Integration of Model Predictive Control, Payload Deployment, and Computer Vision on the BlueROV2 for Enhanced Underwater Autonomy

Mechatronics, Advanced Course
MF2059

Authors:
Awad AlNasrallah
Edvin Aretorn
Oskar Classon
Erik Ebbesen
Albin Gunnarsson
Jacob Holst
Jesper Knobe
Reman Soryani
Simon Spång

December 15, 2024

Abstract

This report documents the development and implementation of a Model Predictive Control (MPC) framework on the BlueROV2 (BROV) platform to improve its autonomous underwater capabilities. Conducted as part of the Mechatronics Capstone Course at the Royal Institute of Technology (KTH), the project integrates advanced control systems, a payload deployment mechanism, and computer vision into a cohesive system for underwater operations.

The MPC was first validated in a Python environment and subsequently tested in a Unity-based simulations, with the help of Swedish Maritime Robotics Centre (SMaRC), where it demonstrated effective integration with mission planning and state estimation. Real-world testing on the BROV revealed its practical viability, with results highlighting acceptable stability, and performance. A custom Underwater Locator Beacon (ULB) deployment mechanism was successfully tested.

Attempts to implement a computer vision pipeline for underwater image enhancement and 3-Dimensional (3D) reconstruction faced significant challenges due to poor visibility conditions, which limited its effectiveness. This underscores the difficulties of underwater imaging and emphasizes the need for advanced techniques tailored to such environments.

The results demonstrate the feasibility of MPC for underwater navigation and validate the systems core functionalities, while also identifying key areas for future improvement, including model refinement and robust solutions for underwater imaging challenges.

Keywords: Model Predictive Control, BlueROV2, Autonomous Underwater Vehicles, Computer Vision, Image Enhancement, 3D Reconstruction, SMaRC, Unity Simulations, Payload Deployment, ULB Deployment Mechanism, Digital Twin.

Acknowledgements

We would like to thank all the people involved in helping forming this project into what it is. First and foremost, our stakeholders Marina Rantanen Modeer, Louise Fuchs and Oscar Hermansson at SAAB for support and providing the necessary hardware. KTH and our examiners for providing us with a budget and support throughout the entire process.

The good people at SMaRC, including Mart Kartasev, Ignacio Torroba Balmori and Özer Özkahraman, for not only giving us valuable advice throughout the entire process, but also providing us with a testing environment in the form of a pool and their own simulation code base in Unity.

Finally, we would also like to thank Professor Lei Feng, and PhD students Nils Jörgensen and Kaige Tan for their support and meaningful discussions regarding the MPC implementation.

We would also like to acknowledge the use of generative AI in the form of ChatGPT to not only help rapidly develop code, but also help refine the report.

Contents

SUMMARY OF WORK DISTRIBUTION

Name	Chapter(s)	Notes
Jacob Holst	4.2 4.3.3 6.2.3 7.1 8.1 8.2	
Simon Spång	1.4, 4.4.1, 6.1, 7.4, 8.5	
Oskar Classon	4.5.4, 4.1.2, 4.5, 6	
Albin Gunnarsson	2.2, 3.2, 3.3, 4.1, 4.1.1, 4.1.3	
Erik Ebbesen	4.5, 7.2, 8.2, 8.3, 8.4, 8.7, 8.9	
Jesper Knobe	1.1, 1.2, 1.3, 1.4, 2.3, 4.7, 5, 6.5, 7.5, 7.6, 7.8, 8.6	
Awad AlNasrallah	1.5, 2.1, 4.3.1, 4.4.2, 6.1, 7.3, 8.4	
Reman Soryani	1.2, 3.3.1, 4.3.2, 6.4, 7.5, 7.8, 8.8	
Edvin Aretorn	4.4.1, 4.6, 6.2.1, 6.2.2, 6.3, 7.4, 7.7	

List of Figures

List of Tables

List of Symbols

The next list describes several symbols that will be later used within the body of the document

δ ROV volume

$\eta = [x, y, z, \phi, \theta, \psi]$ World-frame position vector

$\hat{u}_k$ Predicted input vector at time step k

$\hat{u}_k^*$ Optimal predicted input vector at time step k

$\hat{x}_k$ Predicted state vector at time step k

$\hat{x}_k^*$ Optimal predicted state vector at time step k

κ_t Lumped constant terms for linearized model at time t

$\mathbf{x}_p = [x_p, y_p, z_p]$ Way point vector

$\mathbf{x}_{ref} = [x_{los}, y_{los}, z_{los}, \phi_{los}, \theta_{los}, \psi_{los}]$ Line-of-Sight reference vector

$\nu = [u, v, w, p, q, r]$ body-frame velocity vector

ρ Water density

τ Thruster force input vector

τ_{delay} Time delay

τ_{tet} Tether force vector

$a(t)$ Acceleration profile generated by trajectory planner

A_c Continuous time state matrix

A_t Discrete time state matrix linearized at time t

B Buoyancy force

B_c Continuous time Input matrix

B_t Discrete time input matrix linearized at time t

$B_{delay,t}$ Discrete time delayed input matrix linearized at time t

C Output matrix

$C(v)$ Corioliscentripetal matrix

$C_A(v)$ Corioliscentripetal added mass matrix

$C_{rb}(v)$ Corioliscentripetal rigid body matrix

D Damping linear matrix

$D(v)$ Damping matrix

$D_n(v)$ Damping non-linear matrix

$F(V_i)$ Thruster individual force

g Gravitational constant

$g(\eta)$ Restoring forces vector

h Sampling time

I_x Rotational inertia around x axis

$J(\eta)$ World to body velocity transform matrix

J_1 World to body translational velocity transform matrix

J_2 World to body rotational velocity transform matrix

k Sampling instance

M Inertia matrix

m Mass of BlueRov2

M_A Inertia added mass matrix

M_{rb} Inertia rigid body matrix

N MPC Horizon length

Q Refernce deviation cost matrix

R Control increment cost matrix

$r(t)$ Reference profile generated by trajectory planner

r_s Reference point

T Thruster to body transfrom matrix

t Time

$v(t)$ Velocity profile generated by trajectory planner

V_i Thruster individual control input

W Gravitational force

x_b Buoyancy moment arm in x direction

X_u Linear damping in x axis

$X_{\dot{u}}$ Added mass inertia in x axis

$\delta\hat{u}_k$ Predicted input increment vector at time step k

$\delta\hat{u}_k^*$ Optimal predicted input increment vector at time step k

$\mathbf{r}_k$ Reference point at time step k

$\mathbf{u}_k$ Input vector at time step k

$\mathbf{x} = \left[\eta, \nu\right]$ State vector

$\mathbf{x}_k$ State vector at time step k

$\mathbf{x}_t = \left[\eta_t, \nu_t\right]$ State vector at time t

$\mathrm{R}_{DVLdisplacement}$ DVL distance from CoG

v_{body} Linear body velocities

v_{DVL} Linear DVL velocities

ρ_L Radius of Line-of-Sight-sphere

ρ_s Radius of sphere-of-acceptance

Glossary

3D 3-Dimensional. i, iii, vii, 1, 3, 4, 11, 12, 46, 58, 59, 66

AI Artificial Intelligence. ii

AUV Autonomous Underwater Vehicle. v, 1, 10, 30, 36, 67, 71, 72

BROV BlueROV2. i, vii, ix, 1, 3, 4, 11–18, 21–23, 27, 30, 34, 35, 39, 44, 46, 47, 60, 61, 63, 64, 67–72

BT Behaviour Tree. 20

CoG Centre of Gravity. xii, 34

DoF Degrees of Freedom. 14, 15, 18, 35

DVL Doppler Velocity Logger. vii, 3, 13–15, 18, 34, 39, 63, 65, 66, 71

FHOCP Finite Horizon Optimal Control Problem. 32

FSM Finite State Machine. 19, 20

GAN Generative Adversarial Network. 10, 11, 46, 66

GCS Ground Control Station. 18

GPS Global Positioning System. 71

HMI Human Machine Interface. 18

IMU Inertial measurement unit. 14, 15, 18, 34, 65, 71

KTH Royal Institute of Technology. i, ii, 1

LOS Line-of-Sight. iv, vii, x, xii, 37–39, 49, 53, 55, 56, 62–64, 70

LTV Linear Time Varying. 8, 69

MIMO Multi-Input Multi-Output. 6

MPC Model Predictive Control. i–iv, vii, xi, 1–8, 12, 13, 17, 18, 21, 22, 30–34, 36, 39–41, 43, 45, 47, 49, 51–53, 56, 60–64, 68–70

MVS Multi-View Stereo. 11, 46

NED North, East, Down. 42

PID Proportional-Integral-Derivative. 1

PWM Pulse Width Modulation. 18, 27, 35, 36, 43, 46, 62, 69, 70

ROS Robot Operating System. iii, iv, vii, 12, 15, 16, 18–21, 34, 43, 44, 46, 63

ROV Remotely Operated Vehicle. x, 1, 12, 37–39, 57, 60, 67, 68, 71, 72

RPM Revolutions Per Minute. 43, 44

RUF Right, Up, Forward. 42

SfM Structure from Motion. 11, 46

SLAM Simultaneous localization and mapping. 71

SMaRC Swedish Maritime Robotics Centre. i, ii, 1, 13, 41, 43

SOTA State Of The Art. iii, 5, 6, 11, 37

SQP Sequential Quadratic Programming. 7

ULB Underwater Locator Beacon. i, iv, vii, 43, 45–47

ZOH Zero-Order hold. 31

1 Introduction

The following section gives an introduction to the project. It includes the background to the project, the project description and requirements and finally, a readers guide.

1.1 Background

Underwater archaeology has become an increasingly popular discipline in the past two decades due to commercial and industrial interests in the exploration and preservation of the seabed [1]. However, the exploration and investigation of underwater sites presents an array of challenges that differ from their terrestrial counterpart. Sites are often situated in deep water and can be inaccessible and dangerous to human divers due to for example high pressures, cold temperatures and strong currents [2]. Technological advancements have allowed underwater archaeologists to overcome some of these obstacles by utilizing underwater vehicles such as Remotely Operated Vehicle (ROV) for deep-sea archaeological missions.

The project explores the possibilities of the BlueROV2 (BROV), an affordable and widely-used ROV platform, to be implemented as an AUV to autonomously navigate around an underwater site while simultaneously deploying underwater payloads and capturing images of the site for 3-Dimensional (3D) mapping. Currently, the BROV has path-following capabilities using Proportional-Integral-Derivative (PID) control. The stakeholder mentioned that this control algorithm lacks precision and robustness, hence the implementation of Model Predictive Control (MPC) was explored. MPC was also chosen since it has shown promising results for autonomous underwater applications [3] combined with the stakeholder's desire to test its performance.

1.2 Project Description

The nine month long project has been conducted at Royal Institute of Technology (KTH) as a part of the *Mechatronics Capstone Course*. The platform used in this project has been the BlueROV2 Heavy configuration. The stakeholder of the project is SAAB underwater systems, who requested Model Predictive Control (MPC) to be implemented on the platform combined with image enhancement, 3D-reconstruction of underwater sites, an information model of the entire system and development of a manipulator for underwater usage. A collaboration has also been started together with the Swedish Maritime Robotics Centre (SMaRC) at KTH which will provide the project with digital and physical resources and assistance.

The project will provide the stakeholder with valuable insights into technology that can be used for ROVs and AUVs, especially focusing on the aspect of Model

Predictive Control, as the advancements in the field combined with the increased available computing power in embedded processors now make control algorithms with heavier computational load, such as MPC, more viable for real-time application. [4][5][6].

This is a research-orientated project with soft requirements which does not diminish the value of the final product [7]. The project is meant to serve as a vessel to increase the knowledge within the field, and any effort that yields results that can serve as a foundation for future work will hold significant value for the stakeholder. Nevertheless, the team has made sure to establish clear requirements to ensure focused and purposeful efforts throughout the project.

The team consists of final year master students in mechatronics with a background in applied controls system design, embedded systems and interdisciplinary systems. Since the project involves several disciplines that need to be integrated into one platform as a complete system, the profile of the team suits the task at hand.

1.3 Requirements

The requirements for this project are categorized into two main groups: **stakeholder requirements** and **technical requirements**.

- **Stakeholder requirements** reflect the needs and expectations set by SAAB, the project's primary stakeholder.
- **Technical requirements** are defined by the project team to ensure the system fulfils the stakeholder's needs.

The following conventions are used to differentiate between types of requirements:

- **Shall**: Denotes mandatory requirements that must be met.
- **Should**: Refers to desirable but non-essential requirements.
- **Will**: Provides additional contextual information to clarify the project's objectives.

1.3.1 Stakeholder Requirements

The following stakeholder requirements must be met by the project:

- The robot shall use MPC for navigation.
- The project shall assess the viability of MPC for underwater navigation.
- The systems information flow shall be modelled.
- A mechanical system shall be designed, developed and implemented on the BROV.
- Images should be captured and enhanced.
- 3D image reconstruction should be utilized to reconstruct key objects.
- A BlueROV2 will be used to conduct the study.
- A Doppler Velocity Logger (DVL) will be used to measure the state of the BROV
- The work will focus on underwater archaeology applications.
- The project will aim to transition to autonomous navigation for underwater robots.

1.3.2 Technical Requirements

The technical requirements are as follows:

- A simulation of the BROV shall be developed.
- The simulation will be used to verify the viability of the controller and other system components.
- A safety mechanism should be in place to stop the robot during testing.

1.4 Delimitations

Due to the complexity of the system and the environment of the sea, delimitations have been made to create a more feasible project. Since the main scope of this project is to explore the design and implementation of MPC as the control strategy for the BROV, together with the implementation of image enhancement and 3D-reconstruction for underwater sites, the following delimitations have been made.

- All development and most testing will be done in a pool to reduce the influences of the environment. Influences such as currents and bad visibility.
- The system parameters describing the BROV will not be evaluated in this report. The parameters will be acquired from other reports using the same platform.
- The system model used was developed by another independent study and has also been slight reduced, primarily removing the tether force applied on the system.
- All added software and control will be executed on the topside computer since these requires a lot of computational resources.
- The image enhancement algorithm and 3D-reconstruction algorithm will not be developed in this project. Existing algorithms for this purpose will be implemented. The implementation will be on the topside computer and not on the BROV.

1.5 Readers Guide

This section provides an overview of the structure of the report.

The Introduction sets the stage for the report by presenting the background of the project, outlining the problem and listing out the requirements. This section is essential to understand the context in which the report was written and the main focus of the project. Then comes the State Of The Art (SOTA) section, where previous research and relevant background information are summarized. This section focuses on providing the reader insight into key concepts used such as MPC as well as image enhancement and reconstruction, which are essential for understanding the report.

The Methodology section follows, detailing the approach used to carry out the project. This includes, the engineering methods employed, as well as the group organization and management. The section also includes the tools (hardware and software) required to replicate the project. Then comes the main body of the report, which goes in depth into the technical aspects of the project explaining how all the work done was implemented to reach the goals set and satisfy the requirements of the project. After which, the approach used to collect data, conduct experiment, verify and validate the results are presented in the Verification and validation section.

The Results section then presents the findings of the study. Here, the data collected from the conducted tests is provided without interpretation, allowing the reader to see the outcomes of the work. Afterwards, in the Discussion and Conclusions section, the results are analysed and interpreted by the team. This section explores the implications of the findings, addresses any limitations of the study and draws relevant conclusions.

Finally, there is a section dedicated to future work, where possible improvements of the current project are proposed. The report then concludes with a References section where all the sources cited throughout the report are listed, ensuring proper attribution to prior research and materials used in the preparation of the document and an Appendices section, which includes supplementary materials, not necessary for understanding the report, but which can provide additional insight for the interested reader.

2 State Of The Art

2.1 Model Predictive Control

Model Predictive Control or MPC is an optimal control strategy where, using a model of the system and a current estimate of the state, the optimal control actions can be calculated.

Unlike more traditional control methods, such as PID controllers, optimal controllers focus on minimizing the cost of a certain function. A few of the advantages of MPC is that it can explicitly handle constraints on inputs, states and time delays, it can also easily handle Multi-Input Multi-Output (MIMO) systems. A general formulation of the optimization problem can be seen in equation 2.1. This is however, far from the only formulation that exists.[8]

$$\begin{aligned}
&\min_{x_k, u_k} \quad \sum_{k=0}^{T-1} \ell(x_k, u_k) + \ell_T(x_T) \\
&\text{subject to:} \\
&\quad x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, T-1 \\
&\quad x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \quad k = 0, \ldots, T-1 \\
&\quad x_T \in \mathcal{X}_T \\
&\quad x_0 = x_t
\end{aligned} \tag{2.1}$$

Over a specified time horizon T, the problem is solved for the optimal sequence of states $\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_T$ and inputs $\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_T$ that minimize the accumulated cost over the horizon defined by $\sum_{k=0}^{T-1} \ell(x_k, u_k) + \ell_T(x_T)$ and subject to the constraints in equation 2.1.

Here, $\ell(x_k, u_k)$ is the stage cost, i.e. the cost at the specific time step, whereas $\ell_T(x_T)$ is the terminal cost or the cost of the state at the end of the prediction horizon T. As for the constraints on the state x and input u they must belong to the domain of allowable states and inputs, $\mathcal{X}$ and $\mathcal{U}$ respectively. Furthermore, x_T belongs to a terminal set, $\mathcal{X}_T$. Naturally, we also have that x_0 must be equal to the state at the start of the optimization (the estimated current state).

In standard form, the problem to be solved can be formulated as in equation 2.2, where the system is assumed to have linear dynamics and constraints and the cost function is chosen to be quadratic with semi-positive definite weight matrices Q, R, Q_T. This formulation of the problem is convex quadratic and can be solved reliably and quickly.

$$
\begin{aligned}
\min \quad & \sum_{k=0}^{T-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right) + x_T^\top Q_T x_T \\
\text{subject to:} \\
& x_{k+1} = Ax_k + Bu_k, \quad k = 0, \ldots, T-1 \\
& M_x x_k \leq m_x, \quad k = 0, \ldots, T-1 \\
& M_u u_k \leq m_u, \quad k = 0, \ldots, T-1 \\
& x_T \in \bar{X}_T \\
& x_0 = x_t
\end{aligned}
\tag{2.2}
$$

At every time step, the optimization problem is solved and the optimal control and state sequences are obtained after which, only the first control input, $\hat{u}_0$, will be applied. This process is repeated for each time step and therefore results in a feedback policy. In this case, the cost matrices Q and R can be tuned to achieve desired behaviour. For example, increasing Q puts more emphasis on driving the state x_k to 0 which will result in a faster system response. Conversely, increasing R makes inputs u_k more expensive which will result in more conservative control.

2.1.1 MPC for Non-linear Systems

It is possible to extend the formulation in equation 2.1 for non-linear systems. Using the non-linear model, the full system behaviour can be captured and the MPC will be able to accurately predict future states. However, since the optimization problem will now contain non-linear constraints, the problem will lose its convexity. This drastically increases solving difficulty and makes the problem more computationally heavy. Moreover, non-linear optimization problems are most commonly solved using iterative methods such as the Sequential Quadratic Programming (SQP) method. Methods of this kind require an initial guess, and if the choice of initial guess is bad then the solver might converge to a local optimum. Global optimality can therefore not be guaranteed.[9]

To combat these issues, a few alternatives exist [10]. For example, if the operating point is known before hand, then simple linearization can be used to linearize the model around the operating point and then formulate the problem as a convex quadratic problem as in equation 2.2. In this case, A and B will instead represent the linearized model around the chosen equilibrium point. This significantly simplifies solving, but if the operating point changes drastically then the model can no longer accurately describe the system.

Another method is what Matlab calls adaptive MPC [11]. This method is quite common in the literature, but is usually referred to as successive linearization [12] [13] [14]. The idea is to, at every time-step, linearize the model at the current operating point and then use the linearized model to compute the optimal control inputs over the prediction horizon. This method provides a better approximation of the non-linear model than the simple linearization method described above, but could also suffer from the same issues if the prediction horizon is too long, since the model could end up far away from the linearization point.

