Raybot

Design and control of an underwater quay wall inspection robot

Tijmen van Enckevort

Delft University of Technology

TU Delft

Raybot

Design and control of an underwater quay wall inspection robot

by

Tijmen van Enckevort

Student Name	Student Number
Tijmen van Enckevort	4552660

Academic Supervisor:	Dr. L. Ferranti
Supervisor:	B. Bootsma & R. Bérci-Hajnovics
Project Duration:	November, 2021 - December, 2022
Faculty:	Faculty of Mechanical, Maritime and Materials Engineering

Preface

My own robot! What a fantastic way to finish my academic life at the TU Delft. Laura, Réka, and Bart, I want to thank you from the bottom of my heart for all your guidance that has made my last months an incredible experience. Laura, I would like to thank you for all the insightful meetings and discussions we have had. Every single one of them has left me astonished, learning more and more about the deep insights of the wonderful world of model predictive control. I am deeply impressed by the work you and the researchers in your lab are doing and feel privileged to have had the chance to meet them and learn from them. Réka, you have made my first introduction with a real robot a blast. It is you who has translated my academic knowledge into a true understanding of robots. I would like to thank you for all the fun meetings and sessions we have had, and for all your patience in transmitting your engineering experience to me. Bart, I would also like to you. You have made the last phase of my thesis a joy, and I was always looking forward to meetings and office days with you. I have learned a lot from you, both in an engineering and academic way, and I will never forget your problem breaking-down and solving technique. I wish all of you the best on your future robotic endeavours!

Tijmen van Enckevort
Delft, December 2022

Abstract

Quay walls are important structures that keep the water in harbours and canals within their bounds, and accommodate large infrastructure like roads or ship-handling structures on top of them. Due to their importance it is critical that they do not fail or collapse. Inspections are done to prevent this, as a better knowledge of the state of quay walls can help in predicting future behaviour of the quay walls. However, current inspection methods are not satisfactory and can be improved upon. This thesis proposes the Raybot. This is an Autonomous Underwater Vehicle that can move along the quay walls like rays swim along the walls of their aquariums. The design of the Raybot is presented, which keeps in mind the requirements that follow logically from the quay wall inspection mission. This resulted in a rectangular robot with an outward modular frame. The thrusters, on-board computers, and sensors can be mounted on the inside. Next, a Model Predictive Controller is proposed for the motion control of the Raybot. For this, the extensive kinematics and kinetics of the Fossen model are explained. The parameters of this Fossen model are estimated for the Raybot using various methods. A Model Predictive Controller is formulated for Autonomous Underwater Vehicles and implemented in the Robot Operating System 2. A physical and visual representative simulation environment is set up, which can simulate the motion of the Raybot. This is used to assess the path tracking behaviour of the Raybot. The Model Predictive Controller is compared to a cascaded PID controller by path tracking of a zig-zag path along the quay wall. An analysis of the tuning parameters of the Model Predictive Controller is presented. The Model Predictive Controller outperforms the cascaded PID controller on both the Mean Squared Error of the path tracking error and the completion time of the inspection mission, whilst adhering to constraints set by the minimum and maximum velocity of the Raybot and the minimum and maximum thruster inputs. This improved performance leads to a higher quality of the inspections, as they can be done more up-close, as well as a higher quantity of the inspections, as more inspections can be done in the same time. For future work it is recommended to estimate some parameters of the model on a real robot, as well as testing the Model Predictive Controller on a real robot in a tank. This will showcase the performance of the motion controller in an even more realistic scenario.

Contents

1 Introduction

1.1. Collapsing Quay Walls

Over the years the human population has used water as a source of drinking water, irrigation and as a mode of transportation. Due to this development cities have formed around rivers leading to the construction of harbors and quay walls. Quay walls are large man-made underwater structures build to keep the water of the canal within the desired bounds of the structure. Also, infrastructure like roads or various ship handling structures are placed on these quay walls. Both keeping water in place and facilitating infrastructure make that quay walls are vital structures that must not fail. The impact of a failure can be enormous, as can be seen in the collapse of the Grimburgwal in Amsterdam in Figure 1.1. This lead to the canal being closed for a long period of time, and need for an emergency repair to prevent the foundation of closeby buildings from washing away.

Figure 1.1: The collapsed Grimburgwal in Amsterdam [6]

In order to prevent these quay walls from collapsing inspections can be done to gather knowledge of the state of the quay wall, and act proactively if there is a high risk of a potential collapse. There are various types of inspection methods, but one import type is the visual inspection. Here, data in the form of photographs, video's or descriptions from people looking at the wall is collected and used to assess if there is any damage on the wall. The formation of large cracks, buckling structures or pillars that have shifted from the original building plans can be identified this way. These visual methods can be performed in several ways and are outlined in Section 1.2.

1.2. Current Inspection Methods

At the moment inspections are carried out by diving teams. These consist of multiple highly specialized divers, safety personnel, and use specialized and costly equipment. The divers have to dive in places that can have low vision, unknown obstacles and sometimes sudden water currents. Hence, diving inspections are costly and dangerous. This aspect of being dangerous is also the reason why multiple divers are needed, so that they can assist each other during possible emergencies. Still, accidents can happen, even fatal ones. Take for example a diving inspection mission on a lock at Rijn bij Driel, which led to the death of a diver who was not able withstand the forces of the flow of water due to a leak in the lock wall [17]. The severe accidents happen mostly in the underwater part of the inspection. So, by decreasing the human involvement below the water surface, the risk of severe accidents can be decreased too.

In order to decrease this human involvement below the surface of the water, remote vehicles can be used. Remotely Operated Vehicles (ROV's) that can move underwater with the help of fins and/or thrusters exist. Operators can move these remotely along a quay wall. These ROV's can be equipped with different kinds of sensors and actuators, and can thus be used to make photographs or videos of the quay wall. These can be assessed real-time or offline by a human operator, which can then identify any damage that might be present on the quay wall. These ROV's are operated by an operator with a controller. Usually this operator has a live video feed of the camera's of the ROV in order to orientate and send commands to the robot. This way the need of divers underwater is eliminated, as they can be replaced with ROV's. This will make the inspection safer. Besides, the use of ROV's will create possibilities to perform complex maneuvers that would otherwise be too dangerous for the human divers, for example in confined spaces or inspections close to moving ships.

A drawback of this method with ROV's is the need for constant human supervision and operation. The human operator needs to pay attention to the movements of the ROV at all times, and needs to have a good understanding of the surroundings and obstacles the ROV might collide with. This makes the operation of a ROV difficult, and can lead to long inspections times and inaccurate results.

1.3. Inspection with Autonomous Underwater Vehicles

To improve the method with ROV's from Section 1.2 it is proposed to design an autonomous ROV that can inspect the quay walls without the need for operation by the human operators, also known as an Autonomous Underwater Vehicle (AUV). This will speed up the procedure, allowing for more quay walls to be inspected routinely. Routine inspection can improve the knowledge of the state of the quay walls, and can possibly be used to predict future behavior of the quay walls.

The AUV will be a robot that can autonomously follow a path along a quay wall. In order to do this it needs to adhere to certain requirements. First and foremost the AUV will need to take pictures of the quay wall. These pictures can then be used in photogrammetry software to create a 3D model of the quay wall, upon which a detailed assessment on the state of the quay wall can be made. The AUV may also house other sensors, like depth sensors, Inertial Measurement Units, and eddy current sensors. In essence, it will be a set of sensors that are moved along the quay wall to inspect it. In order to move the AUV will need thrusters. These thrusters need to be controlled with an onboard computer and motor controller. The sensors, thrusters, and computer need to be mounted on a frame so that they stay rigidly together. In order for the AUV to have an extended battery life, and also to get a working GPS signal for localization (GPS signals cannot be received under water) an Unmanned Surface Vehicle (USV) should be floating above the AUV. They will be connected to each other via a tether. This proposed solution can be seen in Figure 1.2.

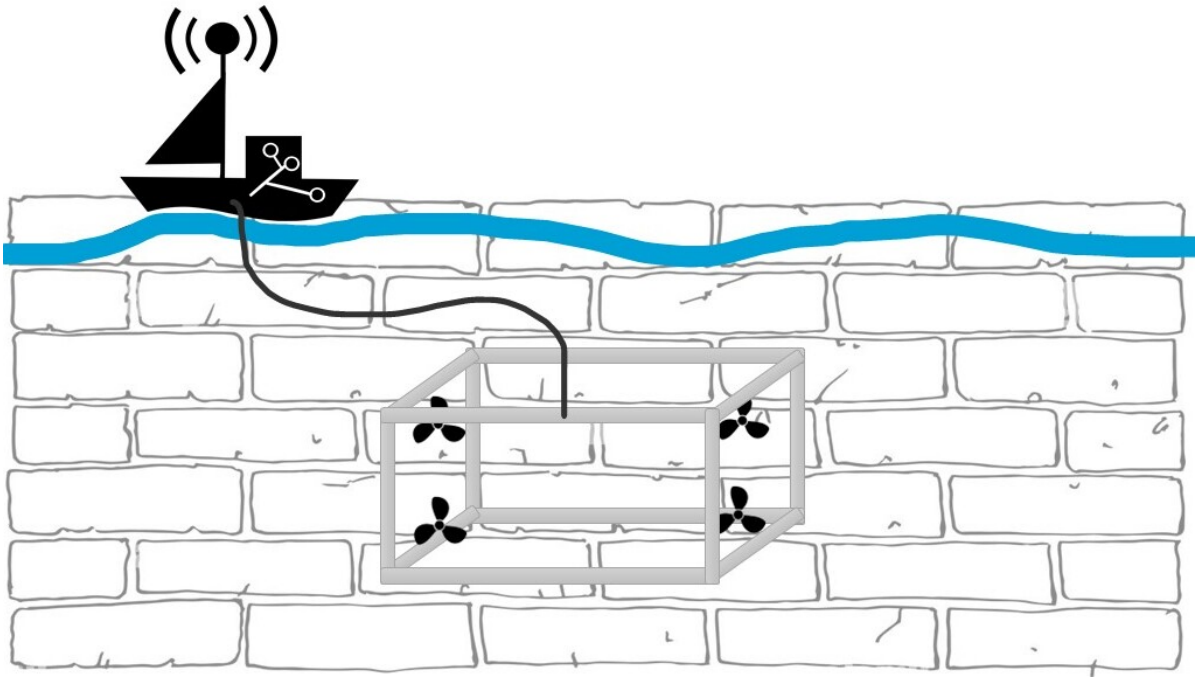


Figure 1.2: The proposed solution with an AUV and USV connected to each other by a tether. The USV can house extra batteries, a GPS, and on-board computers which can help the AUV with its inspection mission.

In order to complete the inspection autonomously the ROV needs to be able to follow a path along the quay wall. This path can be preprogrammed, commanded by a human operator or planned autonomously by a path planning algorithm on the ROV. Irrespective of how the path is computed, the ROV will need a control algorithm to make a translation between the desired positions of the path and the thruster inputs that will actually move the ROV along that path. There are several types of control algorithms, and each comes with its advantages and disadvantages. This thesis will focus on these control algorithms and their implementation in underwater robotics.

1.3.1. Motion Control

There are multiple motion control algorithms suitable to be used on a ROV. One division that can make is to classify them based on whether they need a model of the ROV or not. Classical control techniques, that do not need a model of the ROV, include Bang-Bang types of controllers, as well as the widely used Proportional Integral Derivative (PID) control algorithms. Model-based controllers include Linear Quadratic Regulator (LQR) control, and Model Predictive Control (MPC). Previous literature research has shown that MPC is the best suited control algorithm, due to its ability to handle Multi-input Multi-output (MIMO) systems, its ability to handle constraints and it has better performance compared to the other control techniques. This thesis will focus on the implementation of such a MPC controller for an AUV that can perform quay wall inspection missions, and will analyse its performance by comparing it with other existing controllers.

1.3.2. Research Questions

In order to implement and analyse the performance of the MPC controller for the AUV methodically a research question and several sub-research questions have been set up:

Research Question

- How can Model Predictive Control be used for an Autonomous Underwater Vehicle to improve the performance of quay wall inspection missions?

Sub-Research Questions:

- How can the inspection requirements be translated to a proof-of-concept AUV?
- How can the proof-of-concept AUV be modelled dynamically?
- How can the MPC controller of the AUV be tested in a simulation environment?
- How does the MPC controller compare to the baseline controller?

This thesis aims to answer these research questions. The structure of the thesis report is as follows. Chapter 2 explains the translation of the inspection requirements into a proof-of-concept AUV design that will be used as an use-case scenario for the remainder of the thesis. A model of the AUV dynamics is needed for the simulation environment and the model-based controller. This thesis uses the Fossen model for underwater vehicles [11]. The theory of the Fossen model and its corresponding parameter estimation will be outlined in Chapter 3. Chapter 4 outlines the MPC control algorithm and its implementation for the AUV. Chapter 5 explains the simulation environment that is used to test the AUV, and the setup of the simulation experiments. A simulation environment is used, as tuning and testing the controller in a real environment is expensive and will take long. A comparison between the performance of a cascaded PID and the MPC controller is presented in Chapter 6, where Chapter 7 contains an analysis and conclusion on these performance results, answering the main research question. Chapter 7 will also explain limitations and shortcomings of the thesis, as well as recommendations for future research. As a conclusion Chapter 8 answers the research questions concisely. The code for this thesis can be found on the open Github repository, where also the literature research mentioned in Section 1.3.1 can be found.

