ACIT5900

MASTERS THESIS

in

Applied Computer and Information Technology (ACIT)

May 2023

Robotics & Control

Target Tracking Control for an Unmanned Surface Vessel: Optimal Control vs Reinforcement Learning

Aksel Johan Frafjord

Department of Mechanical, Electrical & Chemical Engineering

Faculty of Technology, Art & Design

OSLOMET

Preface

This master thesis (30 credits) was written during the spring of 2023 as the finishing contribution to the Master of Science in Applied Computer and Information Technology (ACIT) at OsloMet. This paper presents the methods and implementation of two control strategies for tracking submerged targets using nonlinear model predictive control (NMPC) and reinforcement learning (RL).

This thesis was inspired by recent advances in RL that have shown feasibility in learning to play arcade games and other toy control problems. Some work on using RL for real control problems has been undertaken. However, further development is needed for RL to be a reliable asset in controller development. In contrast, classical optimal control techniques, such as NMPC has a solid reputation and have proven a reliable controller within academia and the industry for decades. This thesis seeks to contribute to reducing the gap between the RL domain and the classical control theory domain, and by doing so, contribute to the progress of RL in the field of control.

The affiliated chapters will present the theory needed to understand the content in each chapter respectively. However, familiarity with control theory, artificial intelligence and nonlinear modelling of marine crafts is beneficial.

Acknowledgement

I want to express my sincere gratitude to my advisor, Ivar Bjørgo Saksvik, for his invaluable guidance and support throughout the implementation and writing of this thesis. Your expertise and insightful knowledge have been particularly crucial in my understanding of marine craft modelling. Thank you for your dedication and encouragement throughout this project.

I also want to thank Jan-Philip Radicke, Pierre Odin Boniface and Alexander Halseth for creating a meaningful community during our master's. Your witticisms, support and constructive criticism are very much appreciated.

Dear Julie. Thank you for standing by me through stressful times involving frustration and long working hours. Your support and unbound optimism have enabled me to keep my head straight throughout this semester's endeavours.

Aksel Johan Frafjord

Oslo, May 15th 2023

Abstract

This thesis studies the development and performance of Nonlinear Model Predictive Control (NMPC) and Reinforcement Learning (RL) for a target-tracking problem. The methodology involves developing the NMPC and RL approach and comparing their performance through simulated experiments. In the simulations, the controllers steer the Otter unmanned surface vessel (USV) to track a virtual target.

The resulting NMPC controller performed with a stable error of approximately 0.7m with a refresh rate of 1.6 Hz. Whereas the best-performing RL Agent demonstrated a twofold performance. In the first part, the Agents managed an error between 0 and approximately 2m. However, when surpassing the experienced observation space from the training session, the Agent generated unfeasible controller signals, resulting in the Otter circling the target while tracking it. The Agent achieved a 1kHz refresh rate.

In conclusion, the NMPC may need to be faster for practical implementations, and RL Agents require further development to be reliable. Therefore, for future work, it is suggested to use NMPC as an expert and apply imitation learning when training the Agents to achieve the best of both methods.

Keywords: Target-tracking, nonlinear manoeuvring modelling, USV, nonlinear model predictive controller, NMPC, MPC, Reinforcement Learning, RL

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Climate change and direct human impact have increasingly affected the Earth's oceans and marine life. To equalise the consequence and make the marine environment sustainable, we must make rational choices when determining mitigation to counteract the human impact on the ocean. Hence, the collection of data about the ocean's conditions is crucial. However, this task is often expensive and requires specialised equipment and operations conducted in high-risk environments, such as offshore maritime areas with a risk of waterline exposure.

The project Ocean-Air synoptic operations using coordinated autonomous SYStems and micro underwater gliders (OASYS) were initiated to create an affordable data collection system [33]. The OASYS project involves the use of unmanned surface vehicles (USVs), autonomous underwater gliders (AUGs), and unmanned aerial vehicles (UAVs) to complete various missions and collect data about a sea area.

One of the notable contributions from the OASYS project is the OASYS glider, as shown in figure 1.1. This vehicle uses a unique variable buoyancy system (VBS) to move up and down in the water column. The VBS consists of an external bladder and an internal oil reservoir that can be adjusted using a miniaturized pump to displace the net volume of the vehicle and cause it to sink or rise in the water column. The glider's speed typically ranges from 0.1-0.3 m/s and depends on the vehicle's pitching angle. Instead of relying on conventional control surfaces like rudders and dive planes, the glider uses internal moving mass actuators to control its pitch and heading angles [34] [12].

Figure 1.1: OASYS glider

To ensure that the vehicles can operate autonomously and make decentralised decisions, it is crucial to have seamless communication between all units involved in the mission. Communication over the sea is well established and has the sturdy infrastructure the units above water can use to provide cohesive communication. On the other hand, the availability of low-cost and satisfactory communication infrastructure equipment is not well-established for underwater communications. OASYS seeks to mitigate this challenge by developing communications between the units using optical relays to transmit data between vehicles under the water surface. Unfortunately, there seems to be no way to communicate effectively through water and air. However, there is a potential solution by using a topside USV with access to both an underwater optical transceiver and aerial radio communications, effectively creating a communication relay between the two media.

One challenge with optical communications is that the line of sight between the sender and receiver must be within the limits of the system's configurations. To communicate, the line of sight between the topside vessel receiver and the glider transmitter must be constant to transmit data successfully. This thesis aims to contribute to the communication system by studying the motion controls of the topside USV as it tracks the glider and stays within the threshold of the transceiver. We assume that the Otter is equipped with a short baseline acoustic positioning system (SBL) so that the target position is always known, as shown in figure 1.2.

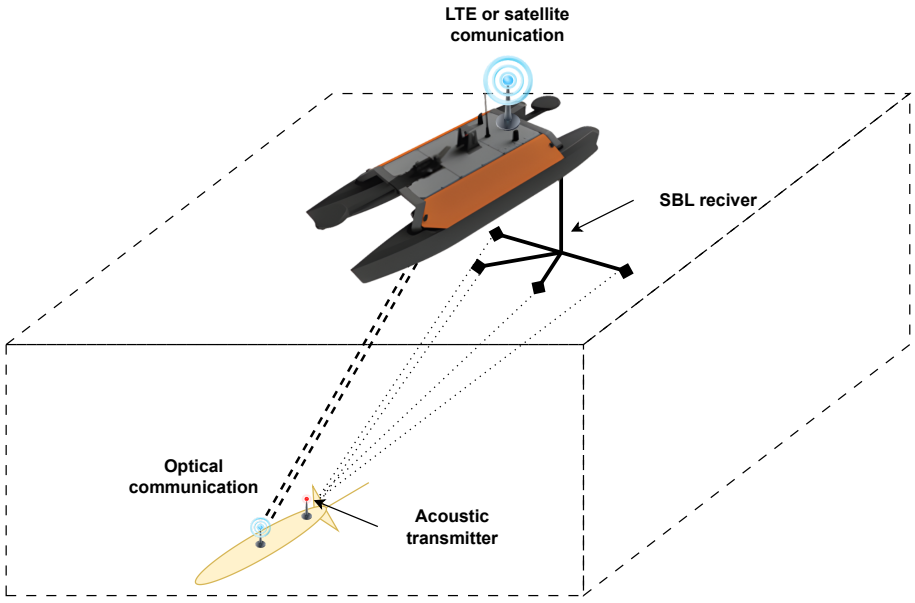


Figure 1.2: USV relay communications from AUG

1.1 Thesis Scope

For the controller to be a viable asset in the data-transfer operations, we assume it must keep the Otter USV within a radius of less than 10 meters of a moving target. As the controller only will have access to the target position, we define this as a target tracking problem, where the submerged vehicle's future motions are unknown. The topside vessel only receives information about the instant position of

the underwater glider through acoustic localisation.

This thesis will study two advanced controller techniques in a moving-target-tracking use case as shown in figure 1.3. Approach (i) is a non-linear Model predictive controller (NMPC). Approach (ii) is to use an Artificial Neural Network (NN), i.e. a policy trained using deep reinforcement learning (RL). The study will investigate the development, performance and practical feasibility of using the controller in the aforementioned optical communication relay objective.

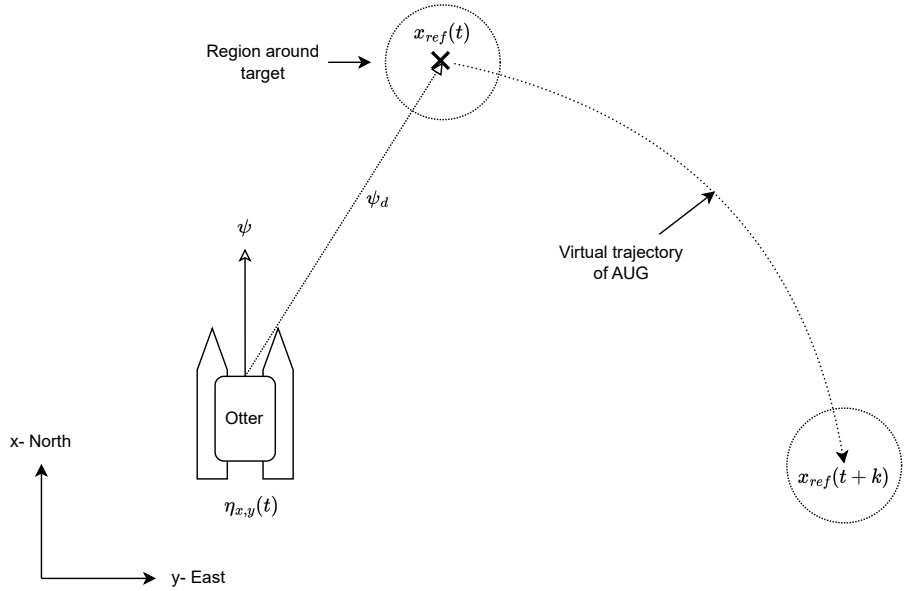


Figure 1.3: Controller objective

In figure 1.3, the Otter USV and the target are illustrated. The target will be a virtual target simulating the movement of the OASYS AUG, and the goal is for the Otter USV to stay within the region surrounding the moving target.

1.2 Thesis Layout

Chapter 2 provides a background of guidance, navigation and controls before a literature review is given. The literature review presents relevant previous work related to advanced controller methods involving NMPC and RL approaches. Chapter 3 describes the theory and the implementation of the dynamic model of the Otter USV that is used in the two approaches. Chapter 4 Model Predictive Controller and chapter 5 Reinforcement Learning approach, describes the theory, implementation and results of each approach, respectively. Then, in 6 there is a common discussion, presenting the differences in the approach followed by future work. Lastly, a conclusion is presented.

Chapter 2

Background

This chapter is twofold. First, some required knowledge is presented regarding the fundamental aspects of motion control of marine vessels. Second, there is a literature review of previous work regarding motion control using MPC and RL.

2.1 Guidance, Navigation and Control

This section will define Guidance, navigation and control and their respective tasks in controlling to achieve a control objective. In marine vessel motion control, it is common to use a cascade system architecture to achieve a motion objective. The guidance system provides the details of path-planning, situational awareness and other mission-planning objectives. Additionally, it provides a human interface for personnel interaction. The output from the guidance system defines the mission objective. The control system is responsible for translating the mission objective into thruster commands for the vessel to conduct the control objective. Lastly, the navigation system is responsible for determining the vessel's position [13].

2.1.1 Guidance Systems

One may define guidance systems as "*The process of guiding the path of an objective towards a given point, which in general may be moving*" [32]. Hence, guidance systems are used to compute the desired position, velocity and attitude of the craft. Then, as explained previously, the information is used in the motion control system to allocate controller signals for the crafts actuators to solve the control objective. The guidance system takes data from various sources as sensor data, weather data or user inputs. The data is then processed to compute the desired controller objectives represented as desired velocity and position. The guidance system may use various methods, from ad-hock techniques to advanced dynamic optimisation, depending on the complexity of the control objective and the amount considered optimisation parameters [13].

2.1.2 Control Systems

Whilst the guidance system's responsibility is to provide a desired attitude and velocity, the control system is responsible for providing the control signals to the crafts actuators. The actuators again affect the vessel's effector, i.e. the rudder or propellers, that generates control forces. The effectors on marine craft are mostly propeller devices for industrial applications, but they can also be water jets, fans, sails or buoyancy propulsion. The control system has two tasks.
(i) Based on some control law, the motion control system uses some method to compute the desired forces to conduct the controller objective. Such state-of-the-art methods are based on the principles of modelling a craft as a spring-damped system. Then, adding a stability analysis, we have the fundamental parts that lay the bedrock for the different algorithms. The simplest is the autopilot using *Successive Loop Closure*, which uses simple linearised transfer functions and successively computes system parts to make it stable. The *PID pole-placing algorithms* use Lyapunov stability analyses to optimise the craft's kinetic and potential energy to achieve optimal pole placement. PID controllers are widespread in industrial applications such as dynamic positioning, autopilots and path-following [13].
(ii) The second responsibility of the control system is, through controller allocation, to decode the desired forces to controller signals for the actuators [13].

2.1.3 Navigation Systems

The function of navigation system is responsible for determining the craft's position, velocity and attitude through a state estimator. Most modern crafts use global navigational satellite systems (GNSS) that determine the global position at sea. However, more than the GNSS is required to estimate the full state representation of the craft. Following Fossen, there are two domains within the navigation system. First, model-based Navigation Systems (MBNS), use a craft model supplemented with various techniques, such as a Kalman filter, as the state estimator. In contrast, Inertial Navigation Systems (INS) use accelerometers and attitude sensors (IMU) to determine the state of the craft without the need for a numerical model of the craft [13].

2.2 Previous Work

This section presents a short literature review on existing control methodologies in marine vehicles. The details about motion control are mainly discovered in the "Handbook of Marine Craft Hydrodynamics and Motion Control" [13], and references therein. The review unfolds in two parts: **(i)** a survey on non-linear model predictive control (NMPC), and **(ii)** deep learning approaches,i.e., reinforcement learning (RL). Since this thesis considers both the implementation and the performance of the two approaches, literature that provides detailed insights into well-documented implementations is prioritised. Furthermore, there is rapid development within the domain of RL. Therefore, the literature search has focused on articles and theses published after 2018 within the RL- domain. Besides the mentioned

approaches above, the documentation for the Casadi framework has been used to gain knowledge and practical insights into the domain of optimisation and its regard to MPC. To limit the scope, the following keywords have been used: Model Predictive Control, MPC, efficiency, Target tracking, Autonomous Docking, Artificial Intelligence, and Path following, MPC efficiency. Non-linear MPC, Linear MPC.

2.2.1 Model Predictive Controllers

We will now investigate the Model predictive controller (MPC) and previous works using MPC to solve control objectives. MPC is an advanced control technique that control simple, single-input/output, multi-input, and multi-output problems. The MPC utilises a mathematical model of the system as a constraint in an optimisation problem and predicts the best controller trajectory within the prediction horizon for the controller. An MPC using a linear model is abbreviated as LMPC, whereas a non-linear MPC is abbreviated to NMPC. An in-depth explanation of the MPC and its components will be presented in chapter 4. During controller development, using the physical or real system to observe the response is not practical nor safe. Therefore, we use a model to simulate the controller development and tuning response. However, during operations, the controllers use the model of the system to solve an optimisation problem to gain the optimal controller gain for the actuators in the system [30].

MPC has shown merit in control problems such as trajectory tracking, path following, and docking/dynamic positioning. In [18], the feasibility of using an NMPC in autonomous docking is essayed. The study involves three strategies to achieve the autonomous docking task, where the goal was to dock the under-actuated *Piraya* vessel autonomously. The control objective was formulated as a numerical expression within the cost function in the optimisation problem. In the most advanced scenario, obstacles were defined as constraints in the optimisation problem, enabling the controller to evade identified obstacles simultaneously while solving the controller objective. Thus, the controller embeds the entire problem, including obstacle avoidance and objectives, into one controller. The strategy relevant to this paper is where a single point is placed beside the doc. Additionally, the vessel's orientation was constrained with a yaw angle, restraining the docked vessel's orientation to be parallel to the dock while oriented in a predefined direction in the local frame. All controller approaches were successful and performed accurately in simulations and physical tests. The test showed that the MPC used between 0.1 and 0.9 seconds to compute controller signals.

Other papers also describe scenarios where the NMPC handles obstacle avoidance with the controller objective. As in [21], which develops an NMPC for autonomous docking that embeds an operating region for the vessel. The region is defined as a polyhedron shape in the plane. It surrounds the waters in the harbour so that the vessel can operate without intersecting with obstacles or land during the docking manoeuvres.

The NMPC has also shown merits in open-water path following, where the vessel follows a defined path [38], [18]. The path is then defined as a series of targets.

When the vessel has reached the target, the new target is algorithmically fed to the controller continuously as the vessel progresses. Target tracking is often seen as a sub-problem of the path-following problem.

As mentioned previously, an MPC, as the name implies, relies on a model to predict a controller gain. The model used is often a linear representation between the model inputs and outputs. However, for most systems, the relationships are non-linear. Therefore, they are simplified to a linear system that can be solved with a linear cost function with reduced computational effort than non-linear systems. In [38], they created a linear (LMPC) and non-nonlinear MPC controller (NMPC) on the non-linear optimisation problem of trajectory tracking. The non-linear problem was solved by implementing a linearisation around the state- of the system in the first time step in the prediction horizon. The linear model was then optimised using linear methods and giving the optimal output for the next timestep based on the linearised model. The other part they tested was the non-linear method, which used a non-linear optimisation solver to solve the non-linear problem. In a comparable test scenario, where the measured variable was the mean distance from the trajectory and the vessel's position, the NMPC had a mean of 0.0(m), and the LMPC had a mean of 2.3 (m). However, the NMPC used in comparable time 828 (s) whereas the LMPC used 29 (s) [38].

2.2.2 Deep Reinforcement Learning

The RL-based controller approach uses deep machine learning (ML) to derive a model that computes controller outputs. According to the Oxford Dictionary, Machine Learning (ML) is a "(..)system that becomes smarter as it encounters additional data".

In [22], the concept of Deep Reinforcement Learning is utilised to derive a path-following controller. The utilised algorithm was Deep Deterministic Policy Gradients (DDPG) because it enables the model to learn the value of both a state (provided by an actor-only architecture) and an action (provided by a critic-only architecture). Thus leading to potential benefits of learning the value of an action in combination with the value of a state.

Another paper has used the PPO algorithm to train marine AI controllers for target tracking, path following and auto-docking [37], [8]. Another RL algorithm that has shown promising results is the Twin Delayed deterministic policy (TD3). In the paper [25], they evaluated the performance between proximal policy optimisation (PPO), TD3 and soft-actor-critic (SAC) algorithms where the TD3 algorithm gained better rewards than the PPO or the SAC.

When training an RL model, the model is only bound by the implemented restrictions of the dynamic model. Models are always prone to errors from reality, and this is called the sim-to-reality gap. The models trained on a dynamic model that tries to replicate a system's realistic dynamics tend to create unrealistic manoeuvres that might not be possible in real-world scenarios. In the case of marine craft controllers, these mistakes might be rapid changes in thruster commands, leading to inappropriate controller actions resulting in attrition on the mechanics of the ef-

fectors on the craft. In [5], aggressive changes in controller signals were experienced when the feasibility of two machine learning algorithms in an auto-docking scenario was studied. The PPO algorithm exploited a possible deficiency in the model, in its greed for reward during training, resulting in unfavourable thruster commands for the craft. However, the PPO was chosen for further implementation by the reasoning that the learning speed was better for the PPO [5].

Chapter 3

Modelling

In this section, the modelling of the Otter USV is presented. The Otter USV is for scientific and industrial applications and is manufactured by Maritime Robotics. It weighs 55 kg without payload and has a catamaran hull type. For propulsion, it uses fixed-pitched differential propellers at the rear of each pontoon. For steering, the propeller rotation speed is unevenly set between the two fixed propellers, inducing momentum on the Otter's body.

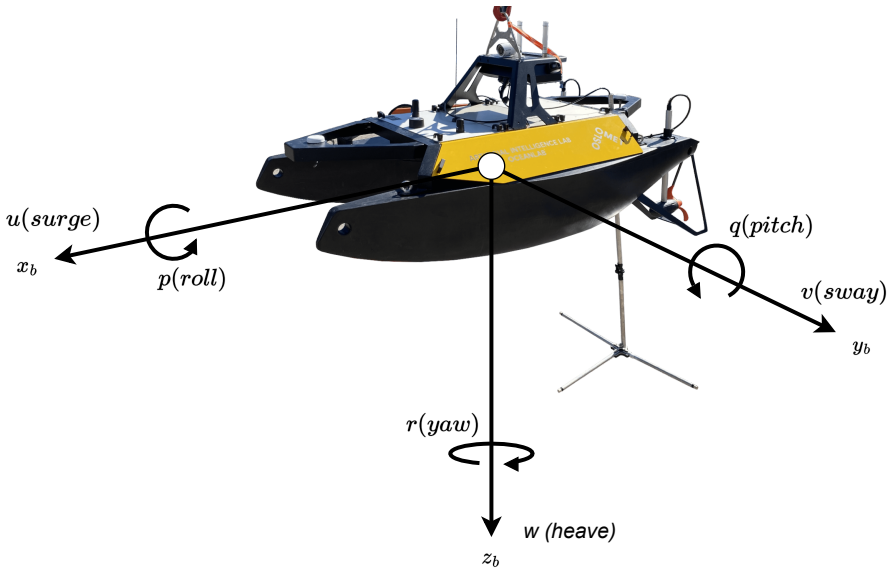


Figure 3.1: Figure of notation used in the Otter model

The following assumptions were made when modelling the Otter:

Assumption 1. *The motion in Heave, Pitch and Roll are neglected, resulting in a 3-degree-of-freedom (DOF) model.*

Assumption 2. *Environmental forces such as wind, and wave loads acting on the vessel are neglected.*

Assumption 3. *Ocean currents are neglected.*

Assumption 4. *The SBL receiver mass and drag are neglected.*

3.1 Kinematics

Kinematics is the study of the geometrical concern of an object and explains how an object changes its position through time. While working with marine vessel models in 3 or 6 degrees of freedom (DOF) it is convenient to represent the motion and orientation in an Earth-centred coordinate frame. In this thesis, we will use the Geographical reference frame, also known as tangent planes. The reference frame is noted as a Nort-East-Down frame (NED) and will be denoted as $\{n\} = \{x_n, y_n, z_n\}$, where the x_n points to true north, y_n is pointed towards east, and z_n point downwards normal to the earth surface. The origin of NED is defined as relative to the earth's ellipsoid. NED represents a tangent plane that we can use for local navigation, and the vessel's position and orientation will be described in the NED frame [13].

The body-fixed reference frame is fixed to the vessel and, in this thesis, is a moving coordinate frame relative to the NED frame. The origin of the frame o_b (CO) places midships on the waterline enabling port-starboard symmetry. The centre of gravity (CG), centre of buoyancy (CB) and centre of flotation (CF) are all located relative to the CO.