This same concept could be taken a step further, if the operating points over the prediction horizon are known beforehand, then the non-linear model can be approximated over the prediction horizon as a sequence of linear models obtained by linearizing the non-linear model around this so called "nominal trajectory". It is then possible to formulate the MPC problem as a Linear Time Varying (LTV) system where the model parameters A and B change over the prediction horizon. This provides an even better representation of the non-linear system, but requires that the operating points are known before hand. [15]

2.1.2 MPC for Reference Tracking

The MPC problem for reference tracking is typically formulated according to equation 2.3.

$$
\begin{aligned}
\min \quad & \sum_{k=0}^{T-1} \left(\Delta\hat{x}_k^\top Q \Delta\hat{x}_k + \Delta\hat{u}_k^\top R \Delta\hat{u}_k \right) + \Delta\hat{x}_T^\top Q_T \Delta\hat{x}_T + \sigma(s) \\
\text{subject to:} \\
& \Delta\hat{x}_{k+1} = A\Delta\hat{x}_k + B\Delta\hat{u}_k, \quad k = 0, \dots, T-1 \\
& M_x(\Delta\hat{x}_k + x_{\text{ref}}) \leq m_x, \quad k = 0, \dots, T-1 \\
& M_u(\Delta\hat{u}_k + u_{\text{ref}}) \leq m_u, \quad k = 0, \dots, T-1 \\
& \Delta\hat{x}_T \in \bar{X}_T \\
& \begin{bmatrix} A & I \\ B & 0 \end{bmatrix} \begin{bmatrix} x_{\text{ref}} \\ u_{\text{ref}} \end{bmatrix} = \begin{bmatrix} 0 \\ r + s \end{bmatrix} \\
& \Delta\hat{x}_0 + x_{\text{ref}} = x_t
\end{aligned}
\tag{2.3}
$$

Here the constraints represent the following: first the dynamic constraints, then the state and input constraints, the terminal set constraint. Reference tracking constraint, which optimizes u_{ref} and x_{ref} for the desired reference. And finally, a variable transformation from Δx to x.

The control input applied is defined as:

$$u_t = \Delta\hat{u}_0 + u_{\text{ref}}. \tag{2.4}$$

An important observation here, is that using this formulation, if the reference will change online, the terminal set will need to be recalculated online. The advantage is that, with reasonable choices of the terminal weight and set, the problem can be guaranteed to be recursively feasible and stable. For more information, the interested reader is referred to other papers such as [8].

While this is a very rigorous formulation of the problem, it is difficult to implement in practice since it is not possible to guarantee that the problem is feasible for all references and initial states. Therefore, a more common and practical approach is used in this project (see section 4.3.1).

2.2 Image Enhancement

For underwater robot exploration using a camera, a common problem is poor image quality. This is mainly due to light absorption and scattering. The light absorption occurs because of the different wavelengths in light. As visualized in Figure 2.1, the light is gradually absorbed corresponding to its wavelength, red light being the longest wavelength and blue the shortest. That is why blue and green are more dominant in underwater images. [16]

In addition to this, scattering is another phenomenon that worsens the quality of underwater images. It is a phenomenon where underwater particles are reflected into the camera and creates blur and poor vision within the image. This is also visually shown in Figure 2.1. Therefore, the purpose of image enhancement is to generate a better quality image of a given image by de-noising and color-correcting for improved visibility. [16]

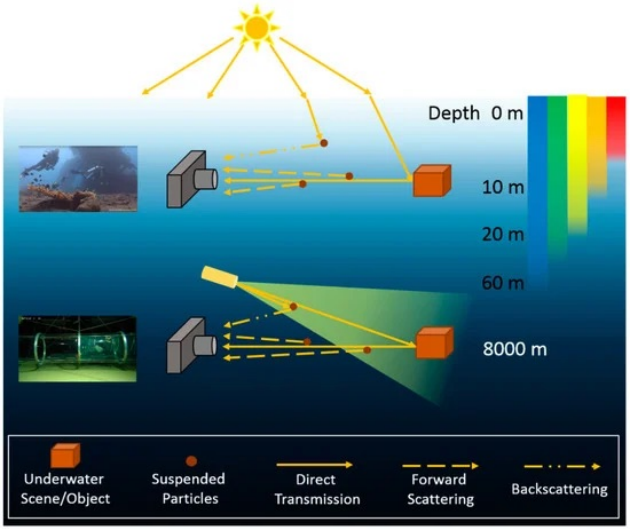


Figure 2.1: Underwater light absorption and scattering

This can be done in many ways. To begin with, there exist physics-based methods to solve this. However, while this physical process is well described theoretically, the model depends on many parameters such as water characteristics, depth and structure of the scene. These factors make recovery of these parameters difficult without simplifying assumptions or field calibration; hence, restoration of underwater images is a non-trivial problem. [17]

However, by using deep learning and only using a camera, this problem can be solved to work in different water settings without the need to purchase additional sensors. Deep learning for image enhancement is a state-of-the-art field, and there exist different methods to approach it. The most popular approach in this context is known as synthesis, which involves generating new content from the given input. Within synthesises, the three most common types are Generative Adversarial Network (GAN), auto-encoders, and diffusion models. All three of these can be used for image enhancement. Their broad advantages and disadvantages can be visualised in Figure 2.2. [18] For autonomous underwater vehicles (AUVs), the desired outcome is for the image enhancement to run in real-time on the onboard computer. Fast sampling is needed while providing high-quality samples, making GAN a common choice for image enhancement for underwater robots. [19].

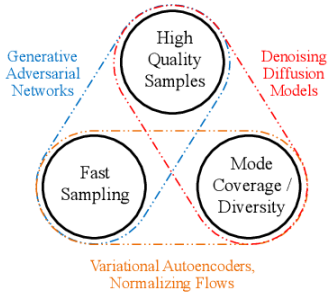


Figure 2.2: The generative learning trilemma

However, a disadvantage with GANs is mode coverage, meaning its ability to adapt in new environments. To solve this, Zhisheng Xiao et al. has developed a combined method with GAN and diffusion model that is claimed to solve the generative learning trilemma. [18] Conclusively, using deep learning for underwater image enhancement is SOTA within this field and can help with further underwater tasks that require clear vision.

2.3 3D Image Reconstruction

In underwater archaeology, a central part is mapping the sites. The harsh underwater environment combined with poor visibility makes mapping very difficult. In this project, the main application of the computer vision is to map and get a clear view of a site on the ocean floor using the BROV on-board camera. 3D image reconstruction is a complex subject that can be solved in multiple ways, for example with stereo cameras. Quite a lot of research has been conducted in reconstruction of smaller indoor scenes and less in large scenes with worse conditions [20].

There are two main methods in the reconstruction from unstructured images, Structure from Motion (SfM) and Multi-View Stereo (MVS). SfM is a photogrammetry method that uses a series of images from different viewpoints and reconstructs the position and viewpoints by matching points in the different images. MVS uses a set of images of known positions and viewpoints to infer the 3D geometry of an object[20].

The advancements in deep learning have come with great improvement for 3D reconstruction from a single 2 dimensional camera without the need for prepossessing to specify angles and distances to the object. These methods train Neural Radiance Fields to find the point cloud representation of the object from unstructured images.[21].

3 Methodology

3.1 Research Process

To conduct the project, the first step was to gather information about the state of the art of the different subsystems for the project. The subsystems in this case being the control algorithm MPC, underwater archaeology, and computer vision.

The second phase of the project consisted of sketching general ideas for the different subsystems for the project. During this phase, the project work was also more thoroughly planned throughout the period. Moreover, it was found that all different subsystems for the project could have been made into their own project. It was therefore decided that the project would mainly revolve around MPC and its implementation on the ROV. For the other subsystems, it was decided to try to implement something that would require less effort so that most of the effort could be put on the MPC.

The third phase consisted of creating a more detailed design for the dropper mechanism in Solid Edge, as well as trying out some open source computer vision software. Since there was little knowledge about Model Predictive Control within the group at the start of the project, the focus during this phase was on learning about MPC and implementing some pre-made libraries in Matlab to get some initial knowledge. An important part of the MPC is that it requires a good model of the system dynamics to function properly; this was also investigated during this phase. How this was solved is described in Section **??**.

The fourth phase of the project consisted of creating a physical prototype of the dropper mechanism. The computer vision part of the project was not worked on during this phase as it required more data from real life driving with the BROV. For the MPC, it was implemented in matlab/simulink to be tested. For simulation of the Robot Operating System network along with the MPC, a simulation was developed in Unity in which communication between the ROS network and a complete 3D visualized and dynamically simulated BROV through the Unity game engine. This was crucial for the development.

The fifth phase consisted of implementing the MPC in python for further testing. Moreover, the ROS network was developed along side the MPC in python to be able to integrate seemlessly at a later stage. The communication between ROS and the Unity simulation was also set up and tested during this stage.

In the sixth phase, the MPC was implemented and tested in the unity simulation. It was found that some path planner was needed. The path planner was created, implemented and tested and was found to work well with some small tweaks.

It should be mentioned that these six phases mainly describe the MPC and its implementation. Throughout all phases, great effort was put in integrating the dropper mechanism and the Doppler Velocity Logger (DVL), as well as researching how to manipulate the BROV in the way we wanted. All of these were crucial steps in the project as a whole and it would not have been possible without them.

3.2 Project Management

The workflow and project management approach adopted encompassed several methodologies and communication strategies to ensure efficiency and adaptability. The team primarily utilized agile methodologies, implementing Jira for task management and scheduling two-week sprints to facilitate regular progress checkpoints and adaptive planning. The V-model was also an integral part of the project management strategy, providing a structured approach to system development that emphasizes the importance of verification and validation processes at each development stage.

Regarding the group structure, a flat hierarchy was maintained, eliminating the role of a traditional project manager. Instead, team members shifted roles depending on the needs of various subsystems, promoting flexibility and comprehensive understanding among team members. Communication within the team and with external entities was strategically planned. Bi-weekly meetings with the stakeholder were implemented, which ensured consistent and structured information exchange. Communication with SMaRC was managed effectively using Slack, facilitating real-time updates and discussions.

Overall, the combination of agile practices, strategic use of the V-model, adaptable group structure, and targeted communication strategies effectively supported the project's goals by fostering an environment conducive to continuous improvement and collaboration.

3.3 Required Hardware/Software

The robot platform on which this project is based is the BlueROV2 Heavy Configuration. It is an open-source and modifiable underwater vehicle that is created and developed by Blue Robotics. The following sections list the necessary components to duplicate this project.

3.3.1 Hardware

The specific hardware specification for this project can be shown in table 3.1. One of

Figure 3.1: Standard BlueROV2 Heavy Configuration

the critical upgrades in this project to the BROV is the inclusion of the Waterlinked DVL A50, which significantly enhances the vehicle's navigational capabilities. It estimates velocity relative to the seabed using acoustic waves. It emits signals from four angled transducers, and by analysing the Doppler effect frequency shift between transmitted and received echoes it can calculate the speed and direction of movement. The used Waterlinked DVL A50 is provided with a built-in Inertial measurement unit (IMU) that provides orientation data [22]. These measurements are combined in a Kalman filter on the DVL to produce highly accurate velocity estimates, crucial for precise positioning and control. The DVL also outputs a dead reckoning estimate of the position. The DVL does not output angular velocities.

The BROV comes standard with a 6-Degrees of Freedom (DoF)IMU and a 3-DoF magnetometer, enabling data gathering of angular rates and compass heading. The BROV that was received from the stakeholder also came equipped with a Ping360 imaging sonar, however, it was deemed not worth using in the scope of this project.

Additionally, the thrusters' configuration in the BROV Heavy enhances its manoeuvrability, allowing for 6-DoF motion.[23] Control and computation are divided between the Navigator Flight Controller and the Raspberry Pi 4B. The Navigator handles low-level operations, such as sensor data acquisition and real-time thruster control, while the Raspberry Pi is responsible for higher-level tasks such as interfacing with external sensors, managing data logging, and coordinating communication with the surface operator. Together, these systems ensure seamless

Table 3.1: BlueROV2 hardware components

Category	Components
Computational devices	Navigator Flight Controller Raspberry Pi 4B (2GB RAM) Top-side computer: Dell Precision 3581 (i9 CPU)
Sensors	IMU 6-DoF (on Navigator) Dual 3-DoF magnetometers (on Navigator) Internal barometer (on Navigator) Integrated leak sensor (on Navigator) Waterlinked DVL A50 Camera (Low-Light HD USB Camera) Pinger360 Sonar
Actuators	Thrusters (8 Blue Robotics T200)
Battery	Lithium-Ion Battery Pack (14.8V)

and efficient operation during complex underwater missions.

3.3.2 Software

The software suite used in this project for the BROV forms the backbone of its operational and development capabilities. Each software component plays a critical role in enabling seamless interaction between hardware and control systems, ensuring the robot's functionality across diverse missions. Table 3.2 categorizes the key software components along with their versions, highlighting their respective purposes within the project.

The firmware, ArduSub, serves as the core software running on the Navigator flight controller. It provides the low-level control required for sensor integration, thruster actuation, and basic operational stability.

The operating systems bridge the computational hardware with mission-specific software. BlueOS, running on the Raspberry Pi, acts as the operational environment for managing onboard systems, network interfaces, and real-time monitoring tools. The top-side computer, Dell Precision 3581, running Ubuntu 22.04 LTS offers a stable platform for executing control algorithms, simulation interfaces, and remote telemetry.

For control and coordination, the ROS serves as a middleware framework,

Table 3.2: BlueROV2 software components

Category	Software	Version
Firmware	ArduSub (BROV firmware)	4.5.0
Operating System	BlueOS (for Raspberry Pi)	1.3.1
	Ubuntu (for top-side computer)	22.04 LTS
Control Software	ROS2	Humble
	Mavros	2.9.0
Simulation	Unity	2023.1.13

facilitating communication between subsystems. ROS2 Humble, combined with MavROS, integrates MAVLink-based telemetry and control channels, ensuring compatibility with ArduSub and other components. These software tools enable a modular and scalable control architecture, supporting both current needs and potential future expansions. An in-depth description of how these are used and integrated will be explained in the implementation section.

In addition, simulation tools are indispensable for validating mission scenarios, testing algorithms, and visualizing robot behaviour. Unity, a widely used simulation and visualization software, allows the development of realistic underwater environments to replicate operational conditions. This enables rigorous testing and troubleshooting before deployment, minimizing the risk of errors during actual missions.

The integration of these software tools not only ensures robust and reliable operation but also supports flexibility in adapting the system for future requirements. Continuous updates and compatibility improvements are vital to maintaining the performance and expanding the capabilities of the BlueROV2 platform.

4 Implementation

To integrate all components of the project into a cohesive mission, the following scenario was developed and is illustrated in Figure 4.1. The operator starts by inputting a waypoint to a location of interest within an underwater site. The BROV then generates a path to that waypoint and travels to that waypoint autonomously using the MPC. Upon reaching the waypoint, the system will drop a pinger, then proceed to the next given waypoint and repeat the process. During this mission, the BROV records a video stream, which is later used for computer vision tasks. These tasks are conducted post-mission due to their substantial computational demands.

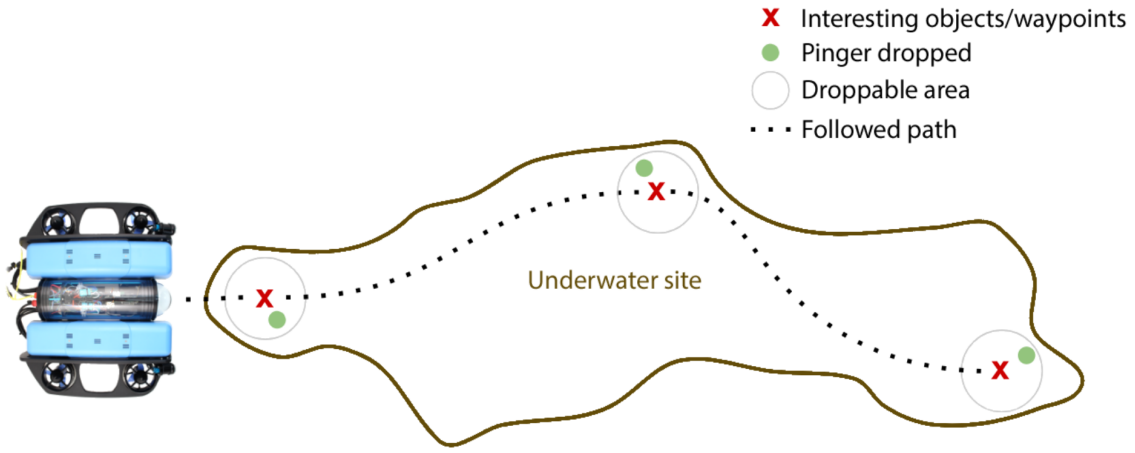


Figure 4.1: Mission

This section begins with explaining the system architecture and then goes into detail how all subsystems work. These are system model, control, path planning, simulation, ULB deployment system and computer vision.

4.1 System Architecture

Before delving into the specifics of each subsystem, it is crucial to thoroughly understand the overall system architecture to understand how everything works. This is particularly vital for mechatronic systems, given the complexity and integration required between numerous subsystems.

4.1.1 Information Modelling

The information model provides a detailed framework outlining the various elements and communication pathways within the robot's system. It highlights how data flows between components, ensuring coherent interaction and coordination among sensors, processors, and actuators. This visualization aids in understanding the integration and functionality of different modules, crucial for both development and troubleshooting.

Appendix A.1 illustrates the model, with added or modified components highlighted in gray for clarity. Only components that are used for this project are displayed. Hence it is a simplified version. The standard Ground Control Station (GCS), QGroundControl, for the BlueROV2, proved inadequate for the project's low-level control and customization requirements. Therefore, a custom GCS was developed utilizing MavROS. enabling the same Mavlink communication but with lower level control. Mavlink is the communication protocol between the Navigator and the Raspberry Pi on the BROV. The use of ROS facilitated seamless integration and synchronization between all components of the project.

4.1.2 ROS Structure

The ROS network has the following structure, visualised in Figure 4.2. The robot sends the DVL and IMU data over ROS to a node named state_estimator. The state estimator converts the data from the DVL together with the data from the IMU to create an estimate of the state of the robot in all 6-DoF both in terms of position and velocity. The state estimator publishes the position and velocity data on the topic `/state_estimation` which the `mpc_publisher` subscribes to. The message on the topic is a self-developed simple message containing the 12 values corresponding to positions and velocities in the 6-DoF. The `mpc_publisher` does all calculations for the MPC, given the state-estimate and a reference waypoint. The reference waypoint is given by the mission planner, described in detail in section **??**. The MPC then outputs an 8 by 1 output vector consisting of the Pulse Width Modulation (PWM) from -1 to 1 for each individual thruster. This output is sent on the topic `/BROV2/mpc` to the `motor_actuation node`. The `motor_actuation` node does a conversion described in section 4.3.3. This allows the ROS network to publish to a topic named `/mavros/rc/override` which allows control of the robot by controlling its forces and torques in its 6-DoF.

The mission planner acts as the Human Machine Interface (HMI) to input reference waypoints to the ROS network. The mission planner also publishes directly to the `/mavros/rc/override` to actuate the dropper. The mission planner is also

responsible for the arming and disarming of the robot to ensure it operates safely.

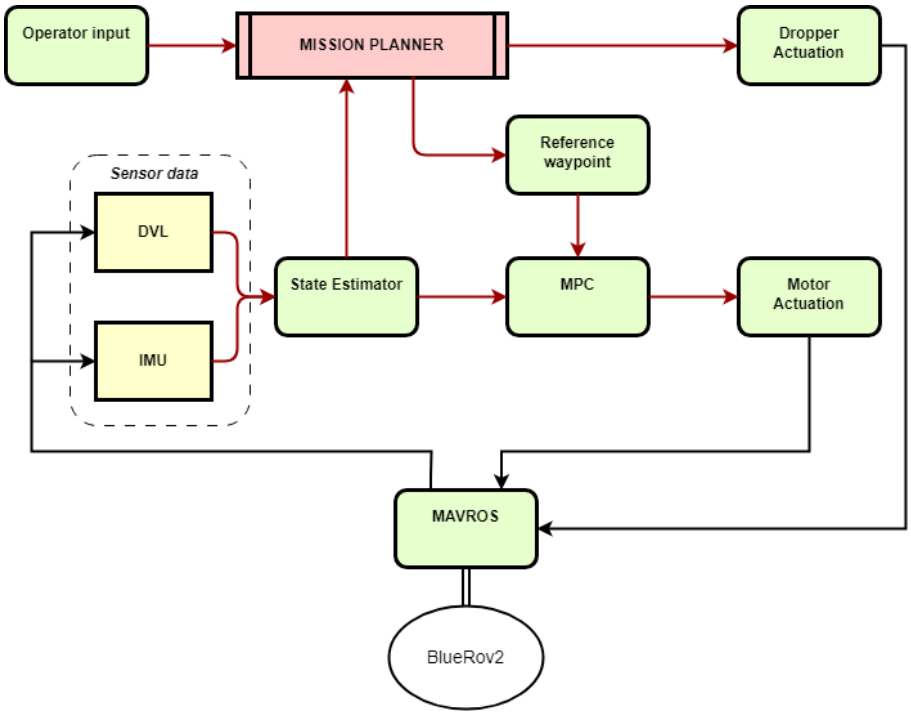


Figure 4.2: ROS structure overview

4.1.3 Mission Planner

A mission planner serves as the decision-making core of autonomous systems, coordinating the tasks and behaviours required to achieve mission objectives. For this project, the mission planner is responsible for determining waypoint navigation, coordinating task execution, and adapting to environmental or system changes in real time. Selecting an effective architecture for the mission planner is critical for ensuring robust, flexible, and maintainable behaviour management.