This thesis was done as a graduation project for the track BioMechanical Design of the MSc Mechanical Engineering at the Delft University of Technology, in collaboration with the company Dobots. Dobots provided the use-case scenario of the AUV.

2

Design of the Raybot Proof-of-Concept

This chapter aims to answer the first sub-research question as presented in Section 1.3.2: *How can the inspection requirements be translated to a proof-of-concept AUV?* The design requirements as given by Section 1.3 are further detailed in Section 2.1. These are translated into a proof-of-concept AUV called the Raybot, which is further explained in Section 2.2.

2.1. Design Requirements

The ultimate goal of the AUV is to take pictures of the quay wall, which can then be used in photogrammetry software to make a 3D model of the quay wall. For this it needs to take a picture of each section of the quay wall, with a certain minimum amount of overlap for the photogrammetry software to work properly. However, when it takes too many pictures the photogrammetry software will take longer to process its model, and the path along the quay will take longer to complete. For this reason the AUV needs to move vertically up and down the quay wall, with a distance in between each stroke to allow for the proper overlap. This zigzag path can be seen in Figure 2.1. As the planning of this zig-zag path is outside the scope of this thesis, it is assumed that this path is given to the AUV as an array of points containing desired positions and rotations, spaced a distance apart from each other. For now, it can be inputted by a human operator, but future prototypes can make use of a motion planning algorithm.

Figure 2.1: The zig zag path along the quay wall. The Raybot in the upper left corner will attempt to follow the green path, which will allow it to inspect the wall thoroughly. This path will be used in the simulation experiments.

The main design requirements follow from this mission path. It is desired for the AUV to move in all 6 degrees of freedom (DoF). Although the path only shows a translation in 2 degrees of freedom, the other 4 degrees of freedom are not constrained. The AUV will also operate closely to the quay wall. Hence, in order to avoid colliding into the wall or other obstacles full control over the other 4 degrees of freedom is required. To avoid collisions with the wall it is also desired to have an accurate control of the movements in the direction of the wall. The AUV needs to house a set of sensors in the middle of its frame, where the frame protects it from possible collisions. The dimensions of this set of sensors are 0.33m in x-direction, 0.94m in y-direction, 0.31m in z-direction and cannot extend out of the frame. The thrusters, and on-board computer and motor controllers to be used are the T200 thruster and control tube with electronics of Bluerobotics. These are open-source and are used extensively in previous projects of the company, which make them a desirable choice. The frame of the AUV needs to be modular and easily adjustable, with sufficient mounting places for all the components. This allows for easy adjusting of the layout of the AUV which speeds up prototyping for the first proof-of-concept.

Besides these requirements for the inspection mission there are also some requirements from a hydrodynamical point-of-view. First of all it is desired for the AUV to be neutrally buoyant. This means the buoyant forces due to volumetric displacement of water, due to the submerged AUV, are equal but opposite to the gravitational forces acting on the mass of the AUV, when the AUV is in upright position. Due to this there are no resultant forces acting on the AUV when the thrusters are not active. This makes controlling the AUV easier as no force is required to keep the AUV at a static position. Secondly, it is desired for the AUV to be passively stable. Although the buoyant forces and gravitational forces are equal and opposite they do not act on the same point on the AUV. The buoyant forces act on the centre of buoyancy (CoB), which is the volumetric centre of the AUV, whereas the gravitational forces act on the centre of gravity (CoG), which is the centre of mass of the AUV. This difference can be used to make the AUV passively stable. By placing the CoG directly below the CoB the force couple will always move the AUV into an upright position. Third, it is desired for the AUV to be fully symmetrical. This way the movements of the AUV are uncoupled. That is to say, a movement in one degree of freedom does not create any force or moment in any other DoF. This will make modelling and controlling the AUV more straightforward, as will also be explained in Chapter 3.

2.2. Design of the Raybot

The design requirements need to be translated into a design for a proof-of-concept AUV. This proof-of-concept AUV will be called Raybot hereafter, inspired by the way rays swim smoothly and closely to surfaces like walls of aquariums. The final Computer Aided Design (CAD) of the Raybot can be seen in Figure 2.2.

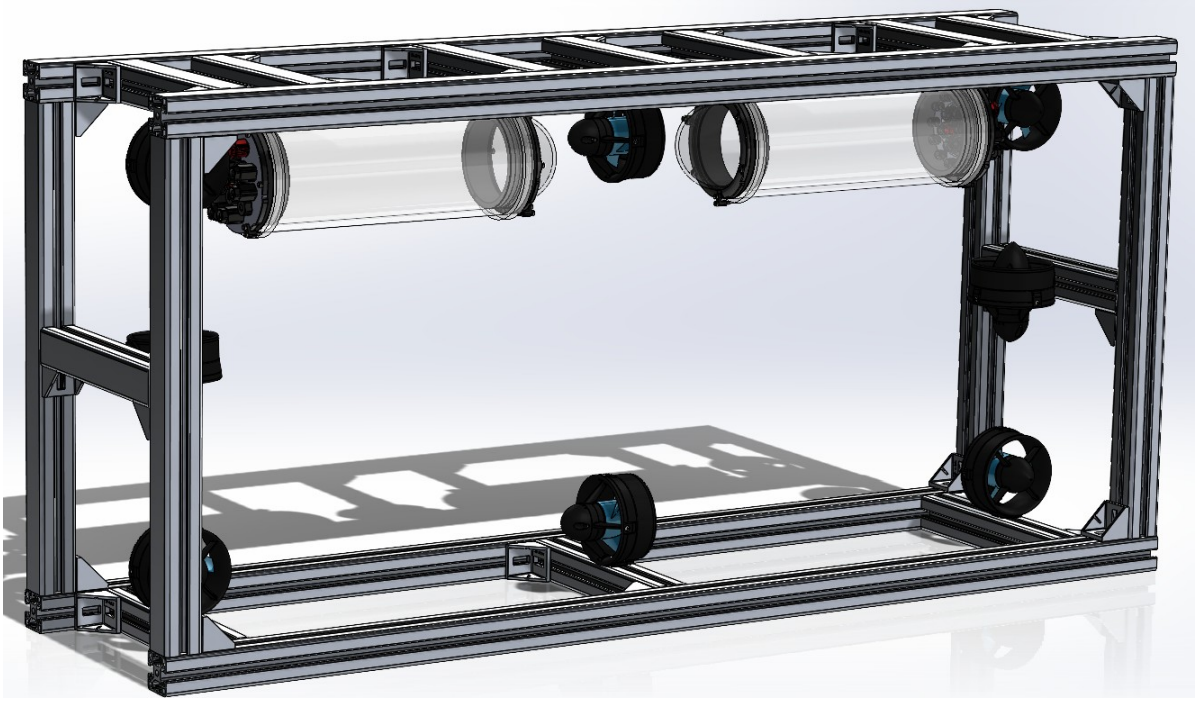


Figure 2.2: A CAD drawing of the Raybot design

2.2.1. Sensors

As stated in the Section 2.1 the Raybot should house a set of sensors with dimensions 0.33m in x-direction, 0.94m in y-direction, 0.31m in z-direction. This set of sensors is to be fitted inside of the frame, in order to protect it. However, this set of sensors is not yet fully defined, and thus their actual shape cannot be taken into account for the dynamical modelling and control. For this reason it was decided to leave an open space in the middle of the frame. The set of sensors can then be mounted inside this open space at a later moment.

2.2.2. Thrusters

The raybot needs to be able to move underwater. In order to do this it makes use of the aforementioned underwater thrusters of the type T200 by Bluerobotics. These thrusters can exert a force on the AUV by propelling a flow of water through it. This force is bounded by the current supplied to the thruster. In the backwards direction it is bounded by -28N at 12v, in the forward direction it is bounded by +36N at 12v. In order for the AUV to move in 6 degrees of freedom it needs to have a mimimum of 6 independent thrusters, with a couple of thrusters acting in each plane of the Raybot. Due to tight constraints in the x-direction of the AUV, the direction where it faces the wall, it is decided to use 2 extra thrusters. This means there will be 4 thrusters in the x-direction, one in each corner. This over actuation can help prevent a collision with the quay wall. The placement of these thrusters can be seen in Figure 2.2.

2.2.3. On-Board Computer

These thrusters and sensors need to be controlled and computed. For this, two modules with an on-board computer and an array of motor controllers are placed on the AUV. The on-board computer is an Raspberry Pi 4 model B. These Bluerov 2 electronics enclosures are placed at the top of the frame, due to its high buoyancy. This creates the desired passive stability. Also, because there are 8 thrusters present, two modules are needed to be able to drive all the 8 thrusters, as one module only goes up to 6 thrusters.

2.2.4. Frame

The main function of the frame of the AUV is to mount the set of sensors, thrusters and modules onto it. It also has to be rigid in order to protect all the expensive equipment whenever a collision occurs. For this reason it was chosen to use aluminium profiles. These aluminium profiles are easy to assem-

ble, are easily adjusted in case a different layout is needed, and it is straightforward to assemble the components onto them. Another benefit is that they will not rust as easily as steel frames. Because they are slightly heavier than water, the combination with the modules, which are buoyant, will result in a robot that is almost perfectly buoyant. This means a relatively low amount of adjusting with weights/buoyancy foam needs to be done. The dimensions of the frame are given by the layout of the set of sensors and the thrusters. It was chosen to place the thrusters around the set of sensors. This way the dimensions of the frame turn out to be 0.33m in x-direction, 1.29m in y-direction, 0.59m in z-direction.

2.3. Prototype

A first prototype of the Raybot can be seen in Figure 2.3. It was built according to the design from Section 2.2.

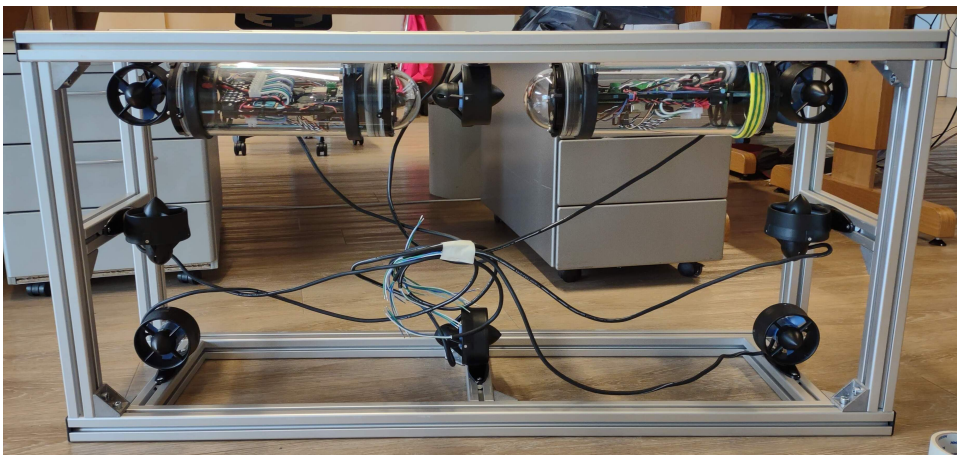


Figure 2.3: A prototype of the Raybot

3

Model

A model of the Raybot is needed in order to use a simulation environment and control the Raybot with a model-based controller. The model used in this thesis report is the Fossen model [11], and is used in both the simulator and in the prediction step of the MPC algorithm. This chapter will explore this model in more detail by both explaining the theory behind it, and a parameter estimation of the model on the Raybot use-case. This will give an answer to the second sub-research question as presented in Section 1.3.2: *How can the proof-of-concept AUV be modelled dynamically?*

3.1. The Fossen Model

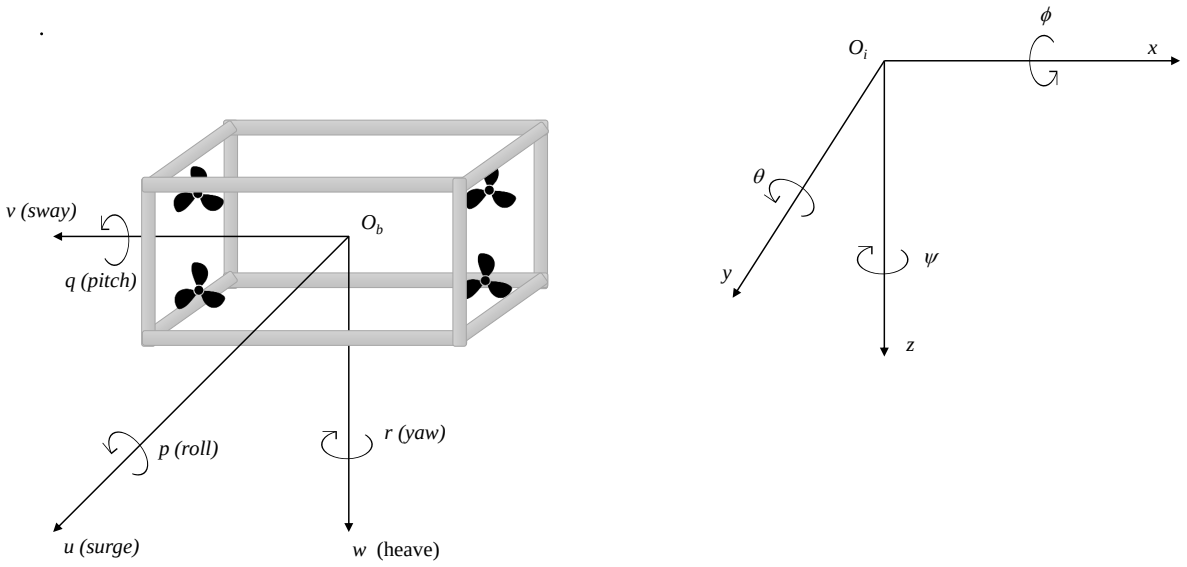


Figure 3.1: Motion of an underwater robot. O_i is the reference or inertial frame NED and O_b is the local robot frame BODY.

