Master's thesis

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

Simon Mork Sætre

# Laying The Foundation For an Artificial Intelligence-Powered Extendable Digital Twin Framework For Autonomous Sea Vessels

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
June 2022

NTNU
Norwegian University of Science and Technology

Simon Mork Sætre

# Laying The Foundation For an Artificial Intelligence-Powered Extendable Digital Twin Framework For Autonomous Sea Vessels

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
June 2022

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

NTNU
Norwegian University of
Science and Technology

*Engineering Cybernetics*
*TTK4900 Master Thesis*

# Laying The Foundation For an Artificial Intelligence-Powered Extendable Digital Twin Framework For Autonomous Sea Vessels

*Simon Mork Sætre*

*Supervisor:*
*Professor Adil Rasheed*

*Trondheim, June 9th, 2022*

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology and Electrical Engineering
DEPARTMENT OF ENGINEERING CYBERNETICS

# Acknowledgements

I would like to thank Adil Rasheed, my supervisor and professor at the department of Information Technology and Electrical Engineering at NTNU for his help during this project. Thank you for providing me guidance for when I was stuck and for sharing your insight.
I would also thank PhD. student Thomas Nakken Larsen for assisting me whenever I had questions about reinforcement learning and gave me advice when needed.

# Abbreviations

Some abbreviations and acronyms are used in this thesis.

**AI** - Artificial Intelligence  
**AIS** - Automatic Identification System  
**DSM** - Digital Surface Model  
**DTM** - Digital Terrain Model  
**LIDAR** - **LI**ght **D**etection **A**nd **R**anging  
**LSTM** - Long Short-Term Memory  
**ML** - Machine learning  
**MLAgents** - Unity Machine Learning Agents Toolkit  
**MET** - Norwegian Meteorological Institute/Meteorologisk Institutt  
**MMSI** - Maritime Mobile Service Identity  
**PPO** - Proximal Policy Optimization  
**RL** - Reinforcement Learning  
**SAC** - Soft Actor Critic  
**VR** - Virtual Reality

# Contents

# List of Figures

# List of Tables

## Abstract

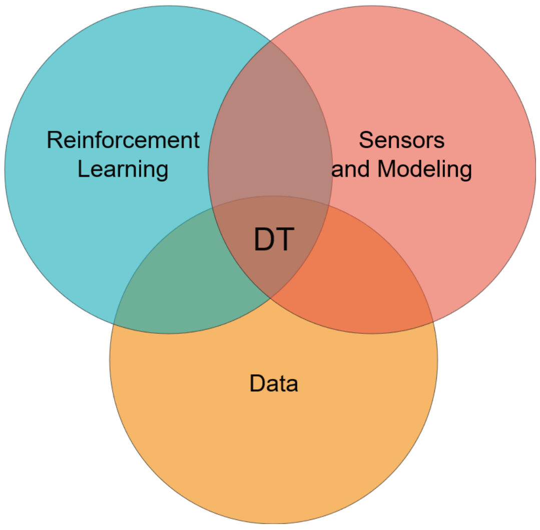
Humans are the main cause of accidents at sea. It is therefore desirable in some cases to create autonomous vehicles capable of operating without any human intervention. A framework for visualisation and development of a digital twin and RL algorithms is developed in Unity Game Engine. A digital representation of the Trondheim Fjord terrain is created, where wind and rain is implemented by using real-time real world values for the area. Boats emitting AIS signals in the area are also used to create digital representations of the ships. Virtual Reality was implemented in the project which allows for both normal windowed visualisation and virtual visualisation. In addition, the Milliampere 2 boat is represented accurately in the project as a virtual object dictated by a flexible dynamical model. This dynamical model is affected by external forces like wind and two thrusters attached to the bottom of the vessel. As the real boat should have a LIDAR, a ray casting sensor array is used in the project to mimic its real counterpart. A pooling algorithm for the sensor readings is used to reduce the dimensions in the sensor from 220 to 9. This is used in an RL algorithm for controlling the boat.
A RL algorithm using PPO is trained inside of Unity, where the goal is to follow a given path as closely as possible while avoiding obstacles like land and other boats. After RL training, the boat was given over 100 attempts to complete four paths of varying degree of difficulty. The boat had a 100% success rate on both of the paths containing no obstacles. This success rate dropped to 17% when obstacles were introduced. While the success rate was low when obstacles were introduced, it is suggested that minor changes in Unity will improve future training. A safety filter could be an good addition to ensure the RL model will not act dangerously.

**Video: https://youtu.be/h8PAtYfLLOc**

## Sammendrag

Mennesker er hovedårsaken til at det oppstår ulykker til sjøs. Det er derfor ønskelig i noen tilfeller å ta i bruk autonome kjøretøy som er i stand til å fungere uten at mennesker må blande seg inn. Et rammeverk for visualisering og utviklinger av en digital tvilling og RL-algoritmer blir skapt i spillmotoren Unity. En digital representasjon av Trondheimsfjorden blir skapt, der vind og regn blir simulert ved hjelp av sanntidsdata fra ekte målinger. Båter som sender AIS-signaler i området blir også tatt i bruk for å skape digitale representasjoner av hvert skip. VR blir også implementert i prosjektet, noe som gjør virtuell og vanlig visualisering mulig. I tillegg så er en dynamisk modell av båten Milliampere 2 implementert på en nøyaktig måte. Denne dynamiske modellen blir påvvirket av både vind og thrusterne som er montert under båten. Siden den ekte båten skal ha en LIDAR, så blir en strålekaster brukt i prosjektet for å etterligne LIDARen. En algoritme blir brukt for å slå sammen verdiene den beregner og gjør at dimensjonene blir redusert fra 220 til 9. Dette blir brukt i en RL-algoritme for å kontrollere båten.
En RL-algoritme som bruker PPO blir trent i Unity, der målet er å følge en gitt sti. Den skal gjøre dette så godt som mulig mens den unngår hindringer som andre båter og land. Etter treningen ble den gitt over 100 forsøk på å fullføre fire baner med økende vanskelighetsgrad, der to av de inneholder hindringer. Båten klarte de letteste banene 100% av forsøkene, men den fullførte bare 17% av gangene på de to banene med hindringer. Selv om fullføringsraten var lav på banene med hindringer, så er det foreslått at noen små endringer kan forbedre resultatene betydelig. Et sikkerhetsfilter kan være en god løsning for å ytterliggere forbedre sikkerheten slik at båten aldri er i fare. **Video: https://youtu.be/h8PAtYfLLOc**

# 1 Introduction



As the world develops, new ways of solving problems have been unlocked. With the current advances within machine learning and thus reinforcement learning, one can use data gathered from sensors together with modeling and simulations to create and improve digital twins(DT). This project's goal is to create a suitable framework for developing reinforcement learning algorithms, and looking into using Unity Game Engine for Digital Twin capabilities.

## 1.1 Motivation and Background

Autoferry is a multidisciplinary project being developed by several NTNU departments. The goal of the project is to create a ferry capable of navigating autonomously without being controlled by any human[1]. The Autoferry project aims to eliminate the need for bridges across any river by utilizing an autonomous boat to transport pedestrians. Thus, the boat serves pedestrians at multiple locations and provides a more flexible solution than a bridge would. By making the boat autonomous, one can essentially remove the human from the loop during normal operation, which has many advantages. According to the Sanchez [2], humans are the cause of 80% of all errors at sea. These errors can be caused by several factors like lack of training, overconfidence, or lack of attention. Removing humans from the loop and replacing them with an algorithm is thus desired if one would like to increase safety while navigating the seas. Removing humans from the loop also means lower costs regarding salaries and management. Unfortunately, manually programming an algorithm to control a sea vessel is not a trivial task. The sea is a highly stochastic environment featuring external forces like wind and ocean waves, and currents. An algorithm also has to consider any possible scenario that could happen at sea. Programming an algorithm suitable for every possible scenario in a highly stochastic environment will be a daunting task that will be significantly time-consuming and costly, or in other words practically impossible. This is why taking a different approach and using Reinforcement Learning(RL) might be more feasible. It should be worth seeing if it is capable of creating an algorithm capable of navigating the seas. An RL algorithm should cost just a fraction of the development cost because of the assumed reduced development cost because a hand-designed algorithm would need to be capable of handling every situation without human intervention. Unfortunately, an RL algorithm needs to train and learn how actually to navigate a selected sea vessel. To let an algorithm learn by using a real ship in the real world would be both dangerous and costly. This is why it is desirable to look at creating a digital simulated environment that is realistic enough to replicate the real counterpart. After training, the environment could be viable for providing additional feedback to the algorithm during regular operation. This approach is called Digital Twin and will be discussed later.
This project is a continuation of the specialisation project [3] and thus uses Unity Game Engine to further develop an environment for development, visualisation and potential digital

twin capabilities for an RL model. The specialisation project did discover that it was possible to train RL models and represent the Milliampere boat in a digital representation of a real environment inside of the game engine. The specialisation project did succeed, but fell short in certain aspects. Utilising a game engine for scientific purposes is rather novel, but still not quite unheard of. As of 9th of May 2022, 425 results match the search term ""Unity game engine" digital twin" on Google Scholar. Out of these results, 198 articles were published after the start of 2021. This suggests that using Unity Game Engine for digital twins is still rather new.

