Model Predictive Control for the BlueROV2

Theory and Implementation

Emil Már Einarsson, Andris Lipenitis
Intelligent Reliable Systems, IRS10-1-E20, 2020-10

Master's Thesis

Department of Energy Technology
Niels Bohrs Vej 8
DK-6700 Esbjerg
https://www.et.aau.dk/

AALBORG UNIVERSITY
STUDENT REPORT

Title:
MPC control for the BlueROV2 - Theory and Implementation

Theme:
Scientific Theme

Project Period:
Fall Semester 2020

Project Group:
IRS10-1-E20

Participant(s):
Emil Már Einarsson
Andris Lipenitis

Supervisor(s):
Simon Pedersen

Copies: 0

Page Numbers: 101

Date of Completion:
October 18, 2020

Abstract:

Model Predictive Controller (MPC) controllers are commonly used in control of process plants, but due to the high computation load, they have not been that popular for small embedded systems. Due to big leaps in the computational power of small embedded computers, MPCs can be implemented on a wide variety of other embedded control problems. In this thesis, we modify the software and hardware of the original commercially available open-source platform BlueROV2 to make it more suitable and better optimized to run an MPC and a Kalman filter. The code is written in Python using object-based programming principles. A comparison between the original system and our modifications is made. To implement an MPC controller on this modified system, the Remotely Operated Vehicle (ROV) is modelled in 6 Degrees of Freedom (DoF), and the model is validated. State feedback for the controller is done using a Kalman filter for sensor fusion of the Inertia Measurement Unit (IMU) and Underwater Global Positioning System (UGPS). Then, a comparison is made with a Proportional–Integral–Derivative (PID) and an LQR implemented on the BlueROV2.

Contents

Preface

The project entitled *MPC control for the BlueROV2 - Theory and Implementation* was made by two master students from the Intelligent and Reliable Systems Engineering Programme at Aalborg University Esbjerg, as a master's thesis.

Aalborg University, October 19, 2020.

Emil Már Einarsson
<eeinar14@student.aau.dk>

Andris Lipenitis
<alipen14@student.aau.dk>

Acronyms

AHRS	Attitude and Heading Reference System
AI	Artificial Intelligence
API	Application Programming Interface
APS	Acoustic Positioning System
AUV	Autonomous Underwater Vehicle
CB	Center of Buoyancy
CG	Center of Gravity
CM	Center of Mass
CO	Center of Origin
CPU	Central Processing Unit
DoF	Degrees of Freedom
DVL	Doppler Velocity Logger
ENU	East-North-Up
ESC	Electronic Speed Control
GPIO	General Purpose Input/Output
GPS	Global Positioning System
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
I2C	Inter-Integrated Circuit
IMU	Inertia Measurement Unit
INS	Inertia Navigation System
IPOPT	Interior Point OPTimizer
KKT	Karush–Kuhn–Tucker

LQG Linear-Quadratic Gaussian

LQR Linear-Quadratic Regulator

MAVlink Micro Air Vehicle Link

MEMS Micro-electro-mechanical systems

MIMO Multiple-Input-Multiple-Output

MPC Model Predictive Controller

MUMPS MUltifrontal Massively Parallel sparse direct Solver

NED North-East-Down

NMPC Non-linear Model Predictive Controller

NPU Neural Processing Unit

OPC Optimal Control Problem

PD Proportional–Derivative

PID Proportional–Integral–Derivative

RAM Random Access Memory

ROV Remotely Operated Vehicle

SBC Single Board Computer

SBL Short BaseLine

SISO Single-Input-Single-Output

SNAME Society of Naval Architects and Marine Engineers

SPI Serial Peripheral Interface

TOPS Tera Operations Per Second

UAPS Underwater Acoustic Positioning System

UART Universal Asynchronous Receiver/Transmitter

UGPS Underwater Global Positioning System

USBL Ultra-Short BaseLine

Chapter 1

Introduction

1.1 Overview

Water covers more than 70% of the earth's surface, and around 95% of the ocean remains unexplored [1]. With the world's increasing energy demand, there is a need for more offshore energy projects. These projects include but are not limited to wave energy, tidal energy, ocean thermal energy and offshore drilling, which also demand more and better underwater exploration, construction, inspection and maintenance. In the oil industry alone, the offshore production percentage is about 30% [2], as seen in figure 1.1.

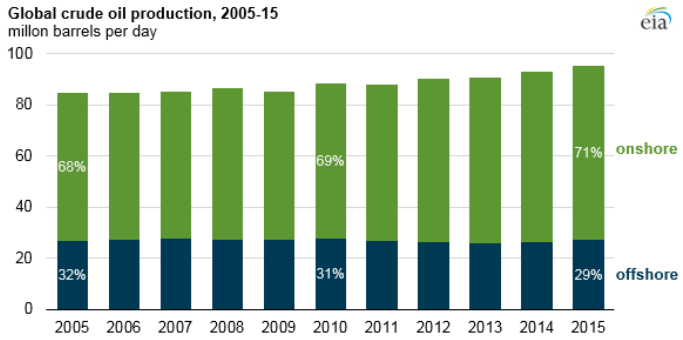

Figure 1.1: Offshore oil production [2]

Underwater vehicles can be classified into manned and unmanned vehicles; this project looks further into the unmanned vehicles. The unmanned vehicles can also be further classified into Autonomous Underwater Vehicle (AUV)s and Remotely Operated Vehicle (ROV)s and from there into inspection or intervention class. The classification is not as straightforward as this, since an ROV can have the same functions and controls as an AUV but with the opportunity for human intervention or full human control [3] [4]. Both the usage and the cost of operating ROVs have increased in the last years, and plans show that they will continue to increase [4], see figure 1.2.

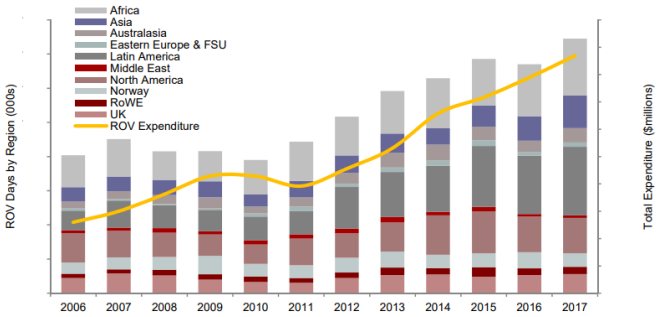

Figure 1.2: ROV increased use and expenditure [4]

The cost of an underwater project is a determining factor in whether a manned or unmanned vehicle option is selected. The daily operating cost of an ROV starts from 5000-10,000 USD/day [5] and the cost of a single diver starts from 1500-5000 USD/day [6] depending on the project. Neither group works alone, and therefore the cost is quick to add up. Some of the main differences in usage of the two are limitations with depth and work rate with different assignments. For manned diving, the most significant limitation is with depth and time in the water. Current oil platforms are built down to almost 3 km depth, see figure 1.3 [7]. Time at depth is also a limitation for a diver. A diver can dive down to 6 m and stay there for unlimited time without the need to decompress on the way back. Diving to a depth of 36 m, a diver can only stay down for 15 minutes without needing to decompress on ascent. Even with decompression, there is an absolute maximum dive time of only 4-6 hours. After 2 hours diving at 36 m, the diver would need 210-580 minutes of decompression time in stages on the way up, depending on the gas mix used [8]. Saturation divers can and do work down to 300 m using a diving bell and working 4-6 hours in the water at a time. The divers will then stay down there up to 6 weeks before needing to come up and decompress. The advantages of using divers are that they can sometimes work quicker than an ROV, especially in tight spaces, and divers can use off-the-shelf tools while ROVs need specialised tools. The benefits of using an ROV are that it can dive deeper and stay in the water until the task is finished [9].

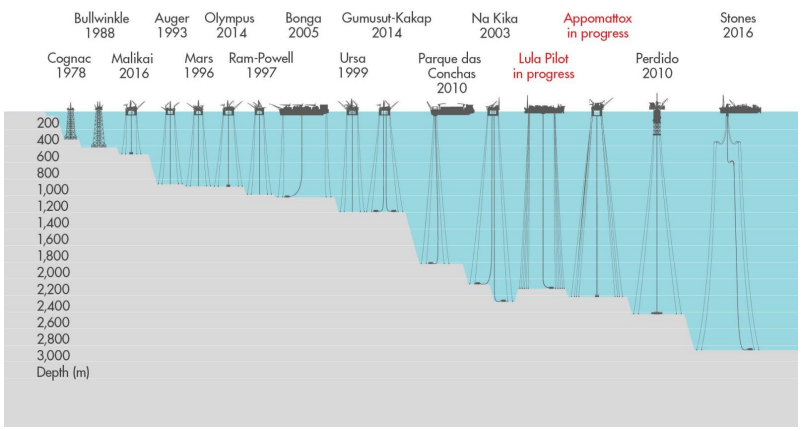

Figure 1.3: Shell oil platforms overview [7]

1.2 State of the Art

In 2017 the globally available ROV fleet only saw 32% use, but analysts expect that number to reach 50% in 2023 as demand for offshore support roles and use in traditional oil and gas applications continues to increase. With the future projects moving into deeper waters and the industry looking at optimizing costs, the use of AUVs is also on the increase [10].

The use of underwater vehicles has been proven in marine geoscience [11] [12] [13], in deep-sea archaeology [14] [15] and in the offshore industry [10] [16] [17]. This success would not be there if the underwater vehicles did not use some type of navigation. The first autopilot was designed by Elmer Sperry in 1911, and it has since been an active research field [18]. The first control systems used only Proportional–Derivative (PD) and Proportional–Integral–Derivative (PID) as control methods to steer the marine vessels, and it was not until the 1970s when the control precision was down to 1 meter and a refresh rate of 1 Hz that a control system could be used on underwater vehicles [19].

Today most of the ship motion control systems are still based on PID, but more recently a lot of them have been replaced by controllers based on Linear-Quadratic Gaussian (LQG) and H_∞ control design techniques. These design techniques allow frequency-dependant notch filtering of first-order wave-induced forces in order to prevent wear and tear of thruster and propeller systems. Multiple-Input-Multiple-Output (MIMO) control strategies are advantageous because they can deal with coupling between different inputs and outputs. That would not be possible with a classical PID controller which would be a Single-Input-Single-Output (SISO) system and decoupled [20].

These linear control theories rely on taking the nonlinear model of the system and linearising it. Certain conditions need to be fulfilled to make the linear model very close to the nonlinear model: the pitch, yaw and roll angles must be small, and the surge, heave and sway speeds must be constant. [20]. In 1992 Healey and Marco also proposed a decoupled 6 Degrees of Freedom (DoF) system that simplifies the control into three main subsystems: forward speed, steering and diving [21].

Optimal control deals with the problem of finding the control law such that some cost criterion would be satisfied. Nowadays, efficiency is essential to minimize the time, the cost and the environmental footprint of a project. Linear-Quadratic Regulator (LQR)s are already widely used for motion control of marine craft, but for these controllers the control law is calculated only once. This means that some variables like, for example, velocity, have to be constant. [20] Another type of optimal controller, a Model Predictive Controller (MPC), avoids this problem by doing similar calculations as for LQR at different intervals. MPCs require more computational power than LQRs, but with the performance increase attained in technology over the last decade, MPCs find their use in an ever-growing array of applications. The use of MPCs with AUVs can also be seen in Chao 2018 [22].

Many factors make the autonomous control of an underwater vehicle difficult, but the main contributors are the following [18]:

- Underwater position systems' resolution and refresh rates.
- The nonlinear system dynamics.
- Multiple inputs and multiple outputs systems.
- Parameter uncertainties from poor knowledge of the hydrodynamic coefficients.
- Unpredictable external disturbances caused by underwater currents, tether and payload.
- Limitations on actuators (underactuated).

This list is not finite but draws up some of the significant factors that need to be looked into when designing a controller for an underwater vehicle.

1.3 Problem Statement and Definition

Articles like [10] show clear trends in AUV research. The overall trends are decreasing operating costs and increasing uptime. More specifically, these goals are achieved by increasing the travel distance before recharge, decreasing the assistance needed from divers, adding more operations that can be done unattended, increasing the lifting capabilities in order to carry more extensive instrumentation and more [10]. Some of these can be expressed as optimisation problems. For example, to increase the range capability of one charge, the controller has to use as little force as possible to achieve the necessary control objective. Also, load-carrying is an optimisation problem. Besides the current trends, the core needs of an AUV are the same: to achieve the necessary position and velocity, and maintain them. Optimal controllers, more specifically MPCs, can deal with both the core objectives and current trends with relative ease, but because the MPC controllers are computationally heavy, actual implementation in embedded systems can be challenging.

Problem definition:

- Analyse the current hardware and software on the BlueROV2 for their suitability for MPC implementation.
- Identify the shortcomings and possible improvements that can be made.
- Implement an embedded MPC controller for the ROV and analyse the controller performance compared to controllers already commonly used in the industry, namely PID and LQR.

In our project, we will design an MPC controller in order to control an underwater vehicle's motion in 6 DoFs, while also looking at limiting the energy consumption of the BlueROV2 operation.

Chapter 2

Modelling and Hydrodynamics of the BlueROV2

The platform used in this project is the BlueROV2 made by Blue Robtics Inc. The BlueROV2 is an open-source commercially available underwater vehicle that has a large and growing community of both academic and privately-motivated people. The introduction of an open-source community and a platform that is in the price range of the average consumer opens up testing and development of new applications for underwater vehicles where there was limited or no research done before. The ROV can be seen below in figure 3.1.

Figure 2.1: BlueROV 2 taken from Blue Robotics Inc. website [23]

The notations and equations of motion are based on Fossen [18]. Fossen's modelling for ships, rigs and underwater vehicles is an efficient way to describe the 6 DoF with differential equations in matrix form; the model can also include the effects of coupling between degrees. There also exist a number of books on the mathematical modelling of ships; see [24], [25], and [26]. For underwater vehicles, see [27], [28], [29], [30], and [31]. For ships' hydrodynamics, see [32], [33], and [34]. The mathematical model will be derived in this chapter and the work will be explained, and then the model will be validated and tuned if needed. The hydrodynamic model of the ROV is used to simulate the ROV motions and exploit the knowledge gained about the ROV dynamics for designing observers and controllers.

2.1 Notations

The BlueROV2 uses six thrusters to move around in the water. Four of them are laid in a vectored thrust orientation that gives the ROV freedom to move in sway, surge and yaw directions. The other two thrusters give the ROV freedom to move in the heave direction. The thruster set-up also gives the ROV the capability to move in the roll direction. Limitations in the current programming do not allow for that. The only degree of freedom that the ROV cannot be programmed to move in without the addition of two extra thrusters is the pitch. The ROV, therefore, has 4 DoF to move in. With a little change in the code, the 5th one, roll, can be added. The DoF and the set-up of the thrusters can be seen in figure 2.2 and the Society of Naval Architects and Marine Engineers (SNAME) notation for the degrees of freedom are in table 2.1.

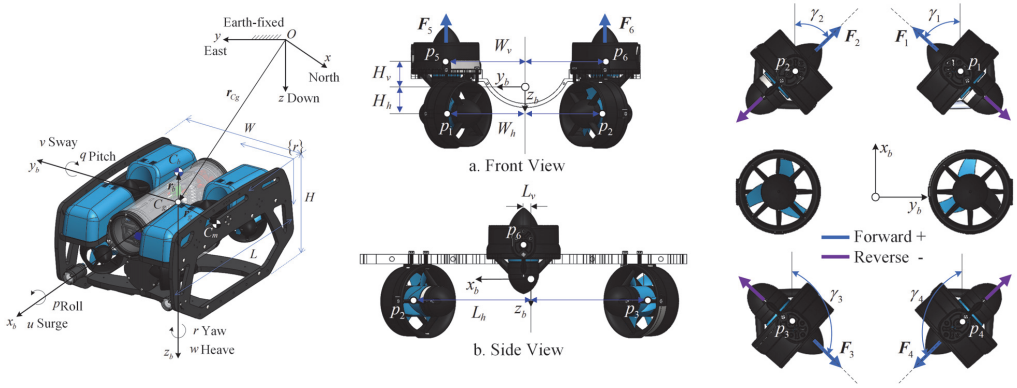

Figure 2.2: The BlueROV2 coordinates, structure frames and thrusters placement [35]

DoF	Movement Notation	Forces and moments	Linear and angular velocities	Positions and Euler angles
1	Motions in x direction (surge)	**X**	u	x
2	Motions in y direction (sway)	**Y**	v	y
3	Motions in z direction (heave)	**Z**	w	z
4	Rotation about the x axis (roll)	**K**	p	ϕ
5	Rotation about the y axis (pitch)	**M**	q	θ
6	Rotation about the z axis (yaw)	**N**	r	ψ

Table 2.1: SNAME notation for marine vessels [36].

It is not only the degrees of freedom the ROV can move in that are essential; it is also the sensory data available that the ROV can use for feedback. The ROV has four primary sensors available. They are the Inertia Measurement Unit (IMU), the pressure sensor, the camera, and an Underwater Acoustic Positioning System (UAPS). From these sensors, the movement of the ROV is used as feedback for the system.

2.1.1 Coordinate

Using the SNAME notation seen in table 2.1, equations (2.1) and (2.2) gives the generalized position, velocity and forces [18].

$$\boldsymbol{\eta} = [x\ y\ z\ \phi\ \theta\ \psi]^\top \tag{2.1}$$
$$\boldsymbol{\nu} = [u\ v\ w\ p\ q\ r]^\top \tag{2.2}$$

The linear and angular position and velocity are seen in (2.3).

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad , \quad \mathbf{v} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad , \quad \boldsymbol{\Theta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad , \quad \boldsymbol{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.3}$$

where $\mathbf{p} \in \mathbb{R}^{3x1}$ is the liner position, $\mathbf{v} \in \mathbb{R}^{3x1}$ is the linear velocity, $\boldsymbol{\Theta} \in \mathbb{R}^{3x1}$ is the angular position, also called attitude. $\boldsymbol{\omega} \in \mathbb{R}^{3x1}$ is the angular velocity, also know as the rotational velocity.
The force vectors can be seen in equation (2.4).

$$\boldsymbol{\tau} = [\mathbf{X}\ \mathbf{Y}\ \mathbf{Z}\ \mathbf{K}\ \mathbf{M}\ \mathbf{N}]^\top \tag{2.4}$$

2.1.2 Frame of Reference

The system has multiple frames of reference that can be seen in figure 2.3. The frames are the IMU East-North-Up (ENU) frame, the ROVs body frame, the pool frame and the global North-East-Down (NED). To be able to implement a control system, the calculations need to be projected to the same frame of reference. For this project, the control calculations will be done in the body frame and shown in the pool frame so they will be more intuitive. Rotational matrices are used to move the different measurements between the different frames; the rotational matrices are shown in chapter 2.2.

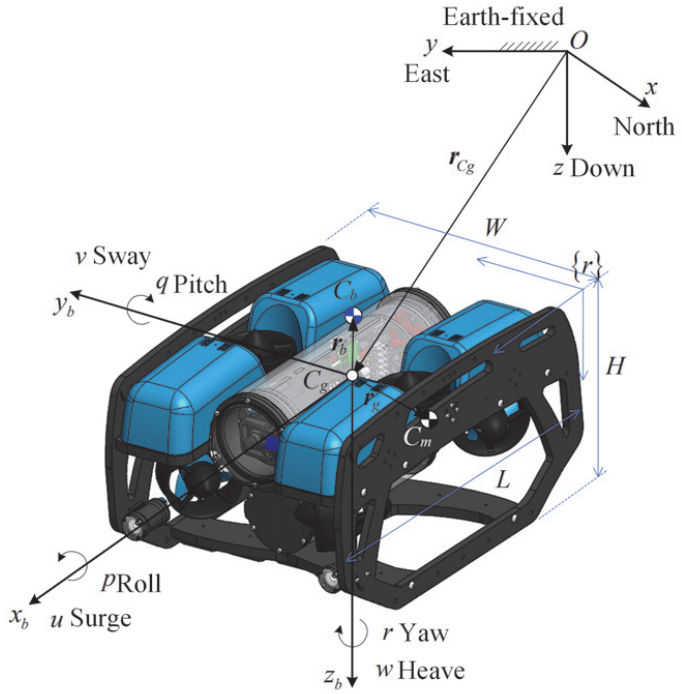

Figure 2.3: The BlueROV2 frame of reference [35]

2.2 Kinematics

The background material needed to understand the mathematical model of the ROV, and how the equations of motions are presented in matrix form with the reference frames, are explained in this section. To rotate from the body frame to the NED frame a set of rotational matrices is needed. Equations (2.5), (2.6), (2.7), and (2.8) show how the transformation can be made.

$$\dot{\eta} = J(\eta)\nu \tag{2.5}$$

$$J(\eta) := \begin{bmatrix} \boldsymbol{R}(\boldsymbol{\Theta}) & 0_{3x3} \\ 0_{3x3} & \boldsymbol{T}(\boldsymbol{\Theta}) \end{bmatrix} \tag{2.6}$$

where

$$\boldsymbol{R}(\boldsymbol{\Theta}) = \begin{bmatrix} c\psi \cdot c\theta & -s\psi \cdot c\phi + c\psi \cdot s\theta \cdot s\phi & s\psi \cdot s\phi + c\psi \cdot c\phi \cdot s\theta \\ s\psi \cdot c\theta & c\psi \cdot c\phi + s\phi \cdot s\theta \cdot s\psi & -c\psi \cdot s\phi + s\theta \cdot s\psi \cdot c\phi \\ -s\theta & c\theta \cdot s\phi & c\theta \cdot c\phi \end{bmatrix} \tag{2.7}$$

with $s\cdot = sin(\cdot), c\cdot = cos(\cdot)$ and $t\cdot = tan(\cdot)$. The $\mathbf{R}$ matrix is called the Euler angle rotation matrix, satisfying $\mathbf{R}\mathbf{R}^\top = \mathbf{R}^\top\mathbf{R} = \mathbf{I}$ and the $det(\mathbf{R}) = 1$, which implies that $\mathbf{R}$ is orthogonal. The inverse rotation matrix is: $\mathbf{R}^{-1} = \mathbf{R}^\top$. The Euler rates $\dot{\boldsymbol{\Theta}} = \mathbf{T}(\boldsymbol{\Theta})\mathbf{w}$ are singular for $\theta \neq \pm\pi/2$. Singularity can be avoided by using quaternions units [18].

$$\boldsymbol{T}(\boldsymbol{\Theta}) = \begin{bmatrix} 1 & s\phi \cdot t\theta & c\phi \cdot t\theta \\ 0 & c\theta & -s\theta \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \quad , \quad \theta \neq \pm\frac{\pi}{2} \tag{2.8}$$

2.3 Equations of Motion

The invention of the gyroscope led to the development of the automatic pilot, making it the critical breakthrough for ship control - the invention of the Global Positioning System (GPS) allowed ships to be automated even further. Though the first autopilot could only maintain the heading, nowadays it can do even sophisticated manoeuvres such as docking [20].

The motion control systems for ships and other water vehicles consist of three independent blocks: guidance, navigation and control [20].

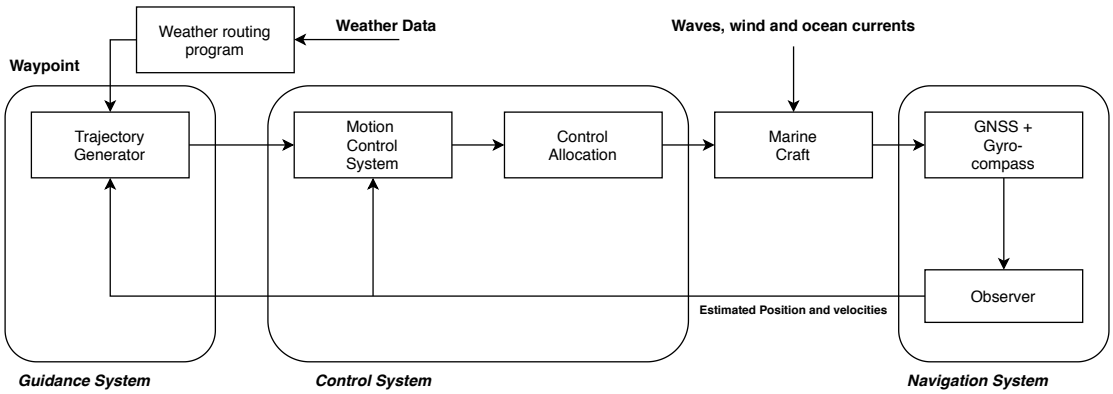

Figure 2.4: Motion control systemn [20]

In short, guidance takes in weather data, navigation data and human input to continuously calculate the desired position, velocity and acceleration of a marine craft. Navigation generally deals with determining its position, attitude, course and distance travelled. Finally, control determines the necessary control forces and moments for achieving the control objective. In our project, we intend to work with only the navigation and control and provide the reference position ourselves [20].

An important distinction for motion control is whether the marine craft is underactuated or fully actuated. An underactuated marine craft has limits in attainable control objectives. In this case, the control objective has to be formulated in a way that allows the craft to satisfy the requirements even when the control forces can be applied only in some directions. Unfortunately, most marine crafts are underactuated, including the platform used in the project BlueROV2, meaning they cannot produce control forces and movements in all DoF [20].

DoF can be defined as follows: "For a marine craft, DoF is the set of independent displacements and rotations that can completely specify the displaced position and orientation of the craft. A craft that can move freely in 3-D space has a maximum of 6 DoFs, three translational and three rotational elements" [20].

When simulating motion of a craft that is fully actuated in 6 DoFs, it is necessary to use 12 ordinary differential equations [20].

$$Order = 2 \times \text{DoF} \tag{2.9}$$

To design a control system, it is necessary to define a workspace in which the control objective is specified. To do this, the configuration space and workspace have to be defined. All the possible positions and orientations a craft can attain are defined as configuration space [20]. Underwater vehicles operate in 6 DoF, and the displacement and rotations are described by six generalized positions and velocities, where Euler angles are defined on the interval $S = [0, 2\pi]$ [20].

2.3.1 Kinetics

The process plant model seen in equations (2.10) and (2.11) describes the Newton-Euler equations of motions about the Center of Origin (CO). This model is based on Fossen [20].

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{2.10}$$

$$\underbrace{\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}}_{\text{rigid-body forces}} + \underbrace{\mathbf{M}_{A}\dot{\boldsymbol{\nu}} + \mathbf{C}_{A}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{\text{hydrodynamical forces}} + g(\eta) = \boldsymbol{\tau} + \boldsymbol{\tau}_{ext} \tag{2.11}$$

Where

$M_{RB} \in \mathbb{R}^{6x6}$ is the rigid body mass matrix in CO.

$C_{RB}(\boldsymbol{\nu}) \in \mathbb{R}^{6x6}$ is the rigid body Coriolis and centripetal forces matrix.

$M_A \in \mathbb{R}^{6x6}$ is the added mass in CO.

$C_A(\boldsymbol{\nu}_r) \in \mathbb{R}^{6x6}$ is the Coriolis forces for added mass.

$D(\boldsymbol{\nu}_r) \in \mathbb{R}^{6x6}$ is the damping matrix.

$g(\eta) \in \mathbb{R}^{6x1}$ is the hydrostatic restoring forces vector.

$\tau \in \mathbb{R}^{6x1}$ is the propulsion forces vector.

$\tau_{ext} \in \mathbb{R}^{6x1}$ is a an external force vector that includes tether.

Equation (2.11) is Newton's second law of motion expressed in a moving coordinate frame, thus the need to compensate for the Coriolis and the centripetal forces [18]. $\boldsymbol{\nu}_r \in \mathbb{R}^{6x1}$ is the relative velocity vector with respect to the water and is combined of $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$, where $\boldsymbol{\nu}_c \in \mathbb{R}^{6x1}$ is the underwater current decomposed in the body frame [20].

2.3.2 Underwater Vehicle Model

The rigid body inertia matrix $\mathbf{M}_{RB}$ can be seen in equation (2.12). Noted that the Center of Gravity (CG) $\mathbf{r}_g := [x_G, y_G, z_G]^\top$ and Center of Buoyancy (CB) $\mathbf{r}_b := [x_B, y_B, z_B]^\top$ vectors are placed from CO. For the BlueROV2 $x_g = x_b = 0$, $y_g = y_b = 0$ and $z_b = 0$. Only the CG in z_G is placed lower than the CO. Therefore $I_{xy} = I_{yz} = 0$ [37].

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & I_{xy} & I_{xz} \\ mz_G & 0 & -mx_G & I_{yx} & I_y & I_{yz} \\ -my_G & mx_G & 0 & I_{zx} & I_{zy} & I_z \end{bmatrix} \tag{2.12}$$

The $\mathbf{M}_{RB}$ can be simplified further since the Center of Mass (CM) is only offset from the center of origin in z direction. All off-diagonal terms that do not include z_G can be removed and only the momentums around the diagonal terms will stay because of the ROV's symmetry. The same thing can be said for the $\mathbf{C}_{RB}$ matrix, see (2.13) [20] [37].

$$\mathbf{C}_{RB} = \begin{bmatrix} 0 & 0 & 0 & m(y_G q + z_G r) & -m(x_G p - w) & -m(x_G r + v) \\ 0 & 0 & 0 & -m(x_G q + w) & m(z_G r x_G p) & -m(y_G r - u) \\ 0 & 0 & 0 & -m(x_G r - v) & -m(z_G q + u) & m(x_G p + y_G q) \\ -m(y_G q + z_G r) & m(y_G p + w) & m(z_G p - v) & 0 & I_z q & -I_y p \\ m(x_G q - w) & -m(z_G r + x_G p) & -m(x_G p + y_G q) & -I_z q & 0 & I_x p \\ m(x_G r + v) & m(y_G r - u) & -m(x_G p + y_G q) & I_y q & -I_x p & 0 \end{bmatrix} \tag{2.13}$$

The BluROV2 is modelled as having three planes of symmetry. That suggests that only the diagonal terms in $\mathbf{M}_A$ are used, as seen in (2.14). This diagonal method is often popular for small, slow-moving underwater vehicles since the off-diagonal terms are often hard to determine. The off-diagonal terms are also much smaller than the diagonal so leaving them out should not effect the modelling too much [20] [37].

$$\mathbf{M}_A = -\begin{bmatrix} \mathbf{X}_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{Y}_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{Z}_{\dot{w}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{K}_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{M}_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{N}_{\dot{r}} \end{bmatrix} \tag{2.14}$$

The added mass Coriolis force matrix, seen in (2.15), takes up a skew-symmetrical form when moving through ideal fluid [20] [37].

$$\mathbf{C}_A = \begin{bmatrix} 0 & 0 & 0 & 0 & -\mathbf{Z}_{\dot{w}} w & \mathbf{Y}_{\dot{v}} v \\ 0 & 0 & 0 & \mathbf{Z}_{\dot{w}} w & 0 & -\mathbf{X}_{\dot{u}} u \\ 0 & 0 & 0 & -\mathbf{Y}_{\dot{v}} v & \mathbf{X}_{\dot{u}} u & 0 \\ 0 & -\mathbf{Z}_{\dot{w}} w & \mathbf{Y}_{\dot{v}} v & 0 & -\mathbf{N}_{\dot{r}} r & \mathbf{M}_{\dot{q}} q \\ \mathbf{Z}_{\dot{w}} w & 0 & -\mathbf{X}_{\dot{u}} u & \mathbf{N}_{\dot{r}} r & 0 & -\mathbf{K}_{\dot{p}} p \\ -\mathbf{Y}_{\dot{v}} v & \mathbf{X}_{\dot{u}} u & 0 & -\mathbf{M}_{\dot{q}} q & \mathbf{K}_{\dot{p}} p & 0 \end{bmatrix} \tag{2.15}$$

Generally the damping of an underwater vehicle is highly non-linear and coupled. For simplification and for how difficult it is to measure the coupled coefficients of the system, a rough estimation is that since the ROV has three planes of symmetry and performs non-coupled movements, the terms of higher than the second order

can be ignored. The damping matrix takes a diagonal form seen in (2.16) [20] [37].

$$\mathbf{D} = -\begin{bmatrix} \mathbf{X}_{u|u} + \mathbf{X}_u & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{Y}_{v|v} + \mathbf{Y}_v & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{Z}_{w|w} + \mathbf{Z}_w & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{K}_{p|p} + \mathbf{K}_p & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{M}_{q|q} + \mathbf{M}_q & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{N}_{r|r} + \mathbf{N}_r \end{bmatrix} \tag{2.16}$$

Let $W = mg$ and $B = \rho g V$. The ROV is made to be neutrally buoyant, $W = B$. Therefore $\mathbf{g}(\nu)$ can be simplified further and only leave the terms containing $z_g W$ in (2.17) [20] [37].

$$\mathbf{g}(\nu) = -\begin{bmatrix} (W-B)s\theta \\ -(W-B)c\theta \cdot s\phi \\ -(W-B)c\theta \cdot c\phi \\ (z_G W - z_B B)c\theta s\phi \\ (z_G W - z_B B)s\theta + (x_H W - x_B B)c\theta \cdot c\phi \\ -(x_G W - x_B B)c\theta \cdot c\phi \end{bmatrix} \tag{2.17}$$

where

Notations	Description
$\mathbf{I}_b$	Inertia matrix for rotation around the CO
$\mathbf{I}_g$	Inertia matrix for rotation around the CG
$\mathbf{K}_p, \mathbf{M}_q, \mathbf{N}_r$	Linear damping coefficients for rotation in water
$\mathbf{K}_{p\|p}, \mathbf{M}_{q\|q}, \mathbf{N}_{r\|r}$	Quadratic damping coefficients for rotation in water
$\mathbf{K}_{\dot{p}}, \mathbf{M}_{\dot{q}}, \mathbf{N}_{\dot{r}}$	Increased inertia about $x, y, z - axis$ due to rotation in water
$\mathbf{X}_u, \mathbf{Y}_v, \mathbf{Z}_w$	Linear damping coefficients for translation in water
$\mathbf{X}_{u\|u}, \mathbf{Y}_{v\|v}, \mathbf{Z}_{w\|w}$	Quadratic damping coefficients for translation in water
$\mathbf{X}_{\dot{u}}, \mathbf{Y}_{\dot{v}}, \mathbf{Z}_{\dot{w}}$	Added mass in $x, y, z - direction$ due to translation in water
$I_{x_i}, I_{y_i}, I_{z_i}$	Moment arms from centre of gravity to each thruster
m	ROV's mass
V	ROV's displaced volume
ρ	Water density
g	Gravitational constant
z_G	Distance between CO and CG in z-axis

Table 2.2: Notation and description of the parameter used in the ROV model [18] [37]

2.3.3 Simplified Model

The simplified model seen in 2.18 and 2.19 is used for analysis, design of observers and controllers. It assumes that

- The vehicle velocity is small. Therefore Coriolis, centripetal forces and non-linear damping can be neglected.
- Underwater currents are constant or non-existent. Therefore the equation is given in terms of the vehicle velocity ν.
- Roll and pitch motions are below 10 degrees, and the ROV is assumed to be neutrally buoyant with the CB straight over the CG. Then linearization can be used on the restoring forces using $\mathbf{G}$.