The underwater environment that the robot operates in is a 3D environment with 6 Degrees of Freedom (DOF), which can be seen in Figure 3.1. For consistency this thesis report uses the Society of Naval Architects and Marine Engineers (SNAME) notations to express the physical states of the robot, which can also be found in Table 3.1 [11]. Note that the position and orientation of the robot is expressed in the North East Down (NED) inertial frame of the robot whereas the velocities, and forces and moments of the robot are expressed in the Body frame of the robot.

Degree of Freedom	Position and orientation in Euler angles in NED frame	Linear and angular velocities in Body frame	Forces and moments in Body frame
surge	x	u	X
sway	y	v	Y
heave	z	w	Z
roll	ϕ	p	K
pitch	θ	q	M
yaw	ψ	r	N

Table 3.1: The SNAME notation which is used in the Fossen model.

3.1.1. Kinematics

The kinematic model describes the motion of the body of the robot without considering the forces and moments that are causing or resulting from those motions, see also Figure 3.1. The kinematics in Euler angles that describe the motion of the underwater vehicle can be found in Equation 3.1 [11].

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{\Phi}} \end{bmatrix} = \mathbf{J_e}(\phi) \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_b^i(\Phi)} & \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} & \boldsymbol{T_b^i(\Phi)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{bmatrix} \tag{3.1}$$

Here:

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3, \quad \boldsymbol{\Phi} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \in S^3, \quad \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \in \mathbb{R}^3, \quad \boldsymbol{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \in \mathbb{R}^3 \tag{3.2}$$

$\boldsymbol{R_b^i(\Phi)}$ is the linear velocity transformation matrix from the body frame BODY to the inertial frame NED given by:

$$\boldsymbol{R_b^i(\Phi)} = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\psi s\theta s\phi & -c\psi s\phi + s\psi s\theta c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{3.3}$$

Where $s = sin, c = cos$, and $t = tan$.

$\boldsymbol{T_b^i(\Phi)}$ is the angular velocity transformation matrix from the body frame BODY to the inertial frame NED given by:

$$\boldsymbol{T_b^i(\Phi)} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \tag{3.4}$$

Note that this kinematic model is given in the Euler representation and only holds when $\theta \neq 90$. If this does not hold a quaternion representation can be used instead.

3.1.2. Kinetics

In order to describe the motion of the robot body whilst taking into account the forces and moments that are acting on the body a kinematic model of the robot can be used. The equations of motion (EOM) of a kinetic robot model are given in Equation 3.5. Here, six different parts can be identified which together describe the motion of the robot. These six parts are further explained in the following sections.

$$\underbrace{\boldsymbol{M_{RB}\dot{v} + C_{RB}(v)v}}_{Rigid-Body} + \underbrace{\boldsymbol{M_A\dot{v}_r + C_A(v_r)v_r + D(v_r)v_r}}_{hydrodynamics} + \underbrace{\boldsymbol{b(R_b^i)}}_{restoring} = \underbrace{\boldsymbol{\tau_{thrusters}}}_{actuation} + \underbrace{\boldsymbol{\tau_{cable}}}_{tether} + \underbrace{\boldsymbol{\tau_{dist}}}_{disturbances} \tag{3.5}$$

Rigid Body Motions

The rigid body kinetics of an underwater robot can be described by [11]:

$$\mathbf{M}_{RB}\dot{\boldsymbol{v}} + \mathbf{C}_{RB}(\boldsymbol{v})\boldsymbol{v} = \boldsymbol{\tau}_{RB} \tag{3.6}$$

Here, the $\boldsymbol{v}$ is the generalized velocity vector in the body frame, where $\boldsymbol{v} = [u, v, w, p, q, r]^T$ and the $\mathbf{M}_{RB}$ is the rigid body mass matrix. It is defined as:

$$\mathbf{M}_{RB} = \begin{bmatrix} m\mathbf{I}_{3\times3} & -m\mathbf{S}(\mathbf{r}_g^b) \\ m\mathbf{S}(\mathbf{r}_g^b) & \mathbf{I}_b \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & -mx_g & 0 \\ 0 & -mz_g & my_g & I_x & -I_{xy} & -I_{xz} \\ mz_g & 0 & -mx_g & -I_{yx} & I_y & -I_{yz} \\ -my_g & mx_g & 0 & -I_{zx} & -I_{zx} & I_z \end{bmatrix} \tag{3.7}$$

$\mathbf{S}(\mathbf{r}_g^b)$ is a skew-symmetric matrix, where $\mathbf{r}_g^b$ is the location of the Centre of Gravity (CoG) with respect to the center of origin. It is defined as:

$$\mathbf{S}(\mathbf{r}_g^b) = \begin{bmatrix} 0 & -z_g & y_g \\ z_g & 0 & -x_g \\ -y_g & x_g & 0 \end{bmatrix} \tag{3.8}$$

Note that when the CoG is the same as the centre of origin: $\mathbf{S}(\mathbf{r}_g^b) = \mathbf{0}_{3\times3}$ and

$$\mathbf{M}_{RB} = \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_b \end{bmatrix} \tag{3.9}$$

$\mathbf{C}_{RB}(\boldsymbol{v})$ is the Coriolis and centripetal matrix. If it is noted that $\boldsymbol{v}_1 := \boldsymbol{\nu}_{b/n}^b = [u, v, w]^T$, and $\boldsymbol{v}_2 := \boldsymbol{\omega}_{b/n}^b = [p, q, r]^T$, $\mathbf{C}_{RB}(\boldsymbol{v})$ can be written in the following lagrangian parametrization:

$$\mathbf{C}_{RB}(\boldsymbol{v}) = \begin{bmatrix} \mathbf{0_{3\times3}} & -m\mathbf{S}(\boldsymbol{v_1}) - m\mathbf{S}(\mathbf{S}(\boldsymbol{v_2})\mathbf{r_g^b}) \\ -m\mathbf{S}(\boldsymbol{v_1}) - m\mathbf{S}(\mathbf{r_g^b})\mathbf{S}(\boldsymbol{v_2}) & -\mathbf{S}(\mathbf{I_b}\boldsymbol{v_2}) \end{bmatrix} \tag{3.10}$$

Hydrodynamics

To accurately model the forces of the surrounding fluid on the body moving under the water surface one has to solve the Navier-Stokes equations. However, these are computationally expensive and are not feasible in a real-time setting. Another approach is the use of a so-called added mass matrix. This added mass matrix can be seen as a virtual mass added to the inertia of the body that is caused by the accelerating of the surrounding fluid due to the motion of the body.

The added mass matrix is defined as:

$$\mathbf{M}_A = \begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \tag{3.11}$$

The hydrodynamic Coriolis and centripetal matrix is defined as:

$$\mathbf{C}_A = \begin{bmatrix} \mathbf{0}_{3\times3} & -\mathbf{S}(\mathbf{A}_{11}\boldsymbol{v}_1 + \mathbf{A}_{12}\boldsymbol{v}_2) \\ -\mathbf{S}(\mathbf{A}_{11}\boldsymbol{v}_1 + \mathbf{A}_{12}\boldsymbol{v}_2) & -\mathbf{S}(\mathbf{A}_{21}\boldsymbol{v}_1 + \mathbf{A}_{22}\boldsymbol{v}_2) \end{bmatrix} \tag{3.12}$$

Also, the relative velocity $\boldsymbol{v_r}$, which is caused by water currents, is given by $\boldsymbol{v_r} = \boldsymbol{v} - \boldsymbol{v_c}$, where $\boldsymbol{v_c} = [u_c, v_c, w_c, 0, 0, 0]$ and consists of the velocity components of the water current.

The coefficients of the added mass matrix $\mathbf{M_A}$ can be found numerically or empirically.

Damping

Besides the added mass to the body, there is also a damping effect due to the body moving in a fluid. The damping effects are highly complex and not fully understood yet. This makes them hard to model. An approach to model the damping is by defining:

$$\mathbf{D}(\boldsymbol{v}_r) = \mathbf{D_l} + \mathbf{D}_{nl}(\boldsymbol{v}_r) \tag{3.13}$$

It is composed of the linear damping matrix $\mathbf{D_l}$ and the nonlinear damping matrix $\mathbf{D}_{nl}(\boldsymbol{v}_r)$. In the assumption that the symmetric robot moves at low velocities and below the water surface the linear damping matrix is due to viscous skin friction, whereas the nonlinear damping matrix is due to vortex shedding. [10].

$$\begin{aligned} \mathbf{D_l} &= diag(X_u, Y_v, Z_w, K_p, M_q, N_r) \\ \mathbf{D_{nl}} &= diag(X_{u|u|}, Y_{v|v|}, Z_{w|w|}, K_{p|p|}, M_{q|q|}, N_{r|r|}) \end{aligned} \tag{3.14}$$

The coefficients of the nonlinear damping matrix $\mathbf{D}_{nl}(\boldsymbol{\nu}_r)$ can be described by one of the terms of the Morison equation [10]:

$$\mathbf{F}_D = \frac{1}{2}\rho C_D A_p |u| u \tag{3.15}$$

Here, ρ is the water density, C_D is the drag coefficient, A_p is the projected cross-sectional area and u is the velocity of the robot body. The drag coefficients C_D are to be determined empirically, as are the coefficients of the linear damping matrix $\mathbf{D}_{nl}$.

Restoring Forces

The hydrostatic or restoring forces acting on the robot are caused by both gravitational forces, and buoyancy forces due to the robot being submerged in water. The gravitational force is defined as $W = mg$, where it acts on the CoG of the robot in the positive direction along the heave axis.

The buoyancy force is the force is caused by the displacement of water due to the robot being submerged under water. It is defined as $B = \rho \nabla g$ where it acts on the Centre of Buoyancy (CoB). This is the volumetric centre of the body. It acts in the negative direction along the heave axis. Both gravitational and buoyancy forces are expressed in the inertial frame i as:

$$\mathbf{f}_G^i = W \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{f}_B^i = B \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{3.16}$$

In the body frame b the restoring forces and moments are given by:

$$\mathbf{b}(\mathbf{R}_b^i) = \begin{bmatrix} \mathbf{R}_b^{i\,T}(\mathbf{f}_G^i + \mathbf{f}_B^i) \\ \mathbf{r}_G^b \times \mathbf{R}_b^i \mathbf{f}_G^i + \mathbf{r}_B^b \times \mathbf{R}_b^{i\,T} \mathbf{f}_B^i \end{bmatrix} \tag{3.17}$$

These are the two forces acting on the robot in hydrostatic conditions, i.e. when there is no relative flow of water across the robot body ($\mathbf{v_r} = 0$).

Actuation

The body of the robot is being actuated with underwater thrusters. For a fully actuated system the amount of independently orientated thrusters r should be equal to the number of DoF n the robot is planning to move in. In the case that $r > n$ the system is over actuated and the thruster inputs have to be allocated using an optimization algorithm. One way to model the individual thruster output is:

$$f = kf^2 \tag{3.18}$$

Here, u is the thruster force, k is the thruster coefficient, and f is the input in rotations per minute. The thruster coefficient can be determined from the relationship between the thruster force and the rotations per minute of the thruster.

The sum of the force and moments from the thrusters $\boldsymbol{\tau_{thrusters}}$ are expressed as:

$$\boldsymbol{\tau}_{thrusters} = \mathbf{Tu} \tag{3.19}$$

$$= \mathbf{Tf^T Kf} \tag{3.20}$$

Here, $\mathbf{T}$ is the Thruster Allocation Matrix, $\mathbf{u} = [u_1, u_2, ..., u_n]$ is the thruster force vector with n thrusters, $\mathbf{K}$ is a matrix with thruster coefficients, and $\mathbf{f}$ is the vector with thruster inputs.

Cable Forces

This section is about the forces that are present due to the tether that is coming from the USV to the AUV. This tether can exert forces acting on its attachment point on the robot. These forces are caused by drag due to water currents, and/or a nonzero robot vehicle velocity in the tangential direction of the tether, and hydrostatic forces due to gravity or buoyancy.

According to [2] the cable forces can be modelled as:

$$\tau_{cable} = \begin{bmatrix} -\frac{1}{4}\rho d C_d \int_0^h |u_r| u_r \mathrm{d}z \\ -\frac{1}{4}\rho d C_d \int_0^h |v_r| v_r \mathrm{d}z \\ 1.2 w_{cable} h \\ r_z \tau_{cable}(1) \\ r_z \tau_{cable}(2) \\ 0 \end{bmatrix} \tag{3.21}$$

Here, the ρ is the density of the water, d is the diameter of the cable, C_d is the drag coefficient of the cable, h is the depth of the AUV relative to the USV, w_{cable} is the weight of the cable submerged in water, and u_r and v_r are the velocities in the u and v direction of the current relative to the tether. The u_r and v_r can be caused by both the water current velocity as the velocity of the AUV in the horizontal plane.