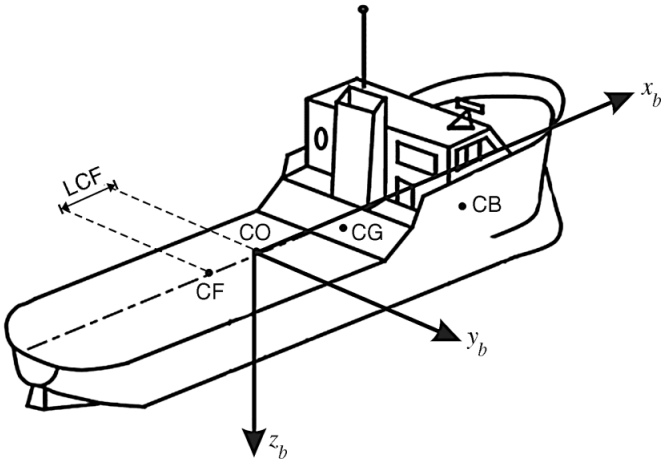


Figure 3.2: Body fixed reference point [13]

The naming convention in the thesis is in compliance with *The Society of Naval Architecture and Marine Engineers(SNAME)* convention and is described in the following table:

DOF	**Description**	**Forces and moments**	BODY **Velocities**	NED **Position and orientation**
1	Surge	X	u	x^n
2	Sway	Y	ν	y^n
3	Heave	Z	w	z^n
4	Roll	K	p	ϕ
5	Pitch	M	q	θ
6	Yaw	N	r	ψ

Table 3.1: Table of the SNAME convention

In table 3.1 we can see the generalised coordinate for NED and BODY listed. We define a common notation for the generalised position and orientation in NED:

$$\boldsymbol{\eta} = \begin{bmatrix} x^n & y^n & z^n & \phi & \theta & \psi \end{bmatrix}^T \tag{3.1}$$

Further, we denote a common vector for the velocities (time derivative for the generalised coordinates in NED):

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \dot{x^n} & \dot{y^n} & \dot{z^n} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{3.2}$$

Additionally, we denote the velocities in the BODY frame, due to its convenience when deriving the rigid body kinetics and equations of motion.

$$\boldsymbol{\nu} = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T \tag{3.3}$$

In calm waters, the changes in heave, roll, and pitch are small. Therefore, we can neglect these coefficients and simplify the representation to a 3 DOF model that considers changes in surge, sway, and yaw. We redefining definitions 3.1,3.2 and 3.3 to the following.

$$\boldsymbol{\eta} = \begin{bmatrix} x^n & y^n & \psi \end{bmatrix}^T \tag{3.4}$$

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\psi} \end{bmatrix}^T \tag{3.5}$$

$$\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}^T \tag{3.6}$$

In the next section we will define the rigid body kinetics in the convenient BODY frame. To transform the motion into NED- frame we need to define a rotation matrix that transform the velocity vector from BODY to NED for a 3 DOF model [13].

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{3.7}$$

where:

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

3.2 Rigid Body Kinetics

Kinetics is the study of forces causing a movement of an object. In this thesis we will model the vessel according to the marine vessel dynamics presented in [13]. The latter is fundamentally based on Newton's second law, which defines force as a product of mass and acceleration. Representing this in the previously mentioned $\{n\}$ frame, the convenience of the transformation 3.8 is evident, transforming the motion from BODY to NED.

$$mv_{ng} = f_g \tag{3.9}$$

Euler's first and second axiom expresses Newton's second law in terms of the conservation of energy, more specifically in terms of linear momentum and angular momentum. The Euler's representation makes it possible to represent the forces, moments and velocities in expressed in the BODY frame. Based on these fundamental principals, Fossen derived the robotic-inspired matrix-vector representation of the equation of motion [15]. The 3 DOF representation is defined as follows:

$$\boldsymbol{M_{RB}}\dot{\boldsymbol{\nu}} + \boldsymbol{C_{RB}}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau_{RB}} \tag{3.10}$$

where $\boldsymbol{\nu} = [u, v, r]^T$, expressed in $\{b\}$, $\boldsymbol{\tau_{RB}} = \begin{bmatrix} X & Y & N \end{bmatrix}^T$, is the generalised forces and moments acting on the vessel. $\boldsymbol{M_{RB}}$ is the rigid body system inertia matrix, representing the transnational forces acting on the vessel, and is defined as:

$$\boldsymbol{M_{RB}} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & I_z \end{bmatrix} \tag{3.11}$$

The $\boldsymbol{C_{RB}(\nu)}$ is the rigid body Coriolis and centripetal matrix, representing the rotational forces on the vessel and is defined as:

$$\boldsymbol{C_{RB}}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -mr & -mx_g r \\ mr & 0 & 0 \\ mx_g r & 0 & 0 \end{bmatrix} \tag{3.12}$$

The equation is usually defined as 6, 4, or 3 DOF [13]. In this thesis, we make use of both the 6 and 3 DOF variants. However, only the 3 DOF variants will be written out in this Theory section.

Remark. *3 DOF is used for the constraint in the optimisation problem for the NMPC, while the 6 DOF variant is used by the RL approach, and in the simulated performance test.*

3.3 Hydrodynamic Forces

A multitude of hydrodynamic forces exerts an influence on the motion of a surface vessel navigating within a marine environment. These forces arise due to the interaction between the vessel and the surrounding fluid medium, and their effects can have significant implications for the vessel's attitude and movement. In this section, we will build an understanding of the characteristics of these forces in order to take them into consideration when deriving the Manoeuvring Models for the Otter.

3.3.1 Hydrodynamic Mass-Damper

$$\boldsymbol{\tau_{hyd}} = \underbrace{-\boldsymbol{M_A}\dot{\boldsymbol{\nu}}_{\boldsymbol{r}} - \boldsymbol{C_A}(\boldsymbol{\nu_r})\boldsymbol{\nu_r}}_{Added\ mass} - \underbrace{\boldsymbol{D_p}\boldsymbol{\nu_r}}_{potential\ damping} + \boldsymbol{\tau_{visc}} \tag{3.13}$$

Where the added mass matrix $\boldsymbol{M_A}$ is due to the fluids surrounding the vessel

Lifting Forces Lifting forces occurs from two physical mechanisms. Linear Circulation around the linear circulation of water around the hull. The second is the crossflow drag.

Added mass definition: *"Hydrodynamic added mass can be seen as a virtual mass added to a system because an accelerating or decelerating body must move some volume of the surrounding fluid as it moves through it. Moreover, the object and fluid cannot occupy the same physical space simultaneously" [13].*

This means that the motion of a moving vessel will induce a force on the fluid surrounding it, making the fluid motion to the side and then close back behind when the vessel has surpassed the fluid. By considering that the fluid then possesses kinetic energy [19] we can follow Fossen and write the added mass matrix for a surface vessel in 3 DOF that includes system inertia M_A and Coriolis- Centripetal $C_A(\nu)$ for added mass:

$$\boldsymbol{M_A} = \boldsymbol{M_A}^T = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & N_{\dot{r}} \end{bmatrix} \qquad (N_{\dot{v}} = Y_{\dot{r}}) \tag{3.14}$$

$$\boldsymbol{C_A}(\boldsymbol{\nu_r}) = -\boldsymbol{C_A}^T(\boldsymbol{\nu_r}) = -\begin{bmatrix} 0 & 0 & Y_{\dot{v}}v_r + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u_r \\ -Y_{\dot{v}}v_r - Y_{\dot{r}}r & X_{\dot{u}}u_r & 0 \end{bmatrix} \tag{3.15}$$

where the following assumption where made for the Otter model (presumably best practise [14]):

$$X_{\dot{u}} = -1.0 * m \tag{3.16a}$$

$$Y_{\dot{v}} = -1.5 * m \tag{3.16b}$$

$$N_{\dot{v}} = -1.7 * R_{66} \tag{3.16c}$$

3.3.2 Dissipiative Forces

Dissipative forces exert force on the vehicle in a multitude of ways. In this thesis, we will consider the linear damping, non-linear surge-damping, and cross-flow as these are the component used in the Otter maneuvering model.

Linear damping
The linear damping matrix takes into consideration the frictional forces and the potential damping that exert the vessel.

$$\boldsymbol{D} = -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & Y_r & N_r \end{bmatrix} \tag{3.17}$$

where:

$$-X_u = B_{11v} = \frac{M_{11}}{T_{surge}} \tag{3.18}$$

$$-Y_v = B_{22v} = 0 \tag{3.19}$$

$$-N_r = B_{66v} = \frac{M_{66}}{T_{yaw}} \tag{3.20}$$

Remark. *In [14]* $X_u = \frac{-24.4g}{U_{max}}$

Nonlinear sway and yaw damping by crossflow
In the Otter model, there is used linear surge, resistance, and linear yaw damping. However, the sway damping term in the D matrix is, Y_ν is zero [14]. The damping in sway relies fully on the cross-flow drag. Following Fossen, we define the Nonlinear cross-flow damping terms to model the damping forces exerted on the vessel due to the flow of fluids normal to the hull.

$$Y = -\frac{1}{2}\rho \int_{-\frac{Lpp}{2}}^{\frac{Lpp}{2}} T(x) C_d^{2D}(x) |\upsilon_r + xr|(\upsilon_r + xr) dx \tag{3.21}$$

$$N = -\frac{1}{2}\rho \int_{-\frac{Lpp}{2}}^{\frac{Lpp}{2}} T(x) C_d^{2D}(x) x |\upsilon_r + xr|(\upsilon_r + xr) dx \tag{3.22}$$

3.4 Restoring Forces (hydrostatics)

To consider the restoring forces due to Archimedes (weight and buoyancy) we use the expression:

$$\tau_{hs} = -g(\eta) - g_0 \tag{3.23}$$

where $g(\eta)$ is a vector of generalised gravitational and buoyancy forces and g_0 is static restoring forces and moments due to the ballast system and water tanks. These forces are only present in heave, roll, and pitch, therefore in the 3 DOF-equation, the equations have no implications on the resulting forces. For details concerning the restoring forces please refer to [13, Ch. 4]. Additionally, we consider the waters to be calm, therefore the motion in heave, roll, and pitch will be neglected during the training of the controller, and when deriving the optimal controls.

3.5 Manoeuvring Model

In the previous sections, we defined the kinetics and kinematics of a marine surface vessel which explains how the exerted forces on the vessel change the acceleration and position and velocity of the vessel. The dynamics of the vessel are given by the vessel's kinematics and kinetics and give us a concatenated way of understanding how the vessel's velocity and acceleration change over time.

The dynamics of the Otter, used in the thesis, are defined by the following terms. The terms are derived from the code in [14], and the book [13].

$$\underbrace{\boldsymbol{M_{RB}\dot{\nu} + C_{RB}(\nu)\nu}}_{\textit{Rigid body forces}} + \underbrace{\boldsymbol{M_A\dot{\nu} + C_{RB}(\nu)\nu + D\nu_r + D_n(\nu_r)}}_{\textit{Hydrodynamic forces (added mass, crossflow and damping)}} = \boldsymbol{\tau_{RB} + \tau_{wind} + \tau_{wave}} \tag{3.24}$$

where:

$$\boldsymbol{\nu_r} = \boldsymbol{\nu} - \boldsymbol{\nu_c} (\text{is the relative velocity vector}) \tag{3.25}$$

$$\boldsymbol{\nu_c} = [u_c, v_c, 0]^T \tag{3.26}$$

where the ocean current is of constant speed V_c and direction β_{V_c}, the heading is defined as ψ

$$u_c = V_c cos(\beta_{V_c} - \psi) \tag{3.27}$$

$$v_c = V_c sin(\beta_{V_c} - \psi) \tag{3.28}$$

Time derivatives give according to to Fossen [13, p.157]

$$\dot{u}_c = r v_c \tag{3.29}$$

$$\dot{v}_c = -r u_c \tag{3.30}$$

Equation 3.24 defines the dynamics of the vessel. On the left-hand side, we see rigid body forces together with the hydrodynamic forces. The left-hand side must be equal to the forces applied by the effectors (propellers) on the vessel, in addition to wind and wave forces.

From a controller perspective, it is advantageous to represent equations in a state-space format, as it provides a means to comprehend the dynamic system's state. In order to ascertain the subsequent time step for the Otter, or any other vessel, it is imperative to define its acceleration and velocity. Subsequently, by utilizing integration techniques along with an initial position in NED, the vessel's position and velocity at the next time step can be calculated in a convenient way provided by the state space.

Since the Otter model is defined with relative ocean currents, we follow Fossen on [13, p. 157] and define the generalized position and relative velocity in a state-space manoeuvring model.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\boldsymbol{\psi})(\boldsymbol{\nu_r} + [u_c, v_c, 0]^T) \tag{3.31a}$$
$$\dot{\boldsymbol{\nu}}_{\boldsymbol{r}} = (\boldsymbol{M_A} + \boldsymbol{M_{RB}})^{-1}(\boldsymbol{\tau} - \boldsymbol{C_A}(\boldsymbol{\nu_r}) - \boldsymbol{C_{RB}}(\boldsymbol{\nu_r}) - \boldsymbol{D}\boldsymbol{\nu_r} - \boldsymbol{D_n}(\boldsymbol{\nu_r})\boldsymbol{\nu_r}) \tag{3.31b}$$

Where $\boldsymbol{\tau}$ is the thrust vector defining the forces and moments provided by the propellers in surge (X), sway(Y) and yaw (N) $\boldsymbol{\tau} = [X, Y, N]^\intercal$. Since the Otter is a catamaran with a fixed propeller at each pontoon and on propellers in sway, the Y = 0 at all times.

3.6 Control Allocations

The control allocation problem seeks to distribute the generalised control forces $\boldsymbol{\tau}$, into controller inputs $\boldsymbol{u}$. We follow [13] and [5] and define:

$$\boldsymbol{\tau} = \boldsymbol{B}\boldsymbol{u} \tag{3.32}$$

$$\boldsymbol{B} = \boldsymbol{T}\boldsymbol{K} \tag{3.33}$$

Where $n_{1,2}$ are the left and right thruster rpm respectively. $\boldsymbol{K}$ is the diagonal force coefficient vector. $\boldsymbol{T}$ is the thrust configuration matrix and $\boldsymbol{u}$ is control input in rpm for each thruster respectively.

We solve for $\boldsymbol{u}$:

$$\boldsymbol{u} = \underbrace{\begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix}^{-1}}_{\boldsymbol{K^{-1}}} \underbrace{\begin{bmatrix} 1 & 1 \\ -l_1 & -l_2 \end{bmatrix}^{-1}}_{\boldsymbol{T^{-1}}} \begin{bmatrix} \tau_1 \\ \tau_3 \end{bmatrix} \tag{3.34}$$

where

$$l_1 = -l_2 = -Y_{pont} \tag{3.35}$$

$$k_i = \begin{cases} k_{pos} & \text{if } n_i > 0 \\ k_{neg} & \text{otherwise} \end{cases} \tag{3.36}$$

Chapter 4

Model Predictive Controller approach

This chapter is presenting the theory, implementation, and results of the MPC.

4.1 Theory of Model Predictive Controller

Model Predictive Controller (MPC) is an optimal control approach that computes optimal control signals for complex multi-variate control problems. In order to effectively control a system, an MPC relies on a numeric dynamic model and an optimisation algorithm. It is crucial that the dynamic model accurately represents the input-response of the system. Furthermore, the MPC uses optimisation techniques to determine the optimal control inputs that will lead to reaching the setpoint while staying within the system's boundaries. MPC has proven viable for driving the output constraints to their optimal set points while maintaining the boundaries in the optimisation problem, without excessive movement of the input variables [30], [26].

NMPC algorithms are implemented to solve the optimal control problem and compute an optimal control trajectory in prediction horizon N. At each time step T, the algorithm solves the optimal controls u for the time horizon N with the sampling time k.

We generalise the NMPC problem as follows:

$$\begin{aligned}
\underset{x,\,u}{\text{minimize}} \quad & J(x,u,x_{ref}) = \sum_{k=1}^{N+1} \|x_i - x_{ref,i}\|_Q^2 + \sum_{k=1}^{N} \|\Delta u_i\|_R^2 && \text{(4.1a)}\\
\text{subject to} \quad & x_{i+1} = f(x_i,u) \forall i \in 0...,N, && \text{(4.1b)}\\
& x_0 = x_{initial}, && \text{(4.1c)}\\
& x_i \in \mathcal{X}, && \text{(4.1d)}\\
& u_i \in \mathcal{U} && \text{(4.1e)}
\end{aligned}$$

The objective is to minimise the cost function $J(x,u,x_{ref})$. J is the summed costs over the prediction horizon N, representing a function one wants to minimise. $\boldsymbol{x}$ is the state of the system, $\boldsymbol{u}$ is the control inputs, and $\boldsymbol{x_{ref}}$ is the set point or

the wanted state for the system. The objective function has two elements; the state part $\boldsymbol{x}$ multiplied with a weighting matrix $\boldsymbol{Q}$. The weight matrix prioritises parts of the state to which we want the controller to pay attention. The same applies to the second element $\boldsymbol{u}$ and the $\boldsymbol{R}$ matrix, whereas this is to keep the controller input from oscillating [7] [30].

NMPC has numerous parameters for tuning. Firstly the prediction horizon N and the sampling time Δt should be set so that $N\Delta t = t_s$ where t_s is the settling time for the system. Typical ranges of N is $30 \geq N \leq 120$ [30, Chp. 20]. Secondly, if the objective function includes values of different ranges and units, it may be advantageous to make these dimensionless [30].

4.1.1 Discretization Methods

NMPC problems are discrete, but the models used to derive system dynamics are mostly continuous state space functions. Thus, we must discretise the models to create a trajectory of the system response based on the control inputs. Various techniques exist, such as Eulers Method and Runge-Kutta Fourth-order Method [13]. Runge-Kutta Fourth-order Method is defined as follows:

$$k_1 = hf(x[k], u[k], t_k) \tag{4.2a}$$
$$k_2 = hf(x[k], k_1/2, u[k], t_k + h/2) \tag{4.2b}$$
$$k_3 = hf(x[k], k_2/2, u[k], t_k + h/2) \tag{4.2c}$$
$$k_4 = hf(x[k], k_3/2, u[k], t_k + h) \tag{4.2d}$$
$$x[k+1] = x[k] + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.2e}$$

4.1.2 Optimisation and Nonlinear Programming

Nonlinear optimisation is a mathematical technique to find the optimal solution to a problem when the objective function and constraints involve nonlinear expressions. Nonlinear Programming Problems (NLPP) is a branch of nonlinear optimisation that optimises objective functions subject to nonlinear constraints [30].

IPOPT is an algorithm that solves NLP using an interior-point line-search filter method. The method involves iteratively solving a sequence of barrier problems, which are a series of problems that penalise points that violate the constraints. IPOPT computes a search direction at each iteration by solving a linearised version of the barrier problem. It then updates the current solution by moving toward the search direction while ensuring the new point satisfies the constraints. IPOPT also utilises a line search algorithm to determine the step size at each iteration, which helps ensure that the algorithm converges to a global minimum of the objective function [30], [36].

4.1.3 Casadi

The Casadi framework is an open-source optimisation framework for scientific and industrial applications. It is written in the programming language C which is commonly known for its high computational efficiency. The Casadi framework is used in [18], [38] and is also used in several other high level control-application within NMPC developent.

4.2 NMPC Implementation

The NMPC was implemented using the Casadi symbolic framework in Python. As shown in figure 4.1 the NMPC implementation consists of a 3 DOF model implemented in the symbolic framework of Casadi together with an integrator that conducts the discretization process of the continuous state space. Lastly, the NMPC contains an optimiser that computes the nonlinear optimisation problem to compute the optimal control forces for the vessel.

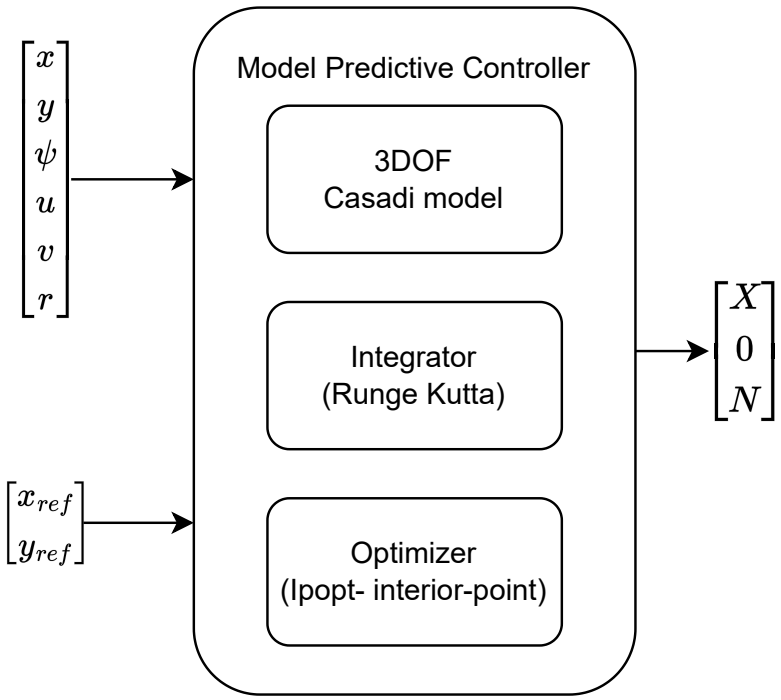


Figure 4.1: The Model Predictive Controller

4.2.1 3 DOF Manouvering Model in Casadi

The state-space model implemented in Casadi is a maneuvering model following the theory section 3.5. We define the ordinary differential equations (ODE) as a state-space with the state vector $\dot{x} = [\dot{\eta}, \dot{\nu}]^\intercal$ where $\dot{\eta} = [\dot{x}, \dot{y}, \dot{\psi}]^\intercal$ and $\nu = [\dot{u}, \dot{v}, \dot{r}]^\intercal$. We define:

$$\dot{x} = \begin{bmatrix} \dot{\eta} \\ \dot{\nu}_r \end{bmatrix} = \begin{bmatrix} R(\psi)(\nu_r + [u_c, v_c, 0]^\intercal) \\ (M_A + M_{RB})^{-1}(\tau - C_A(\nu_r) - C_{RB}(\nu_r) - D\nu_r - D_n(\nu_r)\nu_r) \end{bmatrix} \tag{4.3}$$

$\tau = [X, 0, N]^\intercal$ where X is the force in surge and N is the moment among yaw, see table 3.1. $M_A, M_{RB}, C_A, C_{RB}, D$ and D_n is explained in the theory section 3.5.

Remark. *The ocean currents are 0 and the Otter has no thrust in sway. Therefore, Y is constrained to 0 in the NMPC.*

4.2.2 Model Discretization

The state space model used in this paper is constructed so that both x, y, ψ together with u ν and r in one column vector. The first part in the column vector is the NED- frame as is considering the kinematics, hence the vessel's movement in the NED frame. The last three elements concern the velocities of the vessel in the BODY frame, aka CO. As seen from the ode equation 4.3, the state space is represented by the acceleration and the velocity. The state-space ode must be integrated over time to obtain the velocities and position.

We define the discrete map of the state space denoted f_d, and we redefine the $\tau = u$ for consistency with standard notations for controller inputs. Then we define:

$$\hat{x} = x(k+1) = f_d(\dot{x}, u) \tag{4.4}$$

Where $x(k+1)$ is the state of the system given the controller inputs one step into the future. The discrete map is solved using the integration technique Runge Kutta method showed in equation 4.2, which was implemented as a function by Casadi.