### 1.1.1 Literature Review

This project takes inspiration and discoveries from multiple sources The foundation of this master thesis was made in the specialisation project developed in the autumn semester 2021[3]. While this project laid a decent foundation for this thesis, it still had a lot of shortcomings, like having a very simplified and thus inaccurate mathematical model of the boat. It was attempted to implement a mathematical model of the boat, but Unity lacked proper support for matrix calculations, which reduced the mathematical model into a simple dynamical equation. The friction and dampening coefficients in the dynamical equation were just simple constants that did not care which direction the boat traveled in, just the magnitude of the velocity. The boat also lacked any vision component in the RL algorithm, which made it unable to detect any obstacles. The RL algorithm in other words was only able to follow a path and had to collision implemented. The environment was also quite lackluster by having a low resolution and issues with "blocky" terrain. The environment was also textured with a simple colour gradient, which gave it a cartoon-like style and thus an unrealistic look. While a great start, the specialisation project had a lot of cut corners in order to create a presentable project.
The model identification research paper for the Milliampere ferry[4] explores thoroughly the dynamics of the Milliampere ferry being developed by NTNU. The research paper uses experimental data with optimisation algorithms to develop several dynamical models for the Milliampere ship. Several models are proposed, where a fully-coupled model and a surge decoupled model are developed. Different dynamical models were simulated and compared against a real vessel, where both the surge decoupled, and the fully coupled model were both verified to be accurate enough. The research paper suggests that the surge decoupled model for the Milliampere ship is accurate enough for simulations while being significantly easier to implement. In addition to developing a dynamical model for the ship, the paper also proposes a model for the forces acted on the boat, like the thruster motors and wind forces. While the thruster model was described well enough, the implementation of wind forces in this paper were poorly described and had to be supplemented by the Handbook written by Fossen [5] to give the full picture. Despite this fact, the paper overall provides a great foundation for implementing a digital representation of the Milliampere boat, like for example in this project.
The research paper by Larsen [6] compares different RL algorithms in the task of path following and collision avoidance. This paper used the various RL algorithms on a dynamical model of the Milliampere ferry, which makes this paper very relevant to the work done in this project. The RL algorithms Proximal Policy Optimization(PPO), Deep Deterministic Policy Gradient(DDPG), Twin Delayed DDPG(TD3), and Soft Actor-Critic(SAC) are compared in various environments and evaluated. The PPO algorithm was the best performing RL algorithm of them all and was proven to be robust for all presented cases.
The book by Thor I. Fossen [5], supplements the model identification research paper [4] by providing an expanded explanation of the theory behind for instance the dynamics of a boat and the wind modeling. This is obviously the case when the article finding the dynamical model [4] uses the theory in Fossen's book for most of the fundamentals. This book has been useful for when certain parts of the theory featured in [4] has been too shallow and required deeper insight to be properly implemented in this project. This book will also come in handy for potential expansions of the dynamical model once the possibility of more real world variables are introduced, like precipitation and temperature.

## 1.2 Research Objectives and Research Questions

Determining the objectives is important for guiding the work.

### 1.2.1 Objectives

Primary Objective: To establish an extendable digital twin framework for autonomous sea vessel

Secondary Objectives:

- Develop an expandable digital framework for autonomous ship navigation
- Improve the visual appeal and accuracy of the terrain F
- Fetch real world data and utilise them for calculations and visualisation
- Implement Virtual Reality(VR) visualisation

### 1.2.2 Research Questions

From what is known, Unity Game Engine has not been used much for scientific purposes. It is therefore important to specify questions to guide the work.

- How usable is Unity Game Engine for simulating a realistic RL environment?
- Will the algorithm trained in Unity become safe enough, proving that a RL in Unity is usable for real use cases?

## 1.3 Outline of Report

**Chapter 1** contains the introduction of the report. It also states the research questions, literature reviews and objectives.
**Chapter 2** introduces the dynamical model of the sea vessel and thruster dynamics. The RL section describes an overview of the algorithm. Some sections in the chapter presents first what was done in the specialisation project and then introduces the improvements brought into the master project.
**Chapter 3** presents the methodology of the project, where it presents the development and reasoning behind various aspects of the project. This includes the creation of the map, implementation of live wind data, visualisation of real time rain, use and implementation of a digital Light Detection And Ranging(LIDAR) sensor, the reward function and the setup of the training, and finally the implementation of VR into the project. Keep in mind that the implementation of the map is especially detailed because of how nontrivial the task is. The usefulness of this knowledge will most likely also be useful for other projects, which is why this specific section thoroughly explains the process.
**Chapter 4** provides the results of the testing of the algorithm. In addition, the developed project is discussed.
**Chapter 5** Concludes the thesis by seeing the status on the research objectives and by answering the research questions.

# 2 Theory

## 2.1 Digital twin and capability level

The Digital Twin concept is relatively new. By looking in the top right side figure 2.1, the representation of the physical object can be seen. This physical object is fitted with a deemed significant amount of different sensors. These sensors then provide real-time bigdata for further processing. Big data can be explained as simply being huge amounts of data containing many different variables that can be hard to analyze by normal means. While it's bigdata, the spatio-temporal resolution might be an issue. Spatio-temporal data resolution refers to a data's resolution in both space and time. For example, one can imagine a data set containing the temperature of an area represented as a grid. This data set can also contain past temperatures of said area grid. This data set thus is a spatio-temporal data set. The spatio-temporal resolution of this data might be too low and could be unable to predict the future of the state of the object. Models are because of this used together with the big data to predict the future states of the object and fully create a digital representation of the object. With an accurate digital representation, one can combine its behavior with the physical version of the object and perform better decision-making and optimal control of the physical object. This assumes that both the physical and digital representations receive the same data. As one can see on figure 2.1, the green arrows indicate the real-time data flow and analysis. In addition to optimal control, one can use digital twins to create a more informed risk assessment, What-if analysis, uncertainty quantification, and process optimization. By instead running the digital twins offline and simulating different scenarios, one can obtain deeper knowledge within these. The digital twins that run offline and are separated from the physical object are then called digital siblings. As one can see in the figure, the grey arrows and boxes represent data flow from the digital siblings. The data obtained from the digital siblings can be stored and used to develop future generations of improved objects. This is called a digital threads.

