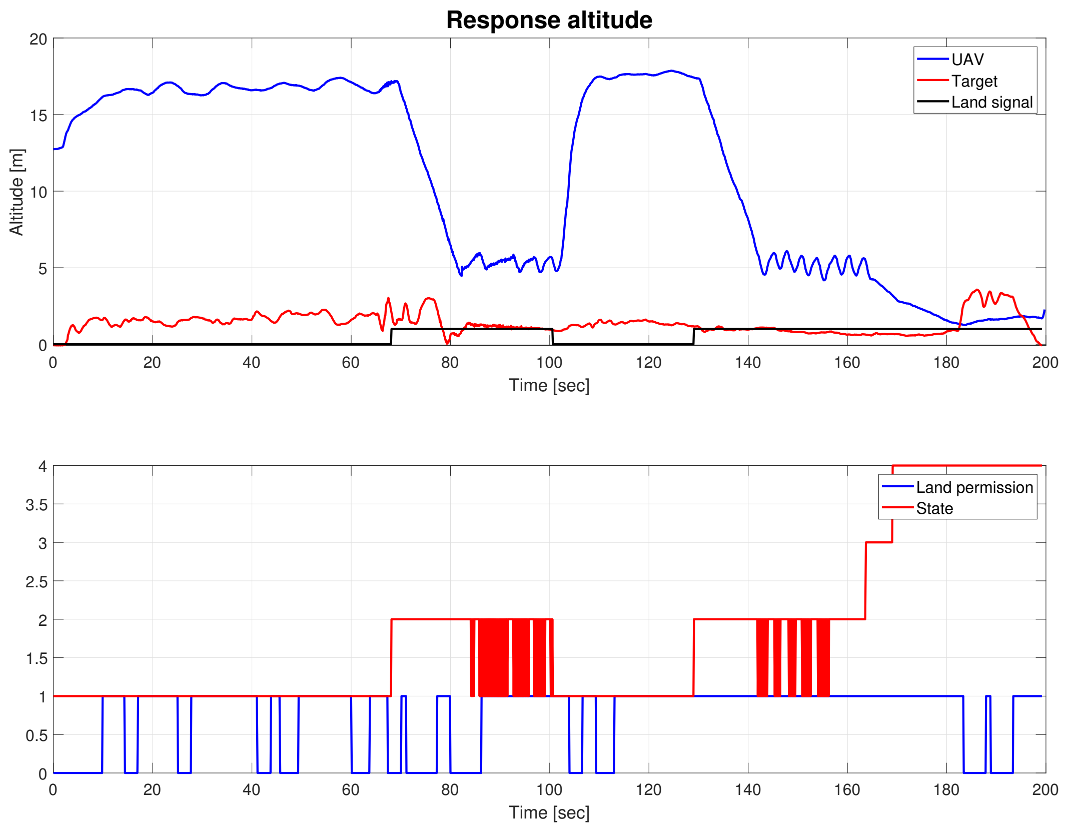


Figure 7.10: State machine problem in test 2

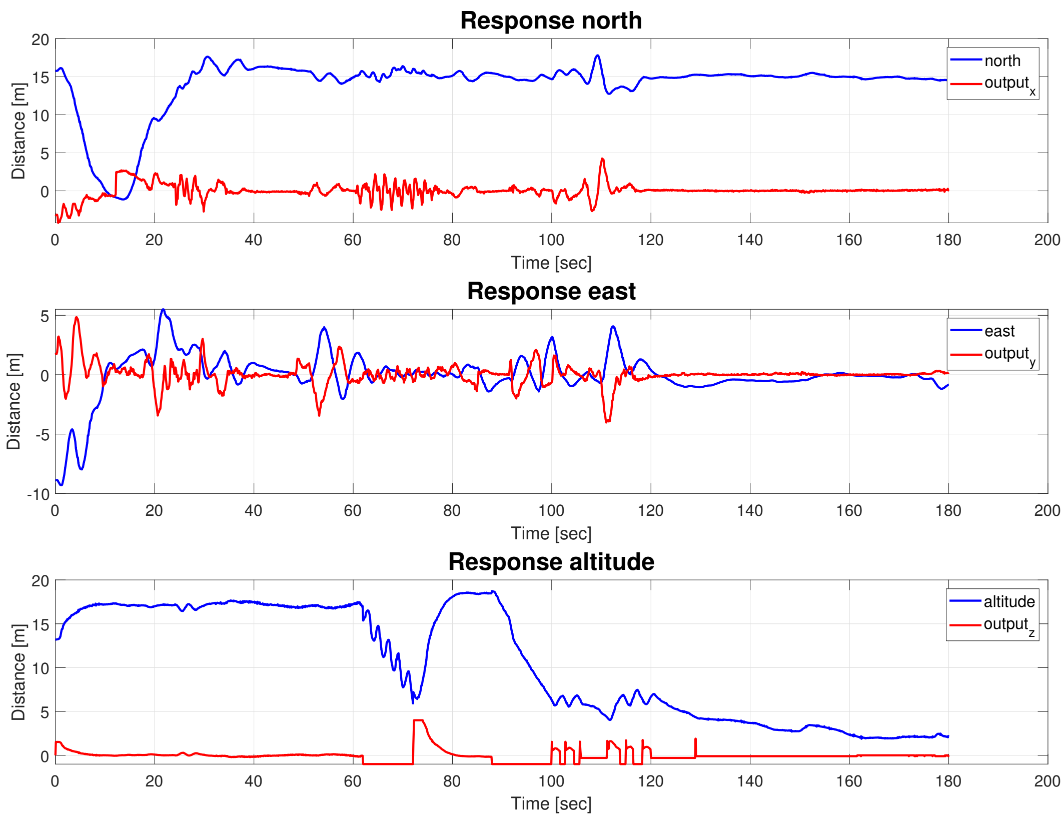


Figure 7.11: Control output in test 2

Figure 7.11 presents the control output form the responses presented in Figure 7.7 and Figure 7.10. The figure confirms the fact that the control output values are low for all controllers. Comparing with control output values from test 1 (Figure 7.3), it is clear that the control outputs in Figure 7.11 can be gained more, meaning that tuning variables from Table 7.1 and Table 7.3 could be increased. The tuning variable increase is discussed later in the report.