The next step is to implement the optimisation problem. The Casadi framework provides a simple class called "Opti", where we define the ODE, integrator and solver in one class. The optimisation problem is as follows:

$$\begin{aligned}
\underset{x,\,u}{\text{minimize}} \quad & J(x, u, x_{ref}) = \sum_{k=1}^{N} (\|\hat{x}_{x,y} - x_{ref}\|_Q^2 + \|(u(k)\|_R^2 && \text{(4.5a)}\\
\text{subject to} \quad & x_{i+1} = f(x_i, u) \forall i \in 0..., N, && \text{(4.5b)}\\
& x_0 = x_{initial}, && \text{(4.5c)}\\
& x_i \in \mathcal{X}, && \text{(4.5d)}\\
& u_i \in \mathcal{U}, && \text{(4.5e)}\\
& u_{min} < u_i < u_{max} && \text{(4.5f)}
\end{aligned}$$

Where $\hat{x}_{x,y}$ is the position of the vehicle in NED. $x_{ref} = \begin{bmatrix} x_{ref} & y_{ref} \end{bmatrix}$ is the target in NED. R is the weighting matrix for the controller change. Q is the weighting matrix for the state vector. u_{min} is the $[X_{min}, N_{min}]$, u_{max} is the $[X_{max}, N_{max}]$.

4.2.3 Solving the Optimal Control Problem

For each solution provided, the NMPC solves the optimal control trajectory within the prediction horizon N. If the target is within reach of the vessel in the prediction horizon, the solver will find the ending solution. Despite that the optimal controls

are computed for the whole prediction horizon, only the first control element in the trajectory is sent to the controller allocation. Then for the next time step, an entire optimal trajectory is computed. The reason for this is to encounter the difference in real system response vs the model-based response from which the controller gain is calculated. The initial guess is set equal to the last prediction control and state trajectory. This means that the u-vector trajectory is set to be the last solution. The solver used in this thesis is the IPOPT algorithm as explained previously.

4.3 Simulation

The simulation of the controller test uses the edited version of the *Fossen Vehicle simulator* for calculating the dynamic response and the NMPC developed by the author to compute actions. The workflow of the simulation is illustrated in figure 4.2.

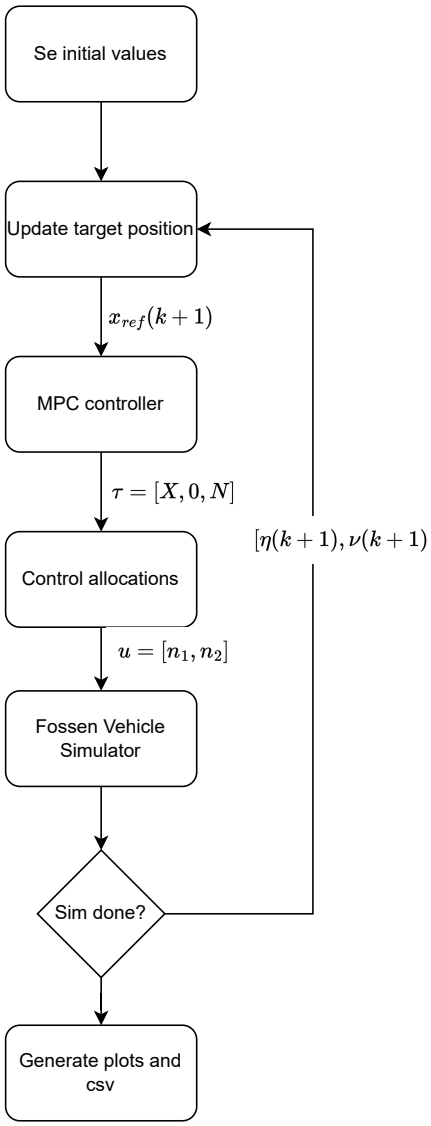


Figure 4.2: NMPC simulation workflow

The NMPC has a variety of parameters that can be adjusted to tune performance and computation speed. In table 4.1 the parameters for the NMPC performance test is listed.

Parameter	**Value**
Simulation sample time	0.1
u_{min}	X = -116 N , N = -73 Nm
u_{max}	X = 150 N , N = 73 Nm
N_{NMPC}	10
NMPC sample time	0.2
R	[0.5,0 ,0.5]
Q	[10000, 10000]
Solver	IPOPT
Max iterations	1000

Table 4.1: NMPC and simulation parameters for the test case

The simulation sample time is the time at which the simulation is computing the dynamic response between each controller action denoted as $\eta(k+1)$and $\nu(k+1)$ in figure 4.2. The forces control limits are u_{min} and u_{max} with the surge element N with unit $Newton$ and yaw element N with unit Nm, and defines the upper and lower force limits that the thrusters can exert on the Otter. The N_{NMPC} element in table 4.1 (not to be confused with yaw) is the prediction horizon and defines how many time steps ahead the NMPC should compute controller signals for. *NMPC sample time* is the sample time for the internal Casadi 3DOF model. R and Q are the weighting matrix and are defined through trial and error and guidance from [30]. The *Solver* is the nonlinear programming solver that is used to solve the optimal control problem, and it has a maximum of 1000 iterations to find a solution to the problem. If no solution is found, the simulations is aborted.

4.4 NMPC Simulated Performance Test

This section presents how the NMPC test was conducted in the simulations and the results of the model predictive controller approach.

4.4.1 Setup

In the test scenario, the Otter is placed in the position $\eta_{x,y,\psi} = [0, 0, 0]$. The target is a virtual target simulating AUGs movement in x and y. The AUGs starting position is set to $x_{ref} = [5, 5]$, and the target is moving in a circular pattern of a radius of 200m with a velocity of 0.25 m/s. The simulation length is 8000 steps and with a sampling time of 0.1 this results in an 800-second simulation period.

4.4.2 Results Case 1

In this section, the results of the simulated NMPC test are presented. First, a plot showing the trajectory of the Otter and the Target in NED, and then some consider-

ations of the error between the two are presented. Next, we move over to controller output before looking into the computation time of the optimal control problem.

The spatial plot, figure 4.3, shows a target starts with an offset from the Otter. The Otter is quickly changing the heading towards the target and with a minor oscillation overshoot from the target trajectory, the Otter settles on the tail of the target and adequately tracks the target, as indicated by the arrows.

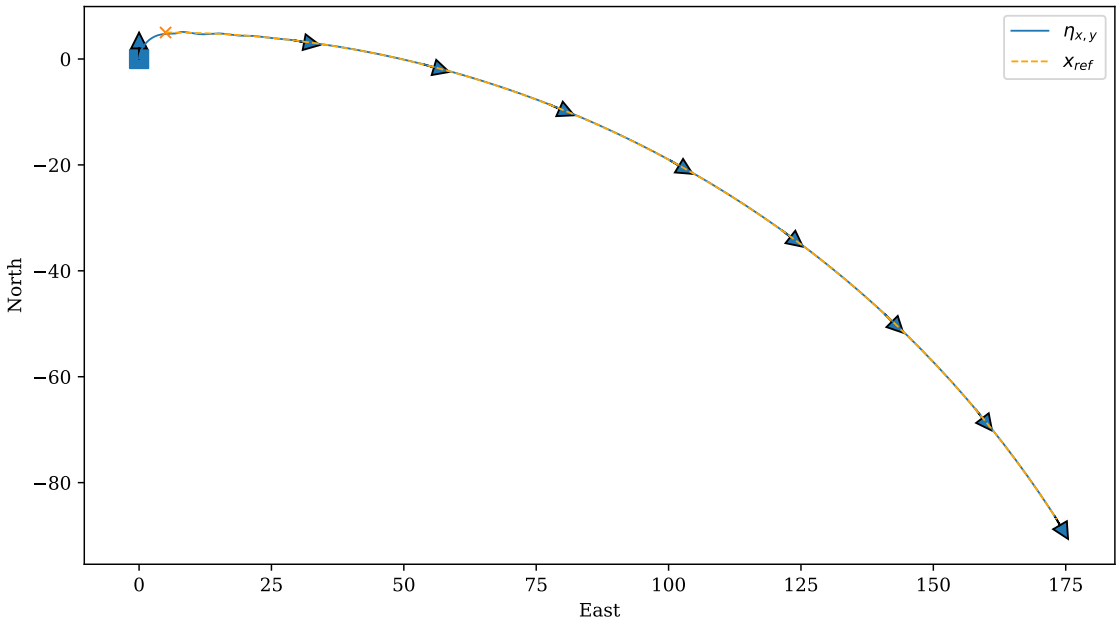


Figure 4.3: NMPC performance in NED

Figure 4.4 shows the absolute distance error between the Otter and the target in units m. There is a rapid change in the error that settles around 0.7m. The error plot might indicate room for improved controller tuning to reduce the error further. The same oscillating behaviour as in the NED plot can be observed at the start, before the line flattens out, where the Otter settles at the same speed as the target velocity.

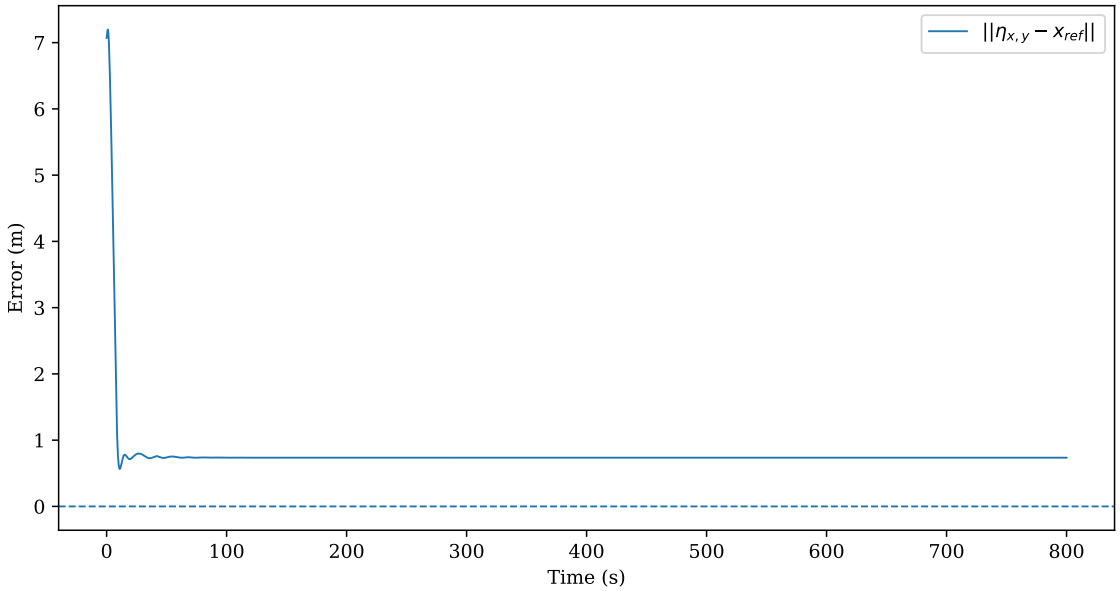
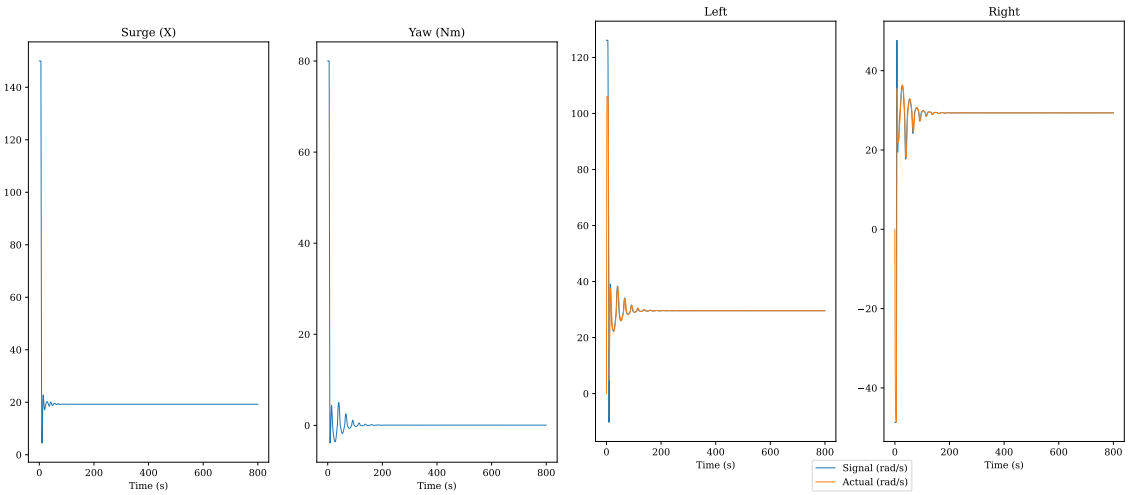


Figure 4.4: NMPC performance error

The control forces computed from the NMPC controller are shown in figure 4.5a and are the inputs to the control allocations. Further, in figure 4.5b, the resulting control allocation outputs, in the form of rpm, for the left and right propeller is shown together with the actual rpm. The actual rpm is a simulation of the thruster dynamics and includes a time lag in ramping the thruster revolutions. The NMPC and control allocations do not have access to the thruster dynamics. As we observe from the figure, the same oscillating start can be observed in the computed optimal forces and the responding rpms.



(a) Control Forces (Surge (N), Yaw (Nm)) (b) Thruster RPM

Figure 4.5: Controller responces

The responding velocities of the Otter during the performance test are depicted in figure 4.6. We can see that the velocities follow the same pattern as the controller signals by oscillating at the start before stabilising and aligning with the target.

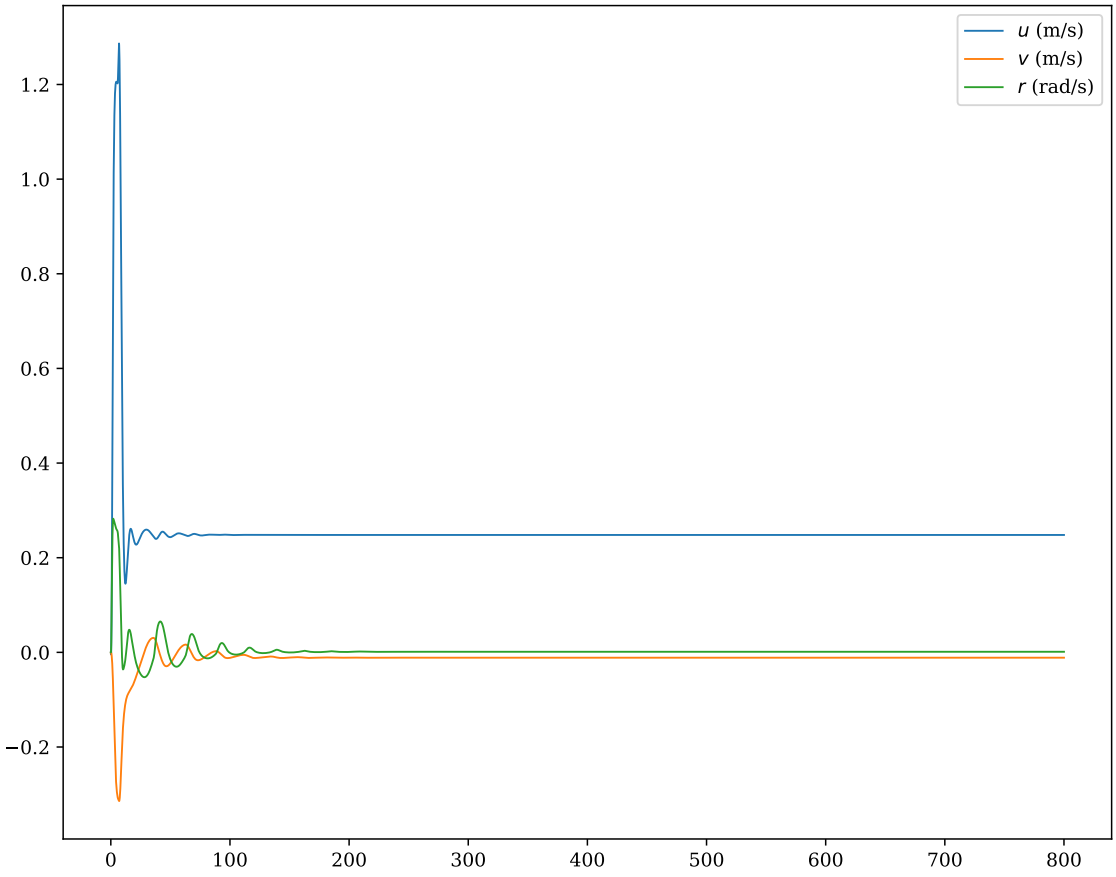


Figure 4.6: Velocities

In figure 4.7, the logged time it takes to solve the nonlinear programming problem and return the optimal control. As shown, it takes approximately from 0.5 to 0.6 seconds to compute the optimal controls for the Otter.

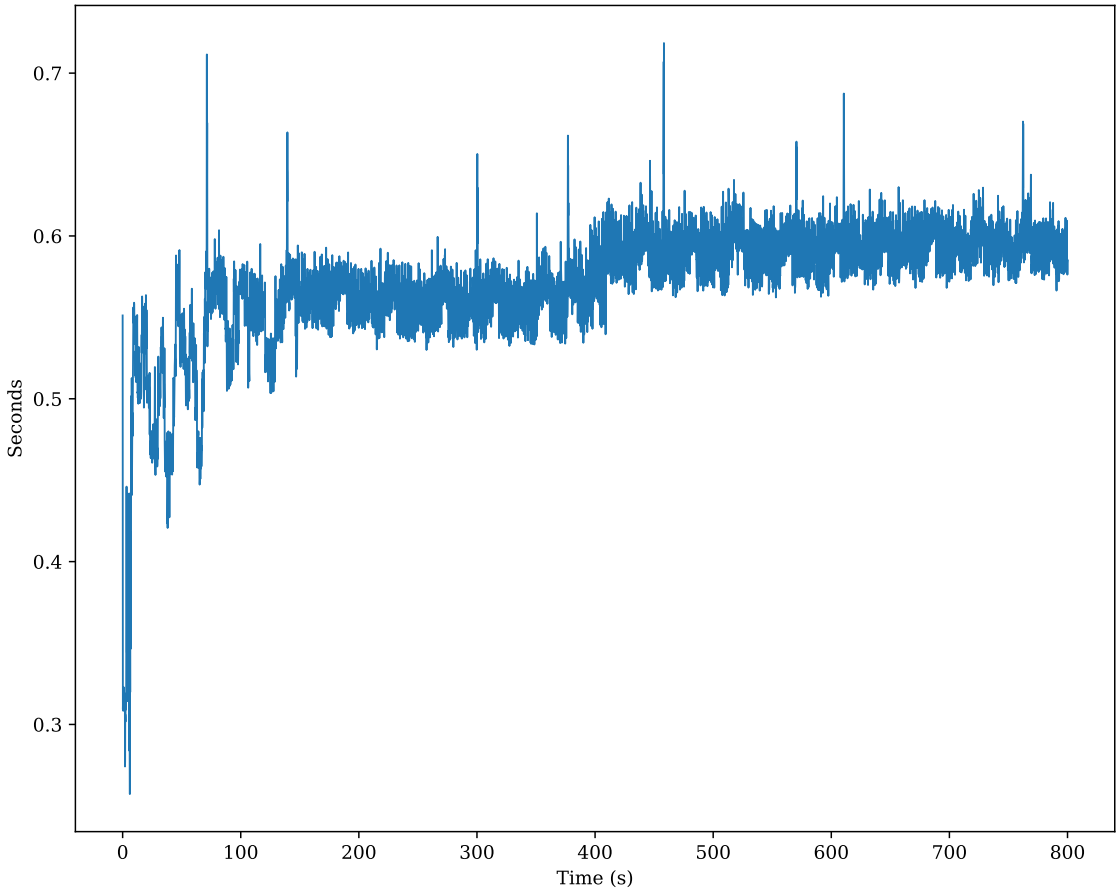


Figure 4.7: Compute time to find optimal control action

4.4.3 Results Case 2

Case 2 is identical to Case 1, except the target is moved to the initial starting position [20,20]. Figure 4.8 and 4.9 show the response of the Otter while tracking the target. The distance to the target is rapidly reduced before adequately following the target. The same oscillating behaviour is present at the start and is settled after about 200 seconds.

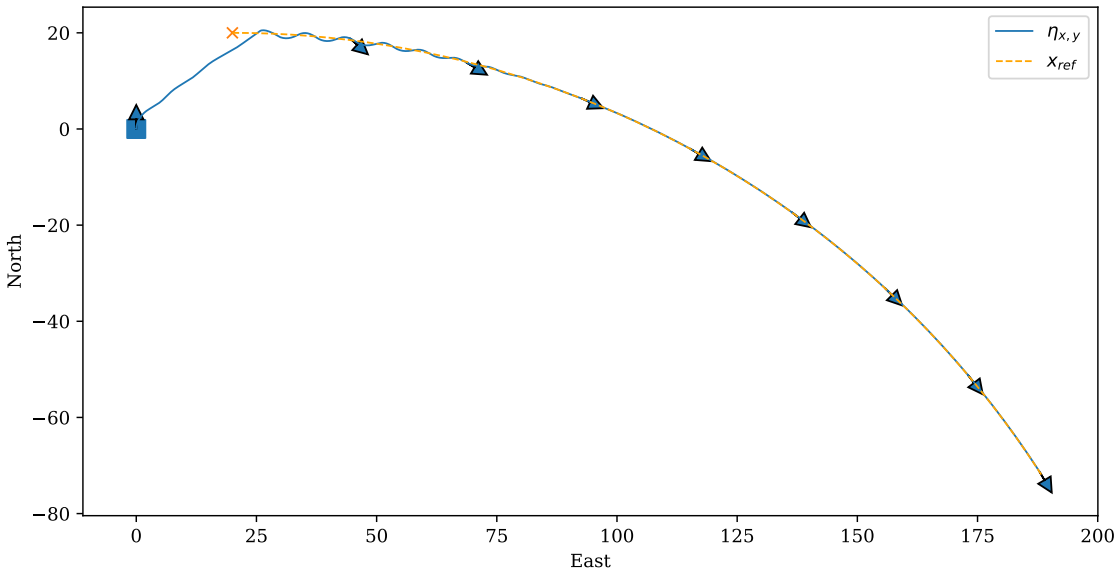


Figure 4.8: Case 2 NMPC performance in NED

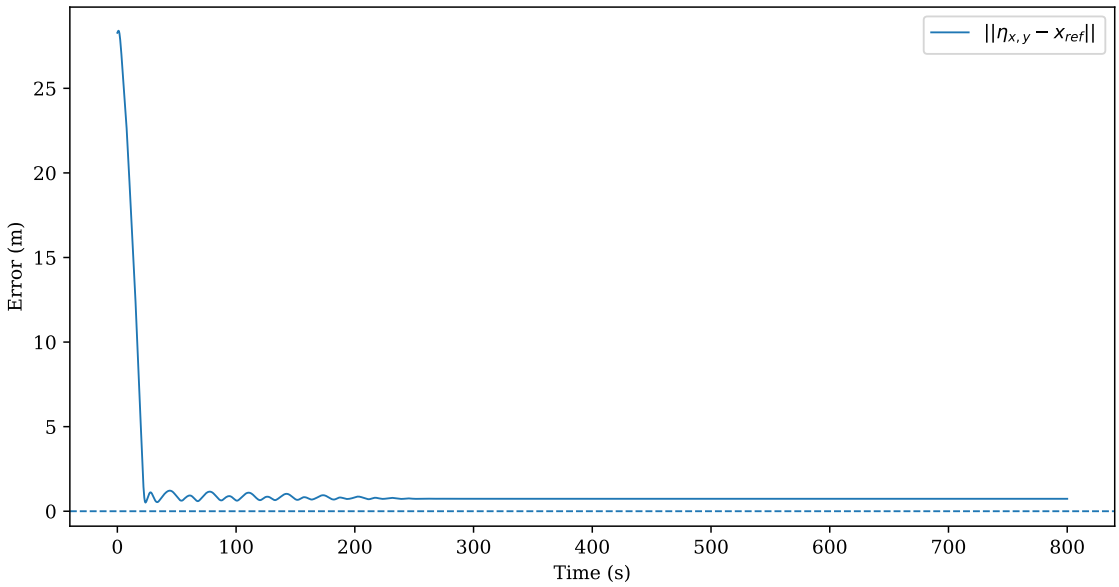


Figure 4.9: Case 2 NMPC performance error

4.5 Discussion

4.5.1 Error from Target

The error settles on 0.7 m. In the simulated environment that is not prone to signal disturbance GPS and position, it is reasonable to presume that the controller should be able to close this gap even further. Conversely, the error is likely because the NMPC observes the Otter intercepting the target within the prediction horizon when deriving the optimal controls. Only the first controller action in the optimal controller trajectory is conducted. Therefore, the Otter never conducts the finishing control action for the trajectory that would reduce the error to zero. Then, at the

next step, the target has moved, and the same situation occurs, resulting in a loop where there is a constant difference between the Otter and the target. There might be several mitigation methods for this, but 0.7 meters is an acceptable accuracy for practical applications for open water target tracking applications. In addition, the NMPC is implemented to use the η that would be measured with a civilian GPS with an accuracy of 3 meters in the horizontal positioning [24]. Thus, the GPS error is over four times larger than the constant error in the simulation. Because the SBL is mounted on the Otter and gives the readings relative to the Otter, the NMPC would calculate its controller actions on a position that is $\pm 3m$. A 3m accuracy might not be enough for the optical communication to work correctly. Furthermore, disturbances such as wind, waves and currents would cause the system to experience even more errors. In such a scenario, where the GPS provides insufficient accuracy for the optical communication operation, the NMPC would need re-implementation and other techniques to measure the position.