3.2. Parameter Estimation of the Raybot

The Fossen model describes the motion of the underwater vehicle on a more abstract and generalised level, but the parameters still need to be determined in order for the model to be usable in simulation or for a control algorithm. The following sections explain how these parameters are determined and estimated for the Raybot for each of the terms of the Fossen model.

3.2.1. Mass/Inertia/Restoring forces

In order to determine the mass and the moments of inertia of the Raybot a 3D cad model is made, see Figure 2.2. This model included all the parts and components, including their mass and mass distribution properties. From this model it is possible to let Solidworks calculate the total mass and the total mass distribution, which can be used to calculate the moments of inertia in each of the axes of the robot [23]. The mass matrix for the Raybot is given by.

$$\mathbf{M}_{RB} = \begin{bmatrix} 26.33 & 0 & 0 & 0 & 0 & 0 \\ 0 & 26.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & 26.33 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6.28 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.96 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5.32 \end{bmatrix} \tag{3.22}$$

The restoring forces consist of the gravitational and buoyancy forces. The scalar force component of the gravity vector can be calculated by: $F = m * g$, where F is the gravitational force component, $m = 26.33$ is the mass of the robot and $g = 9.81$ is the gravitational pull of the earth. The point of application of the gravitational force is the centre of mass of the robot, and it acts in the downwards z-direction.

The scalar force component of the buoyancy forces can be calculated by determining the volumetric displacement of the robot. For the Raybot it is equal to: $0.02633 m^3$. The point of application is the volumetric center of the Raybot which has been determined via Blender to be $(x = 0, y = 0, z = -0.2)$.

3.2.2. Added Mass

The added mass components of the Raybot can be determined via a numerical simulation in a wave energy numerical simulator. This thesis chooses to use the open-source Nemoh simulator to do this [1].

A simplified version of the 3D CAD model of the Raybot is needed to compute the added mass terms. This is useful in order to decrease the computational time due to the large amount of vertices in the original 3D model. This model can be seen in figure 3.2. This 3D model has to be converted to a Nemoh.dat file. This was done by using the open-source software BEM-rosetta and Blender [**zabala**], [7]. First the Solidworks model was converted to a .stl file which can be loaded into Blender. Here, it is important to scale the mesh back to the correct sizes, and to place the robot at a z-location of -50m. This last step is important as to the best of the authors knowledge it was not possible to use Nemoh without any wave interactions, while an assumption that there are no wave interactions were made in the model. At a depth of -50m the wave interactions calculated by Nemoh can be neglected. This .stl file can be imported into BEM-rosetta. Here, the edges and nodes are cleaned up, and a Nemoh.cal file is set up. Afterwards a Nemoh mesh file can be created. This mesh file can then be used to run the numerical simulations via a Matlab script. The results of these numerical simulations can then be loaded back into BEM-Rosetta, via the hydrodynamic parameters tab. This method has been validated, as can be seen in Appendix B. Now the added mass properties are given by the A∞ matrix in BEM-rosetta, which is the $\mathbf{M}_A$ and given by:

$$\mathbf{M}_A = \begin{bmatrix} 27.96 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20.35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 29.58 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3.90 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.48 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.06 \end{bmatrix} \tag{3.23}$$

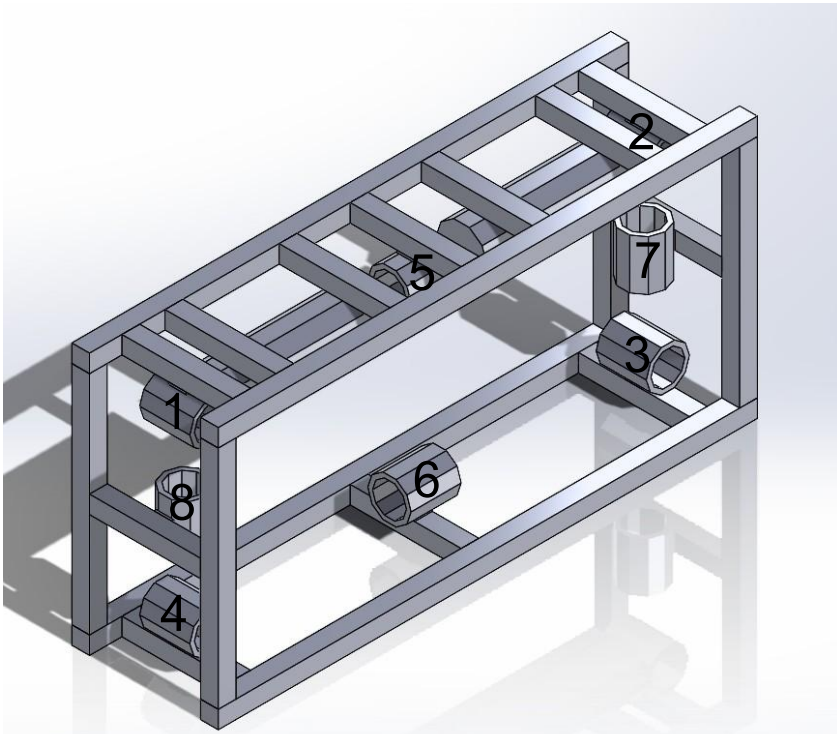


Figure 3.2: A simplified CAD model of the Raybot which is used in the Added Mass simulations

3.2.3. Damping

The damping terms of the model can not be determined via software. Therefore, it was chosen to build a small model of the robot and to use this in towing tests in a small tank of water, see Figure 3.3a. The scaling factor $\lambda = 4.44$. The experimental setup can be seen figure 3.3b, and is similar to the one used in [18]. By applying a constant force or moment to each of the degrees of freedom and measuring the linear and angular velocities, the linear and quadratic components of the damping terms can be determined for the small model, by superimposing them. These can then be scaled up to the damping terms of the real robot via the correct damping scaling factors. These damping scaling factors are the scaling factor λ^2 for the quadratic components, and the scaling factor λ^2 for the linear components. These follow from the following equation [2]:

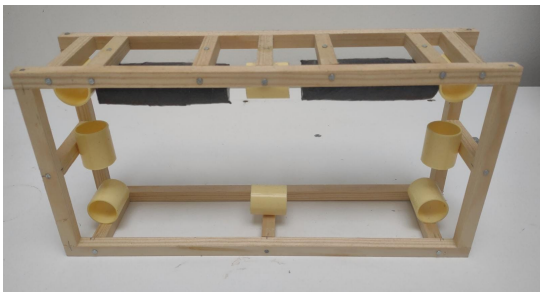
$$F_r = \frac{\rho_r}{\rho_m}\lambda^3 F_m \tag{3.24}$$

Here, F_r are the quadratic damping forces of the Raybot, F_m are the quadratic damping forces of the scaled model, ρ_r is the water density of the Raybot, ρ_m is the water density of the scaled model and λ is the scaling factor. The damping scaling factor for the linear damping terms can best be estimated by applying a scaling factor of λ^2

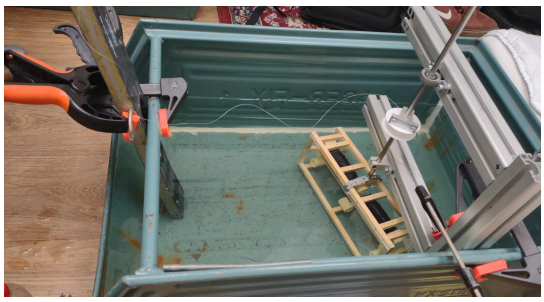
The damping terms that follow from these small mockup model tests are given as:

$$\begin{aligned} \mathbf{D_l} &= diag(24.65, 56.70, 47.01, 0.0312, 0.07, 0.2956) \\ \mathbf{D_{nl}} &= diag(747.06, 181.73, 681.28, 1.9088, 1, 1.33) \end{aligned} \tag{3.25}$$

As the tank was not large enough accommodate a measurement on the M_q and $M_{q|q|}$ terms these were estimated based on values of another robot, the Bluerov robot [25].



(a) Scaled model version of the Raybot



(b) Test-setup for the damping parameter estimation

3.2.4. Thruster Forces

The thruster forces are determined via the datasheet on the Bluerobotics website. Here, the RPM of the thrusters is given against the force. This shows a quadratic relationship for the force outputted by the thrusters given a certain RPM. A different relationship is shown for the backwards force compared to the forward force of the thruster. Following the force-RPM curves as given in [3] the coefficient of the T200 thrusters is determined as $k = 0.000033$

The thruster allocation matrix $\boldsymbol{T}$ is the contribution per thruster (the 8 columns) to that specific force or moment per principal axis (the 6 rows). Following the thruster layout as seen in Figure 3.2, the thruster allocation matrix for the Raybot is given as:

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -0.20 & 0.20 & 0.51 & -0.51 \\ 0.20 & 0.20 & -0.20 & -0.20 & 0 & 0 & 0 & 0 \\ 0.51 & -0.51 & -0.51 & 0.51 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.26}$$

3.2.5. Cable forces

The Raybot will follow a zig-zag path where the longest direction of travel will be in the vertical z direction. This makes that the u_r and v_r will generally be small which causes that the cable forces in the X and Y direction of the AUV can be neglected as they are scaling quadratically with their corresponding velocity directions. Furthermore it can be assumed the tether is neutrally buoyant, with its center of gravity and center of buoyancy being in the same position. This causes the underwater weight of the tether w_{cable} to be approximately equal to 0. Taking also in account that the Raybot is not expected to operate at very deep depths, the length h of the tether cord will be relatively small. This diminishes all the elements in the cable force vector even further. Following these arguments it can be argued that the cable forces acting on the Raybot can be assumed to be negligible.

$$\tau_{cable} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.27}$$

4

Model Predictive Control

A model predictive controller is a controller that uses a model of the system to make a finite time horizon prediction on the future states of the system and computes optimal control outputs by minimizing a cost function [20], [13]. Over the last years it has proven to be a popular choice for controlling systems. This is mainly due to its ability to handle constraints on the system, it is able to handle MIMO systems, it is able to take into account future reference information, and it generally has a good performance. A MPC controller works in three stages: prediction, optimization, and the receding horizon principle.

In the first stage, the prediction stage, the MPC controller uses the model of the system to compute the system states based on possible control actions. It does this on a finite future time horizon, starting from an initial state.

In the optimization stage the MPC controller computes the optimal control input sequence by minimising a cost function J while obeying to the state or control constraints of the system.

In the receding horizon principle stage the MPC controller applies the first input of the optimal control sequence to the system, and then moves one timestep forward. Then it iterates and goes back to the prediction stage. This way, the controller applies the optimal input every timestep which makes that the MPC controller generally has a good performance.

This chapter will give a partial answer to the third sub-research question as presented in Section 1.3.2: *How can the MPC controller of the AUV be tested in a simulation environment?*. It does so by explaining both the formulation and the implementation of the MPC controller for an AUV.

4.1. Formulation

The Fossen model from Chapter 3 can be written in a more shorthand notation as [22]:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{R}(\psi)\boldsymbol{v} \\ \boldsymbol{M^{-1}}(\boldsymbol{Tu} - \boldsymbol{Dv} - \boldsymbol{b})) \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{4.1}$$

Here, $\boldsymbol{x}$ is the state of the system and defined as $\boldsymbol{x} = [x, y, z, \phi, \theta, \psi, u, v, w, p, q, r]$. The velocity of the AUV is given as $\boldsymbol{v} = [u, v, w, p, q, r]$, $\boldsymbol{M}$ is the mass matrix including the added mass and is given as $\boldsymbol{M} = \boldsymbol{M_{RB}} + \boldsymbol{M_A}$, $\boldsymbol{T}$ is the thruster allocation matrix, the control vector $\boldsymbol{u} = [u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8]$, the damping terms are $\boldsymbol{D} = \boldsymbol{D_l} + \boldsymbol{D_{nl}}$, and $\boldsymbol{b}$ are the restoring forces.

$\boldsymbol{R}(\psi)$ is a rotation matrix around the yaw-axis ψ and is given as:

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Note that this is different from the full rotation matrix around the 3 yaw, pitch, roll axis ϕ, θ, ψ. Due to small roll and pitch angles, $\phi \approx 0, \theta \approx 0$, this assumption can be made and will decrease the computational effort needed to compute the dynamics. Due to small angular velocities $p \approx 0, q \approx 0, r \approx 0$ the coriolis term in the Fossen model can be assumed to be zero.