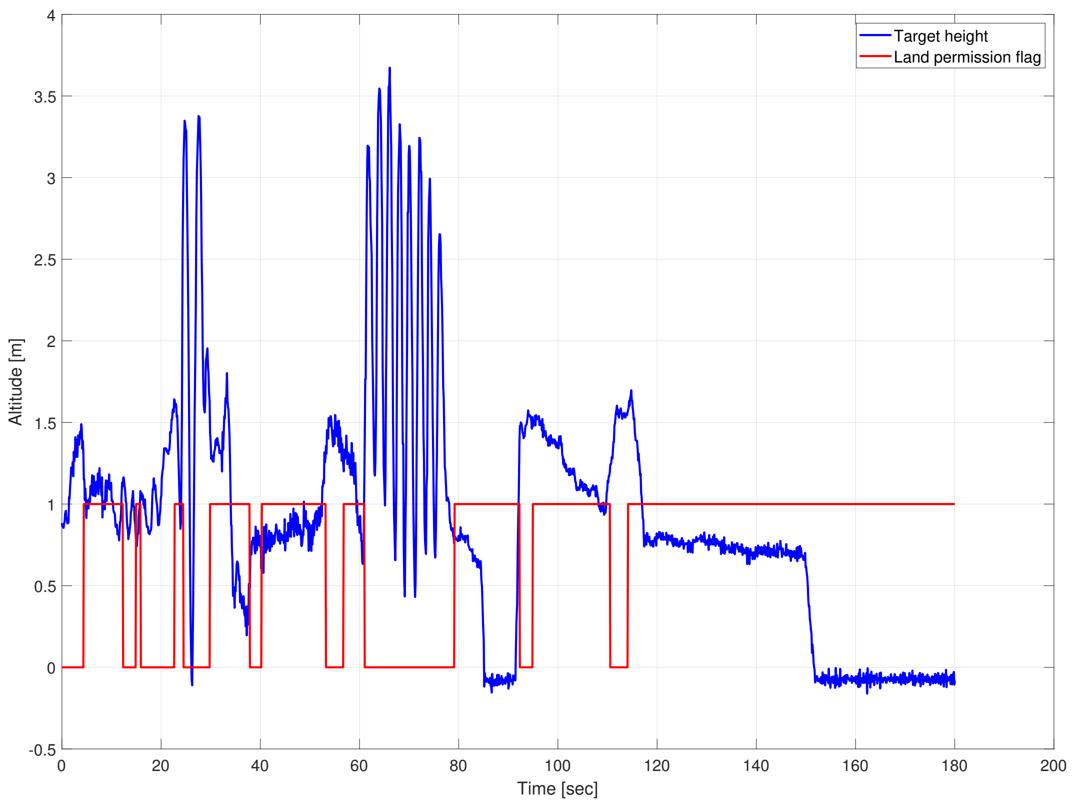


Figure 7.12: Landing permission algorithm performance in test 2

The land permission algorithm performance is presented in Figure 7.12. The algorithm response is satisfying. One of the main points with the algorithm is to deny landing, where the danger is obvious. An example of such a case can be observed between 60 and 80 seconds.

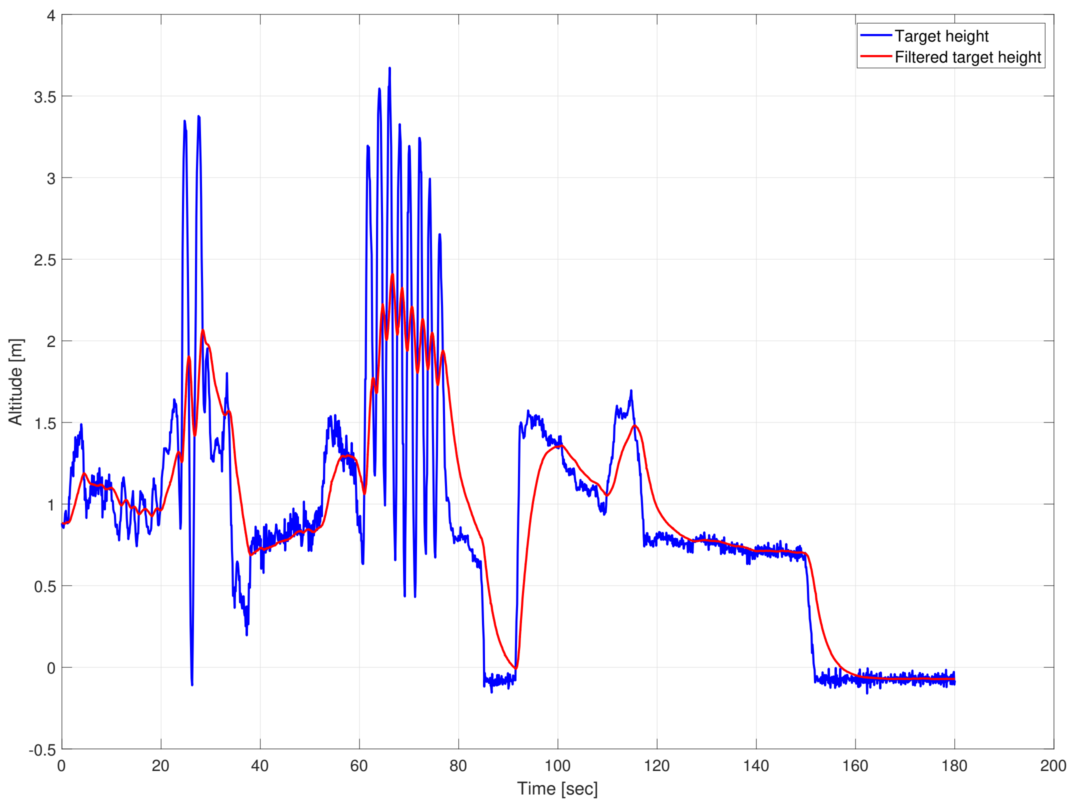


Figure 7.13: Filter performance in test 2

Figure 7.13 shows the performance of the filter algorithm during test 2. The filter is tested for wave movement with large altitude, compared to Figure 7.6, where only signal noise filtering is tested. The performance is slow, as expected, because of the time constant (Section 6.4). The amplitude of high-frequency oscillations are damped, as well as the signal noise.

Chapter 8

Discussion

This section discusses results presented in Chapter 6 and Chapter 7.

8.1 Control

Both horizontal and vertical controllers implemented in this project are PID controllers (Figure 4.1). The horizontal controllers use velocity feedback on the derivative part, while the altitude controller calculates the velocity by taking the derivative of the position. The main reason for using the velocity feedback is to get more precise reference input to the derivative part of the PID controller. Horizontal PIV control was used in the specialization project. The output of the PIV controller was desired acceleration, and seemed to work in the simulations. The reason why PIV control was changed to PID control is that the original ArduPilot low-level controller does not support force input. There exist a modified version of the ArduPilot flight stack, developed at the UAVlab at NTNU, supporting force input, but is not recommended using because of the difficulties of keeping the modified codes up-to-date with the original ArduPilot project. An example of the use of the modified ArduPilot version, supporting the force input, is written about in [37].

Figure 6.16 shows simulation results of a landing process. The results are showing a vehicle able to hold both the horizontal and the vertical position. Some error peaks occur when the target changes the direction of the forward velocity. The peaks are expected looking at the tuning response from Figure 6.5 and Figure 6.6. Both figures show an overshoot, indicating the error peaks on steps in control reference values. The figures also show that the controller is able to recover the error peaks fast. The fast error recovery gives an illusion of very aggressive control parameters.

Figure 7.3 shows the response of the same control parameters as used in Figure 6.16 for a field test. It is clear that parameters that worked for the simulation, did not work for the field test. The reason is that the simulation model does not fit the real-world model perfect. Also, a more correct approach would be to start with slow control parameters and find faster parameters by tuning in the field. To show the model difference, a landing is simulated with parameters from Table 7.1 and Table 7.3, which gave a satisfying response during the field test, presented in Figure 7.7 and Figure 7.9. The simulation results of the parameters are presented in Figure 8.1.