4.5.2 Controller Tuning

The range of the values in the Casadi model is not normalised. This means that the range of x and x_{ref} is larger than that of u. Therefore the weighting matrix of the two elements in the objective function is of different scales. However, this is not crucial for functionality, and as shown in figure 4.3, the NMPC is robust enough to tackle the scaling difference. In future work, making the two variables' ranges the same might be beneficial so that the control element and the spacial error between the Otter and the target are somewhat the same scales.

Referring to figure 4.5, the controls have a smooth transition between the sampling time, despite the oscillating pattern at the start. In the initial controller steps, the controller creates significant controller inputs but settles fast.

4.5.3 Computation Time

In figure 4.7, you can see the time it takes for the NMPC to calculate a control action. The computation time is around 0.6 seconds (1.6Hz), in the same range as in previous works (see section 2.2.1). However, the computation time might be too long for tight control sampling. Nevertheless, since the Otter thruster dynamics settle slowly, taking about 5-10 seconds, a new controller action every 0.6 seconds may be effective in practical applications and on calm waters. On the other hand, the Otter is a relatively small craft with a weight of 55 kg and a full speed of about 1,5 m/s; the Otter will move almost one meter between each controller signal with a sampling time of 0.6 seconds. Additionally, with external environmental forces inducing additional acceleration of the Otter, a change in thruster signals every 0.6 seconds will likely result in oscillating behaviour. The literature review outlined an effective strategy to reduce the control's computation time. The method involves linearizing the model and solving the linear optimal control problem, which can significantly reduce computation time. However, it may slightly impact the accuracy of the controller.

4.6 Summary

In this chapter, the NMPC approach as a whole has been presented. The model used for the NMPC is nonlinear, resulting in a nonlinear model predictive controller. For the implementation, the Casadi framework was used in modelling and discretisation, and the optimisation solver was IPOPT. The simulation and the results show that the NMPC works satisfactorily and reduces the distance between the Otter and the target to $\approx 0.7m$. However, the controller has a long computation time of 0.6 seconds which may make the controller unfeasible in environments that exerts the Otter for forces requiring faster responses than this NMPC will manage.

Chapter 5

Reinforcement Learning Approach

5.1 Artificial Intelligence Theory

As early as 1952, IBM'er Arthur Samuel coined the word Machine Learning (ML), and a few years later, in 1956, the cognitive scientist Marvin Minsky used the term Artificial Intelligence (AI). Today these terms are commonly known among scientists and the population in general. Broadly, AI systems are considered any system that may conduct intelligent behaviour and according to the Oxford Dictionary, ML is a "(..) system that becomes smarter as it encounters additional data". Furthermore, ML is usually categorised into three main approaches: supervised, unsupervised, and reinforcement learning. This thesis will focus on the reinforcement learning category. As shown later in the thesis, Deep Reinforcement Learning is used to train an *Agent* to perform the controller tasks of target tracking. In the following sections, the fundamental parts of machine learning will be presented as it lays the foundation for the reinforcement learning used in this thesis, following the work of [35] and Khan [17].

5.1.1 Artificial Neurons

Artificial Neural Networks(NN) is a type of mathematical function modelling that draws inspiration from the biological neural networks found in humans and animals in nature. A biological neural consists of an axiom, dendrites, a nucleus and synapses, but the two significant parts replicated in an NN are the nucleus and the synapsis. The "calculation" is conducted in the nucleus, and the synopsis connects each neuron. Please refer to figure 5.1. In short, a neuron will receive an input stimulus, and if the chemical threshold is reached, the neuron will spike and output a signal to the connected neurons, possibly creating a chain of spiking neurons [20, Ch. 7], [3].

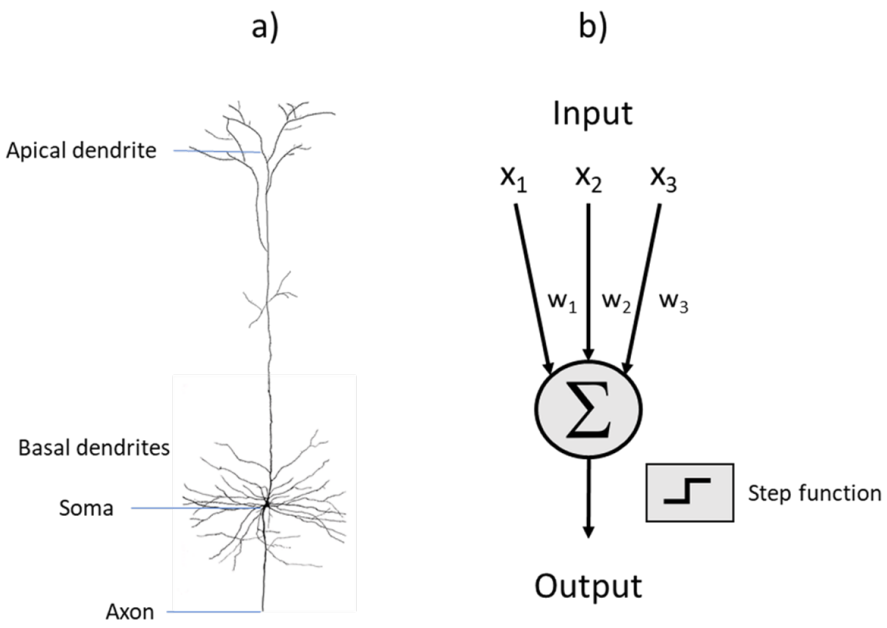


Figure 5.1: Biological (a) and artificial(b) neuron,

In 1943 McCulloch and Pitts created a model that took in unweighted inputs with an activation function that gave a binary output [23]. In 1958, presumably inspired by McCulloc-Pitts in addition to the Hebbian Placisity [11], Frank Rosenblatt developed his Perceptron Machine, enabling weighted inputs and various activation functions to produce a binary output classifier machine [27]. The McCulloch - Pitts neuron and Rosebratts Perceptron Machine are the first commonly known and established models that replicate the functionality of a biological neuron that lays the foundation for the modern artificial neuron. The perceptron machine may be defined as follows: given an input vector x, a weight matrix ω and bias b, the output of the perceptron is given by:

$$f(x) = \begin{cases} 1 & \text{if } \omega^\intercal x + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Modern artificial neurons consist of three major parts: weights and biases, collectively called parameters, and an activation function. The weights are the Hebbian plasticity that makes the neuron *learn*. For example, if a feature is highly related to a class in a dataset, then a high-weight value is beneficial. In figure 5.2, we can observe the input vector x multiplied by the weights ω. Further, we see a summing function that multiplies all the weights (and biases if present) before passing it to an activation function S [3].

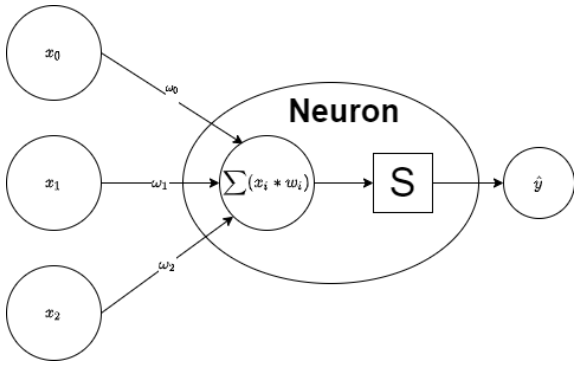


Figure 5.2: Artificial neuron

The activation function is a crucial component in emulating the behaviour of biological neurons. Although a biological neuron will only fire when a specific stimulus threshold is reached, the output is not equivalent to the added inputs. On the contrary, the output may differ greatly from the input stimulus. To emulate this behaviour, we use activation functions that give an output in a scale defined by the function, not by the inputs.

There is a variety of activation functions. However, due to the proficient ability to handle vanishing gradients, the rectified linear (ReLU) is a typical activation function in the hidden layers, showed in eq. 5.2. In contrast, the output layers often have a linear or nonlinear activation function [31]. Nonlinear activation functions may be Sigmoid as shown in eq.5.3.

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

$$\phi(z) = -\frac{1}{1 + e^{-z}} \tag{5.3}$$

The characteristics of the activation function are, in general, that they must be continuous if we want to use gradient descent algorithms to optimise the parameters of the neural network.

5.1.2 Artificial Neural Networks

An artificial neural network consists of several artificial neurons, often connected in a 2D structured grid with rows and layers, as illustrated in figure 5.3. The structure is defined using graph theory, and one can define the topology connecting the artificial neurons. Graph theory provides a variety of structure that follows rules which defines the behaviour of the graph. For example, a classical deep neural network (DNN) uses acyclic graphs. An acyclic graph means that the data can only go in one direction in the graph. Other typologies may involve data reoccurring in a layer or a node within a network and therefore use bidirectional graphs. The latter is called a Recurrent Neural Network (RNN) and is frequently used in temporal models and environments. However, an alternative to RNN is stacking several observations and processing them contemporary through the dense network instead [28], [6].

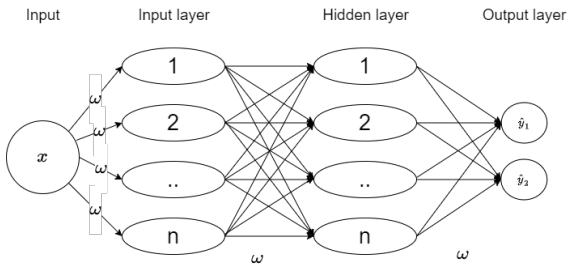


Figure 5.3: Artificial Neural Network

5.1.3 Optimisations - Gradient Decent

As the aforementioned section on NNs describes, designated synaptic weights exist between each neuron in an artificial neural network. These weights are essential for adjusting the trails within the network, thereby training it to produce the desired output. Unfortunately, the numerical value of these weights for each neuron is usually unknown at the outset, and analytical determination is only occasionally possible. Thus, iterative optimisation techniques are necessary to obtain the optimal weights in a neural network [17]. Gradient descent is an iterative technique that uses the gradient in the NN. The gradient is the vector with partial derivatives of the objective function about a synaptic weight within the NN. Accordingly, the gradient explains how we adjust the weights to reduce the error, and the adjustment scale is decided by the learning factor η [17].

$$\nabla_\omega E = [\frac{\partial E}{\partial \omega_1}, \frac{\partial E}{\partial \omega_2}, ..., \frac{\partial E}{\partial \omega_d}]^T \tag{5.4}$$

$$\Delta\omega_i = -\eta \frac{\partial E}{\partial \omega_i}, \forall i \tag{5.5}$$

$$\omega_i = \omega_i + \Delta\omega_i \tag{5.6}$$

where E is the objective function, η is the learning rate and ω are the synaptic weights. To determine the error, we compare the predicted NN output to a ground truth value. The squared error function is frequently used due to its ease of calculation and analytical tractability [16].

$$\epsilon(x) = (x_t - x_p)^2 \tag{5.7}$$

Where x_t is the ground truth and x_p is the output from the NN.

5.1.4 Reinforcement Learning

This section presents the background and the theory for reinforcement learning (RL). The history of RL is rooted in educational psychology, which studies how animals

use the game of trial and error to learn, and optimal controls concerning vale functions and dynamic programming. Optimal controls occurred in 1950 when Richard Bellman introduced the "optimal return function", now known as the Bellman equation. The groups of methods to solve this equation came to be known as dynamic programming and the Markov decision process (MDP) [4]. RL is a training method where we train an Agent to conduct a task using reinforcement techniques.

Markov Decision Process

The foundation for RL is the Markov Decision Process (MDP). MDP is a mathematical discrete-time stochastic process control and provides the bedrock for all RL methods [35]. MDP is categorised into discrete or stochastic processes. In the discrete case, the translation between two states is a function with a deterministic value $x' = f(x, u)$. Conversely, for the stochastic case, the state translation is the probability distribution $P(x'|x, u)$, resulting in a random new state in different scenarios. MPD may be described by a tuple $(\mathcal{X}, \mathcal{U}, \mathcal{R}, \mathcal{T})$ where the $\mathcal{X}$ is the states, $\mathcal{U}$ are the available actions. The available actions will be dependent on the state of the environment. We denote the state-action mapping as $\mathcal{U}(x) \in \mathcal{X}$. The reward $\mathcal{R}$ is the state dependant action denoted $\mathcal{R}(x, u)$, where $x \in \mathcal{X}, u \in \mathcal{U}$

In the case of using an MDP in an RL we may refer to figure 5.4, which illustrates this process and say that we have an Agent in an environment that acts upon a policy denoted as π_t. At each time step t, the RL-Agent will get information about its state s_t, where $s_t \in S$, and choose an action A_t, among the available action denoted as $A(St)$ based on that state information. After selecting and carrying out the action, the Agent is given information about its new state, S_{t+1}, as well as a numerical reward, R_{t+1}.

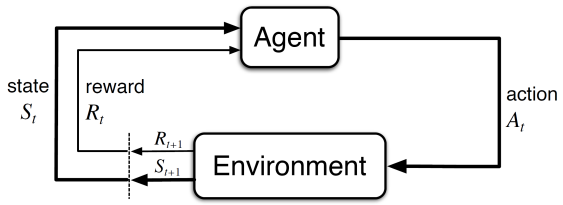


Figure 5.4: Markov Decistion Process

During training, the Agent develops a mapping from states to actions. This mapping is the resulting Agent's policy π_t. Then, a decision is made based on a distribution termed $A_t(s, a) = P(a|s)$, where $P(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Thus, the policy is probabilistic. The goal for the MDP is to find an optimal policy of acting within the state-space to gain the highest accumulated rewards under the policy $\pi(s)$ quantified by the discounted sums of estimated rewards. However, the process can be infinite; therefore, the accumulated rewards must be defined as infinite diverging series. Due to the properties of an infinite diverging series, we can express the accumulated rewards G_t as follows:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{5.8}$$

To define the value of being in a state we can use the state-value function, which determines the value of being in a certain state within the state-space. We can write this as the following equation:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = S] \tag{5.9}$$

The other metric is the mapping between the action and the value, hence the value of conducting an action regarding expected rewards. The function measures the estimated accumulated rewards from the current state and forwards, using the expression of accumulated rewards G_t as mentioned above, and define :

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = S, A_t = a] \tag{5.10}$$

Architectures

RL algorithms are often categorised into model-free and model-based. We may think of model-free policies as a fast-thinking system, learning reflexes through environmental experience. In contrast, model-based methods are like a slow-thinking systems, using analysis and planning for the future based on a learned model of the environment. Within the model-free category, we find value-based methods and policy approximations, as shown in figure 5.5. Value-based methods learn the value of actions and use this information to decide the best action in a particular state. For example, the Q-learning algorithm is a well-established value-based method that relies entirely on the quality of the action in determining the optimal policy. However, value-based policies can be prone to exploiting known valuable states [35]. In contrast, policy approximation methods, i.e. policy-based or policy gradient algorithms, learn a policy that selects actions without consulting a value function. Although a value function may still be used in training the policy, policy approximations are not directly dependent on the value of actions in a particular state. When model-free methods do not have any value estimate of either action or state, they are called agent-only methods, relying entirely on the agent's experience of being in a state. By implementing a critical function that estimates the value of a state, the actor-critic method can assess the value function during training to determine the best actions. In the actor-critic method, a *critic* NN learns to predict the values of actions, while the *actor* NN learns to compute the best actions.

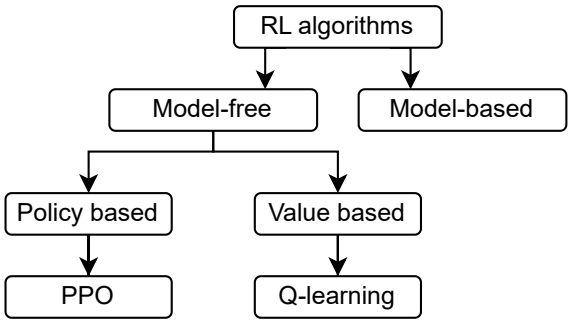


Figure 5.5: Reinforcement algorithms

Policy Optimisation

We use the same definition as previously and say that a policy is a NN with weights as parameters. We follow [35] and denote the parameters as $\theta \in R$: $(a|s, \theta) = Pr\{At = a|St = s, \theta_t = \theta\}$ for the probability that action a is taken at time t given that the environment is in state s at time t with parameter θ. These methods use a performance measure $J(\theta)$ that measures the performance of the policy with respect to the policy parameters. Also, the methods seek to maximise the performance and, therefore, update their weight based on the gradient ascent in $J(\theta)$ and denote the following equation [35, chap 13]:

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{5.11}$$

Where $\alpha\widehat{\nabla J(\theta_t)}$ is an estimate and indicates the direction to adjust the weights θ in order to maximise performance. The term α is the rate and scales the adjustment and is often a hyperparamater for the optimiser used in the algorithms. The above is a general method that is used for policy gradient methods.

Common Hyperaparamaters

Opposed to parameters that are changed during training, hyperparameters are set before the training starts. The value of the hyperparamaters are coefficients which may elevate training speed and the end performance of the trained policy, i.e the parameters in the NN.

The learning rate can be scheduled or fixed throughout the training period. Learning rates for the Adaptive Moment Estimation (Adam), which is used in this thesis, will affect how much of the change in the adjustment should be considered when adjusting the weights in the network.

The architecture of the NN is of great importance and can significantly impact performance and training time. Often the NN is a 2D grid with layers and units. The layer defines how many columns the NN has, whereas units are the number of rows in each layer.

Batch sizes are how many samples are placed in the trajectory before conducting gradient a/de-scend.

5.1.5 Proximal Policy Optimisation

The Proximal Policy Optimization (PPO) is a model-free on-policy gradient method that learns to take action based on experienced states during training. By implementing two policies where $\pi_\theta(a_t|s_t)$ is the current policy and a second one that we use to collect samples $\pi_{\theta k}(a_t|s_t)$, we can evaluate which of these is performing best, also known as the *surrogate* objective [29].

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t)} \tag{5.12}$$

The PPO objective function is defined as follows [29]:

$$L^{CLIPP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{5.13}$$

Where ϵ is the clipping factor, $\hat{A}_t = J(\theta) - V(s)$, $V(s)$ is the estimate of the future discounted rewards provided by the critic network, and $\mathbb{E}_t[...]$ implies the empirical average over a finite batch of samples, which means that the policy parameters are updated based on a stochastic gradient ascend over a trajectory of action and rewards.

5.2 Reinforcement Learning Controller implementation

In this section, all the aspects regarding the RL implementation are documented. First, a general overview of the system architecture is presented before explaining some essential aspects regarding the rewards function and observation space. The system is semi-object-oriented, meaning that several files with Classes hold the functionality for training, training environment and testing.

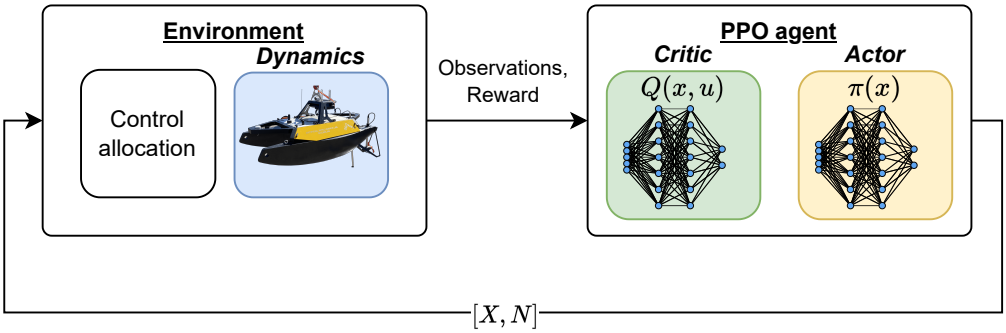


Figure 5.6: Schematic overview of the implementation

5.2.1 Training Environment

The environment is one of the critical elements for successful training and deriving a working policy. As illustrated in figure 5.6, the environment provides the necessary

observations for the Agent to compute the control inputs to achieve the controller objective. In this thesis, there is implemented a Gymnasium Farma environment [10] in the Stable Baseline 3 framework. Following a standardised structure implementing the methods *step, reset, render, info* that interface with the Gym class, one gets a solid simulation framework that can train various RL algorithms.The finalised training workflow is illustrated in 5.7.

Step(action)
The step method is executed at each time step and receives the action from the Agent as input. The Agent provides a normalised value between -1 and 1 to the method. Then the predictions from the Agent are scaled relative to the force boundaries in surge X and yaw N before being converted to propeller rpm through the control allocation method. Then, the new state of the position $\boldsymbol{\eta}$ and the velocities $\boldsymbol{\nu}$ are calculated using the Otter model, described in section 3.5 before the termination evaluation is done. The algorithm should terminate the episode if the Otter has been within the threshold region surrounding the target for 60 seconds or if the episode time limit is reached. Lastly, the rewards and penalties are calculated based on the new state of the vessel and the termination state. The method returns the observation, rewards, and a boolean *done* signal.