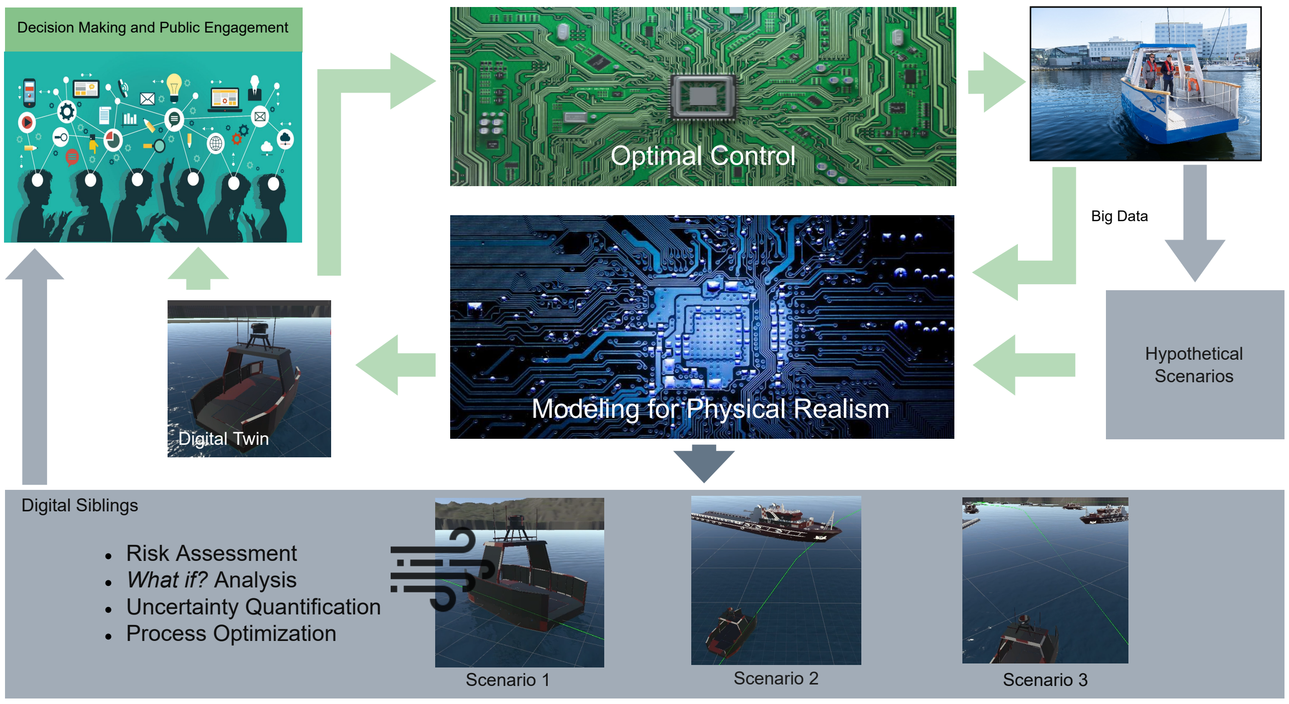


**Figure 2.1:** Process of a Digital Twin. Adapted from [7]

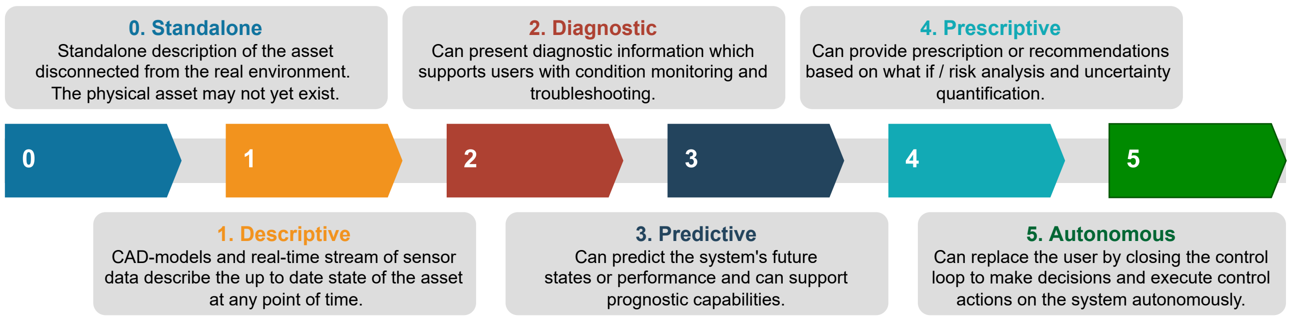

0. Standalone
Standalone description of the asset disconnected from the real environment. The physical asset may not yet exist.
2. Diagnostic
Can present diagnostic information which supports users with condition monitoring and troubleshooting.
4. Prescriptive
Can provide prescription or recommendations based on what if / risk analysis and uncertainty quantification.
0
1
2
3
4
5
1. Descriptive
CAD-models and real-time stream of sensor data describe the up to date state of the asset at any point of time.
3. Predictive
Can predict the system's future states or performance and can support prognostic capabilities.
5. Autonomous
Can replace the user by closing the control loop to make decisions and execute control actions on the system autonomously.

**Figure 2.2:** The capability levels of DTs on a scale from 0 to 5

The digital twin concept has been existed for a while. Unfortunately it can often be used in a misleading way. The way a digital twin resembles its physical counterpart can vary. Because of this, digital twins are divided into six categories as seen in figure 2.2. These categories go from 0 to 5 and are in ascending order of complexity; standalone, descriptive, diagnostic, predictive, prescriptive, and autonomous. A digital twin can be called a digital twin even before the physical object exists. The digital twin can then be used for the cost-benefit analysis, which is useful in the production of the physical object for instance. One can use a CAD model of the physical object combined with live sensor data to gain insight into areas of the physical object that are impossible to measure, like stress forces inside of the object itself. This type of digital twin is called a descriptive digital twin. A digital twin at the capability scale of 2 can be used to detect errors or possible faults within the physical object. As an example, a digital twin can determine from modeling and sensor readings that a part has been damaged. It is important to note that standalone, descriptive, and diagnostic twins cannot provide any insight into the future. However, predictive twins are capable of this and use models to predict the future by using old and new states. This can be pretty useful in, for instance, predictive maintenance. For example, the digital twin can determine that a part of the physical object will suffer from fatigue and affect the rest of the system in the near future. Maintenance personnel can then fix the issue before it happens. Prescriptive digital twins can use what if, risk analysis, and uncertainty quantification to give recommendations. This is quite desirable in decision support systems. At 5, the highest capability, one has an autonomous digital twin. Here the digital twin and the physical object communicate both ways. The physical object provides information regarding the situation, and the digital twin controls the physical object towards the desired state. A fully autonomous system thus does not provide recommendations but instead makes its own decisions without human intervention. This is, for example very useful for controlling vehicles without any drivers.[8]

## 2.2 Boat dynamics

The specialisation project had to significantly simplify the dynamical model of the boat. The dynamical model of the boat was in the project modeled using the model in equation 2.3.

$$p = \begin{bmatrix} x \\ z \\ \theta \end{bmatrix} \tag{2.1}$$

$$\tau = \begin{bmatrix} X \\ Z \\ N \end{bmatrix} \tag{2.2}$$

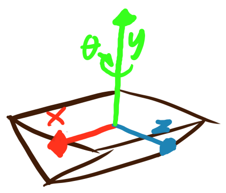


**Figure 2.3:** The Unity's coordinate system. $x$ here is the forward direction for the boat

Equation 2.1 represents the boat's position in x and z, where the boat's position can be represented on an 2D plane. The $\theta$ represents the boat's rotation along the y-axis. With these variables, one can easily represent the boat on a 2D plane. An illustration of this can be seen in figure 2.3. Notice that Unity's coordinate system is left-handed and $\theta$ is therefore represented with positive value in the opposite direction of what would be normal.

$$M\ddot{p} + D(\dot{p})\dot{p} = \tau \tag{2.3}$$

The previous dynamical model as shown in equation 2.3, represents the behaviour of the ship by representing the inertia $M$ and the dampening $D$ as purely diagonal matrices with no interactions between the variables. $D$ contains both linear and nonlinear components derived from the research paper [4]. $\tau$ here represents the input forces from the thrusters mounted underneath the boat. $\tau$, as shown in equation 2.2, contains the forces in $X$ and $Z$ direction and the torque $N$ with respect to the $y$ axis. Because this iteration of the boat had no external disturbances, a disturbance vector $\tau_d$ was not added.

The dynamical model of the boat derived from article [4] is shown in equation 2.5.

$$\dot{\eta} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{2.4}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} + \boldsymbol{\tau}_{\mathrm{d}} \tag{2.5}$$

$$\boldsymbol{\eta} = [x, z, \psi]^T \tag{2.6}$$

$$\boldsymbol{\nu} = [u, v, r]^T \tag{2.7}$$

$$\boldsymbol{\tau} = [X, Y, N]^T \tag{2.8}$$

This new mathematical model uses the surge decoupled model from the original research paper because it was determined to be significantly easier to use in simulations, while at the same time maintaining an accuracy deemed to be accurate enough [4]. The inertia matrix in the surge decoupled model contains interactions between the different variables, which means for example the velocity in $Z$ will be affected by the angular velocity $\psi$. The previous model developed in the specialisation project 2.3 did not contain any of these interactions, which means the velocities and the angular velocity were fully decoupled.

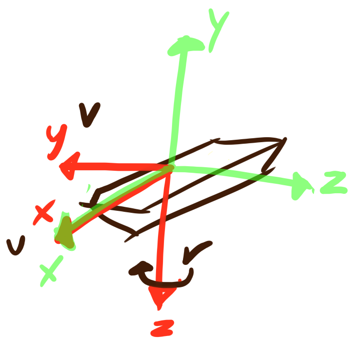


**Figure 2.4:** The coordinate system for the dynamical model (red) and Unity's coordinate system (green)

As shown in equation 2.3, the coordinate systems are not the same and need to be related to each other with a simple transformation matrix shown in equation 2.9. The dynamical system only requires a simple sign change of the variables because the elements in the matrices only contain absolute values of the velocity vector. Do note that the sign of the rotation does not change sign because Unity's coordinate system is left-handed.

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Such that

$$\boldsymbol{\nu} = [u, v, r]^T = [\dot{x}, -\dot{z}, \dot{\psi}]^T \tag{2.10}$$

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{32} & m_{33} \end{bmatrix} \tag{2.11}$$

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & c_{13}(\nu) \\ 0 & 0 & c_{23}(\nu) \\ c_{31}(\nu) & c_{32}(\nu) & 0 \end{bmatrix} \tag{2.12}$$

$$c_{13}(\boldsymbol{\nu}) = -m_{22}v - m_{23}r \tag{2.13a}$$
$$c_{23}(\boldsymbol{\nu}) = m_{11}u \tag{2.13b}$$
$$c_{31}(\boldsymbol{\nu}) = -c_{13}(\boldsymbol{\nu}) \tag{2.13c}$$
$$c_{32}(\boldsymbol{\nu}) = -c_{23}(\boldsymbol{\nu}) \tag{2.13d}$$

$$\boldsymbol{D}(\nu) = \begin{bmatrix} d_{11}(\nu) & 0 & 0 \\ 0 & d_{22}(\nu) & d_{23}(\nu) \\ 0 & d_{32}(\nu) & d_{33}(\nu) \end{bmatrix} \tag{2.14}$$

$$d_{11}(\boldsymbol{\nu}) = -X_u - X_{|u|u}|u| - X_{uuu}u^2 \tag{2.15a}$$
$$d_{22}(\boldsymbol{\nu}) = -Y_v - Y_{|v|v}|v| - Y_{|r|v}|r| - Y_{vvv}v^2 \tag{2.15b}$$
$$d_{23}(\boldsymbol{\nu}) = -Y_r - Y_{|v|r}|v| - Y_{|r|r}|r| \tag{2.15c}$$
$$d_{32}(\boldsymbol{\nu}) = -N_v - N_{|v|v}|v| - N_{|r|v}|r| \tag{2.15d}$$
$$d_{33}(\boldsymbol{\nu}) = -N_r - N_{|v|r}|v| - N_{|r|r}|r| - N_{rrr}r^2. \tag{2.15e}$$