This simplified model will still capture the main dynamics of the ROV.

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{2.18}$$

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{D}\boldsymbol{\nu} + \mathbf{G}(\eta) = \boldsymbol{\tau} + \mathbf{J}^{\top}(\boldsymbol{\eta})\mathbf{b} \tag{2.19}$$

$$\dot{\mathbf{b}} = -\mathbf{T}_b^{-1}\mathbf{b} + \boldsymbol{\omega}_b \tag{2.20}$$

Where $\mathbf{D} \in \mathbb{R}^{6x6}$ is the linear damping matrix and $\mathbf{b} \in \mathbb{R}^{6x1}$ is the bias estimation for slow-changing forces like ocean currents, change of mass and errors in the modelling. $\mathbf{G}$ is the linearized restoring forces matrix [20].

2.4 Forces

The main forces acting on the ROV are the underwater currents, disturbances from the tether and the thrusters. The scope of this thesis will not include modelling the underwater currents and tether dynamics of the system. This thesis will address these forces in theory but will not go into modelling them into the control model. These forces will be classified as external disturbances to the system.

2.4.1 External Forces

The main external forces acting on the ROV would be underwater currents and the tether pulling on the ROV. If the underwater currents were considered, then they could be modelled as vertical and horizontal forces acting on the ROV in the NED frame, and by applying the rotational matrix, seen in chapter 2.2, to the forces, they could be moved to the body frame and implemented as Gaussian white noise. Then the total force in $\boldsymbol{\nu}_r$ would be broken down to $\boldsymbol{\nu}$ force already generated by the system and $\boldsymbol{\nu}_c$ as the currents acting on the system.

2.4.2 Generated Forces

The BlueROV2 uses six thrusters to move the ROV around in the water, and the set-up can be seen in previous figure 2.2. The ROV is under-actuated since it is

missing two thrusters that would be necessary to control the ROV in the pitch motion. The forces generated by every single thruster combined together make up the total control force. It is then essential to measure the actual thrust generated so this can be used for the force calculations of the system. Measuring the thrust is a difficult task since the thruster forces are complex, and an error in modelling them could cause an even more significant error when estimating the total thrust vector τ.

For our test, a 1 m long cable was tied to the top of the ROV, and the two heave thrusters were activated, giving the ROV downwards force. The thrust started from 0% and then stepped up in 5% increments all the way to 100%. A static reading at each step was taken with a newton meter, and from that, a force map was made as seen in figure 2.5.

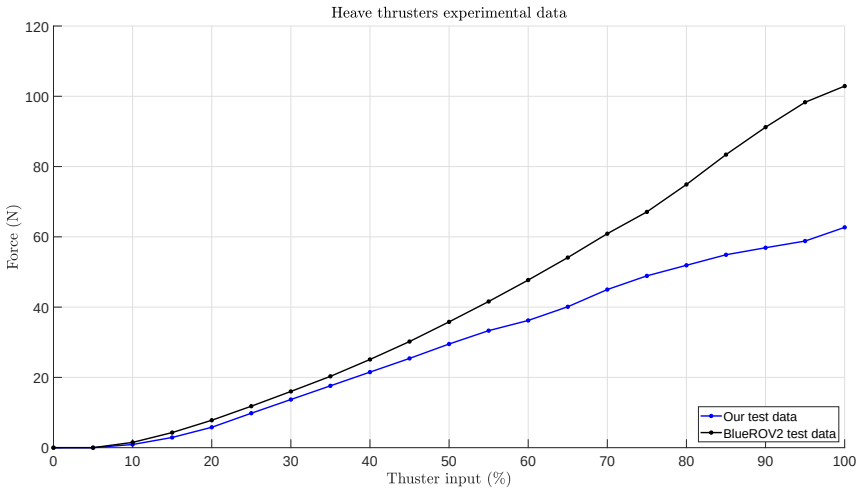

Figure 2.5: Heave thrusters experimental data comparison

In figure 2.5 it can be seen that max thrust is 31 N. That is only 65% of the max thrust of the T200 raw data thruster test on the website [23]. The reason for this considerable difference could be one of the following:

- The BlueROV2 T200 test set-up uses an external voltage source and therefore does not have the same voltage drop as the lithium-ion batteries we used. Assuming that our battery is fully charged at 16.8 v or 4.2 v per cell, at a 50 A load the voltage drop could be about 3.8 v or close to 15.2 v. There is also a voltage drop over the wires that extend from the battery to the Electronic Speed Control (ESC) and finally to the motors.
- The frame of the ROV could be interfering with the flow of the water and could choke the flow or generate turbulence before entering the thruster.
- The BlueROV2 test is a bollard thrust test done in a small 1500 L tank under static conditions. If the water velocity at the inlet is nonzero, meaning that the water in the test environment starts circulating, then the change in momentum will be imparted on the water flowing through, therefore resulting

in a lower measured thrust [38]. Therefore we might miss the initial reading for the thruster, and the static reading would show less thrust than the motor can actually produce.

- Small changes like buoyancy changes from extra air bubbles, the elastic properties of the cable used, the angle of the ROV pulling on the cable, the battery draining, and other changes could affect the test.

From the list above, it can be seen that there could be many uncertainties when measuring the thrust. The list also does not consider the different thrust generated by reversing the thruster and different thruster dead-spaces. From this initial test, the thrust is assumed to be as correct as our current testing facilities can allow and will be used later in the modelling of the system.

2.5 Sensors

The BlueROV2 platform comes with a set of sensors pre-mounted on the platform. This section describes the primary sensors used for navigation of the ROV.

2.5.1 Inertia Measurement Unit

The IMU model is the MPU6000. The 9 DoF IMU is a combination of three accelerometers, three gyroscopes and three magnetometers [23]. The improvement of Micro-electro-mechanical systems (MEMS) technology has resulted in a small and inexpensive but highly accurate IMU that is well-suited for use in small vehicles. The small IMU can have an update rate of 100-1000 Hz. Even though the new MEMS sensors have shown many improvements, there are still some challenges with the gyroscope and accelerometer noise and drift. The magnetometer also needs to be calibrated to its working environment, and it can still be influenced by disturbances from electricity and proximity to metals.

2.5.2 Pressure Sensor

The pressure sensor the ROV uses is the MS5837-30BA, which has a resolution of 0.2 mbar, or about 2 mm. The depth sensor can give readings down to 300 m. It has a refresh rate of 50 Hz [23].

2.5.3 Camera

The ROVs camera is a 1080p low-light USB camera built on the IMX322 sensor [23]. The camera will not be used as an extra sensor in this project. However, the addition of computer vision with the camera could be an option for an additional sensor, if needed.

Chapter 3

ROV Motion Control System

This thesis goes into the development, implementation and testing of a motion control system for the BlueROV2. The model used in this project is based on the initial work done as a bachelor thesis in the spring of 2018 at Aalborg University Esbjerg titled ***"Stabilization of BlueROV2 in three-dimensional space"***, and on the modified model done as a semester project in the spring of 2019, ***"Fault Detection on the BlueROV2 using Multi-Model Residuals"***. Those two models are used as a starting point for this thesis model validation and tuned if needed. This chapter describes the model validation of the system and the motion control system architecture made for this thesis.

3.1 Introduction

The motion control system had a complete overhaul both with software and many hardware components. The motion control system software was redesigned and written mainly in Python, and then compared to the motion control system that comes standard with the system from Blue Robotics. This chapter does not go into all the details of the developments made, but it will give some insight into the new hardware implemented and how the new programming structure compares to the original one. This chapter will also show the results from the model validations of the BlueROV2 mathematical model when run with the new motion control system and validated using Simulink and Matlab.

3.2 Software and Hardware Development

3.2.1 BlueROV2 Hardware

The BlueROV2 hardware diagram can be seen in figure 3.1. It shows how the different sensors and actuators communicate with one another and how the data is

sent up to a top-side control station. The arrows indicate in which direction the data and power flow.

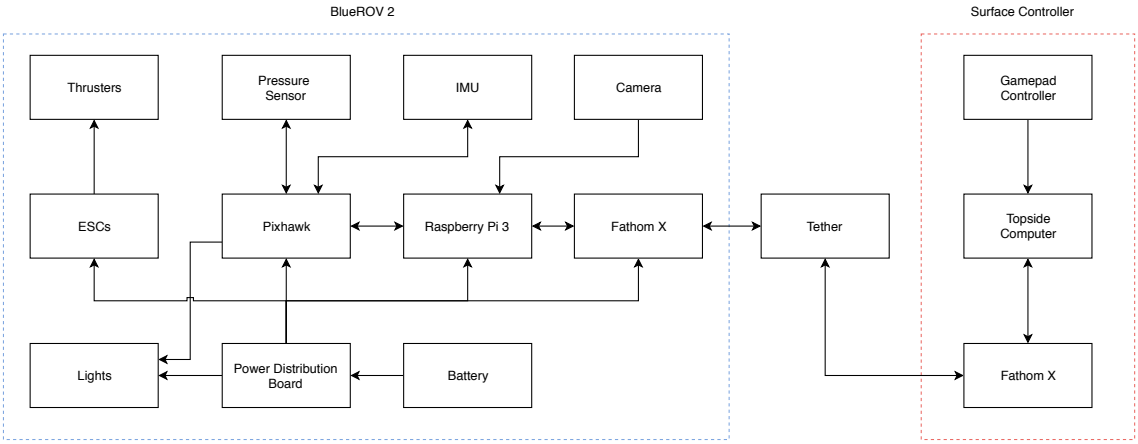

Figure 3.1: BlueROV2 hardware diagram [39]

3.2.2 Raspberry Pi and Pixhawk

The BlueROV2 uses the Raspberry Pi 3b and the Pixhawk Flight controller for the control and communication to the surface. The Raspberry Pi runs Blue Robotics Companion Computer image and links over the tether using Ethernet to send telemetry to the surface. The Pixhawk is the true 'brains' of the system and has the Input/Output (I/O) of the system. The Pixhawk runs the ArduSub open-source solution as the base for the BlueROV2 control code that uses the Micro Air Vehicle Link (MAVlink) as the communication protocol. Using the ArduSub Application Programming Interface (API), Bluerobotics has written their control algorithm specially designed for the BlueROV2 platform. This set-up has its limitations, mostly with being set into a predefined software and code which might not suit all applications of the ROV.

A computer on the top-side, with QGroundControl, is needed to communicate with the ROV. QGroundControl takes the telemetry data sent by the ROV over the tether and puts it into a visual model for the user. The software set-up can be seen in figure 3.2. The figure shows how the different components of the ROV communicate.

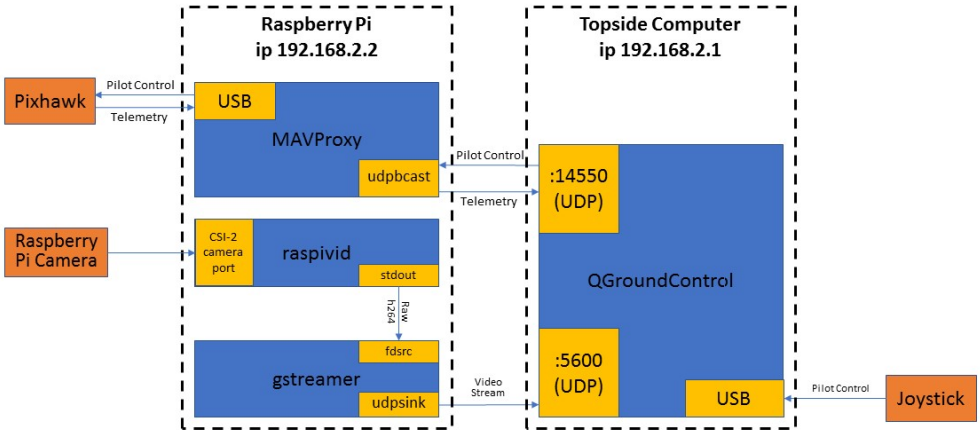

Figure 3.2: BlueROV2 software diagram [40]

3.3 BlueROV2 Hardware and Software Limitations

When working with the original hardware on the BlueROV2, we realized that there are multiple limitations to the Raspberry Pi and Pixhawk combination. Even though the BlueROV2 platform is open-source, there was not enough documentation available about what filtering data is put through. The existing fusion algorithms were a black box, but the raw data had problems with the consistency of the sample update rate. This could be explained by the fact that Pixhawk has essentially the same microcontroller in its core as the Arduino. This obviously results in performance issues when it has to sample data, run the fusion algorithms, control the thrusters, and keep communicating with the companion computer that is running our controller script. Regarding communication between Pixhawk and our script running on the Raspberry Pi, the MAVlink protocol that was implemented for it raised some challenges. Other groups had been working with the same platform; from the code snippets left behind, it was clear to see that they were all making the same mistake and that it was directly caused by the way MAVlink was used in the Pixhawk implementation. The user could select at which frequency the specific data like pressure, orientation or position should come. Then, all the sensor data was streamed on a socket in a continuous stream. We, along with the other groups, originally used the MAVlink function in the following manner:

```
att = autopilot.recv_match(type='ATTITUDE',blocking=True).
```

The goal was to get attitude data and save it in a variable we could later use in the control algorithm, so the function would listen to the stream and block the execution

of the main script until the message of this specific type had been received, ignoring any other messages in the meanwhile. The problem was compounded when we needed data with a low update rate, like Waterlinked's UGPS, which has a maximum 4Hz sampling rate. The whole script would be frozen for as long as 250ms until the message was received. The sample time for the code would vary anywhere between 15ms and 260ms, depending on whether or not the requested data was already in the pipeline.

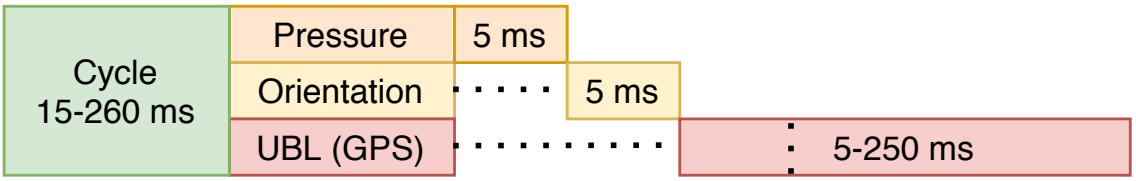

Figure 3.3: Varying sensor sampling rate of the BluROV system

It was evident that we needed a solution to fix the varying sample time. We would create a second thread on a separate Central Processing Unit (CPU) core to receive all of the MAVlink messages, and then when a message of importance had been received, we would update a thread-safe variable that was shared between processes. In this case, we had a state vector that had different parts updated at different rates by another process. So, when the control loop would read the variable, it would have the newest sample for each sensor.

However, the problems did not end there. The next problem that arose was that Pixhawk would randomly start sending data at rates per the factory settings. Then during one of our tests, there was an unexpected failure of the Pixhawk, and it would not boot any more. We also realized that using Pixhawk and MAVlink increased the delay for the UGPS. So instead of ordering a new Pixhawk, we decided to replace the Raspberry Pi 3B+ with a single board computer that has higher computational capabilities. With the new computer, we could communicate with sensors and thruster controllers directly instead of using Pixhawk. The new set-up is described in detail chapter 3.4.

3.3.1 Tests with the BlueROV2 Set-up

Below in figure 3.4, the system is run with an open-loop test set at sampling every 0.02s. The system can be seen to have different sampling rates while the program is kept on hold, waiting for new data. The same problem would affect the control code, and the system could not maintain a set sampling rate. This test was run without the UGPS, which would cause even longer delays as seen in figure 3.5. The system can also be seen changing the set sampling rate at the start of the loop to the system defaults in the middle of a run without being asked.

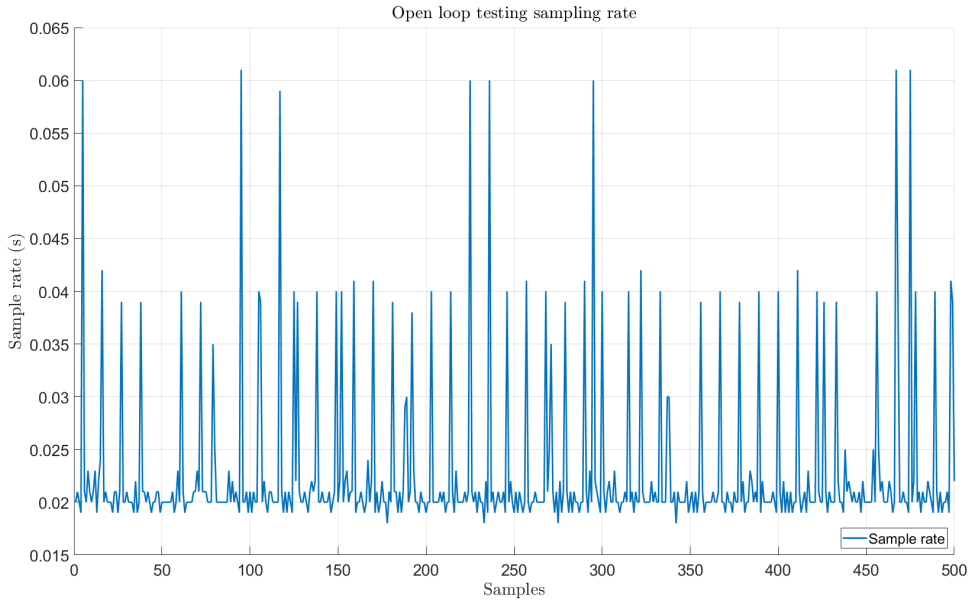

Figure 3.4: Open loop tests sampling rate without UGPS

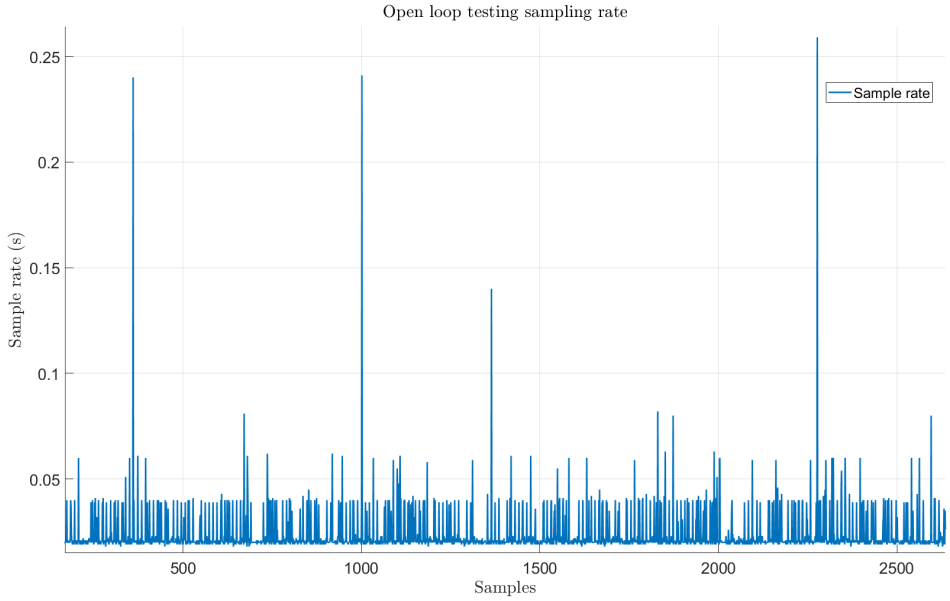

Figure 3.5: Open loop tests sampling rate with UGPS

3.3.2 Conclusion

The BlueROV2 system is easy to use out of the box and has a robust open-source community that is ready to help with any problems that come up. However, after taking a more in-depth look into the software and hardware that comes with the BlueROV2 and its limitations, we concluded that we would need to change out both

the main hardware components and write our own software so we could fulfil the control and data acquisition requirements needed for this project.

3.4 New Control Computer and Sensors

When choosing the new Single Board Computer (SBC) to interact with the sensors and run the control algorithm, it was vital that it would have General Purpose Input/Output (GPIO) capability with communication interfaces like Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI) so that it would not need external interfaces for that, as the space in the enclosure is limited. The most critical requirement was high computing power. We would need considerable computational power just to run the MPC algorithm, and not only that, but the SBC would also need to process the sensor data fusion algorithms and interact with all the hardware.

The IMU used in the original system was a part of the Pixhawk, so when it was removed, a new IMU had to be selected. We decided that in order to simplify communication with the sensors, the IMU should include an accelerometer, a gyroscope and a magnetometer. The next things to look for were high accuracy and low noise.

3.4.1 Khadas VIM3 Pro

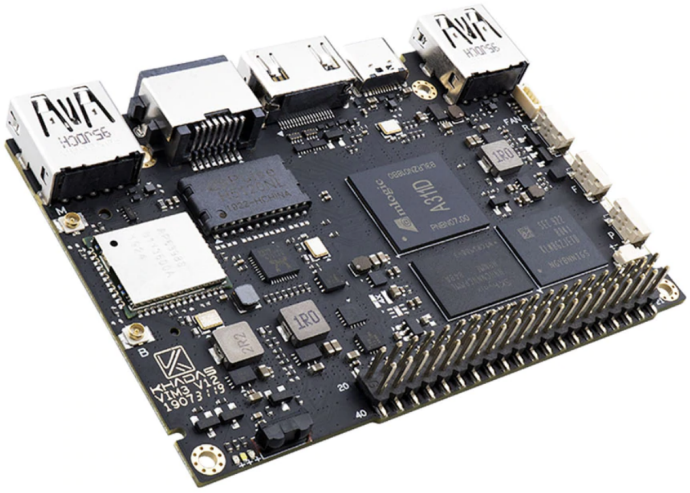

Figure 3.6: Khadas Vim3-Pro [41]

The Khadas VIM3 Pro has an Amlogic A311D processor with six cores: four 2.2Ghz Cortex-A73 cores, paired with two 1.8Ghz Cortex-A53 cores. Multiple benchmarks like Whetstone, Drystone and Linpack show that the VIM3 Pro is not only multiple times faster than the Raspberry Pi 3B+, but also the Raspberry Pi 4 [41].

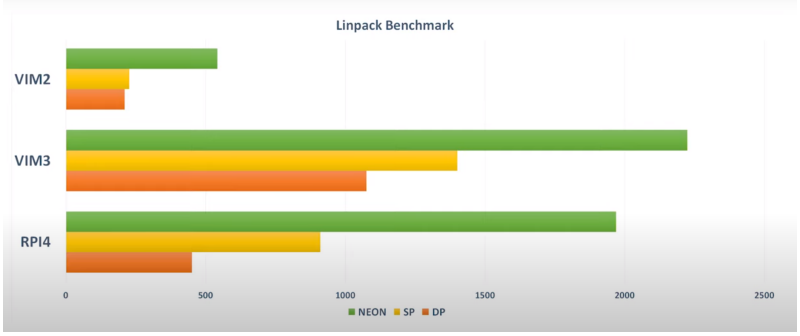

Figure 3.7: Benchmark [42]

The Vim3 Pro also has an inbuilt 5 Tera Operations Per Second (TOPS) Neural Processing Unit (NPU). The NPU can be used for image recognition using a neural network model. It opens up the possibility to expand the ROV to for integrating Artificial Intelligence (AI). More relevant to our project, the NPU could possibly be used to handle the optimization step for the MPC.

Other features of the Khadas VIM3 Pro are [41]:

- 4GB LPDDR4 Random Access Memory (RAM)
- LAN, Wi-Fi and Bluetooth connectivity
- 40-Pin Header (USB, I2C, I2S, SPDIF, UART, PWM, ADC MCU: SWIM, NRST, PA)

3.4.2 Xsens MTi-3-DK

The Xsense MTi-3 Attitude and Heading Reference System (AHRS) is an inertial measurement unit that contains an accelerometer, gyroscope and magnetometer. What makes this IMU stand out is that it already has an onboard Extended Kalman filter. The pitch, roll and yaw are calculated onboard at a 100 Hz sampling frequency. The Kalman filter also outputs free acceleration, so all the data necessary to fuse IMU and GPS data are readily available as a direct output of the IMU. This noticeably decreases load on the application processor [43].

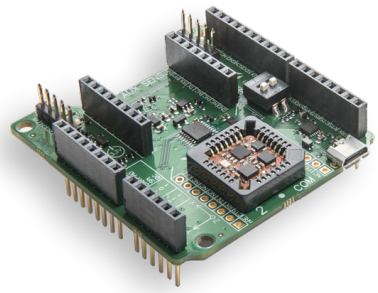

Figure 3.8: Xsense MTi-3-DK [43]

The main characteristics of the IMU are:

- 100 Hz sampling frequency
- 800 Hz data output
- 10 ° \h Gyro bias stability
- 0.5 ° precision of the pitch and roll, and 2° yaw accuracy.
- I2C/SPI/UART interfaces
- 16g full scale of acceleration
- 2000° \s full scale rate of turn

3.5 Changes to the Software Algorithm

3.5.1 Inertia Measurement Unit

The first version of MTi-3 implementation used the available API written in C++. The control loop would be written in Python because it had useful packages we needed. So we set up a process on another core, written in C++, that would sample the data and send it through an interprocess socket to the main python script. The sampled data would come in a continuous stream, so there would have to be another thread running the python code to receive the data and updating the state vector for the main loop. This would utilize the thread-safe multiprocessing arrays the same way as for the Pixhawk. The problem with this solution was that we already used two cores to sample the IMU. The code went through multiple revisions to improve the execution speed for sampling sensors and to decrease the load for

the CPU. Eventually, we decided that using the API for Universal Asynchronous Receiver/Transmitter (UART) had to be changed to using I2C. The advantages would be that the orientation data could be quickly read from within the main loop, and we would free up the two threads that were essentially handling UART communication. There was no ready-to-use API to use I2C, so we had to write our own Python library to handle the low-level communication using I2C.

3.5.2 Underwater Global Positioning System

The ROV uses the Waterlinked UGPS system for positioning. That system is based on Short BaseLine (SBL) UAPS. A locator is placed on the ROV that sends out an acoustic pulse. Four receivers are placed near the surface at a set grid where the distance between the receivers is known. The receivers actively listen for the pulse sent from the locator and use the time-of-arrival at each receiver to calculate the location of the ROV. This system is limited because it only looks at movements in the XY directions and needs an external input to send the depth of the locator to the top-side control in order to get an accurate reading. The top-side computer can then either give the position of the ROV from the base station or transform the position into a GPS coordinate. The communication between the application script and the UGPS is done using Hypertext Transfer Protocol (HTTP) requests: one request to post the depth data to the top-side unit and a second request to get the location data.

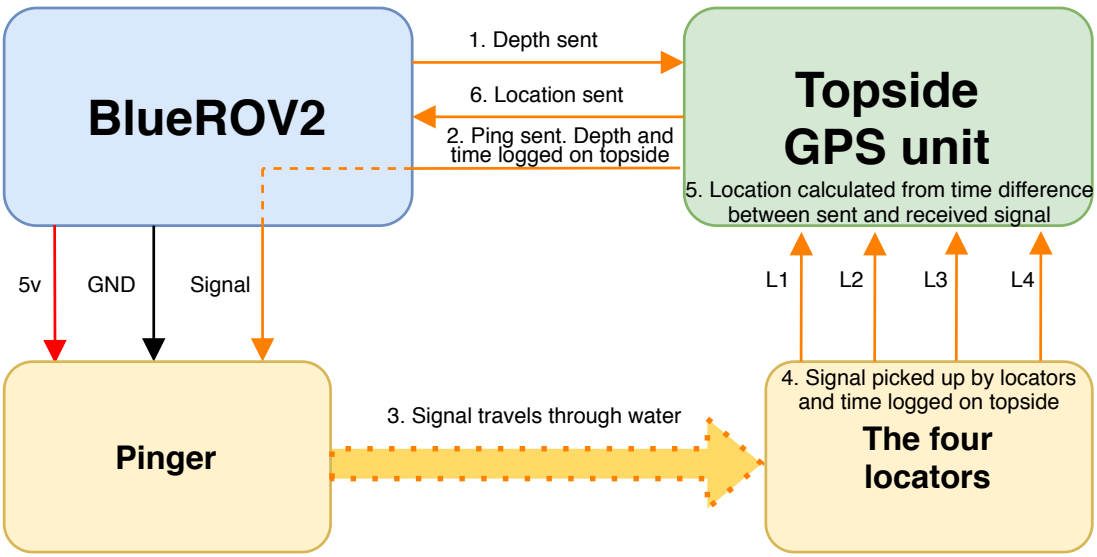

Figure 3.9: Waterlinked UGPS logic

The Pixhawk does not have an ethernet port, so it was actually the Raspberry Pi that communicated with the top-side UGPS unit, yet if we requested the location with the inbuilt MAVlink command, we would get the data from the Pixhawk. By removing the Pixhawk from the system, we made the UGPS integration more fluent.

The HTTP requests through a 50 m long cable were too slow, so unlike the IMU and barometer that used I2C, they could not be made from within the main loop. A response from the request took approximately 100ms, and the main loop could not wait for so long. This again required the solution used earlier, where another thread handles the communication and updates the state vector.

3.6 New Software Algorithm Structure

To get the best out of the new hardware and what we learned from the BlueROV2 code structure, we saw that another approach would be needed. To prevent the control code from being kept waiting for a newly updated variable, be it a new sensor reading or an updated state, we decided to run different tasks of the ROV on different processors and share their variables globally. In figure 3.10, the rough structure can be seen. Processor 1 runs data acquisition from the three sensors. The sensors are the IMU, pressure sensor and the Waterlinked UGPS. When a new value is read, it is written to a global variable that the second processor then reads and runs the sensor fusion. The sensor fusion always runs an update with a new reading from the IMU and then adds the pressure sensor and the UGPS if there is a new value there. The values are run through a Kalman filter, and the new values are sent as updated states to the optimization code. The optimization code runs at varying speeds, depending on how long it will take to get an optimal solution. The optimization code then sends that optimal control input to the control code that takes that value, double-checks it for safety reasons, mainly if the value is over the saturation limits of the thrusters, and then sends that value to the thrusters. The control loop is also in charge of logging from all four processors into a log file. A general flowchart of the four different processes running on different CPUs can be seen in figure 3.10.