Finite State Machine (FSM)s have historically been a popular choice for such tasks, offering a straightforward structure of predefined states and transitions. However, FSMs face significant limitations when applied to complex and dynamic scenarios, such as underwater missions. The scalability of FSMs is impeded by an exponential increase in the number of states, which complicates maintenance, debugging, and scalability efforts. Their rigid design also limits flexibility, as adapting to unexpected events or new requirements often necessitates extensive

redesign. Furthermore, the tightly coupled nature of states and transitions restricts the reusability of components across different tasks or missions.

Behaviour Tree (BT)s overcome these challenges by employing a modular and hierarchical structure. Unlike FSMs, BTs organize "behaviours" into reusable nodes, which simplify development and facilitate scalability without exponential growth in complexity. Their reactive execution model allows nodes to be reevaluated dynamically, enabling real-time adaptability to changing conditions. BTs also support parallel task execution, enabling the system to manage multiple objectives simultaneously, such as navigation and sensor monitoring. These characteristics make BTs particularly well-suited for this project and it is why it is the chosen architecture, providing a robust, flexible, and scalable framework for mission planning in complex underwater environments.

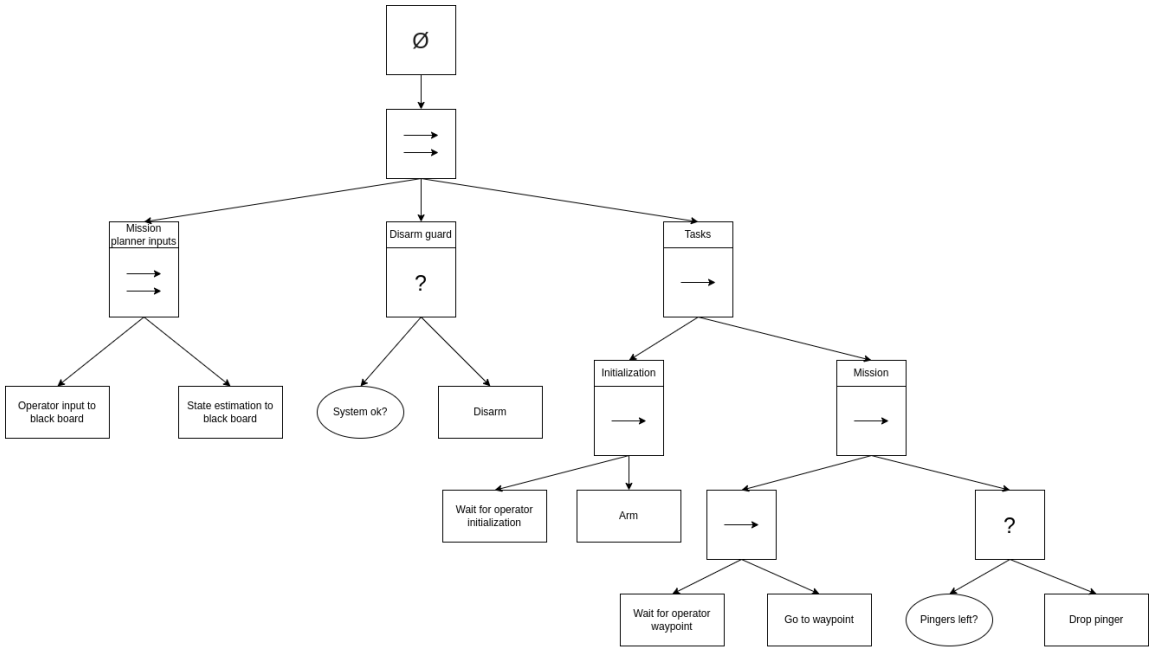


Figure 4.3: Mission planner - behaviour tree

Figure 4.3 illustrates the developed behaviour tree, which has been implemented in Python using the `py_trees` and `py_trees_ros` libraries to ensure synchronization with the ROS network. The initial layer operates in parallel, comprising three distinct components. The first component is responsible for input aggregation, including operator input and state estimation. The second component functions

as a disarm safeguard by subscribing to the MavROS topic `/mavros/diagnostics`, where diagnostic data from the BROV is retrieved. In the event of any malfunction, the system disarms the BROV to prevent actuation, serving as a constant safety measure throughout the mission. The third component encompasses executable tasks that begin by awaiting the initiation by the operator of the initialization process. Upon command, the system arms itself and proceeds to execute the mission task. It subsequently awaits a waypoint input from the operator and, once received, relays the waypoint to the MPC to initiate robot navigation to the specified point. Upon confirmation by the mission planner that the BROV has reached its waypoint, it advances to the next task, which involves deploying a pinger if available. The mission task then awaits the operator's input for a new waypoint. An independent disarm function was developed to bypass the mission planner in scenarios where the system is unable to reach the specified waypoint.

4.2 System Model

A model of the system is essential for an MPC as it relies on simulating the system to find the optimal control input. The accuracy of the model is also of the highest importance as the wrong optimal control input might propagate and grow out of proportion if the model does not match the real system. Any simulations created would also rely on the system model, and a high model fidelity would narrow the sim-to-real. The following chapter describes the system model and how it works.

4.2.1 Differential Equations

The initial model used for the MPC was taken directly from the report "An Open-Source Benchmark Simulator: Control of a BlueROV2 Underwater Robot" by von Benzon and Fogh Sørensen from Aalborg University [24]. The system is modified to fit the need of the project and is using a simpler version. Equations 4.1 and 4.2 below are the main differential equations that describe the entire system.

$$\dot{\eta} = J(\eta)v \tag{4.1}$$

$$M\dot{v} + C(v)v + D(v)v + g(\eta) = \tau + \tau_{tet} \tag{4.2}$$

The vector $\eta = [x, y, z, \phi, \theta, \psi]^T$ describes the absolute position and Euler rotations of the rover related to the world frame, while the vector $\nu = [u, v, w, p, q, r]^T$ is the translational and rotational velocities in the body frame. The following table 4.1 contains the variables of the system and a description of what each part does. The final column also describes what affects and changes each matrix as the system is dependent on the state vectors making the system non-linear.

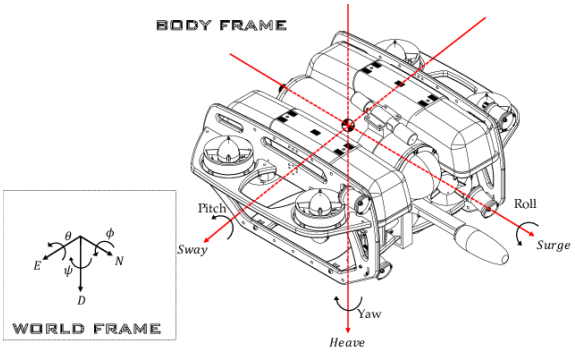


Figure 4.4: BlueROV2 Body and World Frame Reference [25]

Table 4.1: Equation Variables

Notation	Description	Depending
$J(\eta)$	Velocity transform-matrix from body to world velocity	Rotational position of rover
M	Inertia Matrix of rover	-
$C(v)$	Corioliscentripetal Matrix	Rotational and translational velocities of rover
$D(v)$	Damping matrix	Rotational and translational velocities of rover
$g(\eta)$	Vector of gravitational and buoyancy restoring forces	Rotational position of rover
τ	Vector of input forces from thrusters	Thruster input voltage
τ_{tet}	Vector of tether forces	-

The J matrix 4.3 is used to translate the body frame velocities v to the world frame η which are used to integrate into the world positions and angles. The matrix consists of two separate matrices, $J1$ 4.4 and $J2$ 4.5 which both depend on the current rotation of the rover in the world frame. The J matrix is not specific for the BROV but is simply only a geometric transformation of coordinate systems and is widely used in similar situations.

$$J(\eta) = \begin{bmatrix} J_1(\eta) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & J_2(\eta) \end{bmatrix} \tag{4.3}$$

$$J_1(\eta) = \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\phi\sin\theta\sin\psi & -\cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{4.4}$$

$$J_2(\eta) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \tag{4.5}$$

The matrices M, C and D all contain two separate parts, the rigid body and added mass matrices according to equations 4.6 and 4.7 below, where RB stands for rigid body and A for added mass. This is because the under water dynamics differ from those in the atmosphere. The dynamics of the rover itself is not enough to describe the system but rather requires that the additional mass of the water moving with and around the vehicle be taken into account as well. Additionally, the Damping coefficients are also describe using two sub components, one for low speeds where the damping is considerer linear and one where the damping force is related to the square of the velocity as seen in equation 4.8 making the damping non-linear.

$$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A \tag{4.6}$$

$$\mathbf{C}(v) = \mathbf{C}_{RB}(v) + \mathbf{C}_A(v) \tag{4.7}$$

$$\mathbf{D}(v) = \mathbf{D} + \mathbf{D}_n(v) \tag{4.8}$$

The equations below show the composition of the system on matrix form. Equations 4.9 and 4.10 show the rigid body and added mass matrices respectively, where m is the mass of the rover and the I-terms are the inertia in the pitch, roll and yaw directions respectively. The diagonal terms in the added mass matrix 4.10 are the added mass variables for each translational and rotational axis.

The skew-symmetric Coriolis matrices 4.11 and 4.12 describe how the mass and inertia of the system combined with the velocity of the rover affect the Coriolis forces applied on the system. Like the inertia matrix 4.6 has an added mass matrix 4.12 that describe how the water moving around the rover affects the system. Note that the components of the rigid body and added mass Coriolis matrix are the same as in the inertia matrices.

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix} \tag{4.9}$$

$$\mathbf{M}_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{\dot{r}} \end{bmatrix}. \tag{4.10}$$

$$\mathbf{C}_{RB} = \begin{bmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & mw & -mv & 0 & -I_z r & -I_y q \\ -mw & 0 & mu & I_z r & 0 & I_x p \\ mv & -mu & 0 & I_y q & -I_x p & 0 \end{bmatrix} \tag{4.11}$$

$$\mathbf{C}_A = \begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{w}} w & Y_{\dot{v}} v \\ 0 & 0 & 0 & Z_{\dot{w}} w & 0 & -X_{\dot{u}} u \\ 0 & 0 & 0 & 0 & -Y_{\dot{v}} v & X_{\dot{u}} u \\ 0 & 0 & -Z_{\dot{w}} w & Y_{\dot{v}} v & 0 & -N_{\dot{r}} r \\ Z_{\dot{w}} w & 0 & -X_{\dot{u}} u & N_{\dot{r}} r & 0 & M_{\dot{q}} q \\ -Y_{\dot{v}} v & X_{\dot{u}} u & 0 & -M_{\dot{q}} q & K_{\dot{p}} p & 0 \end{bmatrix} \tag{4.12}$$

The damping matrices 4.13 and 4.14, as mentioned, describe both the linear and non-linear damping of the system. X_u is the linear damping in the forward surge direction, while $X_{u|u|}$ is the non-linear damping in the same direction. The $|u|$ of the first element is the absolute value of the speed in the surge direction. This is so that when the matrix multiplication $D(v) * v$ in the differential equation is performed so the direction of the velocity remains in the resulting vector, applying the damping force in the correct direction.

$$\mathbf{D} = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q & 0 \\ 0 & 0 & 0 & 0 & 0 & N_r \end{bmatrix} \tag{4.13}$$

$$\mathbf{D}_n(\nu) = \begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{v|v|}|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{w|w|}|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{q|q|}|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{r|r|}|r| \end{bmatrix} \tag{4.14}$$

4.2.2 Restoring Forces

The restoring forces are the gravitational force due to the mass of the rover and the buoyancy due to the displacement of the water as a function of the rovers volume. Both the buoyancy and gravity will have its effect in the world vertical axis which is why the forces in equation 4.16 and 4.17 are multiplied with trigonometric functions as the forces in the model must be represented in the body frame. The same goes for the torques but as the gravity is always applied at the centre of mass it will not have a moment arm and will therefore not apply any torque. The same cannot be said about the buoyancy as it is applied at the centre of buoyancy which is offset from the centre of mass. The moment arm for the buoyancy force is x_b, y_b and z_b as can be seen in equation 4.15.

$$G = \begin{bmatrix} (W-B)\sin\theta \\ -(W-B)\cos\theta\sin\phi \\ -(W-B)\cos\theta\cos\phi \\ y_bB\cos\theta\cos\phi - z_bB\cos\theta\sin\phi \\ -z_bB\sin\theta - x_bB\cos\theta\cos\phi \\ x_bB\cos\theta\sin\phi + y_bB\sin\theta \end{bmatrix} \tag{4.15}$$

$$W = m \cdot g \tag{4.16}$$

$$B = \rho \cdot g \cdot \delta \tag{4.17}$$

4.2.3 Tether Forces

The final part of the model is the force applied on the rover caused by the tether dragging in the water. This force is however very difficult to model as it depends on a large amount of variables such as the amount of tether releases from the spool and tension of the tether in the water. As this is difficult for both us and the rover to measure it was simply excluded from the model and excluded from the project. This is reasonable since most real world tests will be conducted in a small pool where a tether will not have much effect.

4.2.4 Thrusters

The heavy configuration of the BROV has eight T200 thrusters that allow the robot to actuate in all 6 degrees of freedom [23]. The orientation of the thruster can be seen in figure 4.5.

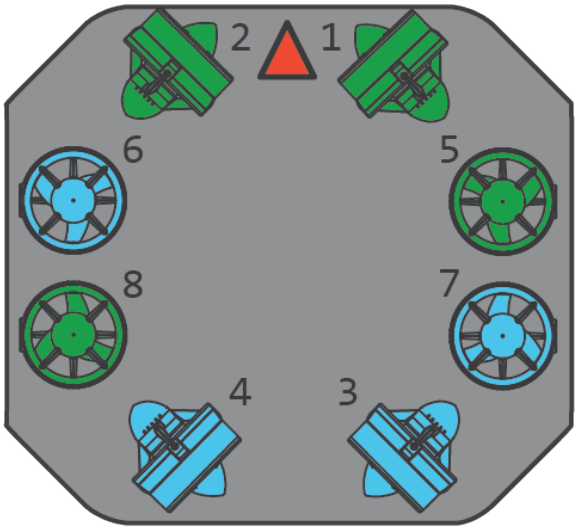


Figure 4.5: BlueROV2 Heavy Configuration Thruster Orientation [26]

The T200 thrusters used on the rover are controlled by sending a PWM signal in the form of 1100µs to 1900µs with zero thrust being 1500µs. To model the force

output from the thrusters, a ninth-order equation was used where the input is a value $V_i \in [-1, 1]$ representing full reverse to full forward thrust respectively. The ninth-order equation returns a value $F(V_i) \in [-30.4, 30.4]$ as thrust in Newtons. [24]

$$F_i(V_i) = -140.3V_i^9 + 389.9V_i^7 - 404.1V_i^5 + 176.0V_i^3 + 8.9V_i \tag{4.18}$$

Equation 4.18 above represents each thruster and each control input V_i is applied separately. To translate the thruster vector $\mathbf{F(V)}$ to force applied in the body frame of the rover a $\mathbf{T}$ matrix was used. The T matrix uses the angle of the thruster relative to the body frame to describe the thruster's interaction to describe the forces applied to the rover in the surge, sway and heave directions. The T matrix also describes the moment arms to calculate the torque in the roll, pitch and yaw directions. The first 3 rows of the matrix are therefore only dependent on the angle of each motor in each direction while the last three rows depend on both the angles and the distance to the centre of mass of the rover. Equation 4.19 describes how the T matrix is applied to the thruster vector $\mathbf{F(V)}$ and matrix 4.20 is the T matrix used in the model.

$$\tau = \mathbf{T} * \mathbf{F}(\mathbf{V}) \tag{4.19}$$

$$\mathbf{T} = \begin{bmatrix} 0.7071 & 0.7071 & -0.7071 & -0.7071 & 0 & 0 & 0 & 0 \\ -0.7071 & 0.7071 & -0.7071 & 0.7071 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0.218 & 0.218 & -0.218 & -0.218 \\ 0 & 0 & 0 & 0 & 0.12 & -0.12 & 0.12 & -0.12 \\ -0.1888 & 0.1888 & 0.1888 & -0.1888 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.20}$$

4.2.5 Model Parameters

The model used is only as good as the parameters used to describe the system, therefore it is important that the parameters used are as accurate to the real world as possible. However, the verification process of inspecting every single variable on every degree of freedom would have been an entire additional project which is outside the scope of this project. The following table 4.2 is a short summary of each variable and where they were obtained. Note that the notation "report" refers to the original project that developed this model. [24]

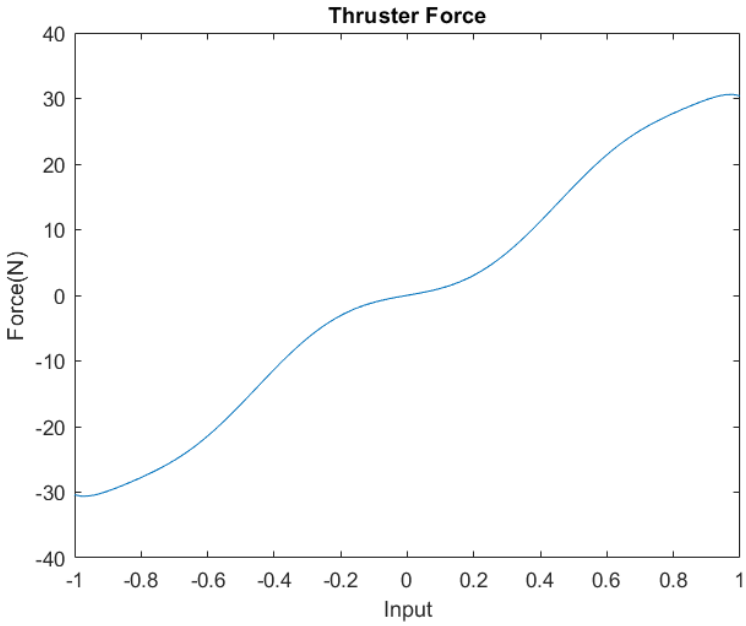


Figure 4.6: Thruster force in newtons as a function of input value

Table 4.2: Model Parameters

Notation	Description	Origin
M_{RB}	Rigid body inertia matrix	Report and CAD
M_A	Added mass inertia matrix	Report
$C_{RB}(v)$	Rigid body Coriolis matrix	Report
$C_A(v)$	Added mass Coriolis matrix	Report
D	Linear damping matrix	Report
$D_n(v)$	Non-Linear damping matrix	Report
$g(\eta)$	Restoring forces	Report
$F(V)$	Thruster force equation	Report and testing
T	Thruster to body frame matrix	Report

All variables are in some way extracted from the previously mentioned report whilst a few have been slightly modified and improved to match our system. The Added mass inertia matrix, Coriolis and Damping matrices are very difficult to verify as they contain multiple variables each and would require real world testing or advanced simulations to determine. However, the restoring forces vector $g(\eta)$ and

Thruster matrix T should have been modified to account for changes to the system in regards to the centre of mass as it changes gravity and buoyancy forces as well as the torque vector in the T matrix.

4.3 Control

This section outlines the primary components involved in controlling the AUV. First, the MPC problem formulation is presented and explained. Then, the use of sensors to estimate the state vector is described. And finally, control allocation, a challenging problem when working with AUVs due to the difficulty of modelling thrusters, is examined [27].

4.3.1 MPC

As can be seen in section 4.2, the BROV is a highly non-linear system. Therefore, a MPC framework that works for non-linear systems needs to be used. In this case, the BROV was controlled using an adaptive MPC framework. This is more accurate than linearizing around an equilibrium point, but also keeps the implementation and solving simple by keeping the problem quadratic and removing the need for a good initial guess.