Reset()
The reset methods implementation is used to initialise and reset the environment. The initialisation is done when the training starts, and the reset is used between episodes. The starting position of the vessel is always standing still in origo, while the position of the target can randomly appear within a defined radius of the vessel. If the target is moving, the target's speed is defined randomly within a specified range defined in the config file.

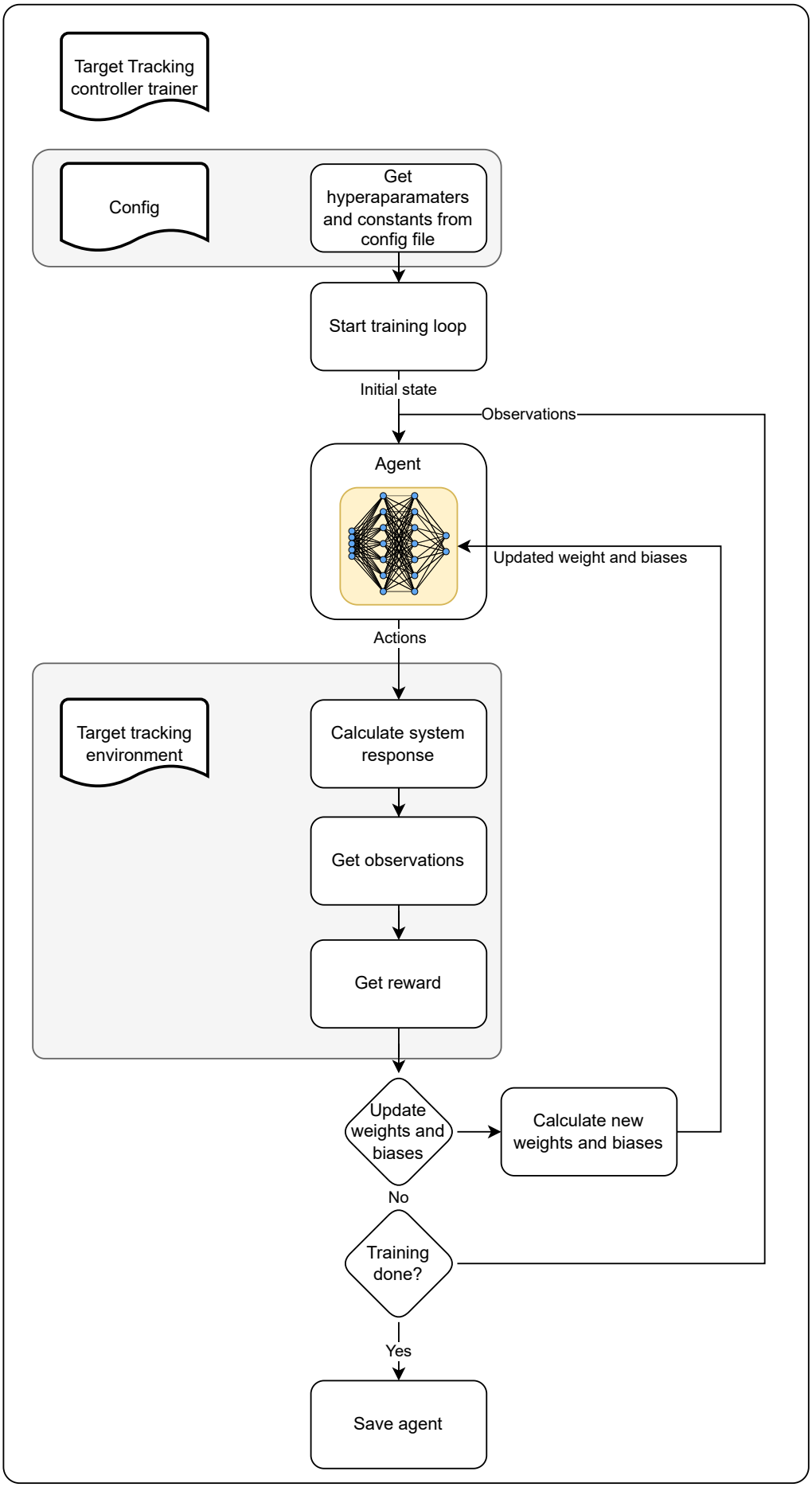


Figure 5.7: Training process workflow

5.2.2 Agent and Hyperparamaters

The Agent holds the policy that makes the regression prediction on which controller action that is optimal. The Agent consists of two neural networks and *actor* and a *critic*. In this implementation, both networks are of the same architecture and have the same number of layers and units. In table 5.1 the architecture and hyperparamaters are listed.

	Description	**Values**
Actor and Critic NN	Layers	[2, 3]
	Nodes in layer	[32, 64, 128, 256, 512]
	Activation function hidden layers	ReLu
	Regression layer	Linear
Hyperparameters	Rollout buffer size	2048
	Batch Size	64
	Discount factor γ	0.99
	Clip range ϵ	0.2
	Optimiser	Adam
	Num epochs during optimisation	10
	Initial learning rate α_0	[0.003 - 0.00003]

Table 5.1: PPO Agent network architecture and hyperparameters

The learning rate is the scale of each gradient descent update step and corresponds to the distance of movement, the adjustment of the weights, and the biases taken during the network parameter optimisation. This implementation uses a scheduled learning rate defined by the function 5.14.

$$f_{LR}(\alpha_0, p) = \alpha_0 * \exp(-(1-p)) \tag{5.14}$$

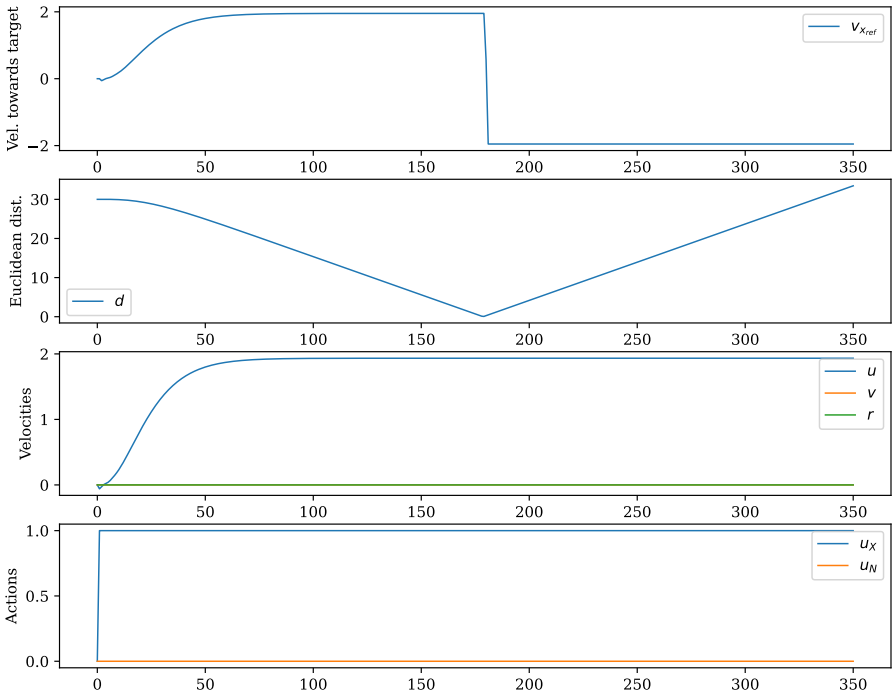
where α_0 is the initial learning rate value, and p is the progress indicated with a number between 0-1.

5.2.3 Reward Function

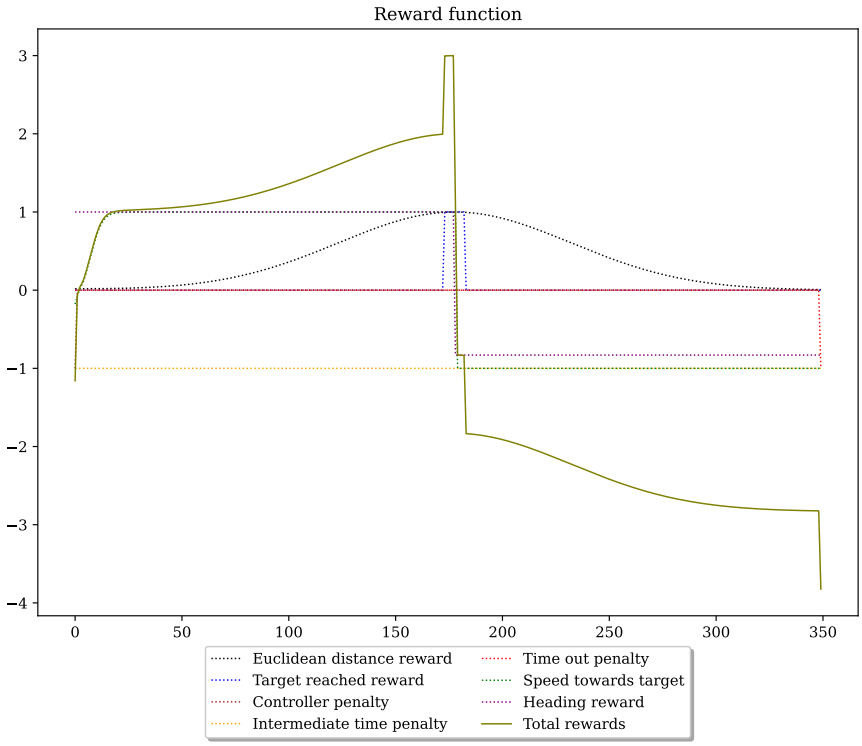
One of the more challenging parts of the RL paradigm is to implement a reward function that allows for both exploration and rewards sufficiently for the wanted behaviour from the Agent. The training might never converge to an optimum if the reward is too sparse in a stochastic environment. On the other hand, if the reward function is too specific, it can lead to exploiting and converging to a sub-optimal behaviour with the perception that it is reaching the intended objective.

During the training process, various reward functions were implemented to get the intended behaviour during training. The reward function was mostly implemented as a continuous reward or penalty element. Then, each reward function element was multiplied by a tuning coefficient and summed to give the total reward per episode. The tuning coefficient was represented as c_1,...c_7 in the *config.yaml* file for easy tuning and to study the effect of each cost function element on training speed and general performance. Figures 5.8 shows an example of a reward function

constellation where the Otter is moving directly through the target, as can be seen in the *eucledian distance, Action*, and *Velocity*. The reward function in the example has coefficients $c_1, ...c_7 = 1$.



(a) State of the Otter



(b) Reward function

Figure 5.8: Example of a reward function constellation

Euclidean Distance Reward

The first reward function element is the euclidean distance between the target and the vessel as defined in equation 5.15. The distance is mapped to a Gaussian bell curve function that gives a continuous reward function which steadily increases as the distance between the vessel and the target is reduced. As illustrated in figure 5.8 the black dashed line is showing the reward contribution as a function of the distance between the target and the vessel. We define the euclidean distance as shown in equation 5.15.

$$d(\eta, x_{ref}) = \|\boldsymbol{\eta_{x,y}} - \boldsymbol{x_{ref}}\| \tag{5.15}$$

Where the $\boldsymbol{\eta_{x,y}}$ is the x and y position of the vessel, and $\boldsymbol{x_{ref}}$ is the position of the target at time t.
Then we define the reward function element r_1 showed in equation 5.16

$$r_1(d) = a \exp -(\frac{d-b)^2}{\sigma^2}) \tag{5.16}$$

where a is the height of the amplitude of the bell curve. The standard deviation σ is the with of the bell curve, and in this implementation, it is set to be half of the *target spawn region*.

Target Reached Reward

Target reached reward is an extra reward given to the Agent for being within a region R of the target.

$$r_2(d) = \begin{cases} 1 & \text{if } d < R \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

Speed Towards Target Reward

Another way of rewarding a favourable state of the vessel is to use the speed towards the target. The function is designed to quickly increase the reward for a small amount of speed towards the target. This function element intends to reward the speed in reaching the target quickly, but it may also provide enough information to derive a working policy on its own. The function is capped on a velocity of ± 1 to ensure a responsible speed of the vessel.

$$r_3(\dot{d}) = \tanh(3\dot{d}) \tag{5.18}$$

where $\dot{d}$ is the speed towards the target in the NED frame.

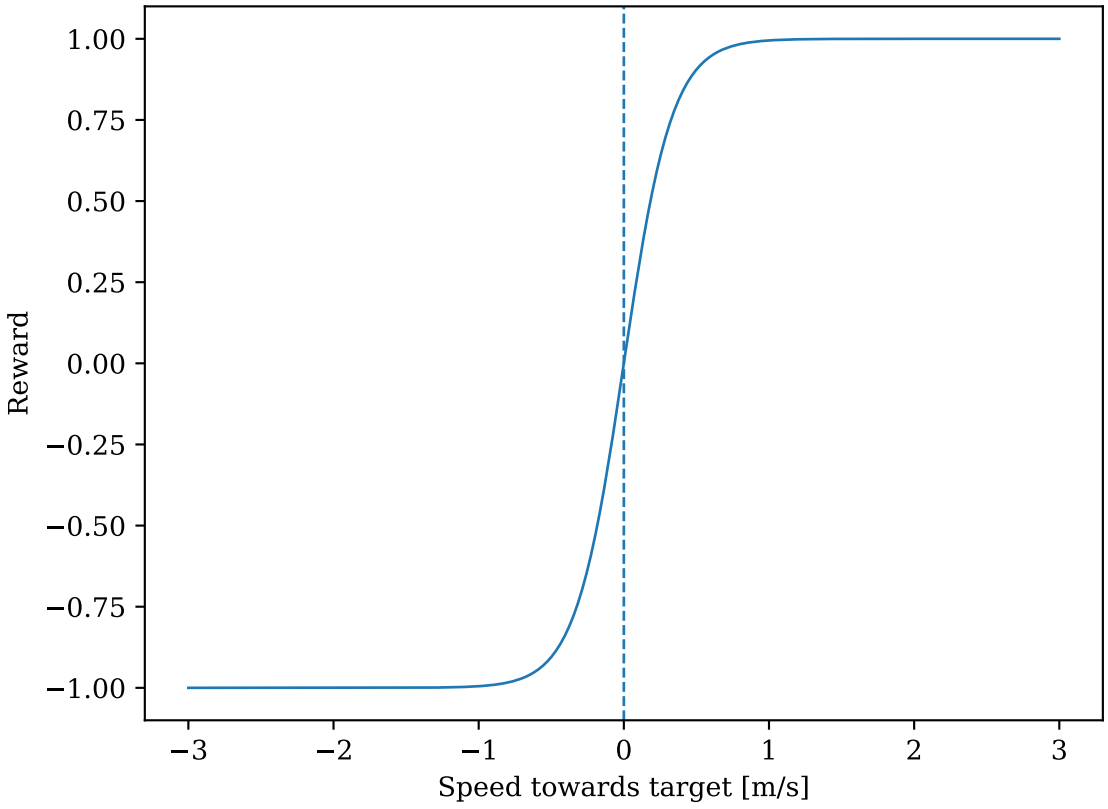


Figure 5.9: Example of speed towards target reward

Heading Reward

None of the previous reward function elements is explicitly considering the orientation of the vessel whilst it is pursuing the target. Thus, by implementing a heading reward expression, we can reward the Agent for controlling the Otter in the desired heading angle. To deduce the angle between the desired angle and the current heading, the *smallest signed angle* is used to map the angle between 0 and 2π

$$\tilde{\psi}(\phi) = modulo(\phi + \pi, 2\pi) - \pi \tag{5.19}$$

where $\phi = \psi_d - \psi$, ψ_d is the desired heading and ψ is the actual heading of the vessel.

Then we use the smallest signed angle in the heading reward function as shown in equation 5.20.

$$r_4(\phi) = -1 + 2\exp(\frac{\tilde{\psi}(\phi)}{2^2}) \tag{5.20}$$

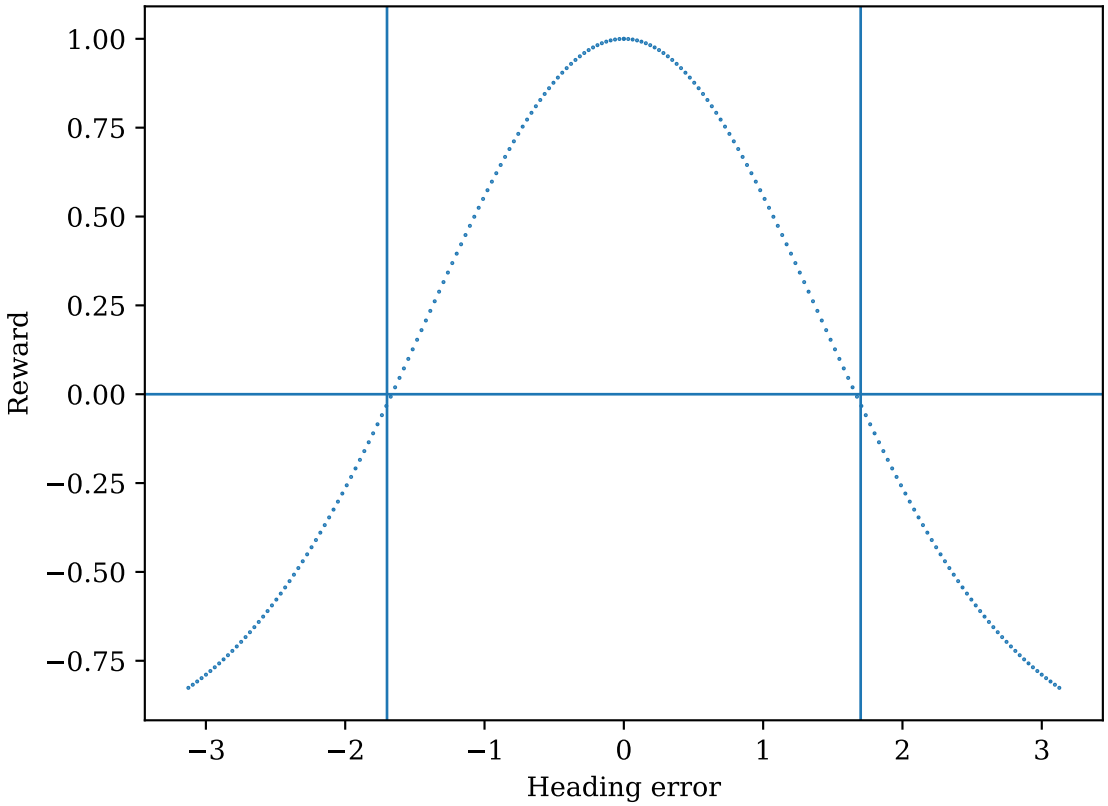


Figure 5.10: Example on heading reward

Controller Signal Penalty

Without consideration, a control signal is regarded as a variable that can be altered promptly. However, actuators exhibit dynamic behaviour in practice, which may result in a temporal lag between the input control signal and the corresponding output response. Moreover, the mechanical components of the effectors, e.g. propellers, are often unable to withstand rapid changes in acceleration over extended periods. Consequently, imposing a penalty during training becomes necessary to prevent oscillatory behaviour and obtain system stability. In this implementation, it is the absolute value of the changes from one time step to the next for the entire control vector that is considered, as described in equation 5.21.

$$p_1 = -\|\tau_t - \tau_{t-1}\| \tag{5.21}$$

where $\tau = [X, N]^\intercal$.

Intermediate Time Penalty

The intention of this penalty is to incentives the Agent to reach a favourable state and gain higher rewards quickly. Oppose a situation with no intermediate time penalty; fast actions make no difference.

$$p_2 = -1 \tag{5.22}$$

This makes the tuning parameter c_6 that defines the scale of the penalty and for most of this implementation, c_6 is set to 1 or 0.1 during training.

Time Out Penalty

The time-out penalty is a penalty that is given if the vessel is not within the target threshold region surrounding the target.

$$p_3 = \begin{cases} -1 & \text{if t} \geq \text{T} \\ 0 & \text{otherwise} \end{cases} \tag{5.23}$$

Total Episode Reward

After each reward and penalty is computed, the reward function is multiplied by the coefficient from the configuration file and summed together:

$$r_{total} = \sum_{i=1}^{T} c_1 r_1 + c_2 r_2 + c_3 r_3 + c_4 r_4 + c_5 p_1 + c_6 p_2 + c_7 p_3 \tag{5.24}$$

5.2.4 Observations

Observations are the input to the Agent during training and inference, and they must contain sufficient data for the Agent to deduce the following controller action. The observation space is a single column 18x1 vector including the elements provided in table 5.2.

Term	Description
$\boldsymbol{\eta}$	$[x, y, \psi]^\intercal$ position of the vessel
$\boldsymbol{\nu}$	$[u, v, r]^\intercal$ The relative velocities of the vessel
$\boldsymbol{x_{ref}}$	$[x, y]^\intercal$ The target position
$\boldsymbol{\Delta\eta}$	$[\Delta x, \Delta y]^\intercal$ Is the delta in x and y
d	The euclidean distance between the vessel and the target
$\Delta\boldsymbol{\tau}$	$\|\boldsymbol{\tau_k} - \boldsymbol{\tau_{k+1}}\|^\intercal$ Is the change in controller action
$\boldsymbol{\tau}$	The actions in $[X, N]^\intercal$
$v_{x_{ref}}$	Is the velocity towards the target
$\tilde{\psi}$	Smallest signed angle between the desired heading ψ_d and ψ
t_r	t/T is the ratio between proceeded time t and the episode time T
R	Radius surrounding the target, i.e. threshold if the optical relay

Table 5.2: Observations for the Agent

The observations are stacked in a set of eight, which means the Agent gets the latest eight observations to conduct its controller actions on. This might be redundant since the observations include temporal observations, e.g. velocity, giving the Agent direct access to that data. This means that the Agent does not have to derive the relationship between the time and the position to derive that data specifically, reducing the computational demand. However, as mentioned in section 5.1, there is an example where the Agents are getting several time steps of observations that increases the performance of the Agent's prediction of actions. Lastly, the environment was implemented with a normalisation with a moving average which adjusted the normalisation metrics as the Agent explores new parts of the observation space.

5.3 Reinforcement Learning Training

This section explains the training process and the intermediate adjustment throughout the process.

5.3.1 Training Procedure

The goal for the Agent is to have the same behaviour as the MPC. That is, reducing the distance between the vessel and a target, regardless if the target is far away or very close or if the target is moving or standing still. Furthermore, the Agent must be able to derive feasible controller actions in a generalised manner with any scalar values of the distance or other observations. In other words, the Agent must be able to conduct feasible controller actions for observations that are both small and large.