The system has both input and state constraints. The state constraints are composed of the constraints on linear and angular velocities of the AUV, see Equation 4.3. These constraints are present to ensure that the robot does not move underwater too fast. This is desirable as the camera has to take pictures which will otherwise be blurry, and because in a test setup it is desirable to move at low speeds for safety reasons. The input constraints are composed of the minimum and maximum force output bounds of the thrusters, see Equation 4.4. As the maximum force output of the thrusters is different in the direction of movement of the thrusters, this bound value is different in the negative direction compared to the positive direction.

$$\underline{\boldsymbol{x}} = \begin{bmatrix} -0.5 \ [m/s] \\ -0.5 \ [m/s] \\ -0.5 \ [m/s] \\ -0.5 \ [rad/s] \\ -0.5 \ [rad/s] \\ -0.5 \ [rad/s] \end{bmatrix} \leq \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \leq \begin{bmatrix} 0.5 \ [m/s] \\ 0.5 \ [m/s] \\ 0.5 \ [m/s] \\ 0.5 \ [rad/s] \\ 0.5 \ [rad/s] \\ 0.5 \ [rad/s] \end{bmatrix} = \overline{\boldsymbol{x}} \tag{4.3}$$

$$\underline{u_i} = \begin{bmatrix} -28 \ [N] \end{bmatrix} \leq \begin{bmatrix} u_i \end{bmatrix} \leq \begin{bmatrix} 36 \ [N] \end{bmatrix} = \overline{u_i} \tag{4.4}$$

These continuous-time dynamics are discretized to $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and integrated using an implicit Runge-Kutta integrator of 4th order. The discrete-time cost function J is then given as:

$$\min_{\boldsymbol{x}_\tau, \boldsymbol{u}_\tau} \quad J = \sum_{k=0}^{N-1} ((\boldsymbol{x}_k - \boldsymbol{x}_{ref})^\top \boldsymbol{Q} (\boldsymbol{x}_k - \boldsymbol{x}_{ref}) + \boldsymbol{u}_k^\top \boldsymbol{R} \boldsymbol{u}_k) + ((\boldsymbol{x}_N - \boldsymbol{x}_{ref,N})^\top \boldsymbol{Q}_N (\boldsymbol{x}_N - \boldsymbol{x}_{ref,N}) + \boldsymbol{u}_N^\top \boldsymbol{R}_N \boldsymbol{u}_N) \tag{4.5}$$

Here, x_τ is the state trajectory $\boldsymbol{x}_\tau = \boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_N$, $\boldsymbol{u}_\tau$ is the control trajectory $\boldsymbol{u}_\tau = \boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_N$, $\boldsymbol{x}_k$ is the state at timestep k, $\boldsymbol{x}_{ref}$ is the reference state, $\boldsymbol{u}_k$ is the control input at timestep k, $\boldsymbol{x}_N$ is the terminal state and $\boldsymbol{u}_N$ is the terminal control input. $\boldsymbol{Q}$ is the weight matrix on the state and $\boldsymbol{R}$ is the weight matrix on the control input. $\boldsymbol{Q}_N$ is the weight matrix on the terminal state and $\boldsymbol{R}_N$ is the weight matrix on the terminal control input. $Cost_{pos}$ is the cost parameter for state errors on position, $Cost_{rot}$ is the cost parameter for state errors on rotation, $Cost_{thrust}$ is the cost parameter for the thruster inputs.

$$\begin{gathered} \boldsymbol{Q} = diag([Cost_{pos}, Cost_{pos}, Cost_{pos}, Cost_{rot}, Cost_{rot}, Cost_{rot}, 0, 0, 0, 0, 0, 0]) \\ \boldsymbol{Q}_N = 2 * \boldsymbol{Q} \\ \boldsymbol{R} = diag([Cost_{thrust}, Cost_{thrust}, Cost_{thrust}, Cost_{thrust}, Cost_{thrust}, Cost_{thrust}, Cost_{thrust}, Cost_{thrust}]) \\ \boldsymbol{R}_N = 2 * \boldsymbol{R} \end{gathered} \tag{4.6}$$

When the cost function, dynamical model, and the state and input constraints are combined the MPC controller can be formulated as:

$$\begin{gathered} \min_{\boldsymbol{x}_\tau, \boldsymbol{u}_\tau} \ J \\ s.t. \begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \boldsymbol{x}_0 = \boldsymbol{x}_{init} \\ \underline{\boldsymbol{x}} \leq \boldsymbol{x}_k \leq \overline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}} \end{cases} \end{gathered} \tag{4.7}$$

Figure 4.1 shows the MPC controller architecture for the AUV [16]. Here, i is the global time and $\hat{x}$ is the predicted state x.

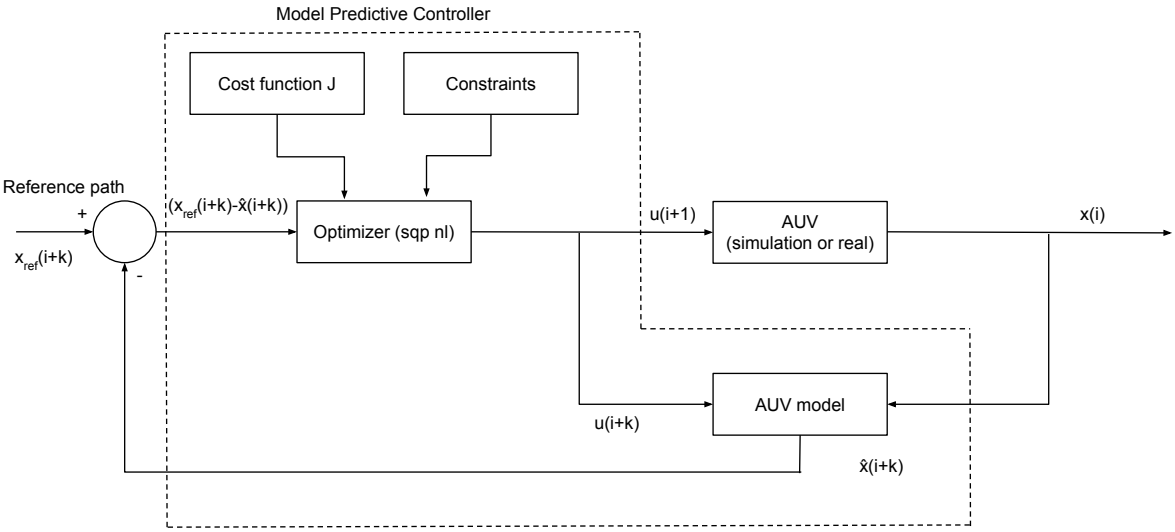


Figure 4.1: MPC controller structure that is used in the Raybot

4.2. Implementation

The implementation of the MPC controller in Robot Operating System 2 (ROS2) can be seen in Figure 4.2 [24], [15]. The pose of the AUV is published on a topic `/raybot/pose_gt`. The `/raybot/mpc_controller` node subscribes to this topic and uses the pose together with the reference path to compute the error on the state of the AUV. To compute the optimal control inputs the `/raybot/mpc_controller` uses the ForcesPro solver [26], [8]. Then it sends the first control inputs of the optimal control input sequence to the `/raybot/thrusters/id_<x>/input` topic. The code for this implementation can be found in [9] in the `uuv_mpc_control` package.

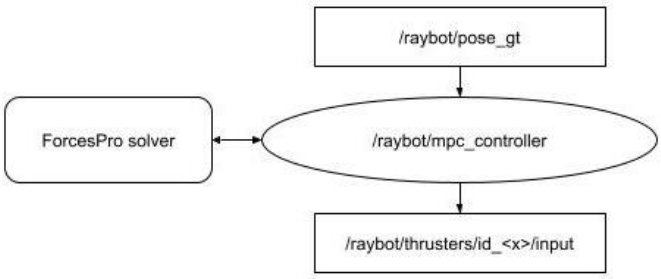


Figure 4.2: MPC controller structure in ROS2, with the ForcesPro solver

The `/raybot/mpc_controller` node receives a pose message at 50 Hz. Once received, it will use the ForcesPro solver to compute the optimal input sequence via a Sequential Quadratic Programming (SQP) method, and send it immediately to the thrusters. Afterwards it will wait till 0.5 seconds have passed since the pose was received, and then iterates. This way the solver will always send a command to the thrusters every 0.5 seconds, given that the time it takes to compute and send the thruster inputs does not exceed 0.5 seconds. For this reason the ForcesPro integrator stepsize is also set to 0.5 seconds.

5 Method

The behaviour of the Raybot and performance of the control algorithms will be tested in a simulation environment. This Chapter will first explain the simulation environment in Section 5.1, by explaining the setup in the Gazebo simulator, and the necessary plugins and setup required to get an accurate physical simulation of the Raybot. Section 5.2 describes the setup and method of the experiments that have been performed in the simulation environment. This will give an answer to the second sub-research question as presented in Section 1.3.2: *How can the MPC controller of the AUV be tested in a simulation environment*

5.1. Simulation Environment

In order to test the AUV without having to do real tests, which are expensive, time consuming and can lead to damage to the robot, it was decided to do simulation tests. For this a physically accurate simulator was needed. Gazebo Classic is a simulation environment that is able do this. Also, because of its easy integration with the Robot Operating System 2 (ROS 2) it allows for easy testing of the AUV. In order to simulate the underwater environment and model of the AUV as explained in Section 3 a plugin can be used. This plugin uses the same Fossen model to compute the forces present on the AUV. This plugin is the `uuv_simulator` package. This is a package developed by the SWARMs project and works with Gazebo classic and ROS 1 [14], [19]. In order to use it with ROS 2 a fork of the package called Plankton is needed [12]. With Plankton and Gazebo a simulation environment that works together seamlessly with ROS 2 is created that can accurately simulate the behavior or the AUV. Because of the integration with ROS 2 the algorithms used can be directly implemented on the real robot, decreasing the effort needed to translate between simulation and the real world. For an overview of the simulation environment see Figure 5.1.

As the AUV makes use of cameras to do its inspection it is also required for the simulation environment to be visually similar to the real inspection environment with the quay wall. To do this a model of a quay wall that has been inspected manually is used. This way the simulation environment of the AUV is very similar to the real operating environment.

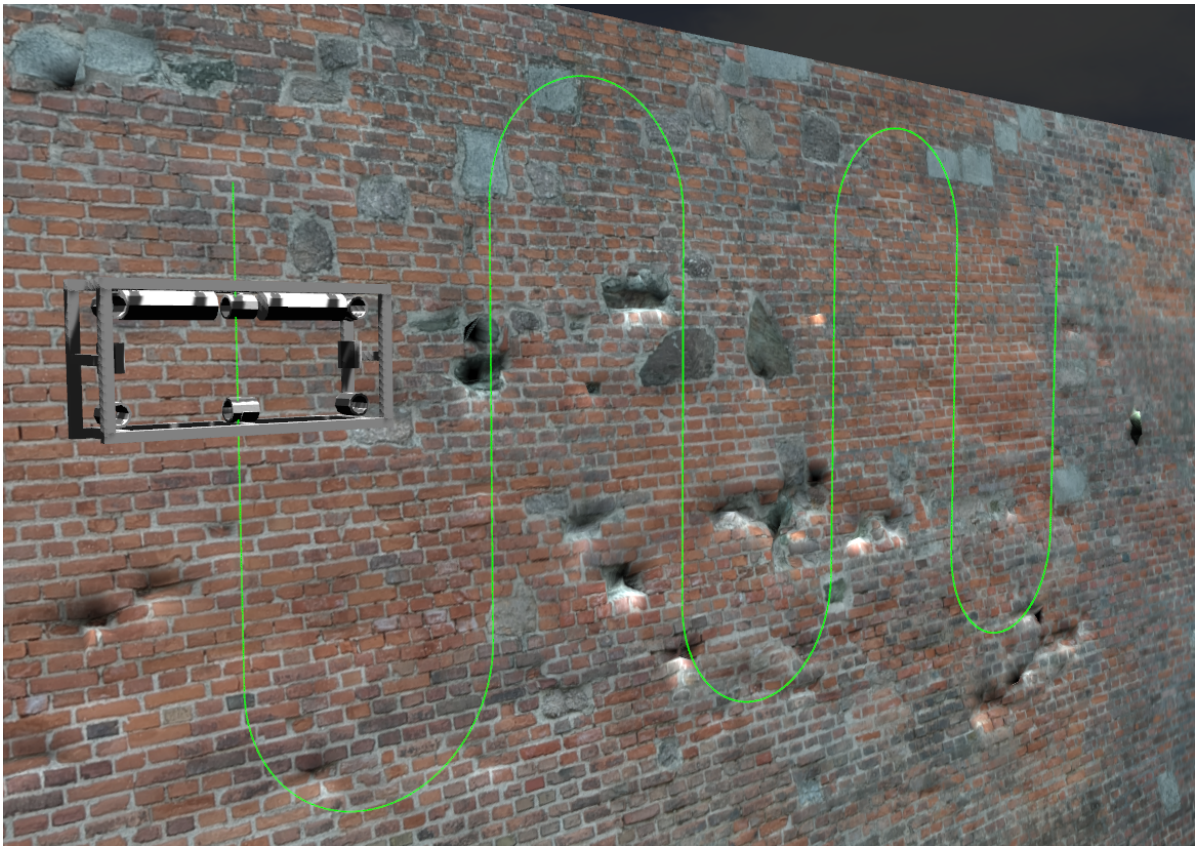


Figure 5.1: A snapshot of the Gazebo simulation environment with the Raybot. The green points depict the reference zig-zag path

5.2. Setup of Simulation Experiments

The goal of the simulation experiments is to assess the performance of the MPC controller compared to the PID controller. To do this a path has been designed, that moves along the quay wall in a zig-zag path, see also Figure 2.1. Both the errors on the path as well as the completion time need to be measured in order to assess them. The errors on the path can be measured as the Mean Squared Error (MSE) of the path, by adding the positional errors (x, y, z) and the rotational errors (ϕ, θ, ψ) together. The completion time is measured as the time needed from the first inputs sent to the thrusters till the Raybot has reached the final point on the path. This will be done both with an experiment using a MPC controller, as well as a PID controller. The method to perform a single simulation experiment is described below.