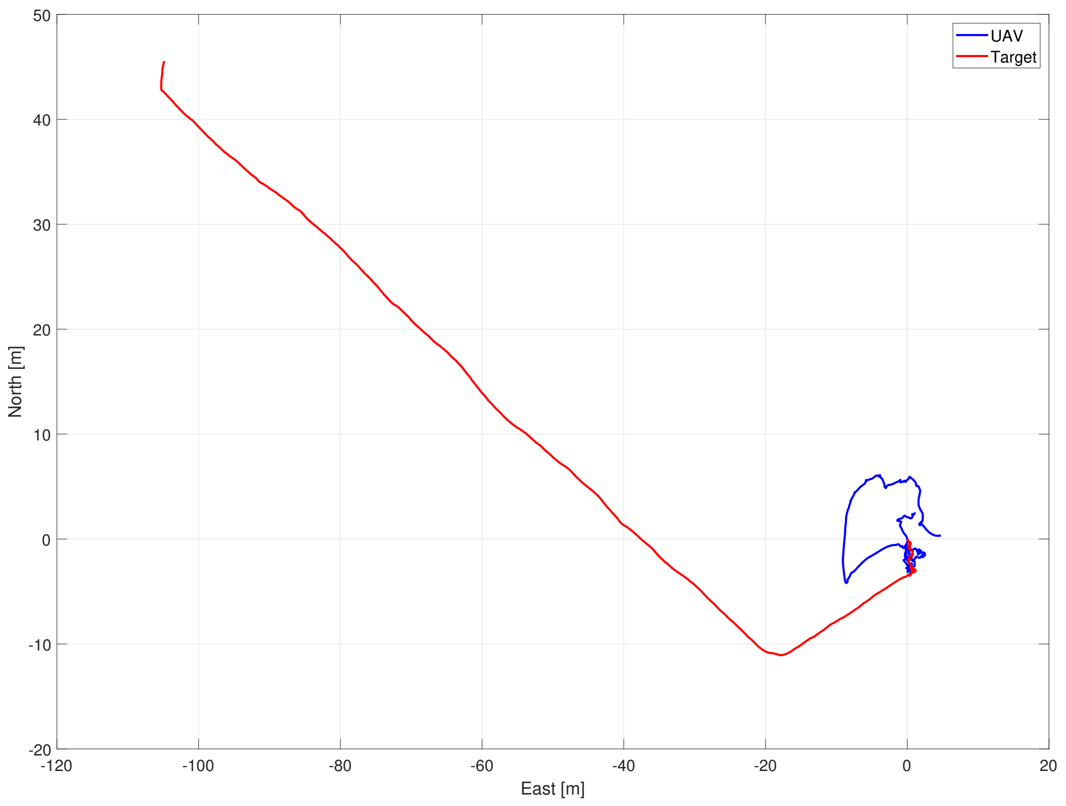


Figure 8.1: Horizontal plane of a simulation using parameters that worked for the field test

From Figure 8.1 it is clear that the UAV is not able to follow target moving in the path presented in Figure 6.12. The controller gains are so low, that the controllers do not produce enough output to move the vehicle fast enough to follow the target movement. The figure confirms the fact that simulation model does not fit the real-world model.

The possibility of increased tuning parameters giving better field test results is mentioned many times in Section 7.3.2. Figure 8.2 shows the x-direction response for a field test with increased tuning parameters. The K_p parameter is increased from 0.2 to 0.4 after 200 seconds and from 0.4 to 0.6 after 270 seconds. The increased values did not give a better and faster response as expected from Section 7.3.2, but the opposite. The controller performance got worse with increased values.

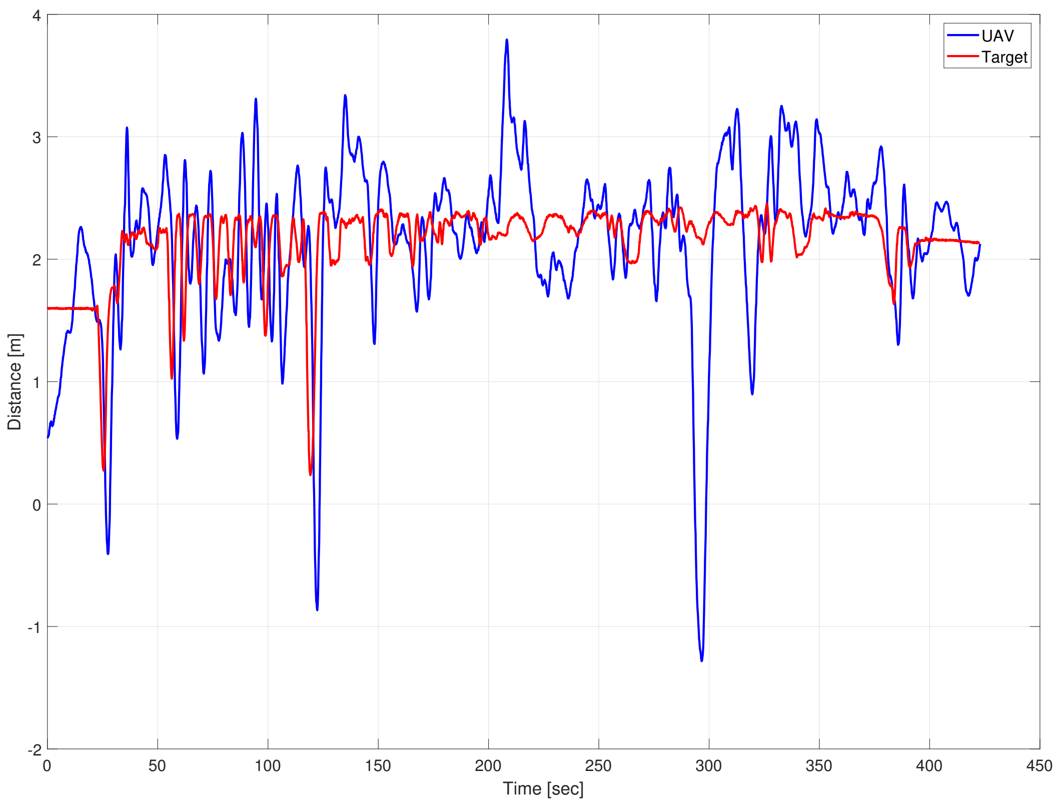


Figure 8.2: X-controller tuning during the test 2

The problem may lay in the simplicity of the PID controller. One solution could be trying another control algorithm to get a better control response. LQR is one of the control algorithms that could be tried in the future. The argument for the PID algorithm is that it does not require a perfect model, which definitely is a benefit when it is suspected that the simulation model does not fit the real world model perfect. Therefore, one has to have in mind that the use of other algorithms may lead to improvements in the simulation model.

8.2 Land Permission

Figure 5.4 and 7.12 shows the performance of the land permission algorithm developed in this project (Section 5.4.2). It can be seen that the algorithm results are partly successful. The problem lies in the parts where the algorithm gives landing permission, and the waves are moving fast towards the vehicle. The reason why that happens is that the landing algorithm needs one second from the upwards wave movement starts till it the algorithm denies a landing. On the other hand, it takes three seconds from the downwards movement start until the permission signal is given.

The problem of the landing algorithm needing one second from the upwards movement starts till the permission signal is given can be solved by taking more measurements. More measurements can possibly lead to instability of the algorithm, using the existing design from Section

5.4. The allowed upwards velocity of the platform is 0.2 m/s. There is no scientific research backing up the chosen velocity, just a guess and use of sense. The threshold target velocity should be investigated deeper, and changed.

One of the suggestions from the specialization project is the development of a more successful landing permission algorithm. Results from Figure 5.4 and 7.12 shows that improvements are made, compared to the specialization project, but there is still a lot of room for further improvement.

8.3 Filter

The low-pass filter, designed as described in Section 6.4 is able to filter out high-frequency signals, as desired. Filter performance is presented in Figure 6.10, Figure 7.6 and Figure 7.13. All figures show the filter's ability to filter out high frequencies. Still, there exists a time delay between the input signal and the output signal of the filter. The time delay value is measured to be $\tau \approx 1.2sec$. Since the purpose of the filter is to filter out small wave movements from the altitude controller's reference signal at hover altitude, the time delay is considered as negligible.

The advantage of using a first-order low-pass filter is that the implementation of the filter is easy. The filter also does not require much computational power, as only two equations have to be computed (Section 5.2). A huge disadvantage of the filter is, of course, the time delay. Although the delay can be neglected at hover altitude, the matter is different at lower altitudes. The time delay makes the filter use dangerous at lower altitudes, as waves can reach several meters on 1.2 seconds. A solution is of course implementation of a filter without such large time delay. An example of such filter is described in [38], using the Kalman filter design.