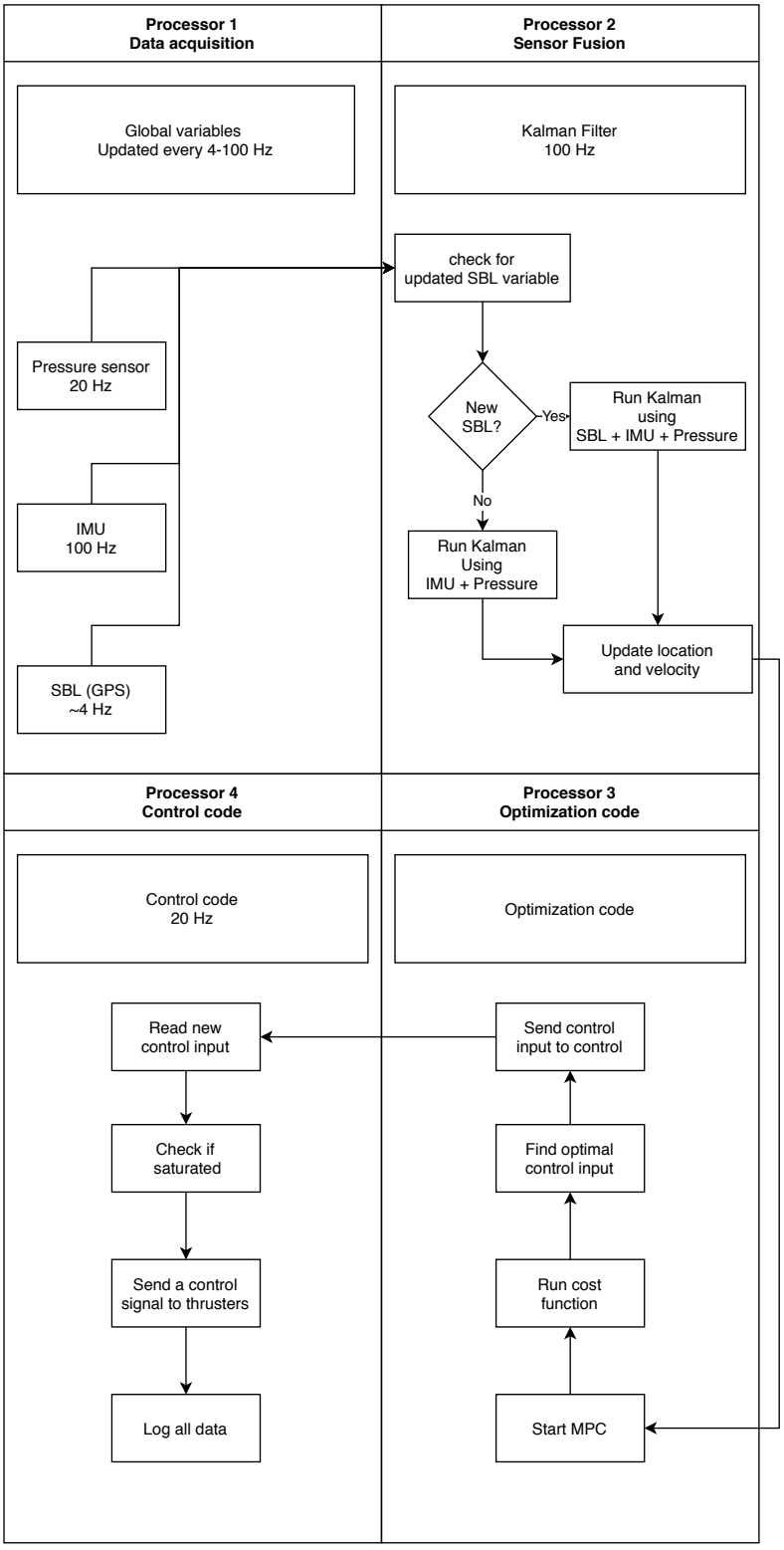

Figure 3.10: Multi-processor control code flow chart

3.7 Experimental Environments

At the start of the thesis, the primary experiment location was the Aurora School in Hjerting, Esbjerg. The school has a 25 m long and 12 m wide swimming pool that goes to a maximum depth of 1.9 m. The pool can be seen in figure 3.11. Only half of the pool could be used because the other half was too shallow, marked as the red zone in the figure. The pool was more than sufficient for the experimental needs for this thesis; however, when the Covid-19 pandemic hit in the spring of 2020, we lost access to the pool.

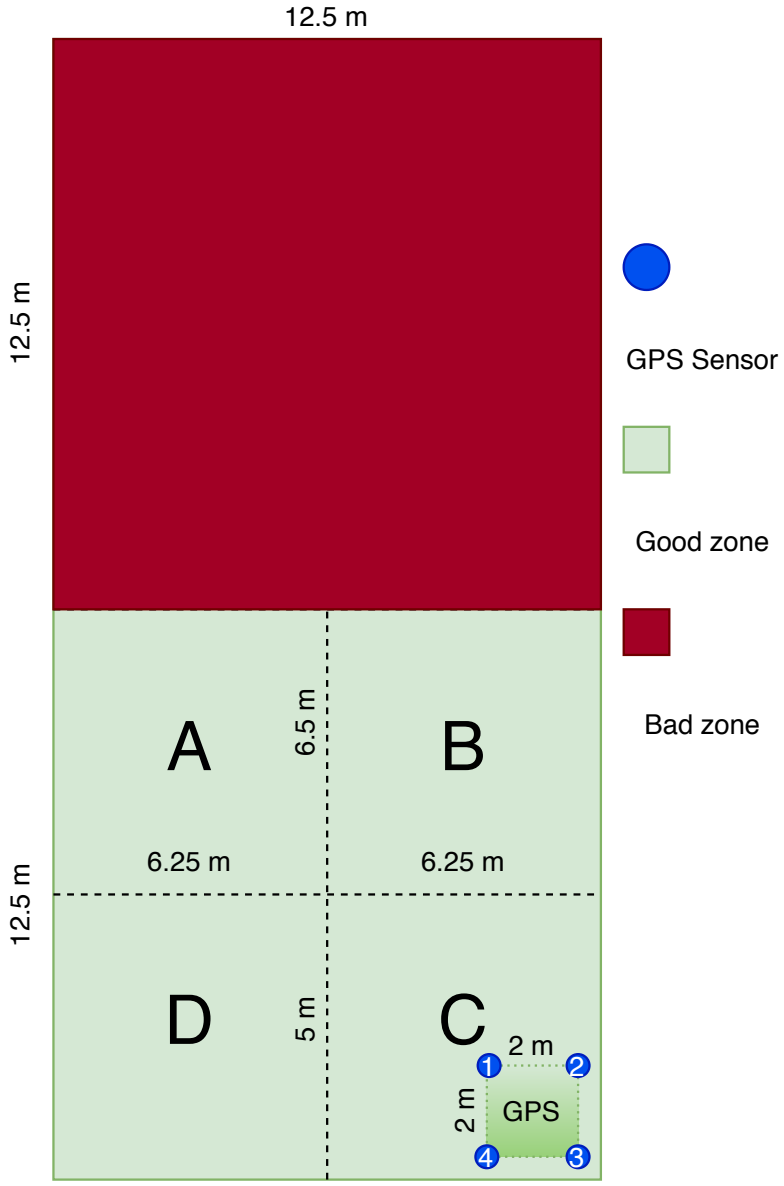

Figure 3.11: Hjerting Auraskolen test pool

To continue testing, we purchased a free-standing swimming pool and put it up in the garden. The dimensions of the pool can be seen in figure 3.12. The new

pool is 7.32 m long and 3.66 m wide. It has a maximum depth of only 1.32 m and that short depth limits the validations for heave. Other limitations of the new pool are its smaller size and that the walls of the pool are made out of soft fabric. The walls are also not straight. Since the UGPS relies on sound travelling through the water to determine locations, this can cause some more significant errors in the testing phase. Therefore, a portion along the edge of the pool is not usable for testing, marked as the red zone in the figure. The remaining model validations and controller implementations were done in the new pool.

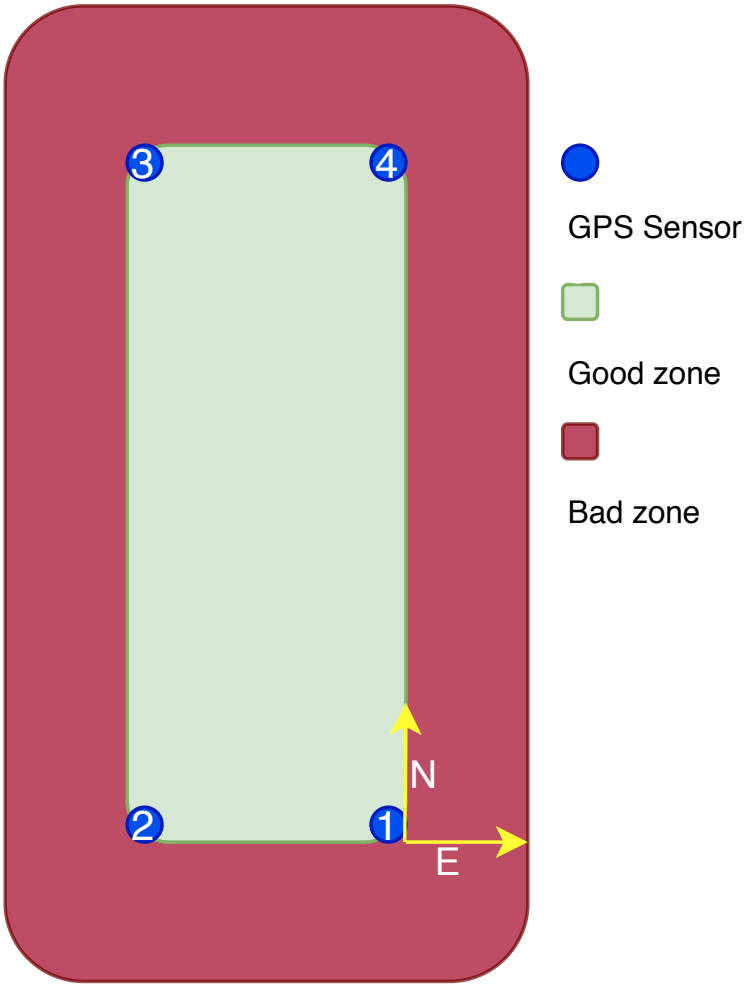

Figure 3.12: Bestway 732x366x132 cm pool

3.8 Sensor Noise

For the model to be validated, the sensors' signal to noise ratio needs to be calculated. First, we tested the Waterlinked UGPS in the Hjerting Aurora school seen in figure 3.11. The first test was letting the ROV sit in the middle of the cross-section between all the zones for 120 seconds and compare the Raw GPS values to the built-in filtered values of the Waterlinked system, seen in figure 3.13. The test showed

that since the ROV had previously been in a test closer to the origin of the SBL locators, it took almost the full 2 minutes for the filtered data to converge to the actual location. Therefore, the filtered data cannot be used for control purposes.

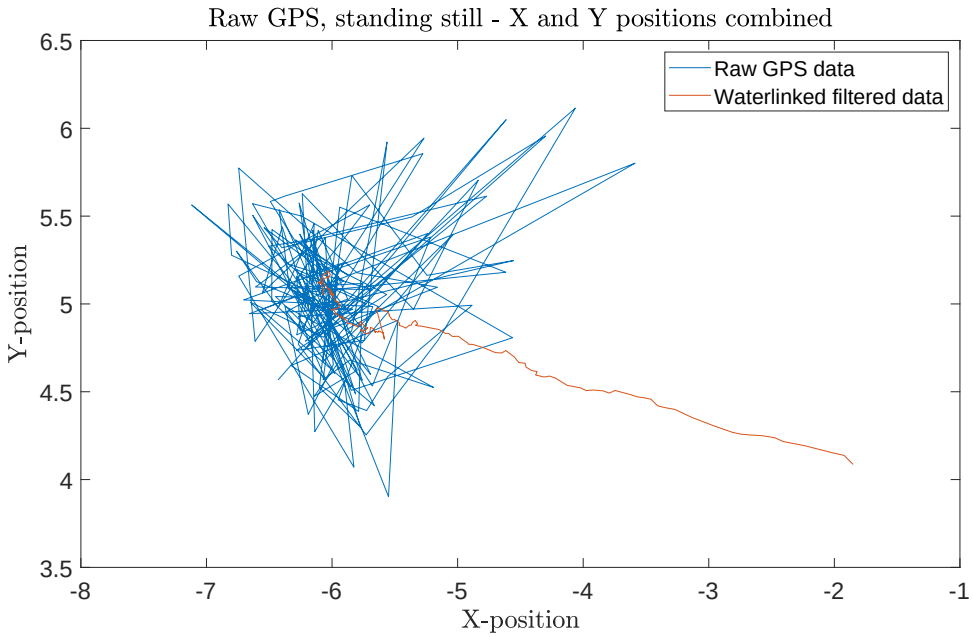

Figure 3.13: GPS raw signal

The second finding was that the system was fairly noisy and had a difference of 3 m in X and 2 m in Y, seen in figure 3.14.

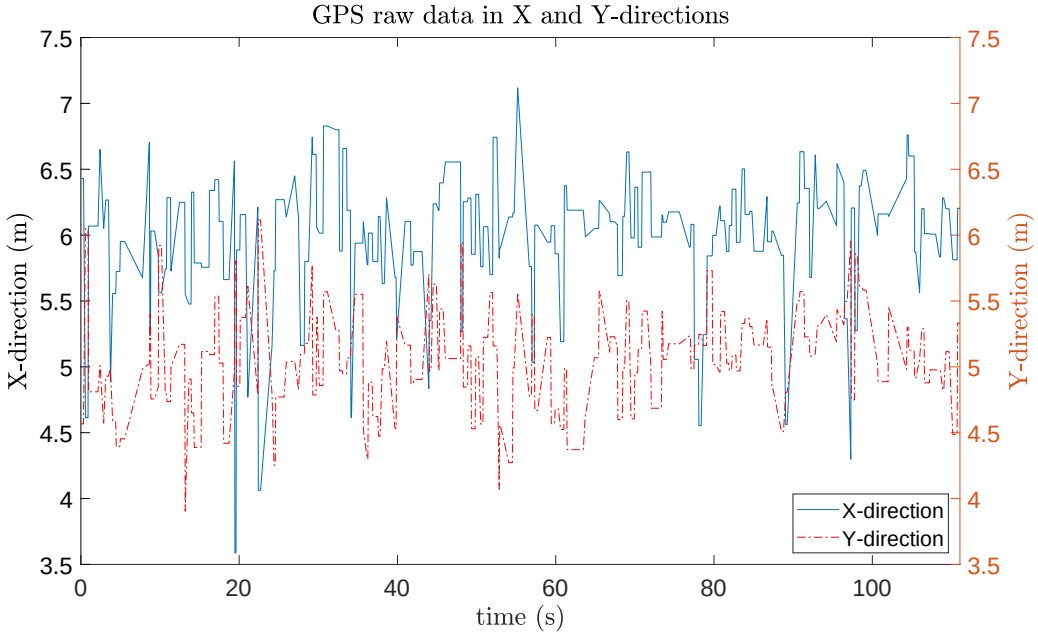

Figure 3.14: GPS X and Y signals

We assumed the signal to be a white Gaussian noise with a mean of 0 and from that, we could calculate the variance of the X and Y positioning of the signal. The results can be seen in figures 3.15 and 3.16.

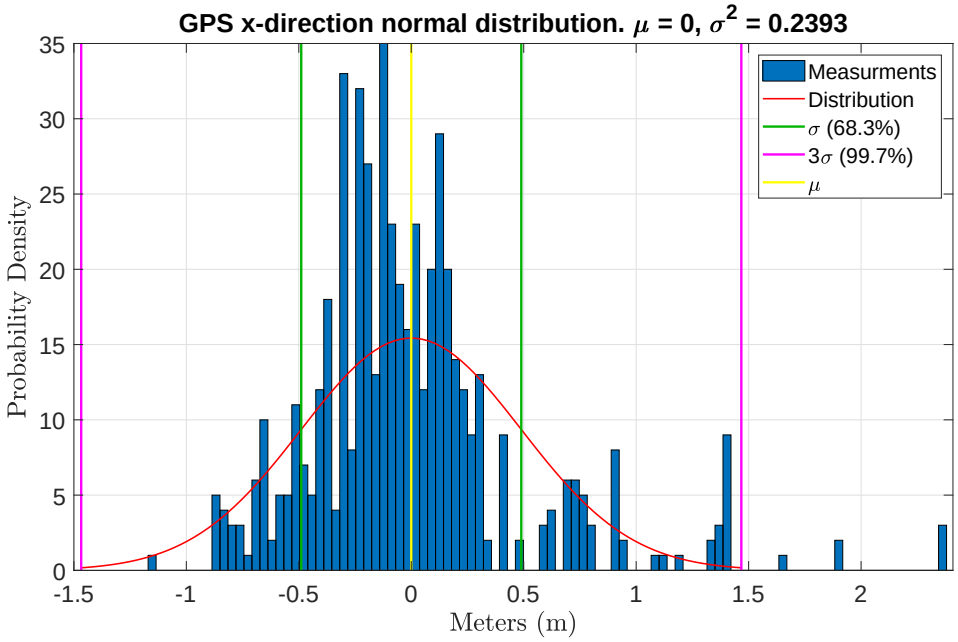

Figure 3.15: X direction variance

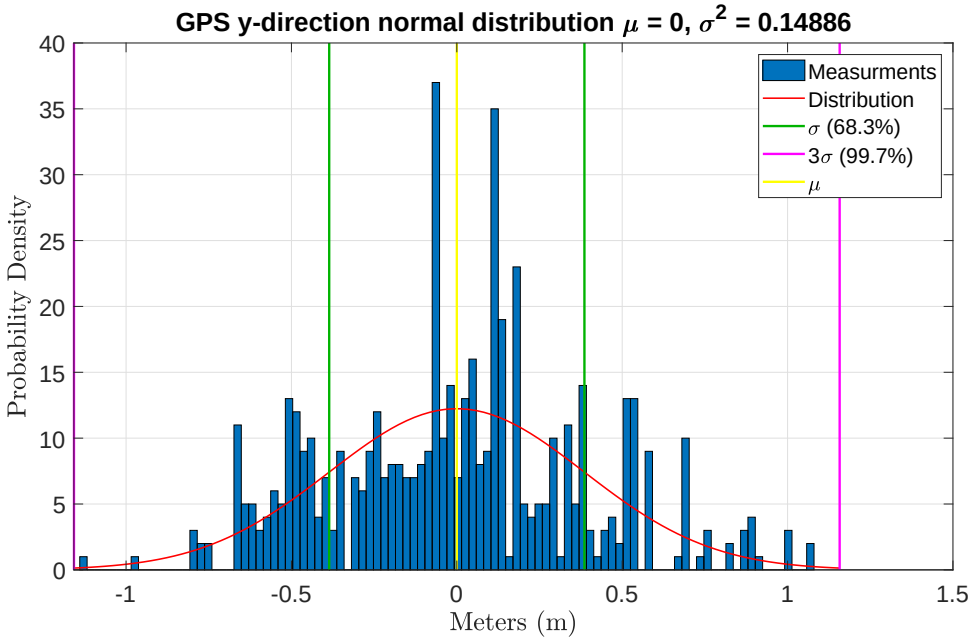

Figure 3.16: Y direction variance

We decided to contact Waterlinked to see if they had some recommendations on how to reduce the noise. They recommended updating the firmware of the system and

relocating the locator on the ROV to be facing upwards. With the new firmware, we could also add a reflective area as a built-in filter of the system and change the search area from 100 m by 100 m to whatever size we wanted. After contacting Waterlinked for assistance, they recommended having the search area the size of the test area plus 15 cm. After those changes, we re-ran the test in the new pool seen in figure 3.12 since we did not have access to the Hjerting pool anymore. The noise of the system had been reduced significantly, and the raw UGPS value can be seen in figure 3.17.

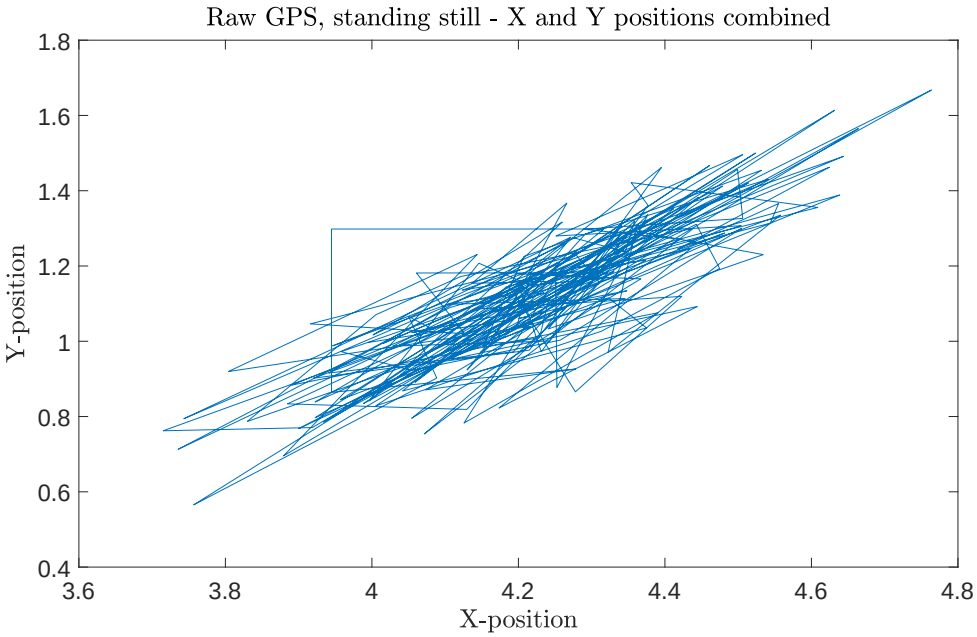

Figure 3.17: New setup UGPS raw data

The new set-up had much less noise on both X and Y directions. The noise was reduced to almost 0.5 m on both channels as seen in figure 3.18.

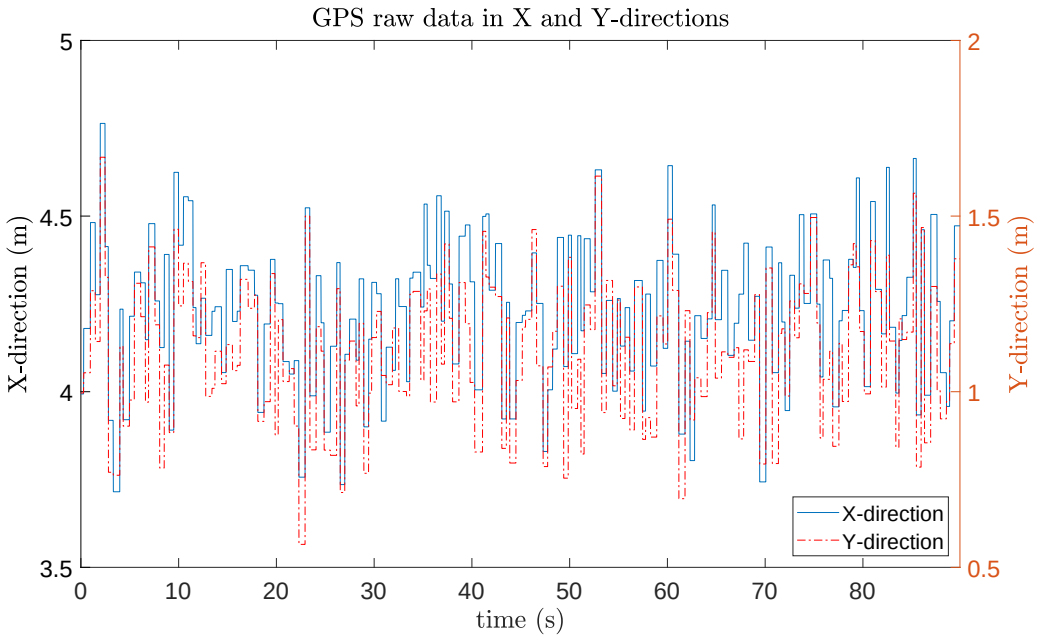

Figure 3.18: New set-up UGPS X and Y signals

The new variances for both X and Y directions can be seen in figures 3.19 and 3.20.

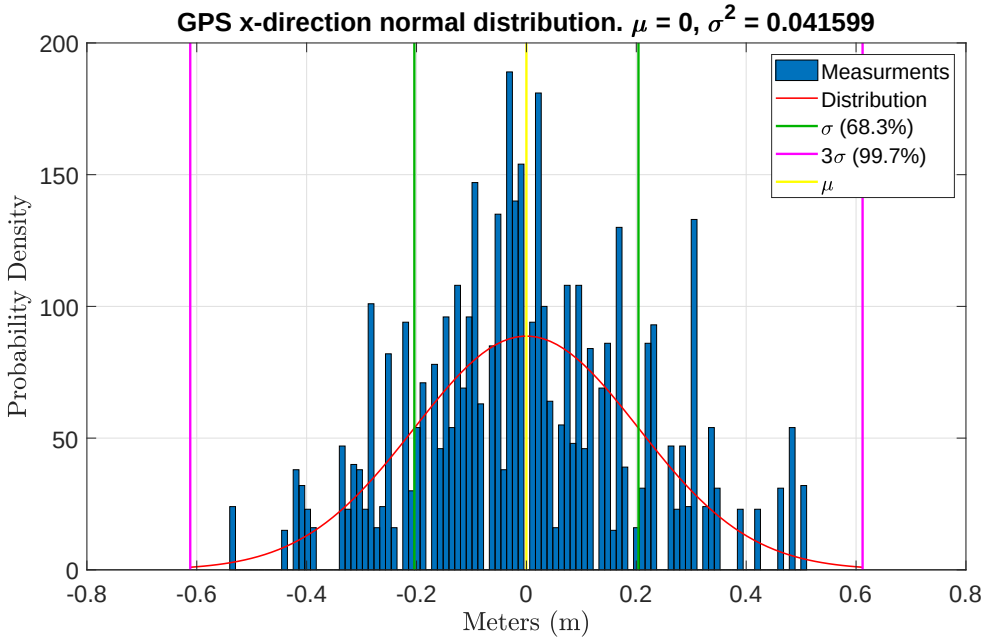

Figure 3.19: New set-up UGPS X direction variance

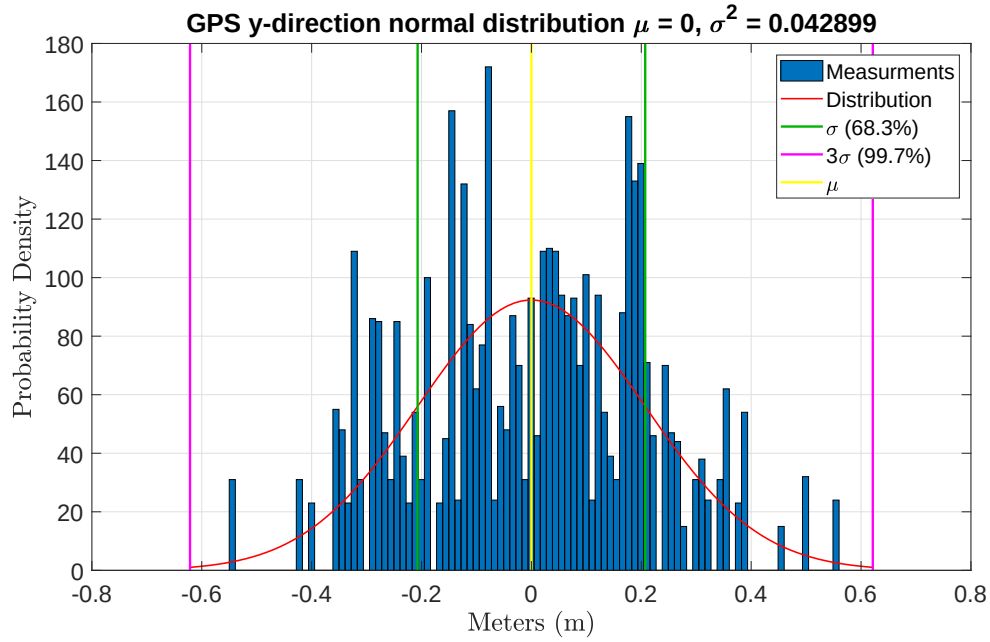

Figure 3.20: New set-up UGPS Y direction variance

The same test was done to find the noise and variance of the pressure sensor. The raw depth data can be seen in figure 3.21 and the variance in 3.22.

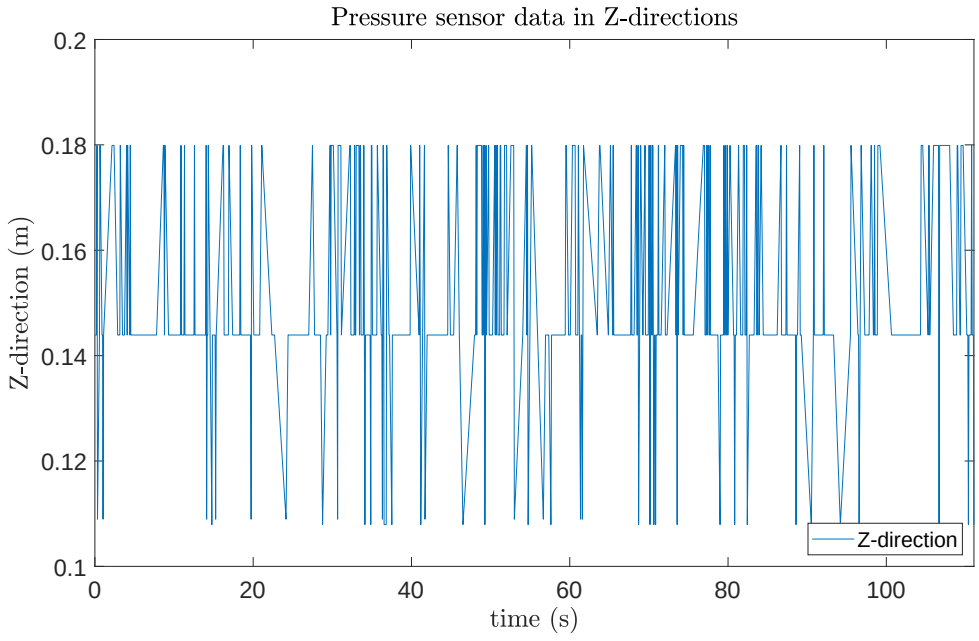

Figure 3.21: Pressure sensor raw data

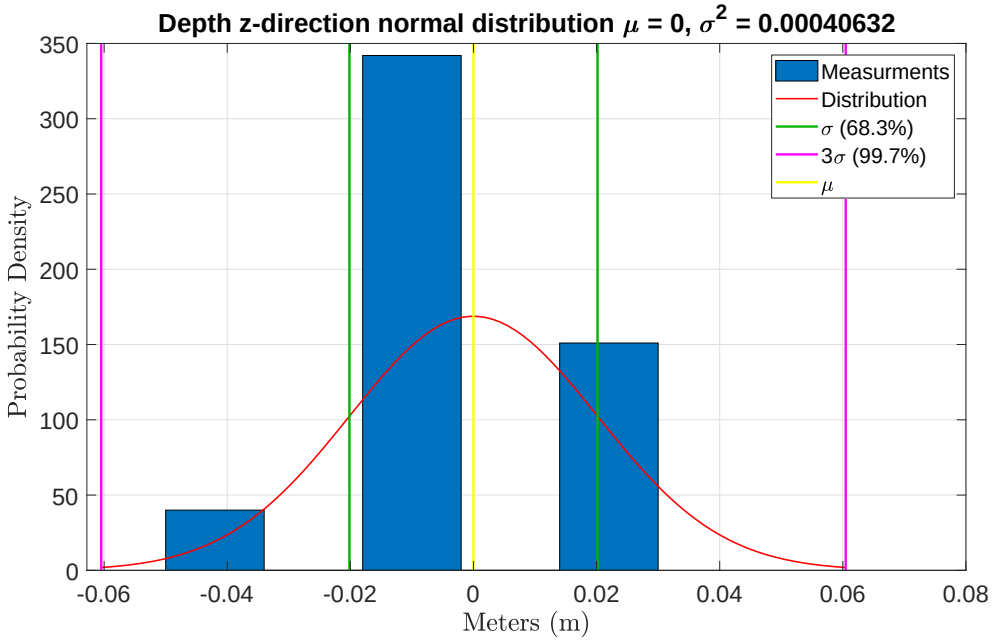

Figure 3.22: Pressure sensor variance

The model validation will also look at velocity and therefore, the variance of the velocity in X, Y and Z directions were calculated. The results can be seen in figure 3.23, 3.24 and 3.25.