| Parameter | Estimated Value | Unit |
|---|---|---|
| $m_{11}$ | 2389.657 | kg |
| $m_{22}$ | 2533.911 | kg |
| $m_{23}$ | 62.386 | kg |
| $m_{32}$ | 28.141 | kg |
| $m_{33}$ | 5068.910 | $\mathrm{kgm}^2$ |
| $X_u$ | $-27.632$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $X_{\|u\|u}$ | $-110.064$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $X_{\mathrm{uuu}}$ | $-13.965$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_v$ | $-52.947$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_{\|v\|v}$ | $-116.486$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_{vvv}$ | $-24.313$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_{\|r\|v}$ | $-1540.383$ | $\frac{\mathrm{gg}}{\mathrm{s}}$ |
| $Y_r$ | 24.732 | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_{\|v\|r}$ | 572.141 | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $Y_{\|r\|r}$ | $-115.457$ | $\frac{kg}{\mathrm{s}}$ |
| $N_v$ | 3.524 | $\frac{\mathrm{sg}}{\mathrm{s}}$ |
| $N_{\|v\|v}$ | $-0.832$ | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $N_{\|r\|v}$ | 336.827 | $\frac{\mathrm{kg}}{\mathrm{s}}$ |
| $N_r$ | $-122.860$ | $\frac{kg}{\mathrm{s}}$ |
| $N_{\|r\|r}$ | $-874.428$ | $\frac{\mathrm{kg}}{s}$ |
| $N_{rrr}$ | 0.000 | $\frac{\mathrm{kg}}{\mathrm{g}}$ |
| $N_{\|v\|r}$ | $-121.957$ | $\frac{kg}{\mathrm{s}}$ |

(2.16)

The coefficients in table 2.16 are used for the matrices 2.11, 2.12, and 2.14. They are derived from [4] and the variable names are unchanged for clarity. By utilising this dynamical system, one will be able to create a system that responds accurately when compared to its real counterpart.

## 2.3 Thruster Dynamics

The thrusters in the specialisation project [3] used a simplified version of the thruster system introduced in the Milliampere identification project [4], where the linear forces were calculated with a simple matrix 2.18. Figure 2.5 displays how the thrusters are configured on the Milliampere ferry. Do note that Y in this illustration will be -Z in the Unity coordinate system.

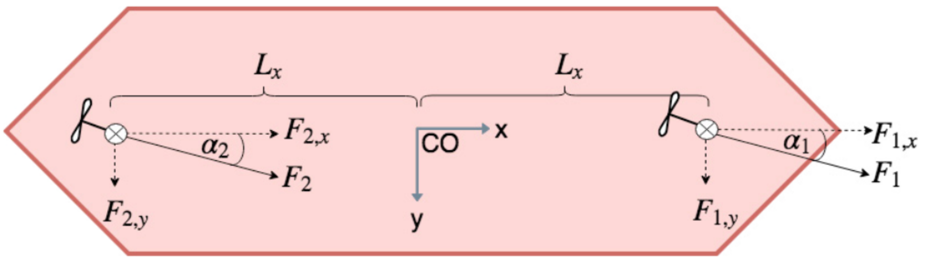


**Figure 2.5:** The thruster configuration. Image borrowed from [9]

$$\tau = T(\alpha)F \tag{2.17}$$

$$\boldsymbol{\tau} = \begin{bmatrix} \sin\alpha_1 & \sin\alpha_2 \\ \cos\alpha_1 & \cos\alpha_2 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{2.18}$$

The old thruster system as shown in 2.18 calculated the forces applied on the sea vessel in $x$ and $z$ direction. $\alpha_x$ and $\alpha_z$ are the angles of the thrusters, and $F_1$, and $F_2$ are the forces applied to the sea vessel from the thrusters. $\tau$ thus calculated the forces in each direction using these variables. It did not calculate the torque though. The torque was calculated by Unity's internal physics engine, where the force vectors and the position of the force was given to the engine to calculate their interactions with the vessel.

An improved model presented in Article [4] is used here instead such that the input vector can be applied to the dynamical model. The previous thruster model was functional, but Unity's physics component's capability of handling scientifically accurate physics is questionable at best. An expanded force vector capable of being used in the dynamical model is used instead.

$$\boldsymbol{T}(\boldsymbol{\alpha}) = \begin{bmatrix} \cos\alpha_1 & \cos\alpha_2 \\ \sin\alpha_1 & \sin\alpha_2 \\ L_x\sin\alpha_1 & L_x\sin\alpha_2 \end{bmatrix} \tag{2.19}$$

The input force vector $\tau$ is capable of calculating the torque explicitly instead of having to be calculated by Unity. This gives a greater control of the forces applied on the sea vessel and is an overall upgrade compared to the specialisation course's approach where the torque was calculated in Unity's physics engine. The variables used in 2.19 are the same as in 2.18, with $L_x$ as an additional variable describing the distance between centre of mass and the position where the force is applied by the thrusters.

The old model used for modeling the thrusters was presented in the specialisation project [3], but did not feature any modeling for the change in thrust and angle. The RL model instead controlled the angle speed and change in force directly. The paper identifying the dynamical model for the force thrust and change in azimuth angle presented equations that fit the behaviour of the real thrusters[4]. Equation 2.20 describes the change in angle speed as a sigmoid function.

$$\dot{\alpha} = K_\alpha \frac{(\alpha_d - \alpha)}{\sqrt{(\alpha_d - \alpha)^2 + \epsilon^2}} \tag{2.20}$$

$\alpha_d$ here is the desired azimuth angle for each thruster. $\alpha$ is on the other hand the actual angle of each thruster. The difference between $\alpha$ and $\alpha_d$ equals the error between the two variables. $\epsilon$ is the tuning parameter and dictates convergence of the sigmoid function. $K_\alpha$ is the rotational transmission velocity constant for the thrusters. Equation 2.20 thus uses these values to find the actual angle dynamics of a given thruster. Pedersen [4] ran several experiments to determine the value of the constants and verified that a simulated environment gave satisfying results that mimicked the real system accurately. The values for the constants are given in Table 2.21.

| Parameters | Estimated value |
|---|---|
| $K_{\alpha_1}$ | $34.458$ |
| $K_{\alpha_2}$ | $37.526$ |
| $\epsilon_1$ | $6.277$ |
| $\epsilon_2$ | $7.721$ |

(2.21)

The dynamics of the thrust forces were also modeled quite simply in the specialisation project [3]. The RL algorithm had direct control of the forces applied. The Optimization Based System Identification for the milliAmpere Ferry research paper [4] also modeled the

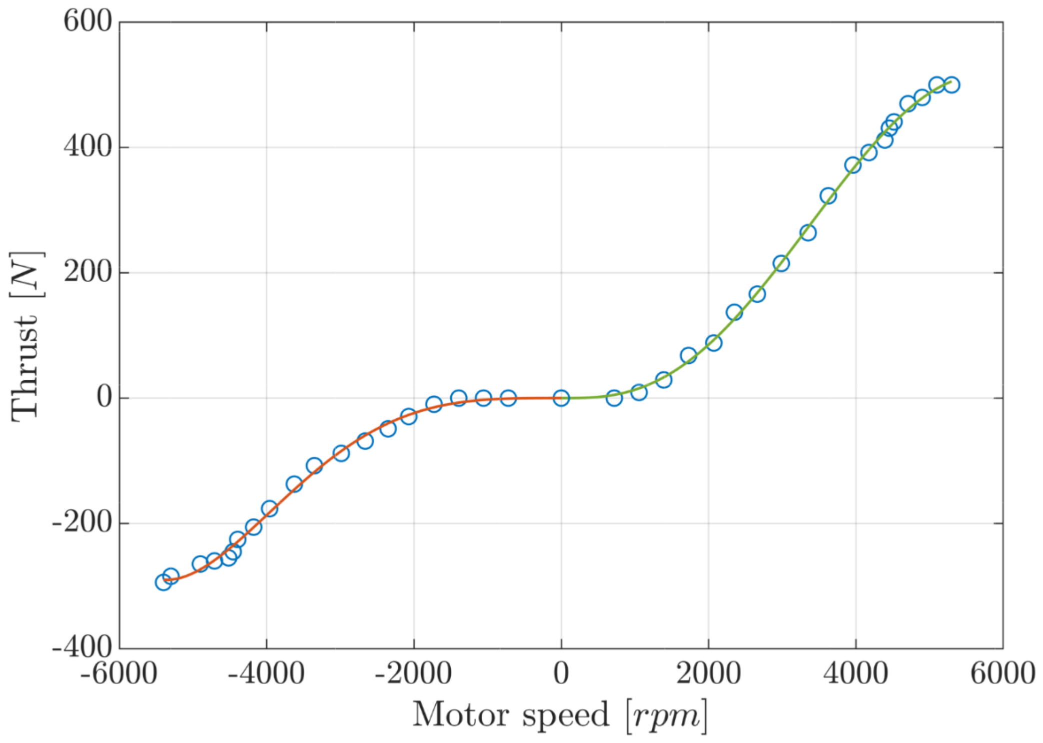


**Figure 2.6:** Thrust relative to RPM. Courtesty of [4]

force thrust of the boat. Equation 2.22 shows the simple equation for modeling the change in rotation speed of the motor as described in the article by Pedersen [4].