Another fact that has to be considered is that the filter cutoff frequency is found by looking at the frequency spectra of pure waves (Figure 4.3). Normally, the heave motion of a ship is damped compared to wave motions [39]. The bigger the ship is, the more will the wave oscillations be damped out. Therefore, frequencies of a real ship will be different than presented in Figure 4.3. The high-frequency oscillations, that the filter is damping out in the simulations, will be damped by the ship itself.

8.4 State Machine

The state machine, described in Section 5.4, gave satisfying results in both simulations and field tests, except one bug. State machine performance is presented in Figure 5.5 and Figure 7.9. Figure 7.10 shows an altitude response during a field test, where the state machine bug is visible. The bug occurs when the horizontal error has a constant value greater than 3m. In that specific situation, the state machine gets stuck between state 1 and state 2. It happens because the state machine sends the vehicle to state 1, when the horizontal error is larger than 3m for over 2 seconds. As long the land flag is given by the user, the state machine tries to send the vehicle back to state 2, once state 1 is entered, creating a loop between two states.

The problem can be fixed in several ways. One option is to always send the vehicle back to a given height when the horizontal error has exceeded 3m for over 2 seconds, and try the landing once again. For example, after exceeding 3m horizontal error over 3 seconds, the vehicle could be sent to 10 m height, and the landing process could start over again from 10m height. A huge drawback with this method is that a landing sequence will last much longer when an error, that is quickly fixed, occurs.

The reason why the problem with the state machine was not fixed during the project is the limited time schedule. The field tests were performed only 2 weeks before the deadline for the thesis delivery, not giving enough time to fix problems discovered during test 2 (Section 7.3.2).

8.5 Platform Size

The minimal platform diameter was found to be 3m, based on simulation results presented in Figure 6.7. Looking at the total horizontal error form field test (Figure 7.7), it can be concluded that 3m is more than enough. Still, one has to have in mind that no significant vehicle movement, like the path in Figure 6.12, was tested in the field. Such movement may cause a larger error, especially on turns, producing the same error peaks that were noticed in simulations (Figure 6.7).

8.6 Reflection

The working process was good in the way that all work in the specialization project was done in Matlab and Simulink, and a great part of the Master's thesis was to implement the work in C++. It felt natural to do the work that way, as it was not enough time during the specialization project to get known with the DUNE system.

Working with DUNE was challenging, but educational. The main problem was the difficulty of finding information, because of the limited use of the system. Fortunately there exist a wiki page, developed by the UAVlab students, with information about the system. The wiki page was used a lot during the project.

The surprise of the real-life model not matching the simulation model could be avoided if the first test was performed earlier. In fact, an early test was scheduled, but had to be canceled because the hexacopter was not ready. The idea at the beginning of the project was to get a ready hexacopter and just focus on software development. Since the working hexacopters were in use by a different project, I and my co-supervisor had to connect all hardware together, install software and deal with the hardware issues. Dealing with hardware issues affected the project, as it took a lot of time away from the software development focus. Still, dealing with hardware was very educational and enriched the project.

Chapter 9

Closing Remarks

9.1 Conclusion

This thesis presents the development of control algorithms, simulation environment in DUNE, landing algorithms and field test system. All developed parts are connected to a large system for the automatic landing of multi-rotor on moving platform.

A simulation environment for the automatic lading is developed and implemented in DUNE. The simulation environment is adequate for landing simulations, as it includes all necessary environmental disturbances. Algorithms for high-level velocity control are developed and tuned using the developed simulation environment. The results are presented. Based on the simulation results, a suggestion of minimal weather requirements needed to carry out a successful landing is presented. The results are also used for the suggestion of the minimal landing platform size.

All hardware necessary for the field test, and connection between those are presented in the thesis. The specific solution developed for the field test is detailed explained. The field test results are compared with the simulation results and the differences are discussed.

Algorithms for error handling during a landing approach are developed. The algorithm performance is tested, both in the field and in the simulations. The results are presented. Some problems related to the algorithms are discovered and discussed. Suggestions for the discovered problems are presented in the thesis.

9.2 Future Work

As slow response and tuning problems were experienced during the project, it is recommended to test other control algorithms ability to carry out a landing task. A recommendation is to test the LQR algorithm. When implementing other algorithms, it is important to be aware of the difference between the simulation model and the real-world model. A suggestion, that unfortunately was not followed in this project because limited time schedule, is to perform early field tests, and get to know differences between the simulation model and the real-life model. Knowing differences early in the process will help in the algorithm design and the controller tuning. It should also be considered investigating the simulation model, and making it more realistic than the existing one.

As mentioned in Section 8.2, the land permission algorithm has to be further developed. First, a test has to be performed to find the threshold of the maximum allowed upward velocity of the platform during a landing. The one second gap between the algorithm giving the landing signal and the sudden upward movement of the landing platform has to be removed as well.

The change of the filter has to be considered, if one wants to use the filter when the vehicle is getting closer to the landing platform. Ideas for implementing a filter without time delay is written about in [38]. Besides, the filter time constant has to be found from real ship motion, and not pure wave motion, as discussed in Section 8.3.

The state machine bug from Figure 7.9 has to be eliminated. Suggestion of how the bug can be eliminated is described in Section 8.4.

A list of suggestions for future work:

- Investigate other control algorithms than PID.
- Consider further development of the existing *hexa* models in DUNE, to get more realistic simulations.
- Perform tests and find the correct threshold for the maximum upward velocity of the landing platform during a landing.
- Fix the state machine bug.
- Find the filter cutoff frequency by using a real ship model.
- Perform tests on the water.

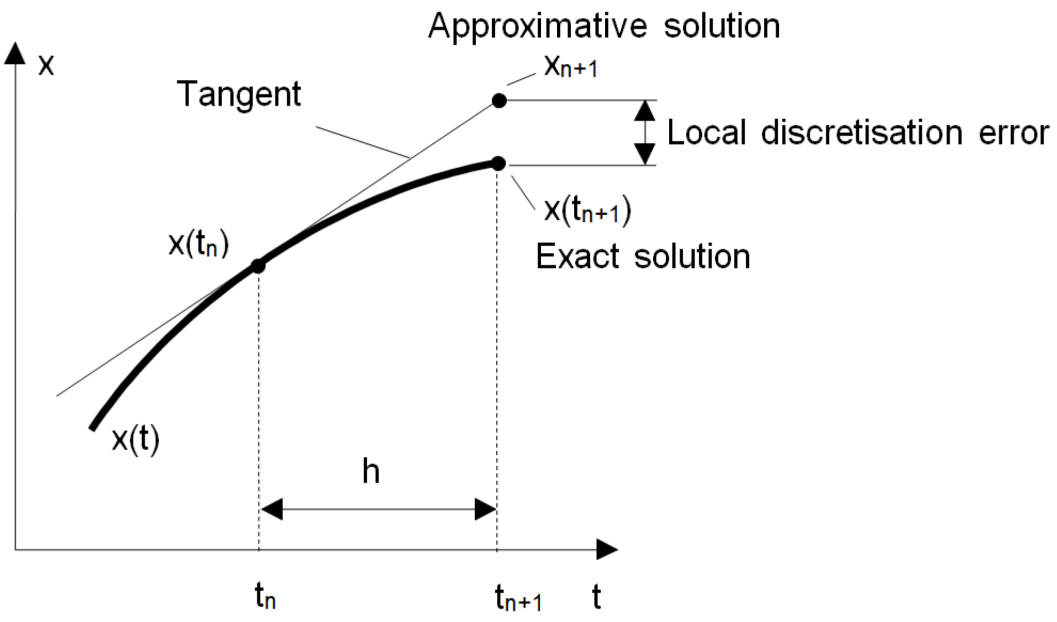
Appendices

A Discretization

All digital machines are discrete, meaning that they are not computing continuously, but given amount of time per second. Working frequency tells how many computations per second a machine is performing. As an example, a machine working on 5Hz is performing 5 computations per second.