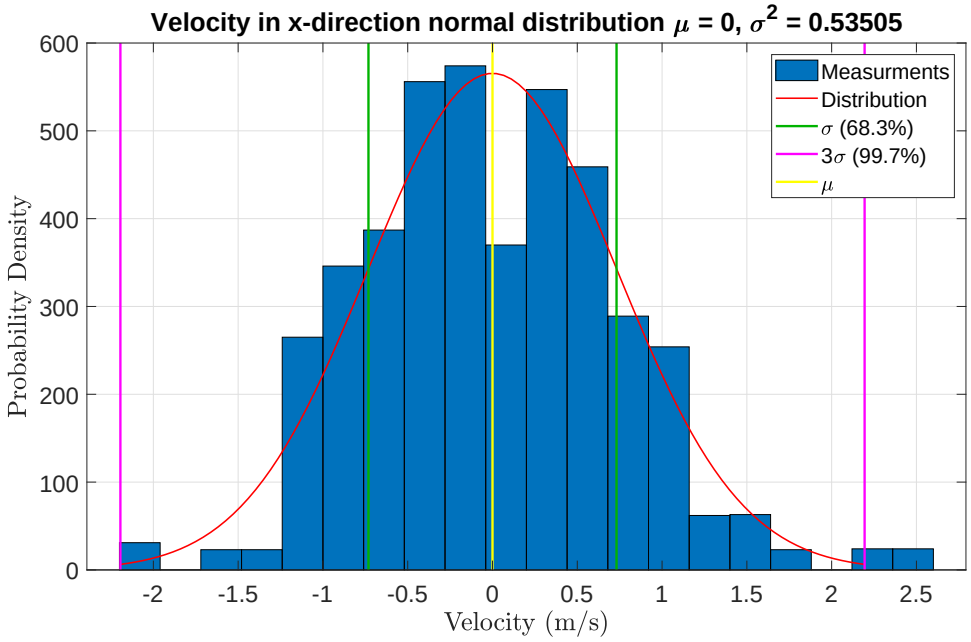

Figure 3.23: UGPS X direction velocity

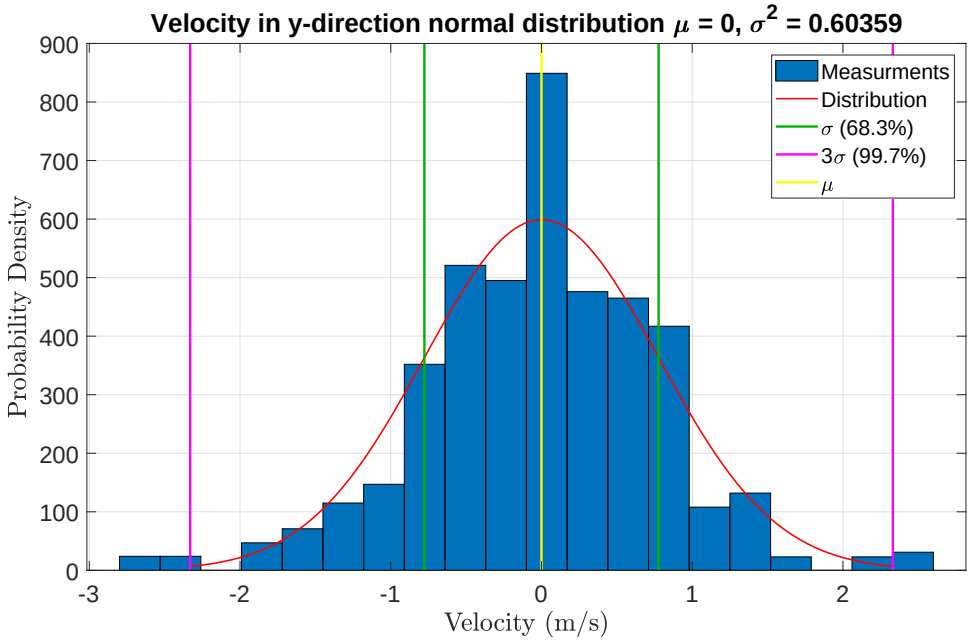

Figure 3.24: UGPS Y direction velocity

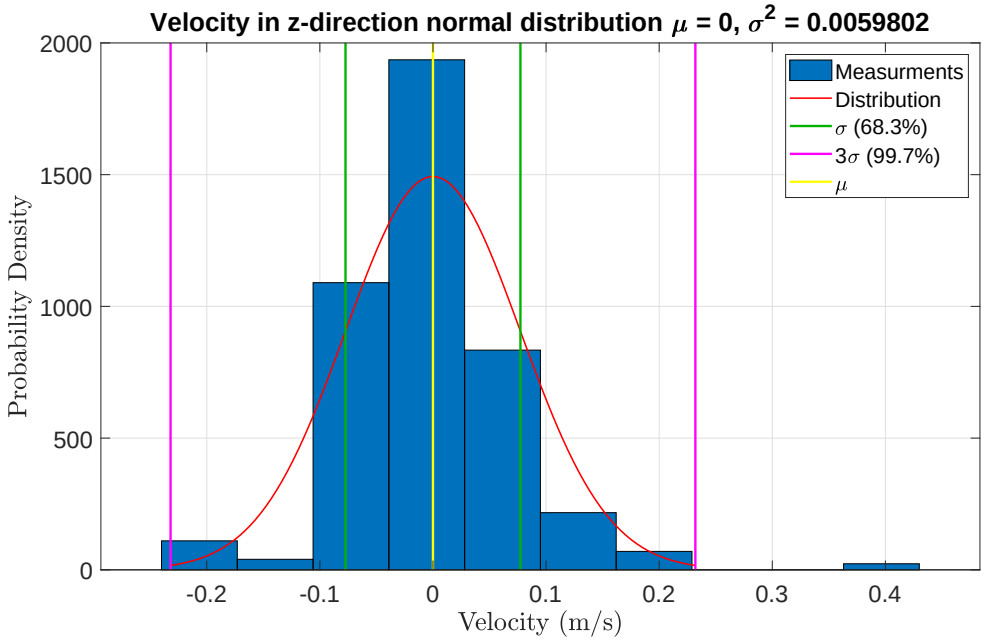

Figure 3.25: UGPS Z direction velocity

Chapter 4

Model Validation

For control systems, it is crucial to have a mathematical model that is as close to the actual dynamics of the real-world plant as possible. This section will show the real-world experiments and compare them to the mathematical model. It will also outline how to tune the model if there is a difference between the model and the plant. For this project, we looked into the model validation in the four directions we intended to control the ROV: the surge, sway, heave and yaw. In table 4.1, the different sensors used to validate different movements of the BlueROV2 experiments are listed. The sensors were also run through a simple Kalman filter where only the sensor output with the variance calculated in chapter 3.8 was put through to validate the model. Detailed information about the Kalman filter set-up can be seen in chapter 5.3.3.

Table 4.1: The different sensors used in the experiment for all movements

Movements	Sensor one	Sensor two
Surge	UGPS	IMU
Sway	UGPS	IMU
Heave	Barometer	IMU
Pitch	IMU	-
Roll	IMU	-
Yaw	IMU	-

With the updated hardware and firmware, the new system had a much more stable sampling rate when running open-loop tests, even with the UGPS signal that had caused significant delays for the old system. The new sampling rate, including new UGPS readings, can be seen in figure 4.1. The system was set to run every 0.01s but the real run time was around 0.013-0.014s. The main reason for that delay is the time it takes to write the log into a file. The later log files were smaller, so the run-time was quicker, but the system could also be set to run at a slower pace to give more time for logging all the data if needed.

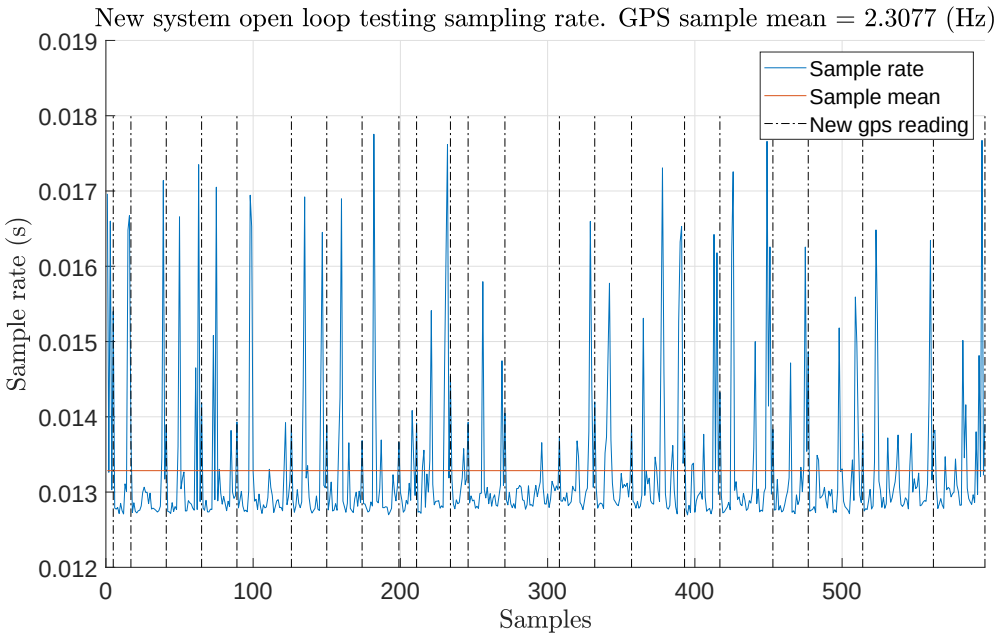

Figure 4.1: New system open-loop sampling rate with UGPS

A comparison of the BlueROV2 hardware/software run time and the new hardware/software can be seen in table 4.2. It shows that the big delay that comes with the old system waiting for a new UGPS and the inconsistent run time is almost completely removed.

Table 4.2: Old and new system open-loop comparison

Hardware/software	BlueROV	BlueROV + GPS	New + GPS
Set sample rate	0.02	0.02	0.01
Actual sample rate	0.025	0.03	0.013
Longest delay	0.06	0.26	0.018
% of delay 2x sample rate	15%	25 %	0%

4.1 Surge Experiments

The first tests of the system were the surge experiments. This section will detail the results and calculations done to validate them, and then bring the results together in a table at the end. For sway, heave and yaw only the final results in the table will be shown.

To validate the surge model, the ROV is run at different thrust inputs, and the steady-state velocity (v_{ss}) is found and compared to the model. Since the accelerometer only outputs the acceleration, it must be integrated to find the velocity. The Xsens Mti-3-DK can output the free accelerations where the gravitational force has been removed. An example of the acceleration outputted can be seen in figure 4.2. The figure also shows how the vibration of the thrusters increases the sensor noise.

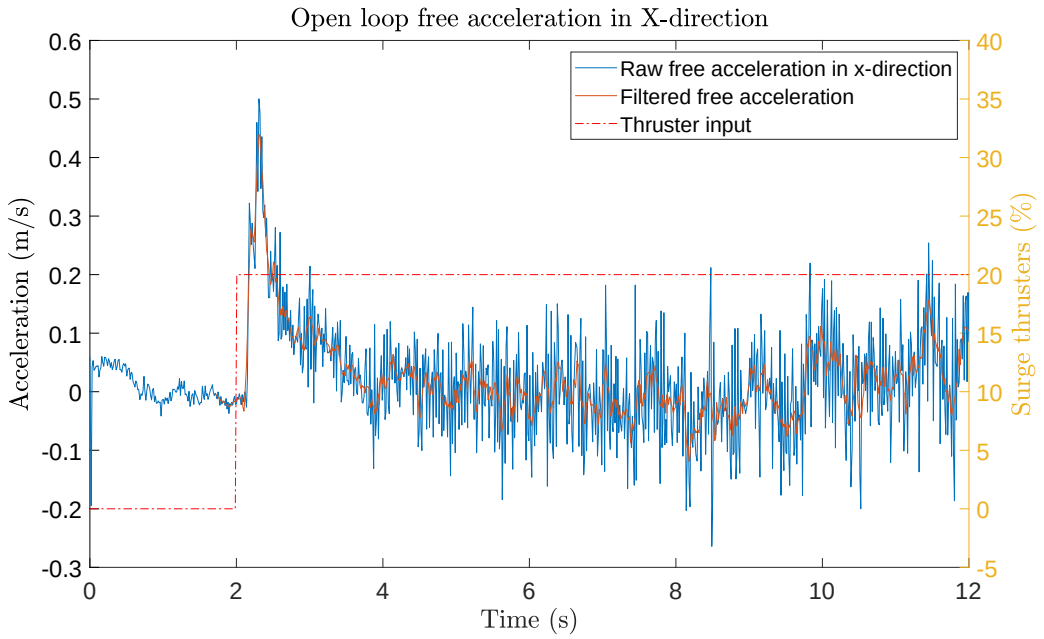

Figure 4.2: Free acceleration in x-direction at 20% thrust

By integrating the free acceleration, the terminal velocity can be found; the results for 20% and 60% can be seen in figures 4.3 and 4.4. It is also essential to look at the acceleration in the Y direction (sway) in the tests. The reason for this is that when the ROV starts going, the tether pulls on the ROV and makes it drift slightly to the right, in our case. The speed is slow compared to the X direction (surge) and can therefore be ignored. If the turn would be more significant, then some coupling terms between surge and sway would need to be added in the model.

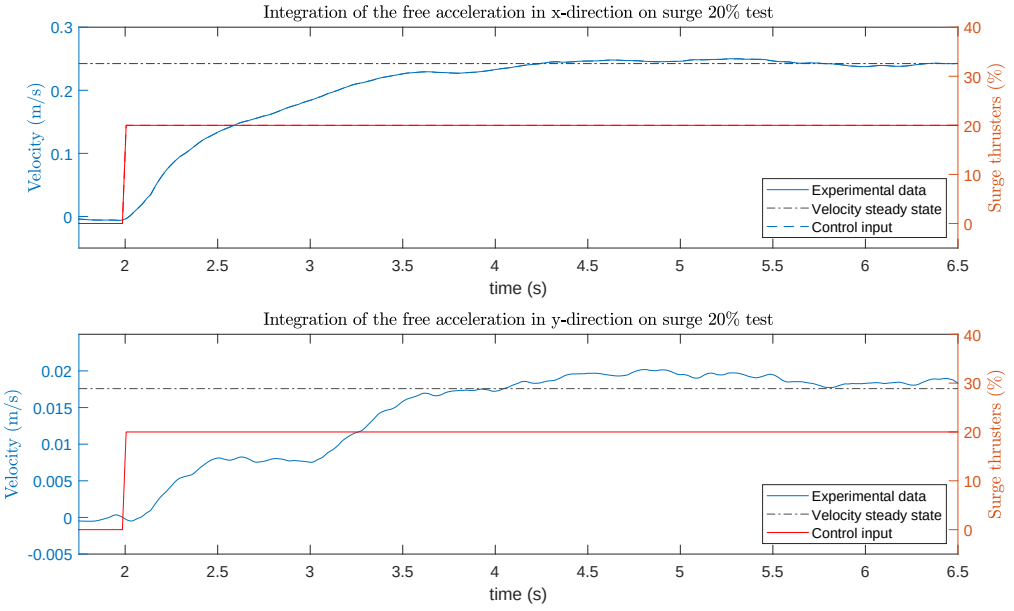

Figure 4.3: Surge velocity in X and Y direction at 20% thrust

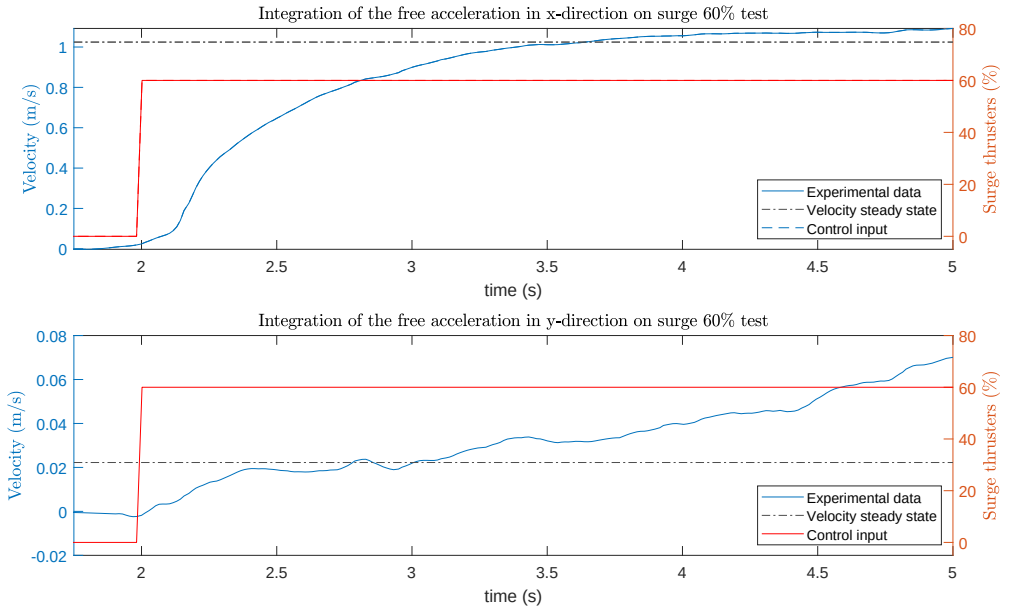

Figure 4.4: Surge velocity in X and Y direction at 60% thrust

The next step is to input the same thrust to the non-linear model and compare it to the experimental data. In figures 4.5 and 4.6 the non linear model and the experimental v_{ss} are compared.

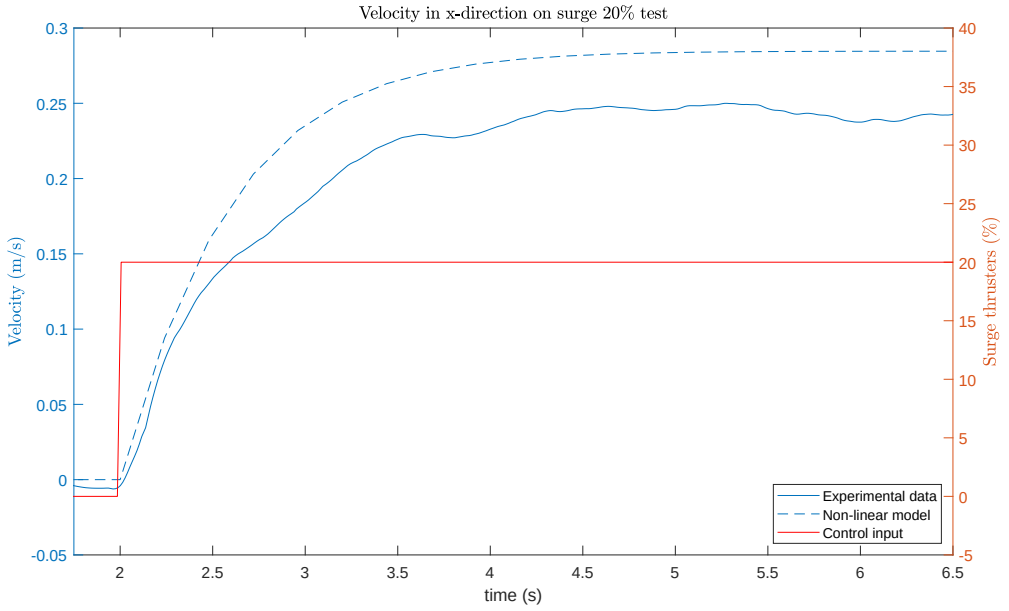

Figure 4.5: Non-linear model and experiment velocities at 20% thrust

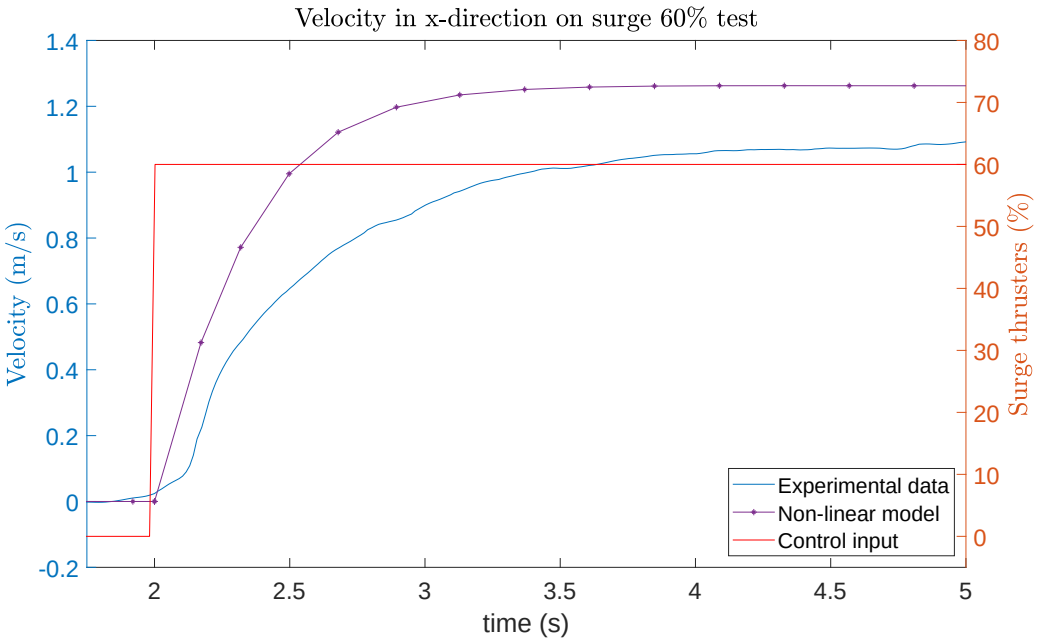

Figure 4.6: Non-linear model and experiment velocities at 60% thrust

Comparisons of the non-linear model and experimental data for 20%, 40%, 60% and 80% thrust can be seen in figure 4.7. There is a significant difference between the model and the experiments, and therefore some tuning of the model is needed. Tuning of the model is explained in section 4.7.

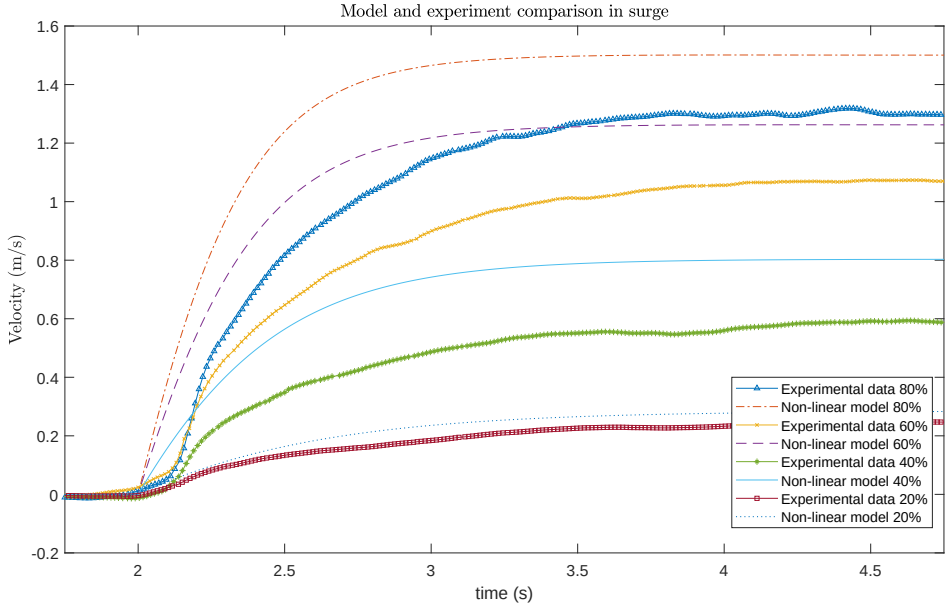

Figure 4.7: Non-linear model and experiment velocities from 20% thrust to 80%

The experimental and the non-linear model v_{ss} can be seen in table 4.3. The surge at 100% is not included because of limitations in the test facility.

Table 4.3: Steady-state surge velocities (m/s)

Thruster %	**Experiment** v_{ss}	**Model** v_{ss}
20	0.25 ± 0.05	0.28
40	0.55 ± 0.11	0.78
60	1.07 ± 0.12	1.26
80	1.30 ± 0.18	1.50
100	-	1.6

4.2 Sway Experiments

The ROV was not expected to use the sway as a control feature, but the motions were still validated in case there would be a change in control strategy later on with the thesis or future work. The method is the same as in the surge experiment: integrate the free acceleration and find the velocity steady-state velocity v_{ss} and compare the findings to the non-linear model. The tether seemed to have more influence on the ROV in sway compared to surge. However, by holding the tether out of the water and moving it with the ROV, the disturbance of the tether was minimal.

Table 4.4: Steady-state sway velocities (m/s)

Thruster %	**Experiment** v_{ss}	**Model** v_{ss}
20	0.18 ± 0.40	0.19
40	0.48 ± 0.42	0.44
60	0.68 ± 0.30	0.65
80	0.72 ± 0.32	0.76
100	-	0.8

4.3 Heave Experiments

By differentiating the positioning data given by the pressure sensor, the upwards and downwards velocities can be calculated. The steady-state velocity v_{ss} of the ROV is obtained when the ROV has reached a constant velocity. The results can be seen in table 4.5 and heave at 40% velocity can be seen in figure 4.8.

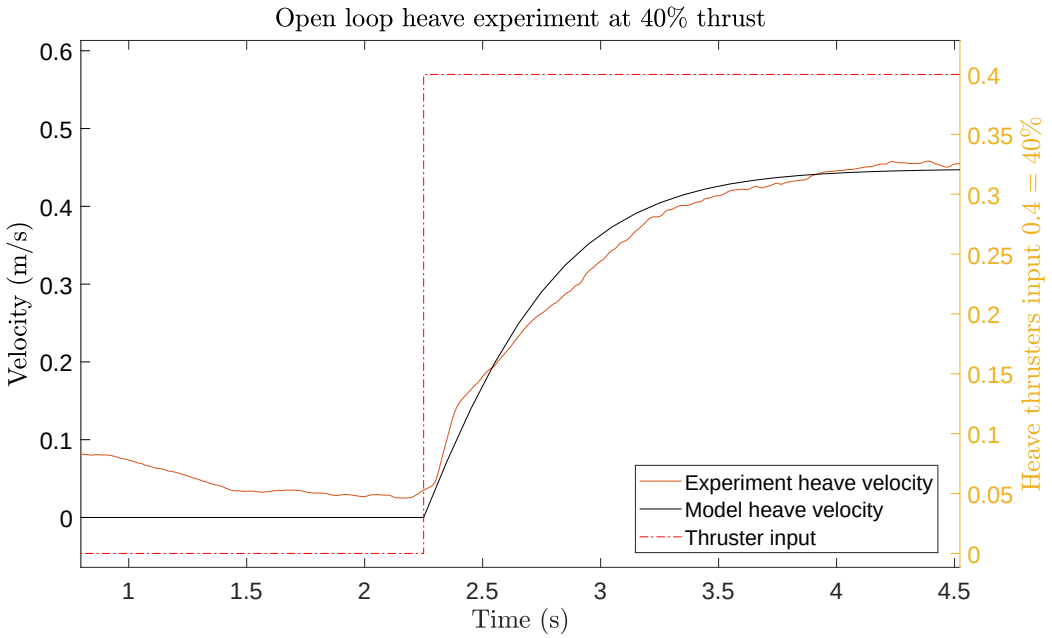

Figure 4.8: Non-linear model and experiment velocities for 40% thrust

Table 4.5: Steady-state heave velocities (m/s)

Thruster %	Experiment v_{ss}	Model v_{ss}
20	0.17 ± 0.02	0.16
40	0.42 ± 0.07	0.44
60	0.58 ± 0.06	0.55
80	0.70 ± 0.05	0.65
100	0.80 ± 0.02	0.71

4.4 Yaw Experiments

By differentiating the gyroscope data, the turning velocity in yaw can be calculated. The steady-state velocity v_{ss} of the ROV is obtained when the ROV has reached a constant velocity. The results can be seen in figure 4.6. The tether became wrapped up quickly and started to interfere with the results, but since the ROV hit the v_{ss} quickly, the interference can be ignored.

4.5 Roll and Pitch Experiments

The pitch cannot be controlled by the ROV, and therefore there is no validation for it. The current thruster set-up on the BlueROV2 and how we plan to implement the controller logic for the ROV will not address roll control. Therefore, roll is left out of the model validation.

Table 4.6: Steady-state yaw velocities (rad/s)

Thruster %	Experiment v_{ss}	Model v_{ss}
20	0.44 ± 0.05	0.60
40	1.66 ± 0.10	2.07
60	2.98 ± 0.15	3.66
80	3.95 ± 0.30	4.57
100	4.00 ± 0.20	4.93

4.6 Model Parameters and Coefficients

In table 4.12 some of the ROV's parameters and coefficients have been gathered both from [44] and new experimental ones.

Table 4.7: Estimated coefficients

Symbol	Value	Unit
F^u_{thrust}	$-392.7 \cdot u^9/109.2 \cdot u^7/ - 113 \cdot u^5/492.6 \cdot u^3/25 \cdot u$	N
F^v_{thrust}	$-392.7 \cdot v^9/109.2 \cdot v^7/ - 113 \cdot v^5/492.6 \cdot v^3/25 \cdot v$	N
F^w_{thrust}	$-280.5 \cdot w^9/779.8 \cdot w^7/ - 808.2 \cdot w^5/351.9 \cdot w^3/17.87 \cdot w$	N
F^p_{thrust}	$-33.7 \cdot p^9/93.6 \cdot p^7/ - 97 \cdot p^5/42.2 \cdot p^3/2.1 \cdot p$	N
F^q_{thrust}	$-280.5 \cdot q^9/779.8 \cdot q^7/ - 808.2 \cdot q^5/351.9 \cdot q^3/17.87 \cdot q$	N
F^r_{thrust}	$-105.9 \cdot r^9/294.5 \cdot r^7/ - 305.2 \cdot r^5/129.2.6 \cdot r^3/6.7 \cdot r$	N
τ^u_{drag}	$17.77 \cdot v^2/25.15 \cdot v$	Nm
τ^v_{drag}	$125.9 \cdot v^2/7.364 \cdot v$	Nm
τ^w_{drag}	$72.36 \cdot v^2/17.955 \cdot v$	Nm
τ^p_{drag}	$0.1246 \cdot v^2/1.888 \cdot v$	Nm
τ^q_{drag}	$0.1246 \cdot v^2/0.761 \cdot v$	Nm
τ^r_{drag}	$0.1857 \cdot v^2/3.744 \cdot v$	Nm
m_{rov}	11.4	kg
I_x	0.21	kgm^2
I_y	0.245	kgm^2
I_z	0.245	kgm^2
z_g	0.02	m
ρ	1000	kg/m^3
Volume	0.0114	m^3
g	9.82	kg/m^2

4.7 Model Tuning Parameters

The system's mathematical model was validated by experimental data; see figure 4.7 for an example of the velocity for surge.

The velocity behaviour of the system acts as a first-order linear system, and therefore the time response of the system consists of the transient and steady-state responses.

From figure 4.7, it can be seen that the steady-state velocity characteristics and the transient performance are close to the experimental data, but some tuning is needed. The transient behaviour of the system can be described by the time constant Tau. Where [45],

$$Tau = t_{(1-\frac{1}{e})v_{ss}} \tag{4.1}$$

In paper [45], tuning parameters, α and β, are introduced to correct the non-linear model to the experimental results. The tuning parameters are used as seen in equation (4.2) [45].

$$\dot{u} = \beta_u(\alpha_u \mathbf{F}_u - \mathbf{D}^u(\boldsymbol{\nu}_u)\boldsymbol{\nu}_u) \tag{4.2a}$$
$$\dot{v} = \beta_v(\alpha_v \mathbf{F}_v - \mathbf{D}^v(\boldsymbol{\nu}_v)\boldsymbol{\nu}_v) \tag{4.2b}$$
$$\dot{w} = \beta_w(\alpha_w \mathbf{F}_w - \mathbf{D}^w(\boldsymbol{\nu}_w)\boldsymbol{\nu}_w) \tag{4.2c}$$
$$\dot{r} = \beta_r(\alpha_r \mathbf{F}_r - \mathbf{D}^r(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r) \tag{4.2d}$$

The steady-state error and Tau can be modified by tuning the α coefficient. However, the β only scales the total expression and therefore only influences the Tau. The paper introduced the tuning parameters as follows [45].

- Tune α for the desired steady-state value (Changes both Steady-State and Tau)
- Tune β for the desired Tau (Changes only Tau)

The model tuning can be seen in tables 4.8, 4.9, 4.10 and 4.11. The tuning mostly helps with the Tau since the parameter estimations were all done using steady-state values. A comparison for tuning the surge at 60% thruster input can be seen in figures 4.9 and 4.10.