$$\dot{\omega} = K_\omega \left(\omega_d - \omega\right) \tag{2.22}$$

Where the constants for each thruster respectively is

| Parameters | Estimated value |
|---|---|
| $K_{\omega_1}$ | 0.563 |
| $K_{\omega_2}$ | 0.591 |

(2.23)

While this equation does not directly provide the thrust force provided by the thrusters, one can map experimental data to the rotational speed of the thrusters. Figure 2.6 shows the experimental results from Pedersen [4] mapping the rotational speed of the thruster motors to thrust in Newtons. To create a mapping from RPM to thrust, a regression of order 5 with a zero point at the origin was done on the data. Keeping the force at zero when the propellers are at 0 RPM makes sense because they should generally not produce any thrust while not spinning. The regression yields the following function:

$$F_x = 10^{-16}\omega_x^5 - 2 * 10^{-13}\omega_x^4 + 6 * 10^{-9}\omega_x^3 + 9 * 10^{-6}\omega_x^2 + 0.0036\omega_x \tag{2.24}$$

One thus finds the force vector $F$, which can be applied to find $\tau$ in 2.17 is then used in 2.5.

| Parameter | Value |
|---|---|
| $A_{Fw}$ | $2.9m^2$ |
| $A_{Lw}$ | $8.6m^2$ |
| $L_{oa}$ | $5m$ |
| $s_L$ | $0m$ |

(2.26)

**Table 2.1:** Parameters used for the Milliampere boat

## 2.4 Wind Force

Strong enough winds can significantly affect vessels at sea. While larger vessels might not be noticeably affected by calmer winds, smaller vessels can easily be affected. While developing the dynamical system of the boat, it was discovered that the boat's lightweight and flat bottom made the wind forces significant enough to affect its position even at low wind speeds [4]. Therefore, it is reasonable to model the wind force and implement it into the dynamical system.
The marine dynamics book written by Fossen [10] proposes a model for how wind forces affect a moving vessel at sea. While the book's model is 6DOF, the current model of the Milliampere as proposed in [4] is only 3DOF. Because this project uses the model proposed in that paper, the unused variables of the wind forces are thus ignored. The forces from the wind on the sea vessel in 3DOF then becomes:

$$\tau_{\text{wind}} = \frac{1}{2}\rho_a V_{rw}^2 \begin{bmatrix} C_X(\gamma_{rw}) A_{Fw} \\ -C_Z(\gamma_{rw}) A_{Lw} \\ C_N(\gamma_{rw}) A_{Lw} L_{oa} \end{bmatrix} \tag{2.25}$$

Be aware that since the simulated environment uses a different coordinate system than the one in the research paper, $Y$ has been swapped with $Z$ in equation 2.25. $\gamma_{rw}$ is the relative wind speed's angle of attack on the vessel. $A_{Fw}$ and $A_{Lw}$ are the frontal area of the sea vessel and the lateral area of the sea vessel respectively.
The parameters $A_{Fw}$, $A_{Lw}$, and $L_{oa}$ used in equation 2.25 are found in [4] by performing analysis of a 3D model of the Milliampere ship. They represent the frontal area, longitudinal area and the length of the vessel.

$V_{rw}$ is the relative wind speed defined by

$$V_{rw} = \sqrt{u_{rw}^2 + v_{rw}^2} \tag{2.27}$$

where the components of the wind speed are defined as

$$\begin{aligned} u_{rw} &= u - u_w \\ v_{rw} &= v - v_w \end{aligned} \tag{2.28}$$

The angle of attack is given by

$$\gamma_{rw} = -\operatorname{atan2}(v_{rw}, u_{rw}) \tag{2.29}$$

The air density $\rho_a$ in Equation 2.25 varies by temperature and height. The air density at sea level is provided in Table 2.2 at certain temperatures.

| $^\circ$C | Air density, $\rho\,(\text{kg}/\text{m}^3)$ |
|---|---|
| $-10$ | $1.342$ |
| $-5$ | $1.317$ |
| 0 | $1.292$ |
| 5 | $1.269$ |
| 10 | $1.247$ |
| 15 | $1.225$ |
| 20 | $1.204$ |
| 25 | $1.184$ |
| 30 | $1.165$ |

**Table 2.2:** Air density relative to temperature. Courtesy of Fossen[10]

The coefficients in equation 2.25 are given by

$$\begin{aligned} C_X(\gamma_{rw}) &= -\underbrace{CD_l\frac{A_{L_w}}{A_{F_w}}}_{CD_{lAF}}\frac{\cos(\gamma_{rw})}{1-\frac{\delta}{2}\left(1-\frac{CD_l}{CD_t}\right)\sin^2(2\gamma_{rw})} \\ C_Z(\gamma_{rw}) &= CD_t\frac{\sin(\gamma_{rw})}{1-\frac{\delta}{2}\left(1-\frac{CD_l}{CD_t}\right)\sin^2(2\gamma_{rw})} \\ C_N(\gamma_{rw}) &= \left[\frac{s_L}{L_{oa}} - 0.18\left(\gamma_{rw}-\frac{\pi}{2}\right)\right]C_Y(\gamma_{rw}) \end{aligned} \tag{2.30}$$

$CD_l$ in equation 2.30 is the longitudinal resistance coefficient.

$$CD_l = CD_{l_{AF}}(\gamma_{rw})\frac{A_{F_w}}{A_{L_w}} \tag{2.31}$$

$CD_{l_{AF}}(\gamma_{rw})$ is a non-linear variable which varies depending on the wind's angle of attack, where 0 is the wind coming from the front, and $\pi$ is the angle when the wind comes from behind. The parameters for the vessel can be chosen in Table 2.32. As the vessel this project is working on is a small ferry for pedestrians, the most sensible choice will be the parameters suitable for a ferry.

With these parameters and with the wind speed and direction provided, one can then insert the wind force into the dynamical model as a part of the input force $\tau$.

| Type of vessel | $CD_t$ | $CD_{l_{AF}}(0)$ | $CD_{l_{AF}}(\pi)$ | $\delta$ | $\kappa$ |
|---|---|---|---|---|---|
| 1. Car carrier | $0.95$ | $0.55$ | $0.60$ | $0.80$ | $1.2$ |
| 2. Cargo vessel, loaded | $0.85$ | $0.65$ | $0.55$ | $0.40$ | $1.7$ |
| 3. Cargo vessel, container on deck | $0.85$ | $0.55$ | $0.50$ | $0.40$ | $1.4$ |
| 4. Container ship, loaded | $0.90$ | $0.55$ | $0.55$ | $0.40$ | $1.4$ |
| 5. Destroyer | $0.85$ | $0.60$ | $0.65$ | $0.65$ | $1.1$ |
| 6. Diving support vessel | $0.90$ | $0.60$ | $0.80$ | $0.55$ | $1.7$ |
| 7. Drilling vessel | $1.00$ | $0.70 - 1.00$ | $0.75 - 1.10$ | $0.10$ | $1.7$ |
| 8. Ferry | $0.90$ | $0.45$ | $0.50$ | $0.80$ | $1.1$ |
| 9. Fishing vessel | $0.95$ | $0.70$ | $0.70$ | $0.40$ | $1.1$ |
| 10. Liquefied natural gas tanker | $0.70$ | $0.60$ | $0.65$ | $0.50$ | $1.1$ |
| 11. Offshore supply vessel | $0.90$ | $0.55$ | $0.80$ | $0.55$ | $1.2$ |
| 12. Passenger liner | $0.90$ | $0.40$ | $0.40$ | $0.80$ | $1.2$ |
| 13. Research vessel | $0.85$ | $0.55$ | $0.65$ | $0.60$ | $1.4$ |
| 14. Speed boat | $0.90$ | $0.55$ | $0.60$ | $0.60$ | $1.1$ |
| 15. Tanker, loaded | $0.70$ | $0.90$ | $0.55$ | $0.40$ | $3.1$ |
| 16. Tanker, in ballast | $0.70$ | $0.75$ | $0.55$ | $0.40$ | $2.2$ |
| 17. Tender | $0.85$ | $0.55$ | $0.55$ | $0.65$ | $1.1$ |

(2.32)

## 2.5 Reinforcement Learning

Reinforcement learning(RL) is a sub-category of machine learning (ML). Unlike supervised learning, one of the main categories of ML, a RL problem does not have a proper "solution." In, for example, a supervised classification problem, an algorithm has to classify a collection of images if they, for example, feature a pony or not. The training set contains pictures where if a picture contains a pony or not is already known. The algorithm then learns to look for patterns that indicate a pony in the picture. In this example, there is a defined answer to a given problem. This is usually not the case in RL. Instead, one has a reward system where one is rewarded for correct behavior and punishment for bad behavior. Say you, for example, have an artificial intelligence(AI) controlling a robot. You want this robot to reach the peak of a mountain as fast as possible. There is no logical way to provide this AI with an "answer" to this problem with supervised learning. Instead, you can use RL and reward the AI for moving the robot higher up. It is also common to provide a constant negative reward to foster efficiency. An AI reaching the peak will thus receive a higher score for reaching the mountain peak faster. RL is thus a great way to find solutions to a non-trivial task where the answer cannot be appropriately defined.