An extensive range of hyperparameter configurations, observations, and reward functions have been exhaustively examined in pursuing a policy that satisfies the aforementioned criteria. The meticulous process of trial and error is a consequence of the stochastic architecture of the environment. There are several hyperparameters to decide for the RL algorithm to derive a well-working policy. A hyper-parameter benchmarking case was created to work more effectively in the hyper-parameter elaboration. The benchmark case was an environment where all episodes in the training session had a fixed target, making it easier for the Agent to derive a feasible policy quickly. Several training sessions were started simultaneously, and after the sessions were finished, the result was evaluated by looking at which of the policies had converged. After that, the changes were made to the hyperparameters and a new session with many policies was run on different hyperparameter configurations was conducted. The same benchmark approach was used for the tuning of the reward function. In figure 5.11, we see a benchmark case where the different NN sizes were investigated. The conclusion was to proceed with a NN with two layers and 128 units each due to the fast convergence and high reward after 800 episodes.

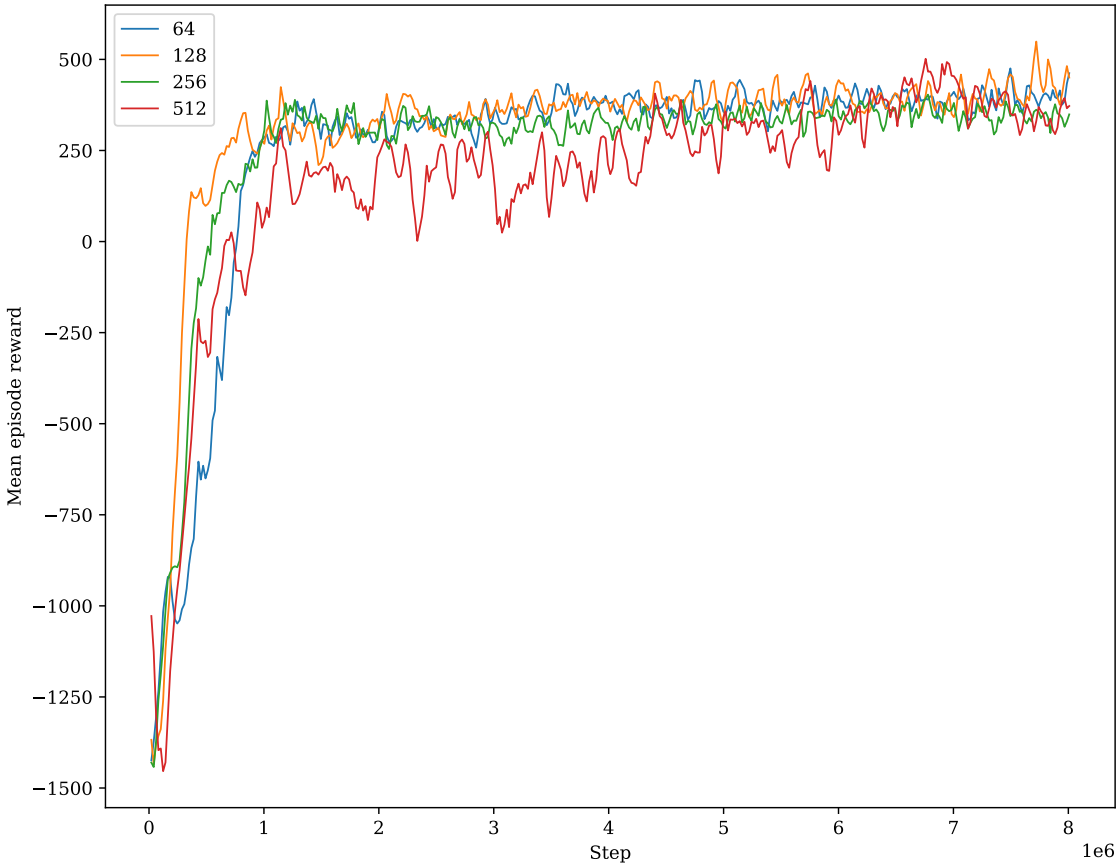


Figure 5.11: Example from the work when deriving the size of Critic and Actor NN

5.3.2 Agent Training

This section describes the training that was conducted to derive the policies for the three Agents used in the performance test in section 5.4.
Several different policies for the controller Agent were trained with different reward functions and environment configurations. To illustrate how hyperparameters and reward functions affect training convergence, results from three policies will be presented.

Training Setup

The environment parameters were the same for all policies and are presented in table 5.3.

Paramater	**Value**
Simulation sample time	0.1
u_{min}	X = -116 N , N = -73 Nm
u_{max}	X = 150 N , N = 73 Nm
RL control sampling	0.1
Episode length	1000
Target spawn region	50m
Target cycle	100
Target confident region	1
Target velocity	[0-0.8]m/s
Number of environments	10
Number of stacked observations	8

Table 5.3: Environment parameters used during training

Simulation sample time is used when computing the dynamics. The u_{min} and u_{max} are the boundaries for the thrust forces for the Otter, since the Agent is providing a number between -1 and 1, the boundaries are used to denormalize the actions before sending them to the control allocations. *RL controller sampling* is how often the controller computes new controller actions. The *Episode length* is the length of each episode during training. The *target spawn region* is a radius around the Otter that indicates the area in which the target may spawn. The *target cycle* is how many episodes the target spawns at the same spot. *Target confident region* indicates that the Agent is sufficiently close to the target and gains the same reward as long as it is within this region. *Target velocity* defines the range of the target velocity. *Number of environments* is the number of parallel environments the Agent is acting. *Number of stacked observations* is how many previous observations the Agent has access to.

Parameter	**Agent 1**	**Agent 2**	**Agent 3**
Reward function			
c_1 (distance to target)	0	0	1
c_2 (target reached)	1	1	0
c_3 (controller penalty)	0.01	0.1	0.01
c_4 (intermediate time penalty)	0.1	0.1	0.1
c_5 (time out penalty)	0	0	0
c_6 (vel. towards target)	1	1	0
c_7 (heading)	1	1	1
Agent hyperparamaters			
Actor and Critic Architecture	[128,128]	[128,128]	[128,128]
Initial learning rate	0.00003	0.000003	0.0003
Total training steps	80M	30M	80M

Table 5.4: Reward function configuration and Agent hyperparamaters

Training Results

In figure 5.12 evolution of the accumulated rewards pr. episode, throughout the training time, is presented. As shown in the figure, the mean reward is rapidly increasing at the start of the training for all Agents. Then *Agent 1 and 2* starts converging to a maximum level. *Agent 3* shows a decrease in rewards after the initial spike but nearly manages to pick up the lag towards the end without converging fully. *Agent 1* and *Agent 2* are trained for 80M steps, and *Agent 2* is trained for 30M steps.

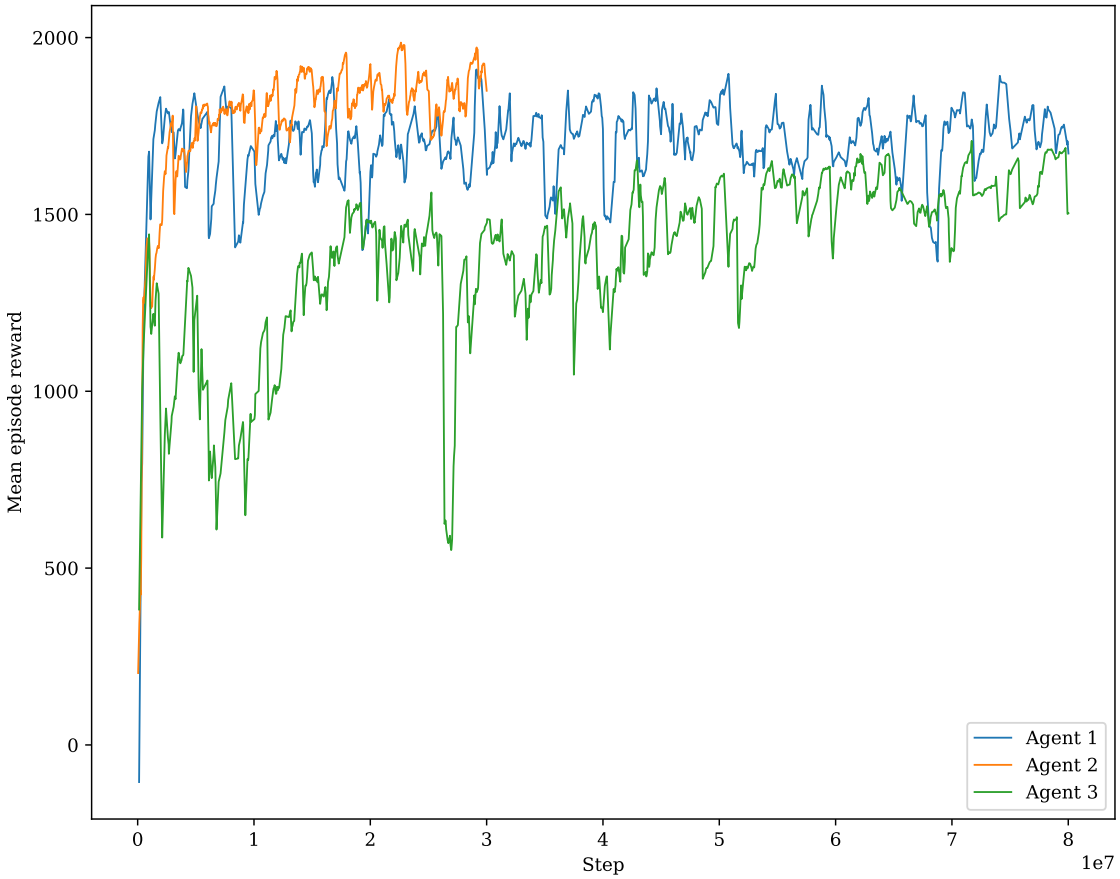


Figure 5.12: Agent policy convergence during training

5.4 Simulated Performance Test

This section presents the simulated controller performance test of the RL controllers. In this context, the RL-controller will be the Agent that *predicts* the controls based on a trained NN, called the *actor policy*. The input to the NN is a vector that has the aforementioned observations for the last eight timesteps. The output is two float values between -1 and 1, representing the normalized desired force in Surge and Yaw.

5.4.1 Setup

The simulation workflow is shown in figure 4.2, except that the controller is RL-controller. The simulations are conducted by utilising the environment used in training, and only the dynamic calculations from *Fossen Vehicle Simulator* are utilised. Two test cases will be presented for each Agent to illustrate key features.

Case 1) is the same as the MPC test case explained in section 4.4.1, with a moving target starting with an offset ([20,20]) from the Otter.

Case 2) is similar to case 1, but the simulation time is set to be 1000 steps, hence the same as the episode length in the training sessions.

5.4.2 Results Case 1

In this section, the results from case 1 are presented. For each Agent, there are plots of the Otter and target movement, the error between the Otter and the target, controller signals computed by the Agent, velocities and a plot of the computation time to predict each action.

In figure 5.13, 5.14, and 5.15 spatial plots of the movement of the Otter and the target is plotted for Agent 1,2 and 3 respectively. All Agents are quickly controlling the Otter towards the target relatively efficiently and intercepting the target's trajectory. As the target is intercepted, the three Agents starts to deviate from each other's controller patterns and have individual manoeuvres to bound the distance towards the target. After closing the distance to the target, Agents 1 and 2 are tracking the target and show a movement that crosses the target trajectory several times. After travelling approximately 100m in the East, Agents 1 and 2 starts circling the target while following the target trajectory. Agent 3 is showing a different pattern and is circling the target from target interception to around 125m East direction, then it closes the distance and follows a smooth path on the target's tail.

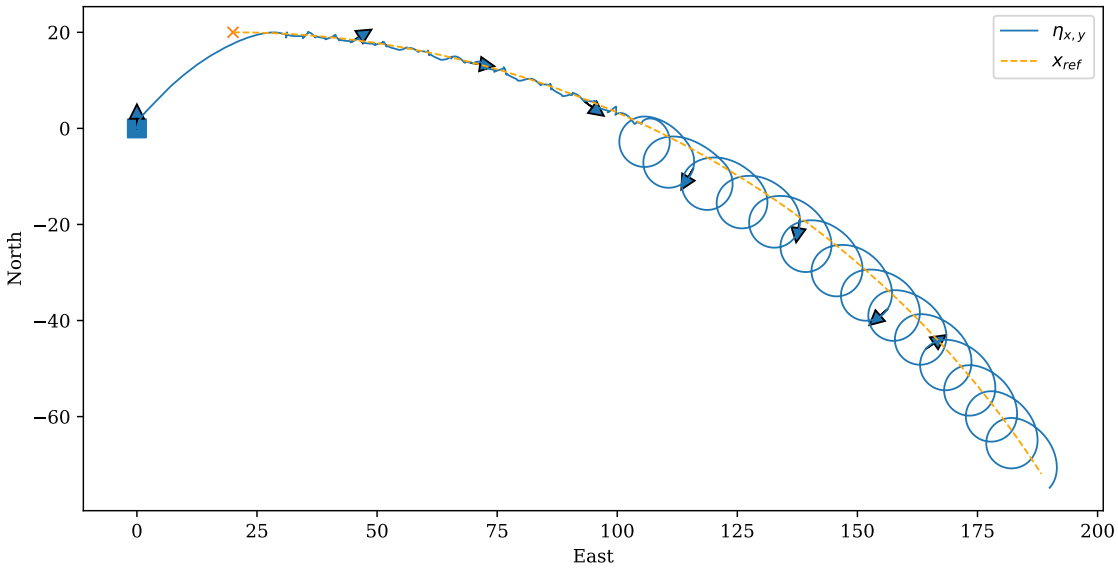


Figure 5.13: Agent 1 performance in NED

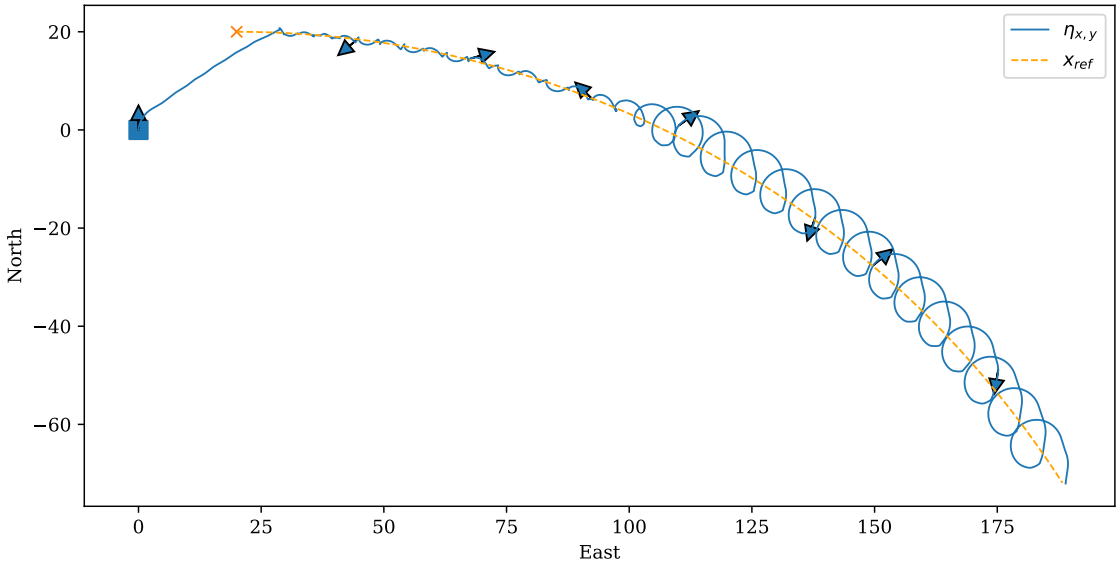


Figure 5.14: Agent 2 performance in NED

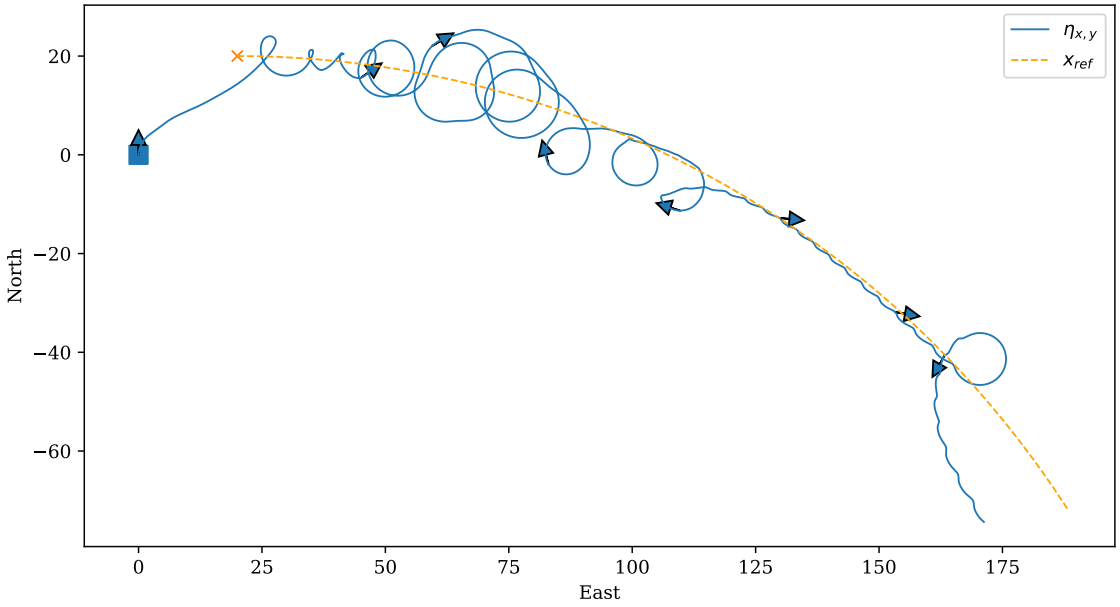


Figure 5.15: Agent 3 performance in NED

Error

In figure 5.16, 5.17 and 5.18 the absolute euclidian error between the target and the Otter is shown. All three Agents quickly reduce the error to a minimum before going unstable, resulting in a fluctuating error throughout the simulation.

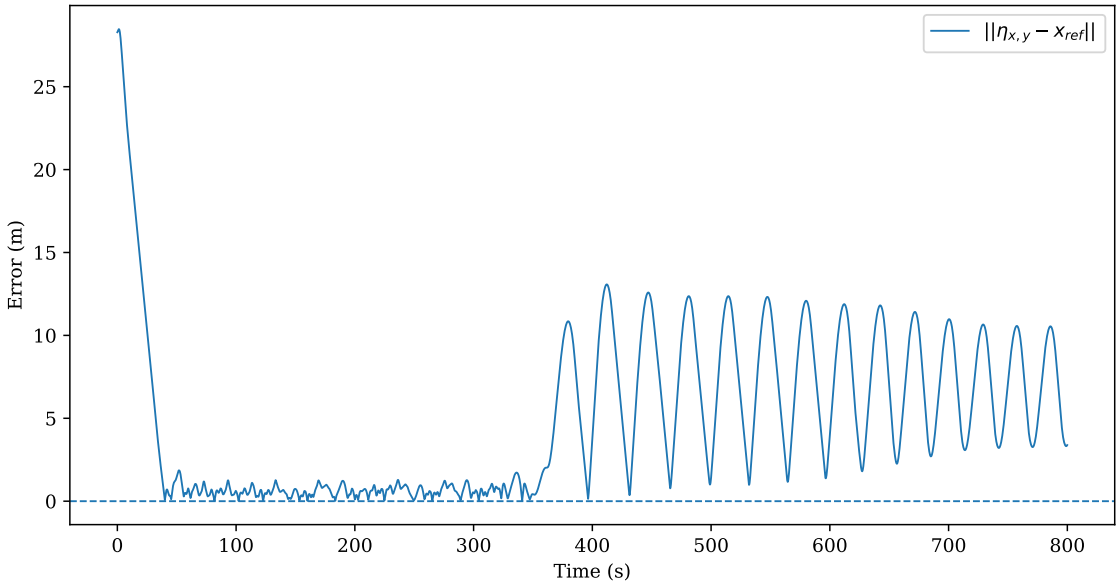


Figure 5.16: Agent 1 performance error

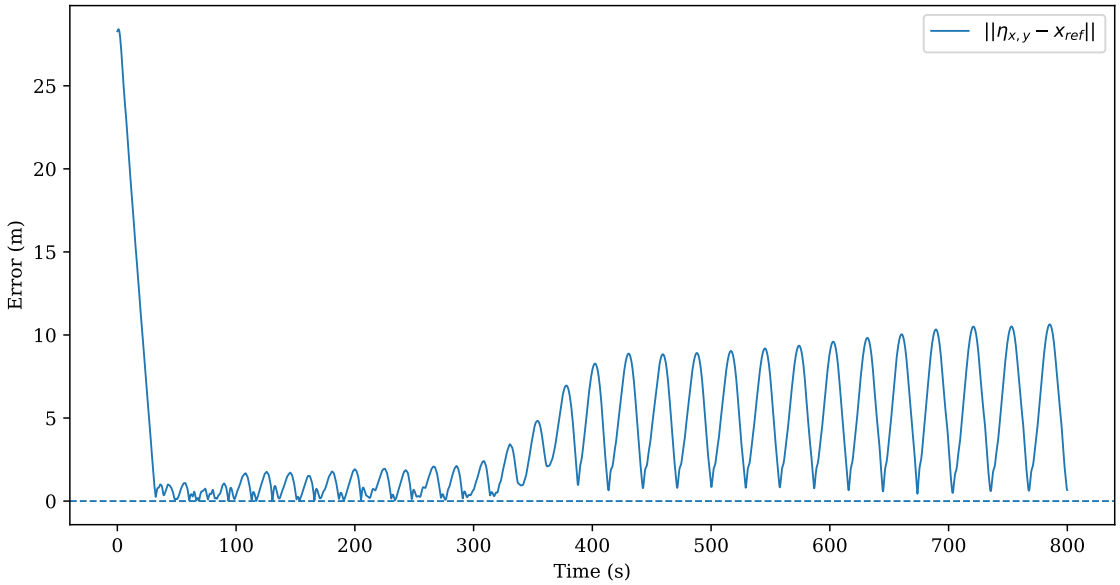


Figure 5.17: Agent 2 performance error

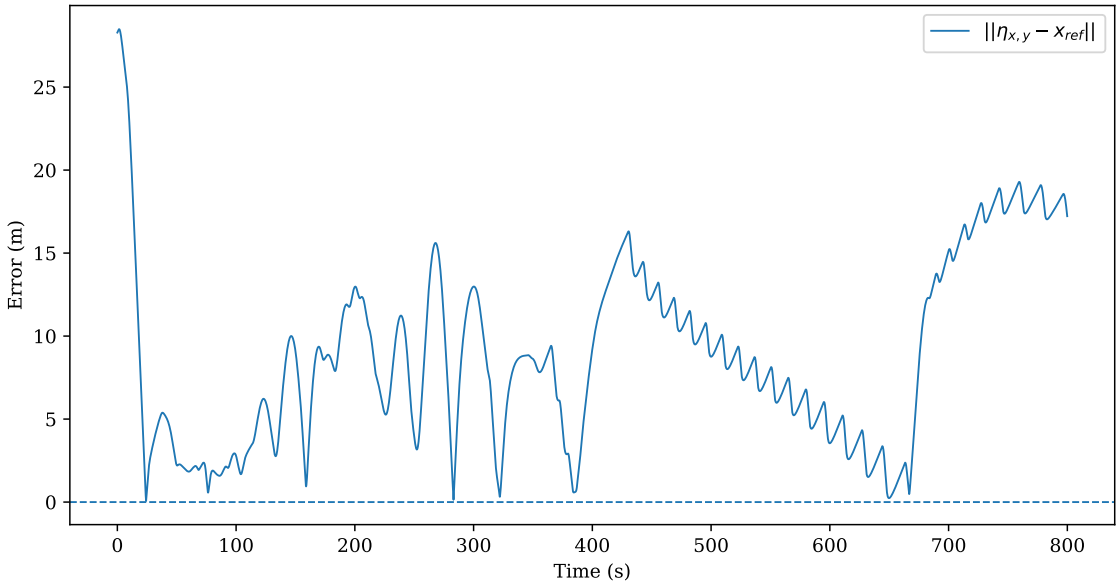


Figure 5.18: Agent 3 performance error

Control Signals

In figure 5.19, 5.20, and 5.21 the Agent control predictions are presented. The controls are normalised between -1 and 1. The controller signals from all three Agents fluctuate highly and move rapidly between -1 and 1 throughout the simulations.