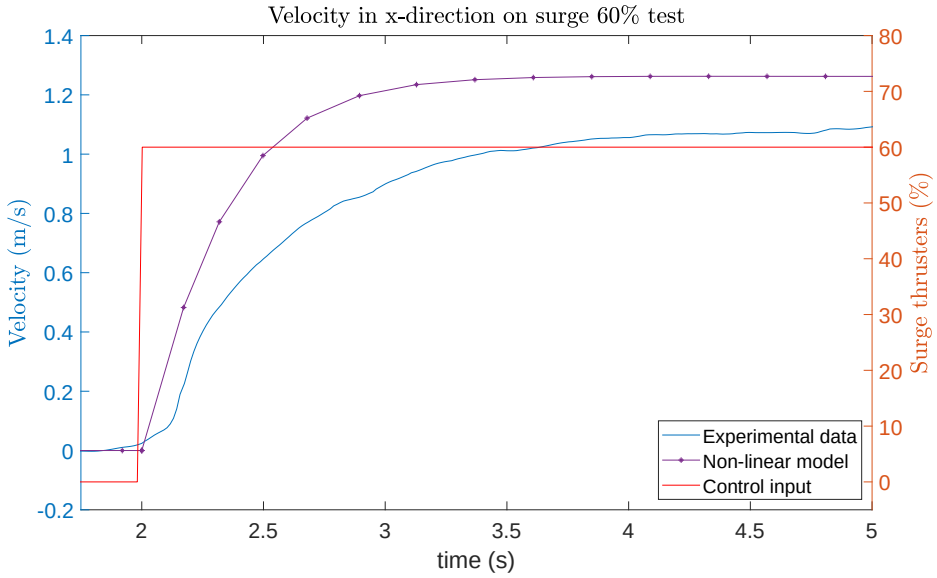

Figure 4.9: Non-linear model comparison to surge 60% experimental data

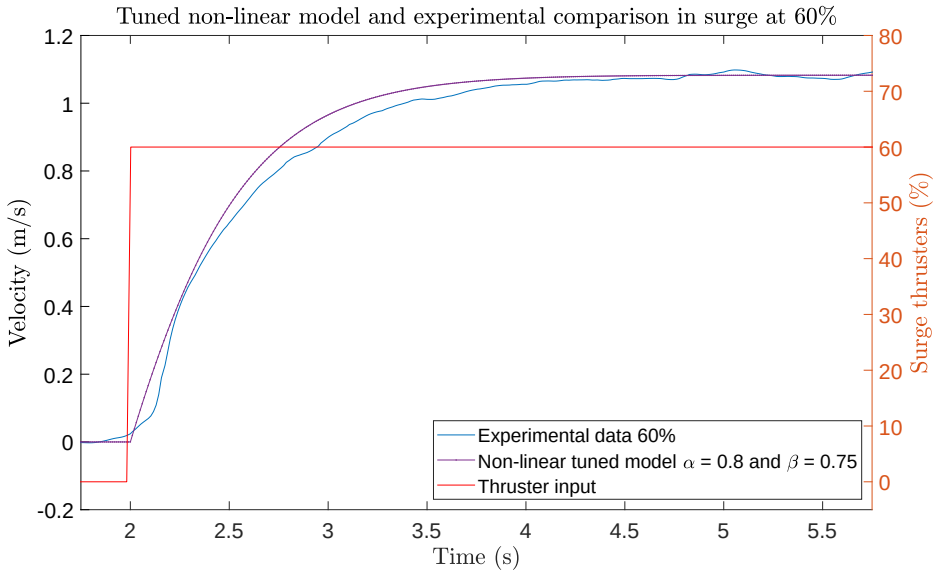

Figure 4.10: Non-linear tuned model comparison to surge 60% experimental data

Table 4.8: Surge (v_{ss} [m/s], Tau [s])

Thrust %	Non-Linear		Scaled NL		Exp	
	v_{ss}	Tau	v_{ss}	Tau	v_{ss}	Tau
20	0.28	0.59	0.25	0.61	0.25 ± 0.05	0.60 ± 0.19
40	0.78	0.42	0.53	0.56	0.55 ± 0.11	0.58 ± 0.18
60	1.26	0.34	1.08	0.50	1.07 ± 0.12	0.55 ± 0.25
80	1.50	0.32	1.32	0.52	1.30 ± 0.18	0.56 ± 0.22
100	1.60	0.29	1.38	0.43	-	-

Table 4.9: Sway (v_{ss} [m/s], Tau [s])

Thrust %	Non-Linear		Scaled NL		Exp	
	v_{ss}	Tau	v_{ss}	Tau	v_{ss}	Tau
20	0.19	0.83	-	-	0.18 ± 0.40	0.81 ± 0.20
40	0.44	0.49	-	-	0.48 ± 0.42	0.40 ± 0.15
60	0.65	0.37	-	-	0.68 ± 0.30	0.35 ± 0.08
80	0.76	0.33	-	-	0.72 ± 0.31	0.32 ± 0.12
100	0.80	0.32	-	-	-	-

Table 4.10: Heave (v_{ss} [m/s], Tau [s])

Thrust %	Non-Linear		Scaled NL		Exp	
	v_{ss}	Tau	v_{ss}	Tau	v_{ss}	Tau
20	0.16	2.65	0.18	2.78	0.17 ± 0.02	2.87 ± 0.05
40	0.44	2.15	0.42	2.23	0.42 ± 0.07	2.33 ± 0.02
60	0.55	1.23	0.60	1.45	0.58 ± 0.06	1.54 ± 0.11
80	0.65	1.08	0.70	1.12	0.70 ± 0.05	1.18 ± 0.22
100	0.71	1.01	0.75	1.08	0.80 ± 0.02	1.18 ± 0.20

Table 4.11: Yaw (v_{ss} [rad/s], Tau [s])

Thrust %	Non-Linear		Scaled NL		Exp	
	v_{ss}	Tau	v_{ss}	Tau	v_{ss}	Tau
20	0.60	0.09	0.48	0.12	0.44 ± 0.05	0.12 ± 0.04
40	2.07	0.08	1.68	0.12	1.66 ± 0.10	0.12 ± 0.03
60	3.66	0.08	3.01	0.11	2.98 ± 0.15	0.12 ± 0.04
80	4.57	0.07	3.80	0.10	3.95 ± 0.30	0.12 ± 0.02
100	4.93	0.07	4.08	0.10	4.00 ± 0.20	0.11 ± 0.02

The chosen tuning parameters can be found in table 4.12.

Table 4.12: Estimated coefficients

Motion	β	α
Surge	0.8	0.75
Sway	1.0	1.0
Heave	0.9	0.9
Roll	-	-
Pitch	-	-
Yaw	0.75	0.8

The β and α values needed to be tuned for surge, heave and yaw. Sway did not need any tuning since the non-linear model and the experimental data were close enough. Minimal tuning was needed for heave, but more significant tuning was needed for surge and yaw. There are multiple possible reasons for the differences between the modelling and the experimental data. Modelling the drag coefficients for the ROV can cause errors if not done correctly. Thruster data was collected at steady-state and as mentioned before in section 2.4.2, there could be many reasons for the thrusters not being modelled correctly. When concluding the tests, the sensors used for validating surge are likely to be further away from real life, and the tether influenced the yaw quickly when spinning in circles. With these new tuning parameters, the non-linear model is as close to the experimental data as possible and will be used for the controller development.

Chapter 5

Underwater Navigation

For autonomous underwater navigation, it is needed to know the position, velocity and attitude of the ROV in its reference frame. One of the main problems is knowing the vehicle's true position. A sensor-based method can then be used. As mentioned before, GPS does not work underwater and therefore, the ROV needs to rely on different sets of sensors. Some submarines are equipped with an Inertia Navigation System (INS) and Doppler Velocity Logger (DVL). INS is based on large mechanical gyroscopes and accelerometers and can be combined with either an DVL or and UAPS. These solutions can be both too big for smaller ROVs and cost-prohibitive. The cheaper MEMS IMU and UGPS sensors are, by comparison, more prone to disturbances and drift.

The mathematical model of the system can be used for positioning. This method relies on a perfect model and measurements of the forces generated. The position could then be obtained using differential equations of motion, knowing the initial states. As discussed earlier in chapter 2.3, acquiring the perfect model is quite difficult. A model-based navigation can be combined with the sensor-based method to improve the accuracy of the position estimate and compensate for noisy and low-frequency measurements.

This chapter will go into sensor and model-based control solutions.

5.1 Closed-Loop Control

Most common control for an ROV uses a decoupled PID control for attitude and heading. This set-up also uses a human-in-the-loop to monitor the system and take over the controls if needed. When an ROV pilot is controlling manually in a closed-loop control, the system does not need to be as strict. A rough heading, depth and visual feedback are often more than sufficient for a pilot to complete the required task. However, when the system does not have a human-in-the-loop, this approach is not sufficient, and more advanced and accurate control methods must be used.

5.1.1 Proportional-Integral-Derivative

The PID controller is one of the most commonly used in the industry. The main advantage is the simplicity of implementing it; the system can be a complete black box model, yet with minimal knowledge of the controller and manual tuning, it is possible to find the gains for decent controller performance. In figure 5.1, a block diagram of a PID controller in a feedback loop can be seen. The reference r(t) is the desired process value or setpoint, and y(t) is the measured process output [46].

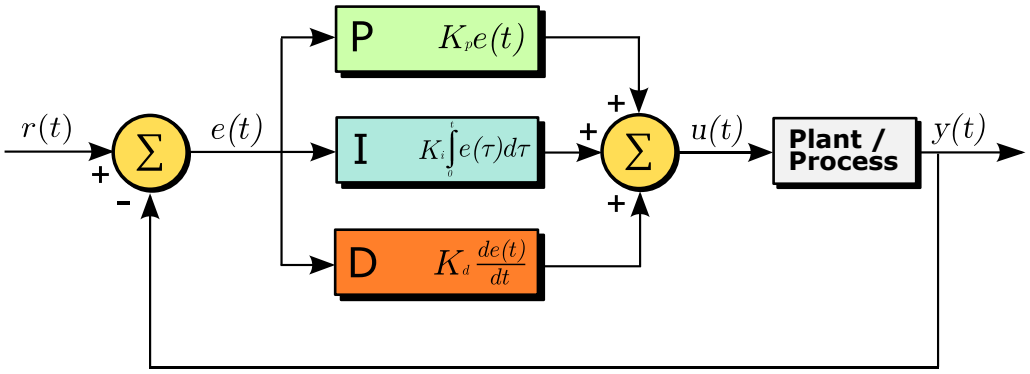

Figure 5.1: PID block diagram [47]

The controller gains seen in figure 5.1 determine the controller performance [46].

- K_P : Decreases rise time, increases overshoot, decreases the steady-state error, gives a small change in settling time and the more it is increased, the more it degrades the stability of the closed-loop system.
- K_I : Decreases rise time, increases overshoot, increases settling time, eliminates the steady-state error; the same as for K_P, the more it is increased, the more it degrades the closed-loop stability.
- K_D : Has a minor change in settling time, decreases overshoot, decreases settling time, has no effect on steady-state error, and it actually increases stability as long as K_D is small.

5.1.2 Linear Quadratic Regulator

A LQR is based on a State-Space representation of the system. It takes a full state vector, multiplies it by a gain, and subtracts it from a scaled reference. It is a type of optimal controller, meaning that we calculate the optimal K matrix from the chosen closed-loop characteristics. The choice of characteristics is basically choosing between the importance of minimizing the time spent reaching the reference and the importance of minimizing the control effort [46] [48]. There is not infinite time to fully minimize the use of energy; neither is there infinite energy to minimize the use of time. Instead, the two have to be balanced by choosing the two matrices Q and R. The Q matrix sets weights for how fast the setpoint should be reached, and

R weights the penalty of power usage. We can choose these weights according to preference. There will not be just one optimal solution for the K matrix. Instead, we have multiple optimal solutions depending on what characteristics we choose to prioritize [46] [48].

Then we have to set up a cost function. The states and input can be positive or negative, but if we want the states to approach 0, we have to minimize the error in each direction. A good option is to square the state and input. So, the cost function can be [46] [48]:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{5.1}$$

By solving the optimization problem, we calculate the optimal gain K to satisfy our preferences [46] [48].

5.2 Model-Based Observers

5.2.1 State Observers

The general problem for state feedback systems is the inability to measure a state directly. What is available in most cases is a system model and a noisy output measurement from the plant. It is important to note that the system model is heavily idealized. It is a simplified, often linearised, time-invariant representation of the system at some certain operating point. The physical plant, on the other hand, is often a non-linear time-variant system [49].

The model's inputs are the forces acting on and driving the physical plant, and the outputs are the measurable signals coming out of the plant. Most often, we do not know all the driving inputs, as unmeasurable disturbances are also acting on the physical plant [49]. The measured outputs will also be different from the prediction we acquire with our limited model; the measurement noise partially causes that [49].

State observers try to predict the states by feeding the same inputs to the plant and the model, and then calculating the difference between the measurement and the output of the model. The error is used to correct the observer to fit the plant better [49].

$$\hat{q}[n+1] = A\hat{q}[n] + bx[n] + \ell(y[n] - \hat{y}[n]) \tag{5.2}$$

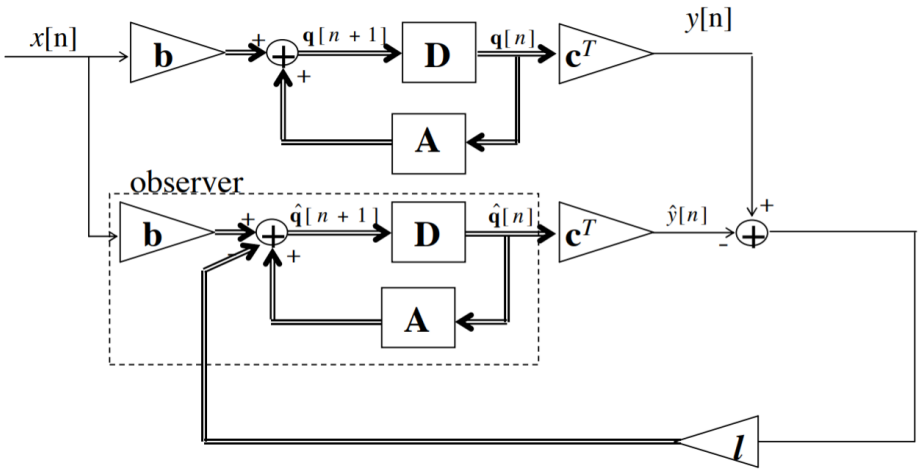

Figure 5.2: An observer for the plant in the upper part of the diagram comprises a real-time simulation of the plant, driven by the same input, and corrected by a signal derived from the output prediction error [49]

This kind of observer is well-suited for deterministic systems and can compensate quite well for some external disturbance, but it falls short with noisy output measurements. If the measurement is noisy, the correction will not function well, and the estimated states will not be suitable to use for feedback [49].

The UGPS in our ROV provides a direct measurement for the position; these are the variables we are trying to control. The problem here is the noise in the position measurement and the low sampling rate. The observer discussed so far in this section is thus not suitable for our goal.

What we need is an observer that can take in measurements from multiple different noisy sensors and combine them to achieve a better estimate of the actual position. For that, it is useful to look into sensor fusion filters.

5.3 Sensor Fusion

Humans sense the world through five basic senses: sight, hearing, smell, touch and taste. Separately, all of these provide very different data and together, they create a detailed picture of the world around us. Some of the organs for these senses come in pairs to give even better awareness. For example, for hearing, we can tell from which direction a sound is coming or for vision, we can see how far away an object is. Similarly, we can use different types of sensors or multiples of the same type of sensor in electronic systems to have a more complete or precise awareness of the world around. There are multiple different approaches, each with their strengths and possible applications, but taken together this concept is termed sensor fusion.

The primary reasons to use sensor fusion in a vehicle's electronic systems are the following [50]:

- Combining multiple sensors with different perceptive abilities for the environment.
- Multiple sensors of the same or different type(s) have different fields of view.
- A single sensor is insufficient to cover the required field of view.

When using multiple sensors or technologies, there can be conflicts in the measurements. One sensor reports the position to be 10 m; the second reports it to be 20 m. Alternatively, one sensor detects an obstacle, and the other does not. The hierarchy of the sensors and the reliability of their readings then play a big part. Which sensor does the system trust, or does it take the average of them? Sensor fusion helps with making those decisions. The sensor fusion algorithm can be inputted with information about the sensors such as the sensor noise, sample frequency and then the option to put weights on the different sensors. The algorithm can then decide how reliable the readings are and make an informed estimate of the actual position.

The reasons for inconsistency can be the following: [50]

- Differences between the processing algorithms.
- Time variant changes in the environment.
- Different sensor characteristics like noise, range, sensitivity, and accuracy.
- Sensor faults.

There are multiple approaches to dealing with these conflicts. In this thesis, we will be using a Kalman filter to fuse accelerometer, gyroscope, magnetometer and UGPS to get a better approximation of the ROV's orientation (pitch, roll, yaw), position (latitude, longitude, depth) and velocity.

One of the main elements in autonomous vehicles is vehicle localization. That is to say, all elements like navigation, decision making, and any position control require precise data about the vehicle's position and orientation at a reasonably high sample rate. Gathering this information requires multiple sensors: accelerometer, gyroscope, magnetometer and UGPS. Even though some sensors like the accelerometer can be used to calculate the position and orientation, using just one type of sensor has its limitations, so different sensors can be used to complement each other and cancel out the weak points of each sensor [50].

5.3.1 Orientation

Orientation describes how an object is rotated with respect to a known reference frame. For example, pitch for aircraft is in reference to the horizon. Once the reference frame has been chosen, there are multiple ways to describe the orientation. The most common ones used for vehicle navigation in 6 DoF are Euler angles and

quaternion. Both of them describe the same movements, but Euler angles (pitch, roll and yaw) are more intuitive for a reader to visualize [51].

The combination of accelerometer and magnetometer can give the ROV its absolute rotational orientation (pitch, roll, and yaw) in regards to the ROV body frame. The magnetometer gives the ROV the magnetic north heading. The magnetic field on the earth is not a static force and does not point to true map north. Therefore the ROV would also need to be fed the offset at its location to get a true north heading. Gravity constantly acts on the ROV with a force of $9.8m/s^2$. From that force, the orientation for pitch and roll can be determined [51] [52].

When the ROV is floating still, the results are static and reliable. However, there are several potential sources of disturbance once it is in motion. The use of low-cost MEMS in a small compact vehicle such as an ROV can easily affect the sensor with external disturbances like vibrations and the magnetic field created by surrounding metal and electrical components. Vibrations could be filtered out of the system and the sensor placed in a more protective spot. Static magnetic disturbances can be calibrated out. In figure 5.3, our calibrations test using the IMU built-in magnetic field mapping algorithms can be seen [51]. Another problem is that unless the accelerometer is placed precisely in the centre of the rotation of the object, it will also throw off the estimation [51].

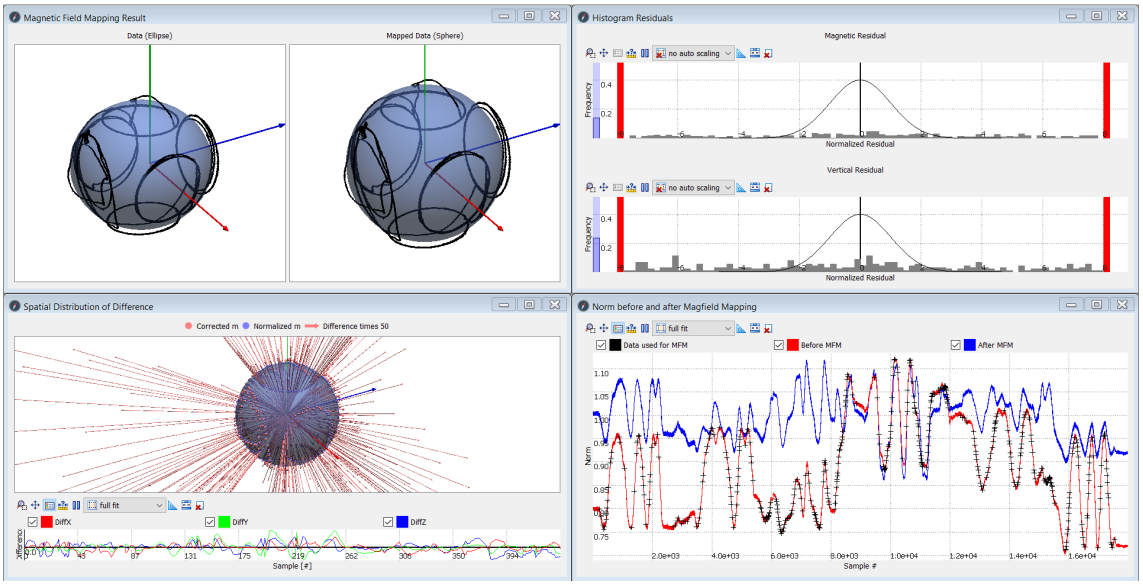

Figure 5.3: Magnetic field maping of Xsense Mti-3-AHRS mounted int the ROV

The tests ask for the ROV to be rotated around each of its axes. In a perfect scenario, the test would generate equal circles with the same offset from the origin of the system. A hard metal source would disturb the results by creating an offset from the magnetometer's origin. A soft metal would distort the shape of the circle into more of an oval shape. The test shows that the system has some hard and soft metal disturbances. The built-in algorithm calculates the necessary gains and offsets for each axis to fix the readings.

In our current set-up, we can increase the reliability of the fusion algorithm by adding the gyroscope, the mathematical system model or a combination of the two.

By adding the system model, the sensor noise in the linear acceleration can be better filtered out. That solution relies heavily on the system to be modelled correctly. A decision on the estimated output weights against the weight of the measurement is then needed.

The combination of accelerometer, magnetometer and gyroscope is quite common, and these are often packed together in IMUs. By adding the gyroscope into the fusion, the angular velocity can be measured. Then, by integrating the gyro measurements, we can determine the orientation of the object. This method is often called dead reckoning. The downsides of dead reckoning are that the initial orientation has to be known, and the sensor bias and high-frequency noise in the system. Even with a lowpass filter, the data will still drift away from the real value because of random walk and the integration of the bias [51].

These two different options of estimating orientation can be combined into one in order to emphasize the strengths and minimize the weaknesses of each of the individual solutions. The Kalman filter sensor fusion initializes the orientation with the data from accelerometer and magnetometer; then it uses the data from these two sensors to slowly fix the drift of the gyro measurements. To do that, we have to choose how much we trust the accelerometer and magnetometer measurements compared to the gyroscope measurements. This can be imagined as a slider between the two solutions, where the slider in the middle means that we trust each of them equally, and the slider fully to one or the other side means that we trust only that solution. For a complimentary filter, this trust coefficient has to be set manually, but for a Kalman filter, it is calculated automatically after defining values like how noisy the sensors are and how good the system model is [51].

5.3.2 Position

The position of the ROV can be determined by using the IMU and or by using the UGPS. There are two main reasons why it is better to use sensor fusion with both instead of just one or the other. They are [50]:

- Integrating the IMU can lead to unbounded growth of error for the position estimate. The smallest amount of error or bias when integrated will eventually lead to an incorrect position estimate. Because of this, the need for an external position measurement, to correct it, arises. The UGPS can be used for this because it provides a bounded measurement error with accurate estimates [50].
- UGPS measurements are subject to jamming, interference and signal outages, and generally have a low sampling rate. The IMU, on the other hand, is self-contained, cannot be jammed and is highly independent of the surrounding environment. Because of this, the IMU can keep providing location estimates during these temporary data outages [50].

To fuse the accelerometer and UGPS data, the first step is the fusion of the IMU data to determine the orientation of the IMU. Then, when the direction of the gravity vector is known, we can subtract it from the accelerometer readings to calculate the so-called free acceleration, the linear acceleration free from gravity. Now it can be double integrated to determine the position of the ROV. As mentioned before, this location estimate will have an error that grows unbounded over time. So, when a UGPS measurement is available with a known accuracy estimate and bounded error, a Kalman filter can be used to establish a better estimation of the location [53].

5.3.3 Kalman Filter

A Kalman filter has a two step structure: predict and update. A prediction of the location is calculated using accelerometer data and the update step is performed each time a new UGPS sample is available. For the Kalman filter we need to define four matrices and two vectors [53]:

- A - State transition matrix
- B - Control matrix
- u - Control vector
- Q - Process variance
- R - Measurement variance
- z - Measurement vector $\begin{bmatrix} p \\ v \end{bmatrix}$

The equation for the two states are the following [53]:

$$p_f = p_i + v\triangle t + a\triangle t \tag{5.3}$$

$$v_f = v_i + a\triangle t \tag{5.4}$$

In matrix form they are like this [53]:

$$\begin{bmatrix} p \\ v \end{bmatrix}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} p \\ v \end{bmatrix}_{k-1} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \times \begin{bmatrix} a \end{bmatrix} \tag{5.5}$$

From the equation above, we can take the matrix A and the matrix B. This corresponds to $A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$. The vector u is just the accelerometer measurement. The Q and R matrices can be derived from the noise and bias values from the datasheet for the accelerometer and UGPS, respectively. Vector z is the UGPS measurement [53].

5.4 System Stability, Observability, and Controllability

Before any of the control methods being used the system needs to fulfil the requirement of being stable, observable, and controllable. The eigenvalues of the open loop system were all negative and therefore the system is stable. The system is also controllable in the 5 DoF

Chapter 6

Optimal Control

An optimisation is a process of finding the 'best' solution for a given problem. Optimum is a quantitative way to find either the minimum or the maximum of a problem and to optimise is the technical word for the method to achieve an optimum. The most general approaches to use optimisation are [54];

1 Analytical methods

2 Graphical methods

3 Experimental methods

4 Numerical methods

The analytical method can mostly be used with a problem that is non-linear and has fewer than four independent parameters. The graphical method also relies on the problem having few variables. It is easiest to graph and visualise a single variable maximum/minimum and to use contours for two variables. When the number of variables increases, the graphical solution stops being simple. The experimental solutions can also be used, but this cannot guarantee optimum solutions, perhaps only near-optimum. The experimental method can lead to a more unreliable result since, in some systems, multiple variables interact with each other and would therefore need to be tuned together. That leaves the numerical method, which is able to handle highly-complex and non-linear optimisation problems. The numerical methods have all but replaced the other methods. The numerical method then branches out into [54]:

1 Linear programming

2 Integer programming

3 Quadratic programming

4 Non-linear programming

5 Dynamic programming

This thesis will look into how to achieve the optimal control solution for the system

when given a new set-point and how to compare the solutions to LQR and PID control methods. To achieve that the system needs to predict what control inputs are optimal in a specific time horizon. To do that, the system will use optimisation algorithms to try and estimate the best sets of inputs based on previous measurements and the mathematical model of the system. The optimal solution can be calculated off-line, for instance LQR, or on-line, for instance MPC.

6.1 Optimization Problem

The basic form of the mathematical optimisation problem can be seen in equation (6.1).

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & f_i(x) \leq b_i \quad i = 1, ..., m. \end{aligned} \tag{6.1}$$

In (6.1) the vector $x = (x1, ..., xn)$ is the variable that will be optimized and n is the number of elements in the state vector. The objective function is $f_0 : \mathbb{R}^n \to \mathbb{R}^n$ and the constraint function is $f_i : \mathbb{R}^n \to \mathbb{R}^n, i = 1, ..., m.$ $b_1, ..., b_m$ are the bounds for the constraints. The solution is called optimal x^* if it satisfies the constraints and has the smallest objective value of all the vectors. This is just a basic form of an optimization problem; there are other classes that have different constraints and objective functions [55].

6.2 Linear Programming

Linear programming assumes the object function and the constraints to be linear, and the variables are constrained to be positive [54]. The basic equation can be seen in equation (6.2).

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & a_i^T \leq b_i, \quad i = 1, ..., m. \end{aligned} \tag{6.2}$$

In (6.2) $c, a_1, ..., a_m \in \mathbb{R}^n$ and scalars $b_1, ..., b_m \in \mathbb{R}$ are problem parameters, which specify the objective and constraint functions. Linear programming is considered to be more complex than least-squares. However, there are good algorithms that are specialised to solve it. The approximate time complexity of the problem is $n2m$ (assuming $m \geq n$) [55].

6.3 Integer Programming

In some instances, linear programming problems require some of the variables to assume only integer values. That renders the programming problem to be non-linear, but since the objective function and constraints are linear, it is considered to be a linear problem [54].

6.4 Quadratic Programming

The basic formulations of a quadratic optimisation problem can be seen in equation (6.3) [54]:

$$\begin{aligned} \text{minimize} \quad & f(x) = a_0 + \gamma^T x + x^T Q x c^T x \\ \text{subject to} \quad & a^T x \geq \beta \end{aligned} \tag{6.3}$$

where $\mathbb{Q}$ is a positive definite or semi-definite square matrix. The constraints are linear and the object function is a quadratic [54].

6.4.1 Least-Squares Method

The least-square method is a case of a quadratic programming problem. It can take up linear constraints and even be used to solve the LQR problem [54]. Most control optimisation tools use dynamic programming to solve the LQR problem that will be addressed later in this chapter.

The least-squares problem is a special case of the least-square that does not include constraints ($m = 0$) and the objective function is the sum of the square value of the terms, seen in equation (6.4) [55].

$$\text{minimize} \quad f(x) = \| Ax - b \|_2^2 \quad = \sum_{i=1}^{k} (a_i^T x - b_i)^2 \tag{6.4}$$

In (6.4) $A \in \mathbb{R}^{k*n}$ (with $k \geq n$), a_i^T are rows of A, and the vector $x \in \mathbb{R}^n$ is the optimisation variable. The least-squares problem can be reduced to solve a set of linear equations. The benefit of using least-square is that it is commonly used and therefore algorithms exist that can solve the problem with high accuracy and with high reliability [55] [56].

6.5 Convex Optimization Problem

A convex optimisation problem definition can be seen in equation (6.5).

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & f_i(x) \leq b_i \quad i = 1, ..., m. \end{aligned} \tag{6.5}$$

In (6.5) the functions $f_1, ..., f_m : \mathbb{R}^n \to \mathbb{R}$ are convex if it satisfies (6.6)

$$f_i(\alpha v + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{6.6}$$

for all $x, y \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$.

Both least-squares and linear programming are special cases of the convex optimisation problem. There is no analytical way to solve a convex optimisation problem, but there exist some methods to solve such problems efficiently. For instance, using the Interior-Point method, a problem can be solved in a number of set steps. Those steps are usually in the range of 10 to 100, where each step requires $\max\{n^3, n^2m, F\}$ operations. F is the cost of evaluating the first and second derivatives of the objective and constraint functions $f_0, ..., f_m$. The data structure of the problem can also be exploited to increase the performance [55] [56].

6.6 Non-linear Programming

When both the object function and the constraints are non-linear the problem falls under non-linear programming. Linear and quadratic programming are viewed as special cases of non-linear programming. Therefore it is possible to solve both linear and quadratic problems using a non-linear solver, but the algorithms specialised for the linear and quadratic should be used since they are more effective [54].

6.7 Dynamic Programming

Some problems require a series of decisions to be made in sequence where subsequent decisions are influenced by earlier ones. Dynamic programming is based on linear, integer, quadratic and non-linear optimisation algorithms where a general solutions strategy has been developed for specific problems. It is useful for systems that are big and where individual sub-systems interact with each other [54].

A simple Optimal Control Problem (OPC) can be defined as [56]

$$\begin{aligned} \min_{x(),u()} \quad & J = \int_{t_0}^{t_N} F(t_k, x(t_k))dt_k + E(t_N, x(t_n)) & \\ \text{subject to} \quad & x(t_0) = x_0 & \text{Initial values} \\ & x_{k+1} = f(t_k, x(t_k), u(t_k)) & \text{System Dynamics} \\ & 0 \geq h(x_{t_k}, u_{t_k}) & \text{Stage Constraints} \\ & 0 \geq r(x(t_N)) & \text{Terminal Constraints} \end{aligned} \tag{6.7}$$

where J is the objective function, F is the stage cost function and E is the terminal cost function. The objective function is minimized in intervals $t_0, ..., t_N . x = x_0, ..., x_n$ denotes the state vector and $u = u_0, ..., u_m$ denotes the control vector of the system. The solution will give the best set of control inputs u_k to deliver the sequence of state x_k, at a set interval of $k = t_0, ..., t_{N-1}$. That satisfies the stage and terminal constraints [56] [57] [58]. The MPC is a dynamic programming of the OPC that begins by solving the problem at a time k for a set prediction horizon. The MPC feeds the solution back into the solver at a time $k+1$ and solves the problem again. The prediction horizon is then shifted to $k+1, .., k+1+N$ delivering the next sequence of predicted states and inputs. This process is repeated for a given control horizon. The general idea can be seen in figure 6.1 [56] [57] [58].

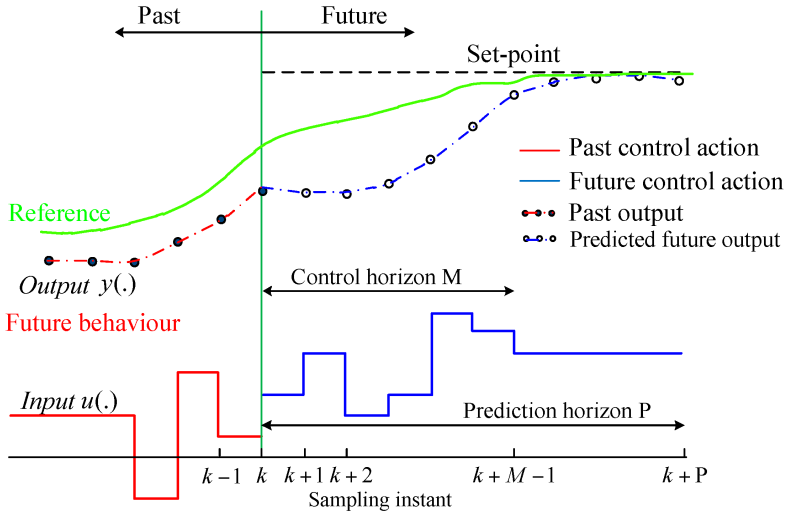

Figure 6.1: Model Predictive Control MPC [59]

A more detailed theory for MPC can be seen in [57], [60], and [58], which were used for implementation of the controller in this thesis.

Chapter 7

Implementation

7.1 Software Optimization

While running the initial tests, we found that the original hardware lacked performance. In the standard set-up, the Raspberry Pi is only used to facilitate communication between the Pixhawk and the topside computer. Essentially, it just streamed the video from the camera and passed the control signal from the topside computer on to the Pixhawk. The sensor fusion and control code ran entirely on Pixhawk, which only had one CPU core, running at 216MHz. This set-up created problems when we wanted to run a custom controller code on Raspberry Pi. The communication between Pixhawk and Raspberry Pi introduced delays in the system as Pixhawk was using its resources on things that were not necessary for our application. All the communication to Pixhawk was done using the MAVlink communication protocol, and it was impractical implementing it in the control code running on Raspberry Pi. The problems with using MAVlink have already been explained in chapter 3.3. For our new system, we aimed to communicate with the sensors directly instead of using Pixhawk.