The tests to be performed in the simulation environment are composed of several steps and have to be carried out the same way every time to ensure the most accurate results. First, the simulation environment has to be spawned. This can be done by launching a world file in Gazebo. Afterwards the Raybot can be launched, as well as the solver node. In order to make the solver faster the `codeoptions.optlevel = 2` has been set, which makes the solver try to optimize for speed. This is done before the solver is computing the robot inputs, but it does take some time, and it has to be done every time the solver settings are changed. For this reason it is recommended to perform this step by running the node once before commencing the experiment with new solver settings. Afterwards the solver already knows its settings optimized for speed and will not need to compute them again. This way the uploading of the solver node is much faster. The Raybot has a very small downward restoring force that cannot be removed unfortunately. Due to this the Raybot will very slowly move downwards. For this reason it is important to launch the Raybot and the solver node at the same time, so that the solver will begin with computing the thruster inputs when the Raybot is still at its initial position. The Raybot will then start following the path that has been set in the solver node. Upon reaching the end of the path the solver will terminate and save all the thruster inputs and states for further analysis.

It should be noted that the simulation environment and solver are different and cannot exactly be launched at the same time, but are dependent on each other. This can create small errors due to clock differences, human errors and integration errors. These errors will be small, but do make that the simulation experiment is not deterministic. Due to this 3 experiments have been run for each setting setup, and their average is used in Chapter 6.

Besides a comparison between a MPC and PID controller simulation experiments have also been carried out while changing the parameters of the MPC solver. To do this one must follow the procedure as described before, and change the parameters as described by Table 6.2. By changing these parameters their influence on the performance can be assessed. This can then be used to tune the parameters in such a way that the best performance is achieved. Also, it can be used when a different use-case is desired, for example when completion time is more important than path tracking performance the parameters can be tuned to gain a lower completion time, at the cost of a lower path tracking performance.

Next to the path following attempts of a zig zag path along a quay wall there has also been an attempt to validate the controller on a downward helical path, while facing inwards all the time. This can demonstrate the rotating behaviour of the Raybot, as it is able to move in all 6 DoF. The use-case for such a type of path would be a large pillar-like structure that needs to be inspected from all sides. Unfortunately there was not enough time in this thesis project to properly implement the following of a helical path in combination with Gazebo. Therefore it was chosen not to showcase this ability, as a correct maneuver was not achieved.

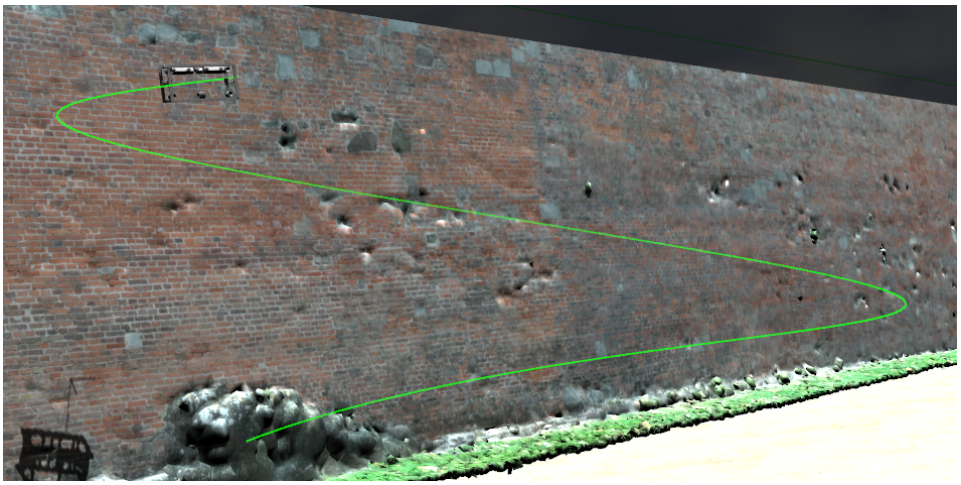


Figure 5.2: A snapshot of the Gazebo simulation environment with the Raybot. The green points depict the reference helical path

6
Results

This chapter presents the results of this thesis. A comparison is presented between the baseline cascaded PID and the MPC controller in Section 6.1, as well as an analysis on the influence of various tuning parameters on the MPC performance are presented in Section 6.2. This will give an answer to the fourth sub-research question as presented in Section 1.3.2: *How does the MPC controller compare to the baseline controller?*

6.1. Performance

During this experiment the MPC and a Cascaded PID were compared, where more information on the cascaded PID controller can be found in Appendix A. Figures 6.1, 6.2, and 6.3 and Table 6.1 show the results of the comparison between the cascaded PID and the MPC controllers on the zigzag path in the simulation environment. Figure 6.1 shows the position on the zy-plane of the trajectory, 6.2 shows the position and velocity states over the trajectory, and Figure 6.3 their corresponding control inputs. It can be seen that the velocities and thruster inputs stay within their bounds. Table 6.1 shows the Mean Squared Error (MSE) on the error between the path and the actual trajectory of the cascaded PID and MPC controller, on both the positions and the orientations. It also shows the completion time to complete one zigzag path. The tuning settings of the MPC controller are the same as simulation 2 of Table 6.2, however do note that the starting positions are different. This causes the difference in results on the MSE and the completion time.

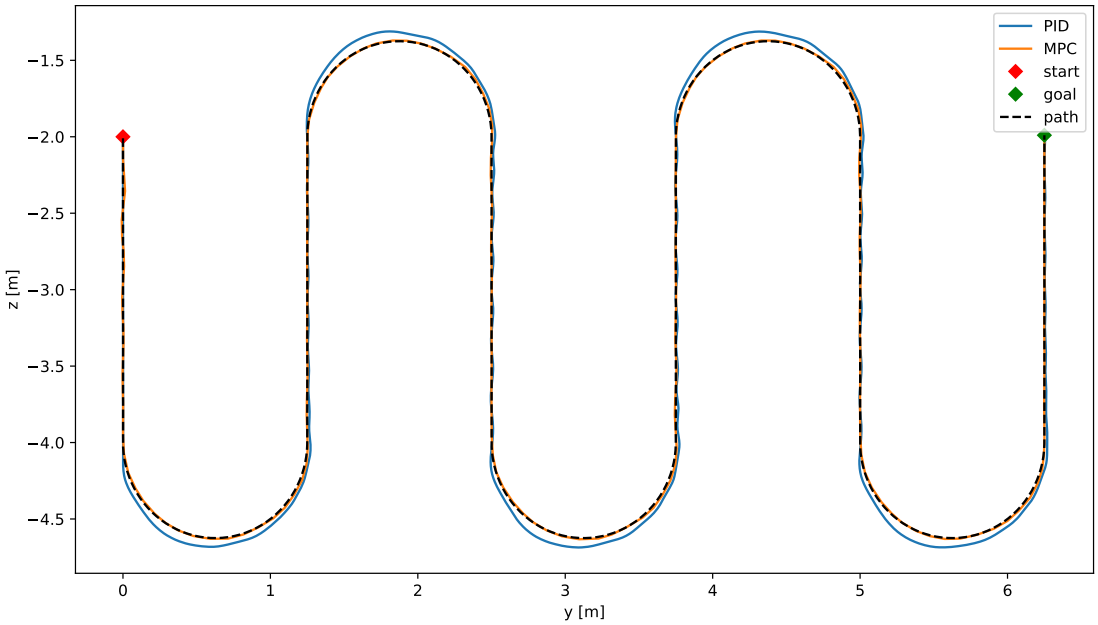


Figure 6.1: The position of the Raybot on the zy-plane of the PID and the MPC controller on the zig-zag path

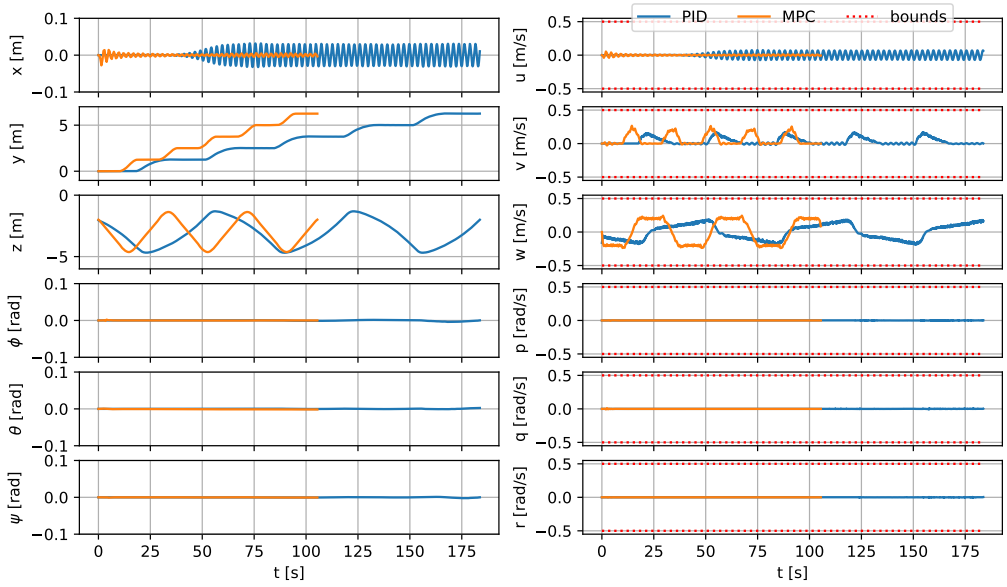


Figure 6.2: The states of the Raybot using the cascaded PID and MPC controller on the zig-zag path

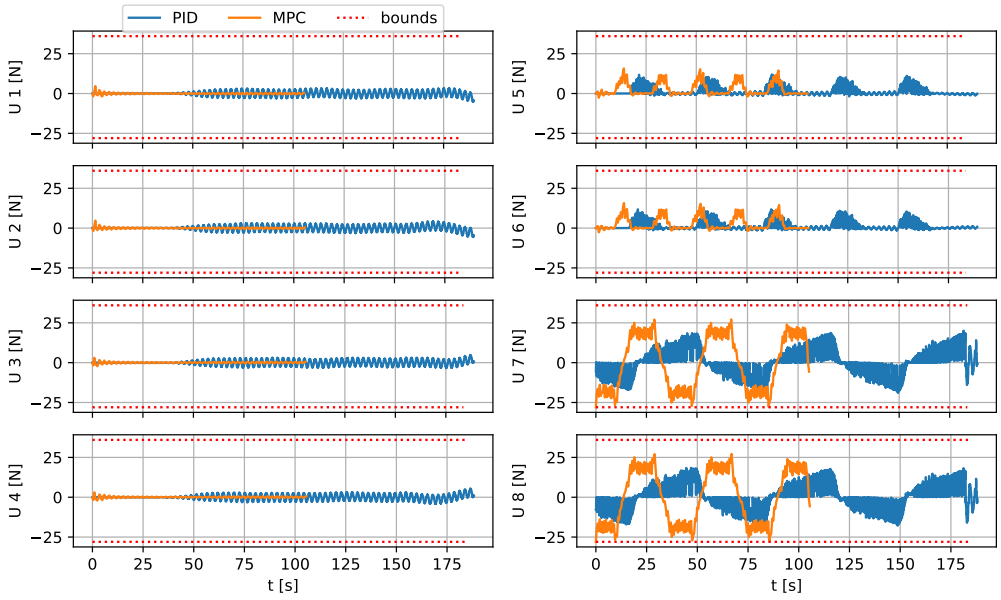


Figure 6.3: The thrusters inputs of the Raybot using the PID and MPC controller on the zig-zag path

	MSE pos [*10^-3]	MSE rot [*10^-6]	Completion time [s]
Cascaded PID	1.208	1.931	183.70
MPC	0.0469	1.19	105.44

Table 6.1: Results of the cascaded PID and MPC controllers

6.2. MPC Parameters

Table 6.2 shows the results of the MPC controller with different tuning settings. Here, N is the prediction (and control) horizon, num is the spacing between the path points, where $num = 100$ is a distance of $0.01m$, and $num = 20$ is a distance of $0.05m$. $Cost\ thrust$ is the cost on the thruster inputs, $Cost\ pos$ is the cost on the position errors, and $Cost\ rot$ is the cost on the rotation errors, see also Equation 4.5 and 4.6.

Sim	N	num	Cost thrust	Cost pos	Cost rot	MSE pos [*10^-3]	MSE rot [*10^-5]	Time [s]
1	15	10	0.06	10000	1000	0.587	0.078	**79.563**
2	***15***	***15***	***0.06***	***10000***	***1000***	***0.315***	***0.119***	***106.72***
3	15	20	0.06	10000	1000	0.226	0.075	142.69
4	15	30	0.06	10000	1000	**0.179**	**0.035**	205.30
5	10	15	0.06	10000	1000	0.226	0.129	106.36
6	20	15	0.06	10000	1000	0.327	0.321	106.29
7	25	15	0.06	10000	1000	0.586	0.280	106.73
8	50	15	0.06	10000	1000	5.487	23.005	106.67
9	20	25	0.06	10000	1000	0.255	0.386	175.38
10	20	20	0.06	10000	1000	0.268	0.171	142.18
11	15	15	0.6	10000	1000	0.343	0.095	129.02
12	15	15	0.006	10000	1000	0.959	47.32	89.19

Table 6.2: Results of the MPC simulation. Results shown are an average over 3 simulations.