Unity Technologies has created its own RL toolbox to make RL easier to implement in the Unity Engine while at the same time keeping it versatile, feature-rich, and open source. This toolbox is called “Unity Machine Learning Agents Toolkit"[11], but is often abbreviated to "MLagents", which will be what it will be called in this thesis. The toolbox handles the RL, the interfacing between Unity and Python, and simplifies the handling of the agent’s observation and action vector. Image 2.7 illustrates how the MLagents framework is built up and interacts with the learning environment. Because MLagents is open source, one can also make low-level changes in the code to tailor the toolbox to one’s needs.

The main policy gradient method used in MLagents is the Proximal Policy Optimisation (PPO) method developed by OpenAI[13]. This is also the policy deemed the most suitable for developing a RL neural network for an autonomous ship according to Larsen[6]. MLagents

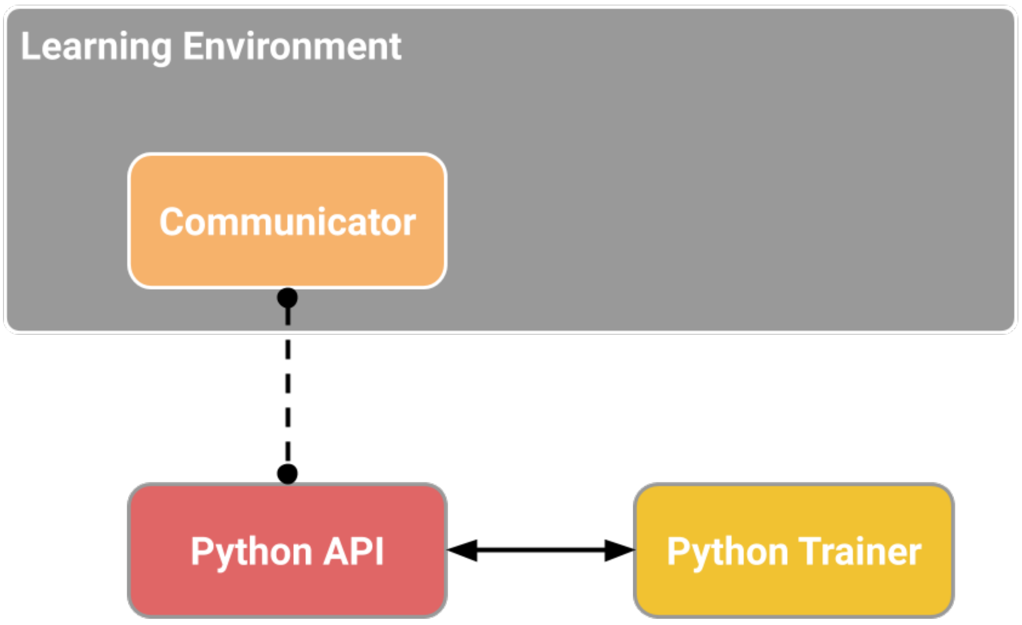


**Figure 2.7:** The MLagents framework overview. Image borrowed from Unity Technologies[12]

also does by default provide the off-policy method Soft Actor Critic (SAC), which also provided good results in Larsen's article [6], but tends to be more unstable. SAC is also able to train on old data and could be reasonable to switch to in case the gathering of training data is too slow.

One major part of RL is the design of the reward function. A proper design of the reward function is essential for the performance of an algorithm and an improper definition of the function can lead to an exploitation of the reward function where the algorithm finds a loophole to maximise the reward in an unintended way. An example of this can be seen in OpenAI's article [14], where the reward function used was the score in the boat racing game Coastrunners 7. While the score did not directly correlate to winning the race, items providing score was spread along the course. Collecting these items could lead to progression in the course and could lead to a neural network trying to complete a course as fast as possible. Unfortunately the agent found an area where the agent could collect an infinite amount of points and thus receive a high reward while not progressing the race at all. A proper design of the reward function is as indicated by this example crucial for the performance of the agent to make sure "loopholes" are unlikely to happen.

For remembering the previous actions, Long Short-Term Memory (LSTM) can be used in the reinforcement algorithm. LSTM was originally introduced in 1997 [15] as a way for algorithms to solve the issue of signals in back propagation being lost over time. One can thus introduce memory cells, input gates, and forget gates which lets the neural network chose which signals that are important to memorise or get rid of. LSTM can thus get utilised for autonomous vehicles so it can remember what consequences its previous actions had and act accordingly. LSTM is implemented in MLAgents and is readily available to be used.

## 2.6 AIS boat data

To add to the digital twin aspect of the project, one can add ship traffic to the simulation. Adding real-world ship traffic to the simulation will provide a more challenging learning environment for the autonomous Milliampere ship. Tracking data of all commercial ships above 15 meters and civilian ships above 45 meters are freely available on BarentsWatch [16]. Barentswatch also provides an API for easy access for developers. All of this tracking data is provided through the Automatic Identification System(AIS). AIS is a communication standard developed by the International Maritime Organisation (IMO), a sub-organization of the United Nations. The purpose of the AIS is to help ships avoid collisions by making their presence known to other ships. The standard requires all commercial boats carrying passengers, international boats above 300 gross tonnes, or boats in national waters above 500 gross tonnes to carry an AIS transponder [17]. In addition to being a requirement for these boats, the AIS system is also freely voluntary for all other ships.

The AIS standard transmits the ship's position in GPS coordinates, ship name, a unique ship ID called MMSI (Maritime Mobile Service Identity), ship type, velocity, heading, and size. Additional info can also be provided, like destination, but are not required. This data can be used to create a digital representation of a ship.

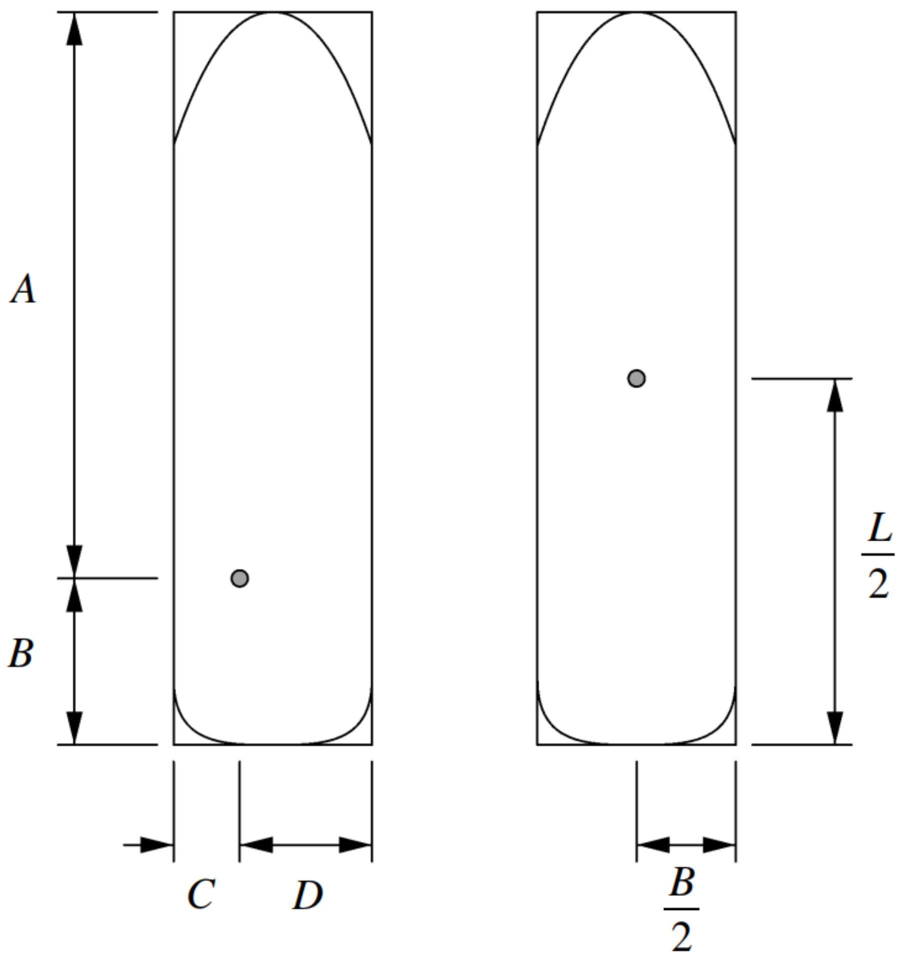


**Figure 2.8:** The dimensions A,B,C,D provided by the AIS. Courtesy of [18]

The ship dimensions are provided in AIS with the four parameters A, B, C, D. As shown in Figure 2.8, the parameters grant the size of the vessel in meters by taking into account the position of the AIS transponder. This makes sure the ship's actual position is known by offsetting the GPS coordinates provided by the transponder. Providing the ship's dimensions in a more naive way, like two parameters providing length and width of a vessel, could in some cases misposition ships by tens of meters on the largest vessels. To provide the most realistic data possible, an offset to the GPS coordinates has to be done.

$$\begin{aligned}\Delta x &= B - \frac{L}{2} \\ \Delta z &= C - \frac{W}{2}\end{aligned} \tag{2.33}$$

$$\begin{aligned}X &= X_{Unity} + \cos\theta\Delta z + \sin\theta\Delta x \\ Z &= Z_{Unity} + \cos\theta\Delta x + \sin\theta\Delta z\end{aligned} \tag{2.34}$$

One first converts the ABCD values given by the AIS system into offsets in X and Z respectively in 2.33. These are then used in equations 2.34, where $\theta$ is the ship's heading given by the AIS. The heading given by AIS is zero when the ship points straight North. The equation also assumes in the equations that $X_{Unity}$ and $Z_{Unity}$ are already converted into meters in Unity with equation 3.4 and 3.5.