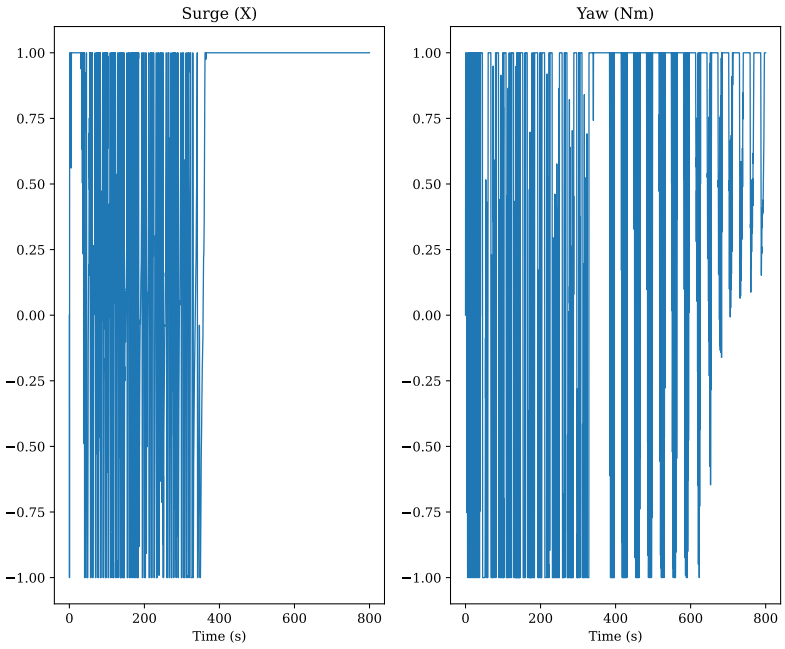


Figure 5.19: Agent 1 Controls

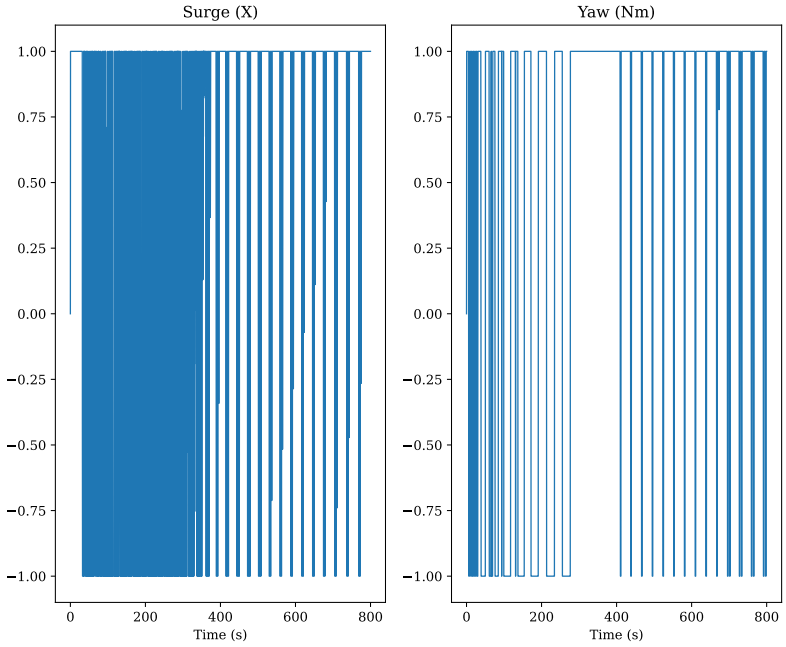


Figure 5.20: Agent 2 controls

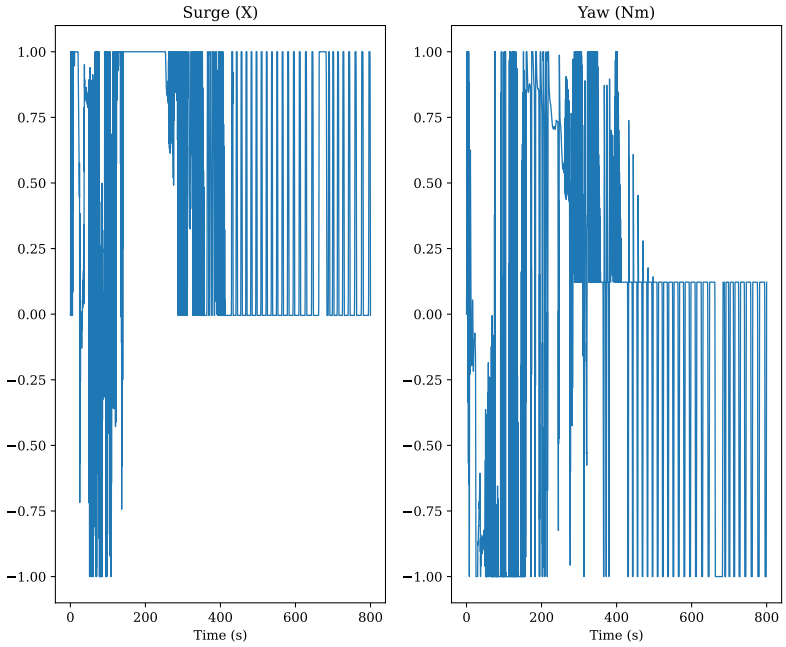


Figure 5.21: Agent 3 controls

Remark. *The figures 5.19,5.20 and 5.21 show normalised forces.*

Velocities

In figure 5.22, 5.23, and 5.24 the velocities for the Otter controlled by the there Agents are shown. As the controls in the previous section indicated, the velocities also fluctuate throughout the simulation. Agent 3 shows somewhat lesser variations in the velocities than Agent 1 and Agent 2.

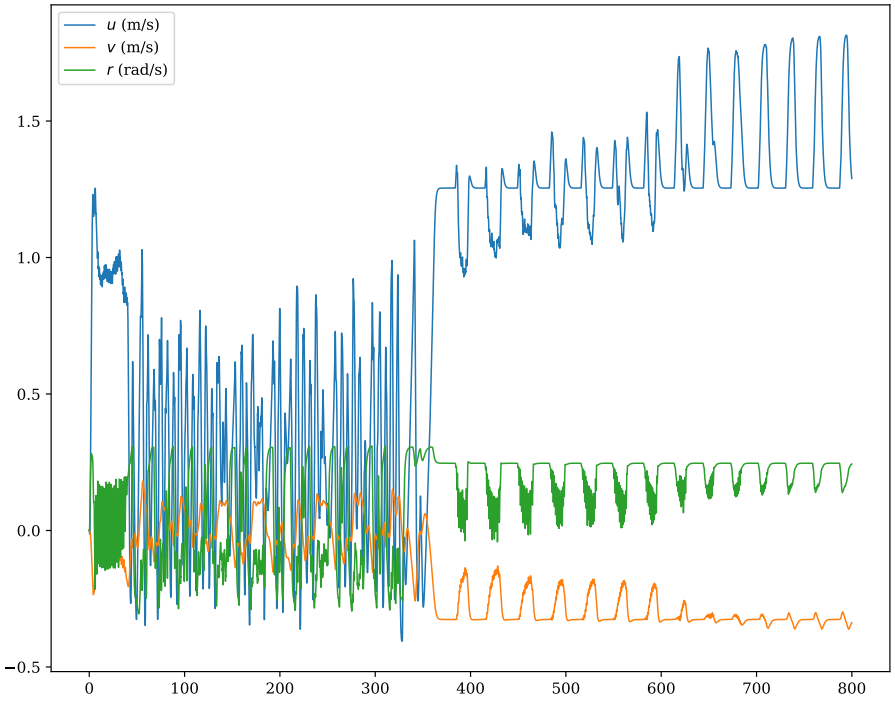


Figure 5.22: Agent 1 velocities

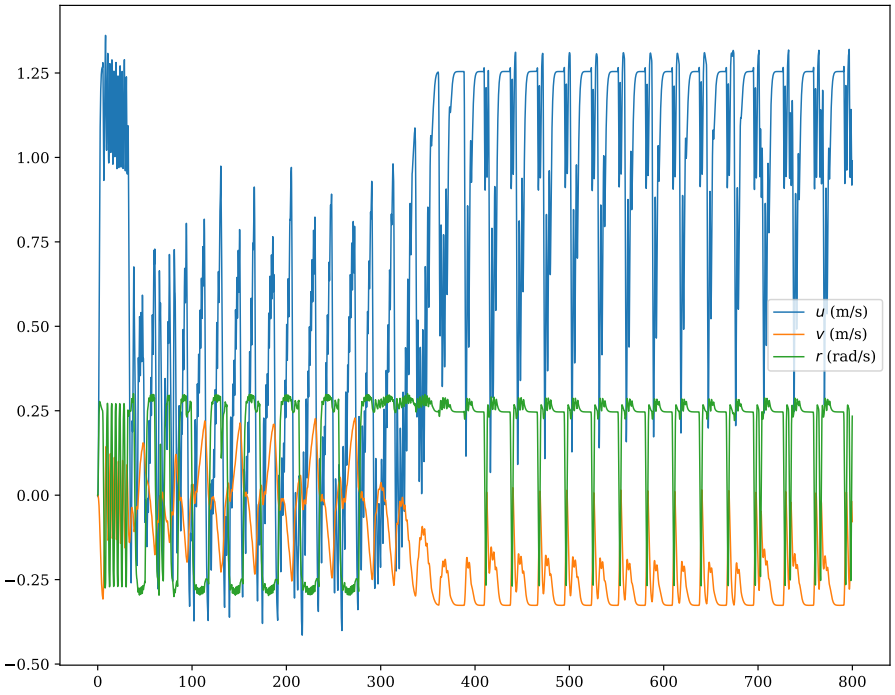


Figure 5.23: Agent 2 velocities

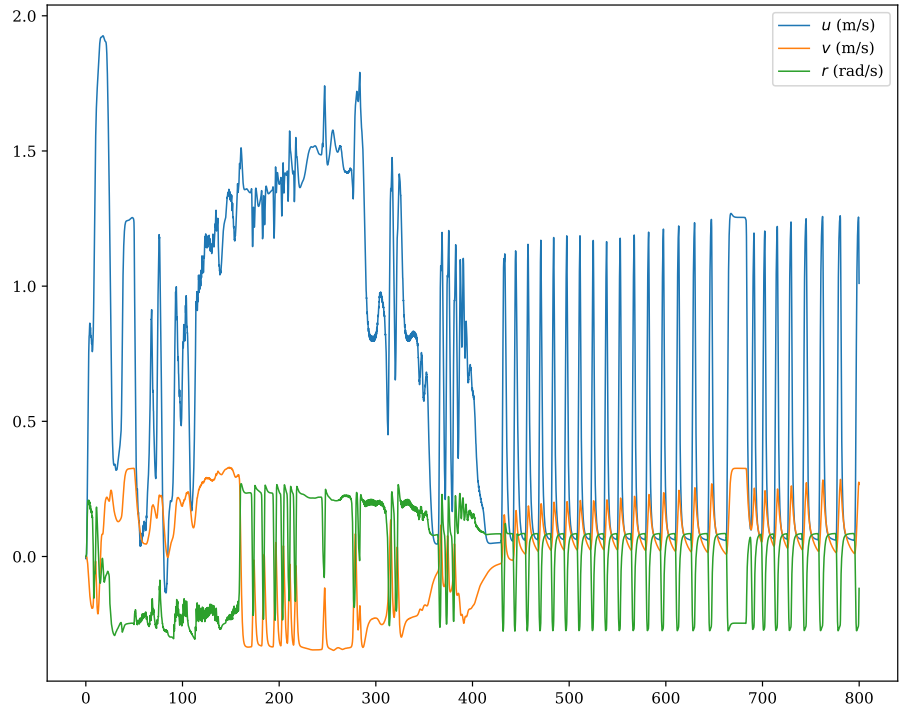
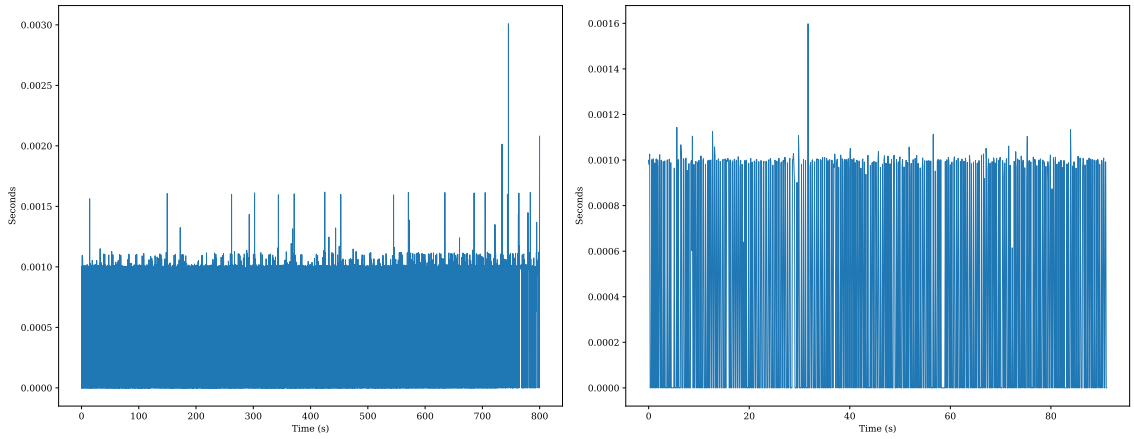


Figure 5.24: Agent 3 velocities

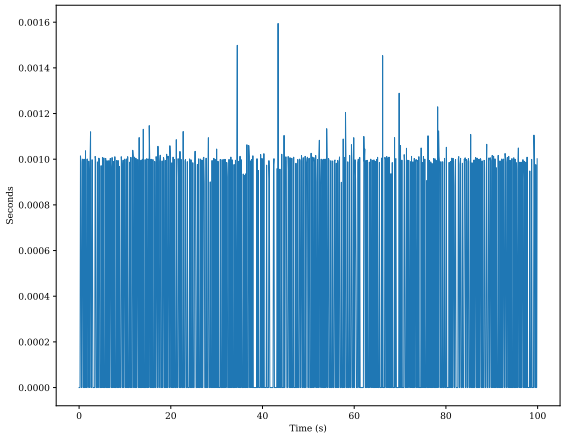
Computation Time

In regards to computation time, figure 5.25a, 5.25b, and 5.25c shows that all there Agents has somewhat the same patters with higher computation time that settles around 0.001 seconds (1kHz). Some spikes to up to 0.005 seconds are also present.



(a) Agent 1

(b) Agent 2

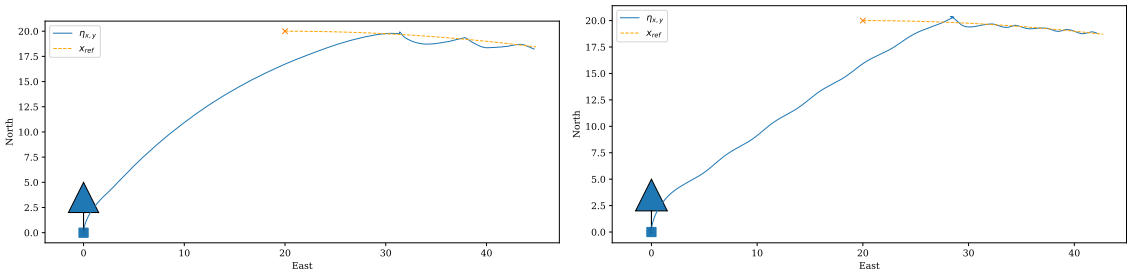


(c) Agent 3

Figure 5.25: Case 1 computation time

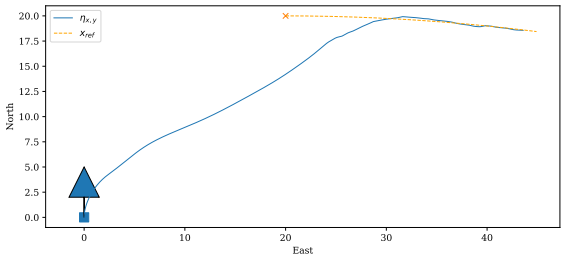
5.4.3 Results Case 2

In this section, a simplified results representation is presented. This section intends to show how the Agents perform when the test simulation length is the same as the episode length. As shown in the following figures, all three Agents efficiently close the distance to the target.



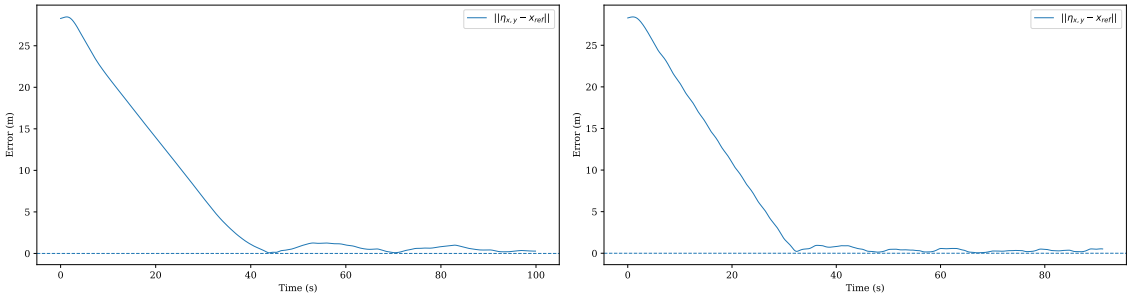
(a) Agent 1

(b) Agent 2



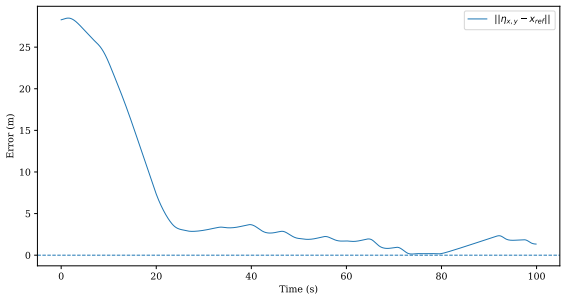
(c) Agent 3

Figure 5.26: Case 2 movement plots in NED



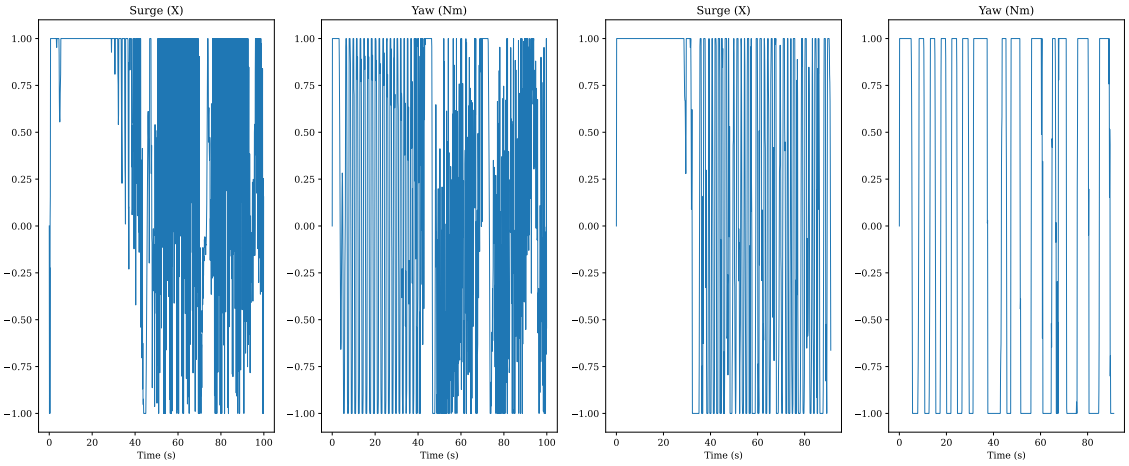
(a) Agent 1

(b) Agent 2



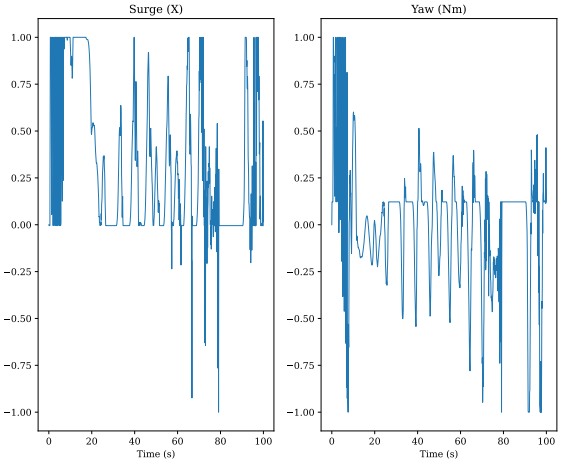
(c) Agent 3

Figure 5.27: Case 2 error



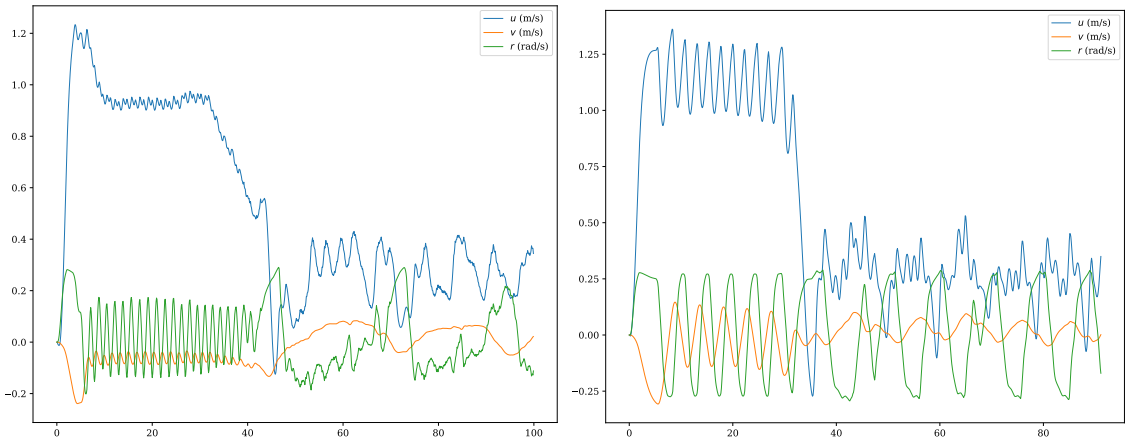
(a) Agent 1

(b) Agent 2



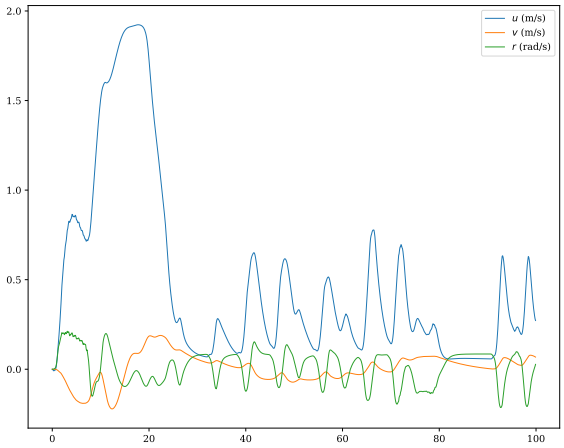
(c) Agent 3

Figure 5.28: Case 2 controls



(a) Agent 1

(b) Agent 2



(c) Agent 3

Figure 5.29: Case 2 velocities

5.5 Discussion

RL has numerous parameters that will affect the performance of the resulting Agents. This section will discuss the training of the Agents and their tested performance in simulations. As the Agents NN are black boxes, meaning they are not analytically explainable, the three Agents' results are presented to derive the differences between their behaviours.