For the new system, we set the requirement that the MPC code had to be able to run at least 20 Hz. Taking into account the massive amount of calculations that happen in each cycle, achieving this goal required a lot of consideration and planning.

After the MPC code, sensor fusion is the second most resource-heavy operation. The full Kalman filter that fuses the accelerometer, gyroscope, magnetometer and the UGPS requires significant computation. To take away part of the workload from our main processor, we decided to use an IMU that has an on-board Kalman filter. Benefits of this include that the orientation can be updated at 100 Hz, and additionally, we get the free acceleration (acceleration readings without the gravity component) and rotation matrix in the same data packet at 100 Hz. Free acceleration is necessary to fuse the IMU data further with the UGPS to get a better position estimate, and the rotation matrix allows for converting the data between the world and the body frame. In both cases, it is necessary to use slow trigonometric functions to calculate them. Getting this data ready to use from the IMU allowed us to lower

the workload on the main processor significantly.

When optimizing the code for speed, our primary focus was on utilizing the multiple cores we have in our main processor. When using python, the code is run on a single core even with the use of a multi-threading library. A multi-thread library spawns multiple threads, but they are run on the same core; thus, they have to wait on each other to get executed on the core. The benefit of using a multi-threading library is the shared memory space between the threads; in this way sharing the data between threads is as easy as adding a variable into the function argument. However, a multi-threading library did not fit our application. So we used a multiprocessing library. This library spawns another thread on a different CPU core, so the two threads can be executed at precisely the same time, but sharing data between the threads is more complicated. The fastest way was to share simple arrays containing floats or integers. We also used NumPy python library to speed up the matrix operations. This library is optimized for matrix operations and actually uses multi-processing so using that instead of regular math library significantly improved matrix calculations. We could not just pass the NumPy array from one multi-processing thread to another because NumPy arrays are objects, but we could only pass arrays, so using multi-processing had some inconveniences.

The final implementation of the Kalman filter utilized everything mentioned above with the object-based programming principles. Writing the Kalman filter as a class allows it to be implemented easily in any control code we want to run, so no matter if we want to run PID, LQR or MPC, the orientation and position data can be accessed instantly with a single line of code.

The Kalman filter runs on a different core than the controller code runs on. The multi-rate Kalman filter predicts the position from the free acceleration data at 100hz. Then we have a barometer providing the position measurement on the z-axis (depth) at 100Hz and the UGPS that gives a position measurement on the x and y-axes at 4Hz. Both the IMU and barometer use I2C to communicate, so the data from these sensors can be fetched during the main Kalman filter loop that runs at 100 Hz. UGPS data was originally fetched by an Hyper Text Markup Language (HTML) request to the topside UGPS unit. To execute this request required approximately 130ms, and that was too slow. So, another multiprocessing thread had to be spawned to continuously send the request and update the shared variable with the main Kalman filter loop. The UGPS readings did not come at precise intervals, so instead of updating the Kalman estimate with the UGPS values at set intervals, at every iteration of the loop we checked if the UGPS value had changed and then updated the Kalman filter estimate if needed.

During the initial tests running the LQR controller, we found that without any artificial delays, the 12 state LQR controller loop was running at more than 1000 Hz even with all the logging that we were doing. From that, we can conclude that our prepared platform is ready for implementing MPC controller.

7.2 Implementing Kalman Filter

Sensor fusion uses the strengths of different sensors to cancel out the weaknesses that others have. That is why the IMU achieves relatively exact orientation estimates. That being said, this section will show that different sensors do not always fit together so perfectly.

7.2.1 Q and R Matrices

The most important parts of the Kalman filter are the Q and R matrices. These are the process and measurement covariance matrices, respectively.

The process model was chosen to be

$$\begin{bmatrix} p \\ v \end{bmatrix}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} p \\ v \end{bmatrix}_{k-1} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \times \begin{bmatrix} a \end{bmatrix} \tag{7.1}$$

If we measure the standard deviation in the data where the accelerometer is standing still, we can square it to get noise variance. Then we take the above process model and insert the sample time $T = 0.01s$ into the model [53].

$$\begin{bmatrix} p \\ v \end{bmatrix}_k = \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} p \\ v \end{bmatrix}_{k-1} + \begin{bmatrix} 0.00005 \\ 0.01 \end{bmatrix} \times \begin{bmatrix} a \end{bmatrix} \tag{7.2}$$

The position is proportional to 0.00005 times the acceleration. The sensors' standard deviations were calculated in chapter 3.8. Since the sensors work on different ranges, they need to be normalised before being used for the Kalman filter. The formula for remapping the sensor data can be seen in equation (7.3).

$$y = \frac{(x - in_{min}) \cdot (out_{max}) - out_{min})}{(in_{max} - in_{min}) + out_{min}} \tag{7.3}$$

In equation (7.3) in_{min} is the lowest detachable value and in_{max} the highest. Out_{min} is the lower limit of the new scale and out_{max} is the higher limit. For the BlueROV2 test environment, the limitations of the pool will be set at min $0m$ and max $10m$ and the velocity of the ROV set to be min $0\frac{m}{s}$ and max $2\frac{m}{s}$

If the standard deviation from the acceleration measurement in the X axis is 0.24, then the variance of the position noise is $(0.00005^2) \times (0.24^2)$. The velocity is equal to 0.01 times the acceleration, so the velocity noise variance is $(0.01^2) \times (0.24^2)$. Finally, state covariance is the standard deviation of the position noise multiplied by the standard deviation of the velocity noise $(0.00005 \times 0.24) \times (0.01 \times 0.24)$. We

repeat the same standard deviation measurement for acceleration's Y and Z axes and normalise the values in order to get the following process covariance matrix [53].

$$Q = \begin{pmatrix} 3.0e^{-10} & 6.0e^{-8} & 0 & 0 & 0 & 0 \\ 6.0e^{-8} & 1.2e^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.0e^{-10} & 6.0e^{-8} & 0 & 0 \\ 0 & 0 & 6.0e^{-8} & 1.2^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.0e^{-10} & 6.0e^{-8} \\ 0 & 0 & 0 & 0 & 6.0e^{-8} & 1.2e^{-5} \end{pmatrix} \tag{7.4}$$

In our set-up, we have an R matrix with two separate sensors: the UGPS for X and Y position and the barometer for depth. The UGPS does not output the vehicle velocity, and the measurement is too noisy to take the derivative, so we only update the position.

In our case, the R matrix is:

$$R = \begin{pmatrix} 0.32 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.82 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.82 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.4e^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.0e^{-3} \end{pmatrix} \tag{7.5}$$

7.2.2 Acceleration

Estimating position by double integrating the acceleration is generally known to be a poor solution. The smallest bias or noise will increase the error with each integration. This was also the case with our IMU, where random walk was apparent in the acceleration measurement. This meant that while standing still, the position estimate from the double integration would drift. In our tests, the fast movements were clearly visible and precise, but for slow movements the signal-to-noise ratio was too low, so the position estimate was fully driven by the noise as seen in the image below. In this test, the ROV was dragged around the inside perimeter of a square pool.

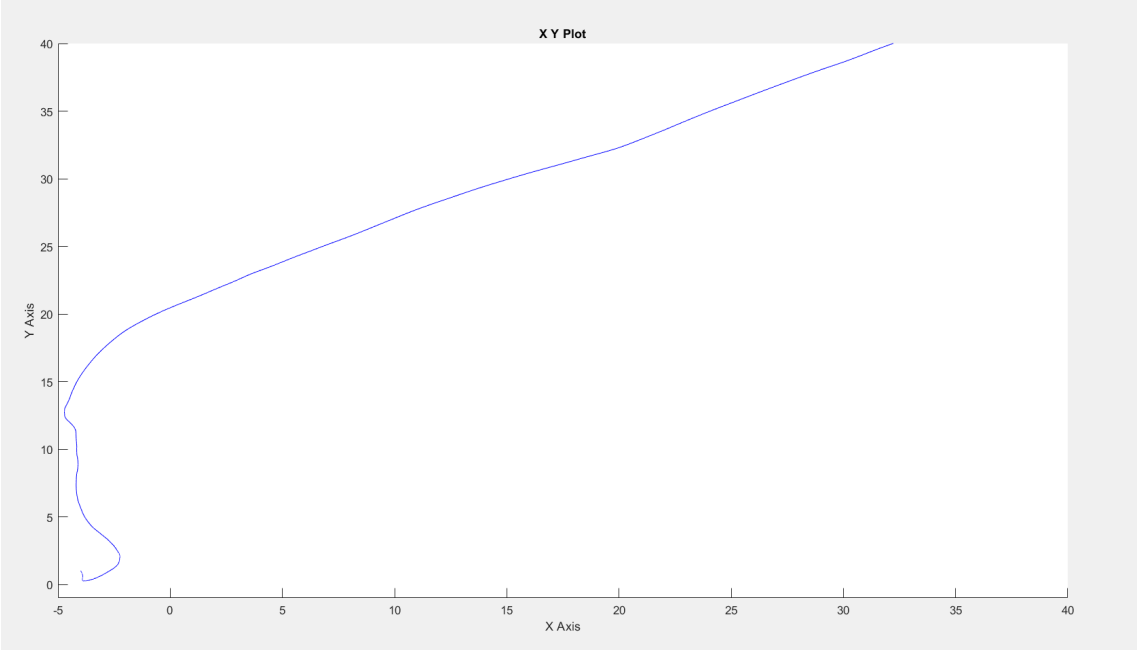

Figure 7.1: Position estimate by double integration of acceleration(X and Y axis are position in meters)

To counter this, we would need to add an absolute position measurement. Our system used the Waterlinked UGPS to prevent the drift and achieve a better position estimate.

7.2.3 UGPS

The UGPS unit has a slow sampling rate with noisy measurements. Besides the usual problems, we also noticed that it did not work well in the small pool.

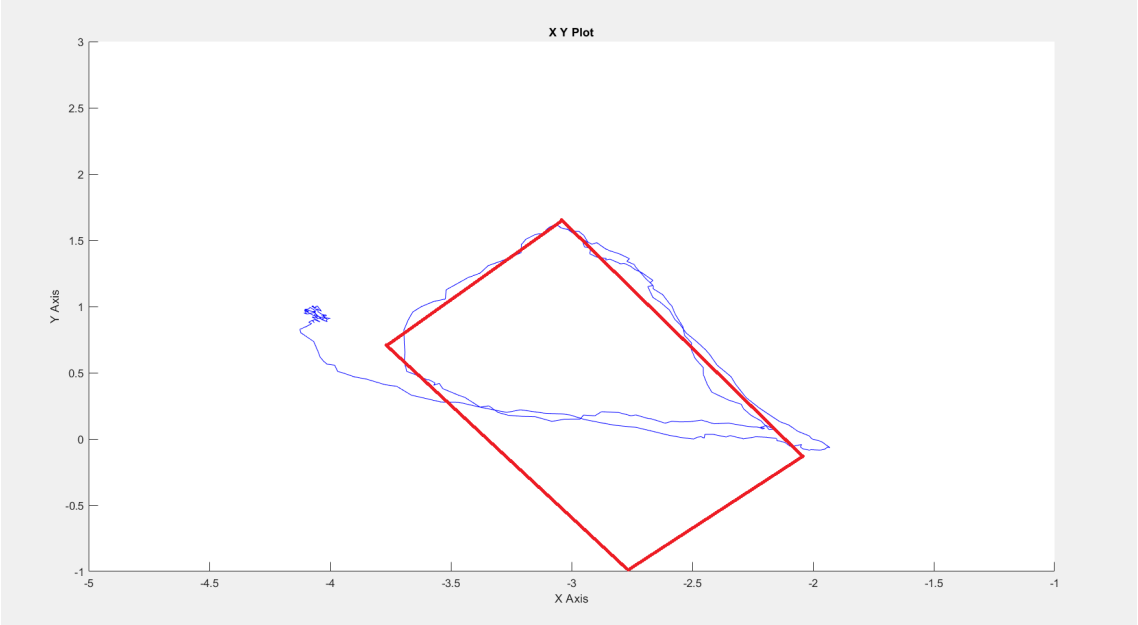

Figure 7.2: Internally filtered UGPS position. The blue path is measurements and the red path is the approximate movement path (X and Y axis are position in meters)

Even though we did not use the filtered UGPS signal in the Kalman filter, because of the delay the filter introduces, the image clearly shows the distortion in the path due to the shallow pool.

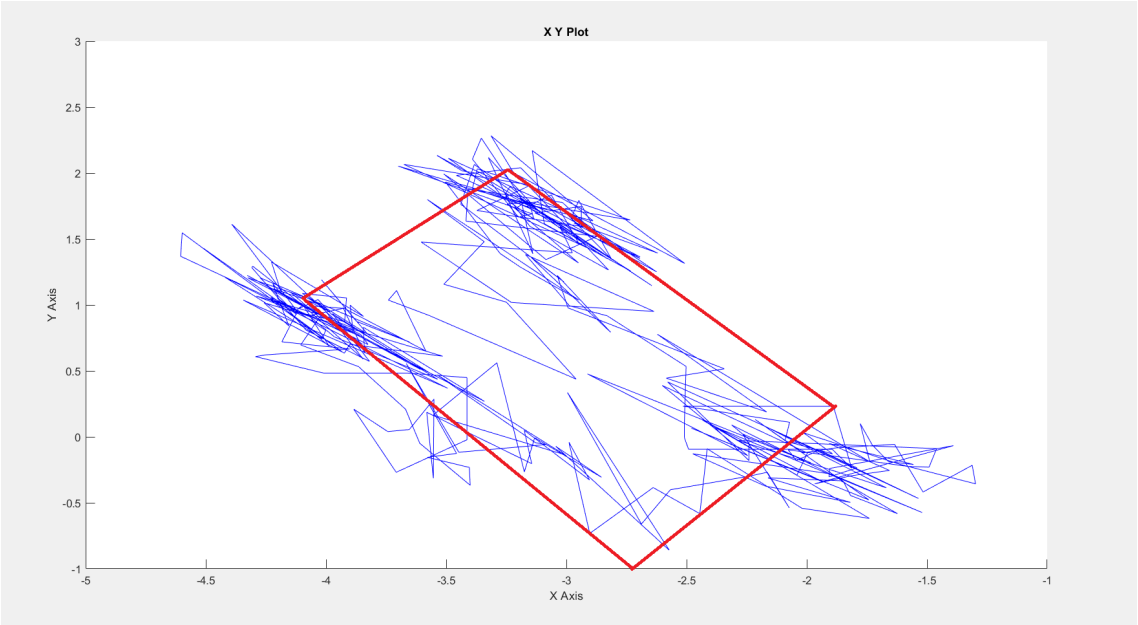

Figure 7.3: Unfiltered UGPS position. The blue path is measurements and the red path is the approximate movement path (X and Y axis are position in meters)

In the unfiltered data, the distortion is harder to see, but the real problems of the UGPS' noisy measurements and slow sampling rate are clear to see. Furthermore, due to the shallow pool, the measurements were even noisier than usual.

7.3 Discussion

When fusing the UGPS and accelerometer sensor data, their individual faults were still quite apparent. If more weight was put on the UGPS readings, the distortion and noise became more visible in the estimation. When the accelerometer was given more weight, the estimation would drift more while the ROV was stationary. The image below shows clearly that the Kalman filter in this state could not provide us with a position estimate decent enough to use for the controller feedback.

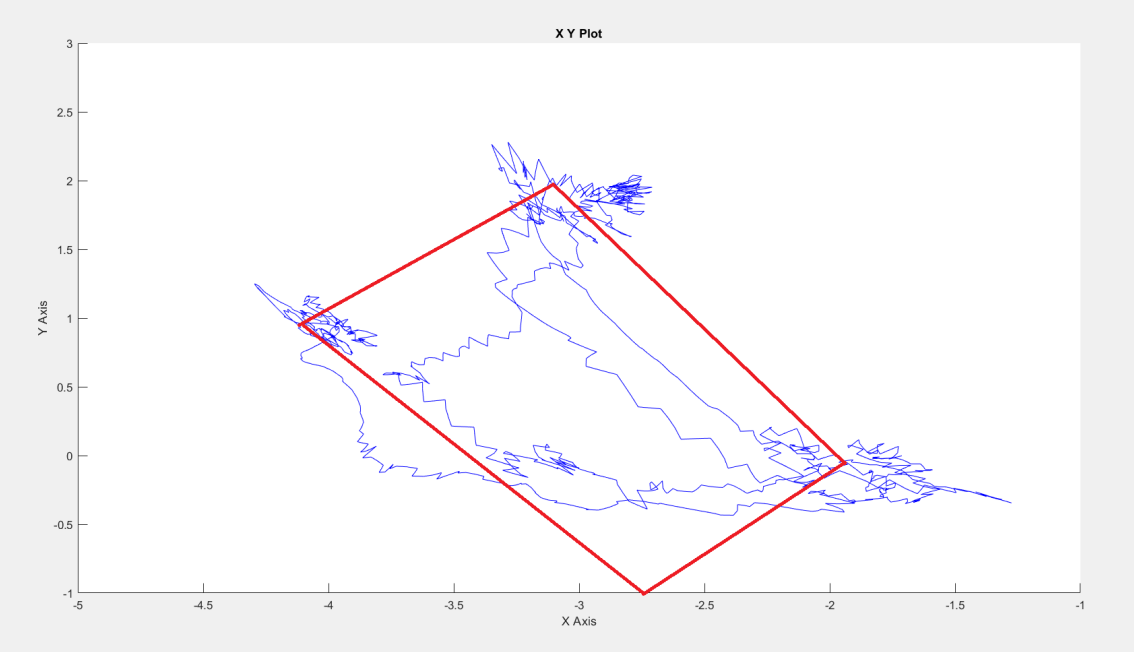

Figure 7.4: Kalman filter estimation. The blue path is measurements and the red path is the approximate movement path (X and Y axis are position in meters)

We consulted with the UGPS manufacturer and experimented with the placement of the locators in the pool until we got satisfactory results.

In the end, we did not place the locators at the corners of the pool. Instead, the locators were placed in a square in the middle of the pool, at least 1.5m away from the pool walls. That helped with getting a more accurate position measurement inside the square between the locators.

However, it did not fix the problem of drift during standstill. Because of the UGPS, the drift in the position estimate did not exceed the error variance of the UGPS, but it would still move around to some degree. To fix it permanently, we would need a sensor that could tell when the ROV is standing still.

For deployment in a pool with a painted pattern on the bottom, a downward-facing camera would be a possible solution. In water where the surface of the bottom is uneven, a DVL would be perfect.

All of the adjustments were made with the consideration that we wanted to use the

fused data for feedback for the surge controller. As seen in the image below, we found a workaround for the sensor limitations to meet the minimum requirements for the position estimate.

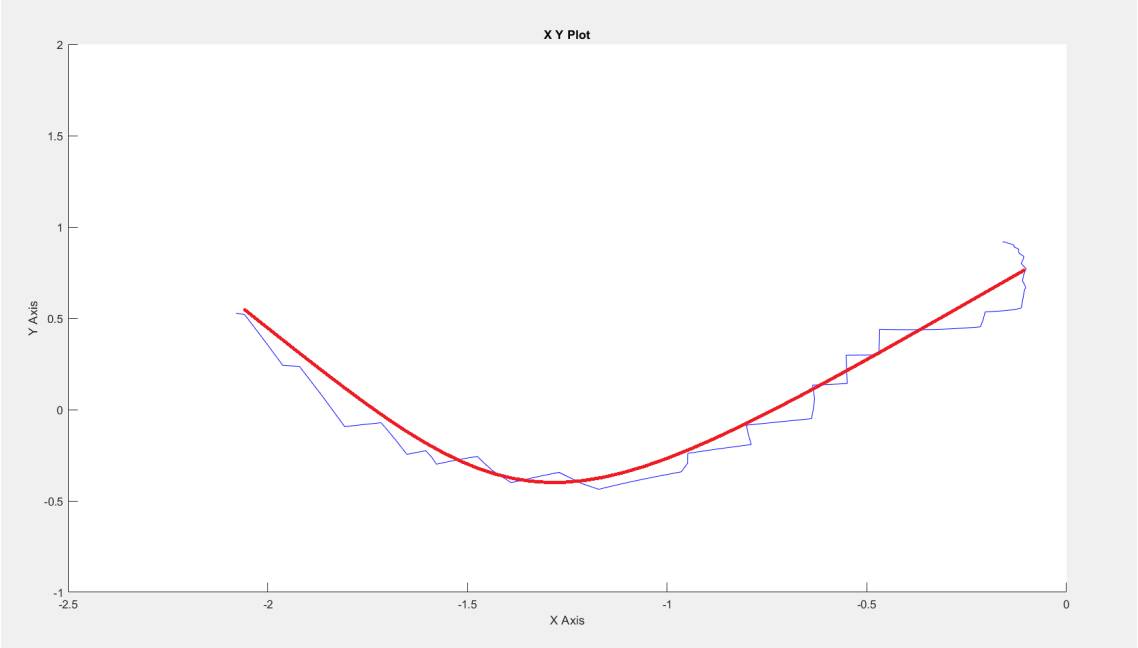

Figure 7.5: Kalman filter estimation. The blue path is the measurements and the red path is the approximate movement path (X and Y axis are position in meters)

In this test, the ROV does not stand still at any time. As soon as the Kalman filter has initialised, the thrusters go forward with 60% thrust. Because of the tether, the path is not straight but curved due to the tether pulling the ROV and steering it to the side.

For a good position estimate, the requirements are:

- The ROV does not stand still at any moment.
- The ROV stays in between the UGPS locators.
- The initial states are close to the real ones.
- The acceleration is not too low.

The Kalman filter should be tested in a big pool in order to see if the bigger pool increases the UGPS' performance. Nonetheless, the tests in the small pool indicated that the sensor fusion should be improved. This could be done by some of the following options:

- Adding a second accelerometer. By doing so, there would be another high sampling rate sensor and it would have different noise and bias drift characteristics. Therefore combining the two would give a better acceleration reading, and we could give less weight to the UGPS readings.

- Adding a second set of UGPS. Again, the two UGPS readings could provide a better absolute position to be fused with the accelerometer readings.
- Adding a completely different sensor. For example, DVL would add a direct reading for the velocity. Thus the velocity and position estimate could be greatly improved.
- Adding the actual linear model of the ROV as the system model in the Kalman filter and using it for the prediction step. Then, updating the predicted states, with the accelerometer, UGPS and depth sensor readings. This could improve the state estimate at slow speeds. When the input is 0 at all thrusters, the predicted states would be the same, so when the ROV is standing still, the position estimate would not fluctuate so much. However, external disturbances and thruster operations outside the range that model was linearised around could decrease the position estimate accuracy.
- Using a non-linear ROV model would have the same benefits, and it would also improve the estimate through the thrusters' whole range of operation. The downside would be that this would require an Extended Kalman filter and it would increase the computational load.

7.4 MPC Implementation

After looking into multiple python libraries, it was determined that the two best-suited candidates were GEKKO [61] and Do-MPC [62]. The main difference between these two is that GEKKO was made to solve a wide range of optimization problems, including MPC, while the Do-MPC was made as Robust Multi-stage NMPC. Both of the packages used Interior Point OPTimizer (IPOPT) [63] as the nonlinear solver. IPOPT would use the MUltifrontal Massively Parallel sparse direct Solver (MUMPS) for linear problems [64]. MUMPS is provided in the IPOPT package as the default, but provided the option to choose different solvers. The problem with the extra solvers is that they require a license that we could not obtain. The other solver could improve the linear solver quite drasticly, but for this implementation we will use the default MUMPS.

We implemented the model first in GEKKO, because we wanted to use a regular MPC, and did some simulation tests. The tests provided promising off-line results as seen in the figure below.

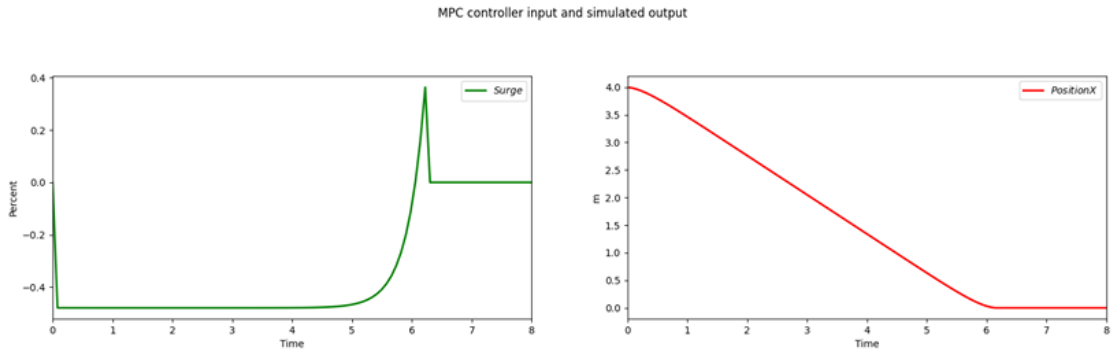

Figure 7.6: Optimal control input calculated by MPC implemented with GEKKO package. The objective for the controller was to return to 0 from the initial position at 4 m in less than 6.8 seconds with the minimum amount of thrust possible

Trying to implement the same model on the ROV, it was only possible to achieve 10Hz for the control loop with a really small prediction horizon. This would not give us enough flexibility to tune the controller. From the IPOPT printed output, we could see that the solution took only 0.01 s, so changing to a different solver would not help, but the solve function in GEKKO had too much overhead and took on average 0.9 s to return the control input. This seemed to be too slow. Tuning the prediction and control horizon would change the IPOPT solution time, but not the GEKKO.solve() function. The GEKKO library was made for a wide range of optimization problems, and that might have made the overhead too large to run the MPC real-time with the necessary sample time.

We opted to use Do-MPC instead. After reading the documentation for the package, it was clear that we could make it function as a regular MPC if we set the specific parameters for the general formulation of the Robust Multi-stage NMPC. A short summary about the package can be read in the Appendix C.

The closed-loop system diagram can be seen in figure 7.7. The Kalman filter and MPC are running on different threads, with the Kalman filter running at 50 Hz. At the start of each iteration of the control loop, the MPC takes the newest estimate of the states and calculates the new optimal control inputs. While we only update the control input for the plant at 10 Hz, during this one loop, the optimizer goes through multiple iterations to find the optimal control input to minimize the cost function within prediction horizon.

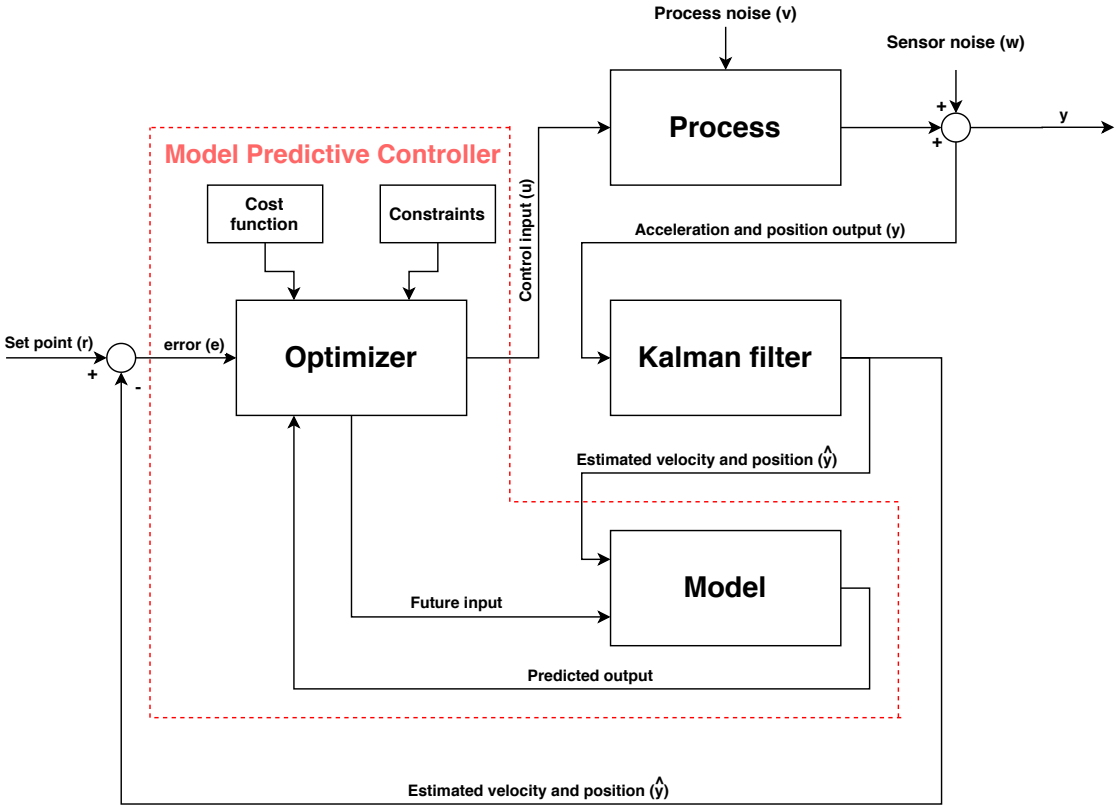

Figure 7.7: Closed-loop system block diagram

7.4.1 Implementation Results

The main steps to initialise and tune the Do-MPC controller are defined below.

1. Define that the model is discrete.
2. Define the matrices A, B, C and set x as the state vector and u as the input vector.
3. Define the state equation $x_{next} = A \times x + B \times u$
4. Define the cost function as row sum of setpoint vector minus state vector, where each element is squared.

   ```
   expr=sum1((setpoint-x)^2)
   ```

5. Set the sampling time and prediction horizon for the controller.
6. Set the hard constraints on the input vector and possibly the state vector.
7. Set the penalty for the change of input to make the change of input smoother.
8. Initialise the model with the initial state.
9. At each time step, ask the controller to calculate the new control inputs while providing the new state measurements.

For this testing, we had access to a large pool for a limited time. The first thing we had to do was to make sure that the axis for the UGPS and IMU matched. The IMU was in the ENU coordinate frame, and UGPS was in an arbitrary frame that depended on the placement of the locators physically and in the software. We had to rotate the gravity-free acceleration data and the yaw data so that the X axis from the IMU would point the same direction as the X axis of the UGPS. This was necessary so that when we moved the ROV in the positive direction on the X axis for the UGPS, the measured acceleration also would be positive in the X direction. Only then could we fuse the UGPS and IMU to get an accurate position estimate. This was something we had considered before, so the Kalman filter code allows us to define the rotation needed when we initialise it.

Once that was done, we started the tests by running the controller just as we designed in Matlab, where we set the weights in the cost function for the position and velocity for each degree of freedom. The resultant controller in simulations had the results as seen in figures 7.8, 7.9 and 7.10.