Figure 6.4,6.5,6.6 and 6.7 show the results for simulation 2. It starts with an initial position of $y =$

$0.1m, z = -1.9m$, which lies outside of the path. It can be seen that the velocities, thruster inputs, and computation times stay within their bounds.

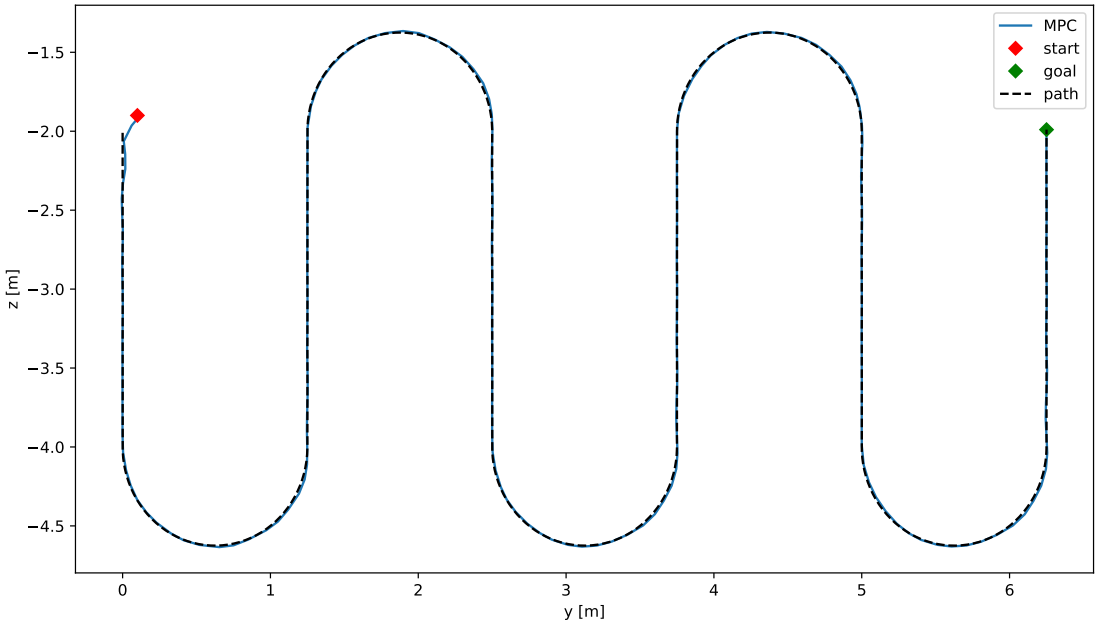


Figure 6.4: The position of the Raybot on the zy-plane of the MPC controller on the zig-zag path with simulation 2

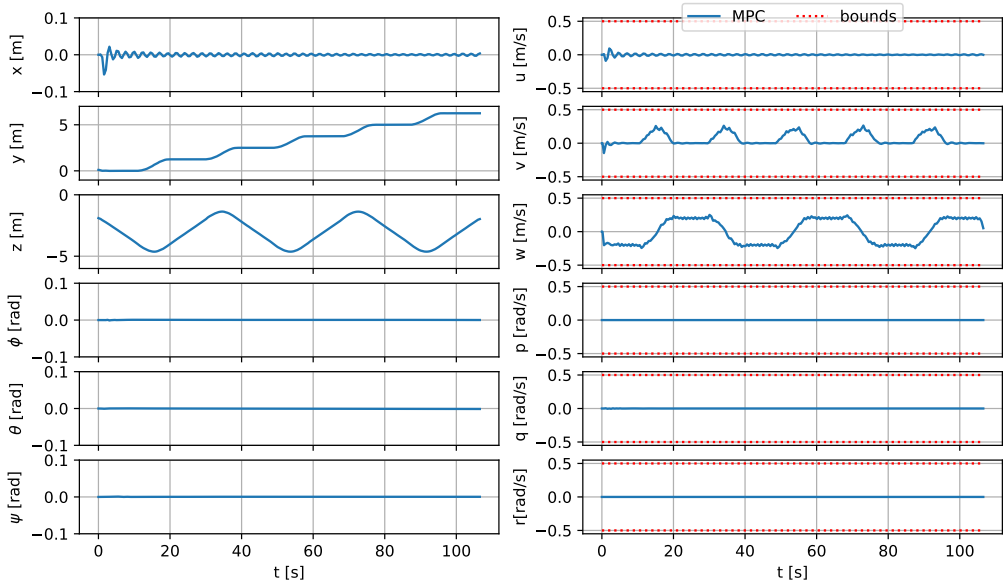


Figure 6.5: The states of the Raybot with the MPC controller on the zig-zag path with simulation 2

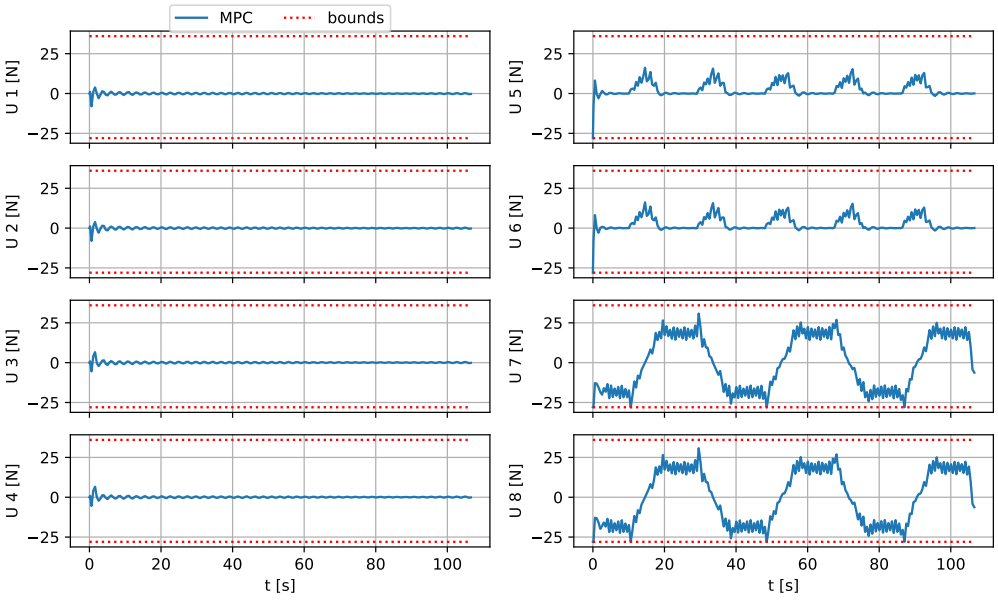


Figure 6.6: The thrusters of the Raybot with the MPC controller on the zig-zag path with simulation 2

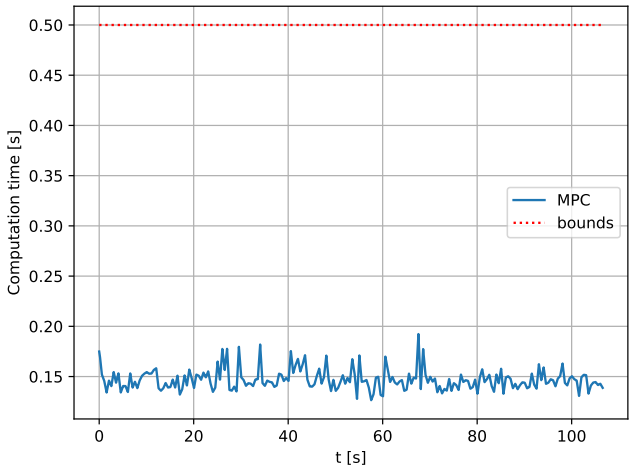


Figure 6.7: The computation time of the solver of the MPC controller on the zig-zag path with simulation 2

7

Discussion

This Chapter will analyze the results found in Chapter 6 in Section 7.1, and present any limitations and shortcomings it might have in Section 7.2. This will give an answer to the research question as presented in Section 1.3.2: *How can Model Predictive Control be used for an Autonomous Underwater Vehicle to improve the performance of quay wall inspection missions?*. It will then present recommendations for future research to look into in Section 7.3.

7.1. Results

This section will explore how MPC can be used on an AUV to improve the performance of quay wall inspection missions. The results discussed here are presented in Chapter 6.

As seen in Figures 6.1, 6.2, and 6.3 and Table 6.1 the MPC controller outperforms the baseline cascaded PID controller. It can be seen that the MPC controller follows the path more closely, which is also proven by the MSE on both the position and orientation. They are both significantly lower compared to their PID counterparts. This improved path tracking ability makes that the Raybot is able to operate closer to the wall, as it needs less bounds to prevent collisions. By operating closer to the wall the Raybot can inspect it in more detail if needed. This can be especially useful if the water conditions are murky, and visual inspection by camera is hard. This leads to an improved quality of the quay wall inspection.

Next to this the completion time is also significantly lower for the MPC controller compared to the PID controller. A lower completion time of the path is desired as this will ultimately lower the completion time of the whole inspection mission. This leads to a lower cost per meter of quay wall inspected, and assuming limited materials and man-hours, to more quay walls being able to be inspected.

In order to select the best tuning settings for the MPC controller several simulations have been run, see table 6.2. By varying the prediction horizon N it can be seen that the MSE on the position and orientation vary. A low prediction horizon leads to larger errors, as less future reference information is taken into account. This can lead to a decreased anticipation on changing reference points, for example in the half-circle corners of the path, or the starting point with a small offset. Large prediction horizons also show an increased MSE. This is caused by too much future reference information being taken into account. This will lead to longer solving times, which causes a larger time delay between the time the current pose of the Raybot is attained and the thruster inputs have been executed.

By varying the parameters of the cost function the completion time can be changed. By setting a higher cost on the thruster inputs the optimization algorithm is less likely to compute high thruster inputs, as these will cost more. In turn, this leads to a decreased velocity of the Raybot, which leads to a higher completion time. The reverse is true for a lower cost on the thruster inputs. As the Raybot aims to decrease the error on the next N path points the Raybot will now have an increased velocity if the thruster cost is lower, but only if the next path point would otherwise not have been reached at

each timestep. This leads to a more aggressive behaviour in minimizing the position and orientation error. This paradoxically leads to a higher MSE on those, as the Raybot now operates on the bounds of the thruster constraints, leading to a decrease in path tracking performance. This does explain why varying the num parameter can lower the completion time more. By varying this parameter the distance between the path points is changed. In essence, it is the number of points per meter of path, so a higher num means a lower distance between the path points. Thus, by decreasing num the Raybot will aim to cover more distance per timestep. This decreases the completion time of the path. The level of regularization of the hessian approximation hes was also decreased to check if this would result in lower MSE, but this was not the case.

7.2. Limitation and Shortcomings

The model predictive control algorithm and simulation environment are both reliant on the model of the AUV in order to control or simulate the real AUV. This is why it is important for the model, and its corresponding parameter estimation, to be accurate. The parameter estimation on the damping terms is flawed however. It uses a scaled model of the Raybot, but it was not possible to get all the dimensions correct, especially because mounting brackets were needed. Furthermore it was not possible to test the damping terms on the pitch movement, as the water tanks was not deep enough for this test. This parameter has thus been estimated, based on similar robots, and the other damping terms. Furthermore the test setup had some friction, especially in the rotational test setup. These causes may add up to large errors in the damping terms, which may not represent the damping physics of the real Raybot. It should be noted however that they have been compared to similar smaller and larger AUV's, and that the damping terms do fit in somewhere between those.

Besides model-mismatches another factor is the rotation matrix issue as explained in Section 5. This issue causes limitations in the movement of the Raybot besides its quay wall mission path. It does not present any issues under normal operation, but when the Raybot is needed for other inspections, where a rotational movement is needed, this may present issues.

Furthermore, the SQP algorithm of the ForcesPro solver has been set to solve 4 Quadratic Programming iterations at each timestep. This was to ensure a low computation time. This does mean however that it does not stop until convergence of the KKT conditions has been met so optimality can be guaranteed. This implies that it cannot be guaranteed that an optimal control input has been applied to the AUV. As the SQP algorithm also does not guarantee feasible iterations this can result in non-optimal or even infeasible control inputs being applied to the AUV. However, during all the simulations this has not seemed to be the case.

7.3. Future work

Future work of this thesis should focus on several issues, partly as presented in Section 7.2. The model-mismatches caused by inaccurate damping terms can be solved by performing the same experiment on the true Raybot. This will solve any issues with the scale-model, and when using a large enough tank allows for full damping term estimation. Also, the friction in the test setup can be linearly superimposed and thus removed from the estimation. This will lead to a more accurate estimation.

Future work should also look into the issues with the rotation around the yaw-axis. Unfortunately there was not enough time to implement and tune the solver in such a way that a yaw-rotation could be performed reliably. In use-case outside of quay wall inspection of the Raybot this yaw-rotation can be important, take for example the helical path around a pillar that was attempted. This issue can be solved by more testing and tuning with the solver, and possibly also with an underwater robot that is not overactuated, as this might lead to less possible solutions the solver will try to explore.