5.5.1 Training

Initially, the training approach taken was relatively naive. The idea was for the Agent to find a feasible policy to close the euclidean distance between itself and the target. Since the distance between the target and the vehicle is fixed at each time step when an Agent calculates its controller action, the idea was that this approach would work for a moving target as it does for a fixed target. The approach was to place the Agent in an environment where the target spawned randomly within a set radius for each episode. Through trial and error, the naive strategy resulted in poorly working policies, giving the Agent undesirable behaviour under a sub-

optimal policy, e.g. going in circles towards the target. Additionally, due to the long convergence time, the training strategy could have been more efficient in deriving the hyperparameters for viable hyperparameters for successful policy training.

After using the hyperparameter benchmarking and when a feasible set was found, the Agent was trained by utilising the naive approach described in the first paragraph. In the simulation, the Otter spawned in origo with a fixed orientation, whereas the target randomly spawned within a limited region from the vessel. The region was fixed throughout the training session. However, this proved unfit when testing the Agent on a moving target. Therefore, a moving target was implemented to train a policy for tracking a moving target.

Regarding the training environment, this has been changed throughout the implementation to adapt to the action taken to improve the training. In this implementation, two approaches were tested regarding preprocessing of observations before the Agent computed the controller action. (i) The first approach had no manipulations of the observations before sending them to the Agent. This approach might be sufficient for a limited observation space, where the observations are within a finite scale. Since the observations are not normalised, the scale can, in theory, be from $[-\infty, \infty]$, an observation space impossible for the Agent to explore fully. (ii) In the second approach, the observations were normalised. The implemented normalisation was a moving average that adjusts its clipping range as the Agent observes new areas in the observation space. This means the Agent is normalising the values, making it more general.

The training is implemented as an episodic task, which means that the episode ends when the target is reached for 60 seconds, but the control task is, in reality, continuous. Thus, the Otter and the target should not necessarily return to their initial state when ending an episode. Alternatively, episodes could be truncated, not terminated, if the time limit was reached. In future work, a similar method should be implemented, but one with a continuous control task fashion. For example, the Agent starts with the health/energy consumed during the task (equivalent to the controller actions). This means the task can continue in infinity, and the Agents know their *energy*, rather than the episode's time.

Figure 5.12 plots the mean episode reward during training. The figure shows that the three Agents rapidly increased their accumulated rewards at the start of the training. Agent 1, with a learning rate of 3e-5, is beginning to converge around 600k steps. Agent 2, with a learning rate of 3e-6, is beginning to converge around 2.5M steps. Agent 3, with a learning rate of 3e-4, shows a sudden dip in performance after the initial increase before slowly climbing towards the end, never converging fully. Agent 3 has the largest learning rate, and as explained in section 5.1.4, the learning rate defines how large steps are taken in the direction of the gradient during optimisation. Agent 3 might suffer from a too-large learning rate because it oscillates heavily during training and takes a long time to converge. Even after 80M steps, it has not had the solid pattern of convergence as the two other Agents. Since there are several variable changes between the Agents, it is problematic to derive any hard conclusion. However, it is conspicuous that the Agent's training follows a pattern

that is coherent with the descriptions of learning rates that affect the training.

5.5.2 Controller Performance

The results section included plots from two simulation cases, demonstrating that the RL training strategy was fairly effective in training Agents to control the Otter and track a target within the limits of 10 meters. Agent 1 and Agent 2 manage to track the target adequately at the start of the simulations in Case 1). On the other hand, both Agents had rapid controller signals oscillating between maximum and minimum throughout the simulation, creating aggressive accelerations of the Otter.

As explained in the implementation section 5.2, numerous parameter is used in the environment to describe how the environment is set up and how the Agent precepts the environment. In the result section 5.4.2, Agents 1 and 2 show the same pattern and have relatively stable controls initially. However, when reaching around 100m in the East direction, they start computing unstable controller signals resulting in the Otter moving in circles. Despite the circular movement, the Otter still follows the target trajectory. As mentioned in section 5.2, normalisation was implemented to reduce the effect of different ranges in the observations. Take Agent 1, for example, trained with an episode length of 1000 steps. In figure 5.13, a simulation run for 8000 steps, we can observe that the predicted actions are unsatisfying towards the second half of the simulation, indicating that the Agent is not making sense of the observations in this part. On the other hand, if we look at Case 2, where the simulation is 1000 steps, for Agent 1 in figure 5.26a, the Agent manages to control the Otter closer to the target. To clarify, the only difference in the two cases is that the simulation has gone from 8000 steps to 1000 steps and, as explained in 5.2, the Agent knows the total time steps. This behaviour indicates that Agent 1 is sensitive to observations surpassing the experienced values during training. A reason for this may be the scales of the values in the observations. During training, the Agent always started in origo. With its limited speed, it has never experienced any observation values greater than the distance the Agent has been moving away from its starting point, which leads to a limited experience of the whole observation space. As mentioned, moving average normalisation was implemented to mitigate this issue. However, all values surpassing the experienced values will be reduced to the clipping range, and the ratio between them is not necessarily obeyed. To solve this, we could implement a new clipping normalisation that truncates the observation values between -1 and 1, where the boundary is the limitation in the NED frame. Another mitigation may be feature engineering which only includes features that do not vary in orders of magnitude. Such features, in this case, are the velocities, velocity towards the target and smallest signed angle. The Ottes thrust force limits the velocity, and an AUGs will have limited speed. Lastly, the brute force method would be to randomise the initial position of the Otter and the target so that the Agent could experience a large part of the observation space.

As for Agent 3, we see another pattern. Please refer to Case 1, figure 5.15, and Case 2, figure 5.28c. We can observe that Agent 3 is not making sense of the observations in Case 1, but in Case 2, it performs closer to the level of the other

two Agents. This finding indicates that Agent 3 is highly time-aware. Remember, the reward function for Agent 3 has a distance to the target and heading as the leading indicator of achievement. In figure 5.27b, we can observe that Agent 3 is reducing the distance to the target quicker than Agents 1 and 2. This may be a reason for the time awareness because Agent 3 has mapped the relationship between distance and time left of the episode to the accumulated rewards. There may be better implementation to make the Agent less time-aware. Future implementation should investigate how to implement an environment that can work in a continuous (non-episodic) fashion rather than episodic as this implementation is, and remove the time-awareness of the Agent.

The spatial plots show somewhat successful training of the Agents. However, the control signals show rapid changes and unstable behaviour in figures 5.19, 5.20 and 5.21, it is shown that three Agents predicts controller actions that oscillate heavily. As explained in the 5.2, the reward shaping was designed to prevent this. However, a value from 0.1 to 0.01 is evidently not enough penalty to avoid oscillating controller signals. Another element in this consideration is the controller penalty function composition, which is taking the absolute delta between two-time steps for the whole controller vector, consequently giving the Agent the opportunity to cheat by switching the controller input for Surge, and Yaw at the same time step, thereby not imposing any penalty. Future implementation should avoid this mistake and penalise each controller signal independently.

5.6 Summary

In this chapter, the theory of Neural Networks, the Markov Decision Process and the Reinforcement learning algorithms is explained, leading up to the state-of-the-art model-free Proximal Policy Optimisation algorithm.
The section 5.2 explains how the implementation uses Stable Baseline as a framework to create an environment for RL that can make use of a pre-implemented algorithm to train Agents.
The training of the Agents went through extensive trial and error on the way towards the final training procedure. As the result shows, the Agents are predicting somewhat feasible controller actions. However, the controls oscillate aggressively, making them unfit for practical implementations. Additionally, the Agents are too time-aware and sensitive to large variations in the observation space, making them impractical in general target-tracking controls of the Otter.

Chapter 6

Discussion and Future Work

In this chapter, a comparative analysis of the two approaches is presented. The thesis aims to investigate the feasibility of a nonlinear model predictive controller and a reinforcement learning controller. Previous work has looked at the two approaches individually, making it challenging to compare them. In this thesis, both approaches are presented in the same dissertation, enabling a viable comparison of the methods and their performance. An in-depth discussion about the individual implementation and the resulting performance is placed in the *Discussion* in chapter 4 for NMPC and chapter 5 for the RL.

6.1 Discussion

6.1.1 Modelling

Two different approaches are implemented when modelling the Otter Dynamics, one in the Casadi Framework and one using the Fossen Vehicle Simulator (FVS). Casadi uses Runge Kutta 4t(RK4) method of integration, whilst the FVS uses Euler's method for discretization. RK4 is generally a more precise method than Euler's integration method. On the other hand, the RK4 is computationally more expensive than the Euler Method.

In contrast to the NMPC, the RL environment uses the FVS directly [14] instead of the simpler 3 DOF Casadi model. The FVS considers the propeller dynamics, which reduces the reality gap and should lead to better controller performance when using the controller in reality. Thus the RL Agents have a richer insight into the real dynamics than the NMPC. Despite this, the Agents could not surpass the performance of the NMPC.

6.1.2 Development

The NMPC is well-tested and has solid foundations in mathematics and optimisation. Thus, it is a relatively straightforward task to implement. Moreover, a significant advantage in developing and fine-tuning the NMPC is its interpretability, which enables direct analysis of its behaviour from the implemented mathematical expression. This feature allows for a better understanding of the system's

decision-making process and facilitates the identification of potential irregularities. In contrast, understanding the results produced by the Agents proved difficult. The Agents *Actor NN* is a black box, making it difficult to analyse what features affect the Agents' behaviour. However, with the meticulous investigation proposed in the thesis, it's possible to understand the Agents' behaviour better. Comparing the implementation effort put into the different methods, the RL lifts the heaviest burden, much because it is not explainable. Consequently, all adjustments during development must be analysed after the time-consuming training, which is hours or days. Furthermore, there is no simple way of tuning the controller after the training. Therefore, in need of further tuning that includes changes in Agent behaviour, it is necessary to conduct new training sessions, often from scratch.

6.1.3 Performance

In section 4.4 and 5.4, we saw that both methods were moderately able to control the Otter to track a moving target. The NMPC and the trained Agents rapidly closed the distance to the target at the simulations' start. The NMPC settled on a stable error of $\approx 0.7m$. At the same time, the superior *Agent 1* in the RL approach oscillated between 0 and $\approx 2m$ at the start of the simulation before increasing the error to oscillating between 0 and $\approx 11m$ in the second half of the test. As we can see from the results, the NMPC has a more robust and consequent performance. However, the Agents is occasionally bounding the Otter and the target. What causes this behaviour is difficult to say, but the Agents are not predicting the controller trajectory as the NMPC is. Therefore, as discussed in sec 4.5, the NMPC is likely predicting to intercept the target at an occurrence within the computed controller trajectory. On the other hand, the Agents see the eight previous observations and may estimate the next position of the target and calculate the next controller actions accordingly. Thus, while the NMPC is confided in intercepting the target in the future as it considers it stationary, the Agent can predict where to intercept it. On the other hand, the unpredictable behaviour of the Agent is not very convincing and it may be more likely that the Agent is lucky with its oscillating controller signals and therefore intercepts the target. Regardless, the Agent occasionally performs better than the NMPC when solely considering the distance from the target.

As for the controller actions, the NMPC show smooth transitions and gives controller inputs largely achievable by the Otter dynamics. In contrast, Agnet 1 gave aggressive oscillating controller action that would result in heavy wear on the Otter's effectors. In physical usage, the controller signals must be achievable by preserving the thruster dynamics.

As previously explained in section 4.5, the NMPC could provide a sampling of 1.6 Hz. The low sampling would likely be ineffective in demanding conditions with noisy signals affected by environmental forces. On the other hand, the RL did not suffer from a large computation time and could provide up to 1000Hz sampling and, if trained on a large enough observation space, provide a highly responsive controller.

6.2 Future work

Future work should look into acquiring the best of best of both methods. A downside with the RL method is the training time, it is both energy demanding and takes a long time and is not explainable. On the other hand, the Agent's actor NN is fast and can compute controller signals in a rapid phase. Combining the insightful NMPC that has access to the dynamics of the Otter, and the RL training, would likely result in fast controllers with high accuracy. Additionally, the training time would likely be reduced dramatically, saving time and energy. A technique within RL is *Imitation Learning* (IL) and uses an *expert* to provide valuable insights for the Agent's mapping of observations and predictions. Others have tried this for autonomous cars [2] and have benchmarked the different IL methods [9]. Stable baseline provides frameworks for IL [1]. Further into the future, one could look into using NMPC as an online expert to evaluate new areas in the observation space and train the RL when needed.

Chapter 7

Conclusion

This thesis has shown the development and comparison of NMPC and RL as target tracking controllers for an Otter USV. The purpose was to study the difference between the approaches' development endeavours and their performance in a simulated experiment. By doing so, this thesis intends to contribute to the general progress of advanced controllers for marine vessels. Furthermore, these contributions desire to facilitate the optical communication system under development in the OASYS project.

The difference in implementation efforts of the two techniques is significant. The MPC is vastly explainable, enabling straightforward tuning to adjust the controller's performance. In contrast, the RL Agents are black boxes requiring an entirely different approach in tuning the controller's behaviour to increase performance.
This thesis has also presented a methodology to analyse the effect the hyperparameters and reward function have on the Agent's behaviour and performance. The results show that the learning rate particularly impacts how fast the RL algorithm can derive feasible parameters for the policy. It is also shown that different configurations of the reward function give different behaviour of the Agents. Lastly, it is highlighted that the implementation of the observation in the training environment will significantly impact the controller performance of the Agents.

The experiment was a simulation where the controllers were tracking the OASYS AUG. The virtual target was emulating the movement of the AUGs position in the NED frame. The controller's objective was to track the target within a 10m radius. During the simulated performance test, the NMPC method showed a feasible performance and generalisation by controlling the Otter in smooth movements before converging to an error of $\approx 0.7m$ from the target. However, the NMPC had a slow sampling of 1.6 Hz. Conversely, the RL method struggled to train Agents that performed on the same level as the NMPC. The Agents performed satisfactorily when the observations were similar to the training environment. However, when the observations surpassed the ones experienced during training, the Agents created unsatisfactory controller signals, consequently making the Agents unfit for general purposes. The sampling for the RL was 1000Hz, which is a major benefit compared to the NMPC.

In conclusion, the procedures and simulated performance tests proposed in this

thesis have provided insight into the benefits and challenges of the two approaches. NMPC is a robust and explainable approach providing a sturdy result. However, it might need to be faster for practical implementations. The RL Agents are difficult to analyse, making it challenging to derive robust controllers. Therefore, the RL approach requires further development to be reliable.

Future work should consider a complementary method by combining the NMPC and RL using *Imitation Learning (IL)*. The NMPC would, in an IL case, act as an *expert* providing insights to the Agent during training.

Bibliography

[1] Stable Baseline 3. Imitation learning. https://stable-baselines3.readthedocs.io/en/master/guide/imitation.html. [Online; accessed 20-03-2023].

[2] Flavia Sofia Acerbo, Jan Swevers, Tinne Tuytelaars, and Tong Duy Son. Mpc-based imitation learning for safe and human-like autonomous driving, 2022.

[3] Instructor at Stanford. Lecture notes, cs231n: Deep learning for computer vision, stanford. https://cs231n.github.io/neural-networks-1/. [Online; accessed 20-03-2023].

[4] RICHARD Bellman. Dynamic programming, princeton univ. *Press Princeton, New Jersey*, 1957.

[5] Henrik Bjering Strand. Autonomous docking control system for the otter usv: A machine learning approach. Master's thesis, NTNU, 2020.

[6] Diana Borsa, Bilal Piot, Rémi Munos, and Olivier Pietquin. Observational learning by reinforcement learning, 2017.

[7] casadi. Optimal control problems in a nutshell. https://web.casadi.org/blog/ocp/. [Online; accessed 20-02-2023].

[8] Leif Christensen, José de Gea Fernández, Marc Hildebrandt, Christian Ernst Siegfried Koch, and Bilal Wehbe. Recent advances in ai for navigation and control of underwater robots. *Current Robotics Reports*, pages 1–11, 2022.

[9] Kamil Ciosek. Imitation learning by reinforcement learning, 2022.

[10] Community effort. Gymnasium farama. https://gymnasium.farama.org/y. [Online; accessed 21-03-2023].

[11] Community effort. Hebbian theory. https://en.wikipedia.org/wiki/Hebbian_theory. [Online; accessed 21-03-2023].

[12] Moustafa Elkolali, Ahmed Al-Tawil, and Alex Alcocer. Design and testing of a miniature variable buoyancy system for underwater vehicles. *IEEE Access*, 10:42297–42308, 2022.

[13] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control, Second Edition*. John Wiley & Sons, 2021.

[14] Thor Inge Fossen. Python vehicle simulator. https://github.com/cybergalactic/PythonVehicleSimulator. [Online; accessed 24-10-2022].

[15] Thor Inge Fossen. *Nonlinear modelling and control of underwater vehicles*. Norwegian University of Science and Technology, Trondheim, Norway, 1991.

[16] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[17] Shahzad Khan. *Ethem Alpaydin. Introduction to Machine Learning (Adaptive Computation and Machine Learning Series). The MIT Press, 2004. ISBN: 0 262 01211*, volume 14. Cambridge University Press, 2008.

[18] Sofia Kockum. Autonomous docking of an unmanned surface vehicle using model predictive control, 2022.

[19] H Lamb. *Hydrodynamics*. Cambridge University Press, London, 1932.

[20] Brett Lantz. *Machine learning with R*. Packt publishing ltd, 2013.

[21] Andreas B Martinsen, Anastasios M Lekkas, and Sebastien Gros. Autonomous docking using direct optimal control. *IFAC-PapersOnLine*, 52(21):97–102, 2019.

[22] Andreas Bell Martinsen. End-to-end training for path following and control of marine vehicles. Master's thesis, NTNU, 2018.

[23] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[24] USA Department of Defence. https://www.gps.gov/technical/ps/2020-sps-performance-standard.pdf. https://www.gps.gov/technical/ps/2020-SPS-performance-standard.pdf. [Online; accessed 30-04-2023].

[25] Mihir Patil, Bilal Wehbe, and Matias Valdenegro-Toro. Deep reinforcement learning for continuous docking control of autonomous underwater vehicles: a benchmarking study. In *OCEANS 2021: San Diego–Porto*, pages 1–7. IEEE, 2021.

[26] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[27] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[28] Stefan Schneider. Dealing with partial observability in reinforcement learning. https://stefanbschneider.github.io/blog/rl-partial-observability . [Online; accessed 21-03-2023].

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] D.E. Seborg, T.F. Edgar, D.A. Mellichamp, and F.J. Doyle. *Process Dynamics and Control*. Wiley, 2017.

[31] SAGAR SHARMA. Activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, September 2017. [Online; accessed 29-April-2022].

[32] Neryahu A Shneydor. *Missile guidance and pursuit: kinematics, dynamics and control*. Elsevier, 1998.

[33] Erik Sollesnes, Ole Martin Brokstad, Rolf Klæ boe, Bendik Vågen, Alfredo Carella, Alex Alcocer, Artur Piotr Zolich, and Tor Arne Johansen. Towards autonomous ocean observing systems using miniature underwater gliders with uav deployment and recovery capabilities. In *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, pages 1–5, 2018.

[34] Erik Sollesnes, Ole Martin Brokstad, Bendik Vågen, Alfredo Carella, Alex Alcocer, Artur Piotr Zolich, Tor Arne Johansen, et al. Towards autonomous ocean observing systems using miniature underwater gliders with uav deployment and recovery capabilities. In *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, pages 1–5. IEEE, 2018.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018, 2020.

[36] Andreas Wächter. Short tutorial: Getting started with ipopt in 90 minutes. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2009.

[37] Luman Zhao, Myung-Il Roh, and Sung-Jun Lee. Control method for path following and collision avoidance of autonomous ship based on deep reinforcement learning. *Journal of Marine Science and Technology*, 27(4):1, 2019.

[38] Huarong Zheng, Rudy R Negenborn, and Gabriel Lodewijks. Trajectory tracking of autonomous vessels using model predictive control. *IFAC Proceedings Volumes*, 47(3):8812–8818, 2014.

Appendices

Appendix A

Materials

The primary software used is listed below.

Software	**Version**
Windows	11
Python	3.10.6
Casadi	3.5.6
StableBaselines3	1.7.0
PyTorch	1.13.1+cpu

Table A.1: Software list

All training and testing runs have been performed on a DELL XPS 17 9720 laptop containing the following hardware components:

Hardware	**Type**
CPU	12th Gen Intel(R) Core(TM) i9-12900HK 2.50 GHz
GPU	RTX 3060 Laptop
RAM	64 GB

Table A.2: Hardware list

Appendix B

Otter Model Parameters

This is a listing of the physical parameters and matrix coefficients that are used in the Fossen Vehicle Simulator.

Parameter	**Description**	**Value**	**Unit**
L	Length	2.0	[m]
B	Beam	1.08	[m]
m	Mass	55.0	[kg]
r_g	Center of gravity	$\begin{bmatrix}0.2 & 0 & -0.2\end{bmatrix}$	[m]
R_{44}	R$_a dii$	0.4B	[m]
R_{55}	R$_a dii$	0.25L	[m]
R_{66}	R$_a dii$	0.25L	[m]
B_{pont}	Beam of one pontoon	0.25	[m]
Y_{pont}	Distance from center line to waterline	0.395	[m]
$C_w, pont$	Waterline area coefficient	0.75	-
$C_b, pont$	Block coefficient	0.4	-
k_{pos}	Positive Bollard, one propeller	0.01108	-
k_{neg}	Negative Bollard, one propeller	0.006445	-
T_{yaw}	Time constant yaw	1	[s]
U_{max}	Maximum speed of craft	6 * 0.5144	[m/s]

Table B.1: Physhical parameters of Otter USV [5]

Appendix C

Code

The code base used in this project may be found in the GitHub repo:
https://github.com/akseljohan/Otter-USV-Target-Tracking-Controllers.git