MPC model, linearization and discretization

This section describes the model used for the Adaptive MPC scheme. This includes linearization and discretization. While linearization is often performed around equilibrium points, this method generalizes the approach to linearizing around an arbitrary operating point. Such a point may not necessarily satisfy the steady-state conditions of the system.[12]

Before that however, the non-linear system described by equations 4.1 and 4.2 have to combined into one. This is done by defining the state vector $\mathbf{x} = [\boldsymbol{\nu}, \boldsymbol{\eta}]^T$, where $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ are defined as in section 4.2.

Now, consider the non-linear system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{4.21}$$

where $\mathbf{x} \in \mathbb{R}^n$ represents the state vector, $\mathbf{u} \in \mathbb{R}^m$ represents the input vector, and $\mathbf{f}(\cdot, \cdot)$ is a non-linear function.

Let $(\mathbf{x}_o, \mathbf{u}_o)$ denote the arbitrary operating point around which the system is linearized. Using a first-order Taylor series expansion, the non-linear system can be

approximated as:

$$\dot{\mathbf{x}} \approx \mathbf{f}(\mathbf{x}_o, \mathbf{u}_o) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_o, \mathbf{u}_o} (\mathbf{x} - \mathbf{x}_o) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\mathbf{x}_o, \mathbf{u}_o} (\mathbf{u} - \mathbf{u}_o), \tag{4.22}$$

where

$$\mathbf{A} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_o, \mathbf{u}_o}, \quad \mathbf{B} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\mathbf{x}_o, \mathbf{u}_o}, \tag{4.23}$$

are the Jacobian matrices evaluated at the chosen operating point.

The term $\mathbf{f}(\mathbf{x}_o, \mathbf{u}_o)$ captures the dynamics at the operating point. If the operating point is not an equilibrium (i.e., $\mathbf{f}(\mathbf{x}_o, \mathbf{u}_o) \neq \mathbf{0}$), this term introduces a constant offset into the system. The linearized dynamics then take the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}_o, \mathbf{u}_o) + \mathbf{A}(\mathbf{x} - \mathbf{x}_o) + \mathbf{B}(\mathbf{u} - \mathbf{u}_o). \tag{4.24}$$

Defining $\kappa = \mathbf{f}(\mathbf{x}_o, \mathbf{u}_o) - \mathbf{A}\mathbf{x}_o - \mathbf{B}\mathbf{u}_o$ we get:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \kappa. \tag{4.25}$$

In order to use this in the MPC it needs to be discretized, but first, MPC is computationally heavy and can add a considerable computational delay to the system. This can negatively affect stability, hence this delay is modelled as part of the problem.

The extended linearized model in continuous time with a time delay, τ_{delay}, is expressed as:

$$\dot{\mathbf{x}}(t) = \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_c(\mathbf{u}(t - \tau_{delay})) + \kappa_c. \tag{4.26}$$

Here A_c and B_c emphasize the fact that these matrices are in continuous time. Finally, assuming Zero-Order hold (ZOH) sampling, i.e. constant inputs during each time step, together with a sampling time, h, that is longer than the delay, τ_{delay}, the discretized model can be expressed as:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{B}_{delay}\mathbf{u}_{k-1} + \kappa. \tag{4.27}$$

where:

$$\mathbf{A} = e^{A_c h}, \quad \mathbf{B} = \int_{s=\tau_{delay}}^{h} e^{A_c s} B_c \, ds, \quad \mathbf{B}_{delay} = \int_{s=0}^{\tau_{delay}} e^{A_c s} B_c \, ds, \quad \kappa = \int_{s=0}^{h} e^{A_c s} \kappa_c \, ds, \tag{4.28}$$

Problem formulation

The MPC problem for reference tracking is typically formulated according to equation 2.3. For the purpose of this project, the problem is instead formulated according to 4.29, which is a simplified formulation. Since no terminal weight or set is used, recursive feasibility and stability are not guaranteed. But the problem is significantly simplified and the implementation is much easier. This is therefore a common approach in practice [28], [29].

$$
\begin{aligned}
\min \quad & \sum_{k=0}^{N-1} (C\hat{x}_k - r_k)^\top Q (C\hat{x}_k - r_k) + \delta\hat{u}_k^\top R \delta\hat{u}_k \\
\text{subject to:} \\
& \hat{x}_{k+1} = A_t\hat{x}_k + B_t\hat{u}_{k-1} + B_t\delta\hat{u}_k + K_t + B_{delay,t}\hat{u}_{k-1}, \quad k = 0, \dots, N-1 \\
& \hat{u}_k = \hat{u}_{k-1} + \delta\hat{u}_k \\
& M_x\hat{x}_k \leq m_x, \quad k = 0, \dots, N-1 \\
& M_u(\hat{u}_{k-1} + \delta\hat{u}_k) \leq m_u, \quad k = 0, \dots, N-1 \\
& u_{-1} = u_t \\
& x_0 = x_t
\end{aligned}
\tag{4.29}
$$

Here the constraints are as follows; first there are the dynamic constraints defined by the linearized model. A_t, B_t and K_t are the linearized discrete-time model parameters at the operating point at time t, which are held constant over the prediction horizon. Then there is the definition of $\delta\hat{u}_k$ followed by the state and input constraints, and finally the initial value constraints on the state and input.

At every sampling instance k the Finite Horizon Optimal Control Problem (FHOCP) is solved to obtain the optimal variables $\hat{x}_k^*$, $\delta\hat{u}_k^*$ and $\hat{u}_{k-1}^*$. Then, the optimal input to be applied is $\hat{u}_t = u_{-1} + \delta\hat{u}_0^*$ where $\hat{u}_t^*$ denotes the optimal control input at time t according to the controller. Over the prediction horizon N, the controller aims to minimize the difference between the outputs $y_k = C\hat{x}_k$, where C is the output matrix and $\hat{x}_k$ is the predicted state at time step k, and the references r_k. At the same time, it also tries to keep the control increment $\delta\hat{u}_k$ to a minimum, in order to achieve a smoother control signal. Analogous to the MPC problem formulation in equation 2.1, the Q and R matrices can be tuned in order to achieve desired behaviour. However, now increasing Q will minimize reference tracking error while increasing R will penalize control increments.

In order to take low-level constraints into account, the MPC output will be the 8x1 thruster vector forces. This will prevent the MPC from computing force and

torque effects that are impossible to achieve with the thruster configuration.

As can be seen, the problem formulation does not include a terminal constraint, this implies that we do not require perfect tracking of the reference. This will however keep the problem feasible regardless of the desired reference and also allow the prediction horizon to be kept short, which is beneficial for the adaptive MPC since the solution will degrade if the current state deviates from the linearization point.

Nevertheless, the optimization problem in equation 4.29 is implemented in Python and solved using cvxpy with the MOSEK solver.

4.3.2 State Estimation

In order for the controller to actuate the system, it first has to be sensed or perceived, in order to feed something into the regulator that will generate an output. The MPC formulation is subject to different constraints, one of which describes the system dynamics. In order to correctly predict the next state of the BROV there's a need to acquire information used to describe the system dynamics. In this case recall the state vector x_t which is also displayed in 4.30.

$$\mathbf{x_t} = [\boldsymbol{\nu_t}, \boldsymbol{\eta_t}]^T \tag{4.30}$$

Two approaches were taken to estimate the state vector. The first approach was using the built-in dead reckoning that was calculated by the DVL and published over the ROS network from the DVL driver used in this project. The Waterlinked A50 DVL provides firstly a highly accurate velocity estimate at a refresh rate between $5-12Hz$ and then provides linear and angular positional estimates using dead reckoning, claimed to have an distance error of 0.13 percent over a distance of 295 meters [22]. The angular rates were taken from the onboard IMU of the BROV. These measurements were all combined in the state estimator node of the ROS network and published as a custom positional message.

The second approach was similar but had some differences in how the computations were made. The linear velocities were received from the DVL and the angular rates from the onboard IMU. All seemingly erroneous velocity measurements were dropped from processing. The absolute angles were taken from the DVL dead reckoning, but the linear positional estimates were calculated from the linear velocities and absolute angles in the state estimator node, as seen below in equations 4.31-4.33. This approach was implemented in response to the DVL behaving erratically towards the end of the project, for more details, see discussion.

In the first approach, handling different frames of reference was straightforward, as the DVL dead reckoning provided the positional estimate relative to the "origin", what is referred to as the odom frame where the BROV was powered on or whenever the DVL was rebooted or reset through the web interface tool. As the to global state estimate of the BROV was interesting, the built-in tf2 ROS library was used in order to handle static and dynamic coordinate conversions. As the DVL is not mounted in the Centre of Gravity (CoG) of the BROV, linear velocities were also transformed using equations for rigid body kinematics, seen in equation 4.34, where $\mathbf{R_{DVLdisplacement}}$ is the distance from CoG to the DVL.

For the second approach, as the velocities were given in the body frame from the DVL, the linear positions were transformed with the tf2 library.

$$x = x_{\text{sum}} + \Delta t \cdot \left(\frac{v_x + v_{x-1}}{2} \cdot \big(\cos\psi + \cos\theta \big) + \frac{v_z + v_{z-1}}{2} \cdot \sin\theta - \frac{v_y + v_{y-1}}{2} \cdot \sin\psi \right) \quad (4.31)$$

$$y = y_{\text{sum}} + \Delta t \cdot \left(\frac{v_y + v_{y-1}}{2} \cdot \big(\cos\psi + \cos\phi \big) + \frac{v_x + v_{x-1}}{2} \cdot \sin\psi - \frac{v_z + v_{z-1}}{2} \cdot \sin\phi \right) \quad (4.32)$$

$$z = z_{\text{sum}} + \Delta t \cdot \left(\frac{v_z + v_{z-1}}{2} \cdot \big(\cos\phi + \cos\theta \big) + \frac{v_y + v_{y-1}}{2} \cdot \sin\phi - \frac{v_x + v_{x-1}}{2} \cdot \sin\theta \right) \quad (4.33)$$

$$\mathbf{v_{body}} = \mathbf{v_{DVL}} - \mathbf{R_{DVLdisplacement}} \times \omega \quad (4.34)$$

4.3.3 Control Allocation

The BlueROV2 was built to be driven by a standard Xbox controller, steering the 6-DoF using the two joysticks and the triggers. Ardusub, the firmware controlling the rover, was therefore built according to this and has a complicated control scheme to drive the multiple thrusters at the same time to achieve the desired movement. Steering the rover is done by sending PWM on the form $PWM \in [1100, 1900](\mu s)$ where $1900\mu s$ corresponds to full thrust, $1100\mu s$ full reverse thrust and $1500\mu s$ zero thrust. This signal is sent to each degree of freedom and Ardusub will automatically compute the signal to pass to the individual thrusters. Here is how.

$$\hat{T} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 \end{bmatrix} \quad (4.35)$$

Matrix 4.35 is Ardusub's thruster allocation matrix, or $\hat{T}$, which is used to actuate the thrusters. The matrix has the same shape, only transposed, as the T matrix 4.20 from earlier, where all the previous values have been replaced by ones only retaining the sign to keep the direction of the thruster. Ardusub translates the PWM signals to the form $Signal \in [-1, 1]$ using equation 4.36 below. This new 6x1 signal vector is then multiplied with the $\hat{T}$ matrix to output the individual thruster control values

on an 8x1 vector $[-1, 1]$ which are then mapped to the PWM values $[1100, 1900]$ for each thruster to use.

Understanding how Ardusub allocates control signals to the thrusters is necessary to understand how the output signals from the MPC are sent to the rover. Since the only way the rover accepts inputs is on this 6x1 PWM vector the 8x1 thruster output vector needs to be translated to this form. The solution is to do what Ardusub does in reverse.

Taking the inverse of $\hat{T}$ allows the output vector to be reversed to the form of the 6x1 vector but since $\hat{T}$ has more rows than columns and is not full rank this is not possible. However a pseudo inverse of $\hat{T}$ approximates this. Using equation 4.37, the new 6x1 output signals can be translated to the PWM form.

$$sig = (pwm - 1500)/400 \tag{4.36}$$

$$pwm = sig * 400 + 1500 \tag{4.37}$$

This is not, however, a perfect solution since there are multiple thruster force combinations that can have the same effects on the rover. This means that when the MPC has chosen a specific thruster output this translation can cause a different set of thrusters to be actuated. This should not be an issue since the same effect is put on the system, however, the constraints set on the MPC to not create outputs other than $[-1, 1]$ might not be fulfilled. The translation can cause control inputs to be larger or smaller than the allowed inputs which then does not have the desired effect that the MPC wants.

The Ardusub system is however prepared for this and can normalize illegal control actions so the same action is taken but with a smaller amplitude. The overall result of this is a system that won't outright break if the wrong action is sent but might not always actuate as desired. This is an issue with Ardusub and has not been looked at in further detail in this project.

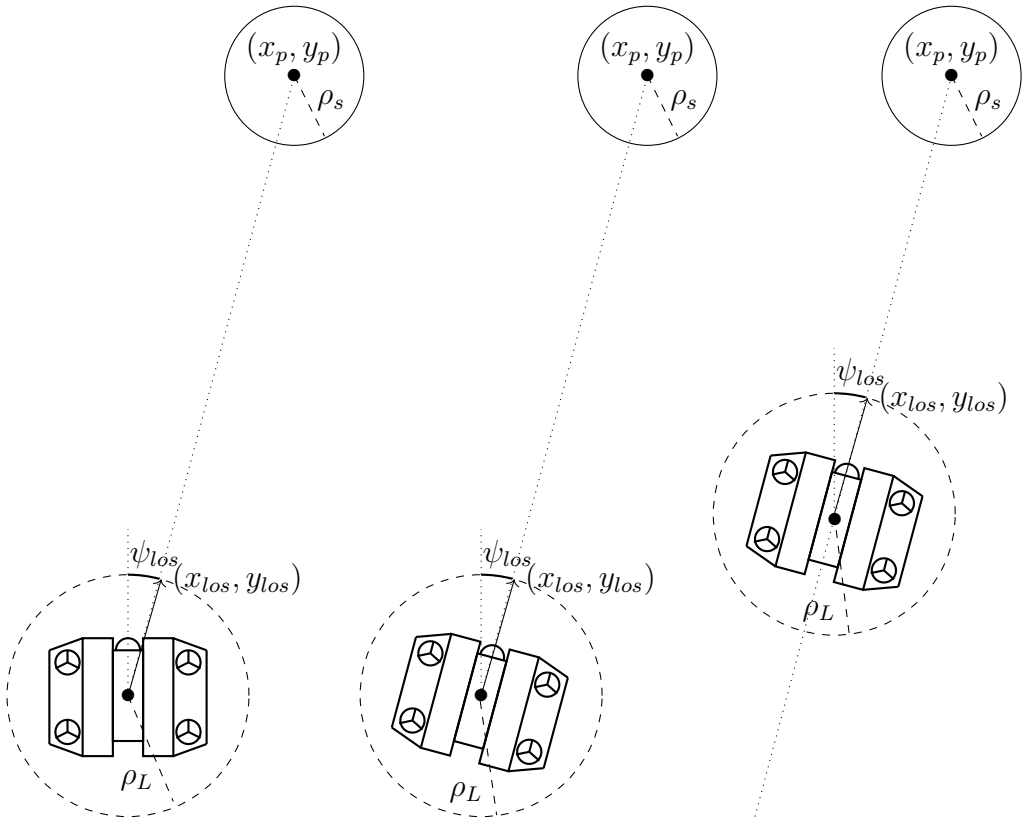
4.4 Path Planning

Since an AUV exhibits coupled motions under various disturbances, it can face challenges in following a certain reference point [28]. To combat this, a guidance system can be used to update the AUV's desired trajectory in real-time based on an estimate of its current position. Furthermore, the MPC formulation in equation 4.29 assumes that a known trajectory is being followed and because of this a path planning system needs to be implemented, which can be done in different ways. In this paper,

the SOTA guidance system Line-of-Sight (LOS) and another simple method, where a trajectory planner, that assumes constant acceleration and deceleration phases, is combined with a simple guidance system, are implemented and compared.

4.4.1 Line-of-Sight (LOS) Guidance System

The guidance system generates reference points to the control system as the ROV progresses along the path to the desired way point, rather than having a single static reference point. The system used in this case is a Line-of-Sight (LOS) guidance system, as this is the most commonly used system for marine vehicles [28]. This is illustrated in figure 4.7 below.



(a) System initiated **(b)** Yaw angle adjusted **(c)** Approaching way point

Figure 4.7: LOS guidance system

The input reference to the ROV is given by the so-called LOS-coordinate. The LOS-coordinate is determined based on the ROV's current position $\mathbf{x}$, the desired

way point $\mathbf{x}_p$ and a so-called LOS-sphere with radius ρ_L that encircles the ROV. The LOS-coordinate (x_{los}, y_{los}) is given by the intersection between the LOS-sphere and the straight line connecting the ROV and the way point. To make the ROV head "face first" to the target, its desired yaw angle is set to ψ_{los}. By trigonometry, these coordinates are given by:

$$\psi_{los} = \tan^{-1}\left(\frac{y_p - y}{x_p - x}\right) \tag{4.38}$$

$$x_{los} = x + \frac{\rho_c}{\sqrt{1 + \tan^2 \psi_{los}}} \tag{4.39}$$

$$y_{los} = y + \tan\psi_{los}(x_{los} - x) \tag{4.40}$$

Since the ROV is moving in 3D, its depth coordinate z_{los} must also be determined. Analogously to the 2D LOS-coordinate, z_{los} is given by the intersection between the LOS-sphere and the path towards the way point. Formally, z_{los} is proportional to the radius ρ_L with a factor of θ, which is the angle between the xy-plane and the desired depth z_p [28] – as illustrated in figure 4.8 below.

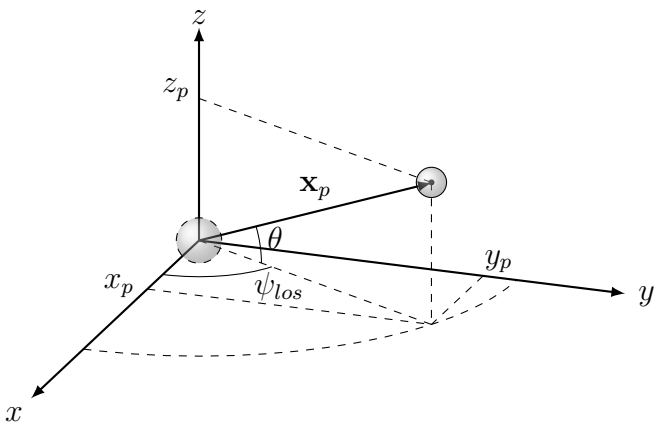


Figure 4.8: LOS guidance system in 3D

By trigonometry, the angle θ is given by:

$$\theta = \tan^{-1}\left(\frac{z_p - z}{\sqrt{(x_p - x)^2 + (y_p - y)^2}}\right), \tag{4.41}$$

resulting in the LOS-coordinate

$$z_{los} = z + \rho_L \tan\theta \tag{4.42}$$

When the controller is initialized, the guidance system gives the initial LOS-coordinate to the ROV. As the ROV has a new position in the next time step, it gets a new LOS-coordinate, since the LOS-sphere follows the ROV along the path. Therefore, this algorithm forces the ROV to chase a coordinate it cannot reach, until it enters the so-called sphere of acceptance. This sphere, with radius ρ_s, indicates the volume which is "close enough" to the way point, and acts as a conditional volume for the ROV to either switch to a new way point or to stay at the current one.

Compared to the 3D LOS guidance system implemented in [28], this system has an adaptive yaw angle ψ_{los}, rather than a static one that is computed at the start of the mission. This enables the ROV to adjust its heading angle in case it drifts away from the path, rather than solely controlling its surge and sway. Furthermore, this guidance system tries to minimize the ROV's pitch, since the testing was done in rather shallow depths and due to the DVL's operating range. This approach differs from another implementation, which adjusts the vehicle's pitch to move in the z-direction [28]. In addition to not use the pitch angle in the path planner, the roll angle is not considered either. Therefore, $\phi_{los} = 0$ and $\theta_{los} = 0$.

The path planner stores the 6 reference states for linear and angular position in a vector described as

$$\mathbf{x}_{ref} = [x_{los}, y_{los}, z_{los}, \phi_{los}, \theta_{los}, \psi_{los}] \tag{4.43}$$

where $\phi_{los} = 0$ and $\theta_{los} = 0$. The MPC uses a certain reference for each step in the prediction horizon. This is done in order to keep the reference up to date with the current position of the BROV. Therefore, the reference vectors need to be stored in a matrix with a size of 6 x N, where N is the number of steps in the prediction horizon.