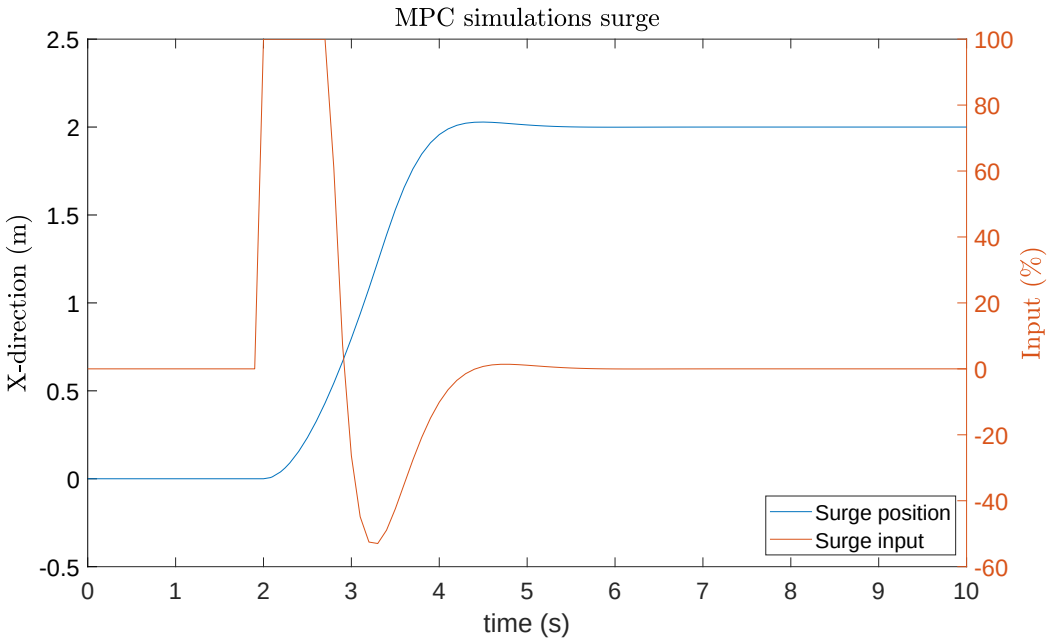

Figure 7.8: Simulated surge position and control input from the closed loop test with MPC

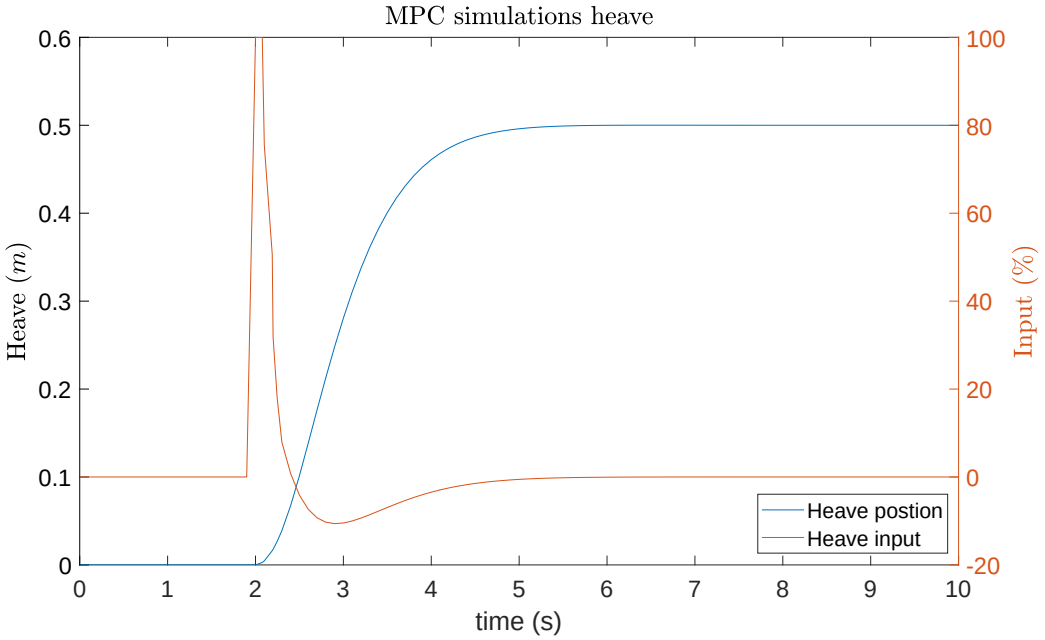

Figure 7.9: Simulated heave position and control input from the closed loop test with MPC

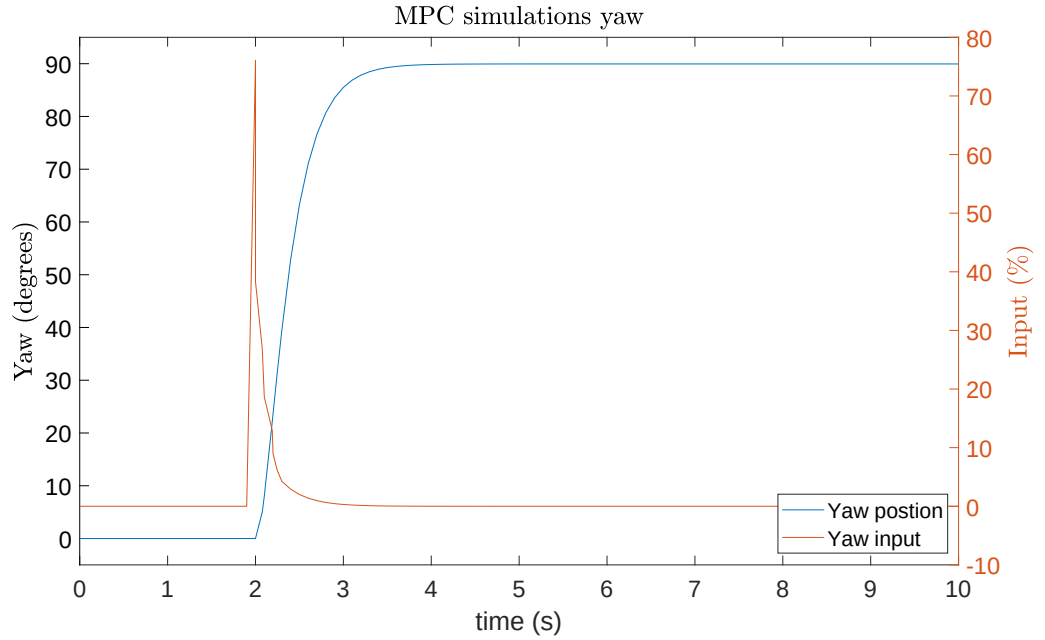

Figure 7.10: Simulated yaw position and control input from the closed loop test with MPC

After the initial tests the result were unsatisfactory and nothing like we simulated. We tried to tune the weights, but realized that the velocity readings were too noisy, so we decided to remove the velocity feedback from different degrees of freedom, while leaving the velocity in the cost function. This would mean that the controller would try to minimize the error in position while not letting the predicted velocity get too high. This would give a response that was acceptable for a start, but as we will show later, it needed improving.

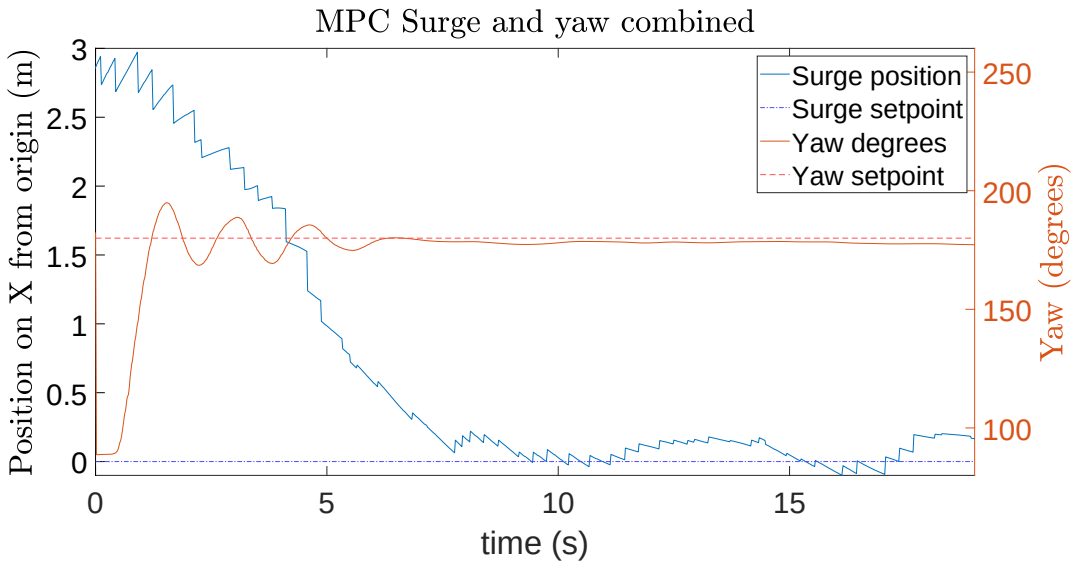

Figure 7.11: Measured surge and yaw position from the closed loop test with MPC

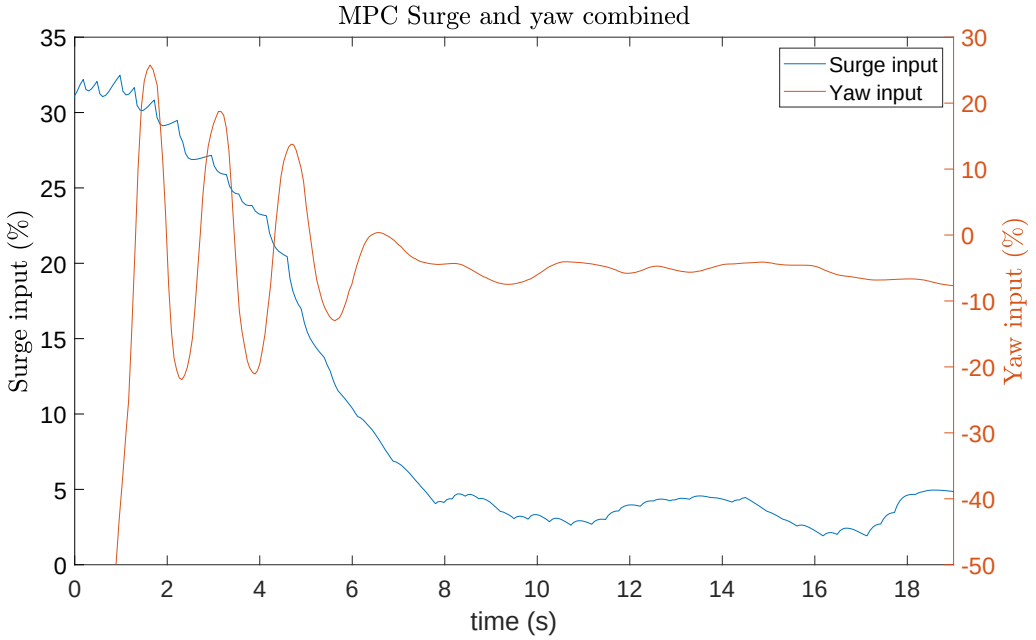

Figure 7.12: The control input from the closed loop test with MPC

As visible in figure 7.11, the controller did well with setpoint tracking. The ROV would get to the position and stay there. It would counter the disturbances to stay in the designated spot. The UGPS also performed better than in the small pool, so we had proved that the Kalman filter output could be used for the state feedback for the MPC.

When we went home and analysed the gathered data, we could see some trends in the input data worth discussing. Comparing the input data in figure 7.12 with the output in the figure 7.11, it looks like the control input is just error multiplied with

some gain. What we essentially had was a P controller. There was overshoot in the system and increasing the weight in the cost function of the position would only make the system oscillate more at the steady state. It was clear to us that we had to investigate why we had problems with the velocity estimate and see if it could be improved.

Improving the Velocity Measurements

We investigated the problems for the velocity estimate for the surge, heave and yaw.

For yaw, it was an easy fix - the velocity had the wrong sign due to the logic for how the setpoint for the yaw was tracked. There was a logic in place that would allow the controller to find the fastest route to the setpoint so that it would not turn +270 degrees when it could turn -90. The output of this logic was fed in the controller as the yaw position state. The velocity was fed to the controller directly. Because of this logic, the velocity was given the wrong sign.

Next, we turned to the heave. We could see that sometimes there was a faulty measurement from the depth sensor. While it was not such a big problem for the position, the velocity experienced huge spikes because of it. We investigated the faulty reading and found out that the culprit was likely the electromagnetic interference. This happened because the last time when we closed the ROV enclosure, the depth sensor wires had fallen right between the two poles of the ESC. We tried increasing the noise variance in the R matrix for the depth sensor and the velocity estimate was greatly improved.

The surge, on the other hand could not be improved. As we had already said it in the 3.8, both the IMU and the UGPS had noisy and drifting measurements and neither measured the velocity. For the depth the fusion was between the IMU and the barometer. The variance for the barometer was much smaller than for the UGPS; that is why it did not have the same problem. For surge, however, while the position estimate was good enough to be used with a P like controller, the velocity had too much noise to be included for the velocity feedback.

Adding the Velocity Feedback

Having improved velocity measurements, we could add them for state feedback. Now the controller knew what speed the ROV was going at, so it would anticipate the effect of the momentum and could choose its future inputs accordingly.

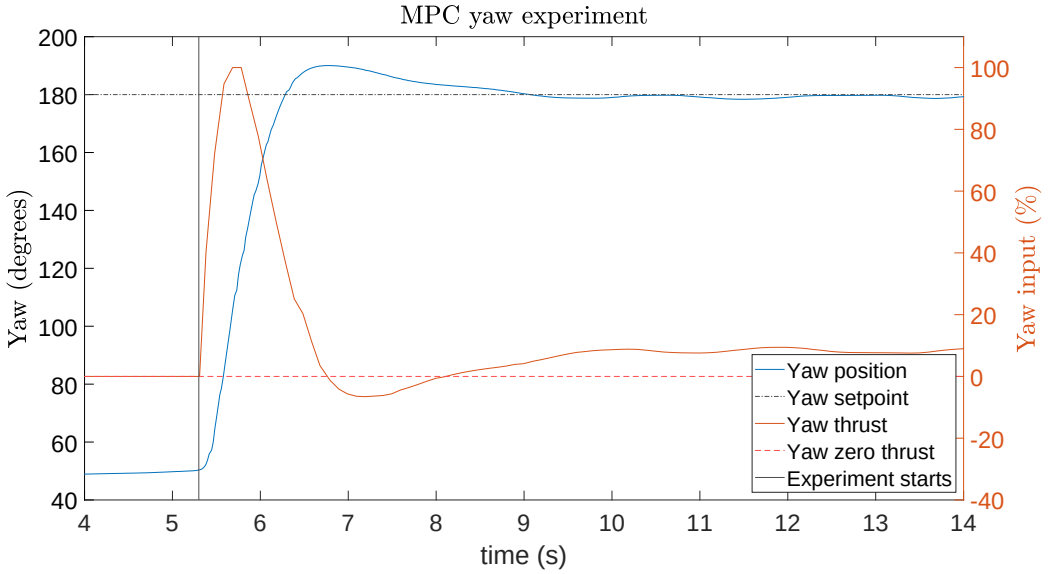

Figure 7.13: Measured yaw position and control input from the closed-loop test, where the velocity feedback was added to the MPC

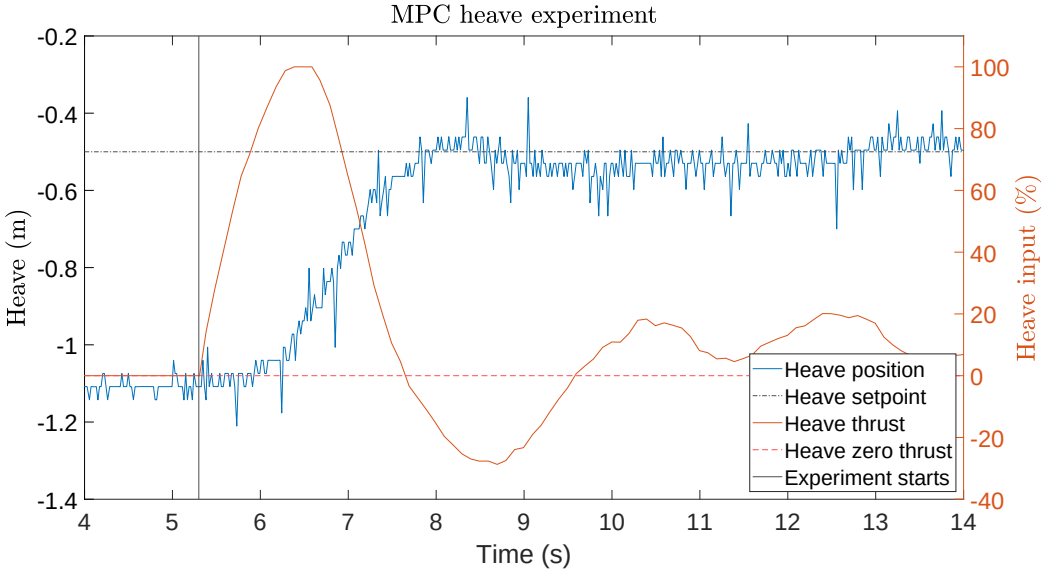

Figure 7.14: Measured heave position and control input from the closed-loop test, where the velocity feedback was added to the MPC

After some tuning of the weights, the new results for the yaw and heave can be seen in the figures 7.13 and 7.14, respectively. Now the control input looked more sophisticated and the output data also looked better. Both overshoot and rise time were decreased. The controller could go to 100 %, which we could not have achieved without the addition of the velocity feedback because the momentum would carry the ROV past the set-point.

MPC and PID Comparison

Adding the velocity as a feedback had a similar effect as adding a D term to a P controller, resulting in a PD controller. The main difference was that a PD controller would calculate the control input by what the error and the rate of change for the error are now, while an MPC could predict whether there would be overshoot going at a given speed, and so could penalise the position error before it happened. So, when tuning the MPC, we could increase the weight of the position error to decrease the rise time, settling time, and overshoot. Increasing the P in a PID controller would decrease the rise time, but increase the settling time and overshoot. As visible in figure 7.15, tuning the PID to minimize the rise time without increasing the settling time and overshoot is difficult, and we were not able to tune it to get as fast a rise time as for the MPC without increasing overshoot, oscillations or decreasing the closed-loop stability.

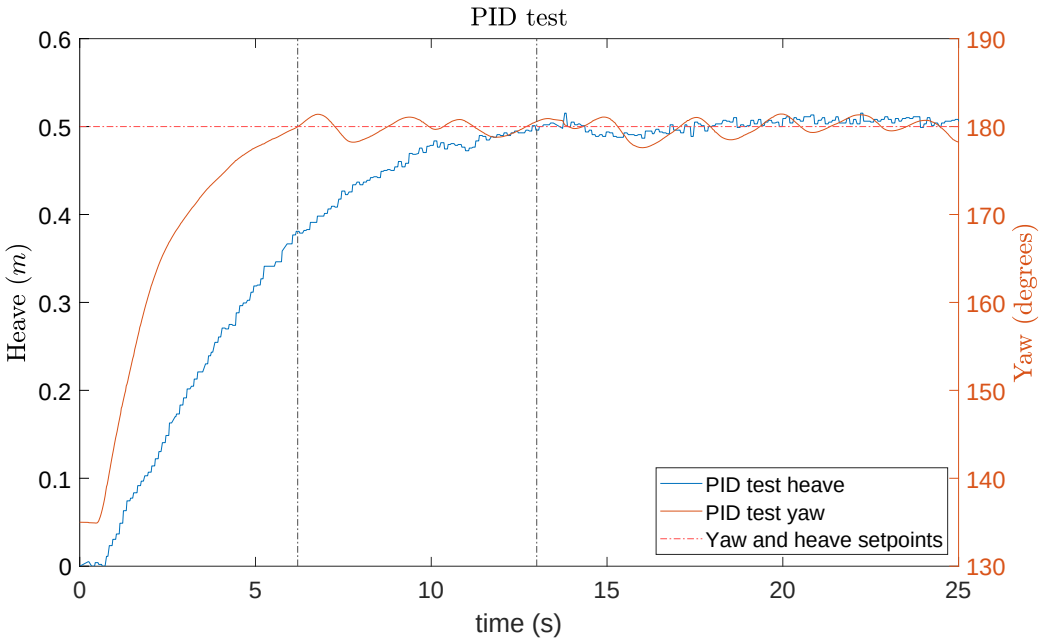

Figure 7.15: Measured heave and yaw position from the closed-loop test with a PID controller, where gain values for heave were $K_p = 0.5$,$K_i = 0.02$ and $K_d = 1$, and values for yaw were $K_p = 0.09$,$K_i = 0.001$ and $K_d = 0.03$

MPC and LQR Comparison

To evaluate the results we got with the MPC, we implemented another MIMO controller, namely the LQR. While doing so, we could see the benefits of the platform we had created by modifying the BlueROV. We could quite easily modify the code to use an LQR controller instead of an MPC. Essentially, the controllers can just be switched out as necessary.

After spending some time tuning the LQR controller, and analysing the results of the tests seen in figures 7.17 and 7.16, we could come to multiple conclusions.

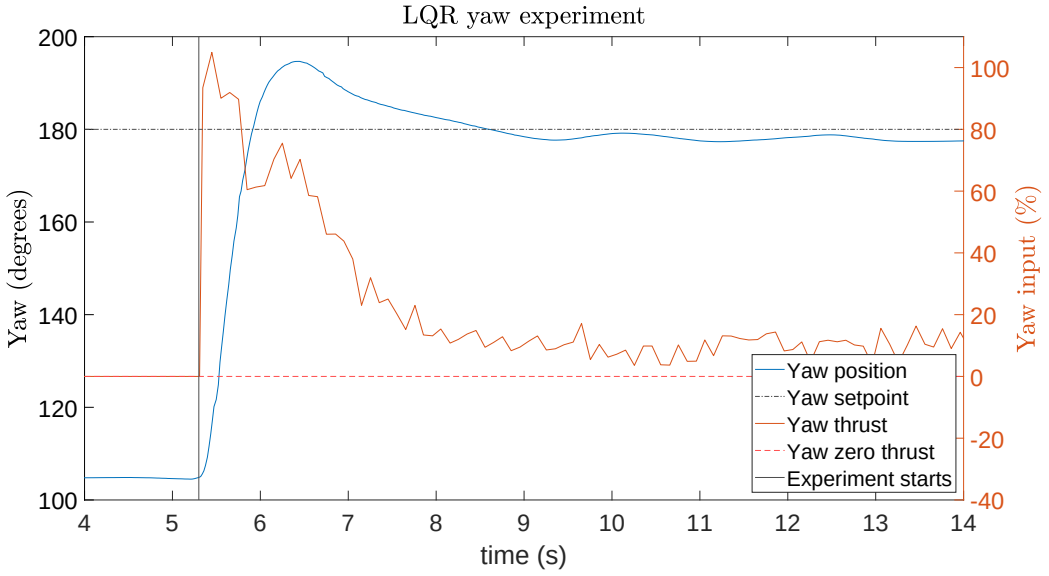

Figure 7.16: Measured yaw position and control input from the closed-loop test with the LQR

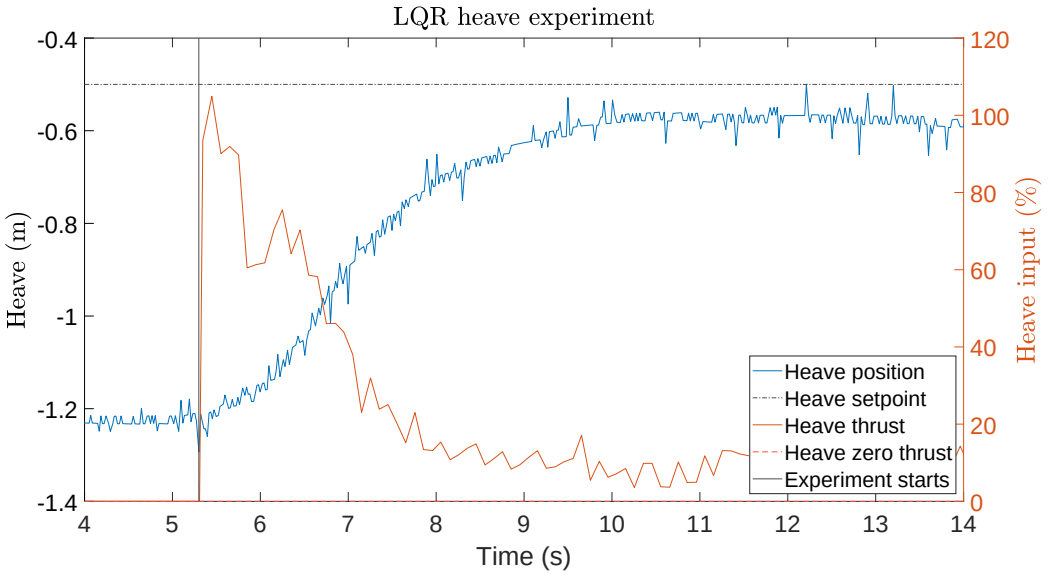

Figure 7.17: Measured heave position and control input from the closed-loop test with the LQR

First, the control input for the MPC is smoother. This is because for the MPC we can penalise the rate of change for the control input. For a physical system, it would make the whole operation smoother. Like when driving in a car, the experience is more pleasant if the driver presses and releases the gas pedal gradually. Under water, this effect is amplified, so gradually increasing thrust would avoid the ROV pitching too much, which is undesirable because in this configuration the ROV is under-actuated to control the pitch. Another benefit is that at the steady state, the thrusters can change the direction quickly to keep the ROV there. It is healthier for the thrusters if the direction does not change so suddenly.

Second, the rise and settling times have increased and there is steady state error for heave. These things can be improved with tuning; doing that revealed a difference between the controllers. The LQR controller can be tuned by changing the weight for penalising the error (set-point-state) for each state and /or penalising the control input. But, it calculates the control value only from the current state, so if the weight for the velocity is increased, it will slow down the whole system because the higher the speed, the more it will try to slow down. The MPC, on the other hand, can predict future states so that there is more flexibility of how to prevent overshoot. One way to do so is to increase the weight in the cost function for velocity, thus slowing the system down, so the momentum would be smaller. Another way is to tune the weight of the position in the cost function together with the prediction horizon. This way avoids penalising the velocity so that the system would not be slowed down overall. Instead, an option is to penalise the future position error that would happen because of momentum. This keeps the system fast, so the rise time is not affected, but when the MPC sees the position error in its prediction horizon, it will start slowing the system down to avoid this overshoot. Intuitively the MPC approach seems better because there is no need to sacrifice the rise time to avoid the overshoot.

7.4.2 Possible Improvements

The performance of the MPC is highly influenced by the accuracy of the model. The model used now is simplified: all the coupling between surge, heave, and sway has been removed, and the model is linearised even though the thruster output is highly non-linear. On the one hand, coupling the states could increase the performance of the controller, because the effect it sees as an unmeasured disturbance would be predicted. On the other hand, it would also make the optimisation problem more complicated.

We could also experiment with different linearised models around other operating points; maybe one of them works better than the other. However, if the linearisation of the model seems to introduce problems in the controller performance, it would be worth investigating the use of NMPC. This would increase the computational load of the controller, but the prediction of the future states would be much better, and thus the computed optimal control input would also be better.

Chapter 8

Conclusion

This chapter outlines the achievements, reflections and outcomes of the thesis. It also contains strategies for future improvements.

8.1 Thesis Outcome

In chapter 1.3 the problem definition is stated. We sought to:

- Analyse the current hardware and software on the BlueROV2 for their suitability for MPC implementation.
- Identify the shortcomings and possible improvements that can be made.

The brains of the BlueROV2 hardware, mainly the Pixhawk 4 and the Raspberry Pi 3, proved to be hard to work with and were not good enough for implementing a reliable control algorithm. Along with the BlueROV2 software, the ArduSub and QGroundControl, the system was difficult to modify to our needs. The control loops did not run at a steady rate. When the Waterlinked UGPS was added to the system, it would slow down the main control loop, and there was no way short of rewriting the whole code to fix these problems.

The decision was made to change out the Pixhawk 4 for a simple I/O board and the Xsens Mti-3-DK IMU, and the Raspberry Pi 3 to the Khadas Vim3-Pro. In combination with the hardware, a new software algorithm was written that used multi-processor logic to run different sensor and control threads. The limitations are detailed in chapter 3 and the comparison between the control loops can be seen in table 8.1.

Table 8.1: Old and new system open-loop comparison

Hardware/software	BlueROV	BlueROV + GPS	New + GPS
Set sample rate	0.02	0.02	0.01
Actual sample rate	0.025	0.03	0.013
Longest delay	0.06	0.26	0.018
% of delay 2x sample rate	15%	25 %	0%

- Implement an embedded MPC controller for the ROV and analyse the controller performance compared to controllers already commonly used in the industry, namely PID and LQR.

With the updated hardware and software, the implementation of the embedded MPC was ready. The main limitation was now the sensors available. In chapter 3.4, we go more into detail of the new sensors and the problems faced with the Waterlinked UGPS. In the end, we managed to implement the MPC to a satisfactory result for heave and yaw but were lacking reliable velocity feedback to implement a good surge controller.

Even though a PID controller is easy to implement, it cannot handle a coupled MIMO system. As explained in the modelling chapter 2.3 and confirmed in the validation chapter 4, the system is complex and coupled. Therefore, a more advanced controller is needed. A comparison with an off-line optimisation controller LQR was also made. Even though we saw better improvement for the LQR compared to previous work done with the BlueROV2, the MPC still has the advantage since it can make predictions in the future and constraints can be added to the system. A more in-depth look is needed for the full implementation of the MPC, such as soft constraints on high thrust inputs and dealing with the thrusters dead-zone. Even without that, the MPC outperformed both controllers. A comparison can be seen in table 8.2.

Table 8.2: Comparison between controllers

Controllers	Settling time	Steady-state	Overshoot
PID	13.0s	0.0s	0.0%
LQR	3.0s	0.1s	8.5%
MPC	2.8s	0.0s	2.5%

8.2 Future Work

Even though this thesis managed to meet all of the problems set out in the beginning, there is still room for more improvement.

- The Kalman filter can be improved by implementing the BlueROV2 linear model to the prediction.
- The BlueROV2 non-linear model can also be used with an Extended Kalman filter; then the predictions would be closer to real life.

- An additional IMU for better sensor fusion would help with the system dead-reckoning
- The biggest improvement would be the introduction of an extra positioning sensor, like a DVL, which would provide reliable velocity feedback.
- The MPC could be improved by fully utilising the constraints and trying to tackle the dead-zone of the thrusters.
- The BlueROV2 model could be further improved by modelling the tether dynamics. The tether can act as a big disturbance on the ROV and by modelling it, the simulations would be closer to real life.
- The MPC could also be improved with a more dedicated and capable optimiser. By applying for a license for some of the more advanced optimisers, the MPC could show significant improvements by increasing the prediction horizon or cutting down computation time.
- Like with the Kalman filter, using the BlueROV2 non-linear model with an NMPC optimiser would help with catching the full ROV dynamics better.

References

[1] National Oceanic and Atmospheric Administraion (NOAA), "National Ocean Service." http://oceanservice.noaa.gov/facts/exploration.html, Accessed March 2020.

[2] U.S. Energy Information Administration, based on Rystad Energy, "Offshore production nearly 30 percentage of global crude oil output in 2015." https://www.eia.gov/todayinenergy/detail.php?id=28492.

[3] ROV - Marine Technology Society, "ROV Categories." http://www.rov.org/rov_categories.cfm, 2017.

[4] A. Reid, "Rov market prospects." https://www.subseauk.com/documents/presentations/ssuk%20-%20rov%20event%20-%20sep%202013%20%5Bweb%5D.pdf, 2013.

[5] UNCW, "Undersea Vehicles Program." https://uncw.edu/uvp/documents/dayratecharges.pdf.

[6] Deep Trekker, "Diver Versus Deep Trekker: A Cost/Benefit Analysis." https://www.deeptrekker.com/resources/diver-versus-rov.

[7] Shell, "Perdido - Overview." https://www.shell.com/about-us/major-projects/perdido/perdido-an-overview.htm.

[8] US Navy, "US Navy Dive Decompression Tables." https://www.dir.ca.gov/oshsb/decompressionAC-USNdivetables.pdf.

[9] J. Bradbury, "Man vs ROV: Offshore operating costs compared, Underwater Technoligy Conference 2019." https://www.utc.no/man-v-rov-offshore-operating-costs-compared-2/.

[10] J. Stump, "Industry advances rov and auv technologies - offshore." https://www.offshore-mag.com/subsea/article/16764019/industry-advances-rov-and-auv-technologies, Accsessed June 2020.

[11] R. B. Wynn, V. A. Huvenne, T. P. L. Bas, B. J. Murton, D. P. Connelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Sumner, S. E. Darby, R. M. Dorrell, and J. E. Hunt, "Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience," *Marine Geology*, vol. 352, pp. 451 – 468, 2014. 50th Anniversary Special Issue.

[12] D. R. Yoerger, M. Jakuba, A. M. Bradley, and B. Bingham, "Techniques for deep sea near bottom survey using an autonomous underwater vehicle," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 41–54, 2007.

[13] S. Yoon and C. Qiao, "Cooperative search and survey using autonomous underwater vehicles (auvs)," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 364–379, March 2011.

[14] B. Bingham, B. Foley, H. Singh, R. Camilli, K. Delaporta, R. Eustice, A. Mallios, D. Mindell, C. Roman, and D. Sakellariou, "Robotic tools for deep water archaeology: Surveying an ancient shipwreck with an autonomous underwater vehicle," *Journal of Field Robotics*, vol. 27, no. 6, pp. 702–717, 2010.

[15] C. Roman and R. Mather, "Autonomous underwater vehicles as tools for deep-submergence archaeology," *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, vol. 224, no. 4, pp. 327–340, 2010.

[16] R. L. Wernli, "Auv commercialization-who's leading the pack?," vol. 1, pp. 391–395 vol.1, Sep. 2000.

[17] W. Naeem, R. Sutton, and s. Ahmad, "Pure pursuit guidance and model predictive control of an autonomous underwater vehicle for cable/pipeline tracking," *IMarEST Journal of Marine Science and Environment, PartC*, vol. 1, 05 2003.

[18] T. I. Fossen, *Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles*. Marine Cybernetics, Trondheim, Norway, 2002.

[19] M. M. Hunt, W. M. Marquet, D. A. Moller, K. R. Peal, W. K. Smith, and R. C. Spindel, "An acoustic navigation system," *Woods Hole Oceangraphuc Institution*, 12 1974.

[20] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*, ch. 9. John Wiley & Sons Ltd., 1. ed., 2011.

[21] A. J. Healey and D. B. Marco, "Slow speed flight control of autonomous underwater vehicles: Experimental results with nps auv ii," 1992.

[22] C. Shen, "Motion control of autonomous underwater vehicles using advanced model predictive control strategy." University of Victoria, 2018.