# 3 Methodology

## 3.1 Implementation of ship traffic

Being able to detect and avoid static obstacles like land can be a challenge in itself, while detecting and avoiding moving traffic can be much more challenging for a RL algorithm. To make the environment as realistic as possible, real ship data is being used to create obstacles that reflect their real counterpart.

Implementing the ship traffic was done by accessing Barentswatch's API [16]. The API required first a token before accessing the AIS data. The token lasts for 3600 seconds and has to be refreshed once the time runs out. After being granted the token, one now has the ability to send requests to the Barentswatch server and retrieve data. While the token runs out after 3600 seconds, it has been decided to refresh the token a couple of minutes before it actually expires just to keep a margin of error. This makes sure the token is much less likely to expire while being used. The token's expiry time is checked every time it is being used for retrieving AIS data as a fail-safe.

```
https://www.barentswatch.no/bwapi/v2/geodata/ais/openpositions?Xmin=9.841570&
Xmax=11.224862&Ymin=63.315710&Ymax=63.817964
```

The request in 3.1 returns all of the open AIS ship data in a selected rectangular area. The data provides coordinates in GPS coordinates. Using this request only works if you also provide BarentsWatch with your access token.

The data was provided in a JSON format, which is a normal way for transferring data to and from servers. The JSON file was converted into an array of objects. Each object from the JSON file then contains all of the necessary information one needs to instantiate the objects in Unity. The objects also contain additional information and variables that are unused right now, but are readily available for future projects. Each object is then instantiated as a game object that is represented on the map. A game object in the Unity engine can be described as an object placed in the virtual world. These objects usually contain the necessary information from the AIS for the gameobjects to behave properly, but backup values are used instead in case the AIS for some reason is lacking critical information. This is usually the case when the boats are moored or anchored while they keep the AIS transponder turned on.

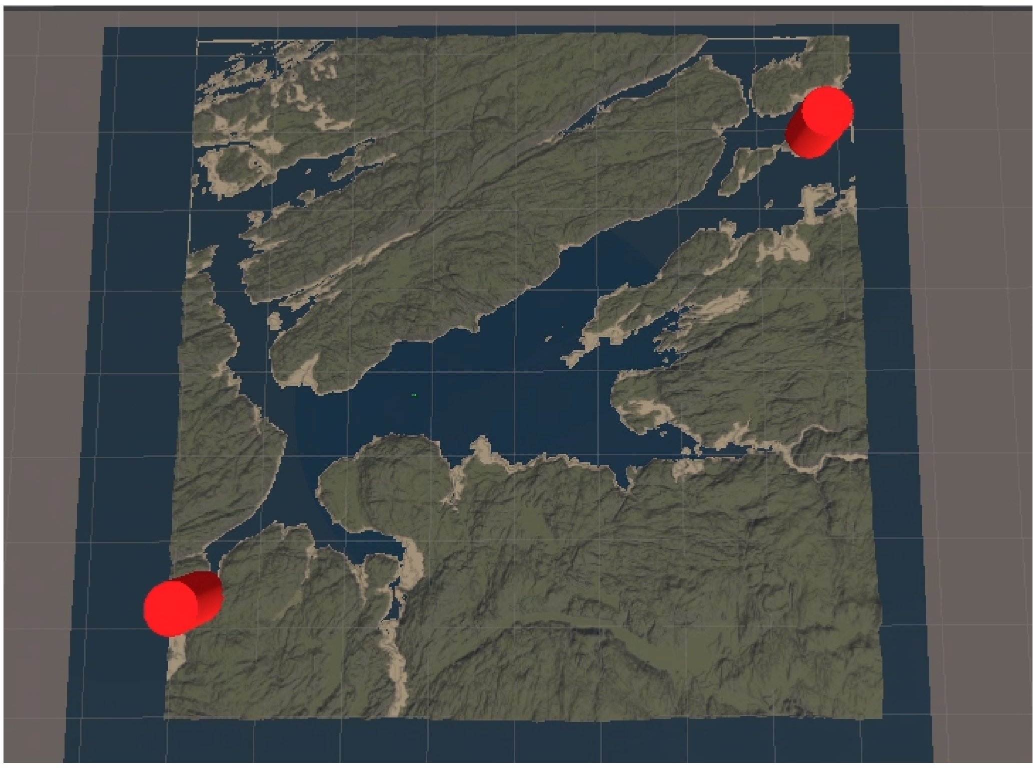


**Figure 3.1:** The simulation with red reference markers. Their size has been increased for visibility

One issue with the coordinates provided by the AIS is that the coordinates in Unity do not correspond with the GPS coordinates. One thus has to map the GPS coordinates to Unity coordinates. Two approaches were attempted to fix this. It was first attempted to use two reference points that were placed on the map as shown in figure 3.1. The reference points contain values for the position in the game engine and the real life coordinates, which had been obtained by finding the GPS coordinates on the marker spots chosen in Google Maps. The mapping was then done for both the x dimension and the z dimension.

$$Map = \frac{GPSEngine_{High} - GPSEngine_{Low}}{GPSReal_{High} - GPSReal_{Low}} \tag{3.1}$$

$$Position = (AISP - GPSReal_{Low})Map + GPSEngine_{Low} \tag{3.2}$$

In formula 3.1, the relation between the GPS coordinates provided by the AIS and the engine coordinates of the markers are done. For clarity, $GPSEngine$ are the *game engine* coordinates of the red markers, while $GPSReal$ are the *real world GPS* coordinates of the red markers. This relation is then used in formula 3.2. In this formula, The difference between the GPS coordinates from AIS, $AISP$, and the GPS coordinates of the lower marker, $GPSReal_{Low}$, are multiplied with the $Map$ constant. The lower marker's engine position is then added to get the proper conversion of the GPS coordinates into Unity engine coordinates. This mapping assumes the curvature of the earth is negligible and that the mapped area is locally flat. While relatively simple to implement, it was discovered that the map of the project was too large for this assumption. Indeed the curvature of the Earth is significant enough to deviate the position of the boats by several meters in positions far away from the reference points. A linear mapping was then deemed to not feasible for this project. A different approach was instead done to ensure an accurate environment.

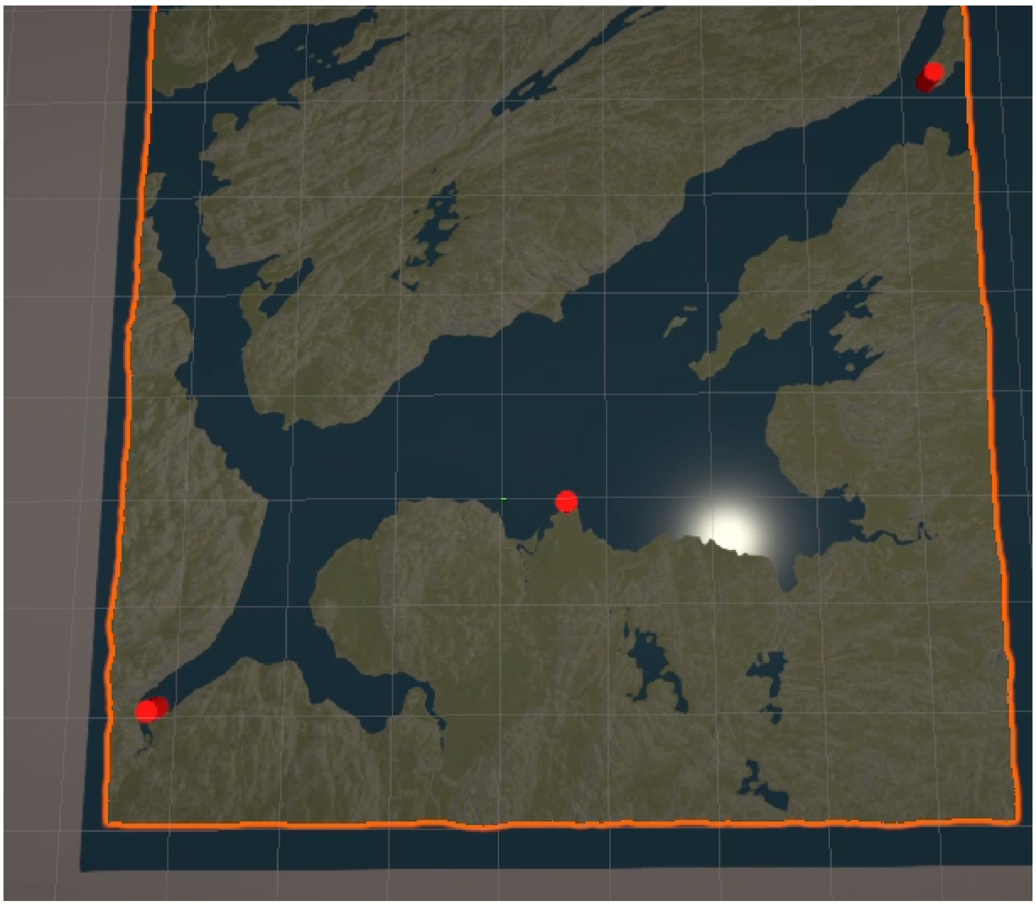


**Figure 3.2:** Three red markers for GPS mapping. Their size has been increased for visibility

Because the Earth's surface is curved, it is safe to assume that one can map the GPS coordinates to Unity coordinates with a quadratic function. A third point on the map was placed in the middle of the map. Three points were placed as shown in Figure 3.2, were used to calculate the quadratic function as it then only has one unique solution. The GPS coordinates and the position in Unity was then used to calculate a quadratic linear regression. Do note that these functions have to be updated if one desires to change the map in any way.