Previous works have shown the advantages of a Lyapunov-based Model Predictive Controller [21] [22]. This controller proves closed-loop stability and recursive feasibility, as well as model parameter errors of up to 30%. These are very desirable traits for the Raybot and should thus be looked into in

future work.

It is also recommended to perform real tests on the Raybot instead of only in simulation. This way the performance of the controller and the model estimation can truly be validated. For this a large water tank and some form of localization algorithm is needed. When budgetary issues are present it is recommended to look into a localization algorithm using fiducial markers, as also seen in [5], [4].

At the moment the simulator and the solver are using the same model, although they are working asynchronously. However, in real-life settings there will be disturbances present on the system. Take for example water currents that might have not been accounted for, or perhaps disturbances on the localization of the AUV that might alter the state. Future research should look into the behaviour of the controller when disturbances are present to research if similar or a least satisfactory results can be achieved. Else, the use-case will be limited to environments with very little disturbances, which might cause that not all quays are able to be inspected.

8

Conclusion

Quay walls are vital structures in water-based infrastructure. For this reason they need be inspected regularly in order to gain knowledge of the state of the quay wall, and prevent any future damage or collapse. To perform these missions an Autonomous Underwater Vehicle is designed and presented in a use-case named the Raybot. This Raybot consists of a frame and thrusters, and is able to carry a set of sensors. This way, it can inspect the quay wall by following a path. In order to follow this path a Model Predictive Control approach is proposed. For this MPC controller the Fossen model is used, where the theory and parameter estimation are explained. This model is also used in the simulation environment based on the Plankton package for a Gazebo simulation environment. The performance of the MPC controller is compared to a baseline cascaded PID controller. The MPC controller outperforms the cascaded PID controller both on completion time and the tracking error of the path. The improved tracking performance leads to the ability to inspect more accurately, which leads to higher quality inspections. The lower completion time leads to the ability to inspect more quay walls. For future work it is recommended to test the MPC controller on a real robot in a tank, as this can truly prove the performance of the controller. Also, it is recommended to further look into path tracking performance on rotations, as this can make the Raybot suited for more versatile inspection missions.

References

[1] Aurélien Babarit and Gérard Delhommeau. “Theoretical and numerical aspects of the open source BEM solver NEMOH”. In: *11th European wave and tidal energy conference (EWTEC2015)*. 2015.

[2] Viktor Berg. “Development and Commissioning of a DP system for ROV SF 30k”. MA thesis. Institutt for marin teknikk, 2012.

[3] Bluerobotics. *T200 Thruster*. URL: https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/. (accessed: 05.12.2022).

[4] João Britto et al. “Model identification of an unmanned underwater vehicle via an adaptive technique and artificial fiducial markers”. In: *OCEANS 2015-MTS/IEEE Washington*. IEEE. 2015, pp. 1–6.

[5] Juan Chen, Caiming Sun, and Aidong Zhang. “Autonomous Navigation for Adaptive Unmanned Underwater Vehicles Using Fiducial Markers”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9298–9304.

[6] WEBREDACTIE COMMUNICATION. *Grimburgwal provides lessons for quay wall renovations Amsterdam*. URL: https://www.tudelft.nl/en/2021/tu-delft/grimburgwal-provides-lessons-for-quay-wall-renovations-amsterdam. (accessed: 11.05.2022).

[7] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org.

[8] Alexander Domahidi and Juan Jerez. *FORCES Professional*. Embotech AG, url=https://embotech.com/FORCES-Pro. 2014–2019.

[9] Tijmen van Enckevort. *Tijmen Graduation*. URL: https://github.com/dobots/tijmen_graduation. (accessed: 01.11.2022).

[10] Odd Faltinsen. *Sea loads on ships and offshore structures*. Vol. 1. Cambridge university press, 1993.

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

[12] Liquid-AI. *Plankton*. URL: https://github.com/Liquid-ai/Plankton. (accessed: 07.12.2022).

[13] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[14] Musa Morena Marcusso Manhães et al. “UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation”. In: *OCEANS 2016 MTS/IEEE Monterey*. IEEE, Sept. 2016. DOI: 10.1109/oceans.2016.7761080. URL: https://doi.org/10.1109%2Foceans.2016.7761080.

[15] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. “Exploring the performance of ROS2”. In: *Proceedings of the 13th ACM SIGBED International Conference on Embedded Software (EMSOFT)*. 2016, pp. 1–10.

[16] Wasif Naeem. “Model predictive control of an autonomous underwater vehicle”. In: *Proceedings of UKACC 2002 Postgraduate Symposium, Sheffield, UK, September*. Citeseer. 2002, pp. 19–23.

[17] NOS. *Duiker verdronken tijdens zijn werk in Driel*. URL: https://nos.nl/artikel/2191985-duiker-verdronken-tijdens-zijn-werk-in-driel. (accessed: 24.10.2022).

[18] Simon Pedersen et al. “Stabilization of a rov in three-dimensional space using an underwater acoustic positioning system”. In: *IFAC-PapersOnLine* 52.17 (2019), pp. 117–122.

[19] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[20] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI, 2017.

[21] Chao Shen, Yang Shi, and Brad Buckham. “Lyapunov-based model predictive control for dynamic positioning of autonomous underwater vehicles”. In: *2017 IEEE International Conference on Unmanned Systems (ICUS)*. IEEE. 2017, pp. 588–593.

[22] Chao Shen, Yang Shi, and Brad Buckham. “Trajectory tracking control of an autonomous underwater vehicle using Lyapunov-based model predictive control”. In: *IEEE Transactions on Industrial Electronics* 65.7 (2017), pp. 5796–5805.

[23] Dassault Systèmes SolidWorks. “SolidWorks®”. In: *Version Solidworks 2020* 1 (2020).

[24] Dirk Thomas, William Woodall, and Esteve Fernandez. “Next-generation ROS: Building on DDS”. In: *ROSCon Chicago 2014*. Mountain View, CA: Open Robotics, Sept. 2014. DOI: 10.36288/ROSCon2014-900183. URL: https://vimeo.com/106992622.

[25] Chu-Jou Wu. “6-dof modelling and control of a remotely operated vehicle”. PhD thesis. Flinders University, College of Science and Engineering., 2018.

[26] A. Zanelli et al. “FORCES NLP: an efficient implementation of interior-point... methods for multistage nonlinear nonconvex programs”. In: *International Journal of Control* (2017), pp. 1–17.

A

Cascaded PID

A.1. Cascaded PID control

The baseline control algorithm is a cascaded PID controller. This is two PID controllers placed in series over the plant, see also Figure A.1. The first PID controller acts on the pose error of the AUV compared to the path. This position error is split into a position error (x, y, z) and an orientation error (ϕ, θ, ψ). These errors are fed through the P, I and D gains of the position controllers and combined together to form a 'Twist' command. This is the desired velocity composed of the linear and angular velocities of the AUV. Again, this error is split up into the linear (u, v, w) and angular velocities $(p, q, r$. These errors are fed through the P,I,D gains of the velocity controllers and this results in the force that the AUV needs to apply in order to follow the reference path.

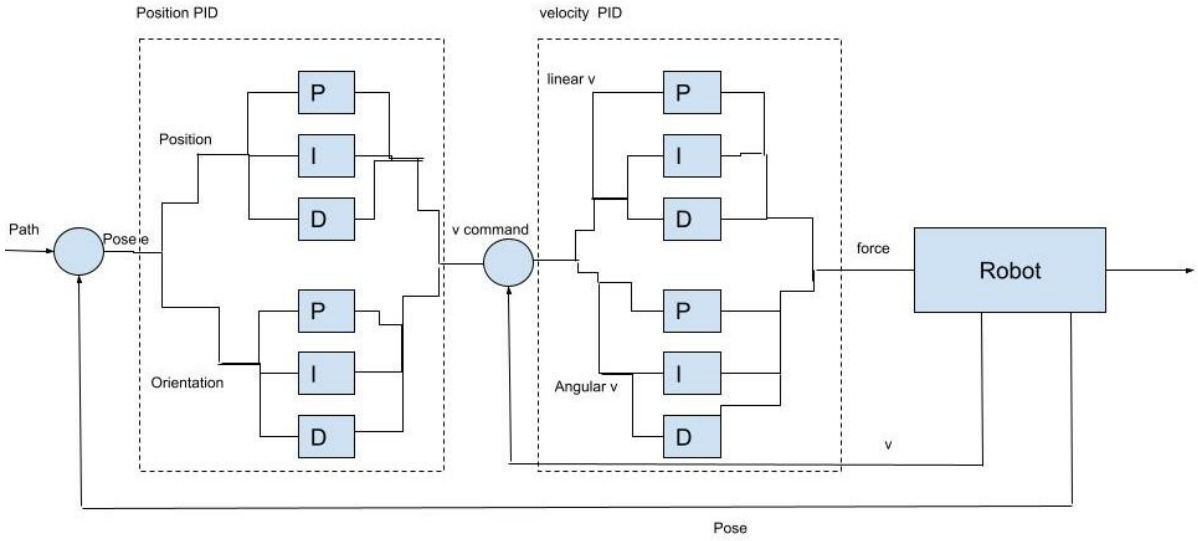


Figure A.1: Cascaded PID controller structure

A.1.1. Formulation

Table A.1 shows the PID gains for the cascaded controller for the raybot AUV.

PID Gains	Position	Velocity
linear P	3	8
Linear I	0.3	2
Linear D	0.5	0.2
Angular P	3	5
Angular I	0.3	2
Angular D	0.5	0.2

Table A.1: The gains of the cascaded PID controller

A.1.2. Implementation

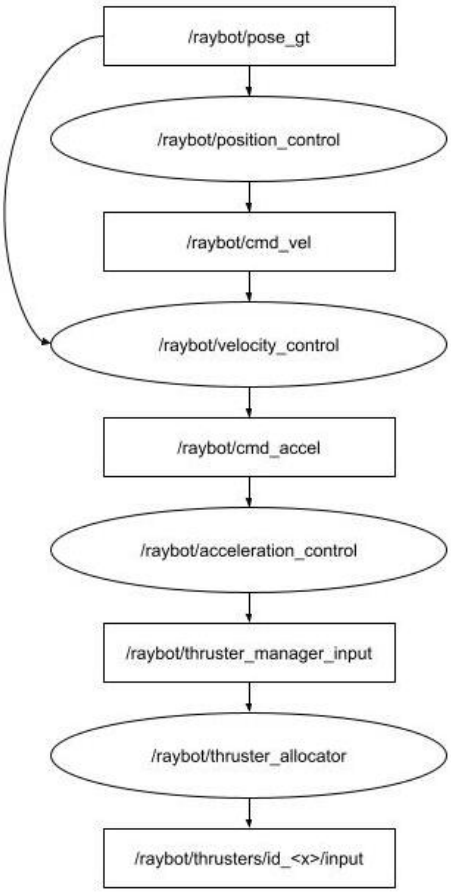


Figure A.2: Cascaded PID controller structure in ROS

Figure A.2 shows the pipeline in the ROS environment for the cascaded PID controller. It starts with the position of the AUV: the `/raybot/pose_gt` message. The `/raybot/pose_gt` message consists of the position and velocity state of the AUV. This position is then used by the first PID controller, the `position_control` node. The path with the reference points is defined in this `position_control` node. The error on the position is calculated and used to compute a reference velocity message, the `/raybot/cmd_vel` message. The `/raybot/velocity_control` node then uses the `/raybot/cmd_vel` and `/raybot/pose_gt` messages to compute the error on the velocity of the AUV, and the control actions necessary to get to the reference velocity. This `/raybot/cmd_accel` command is then published and picked up by the `/raybot/acceleration_control` node. This node only multiplies desired `/raybot/cmd_accel` command with the mass matrix of the AUV, so that a force vector can be computed. This force vector `/raybot/thruster_manager_input` is published and subscribed to by the `/raybot/thruster_allocator` node. Because the AUV can be over- or underactuated the force vector

cannot be directly applied to the thrusters. For this reason the `/raybot/thruster_allocator` computes the force that is needed per thruster. This force is published on the `/raybot/thrusters/id_<x>/input` topic, where `<x>` is the id-number of the thruster. In the case of the raybot $x = [0, ..., 7]$.

B

Validation Parameter Estimation Added Mass

The method to estimate the addes mass components of the Raybot has been validated on a Bluerov Heavy configuration [25]. A simplified model of the Bluerov Heavy was made in Solidworks, and the added mass matrix was estimated in the same way as for the Raybot. This resulted in the following addes mass matrix:

$$\mathbf{M}_A = \begin{bmatrix} 8.262 & 0 & 0 & 0 & 0 & 0 \\ 0 & 19.43 & 0 & 0 & 0 & 0 \\ 0 & 0 & 19.97 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.23 \end{bmatrix} \tag{B.1}$$

This is slightly different than the added mass matrix that has been reported in [25]

$$\mathbf{M}_A = \begin{bmatrix} 5.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 12.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14.57 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.12 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.12 \end{bmatrix} \tag{B.2}$$

This difference can be explained by the fact that the simplified model of the Bluerov Heavy does not have any rounded edges and contains only sharp edges and corners. This makes it less hydrodynamically efficient, and this can explain why the parameters by the Nemoh estimations are slightly higher. Still, it can be considered as a suitable method to estimate the parameters for the Raybot as the values do make sense.