Most of the mathematical models are designed for continuous time, meaning that the computation happens continuously. The model has to be discretized, in order to be more realistic. Many discretization methods exist, but some are more used than others. Forward Euler method is chosen in this task, mainly to find an estimate of the derivative value.

The method is finding an estimate of the next value by looking at the tangent line of the current point. The tangent line itself can be used as an derivative estimate.[40]



Forward Euler method

An estimate of the derivative at current point is:

$$\dot{x} \approx \frac{x_{n+1} - x_n}{h}$$

The next example is showing how forward Euler method is used to discretize state-space equations.

A simple state-space equation can look like this:

$$\dot{x} = K$$

Using Euler derivative estimate:

$$\dot{x} \approx \frac{x_{n+1} - x_n}{h} = K$$

Leading to:

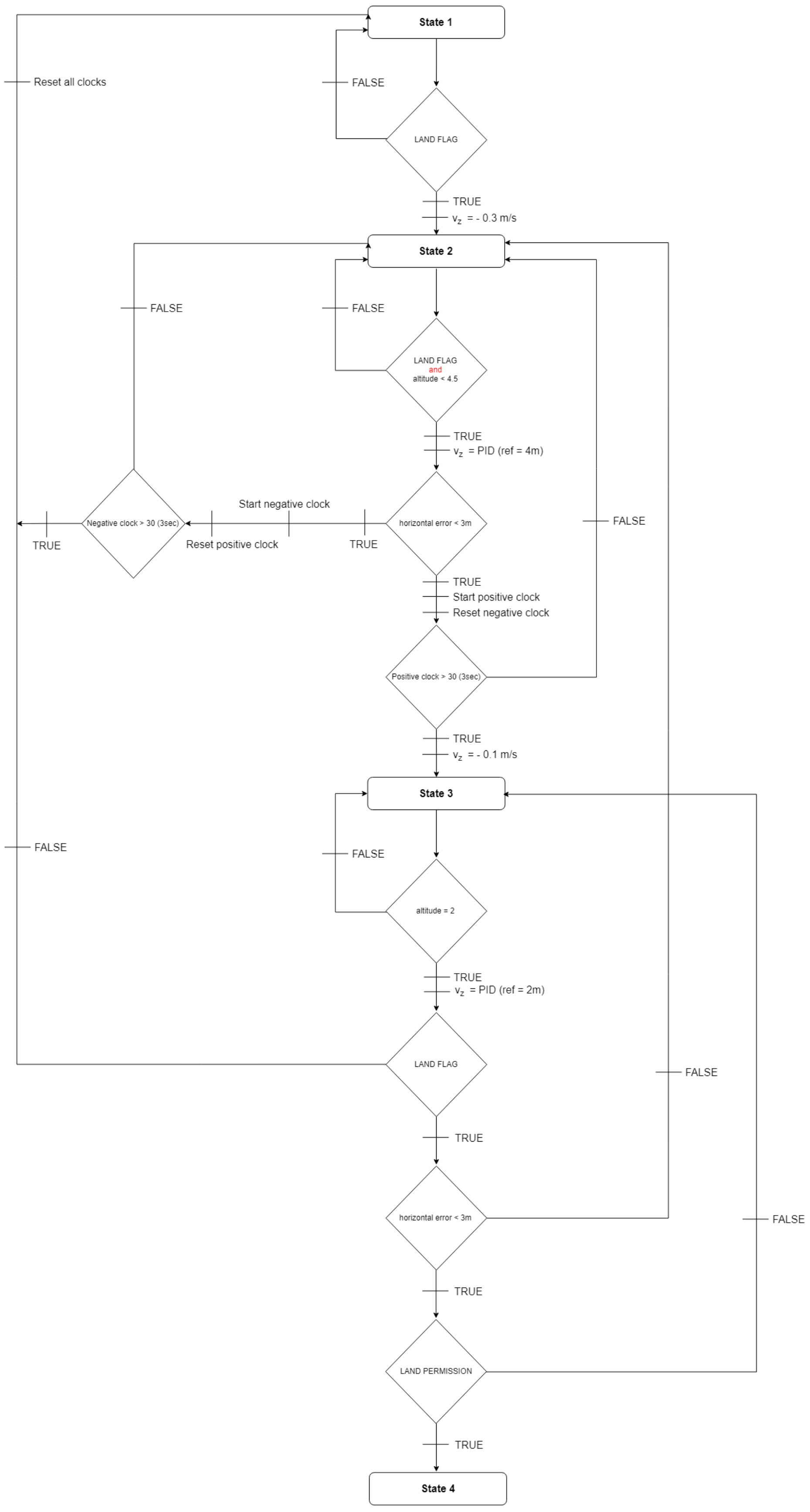
$$\frac{x_{n+1} - x_n}{h} = K$$

The final discretized equation is:

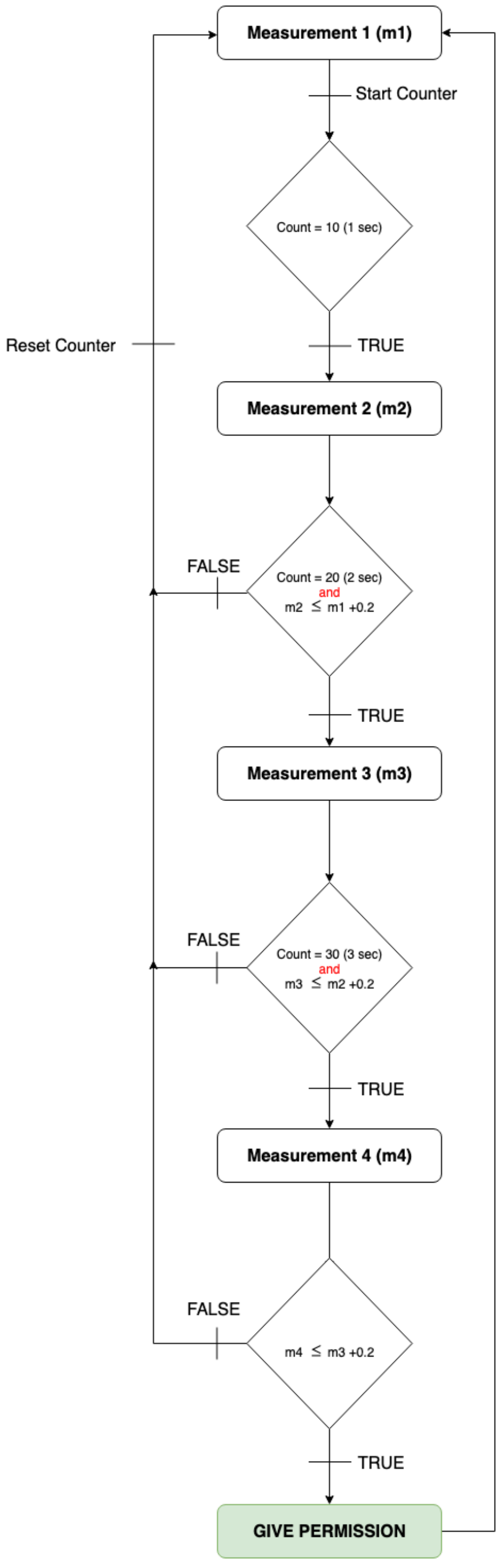
$$x_{n+1} = hK + x_n$$

The showed procedure can be used for discretization of every state space equation.