4.4.2 Simple Trajectory Planner and Guidance

The second method used is a more traditional version, where an acceleration profile is first built assuming constant, equal acceleration and deceleration phases and a constant velocity phase, this profile is then integrated twice to obtain the desired trajectory. The angles and angular velocities are assumed to be zero at all times and therefore the robot will always move towards the reference in a straight line while keeping it's orientation. The acceleration and velocities can be tuned to achieve specific performance but must be kept within the realm of feasibility for the robot's dynamics. An example of the motion profile can be seen in figure 4.9.

Next, the MPC will need to pick out the part of the trajectory that it will use over

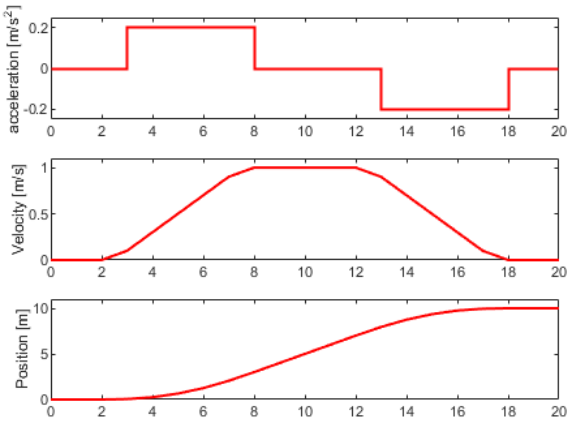


Figure 4.9: Motion profile example for the trajectory planner

it's prediction horizon. Therefore, some sort of guidance system is also necessary. A complete overview of the system can be seen in figure 4.10.

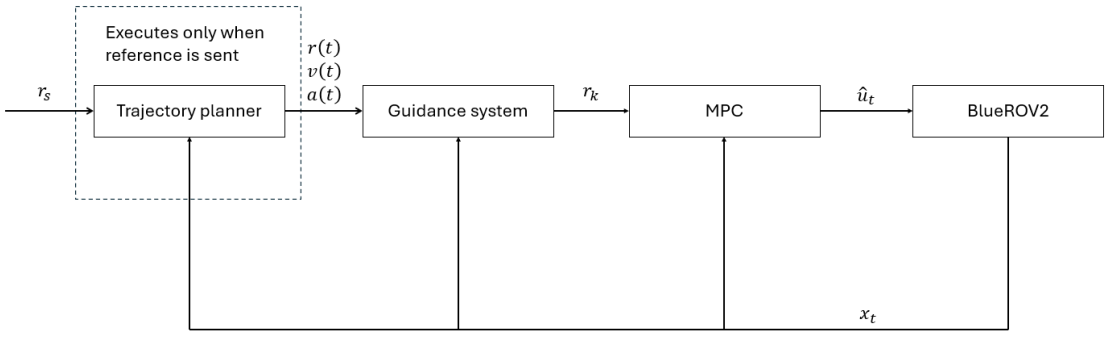


Figure 4.10: System architecture diagram for trajectory planner

For every new reference, r_s, a full trajectory is planned consisting of position $r(t)$, velocity $v(t)$ and acceleration $a(t)$ profiles. Then, a guidance system working in tandem with the MPC at every time step, will check the current position of the robot, x_t, and extract a part of the full trajectory for the MPC to follow over its prediction horizon, denoted r_k. This is simply chosen by taking the point on the trajectory closest to the current position and then feeding it, as well as the next few points, to the MPC.

4.5 Simulation

There were multiple simulations made for the project in order to be able to develop the MPC at a more rapid pace as compared to if all testing had to have been done on the physical robot.

The Unity simulation, as seen in figure 4.11, serves as the primary testing environment for both the system model and the MPC. This simulation was developed in collaboration with the Swedish Maritime Robotics Centre (SMaRC). Within the Unity simulation, the BlueROV2 is modelled and simulated based on the system dynamics, enabling a realistic testing and evaluation platform for the system performance.

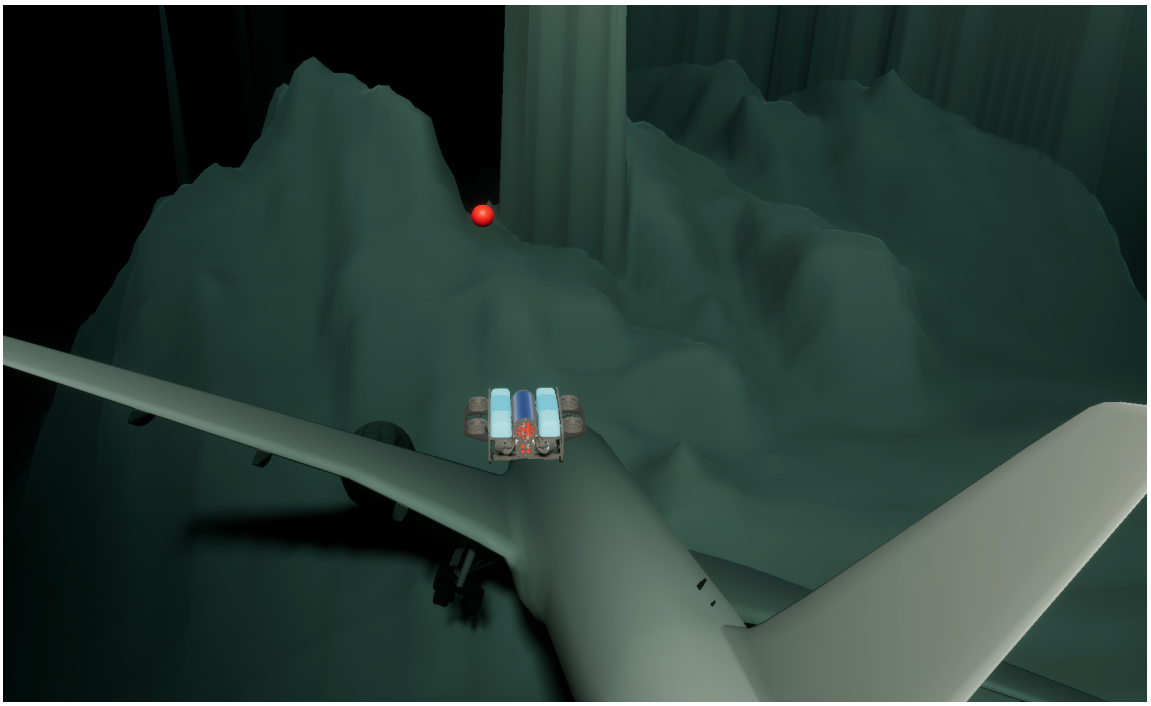


Figure 4.11: A screenshot of the Unity simulation

4.5.1 System Model Implementation

The implementation of the model follows Equation 4.2, with the exclusion of τ_{tet}. The system model is then visualized by applying the force and torque outputs of the model to the centre of mass of an articulated body, which, within the scope of this simulation, acts as a rigid body.

The usage of articulation bodies is to enable joints within the body and thus movement of the body relative to itself. By redistributing the inertia and mass of all the articulation bodies to one main articulation body, a process often known as mass lumping, the main body can be treated as a rigid body while also allowing the inertia and mass of any moving parts to be as close to zero as Unity allows. This means that when they move, any forces that arise can be neglected. In this way, the main body on which the model forces act can still be considered a rigid body, despite having parts that move independently, which is important for the accuracy of the system model. In order to apply the system model to the main body, a unity script, `BlueROV2ForceModel`, was attached to it. This enables interaction with the main body during simulation runtime. The main part of the code relevant to this report lies is executed by, `FixedUpdate()` at a frequency of $50Hz$ during runtime. In this method the current state of the main body is read at the start of every iteration. The state of the main body is then converted from unity's Right, Up, Forward (RUF) left-handed coordinate system to the North, East, Down (NED) right-handed coordinate system used in the system model as well as to the appropriate reference frame, before being used to calculate the actuating forces. The forces are then converted back into the Unity coordinate system and applied to the main body's centre of mass using methods from the articulation body class. The two methods used for this are `AddRelativeForce()`, which adds a force at the centre of mass relative to the local coordinate system of the articulation body, and `AddRelativeTorque()`, which adds a torque relative to the local coordinate system at the origin. For this reason, it is important to ensure that the origin of the articulation body's local coordinate system is aligned with its centre of mass. In addition to the forces generated by the system model, input forces and torques are added to the body, which move it according to either the user input or controller input.

In Unity, when forces are applied to an articulation body, it uses the defined inertia vector and mass of the body to calculate the resulting movement. However, an articulation body in Unity only accepts a single mass value, unlike the system model. The inertia matrix used in the system model is a combination of two different matrices: the rigid body matrix and the added mass matrix. The added mass matrix, in particular, includes directional masses along the cardinal directions (see equations 4.9 and 4.10). To address this, the equation of motion was restructured so that the added mass and rigid body inertia matrices are separated. A negative force, equivalent to the impact of the added mass matrix, is added to both sides of the equation, effectively cancelling out the added mass matrix from the left-hand side and allowing Unity to handle the applied forces appropriately. The rewritten equation

used in Unity is as follows:

$$M_{RB}\dot{v} = \tau - C(v)v - D(v)v - g(\eta) - M_A M^{-1}(\tau - C(v)v - D(v)v - g(\eta)) \quad (4.44)$$

4.5.2 Actuators and Sensors Integration

In the simulation, the Unity scene was structured to closely resemble the one used by SMaRC for their simulations. This approach allowed the simulation to be extended beyond the previously discussed system model, incorporating additional features such as ROS2 communication and simulated actuators and sensors, through the use of the corresponding scripts [30]. Among these extended features, two key scripts were used to enhance the functionality of the simulation: the subscriber script `PropellerCommand_Sub` and the publisher script `Odometry_Pub`. As previously discussed, the main body is composed of articulation bodies, allowing it to contain moving parts. In the simulation, these moving parts include eight different propellers, one for each thruster. By configuring each propeller as an actuator and adding the `PropellerCommand_Sub` script, you can set the RPMs for each propeller via ROS. The RPMs of each propeller are visually observable during runtime and can be read by the `BlueROV2ForceModel` script. When read by the script, the RPMs are processed and converted into the corresponding PWM values, which are then transformed into the input forces and torques used in the system model, following equations 4.20 and 4.18 in sequence. The other `Odometry_Pub` script is used to, at a variable frequency, publish the state of the main body.

4.5.3 Peripherals

In addition, several new features not present in the SMaRC repository were added. One of these is the `referencePoint` object. This object is a collision-less red sphere that is both subscribed to the ROS2 node handling the controller's reference value and publishes to it. This means that whenever the reference value for the controller is updated, the red sphere will appear at that location in the simulation. Moving the sphere will also update the reference value, allowing the reference to be visualized and intuitively adjusted during runtime. The sphere can be moved using the W, A, S, D, Shift, and Space keys. Another addition is the `ULBdropper` object, which acts as a spawning point for ULB objects during runtime. This object is connected to the mission planner via a ROS service but can also be activated by pressing the U key. Lastly, a user interface element was added in the form of a checkbox to the simulation. When ticked, the simulation enters "controller mode", where user inputs control the reference point, which the main body follows using the MPC controller.

When unticked, the input forces received from the controller are no longer applied to the main body, and the user can manually control the body.

4.5.4 ROS Integration

In order for the Unity simulation to work seamlessly with the Model Predictive Controller developed for the physical robot, some of the ROS nodes made for the physical robot were replaced with nodes tailored to the simulation, this is visualized in Figure 4.12. Specifically the state-estimator was replaced by a simulation odometry listener which was named sim_odom_listener. The motor actuation node was replaced with a node named sim thruster control that published Revolutions Per Minute (RPM) values on 8 different topics, corresponding to each and every thruster on the simulated BlueROV2 (BROV). The conversion between this RPM and the force on the ROV in the simulation was handled in unity.

For the communication to work between the Robot Operating System (ROS) network and the Unity simulation, a node named ros_tcp_endpoint has to be running on the same computer running the Unity simulation.[31]

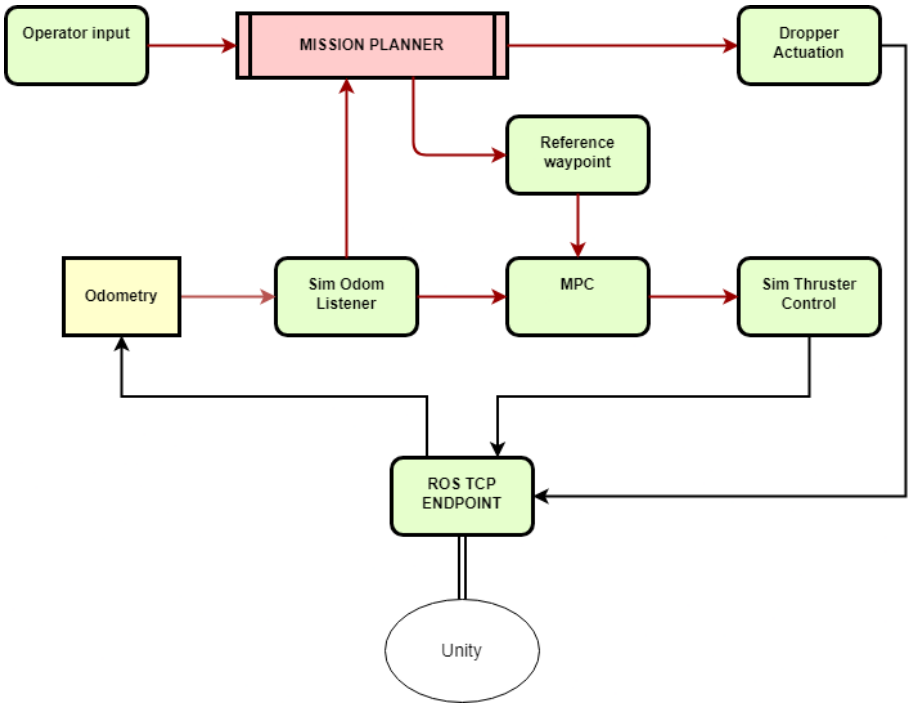


Figure 4.12: Robot Operating System (ROS) Structure for Simulation

4.5.5 Matlab and Python Simulations

The first model of the system was implemented in Matlab Simulink. The purpose of the model was to do some initial testing of the model described earlier to verify that the model and the expected result of the model were equal. A screenshot of the Simulink model can be viewed below in Appendix B where the thruster inputs are visible on the left and the output position on the right.

Using the model of the system some preliminary tests of an MPC were made using the built in MPC library on Matlab. This confirmed that any basic Model Predictive Control could not satisfy the non-linearity of the model and a more complex system would be required. Additionally, the model was used to verify all future simulation models used for the MPC and that the additional simulations were accurate.

Also during development, an intermediate Python simulation was used for testing the system as it grew. It was used to test the models, the path-planning algorithms and the early and mid stages of the MPC.

4.6 ULB Deployment System

The deployment system for the underwater locator beacons (ULB) was designed as a rotating magazine that holds three (replica) ULBs, in cylindrical cutouts that can rotate inside of a cylindrical shell that has an opening towards the bottom. The ULBs is released through the opening in the bottom. The mechanism is actuated by a waterproof servo motor placed at the back of the deployment system. The output axis of the servo is then connected to a transmission with a gear ratio of $n = \frac{1}{2}$. This enables a regular 180° servo motor to rotate the magazine 360° – thus making it possible to make a full revolution and release all three ULBs. The output gear of the transmission is mounted on an axle, which the rotating magazine is attached. The mechanical design of the drop mechanism is illustrated in figure 4.13 below.

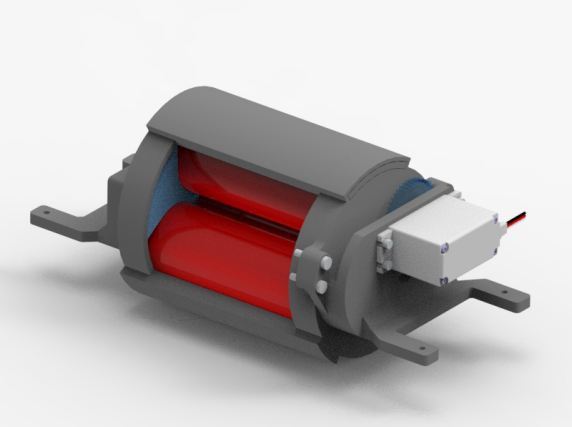
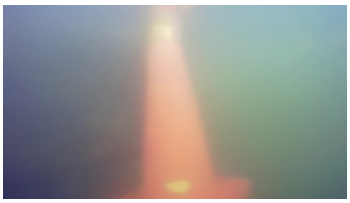


Figure 4.13: ULB deployment design

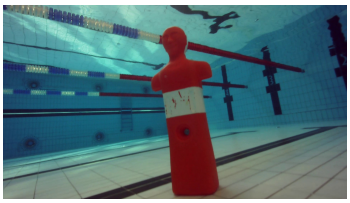
The servo motor was connected to the Navigator on the BROV. The signal line was connected to pin 12 on the Navigator, which supplies the servo motor with PWM signals. The motor is actuated via the MavROS message `OverrideRCIn`, which sets the specified PWM signal on the Navigator's pin.

4.7 Computer Vision

Image enhancement was performed on frames extracted from three test videos, see figure 4.14.



(a) Traffic cone in Brunnsviken.



(b) Life-saving dummy in GIH-badet.

(c) Bottle with pingers in babypool.

Figure 4.14: Example frames from videos used for computer vision.

A pre-existing GAN model was evaluated for this purpose, namely funie-GAN [32]. No additional training was performed to the model. The model was employed by inputting the images from this study, and the outputs were then evaluated. For 3D reconstruction from unstructured images, two SfM & MVSpipelines; Colmap and AliceVision were tested using the same frames [33].

5 Verification and Validation

5.1 Planned Testing Methods

To validate the system, each component and the system as a whole underwent testing.

- MPC Validation:

  The MPC was initially implemented in Python to verify its stand-alone functionality. This phase evaluated its performance i.e. (reference tracking) and stability when applied to the model. *cvxpy*, which is an open source python package for convex optimization problems, was used to solve the optimization problem. The chosen solver was MOSEK.

- Simulation Testing in Unity:

  The MPC was integrated into a Unity simulation of the BlueROV2. This stage tested the communication pipeline and the overall system performance, including state estimation and mission planning. The system was provided with reference inputs, and its stability, speed and deviation from the reference were measured to assess performance.

- Real-World Deployment:

  The MPC was then deployed on the physical BROV to observe its behaviour in real-world conditions. This step was done to determine if the system model was accurate enough for practical application.

- State Estimator Testing:

  The state estimator's performance was verified by comparing its estimated position against ground truth data obtained from a motion capture system.

- Model Validation:

  To ensure the accuracy of the simulation and the model, tests were conducted by actuating the BROV using the simulation. These tests were running the simulation and simultaneously actuating the BROV with the same inputs.

- ULB Deployment System Testing:

  The ULB mechanism was evaluated through drop tests conducted in a pool environment.

- Image Processing and 3D Reconstruction:

  Image enhancement techniques were assessed visually for improvements in clarity, while the 3D reconstruction output was compared against the actual subject.

5.2 Reliability and Validity of Testing Methods

The project was guided by soft requirements, as seen in section 1.3, rather than strict measurable benchmarks, as the primary objective was to deliver a proof of concept demonstrating the practical implementation of the system.

The testing was not done to produce statistically significant results but to provide a qualitative measure of system performance. Due to the many different subsystems and interdisciplinary nature of the project, some limitations in the reliability and validity of the testing was inevitable.

To minimise these problems, individual components were tested independently to identify issues. This approach enhanced the reliability of the results and improved confidence in the integrated system's functionality. By isolating and validating components where possible the robustness of the overall system evaluation was improved.

6 Results

This section presents the performance results of the MPC, both from simulations and from real-world tests. Further, the results of the state estimation, deployment system and computer vision are also presented.