[23] Blue Robotics Inc, "Bluerov2." https://www.bluerobotics.com/store/rov/bluerov2/bluerov2/, Accessed September 2019.

[24] Rawson KJ, Tupper EC, "Basic ship theory," *Longman Group, Harlow/New York*.

[25] Lewanddowski EM, "The dynamics of marine craft," *World Scientific, Singapore MSS*.

[26] Perez T, "Ship motion control," *Springer, Berlin/London*.

[27] Allmendinger E, "Submaersible vehicle system design," *SNAME, Jersey City*, 1990.

[28] Sutton R, Roberts G, "Advances in unmanned marine vehicles," *IEE control series. The Institution of Engineering and Technology, London*, 2006.

[29] Inzartev AV, "Intelligent underwater vehicles," *I-Tech Education and Publishing, Open Access*, 2009.

[30] Anotonelli G, "Underwater robots: motion and forces control of vehicle-manipulator system," *Springer, Berlin/New York*, 2010.

[31] Wadoo S, Kachroo P, "Autonomous underwater vehicles: modeling, control design and simulation," *Taylor and Francis, London*, 2010.

[32] Newman JN, "Marine hydrodynamics," *MIT, Cambridge*, 1977.

[33] Faltinsen O, "Sea loads on ships and offshore structures," *Cambridge*, 1990.

[34] Bertram V, "Practical ship hydrodynamics," *Elsevier, Amsterdam/London*, 2012.

[35] B. Wang, M. Mihalec, Y. Gong, D. Pompili, and J. Yi, "Disturbance Obserbation-Based Motion Control of Small Autonomous Underwater Vehicles," *ASME Dynamic Systems and Control Conference DSCC2018*, 2018.

[36] The society of Naval Architects and Marine Engineers (SNAME), "Nomenclature for treating motion of a submerged body through a fluid," *Technical and Research Bulletin No.1-5*, 1950.

[37] T. I. Fossen, *Guidance and Control of Ocean Vehicles*. John Wiley and Sons, 1999.

[38] National Aeronautics and Space Administration, "Propeller thrust." `https://www.grc.nasa.gov/www/k-12/airplane/propth.html`, Accessed September 2020.

[39] E. M. Einarsson, "Fault Detection on the BlueROV2 using Multi Model Residuals." Aalborg University Esbjerg, 2019.

[40] ArduSub - Blue Robotics, "Software Components." `https://www.ardusub.com/software/components.html`, 2020.

[41] Khadas, "VIM3." `https://www.khadas.com/vim3`.

[42] Don Hui, Novaspirit Tech, "Khadas VIM3 review and benchmarks." `https://www.youtube.com/watch?v=IdcZfH30spg`.

[43] XSense, "Xsense MTi-1 series." `https://www.xsens.com/products/mti-1-series`.

[44] F. F. Sørensen, K. Schmidt, M. von Benzon, and S. S. Klemmensen, "Stabilization of BlueROV2 in three-dimensional space." Aalborg University Esbjerg, 2018.

[45] S. Pedersen, T. T. Enevoldsen, and E. M. Einarsson, "Model comparison of a videoray pro 4 underwater rov," *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 899–904, 2018.

[46] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, ch. 7. Pearson, 7. ed., 2015.

[47] A. Urquizo, "Pid controller—wikipedia, the free encyclopedia." https://commons.wikimedia.org/wiki/File:PID_en.svg, 2011 (Accessed October 2020.

[48] B. Douglas, "State space, part 4: What is lqr control?." https://se.mathworks.com/videos/state-space-part-4-what-is-lqr-control-1551955957637.html.

[49] A. V. Oppenheim and G. C. Verghese, *Signals, Systems Inference*, ch. 4. Prentice Hall, 1. ed., 2010.

[50] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Autonomous Ground Vehicles*, ch. 4. Artech House, 1. ed., 2011.

[51] Brian Douglas, Mathworks, "Understanding Sensor Fusion and Tracking, Part 2: Fusing a Mag, Accel, and Gyro to Estimate Orientation." https://se.mathworks.com/videos/sensor-fusion-part-2-fusing-a-mag-accel-and-gyro-to-estimate-orientation-1569411056638.html.

[52] British Geological survey, "The Earth's Magnetic Field: An Overview." http://www.geomag.bgs.ac.uk/education/earthmag.html.

[53] D. Simon, "Kalman filtering," *Embedded Systems Programming*, 2001.

[54] A. Antoniou and W.-S. Lu, *Practical Optimization: Algorithms and Engineering Applications*, ch. 1,2,10,11,12,13,15,16. Springer, 1. ed., 2007.

[55] T. V. Mikosch, S. I. Resnick, and S. M. Robinson, *Numerical Optimization: Springer Series in Operations Research and Financial Engineering*, ch. 1,2,8,10,11,13,16,18. Springer, 2. ed., 2006.

[56] S. Boyd and L. Vandenberghe, *Convex Optimization*, ch. 1,4,9,10,11. Cambridge University Press, 7. ed., 2009.

[57] Rossiter J. A., *Model-based Predictive Control: A Practical Approach*. CRC Press, 1. ed., 2003.

[58] J. M. Maciejowski, *Predictive Control: with Contraints*. Prentice Hall, 1. ed., 2002.

[59] X. Yang, G. Liu, A. Li, and L. Van Dai, "A Predictive Power Control Strategy for DFIGs Based on a Wind Energy Converter System," *Energies*, vol. 10, p. 1098, 07 2017.

[60] R. F. Stengel, *Optimal Control and Estimation*. Dover, 1. ed., 1994.

[61] L. Beal, D. Hill, R. Martin, and J. Hedengren, "Gekko optimization suite," *Processes*, vol. 6, no. 8, p. 106, 2018.

[62] S. Lucia, T. Finkler, and S. Engell, "Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty," *Journal of Process Control*, vol. 23, no. 9, pp. 1306 – 1319, 2013.

[63] L. A. Wächter, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 2006.

[64] P.R. Amestoy and I. S. Duff and J. Koster and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

Appendix

Appendix A

Fault Detection on the BlueROV2 using Multi-Model Residuals - Linear Model

The [44] linearised model:

$$\mathbf{A}_{en} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3.94 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -7.01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5.69 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -15.4 & -9.44 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -12.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -17.97 \end{bmatrix} \tag{A.1}$$

$$\mathbf{B}_{en} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 5.84 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5.84 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5.84 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.63 & 0 & 36.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1.34 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 64.6 \end{bmatrix} \tag{A.2}$$

Appendix B

Full Surge Model Validation Experiments Figures

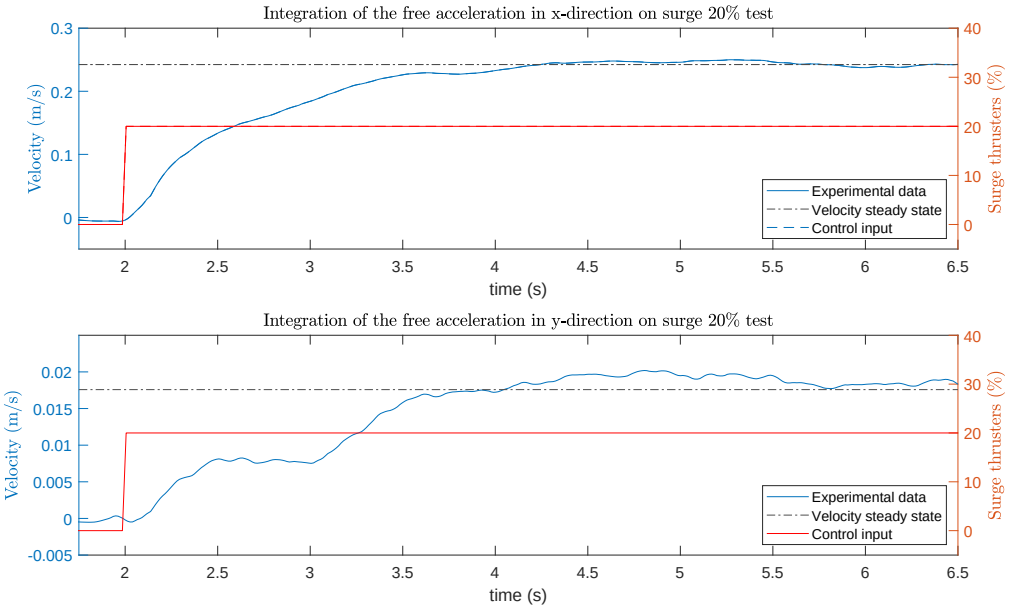

Figure B.1: Surge velocity in x and y direction at 20% thrust.

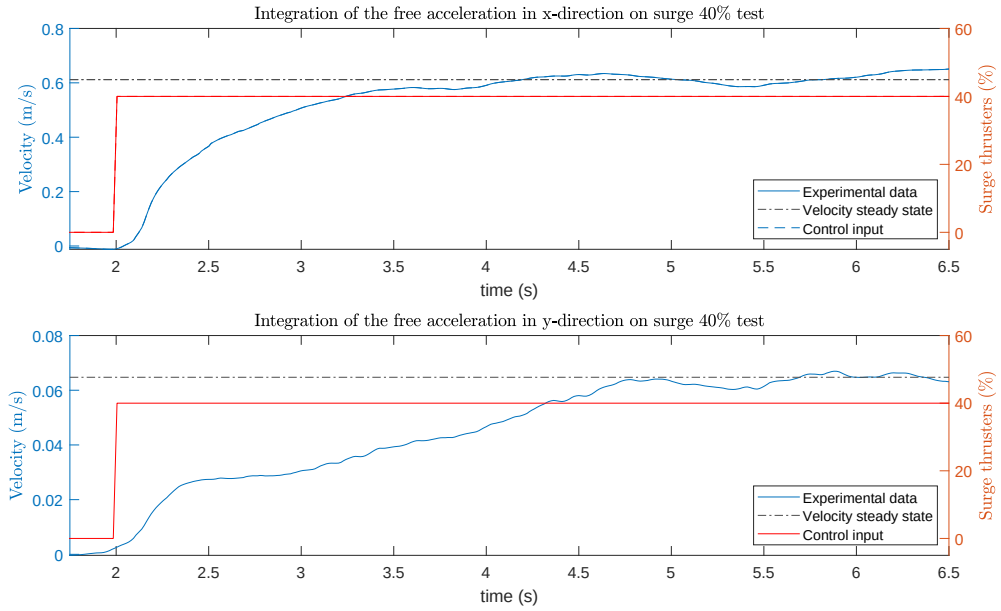

Figure B.2: Surge velocity in x and y direction at 40% thrust.

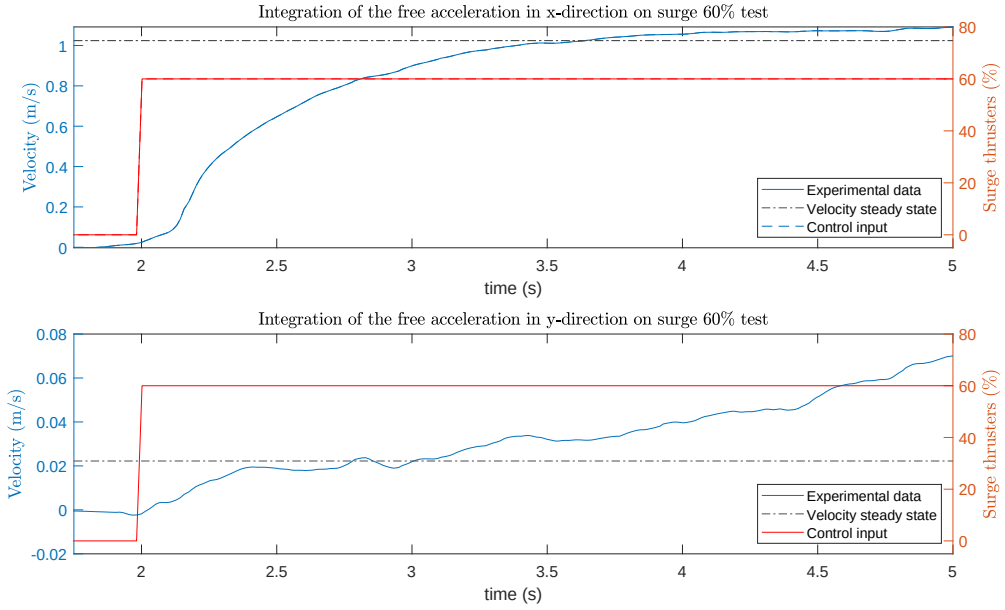

Figure B.3: Surge velocity in x and y direction at 60% thrust.

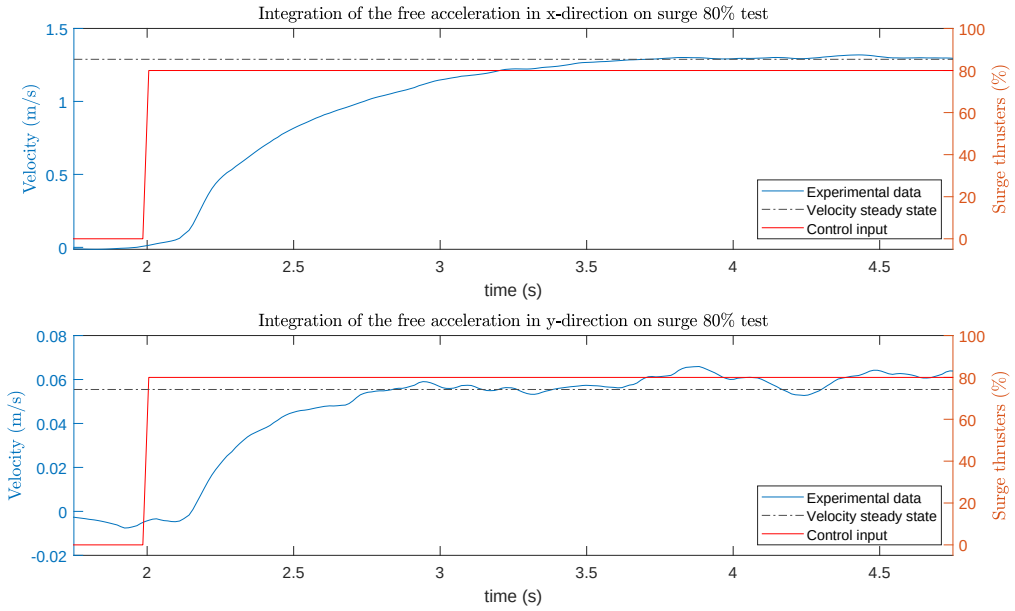

Figure B.4: Surge velocity in x and y direction at 80% thrust.

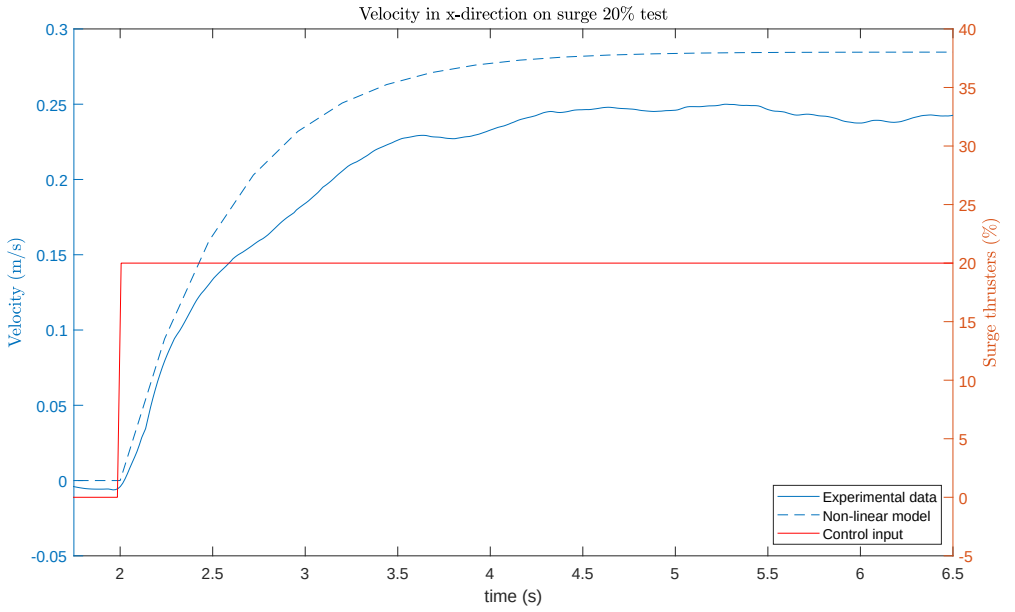

Figure B.5: Non-linear and experiment velocities at 20% thrust.

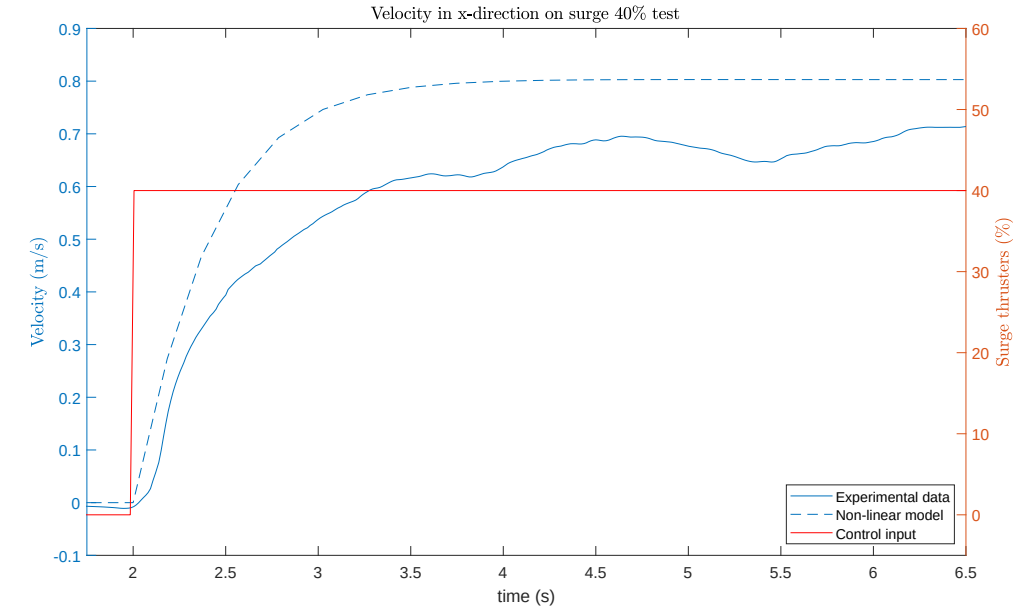

Figure B.6: Non-linear and experiment velocities at 40% thrust.

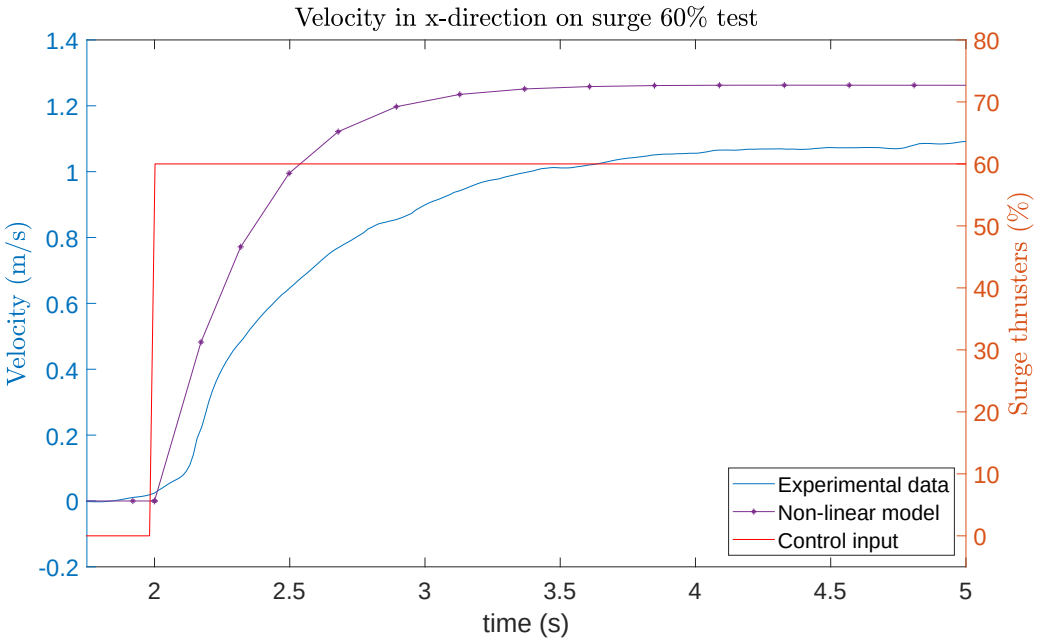

Figure B.7: Non-linear and experiment velocities at 60% thrust.

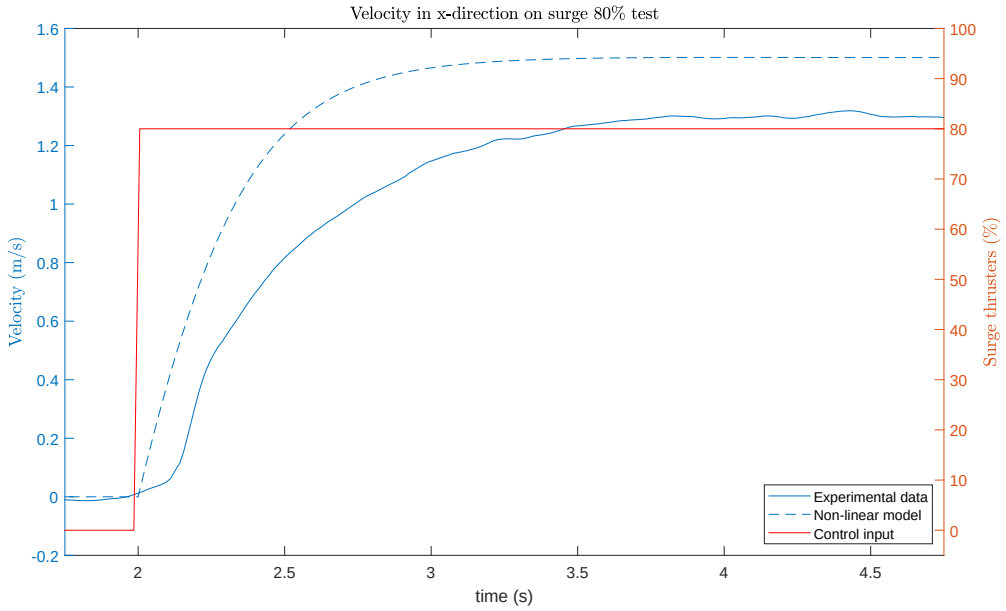

Figure B.8: Non-linear and experiment velocities at 80% thrust.

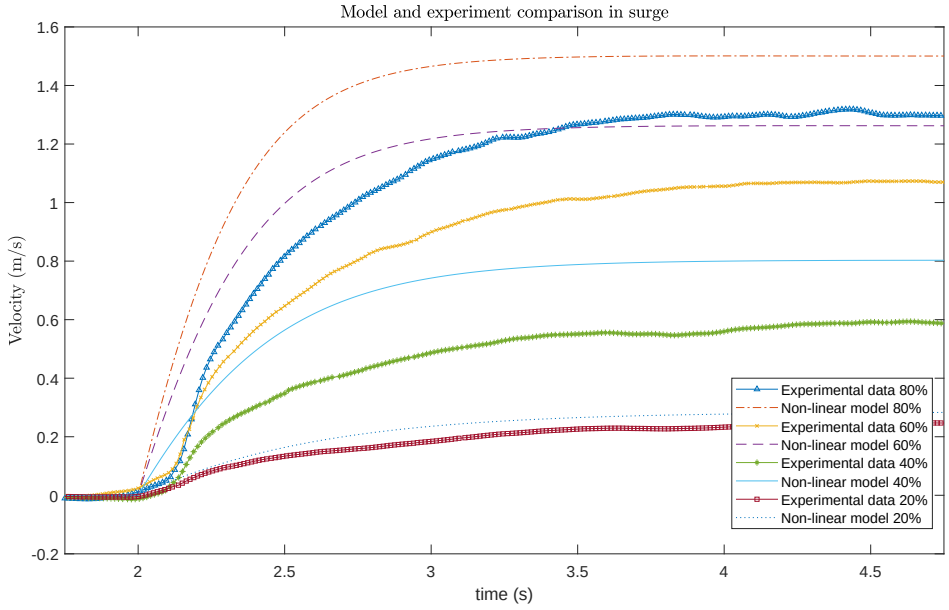

Figure B.9: Non-linear and experiment velocities from 20% thrust to 80%.

Appendix C

Robust Multi-stage NMPC

NMPC has received much attention from both the industry and academics, for the process control. Most commonly, the feedback enters the controller by re-initializing the optimization problem and setting the newest measurement as the initial states. NMPC model accuracy heavily influences its stability and performance. Robust NMPC aims to overcome this limitation. [62]

The main idea behind the Multi-stage NMPC is to take into account that a new measurement will be available at future time samples. Multi-stage NMPC includes standard and min-max NMPC. If the assumption is made that the uncertainty can be modelled by a scenario tree as seen in figure C.1, then a multi-stage NMPC is a promising solution for the robust NMPC, because it computes the optimal closed-loop feedback policy over a finite prediction horizon. If the assumption is wrong, the multi-stage NMPC calculates an approximation of the feedback policy that can be very good, although a robust constraint satisfaction cannot be guaranteed for the values of uncertainty that are not in the scenario tree. The downside of the multi-stage NMPC is that the size of the optimization problem grows exponentially with the increase of the prediction horizon. [62]

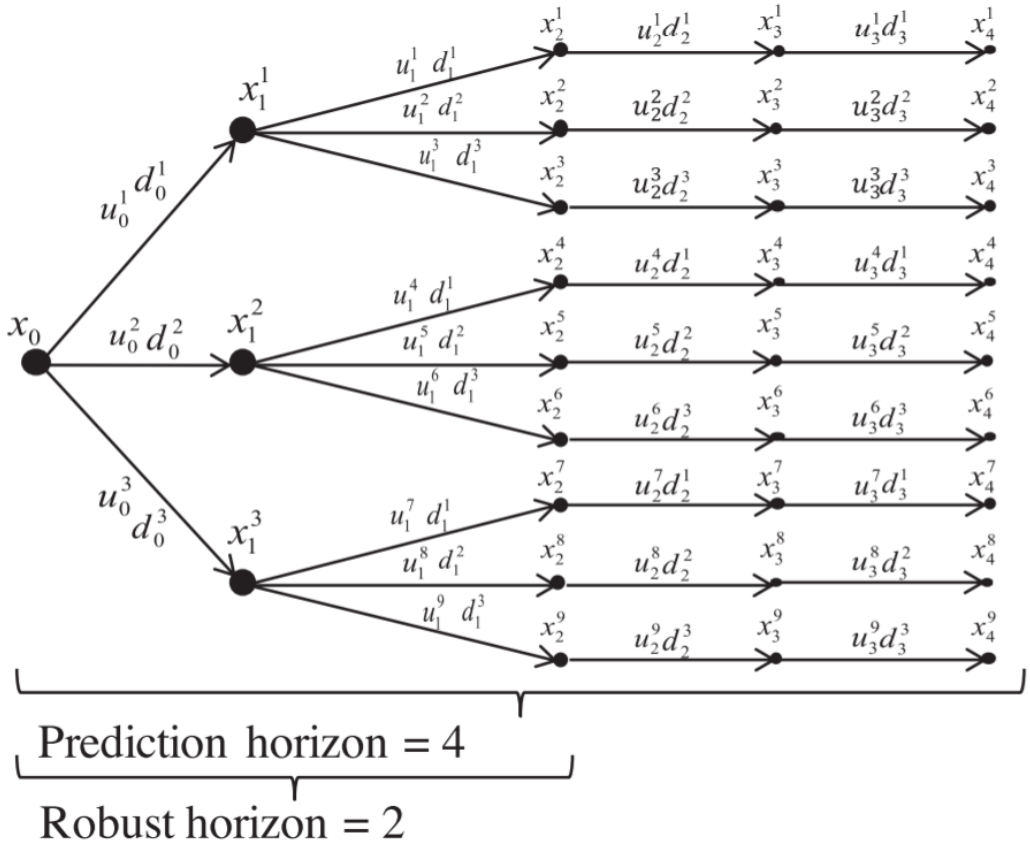

Figure C.1: Scenario tree representation of the evolution of the process for multi-stage NMPC with robust horizon [62]

Each branching of the node in figure C.1 represents the effect of an unknown uncertainty(Disturbance and model error) together with the chosen control input. It is this formulation that allows taking into the account that a new measurement will be available at the future time samples. Therefore the control variables can be seen as recursive variables that are able to deal with the effect of model inaccuracy and disturbances. [62] Each path from the root node x_0 to a leaf node is called a scenario. Even though the tree branches at each stage, it does not mean that the uncertainty changes at each time sample. Instead, it only means that if the uncertainty is not known at sample k, it also is not known at sample $k+1$. Lastly, the control inputs at the same node all have to be equal to model the real-time decision problem, because the control inputs cannot anticipate the future. This restriction is imposed by the $non-anticipativity$ constraints.

The discrete time formulation for the scenarion tree setting can be assumed to be [62]:

$$x_{k+1}^{j} = f(x_k^{p(j)}, u_k^{j}, d_k^{r(j)}), \tag{C.1}$$

where each state x_{k+1}^{j} is dependent on its parent state $x^{p(j)_k}$, its relevant control input u_k^{j} and its realization r of the corresponding uncertainty $d_k^{r(j)}$ at stage k. For

simplicity, the uncertainty and thus, the scenario tree are assumed to be uniform. [62]

Now when the needed notation has been shown, the optimization problem that needs to be solved can be written as follows [62]:

$$\min_{u_k^j, \forall (j,k) \in I} \tilde{J} \tag{C.2}$$

subject to [62]:

$$x_{k+1}^j = f(x_k^{p(j)}, u_k^j, d_k^{r(j)}), \forall (j,k) \in I, \tag{C.3}$$

$$x_k^j \in \mathbb{X}, \forall (j,k) \in I, \tag{C.4}$$

$$u_k^j \in \mathbb{U}, \forall (j,k) \in I, \tag{C.5}$$

$$u_k^j = u_k^l if x_k^{p(j)} = x_k^{p(l)}, \forall (j,k), (l,k) \in I, \tag{C.6}$$

Where $\tilde{J}$ is defined as $\tilde{J} = (\sum_{i=1}^{N} (\omega_i J_i)^a)^{1/a}$. J_i is the cost of each one of the scenarios S_i and ω_i is its probability. The following equation defines the cost of each scenario [62].

$$J_i = \sum_{k=0}^{N_p - 1} L(x_{k+1}^j, u_k^j), \forall x_{k+1}^j, u_k^j \in S_i, \tag{C.7}$$

The $L(x_{k+1}^j, u_k^j)$ is the stage cost and N_p is the prediction horizon. The non-anticipation constraint in C.7 means that all the decisions u_k^j that branch at the same parent node $x_k^{p(j)}$ must be equal. General formulation yields that if $\alpha = 1$ is chosen the optimization problem represents multi-stage NMPC. If $\alpha = \infty$ and all the probabilities ω_i are equal, then it is a closed loop min-max NMPC. Then if also $N = 1$, it turns into a regular NMPC [62].

$$\dot{x} = \Phi(x, u, d) \tag{C.8}$$

$$x_{k+1}^j = x_k^{p(j)} + \triangle \, t(\Phi(x_{k+1}^j, u_k^j, d_k^{r(j)})) \tag{C.9}$$

$$\min_{x^{opt}} f(x^{opt}) \tag{C.10}$$

subject to:

$$x_l \leqslant x^{opt} \leqslant x_u, \tag{C.11}$$

$$b_l \leqslant Ax^{opt} \leqslant b_u, \tag{C.12}$$

$$c_l \leqslant c(x^{opt}) \leqslant c_u, \tag{C.13}$$

$$x^{opt} = x[x_0, x_1^1, x_1^2, ..., x_{N_p}^N, u_1^1, u^2]_1, ..., u_{N_p}^N]^T \tag{C.14}$$

$$\min_{x^{opt}} \sum_{i=1}^{N} \omega_i \sum_{k=1}^{N_p} [(x_{k+1}^j - x_s), Q(x_{k+1}^j - x_s) + (u_k^j - u_s)R(u_k^j - u_s) + \Delta u_k^{'} R_\Delta \Delta u_k], \forall x_{k+1}^j, u_k^j \in \mathbb{S}_i \tag{C.15}$$

subject to:

$$x_k^j \in \mathbb{X}, \forall (j,k) \in I, \tag{C.16}$$

$$u_k^j \in \mathbb{U}, \forall (j,k) \in I, \tag{C.17}$$

$$u_k^j = u_k^l if x_k^{p(j)} = x_k^{p(l)}, \forall (j,k), (l,k) \in I, \tag{C.18}$$

$$x_{k+1}^j = x_k^{p(j)} + \triangle\, t(\Phi(x_{k+1}^j, u_k^j, d_k^{r(j)})), \forall (j, k+1) \in I \tag{C.19}$$