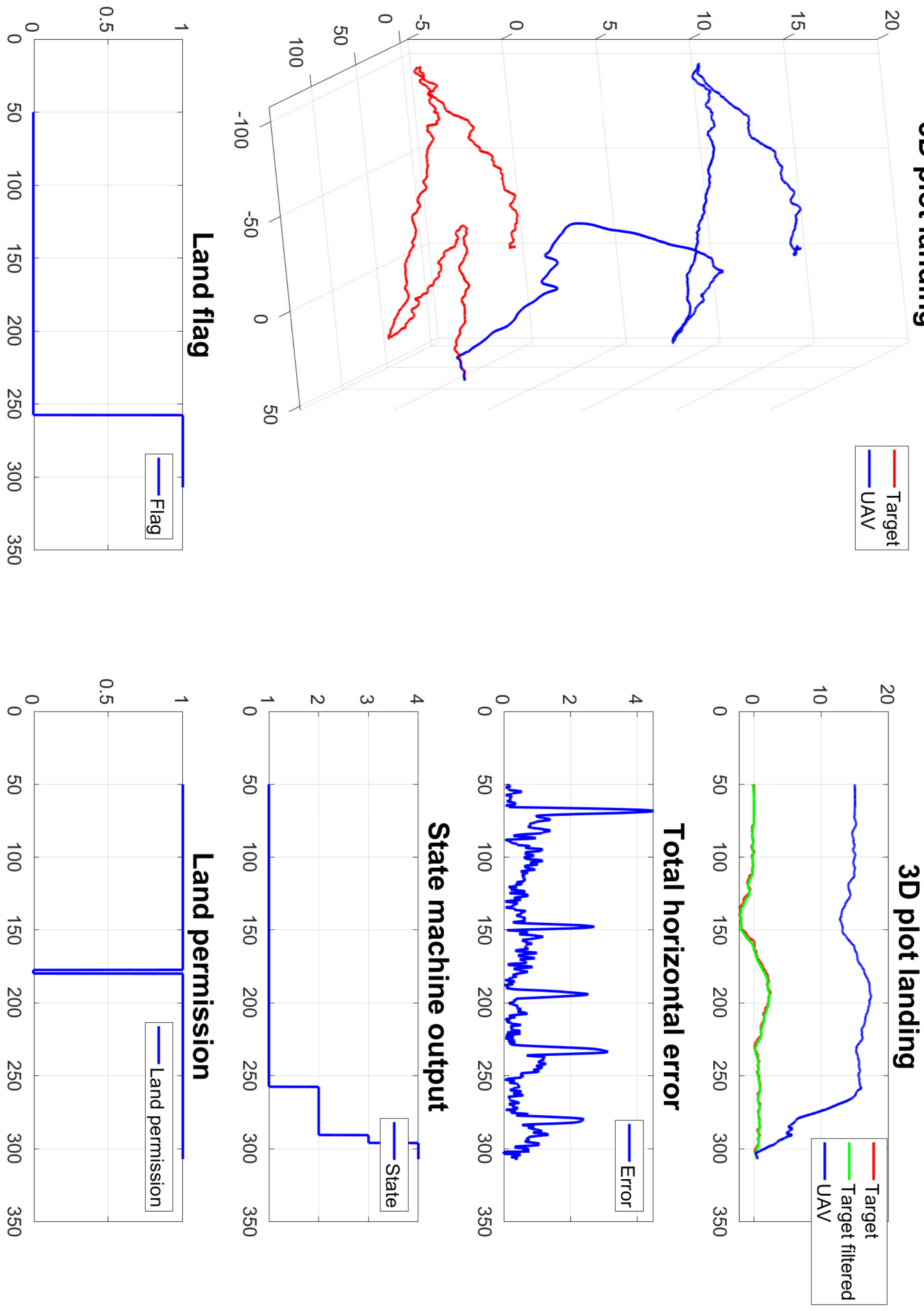
B Flow chart - State Machine



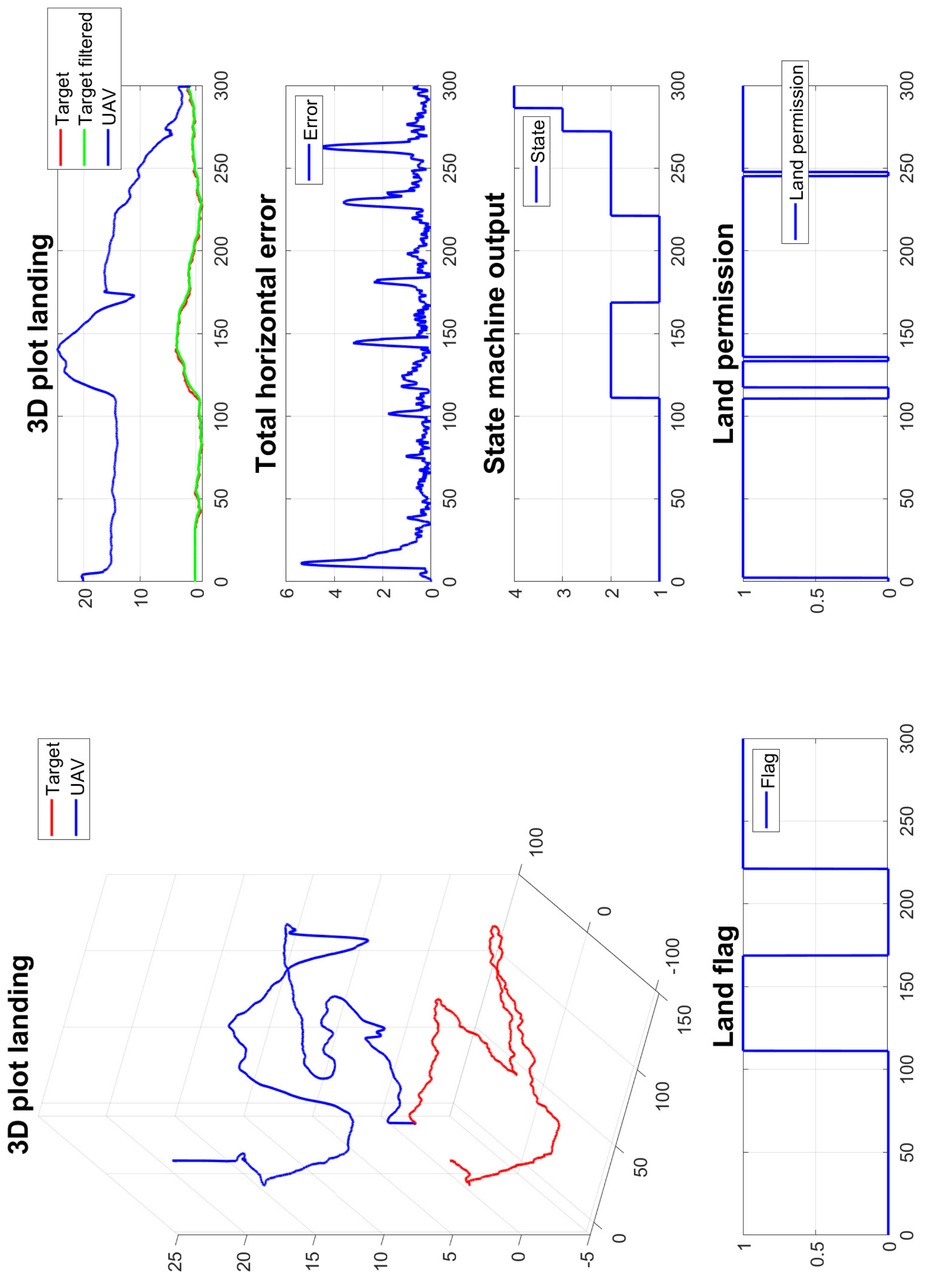
C Flow Chart - Land Permission Algorithm