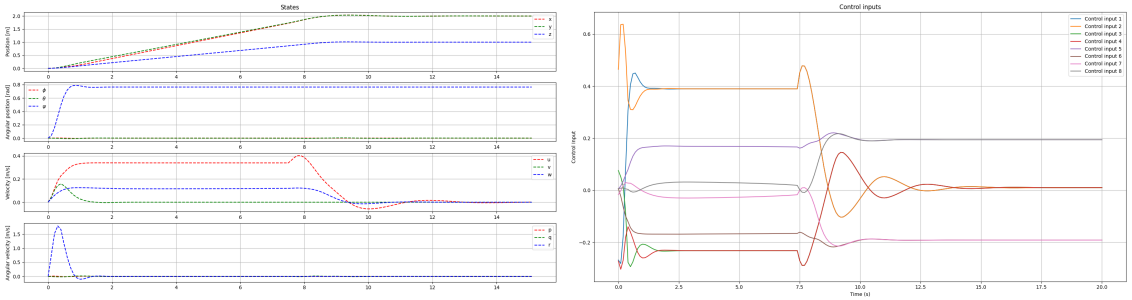
6.1 MPC in Python

In order to evaluate the performance and viability of the proposed controller scheme, it was first tested in Python. The controller is simulated for the non-linear system with different parameters using CasADi, which is an open source symbolic tool for algorithmic differentiation and optimal control. This is done for both the LOS and the simple trajectory planner. The effect of computational delay on model stability is also studied.

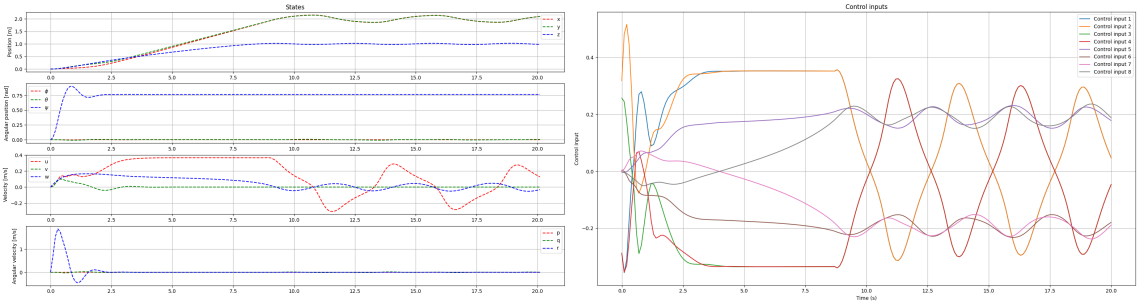
The MPC was tested for varying initial conditions and references, as well as different parameters. For all tests presented below, unless otherwise specified, the sampling time, h, was chosen to be $0.1s$ and the weight matrices Q and R were set to the identity matrices. The system response was compared for two choices of horizon lengths N, for a travel from the initial position at $x_0 = (0, 0, 0)$ to the reference $x_s = (2, 2, 1)$.

6.1.1 LOS Guidance System

The results using the LOS guidance system in the python simulations are visualized in 6.1. Here, the LOS parameters are chosen as: $\rho_L = 0.03$ and $\rho_s = 0.03$. A comparison of the computational time for the different horizon lengths can also be found in table 6.1.



(a) Horizon length $N = 15$



(b) Horizon length $N = 8$

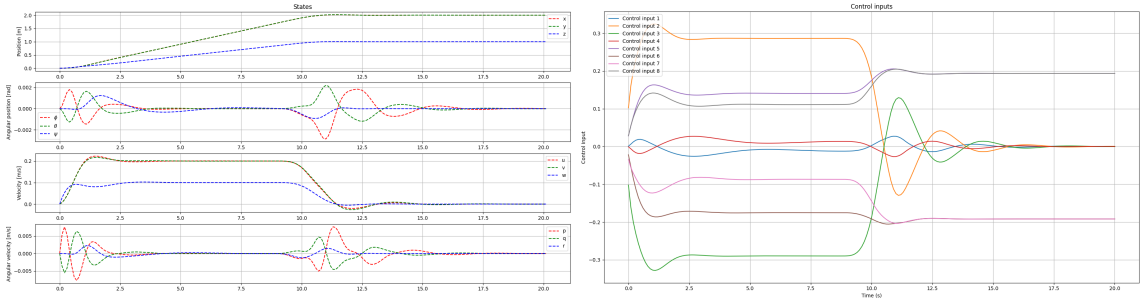
Figure 6.1: Line-of-sight (LOS) in Python simulation for different horizon lengths.

Table 6.1: Average solving times for different horizon lengths.

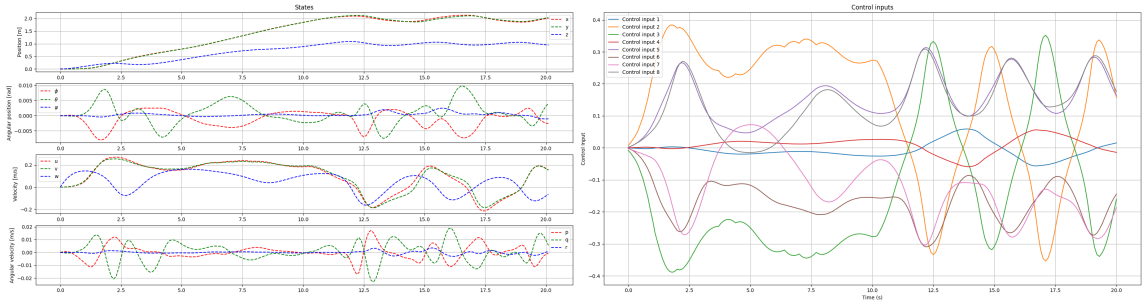
Horizon Length (N)	**Average Solving Time (s)**	**Percentage Decrease (%)**
15	0.069	—
8	0.056	18.8

6.1.2 Trajectory Planner

The results for the Trajectory planner are presented in figure 6.2. The parameters for the trajectory planner was chosen as: $v_{max} = 0.3\ m/s$ and $a_{max} = 0.4\ m/s^2$. A comparison of the computation time for the different horizon lengths can also be seen in table 6.2.



(a) Horizon length $N = 15$



(b) Horizon length $N = 8$

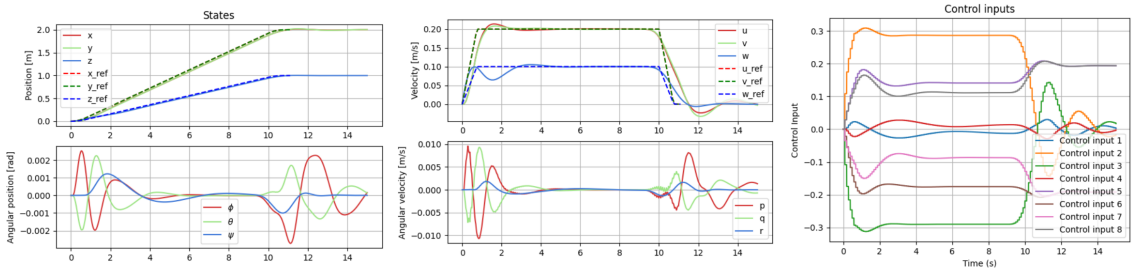
Figure 6.2: Trajectory planner in Python simulation for different horizon lengths.

Table 6.2: Average solving times for different horizon lengths.

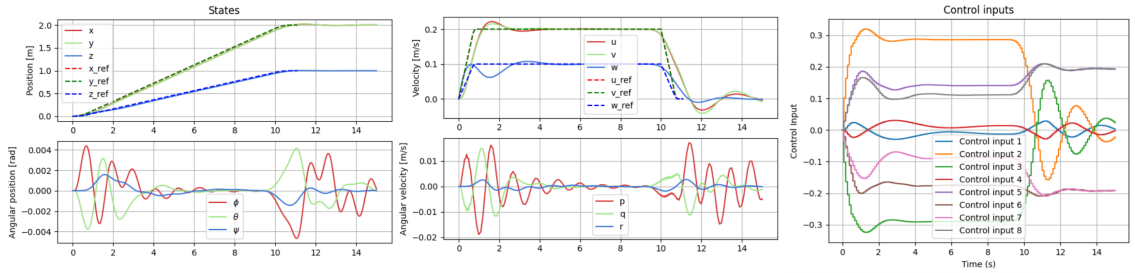
Horizon Length (N)	**Average Solving Time (s)**	**Percentage Decrease (%)**
15	0.066	—
8	0.052	21.2

6.1.3 Computational Delay

Last but not least, a computational delay is simulated in Python and MPCs that do and do not take delay into account are compared. This can be seen in figure 6.3 and 6.4. Where all parameters are kept as above but $N = 15$ and Q is varied.

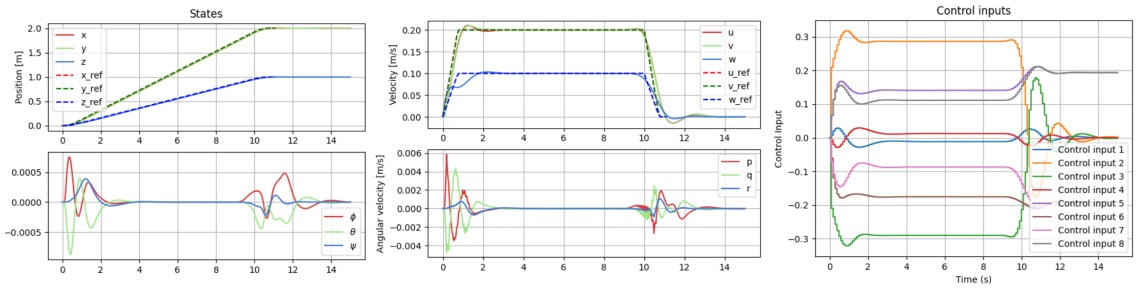


(a) MPC with modelling of computational delay

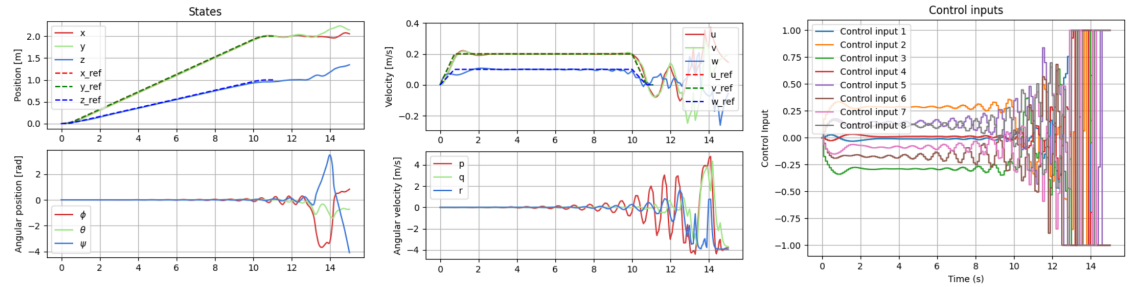


(b) MPC without modelling of computational delay

Figure 6.3: System response with $\tau_{delay} = 0.09$ and $Q = I$ for MPC with and without modelling of delay, the reference trajectory is also plotted for comparison



(a) MPC with modelling of computational delay



(b) MPC without modelling of computational delay

Figure 6.4: System response with $\tau_{delay} = 0.09$ and $Q = 10I$ for MPC with and without modelling of delay, the reference trajectory is also plotted for comparison

6.2 MPC in Unity

The following figures illustrates the control performance in Unity simulations, using both the LOS guidance system and the trajectory planner.

6.2.1 Trajectory Planner

In figure 6.5 and 6.6 below, the step responses of the MPC are illustrated for different references. The upmost plots of the respective figures illustrate the robot's position over time, including the step input. The downmost plots represent the control inputs to each thruster over the same time horizon.

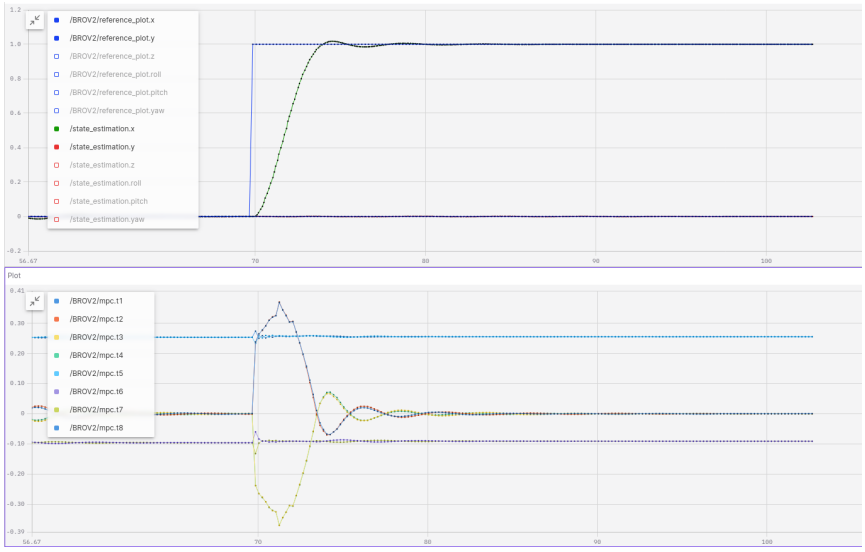


Figure 6.5: Step response to $(x_{ref}, y_{ref}) = (1, 0)$ using trajectory planner

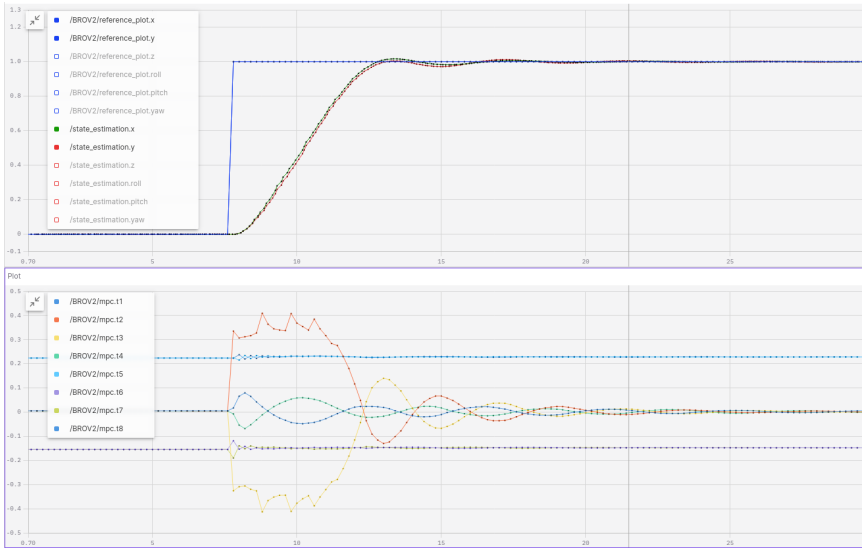


Figure 6.6: Step response to $(x_{ref}, y_{ref}) = (1, 1)$ using trajectory planner

For the reference $(x_{ref}, y_{ref}) = (1, 0)$, the system has a rise time of $2.2s$ and 2% overshoot. Further, the system has a rise time of $3.1s$ for the reference $(x_{ref}, y_{ref}) = (1, 1)$. It has an overshoot in y of 2%, whereas the overshoot in x is negligible.

6.2.2 LOS Guidance System

In figure 6.7 and 6.8 below, the step response for the LOS guidance system for a reference $(x_{ref}, y_{ref}) = (1, 1)$ is presented. In addition to illustrating the position (x, y) over time, as in figure 6.7, the robot's yaw angle over time is depicted in figure 6.8. This is included because the LOS system generates an adaptive yaw angle, unlike the trajectory planner, which maintains a static yaw angle of zero.

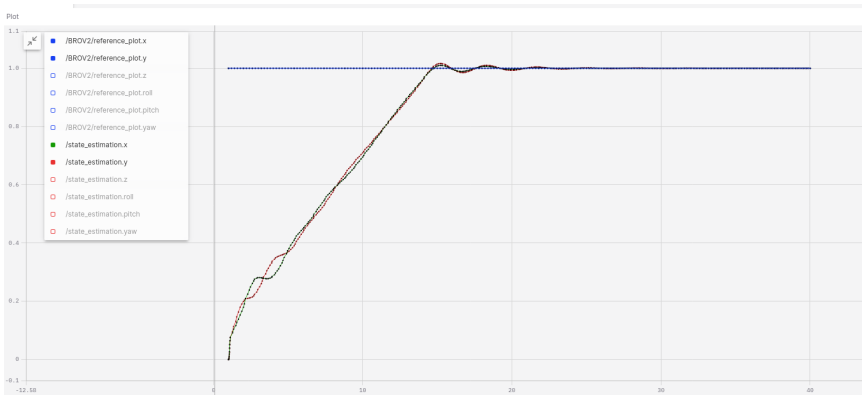


Figure 6.7: Step response to $(x_{ref}, y_{ref}) = (1, 1)$ using LOS system

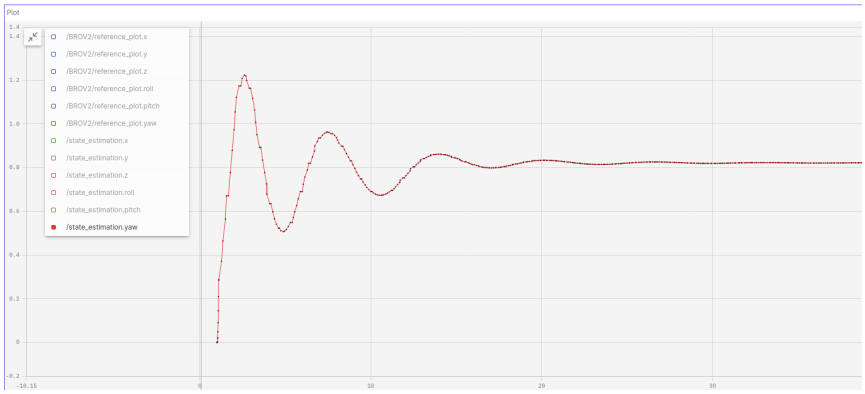


Figure 6.8: Step response of yaw angle to $(x_{ref}, y_{ref}) = (1, 1)$ using LOS system

The position in (x, y) has a rise time of $11.8s$ and an overshoot of 0.9% and 1.8% in x and y respectively. The yaw angle has an overshoot of 49.1% and a settling time of approximately $13s$.

6.2.3 Thrusters

Figure 6.9 below show the thruster actuation with a reference of $x_{ref} = 1$. Note that none of the inputs exceed the values $[-1, 1]$ and while thrusters t_{1-4} are symmetric around 0 thrusters t_{5-8} are not. Comparing the thrusters with the T matrix 4.20 and figure 4.5 indicate the resulting force is in the surge direction as expected.

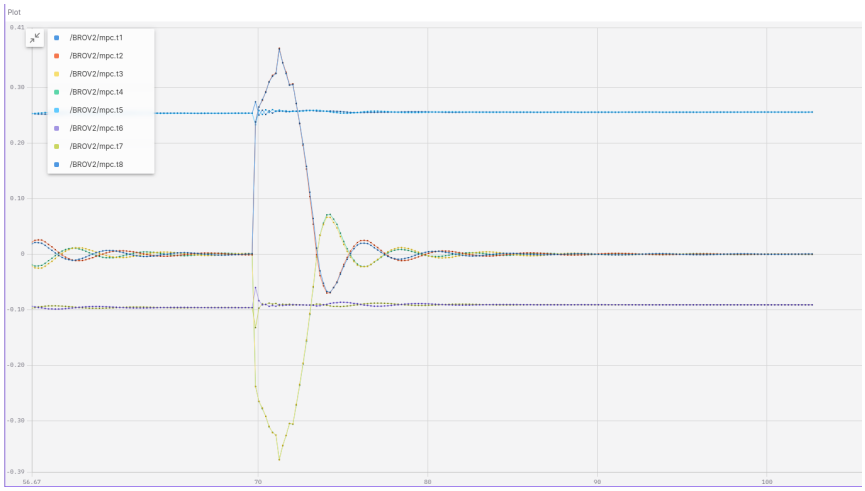


Figure 6.9: Thruster response $x_{ref} = 1$ in unity

6.3 MPC Pool Test

In figure 6.10 below, the step response result of a pool test is illustrated. The reference to the robot is to go diagonally to the coordinate $(x_{ref}, y_{ref}) = (1, 1)$. The graphs represent the estimated states (x, y) of the robot over time. In this case, the trajectory planner was used. No performance data was collected of the LOS guidance system due to unreliable state estimation data (see subsection 7.5).

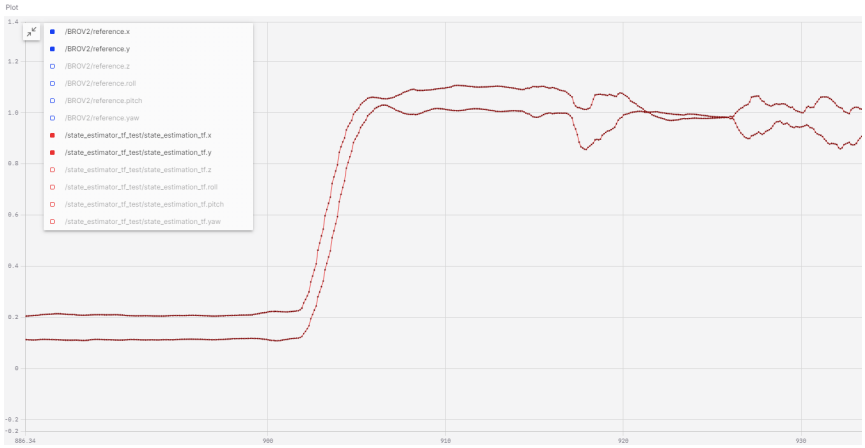


Figure 6.10: Step response to $(x_{ref}, y_{ref}) = (1, 1)$ in pool using trajectory planner

6.4 State Estimation

Seen below in figure 6.11 the first approach to state estimation is displayed and compared to the references and the motion capture data (note that motion capture y coordinate equals state estimator x coordinate and vice versa due to differences in coordinate setup for the systems). Its observed how the ROV first goes towards its reference point and how the state estimator follows the true motion capture positional value. However, the positional estimate from the state estimator gives out an erroneous sudden jump, which makes the control system believe it needs to correct more than it should, leading to the true position of the ROV drifting from the reference.

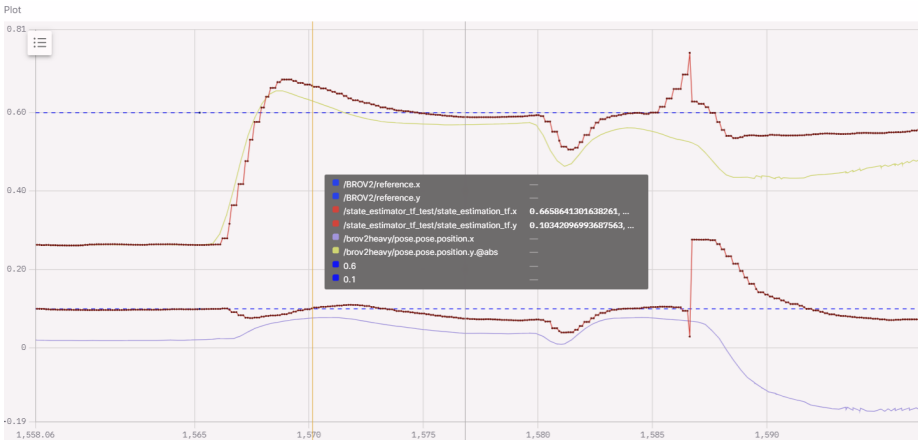


Figure 6.11: Motion capture and state estimator comparison

The second approach had limited testing time, as it was trialled very late into the project and thus, no data is presented.

6.5 Computer Vision

In Figure 6.12 the result of the image enhancement of the three videos can be seen. The first row displays the taken images and the second row shows their enhanced result.

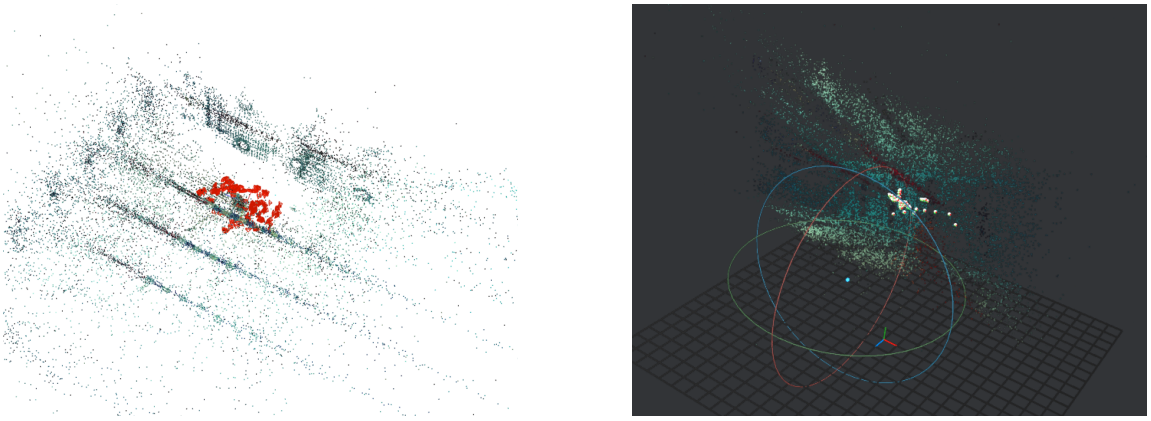
(a) Traffic cone in Brunnsviken.

(b) Life-saving dummy in GIH-badet.

(c) Bottle with pingers in babypool.

Figure 6.12: Comparison of original & enhanced frames of the three videos

In Figure 6.13 the result of the 3D reconstruction of Life-saving dummy in GIH-badet can be seen. The other videos did not produce a point cloud.



(a) Colmap **(b)** AliceVision

Figure 6.13: Point cloud from 3D reconstruction of dummy in GIH-badet.

7 Discussion and conclusions

7.1 System Model

The system model of the BlueROV2 as previously stated multiple times was not developed during this study, but was directly taken from an independent study. That study developed the model as a testing platform for evaluating control algorithms, the decision was made to not use their Matlab program since the software was not easily adaptable, therefore an independent software was developed to run the MPC on an actual BROV. The decision was also made to slightly simplify the model by removing the tether forces on the system as it by far is the most difficult part of the system to model properly. This resulted in a system that was, although complicated, not difficult to understand and most of all was easy to implement in both Python and Unity.

Currently, the model used for the MPC is the same model used to evaluate the system in the Unity model, making the virtual sim-to-real gap close to zero since there is nearly no model error of the system. This is good in regards to testing that the Model Predictive Control system and algorithm is working properly and outputs control action that move the rover to the desired reference. However, part of the task was to implement and test the controller on a real system, and if the model now is not accurate, the ROV might become unstable despite the system working properly. Future work would be to test and evaluate the model error on both the digital and real system, more about that in chapter 8.1.

7.2 Simulation

When implementing the system model in the simulation, certain modifications had to be made, as shown in equation 4.44. These changes were necessary for compatibility with the simulation framework but did not alter the core functionality of the model. Consequently, the differences in testing results between Python and Unity have become an area of interest. As demonstrated by the results, similar tests in both environments yield slightly different outcomes, with Unity consistently exhibiting more oscillatory behaviour.

One potential reason for the observed differences is delays in communication between the MPC and the Unity simulation, particularly varying delays, which are inherently challenging to model or account for accurately. Moreover, the simulation operates at a strict actuation interval of 50Hz, independent of the MPC's calculations. This discrepancy introduces additional delay, as the simulation cannot actuate immediately upon receiving new control signals; instead, it waits until the

next actuation interval. Another contributing factor could be the performance of the computer running the simulation. If the computers battery level is low, the system may enter a power-saving mode, reducing performance. This leads to the simulation running visually slower and exhibiting increased oscillations. In prolonged low-performance states, the simulated BROV can appear unstable. This correlation between slower simulation and more pronounced oscillations suggests that the actuation interval plays a significant role in the systems dynamic response as it is very likely to be impacted.

Differences in the solvers used between Unity and Python might also contribute to the observed variations. While this difference is presumed to be minimal and has been neglected in this analysis, as is the case for other smaller errors such as floating point precision. Regarding state feedback during simulation, the states in Unity are close to ideal because the simulated sensors are bypassed, and the exact state of the BROV is used directly. This approach is similar to the Python simulation, where the solvers output is used as the state. However, a small difference arises in the timing of state calculations. In Python, states are sampled at the exact time the MPC requires them, ensuring minimal delay. In Unity, however, the state is updated at a fixed frequency of 10Hz, which means the MPC may operate on slightly outdated state information. While this is a minor difference, it mimics the behaviour of a real-world state estimator and contributes to the system's overall realism. Adjusting the Unity simulation to align more closely with Pythons timing would improve accuracy compared to the Python simulation but reduce realism in terms of mimicking real-world scenarios.

Comparing simulation data to real-world test results is difficult due to the poor quality of the real-world data. Ideally, the differences between simulation and real-world performance would be analysed to reduce the sim-to-real gap. However, since the real-world data is not reliable, this analysis focuses on the discrepancies between the Python and Unity simulations. By understanding these differences, we can gain insights into how similar issues might affect real-world performance and find ways to improve the overall accuracy and reliability.

In conclusion, the key factors contributing to the differences between Unity and Python simulations include communication delays, actuation intervals, variations in computational performance, and state feedback timing. Further investigation into these factors could provide deeper insights into minimizing discrepancies while maintaining the simulation's applicability to real-world scenarios.

7.3 MPC

Judging from the results in section 6.1, the controller is capable of tracking the reference trajectory while maintaining stability and fulfilling system constraints. It's stability is however dependant on the horizon length N, more specifically, stability improves for larger values of N which is expected. To improve the results even further and either reduce tracking error or get smoother control signals, time could be spent tuning Q and R to achieve desired response.

As for the computational delay, it is evident from figures 6.3 and 6.4 that long time delays τ_{delay} can severely affect stability. In figure 6.3 this is not as evident but it can be seen that delays induce oscillations in the states. Increasing the weight matrix Q to $10I$ has a significant affect on behaviour and the controller turns out to be unstable. This is important as increasing Q will improve tracking (it increases the cost on the reference error) and this will be very relevant if the controller is tuned in the future.

It is also important to highlight the effects of the choice of trajectory planner on the controller. The reference trajectory has to be feasible given the dynamics of the system or else perfect tracking will not be possible. Furthermore, if complicated motions are generated, then it might also be difficult to guarantee stability.

As previously mentioned adaptive MPC keeps the linearized model constant over the prediction horizon, this simplifies implementation but can negatively affect behaviour if the prediction horizon is long and the real system changes drastically over the prediction horizon. Since it is known (see section 4.2) that the non-linear behaviour is a function of the angles and velocities. Keeping those constant over the trajectory is beneficial. This is exactly what the trajectory planners deployed do, although the LOS version does require a heading angle. It can be observed that they both maintain a long constant velocity phase for which the linearized model becomes equivalent to the non-linear model, this is beneficial. If instead, trajectories with quickly varying velocities and angles are generated, the prediction horizon might need to be kept short in order tof keep the linearized model an accurate approximation, but a balance has to be found since, as previously seen, the prediction horizon needs to be relatively long to maintain stability. This could potentially make tuning the controller quite difficult, but needs to be studied more thoroughly and it is nonetheless safe to say that the controller works quite well for simpler paths and missions.

A little has to also be said about the choice of control input. The MPC is configured to output the eight thruster PWM signals directly. As mentioned in section 4.3.1 this was done to ensure that the MPC does not send illegal control actions by taking constraints set by the thruster configuration into account. This

will however increase the computational load, which is detrimental when trying to run the controller in real-time. It could be preferable to translate the constraints on the thrusters to constraints on the forces, if possible, and keep the controller output as the $6x1$ body-fixed forces and moments.

In unity, the controller works quite similarly but with some oscillations, the reason for this is discussed in section 7.2. Currently, the controller only takes the computation delay in the ROS python node into account, but there is also delay induced by the computation done in unity and the communication time between all the different nodes in ROS.

In real life, countless problems were encountered, the biggest of which was the state estimation using the DVL, more on this in section 7.5. There is also a lot to be said about the effects of the ArduSub software on the performance of the MPC. As mentioned in section 4.3.3, because of the way that ArduSub handles inputs, the resulting forces on the BROV will not necessarily be equal to the forces computed by the controller although the robot will always move in the correct direction.The effect of this discrepancy has not been studied and is unknown but it is safe to assume that it will affect the overall behaviour of the system.

7.4 Guidance System

Two different guidance systems were tested and evaluated. The LOS guidance system and the trajectory planner. They were used to either generate a path or trajectory for the BlueROV2 to follow, both with their own characteristics. The main drawbacks of the LOS system was that it always accelerated in the end just before deceleration and it was hard to select the desired maximum velocity for the BROV. Furthermore, this system resulted in oscillations in the robot's yaw angle when travelling both in x and y. This is probably due to the fact that this system always tries to maintain the heading angle towards the final way point. This could probably have been solved with some tuning of the MPC and troubleshooting of the guidance system.

The trajectory planner had the drawbacks of not being able to rotate and only travel along the translational axis. It also needed to have a acceleration and velocity specified to be functional. The choices of acceleration and velocity were chosen heuristically, based on the limitations on the BROV and reasonableness. Another approach would have been to solve for the acceleration from the dynamical model rather than specify a fixed value for the acceleration.

The big difference in rise time between the two systems originates from the algorithms themselves. For the LOS system, the surge speed is mainly determined implicitly by the radius of the LOS-sphere and the sampling frequency. Since the

algorithm prompts the BROV to go to the sphere's edge at each time step, the speed is proportional to the radius and inversely proportional to the sampling frequency. A larger radius "forces" the BROV to reach that specific coordinate in each time step, resulting in a faster surge speed. The main drawback of this logic is that the MPC must be able to find an optimal solution for a reference that is located "one radius away". When the LOS system was tuned, it was clear that the radius had to be rather small, since the MPC failed to solve the optimization problem for larger radii.

7.5 State Estimation

At times, the state estimation initially passed the eye test. The first approach described in chapter 4.3.2 was used for the majority of testing. However, no proper data gathering was conducted until the final tests as the verification method for the state estimator was not ready and work on implementation was prioritized.

As testing hours increased, more erratic behaviour was observed from the state estimation. The yaw angle would drift spontaneously and unfortunately the linear positions would sometimes jerk and shift, something which was observed in the results, see figure 7.1.

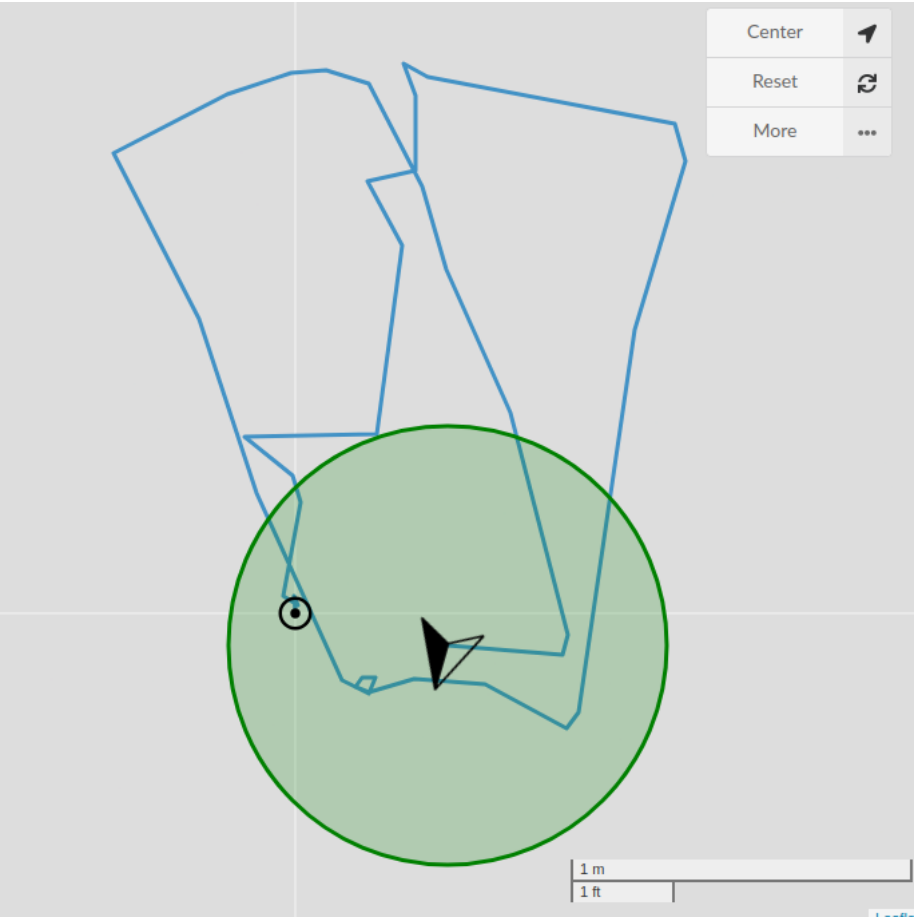


Figure 7.1: DVL user interface state estimation in x & y going 2 laps around a square pool.

The results of the state estimation implementation were not fully satisfactory. It was not possible to replicate the positional accuracy that Waterlinked claim the DVL has during operation. The velocity measurements were orders of magnitude more reliable and therefore an approach in which dead reckoning is performed on data from the DVL and a separate and better performing IMU could be preferable. As mentioned in chapter 4.3.2 this approach was trialled briefly with promising behaviour, but ultimately time ran out before the implementation was fully fleshed out.

While conducting research about localization, it was understood that within the time limitations of the project and constraints given the sensors available, dead reckoning would be the best option available. Unfortunately quite a lot of time was

spent working with the built in dead reckoning, but given the supposed accuracy, it did look like the best approach to state estimation.

During a testing session, the DVL unfortunately heated up past its operating temperature. The manufacturer claims their software includes a thermal protection feature which will turn the acoustics of the sensor off when reaching beyond standard operational temperatures and the DVL was quickly put in the water to cool down. It is not known if this damaged the internals of the sensor, however it is something worth to be considered as a possible source of error.

7.6 Computer Vision

As the computer vision component of this project was not the primary focus, the results were suboptimal. The image enhancement made a slight visual improvement of the images but not as much as anticipated. A suspected reason for this is the established property of GAN having poor mode coverage as the model was trained in a different environment then in this project. The result of the 3D reconstruction did not yield any usable material that was similar to the subject and the image enhancement did not improve the reconstruction.

The challenges faced in the computer vision tasks underscore the inherent difficulties of using cameras in underwater environments. Limited visibility due to low light levels and murky water significantly complicates image-based operations. Interestingly, even under improved visibility conditions during pool tests, the 3D reconstruction did not yield better results. This could be attributed to factors such as light refraction and the lack of distinct, unique features in the underwater scenes, which are essential for effective reconstruction.

7.7 Project Management

The project was challenging due to frequent changes, making the use of agile project management a good decision. Even though the focus of the project and various technical decisions changed along the project, the agile approach enabled the project to be somewhat deliverable over a long time. Responsibilities shifted at times, but this proved beneficial as it allowed everyone to gain a better understanding of the project as a whole. Communication was effective with sprint meetings, but the fast pace made it difficult to maintain the quality of the report and final results. While the project was completed, the data collection was affected by the implementation challenges faced at the end of the project.

7.8 Ethics and Sustainability

This project incorporates several ethical and sustainability considerations in line with good engineering practice. The primary ethical and sustainability goals were centred around improving safety, minimizing environmental impact, and promoting resource efficiency.

The implementation of Model Predictive Control enables a sustainability approach towards automatic control. While in this project, a trajectory following problem was formulated, the optimization problem can be extended to minimize the energy usage of the BlueROV2. By using predictive control, the system can minimize unnecessary movements, reducing power consumption and extending operational time. This approach supports the broader goal of resource-efficient underwater exploration.

Of course, it goes without saying that usage of AUV/ROVs are inherently less resource intensive than manned underwater exploration. This is the result of being able to scale down the size and dimensioning of underwater vessels, due to different size and safety constraints.

Underwater robots provide an alternative to human divers for underwater archaeological missions, eliminating the need for people to operate in dangerous underwater environments. This decreases the risk of archaeological exploration, associated with high-pressure, low-visibility underwater operations which can be problematic to keep safe enough for human exploration.

In alignment with sustainability and ethical testing practices, extensive simulations were conducted prior to real-world deployment. This ensured that the system's functionality could be validated in a virtual environment, minimizing the risk of hardware damage or environmental disturbance. Furthermore, almost all physical tests were performed in a controlled pool environment, which allowed for careful monitoring of the system while reducing the potential impact on natural ecosystems.

By focusing on controlled testing and enabling tools for both energy efficiency and safety for underwater exploration, the project aligns with the principles of sustainable engineering and responsible innovation. These measures demonstrate a commitment to leveraging technology for the benefit of society and the environment while minimizing risks and resource usage.

8 Future work

8.1 Parameter Uncertainty

As mentioned earlier in chapter 4.2.5 it is critical to have an accurate model of the system to achieve a functioning MPC. Developing that model is, however, very difficult and takes a long time, and additionally the model might change over time and during different operations and environments, making it critical to have a good understanding of the system and ideally a structure on how to calculate and determine model parameters as the system changes during development. Future work to obtain a functioning system therefore includes verification of the current system model and development of a pipeline for verification of the model when it changes.