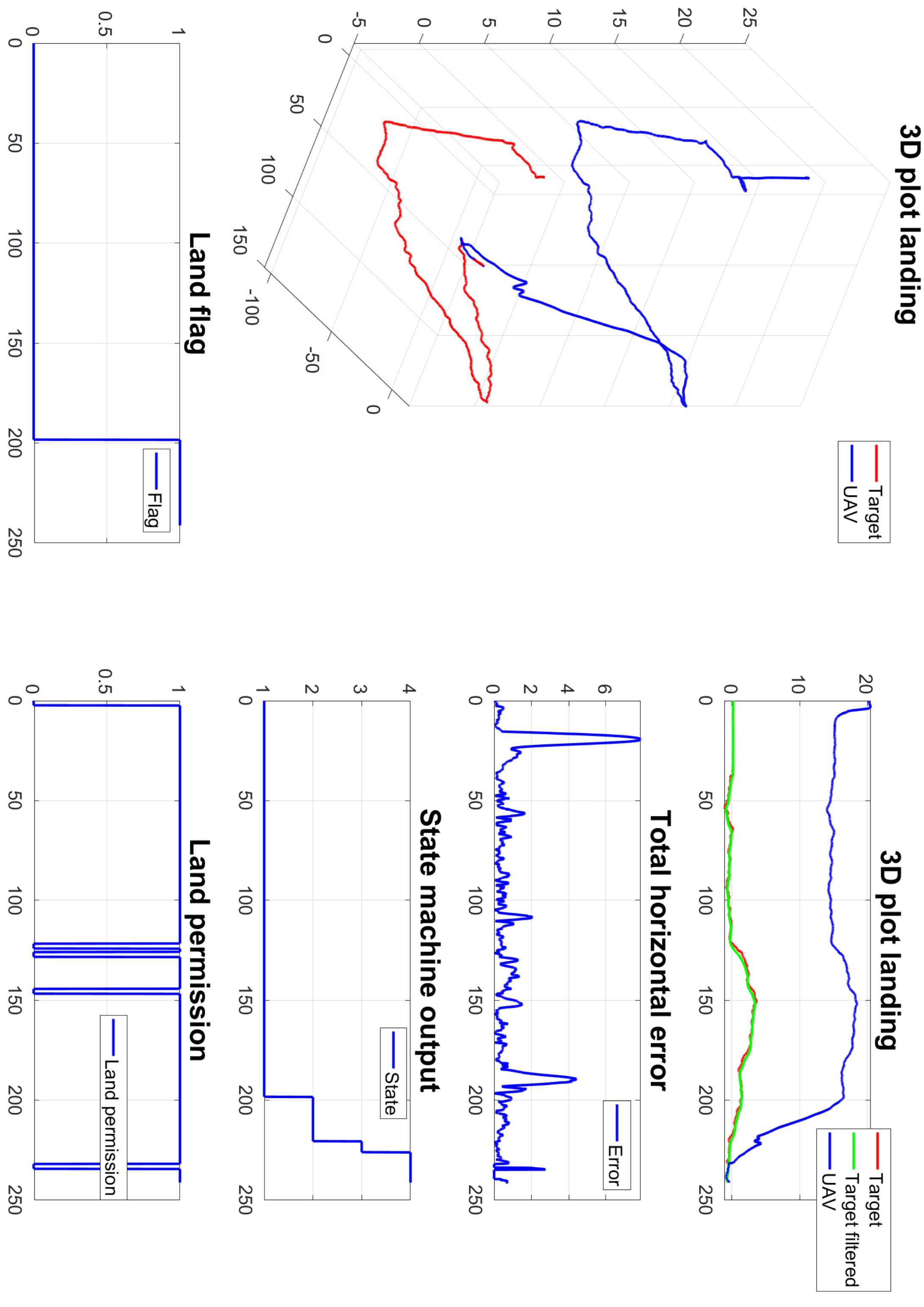
D Simulation Results



Simulation 1 - Full system

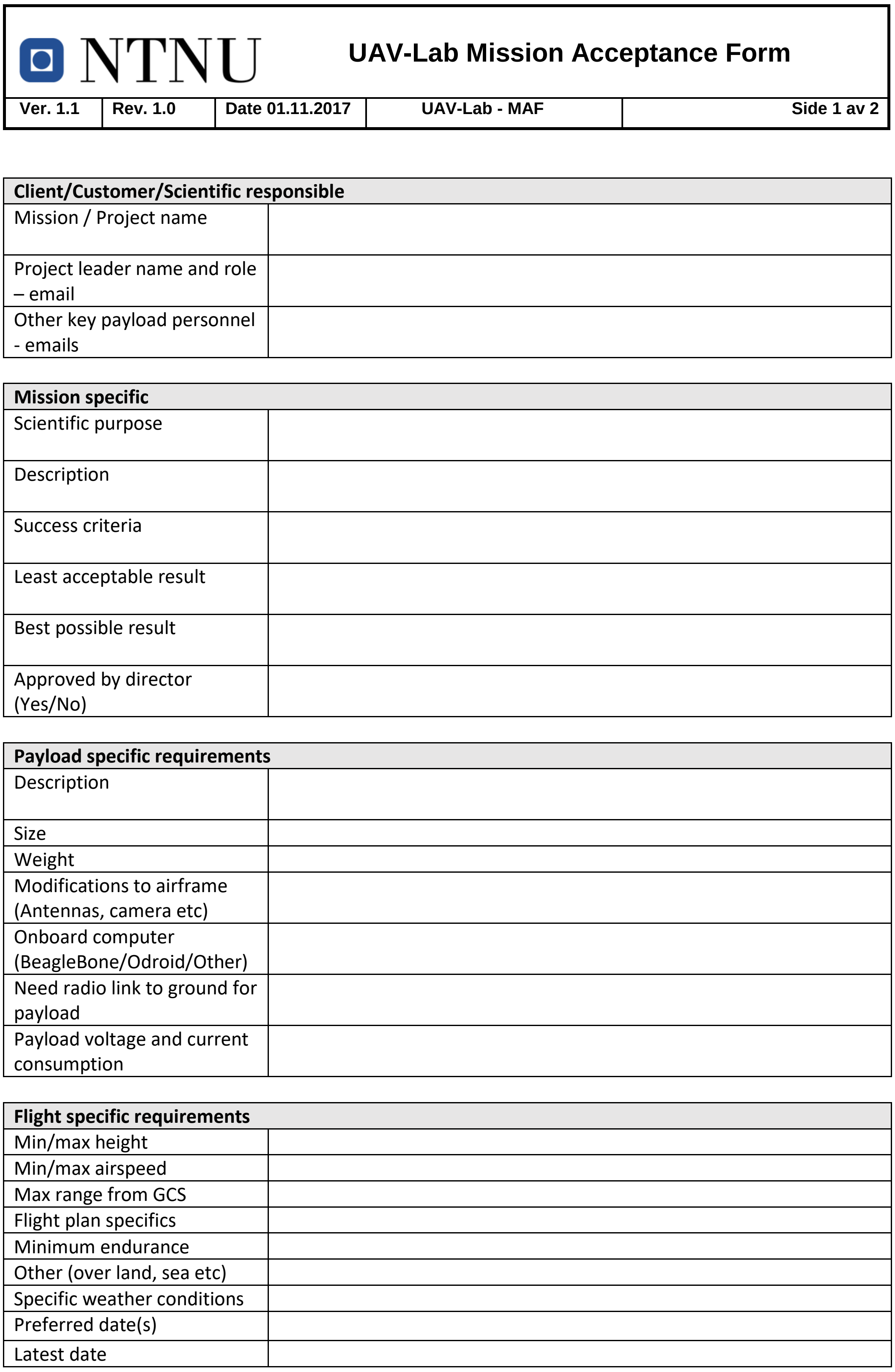


Simulation 2 - Full system



Simulation 3 - Full system

E Mission Acceptance Form



NTNU

UAV-Lab Mission Acceptance Form

Ver. 1.1	**Rev. 1.0**	**Date 01.11.2017**	**UAV-Lab - MAF**	**Side 1 av 2**

Client/Customer/Scientific responsible	
Mission / Project name	
Project leader name and role – email	
Other key payload personnel - emails	

Mission specific	
Scientific purpose	
Description	
Success criteria	
Least acceptable result	
Best possible result	
Approved by director (Yes/No)	

Payload specific requirements	
Description	
Size	
Weight	
Modifications to airframe (Antennas, camera etc)	
Onboard computer (BeagleBone/Odroid/Other)	
Need radio link to ground for payload	
Payload voltage and current consumption	

Flight specific requirements	
Min/max height	
Min/max airspeed	
Max range from GCS	
Flight plan specifics	
Minimum endurance	
Other (over land, sea etc)	
Specific weather conditions	
Preferred date(s)	
Latest date	

For operations and technical manager	
Platform	
Vehicle ID	
Registration number	
Mission date(s)	
Payload ready at lab date	
Payload ready, mounted and tested in UAV date	
UAV and avionics ready date	
Flight plans ready date	

Mission specific – operations manager	
Airfield / Area	
Landowners permission needed	
ATC / CAA permission needed	
NOTAM needed	
Remarks	

Mission specific risk analysis **– mission specific risks not covered elsewhere in POH or operations manual**	

Approvals	**Date**	**Signature**
Customer / project leader		
Operations Manager		
UAV ready (Technical Manager)		
Specific Risk Analysis Done (Director / Operations Manager)		
NOTAM / ATC approved		
Landowners permission		

Bibliography

[1] Vuk Krivokapic. Automatic landing of multi-rotor on moving platform - specialization project. Master's thesis, NTNU, 2018.

[2] Marcus Frølich. Automatic ship landing system for fixed-wing uav. Master's thesis, NTNU, 2015.

[3] Artur Zolich. Systems integration and communication in autonomous unmanned vehicles in marine environments. 2018.

[4] Håvard Lægreid Andersen. Path planning for search and rescue mission using multicopters. Master's thesis, Institutt for teknisk kybernetikk, 2014.

[5] Rubén Vega Astorga. Simulation of a quadrotor unmanned aerial vehicle. B.S. thesis, 2016.

[6] Robin Hofset Vattøy, Inge Nilsen, and Eirik Strøm Fagerhaug. Drone: Dynamic positioning in relation to a given object. B.S. thesis, 2016.

[7] Martin Lysvand Sollie. Estimation of uav position, velocity and attitude using tightly coupled integration of imu and a dual gnss receiver setup. Master's thesis, NTNU, 2018.

[8] Bjørn Amstrup Spockeli. Integration of rtk gps and imu for accurate uav positioning-integrasjon av rtk gps og imu for nøyaktig uav-posisjonering. Master's thesis, NTNU, 2015.

[9] Vegard Line. Autonomous landing of a multirotor uav on a platform in motion. Master's thesis, NTNU, 2018.

[10] Joel Hermansson, Andreas Gising, Martin Skoglund, and Thomas Schön. *Autonomous landing of an unmanned aerial vehicle*. Linköping University Electronic Press, 2010.

[11] Yi Feng, Cong Zhang, Stanley Baek, Samir Rawashdeh, and Alireza Mohammadi. Autonomous landing of a uav on a moving platform using model predictive control. *Drones*, 2(4):34, 2018.

[12] FL Pereira, J Pinto, JB Sousa, RMF Gomes, GM Goncalves, and PS Dias. Mission planning and specification in the neptus framework. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3220–3225. IEEE, 2006.

[13] Dune. http://eigen.tuxfamily.org/index.php?title=Main_Page.

[14] Neptus. https://lsts.fe.up.pt/toolchain/neptus.

[15] Sigurd Andreas Holsen. Dune: Unified navigation environment for the remus 100 auv. *Norwegian University of Science and Technology*, 2015.

[16] Ardupilot. http://ardupilot.org/ardupilot/index.html.

[17] Randal W Beard and Timothy W McLain. *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[18] Martin Lysvand Sollie. Estimation of uav position, velocity and attitude using tightly coupled integration of imu and a dual gnss receiver setup. Master's thesis, NTNU, 2018.

[19] Rtklib. https://wiki.openstreetmap.org/wiki/RTKLIB.

[20] Thomas Thoresen, Jonas Moen, Sondre A Engebråten, Lars B Kristiansen, Jørgen H Nordmoen, Håkon K Olafsen, Håvard Gullbekk, IT Hoelster, and Lorns H Bakstad. Distribuerte cots uas for pdoa wifi geolokalisering med android smarttelefoner. Technical report, Technical report, Forsvarets forskningsinstitutt, FFI-rapport 14/00958, 2014.

[21] Pixhawk. https://www.suasnews.com/2013/08/px4-and-3d-robotics-present-pixhawk-an-advanced-user-friendly-autopil

[22] Beaglebone. https://beagleboard.org/black.

[23] Circuit studio. https://www.altium.com/circuitstudio/.

[24] User manual gnss antenna. https://www.orbitica.com/harxon/pdf-harxon/HX-CH3602A.pdf.

[25] Rocket. https://www.ui.com/airmax/rocketm/.

[26] Mavlink. https://mavlink.io/en/.

[27] Pm spectra. https://www.dune-project.org/.

[28] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[29] Drag coefficient. https://www.engineeringtoolbox.com/drag-coefficient-d_627.html.

[30] Mostafa Moussid, Adil Sayouti, and Hicham Medromi. Dynamic modeling and control of a hexarotor using linear and nonlinear methods. *International Journal of Applied Information Systems*, 9(5), 2015.

[31] Mss toolbox. http://www.marinecontrol.org/.

[32] Karl Johan Åström and Tore Hägglund. *Automatic tuning of PID controllers*. Instrument Society of America (ISA), 1988.

[33] Uavlab wiki. http://uavlab.itk.ntnu.no/wiki/doku.php?id=guides:start_guide.

[34] Eigen library. http://eigen.tuxfamily.org/index.php?title=Main_Page.

[35] Beufort scale. http://www.tranoy.net/stavanger/weather/beauforts.htm.

[36] Kare Bjørvik and Per Hveem. Reguleringsteknikk, 2007.

[37] Chris Meissen, Kristian Klausen, Murat Arcak, Thor I Fossen, and Andrew Packard. Passivity-based formation control for uavs with a suspended load. *IFAC-PapersOnLine*, 50(1):13150–13155, 2017.

[38] Héctor P Rotstein. Optimal filtering with delayed and non-delayed measurements. In *15th IFAC Triennial World Congress*, 2002.

[39] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[40] Bernt Lie, David Di Ruscio, Rolf Ergon, Bjørn Glemmestad, Maths Halstensen, Finn Haugen, Saba Mylvaganam, Nils-Olav Skeie, and Dietmar Winkler. Modeling, identification and control at telemark university college. 2009.

Vuk Krivokapic

Automatic landning of milti-rotor on moving platform

NTNU
Kunnskap for en bedre verden