Achieving a perfect system is however impossible. Hard work can achieve a very accurate model but given that the system can change over time it could be necessary to study the robustness of the MPC in regards to model error. One possibility is to use an accurate model of the system on the MPC and apply the control input on a model that is slightly changed. An analysis can then be made comparing the model error to the stability of the controller to determine how accurate the system needs to be to achieve stability.

8.2 Disturbance Modelling

Developing the model of the BROV to include disturbances would enhance the accuracy of the MPC and simulations, allowing them to account for forces and conditions encountered in realistic testing environments. Modelling underwater currents, wave dynamics, and discrepancies in water density could all prove crucial for real-world testing scenarios. If further testing with the BlueROV2 as an ROV is conducted, it will also be important to model the forces exerted by the tether.

8.3 Simulation

To significantly reduce the sim-to-real gap, a more comprehensive and detailed system model is crucial. A refined model would enable the simulation to more accurately reflect real-world conditions, enhancing its overall reliability and applicability. In addition to improving the system model, there are several aspects of the simulation itself that could be optimized to increase both accuracy and usability. One enhancement involves improving the simulation's state feedback by incorporating data derived from simulated sensors. By doing so, the simulation's state estimation would closely align with that of the real-world system, creating a more realistic

testing environment. Furthermore, adding noise to these simulated sensors would further enhance the fidelity of the simulation, allowing for a more accurate emulation of real-world conditions. Another change to keep in mind is to make it so that the frequency of the simulation, matches the actuation frequency of the real life BROV. One could also simulate the delays of real world use, which would allow for countermeasures to be taken based on the simulation.

When using ArduSub, its core functionality could be simulated within the main Unity force script. This approach would enable a more precise replication of the control and feedback mechanisms of the real-world vehicle. Consequently, the test results obtained from the simulation would be more representative of the real system, ensuring greater relevance and reliability of the data.

8.4 MPC

As mentioned in section 7.3, the MPC is configured to output the eight thruster PWM signals directly. A proposed modification involves shifting the MPC to operate at a higher abstraction level by calculating the six actuating forces instead. This approach would, theoretically, enable the controller to be divided into two components: a high-level MPC operating at a lower frequency and a low-level controller running at a higher frequency to manage the thruster PWM signals. This decoupling would reduce the computational load on the MPC and enhance responsiveness by delegating the real-time control of thruster PWM signals to the low-level controller.

To implement this change, the constraints currently tied to the normalized nature of the PWM signals, such as their bounded range, must be somehow reformulated in terms of forces. This ensures that the output forces remain within the feasible operational range of the BROV, preserving system performance and physical realism.

Moreover, during the project, tuning was limited to ensuring the system's basic functionality, without focusing on optimizing performance beyond the required operational thresholds. Future work could involve tuning the controller to optimize its performance for the specific use case, both in simulation and within a real-world testing environment. This would allow for further improvements to the MPC performance.

As for the proposed controller scheme, changing from adaptive to LTV or non-linear MPC could also be necessary in the future, if more complicated trajectories need to be tracked. However, more data and tests need to be conducted first to evaluate the performance of the proposed controller. Moreover, if the trajectories are kept simple then adaptive MPC will work quite well as evident by the results in sections 6.1 and 6.2.

It is also important to note that, since no model is perfect, a robust MPC framework should be considered in the future to handle model error. Offset-free MPC should also be implemented in order to compensate for disturbances.

8.5 Guidance System

A further developed guidance system would be desired for the BROV. A system that both enables angles and velocities to be specified at each point generated by the guidance system towards the reference point. The LOS did not have the ability to specify velocities and the trajectory planner was not able to rotate.

An interesting development would be to solve for the accelerations in the BROVs dynamical model and calculate the trajectories based on those accelerations rather than use constant accelerations. In that way, non-linearities can be considered and a more feasible trajectory can be generated.

Another development would be to implement obstacle avoidance for the guidance system. None of the systems developed can handle it, even though it is beneficial for motions in the underwater environment.

8.6 Computer Vision

The results of the computer vision part in this project highlight the need for specialized techniques and robust algorithms designed to address the unique challenges of underwater imaging. A deep dive in these algorithms, continued tuning or development of novel models could be tested to improve the results.

8.7 Ardusub Modifications

As previously mentioned, there is a discrepancy between the PWM signals that the MPC sends to ArduSub and the PWM signals actuating the system. One potential solution to this problem is to modify the `AP_Motors6DOF::output_armed_stabilizing_vectored_6dof()` function in the `AP_Motors6DOF.cpp` file in ArduSub, as outlined by Ng and Krieg (2024) in their study on improving BlueROV software in the loop accuracy and the design of a hybrid autopilot [34]. Alternatively, you could also directly control the PWM signals of the BlueROV2, but at a cost of a lot of the functionality provided by Ardusub.

8.8 State Estimation

As mentioned in the discussion and evident from the results, the state estimation implementation is subpar in performance. The brief amount of testing using a custom dead reckoning solution displayed as mentioned, promising results while not being flawless. Needless to say, as no proper data was gathered, it could not be concluded to be a superior approach in the grasp of this project. However, the promising tests make it a reasonable place to start the continued work.

Navigation underwater is not a trivial task and is more often than not quite costly and requires advanced technical approaches to the problem. A common approach is using probabilistic filtering algorithms like the extended Kalman filter, coupled with usage of DVL, underwater Global Positioning System (GPS) or GPS fixes at resurfacing and expensive Inertial measurement unit (IMU)s. Research has also been conducted on using unscented kalman filters as outlined in [35] where GPS was only used at resurfacing. A possible solution given that you only have a DVL sensor and IMUs avaliable is using extended kalman filters and fusing the measured data together with dynamic motion model estimates of where the ROV/AUV should end up given the previous control input. An attempt at something similar was made in [36] which investigated how to handle GPS outages in underwater navigation.

Of course, adding more sensors would enable other possibilities regarding navigation, underwater GPS is as mentioned common within high accuracy ROV/AUV applications [22]. The built in IMU on the BROV comes standard with the Navigator flight controller and is quite cheap, unfortunately it is also prone to drifting in the angular and linear measurements. It would be wise to conduct further research with a higher precision IMU.

For modern robotic systems operating on land, utilizing Simultaneous localization and mapping (SLAM) methods are key for highly accurate localization and of course also for mapping purposes. Underwater SLAM is a rapidly developing research field which aims at improving AUV operation, specificity aiming at navigation underwater. In order to enable the use of SLAM, more sensors are needed as SLAM fuses proprioceptive and exteroceptive measurements in order to create a map and converge to a highly accurate localization estimate. Examples of sensors that could be used in underwater environments are stereo and monocular cameras, multi-beam sonars and side-scan sonars. Underwater SLAM would be an interesting avenue to investigate the implementation of in order to increase the underwater navigation capabilities. [37]

8.9 AUV

Currently, the BlueROV2 operates as a ROV. However, as its hardware and software capabilities are further developed, transitioning to an AUV represents a logical next step for this project. Achieving this transition will require internalizing the topside computer by either upgrading the existing internal computer or integrating a more advanced one. In the current setup, the tether primarily facilitates data transmission between the topside and internal computers. Incorporating a more powerful internal computer would enable the BROV to perform all necessary calculations onboard, eliminating the need for a tether and allowing fully untethered operation. To fully leverage this increased autonomy, enhancing the functionality of the mission planning software will also be critical. This advancement will ensure the vehicle can execute more complex and adaptive missions independently.

References

[1] Geoffrey N. Bailey, Jan Harff, and Dimitris Sakellariou, eds. *Under the Sea: Archaeology and Palaeolandscapes of the Continental Shelf*. en. Vol. 20. Coastal Research Library. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-53158-8 978-3-319-53160-1. DOI: 10.1007/978-3-319-53160-1. URL: http://link.springer.com/10.1007/978-3-319-53160-1 (visited on 04/15/2024).

[2] *International Handbook of Underwater Archaeology*. en. URL: https://link-springer-com.focus.lib.kth.se/book/10.1007/978-1-4615-0535-8 (visited on 04/05/2024).

[3] Li-Ying Hao et al. "Trajectory Tracking Control of Autonomous Underwater Vehicles Using Improved Tube-Based Model Predictive Control Approach". In: *IEEE Transactions on Industrial Informatics* 20.4 (Apr. 2024). Conference Name: IEEE Transactions on Industrial Informatics, pp. 5647–5657. ISSN: 1941-0050. DOI: 10.1109/TII.2023.3331772. URL: https://ieeexplore.ieee.org/document/10352935/?arnumber=10352935 (visited on 12/03/2024).

[4] Moritz Schulze Darup, Gerrit Book, and Pontus Giselsson. "Towards real-time ADMM for linear MPC: 18th European Control Conference, ECC 2019". In: *2019 18th European Control Conference, ECC 2019* (2019). Publisher: IEEE - Institute of Electrical and Electronics Engineers Inc., pp. 4276–4282. DOI: 10.23919/ECC.2019.8796239.

[5] Khai Nguyen et al. *TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers*. arXiv:2310.16985 [cs]. May 2024. DOI: 10.48550/arXiv.2310.16985. URL: http://arxiv.org/abs/2310.16985 (visited on 12/15/2024).

[6] Pablo Krupa et al. "Real-time implementation of MPC for tracking in embedded systems: Application to a two-wheeled inverted pendulum". In: *2021 European Control Conference (ECC)*. June 2021, pp. 669–674. DOI: 10.23919/ECC54610.2021.9654899. URL: https://ieeexplore.ieee.org/document/9654899 (visited on 12/15/2024).

[7] *SAABmarine project*. en. URL: https://github.com/SAABmarine-project (visited on 12/15/2024).

[8] Daniel Simon, Johan Lofberg, and Torkel Glad. "Reference Tracking MPC Using Dynamic Terminal Set Transformation". en. In: *IEEE Transactions on Automatic Control* 59.10 (Oct. 2014), pp. 2790–2795. ISSN: 0018-9286, 1558-2523. DOI: 10.1109/TAC.2014.2313767. URL: http://ieeexplore.ieee.org/document/6778075/ (visited on 12/15/2024).

[9] "(PDF) Sequential Quadratic Programming". en. In: *ResearchGate* (Oct. 2024). DOI: 10.1017/S0962492900002518. URL: https://www.researchgate.net/publication/230872679_Sequential_Quadratic_Programming (visited on 12/13/2024).

[10] *What Is Model Predictive Control?* URL: https://se.mathworks.com/help/mpc/gs/what-is-mpc.html (visited on 12/13/2024).

[11] *Adaptive MPC*. URL: https://se.mathworks.com/help/mpc/ug/adaptive-mpc.html (visited on 12/13/2024).

[12] "(PDF) Successive Linearization Based Model Predictive Control of Variable Stiffness Actuated Robots". en. In: *ResearchGate*. DOI: 10.1109/AIM.2017.8014275. URL: https://www.researchgate.net/publication/319122854_Successive_Linearization_Based_Model_Predictive_Control_of_Variable_Stiffness_Actuated_Robots (visited on 12/13/2024).

[13] *(PDF) Model predictive control with on-line optimal linearisation*. en. DOI: 10.1109/ISIC.2014.6967645. URL: https://www.researchgate.net/publication/289162831_Model_predictive_control_with_on-line_optimal_linearisation (visited on 12/13/2024).

[14] "A linear model predictive control algorithm for nonlinear largescale distributed parameter systems | Request PDF". en. In: *ResearchGate* (Oct. 2024). DOI: 10.1002/aic.12626. URL: https://www.researchgate.net/publication/230174167_A_linear_model_predictive_control_algorithm_for_nonlinear_large-scale_distributed_parameter_systems (visited on 12/13/2024).

[15] *Time-Varying MPC*. URL: https://se.mathworks.com/help/mpc/ug/time-varying-mpc.html (visited on 12/13/2024).

[16] Yidan Liu et al. "An Underwater Image Enhancement Method for Different Illumination Conditions Based on Color Tone Correction and Fusion-Based Descattering". en. In: *Sensors* 19.24 (Dec. 2019), p. 5567. ISSN: 1424-8220. DOI: 10.3390/s19245567. URL: https://www.mdpi.com/1424-8220/19/24/5567 (visited on 05/20/2024).

[17] Jie Li et al. “WaterGAN: Unsupervised Generative Network to Enable Real-time Color Correction of Monocular Underwater Images”. In: *IEEE Robotics and Automation Letters* (2017), pp. 1–1. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2017.2730363. URL: http://ieeexplore.ieee.org/document/7995024/ (visited on 05/20/2024).

[18] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. *Tackling the Generative Learning Trilemma with Denoising Diffusion GANs*. en. arXiv:2112.07804 [cs, stat]. Apr. 2022. URL: http://arxiv.org/abs/2112.07804 (visited on 05/20/2024).

[19] Hong-Gi Kim, Jung-Min Seo, and Soo Mee Kim. “Comparison of GAN Deep Learning Methods for Underwater Optical Image Enhancement”. en. In: *Journal of Ocean Engineering and Technology* 36.1 (Feb. 2022), pp. 32–40. ISSN: 1225-0767, 2287-6715. DOI: 10.26748/KSOE.2021.095. URL: http://joet.org/journal/view.php?doi=10.26748/KSOE.2021.095 (visited on 05/20/2024).

[20] Haitao Luo et al. “Large-Scale 3D Reconstruction from Multi-View Imagery: A Comprehensive Review”. en. In: *Remote Sensing* 16.5 (Jan. 2024), p. 773. ISSN: 2072-4292. DOI: 10.3390/rs16050773. URL: https://www.mdpi.com/2072-4292/16/5/773 (visited on 04/25/2024).

[21] Yang Fu et al. *COLMAP-Free 3D Gaussian Splatting*. arXiv:2312.07504 [cs]. Dec. 2023. DOI: 10.48550/arXiv.2312.07504. URL: http://arxiv.org/abs/2312.07504 (visited on 04/25/2024).

[22] *Doppler Velocity Log*. en-US. URL: https://waterlinked.com/dvl (visited on 12/15/2024).

[23] *BlueROV2 Heavy Configuration Retrofit Kit*. en-US. URL: https://bluerobotics.com/store/rov/bluerov2-upgrade-kits/brov2-heavy-retrofit/ (visited on 11/24/2024).

[24] Malte von Benzon et al. “An Open-Source Benchmark Simulator: Control of a BlueROV2 Underwater Robot”. In: *Journal of Marine Science and Engineering* 10 (Dec. 2022), p. 1898. DOI: 10.3390/jmse10121898.

[25] *Fig. 2. The reference frame used for analysis of BlueROV2 CC*. en. URL: https://www.researchgate.net/figure/The-reference-frame-used-for-analysis-of-BlueROV2-CC_fig2_353013107 (visited on 11/24/2024).

[26] *Building a Vehicle Frame ů GitBook*. URL: https://www.ardusub.com/quick-start/vehicle-frame.html (visited on 11/24/2024).

[27] *https://flex.flinders.edu.au/file/27aa0064-9de2-441c-8a17-655405d5fc2e/1/ThesisWu2018.pdf*. URL: https://flex.flinders.edu.au/file/27aa0064-9de2-441c-8a17-655405d5fc2e/1/ThesisWu2018.pdf (visited on 09/16/2024).

[28] Isah A. Jimoh and Hong Yue. "Path Following Model Predictive Control of a Coupled Autonomous Underwater Vehicle". In: *IFAC-PapersOnLine*. 15th IFAC Conference on Control Applications in Marine Systems, Robotics and Vehicles CAMS 2024 58.20 (Jan. 2024), pp. 183–188. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2024.10.052. URL: https://www.sciencedirect.com/science/article/pii/S240589632401807X (visited on 11/21/2024).

[29] Huixuan Fu et al. "Trajectory Tracking Predictive Control for Unmanned Surface Vehicles with Improved Nonlinear Disturbance Observer". en. In: *Journal of Marine Science and Engineering* 11.10 (Oct. 2023). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 1874. ISSN: 2077-1312. DOI: 10.3390/jmse11101874. URL: https://www.mdpi.com/2077-1312/11/10/1874 (visited on 12/13/2024).

[30] martkartasev. *martkartasev/SMARCUnityAssets*. original-date: 2023-11-01T13:18:30Z. Dec. 2024. URL: https://github.com/martkartasev/SMARCUnityAssets (visited on 12/13/2024).

[31] *Unity-Robotics-Hub/tutorials/ros_unity_integration/setup.md at main ů Unity-Technologies/Un* URL: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/ros_unity_integration/setup.md (visited on 11/21/2024).

[32] Md Jahidul Islam, Youya Xia, and Junaed Sattar. "Fast Underwater Image Enhancement for Improved Visual Perception". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020). Conference Name: IEEE Robotics and Automation Letters, pp. 3227–3234. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.2974710. URL: https://ieeexplore.ieee.org/document/9001231 (visited on 12/15/2024).

[33] Carsten Griwodz et al. "AliceVision Meshroom: An open-source 3D reconstruction pipeline". In: *MMSys '21: Proceedings of the 12th ACM Multimedia Systems Conference : ISBN: 978-1-4503-8434-6*. Istanbul, Turkey: ACM: Association for Computing Machinery, Sept. 2021, pp. 241–247. DOI: 10.1145/3458305.3478443. URL: https://hal.science/hal-03351139 (visited on 12/15/2024).

[34] Patrick Ng and Michael Krieg. "Modifications to ArduSub That Improve BlueROV SITL Accuracy and Design of Hybrid Autopilot". en. In: *Applied Sciences* 14.17 (Aug. 2024), p. 7453. ISSN: 2076-3417. DOI: 10.3390/app14177453. URL: https://www.mdpi.com/2076-3417/14/17/7453 (visited on 11/25/2024).

[35] B. Allotta et al. "An unscented Kalman filter based navigation algorithm for autonomous underwater vehicles". In: *Mechatronics* 39 (Nov. 2016), pp. 185–195. ISSN: 0957-4158. DOI: 10.1016/j.mechatronics.2016.05.007. URL: https://www.sciencedirect.com/science/article/pii/S095741581630037X (visited on 12/15/2024).

[36] Mohammad Taghi Sabet et al. "A Low-Cost Dead Reckoning Navigation System for an AUV Using a Robust AHRS: Design and Experimental Analysis". In: *IEEE Journal of Oceanic Engineering* 43.4 (Oct. 2018). Conference Name: IEEE Journal of Oceanic Engineering, pp. 927–939. ISSN: 1558-1691. DOI: 10.1109/JOE.2017.2769838. URL: https://ieeexplore.ieee.org/document/8126797 (visited on 12/15/2024).

[37] Xiaotian Wang et al. "An Overview of Key SLAM Technologies for Underwater Scenes". en. In: *Remote Sensing* 15.10 (Jan. 2023). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 2496. ISSN: 2072-4292. DOI: 10.3390/rs15102496. URL: https://www.mdpi.com/2072-4292/15/10/2496 (visited on 12/15/2024).

A Information modelling

[Back to information modelling section 4.1.1.]

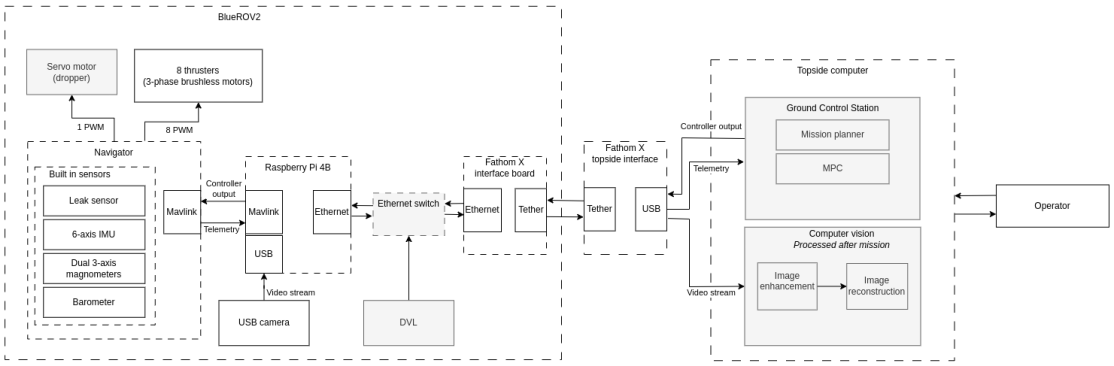


Figure A.1: Information model

B Simulink Model