$$Y = A + Bx + Cx^2 \tag{3.3}$$

For this specific project, the quadratic equations for mapping the position to Unity units is as follows. This regression assumes the origin lies in the middle of the map and that the map has a width of 81920 units.

$$Unity_x = -472256.0099033 + 25735.9430697x + 1824.409308939x^2 \tag{3.4}$$

$$Unity_z = -126048400 - 4109910.1z + 33460.137z^2 \tag{3.5}$$

Do note there are other ways of translating the real position to the equivalent position in Unity. A map at a larger scale could benefit by representing the objects in a spherical coordinate system. This is for example the obvious choice if one were to represent the entire planet. The position of the boats are updated every time the simulation receives new GPS data. The GPS position is almost never 100% accurate and will usually cause some sudden movements whenever the position is updated. To prevent this, the position is linearly interpolated between the old and new position to ensure a smooth transition between the old position and the updated position.

## 3.2 Terrain Generation

Proper terrain representation is important when it comes to visualisations that represent the real world. Improper texturing can lead to incorrect representations of the terrain and a lack of visual appeal. Terrain generation has been made sure to be made properly both to maintain the details of the real world and to make the visualisation appealing and presentable for an audience with limited knowledge of the project. As I have been approached by several people at the university on how to properly represent real world terrain in Unity, parts of this subsection will contain a detailed guide on how to do this. The guide in this subsection is the result of trial, exploration, and dead ends. Its purpose will be to reduce the time spent on terrain generation in future Unity projects.

The old project used a low resolution 8-bit RAW file to generate the terrain. This made the terrain heights extremely blocky because of the limitation of only having $2^8$ or 256 height levels. The terrain was also poorly and unrealistically textured because the colouration was done with a gradient script. The RAW file was also incredibly hard to handle in large resolutions, as one RAW file was usually close to one gigabyte and would crash Unity. Because of the large size of the RAW file, one was limited to using low resolution heightmaps.

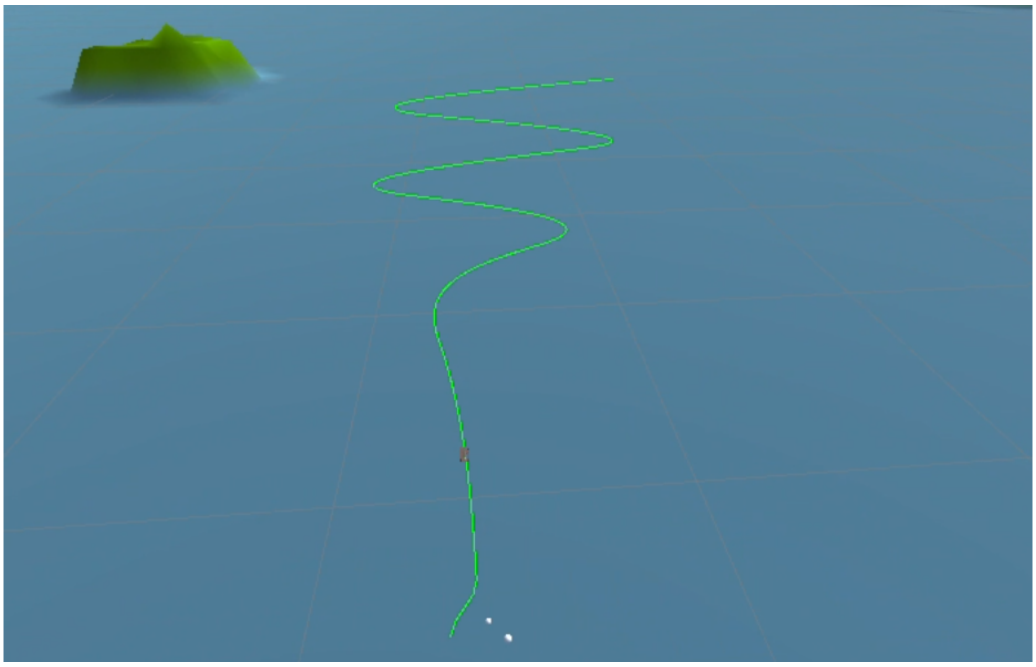
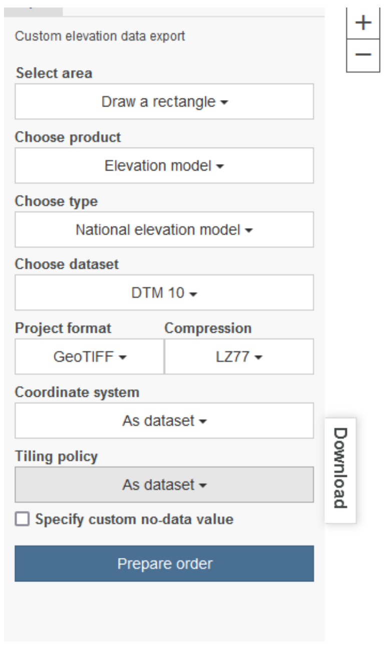


**Figure 3.3:** The old map

As shown in figure 3.3, the textures were poor and the terrain had rather pointy features. The ocean in this picture was also not really an ocean, but the lowest level of the terrain. As a result of this, the water colour can be seen "climbing" up the island in figure 3.3. Because of this, a new approach to generating terrain had to be done after the specialisation project.

This part of the terrain generation serves a detailed guide for generating real world terrain, and is the best approach found. Terrain data for Norway was gathered by using Kartverket's Høydedata pages [19].



Custom elevation data export

Select area: Draw a rectangle

Choose product: Elevation model

Choose type: National elevation model

Choose dataset: DTM 10

Project format: GeoTIFF

Compression: LZ77

Coordinate system: As dataset

Tiling policy: As dataset

☐ Specify custom no-data value

**Figure 3.4:** The options used for the map data

Figure 3.4 displays the options used for the elevation data. "Elevation model" provides literally the elevation model instead of a point cloud. "National elevation model" is then the option used in the "type" category. This option combines the different height model measurements into one large map. The dataset chosen can vary depending on what kind of project one is looking for, but the one chosen for this project's terrain was DTM10. There are two categories of elevation models featured in Kartverket; DTM and DSM. The main difference between them is that DTM (Digital Terrain Model) measures the height of the terrain without any artificial constructions or trees, while DSM (Digital Surface Model) measures the terrain height with buildings and trees included. DSM could be used in this project to display the various cities and towns in the terrain, but the texturing method used in this project makes this difficult. It is also unsure whether it will have a large performance impact. The number after DTM represents how many square meters one datapoint represents, which means DTM1 will be very detailed at a resolution of 1 square meter per data point, while DTM50 will be not detailed at all. If one represents the data height as a 2D image, each datapoint will be represented by a pixel. DTM10 was thus chosen for this project because it is in a sweet spot between being detailed while not impacting performance too much. The project format was then chosen to be GEOTIFF which is the standard file format used for height maps.
After the data has been downloaded, it has to be processed. The data given is likely divided into several square tiles depending on how large the terrain is. These square files could actually be imported into Unity as separate terrain tiles, but it was determined to be better to combine them. The tiles have different heights and will not be tiled seamlessly in Unity, which means combining the files is better. The amount of work required also decreases significantly if one only work with a single large tile. The merging of the tiles was done by using the program QGIS [20]. QGIS is a program that is used for visualising geographical data. The data it can visualise is everything from wind data and temperature, to height maps and cityscapes. It also uses GDAL [21] for processing the height maps. The .tif files were imported and merged with the raster merge tool in QGIS. It is extremely important to merge the files in the proper data type. UInt16 provides a much higher resolution than UInt8, which is what Unity defaults to if the provided files are not UInt16. If done correctly, one should be provided with a merged dataset. This dataset is then converted to PNG by using the following GDAL command.

```
gdal_translate -ot UInt16 -of PNG  -scale InputFile.tif Destination.png
```

The command should give a PNG file that can be displayed in any image viewer. The image can now be cropped to the desired selection in any image editing program that supports handling 16 bit image files. Image editing programs that support 16 bit images are quite rare, and the best option found was Photoshop. It is important to crop the image to a square image of a size that fits $2^x$, where x is an integer, like 2048, 4096, 8192, 16384. The image is now ready to be used in Unity.

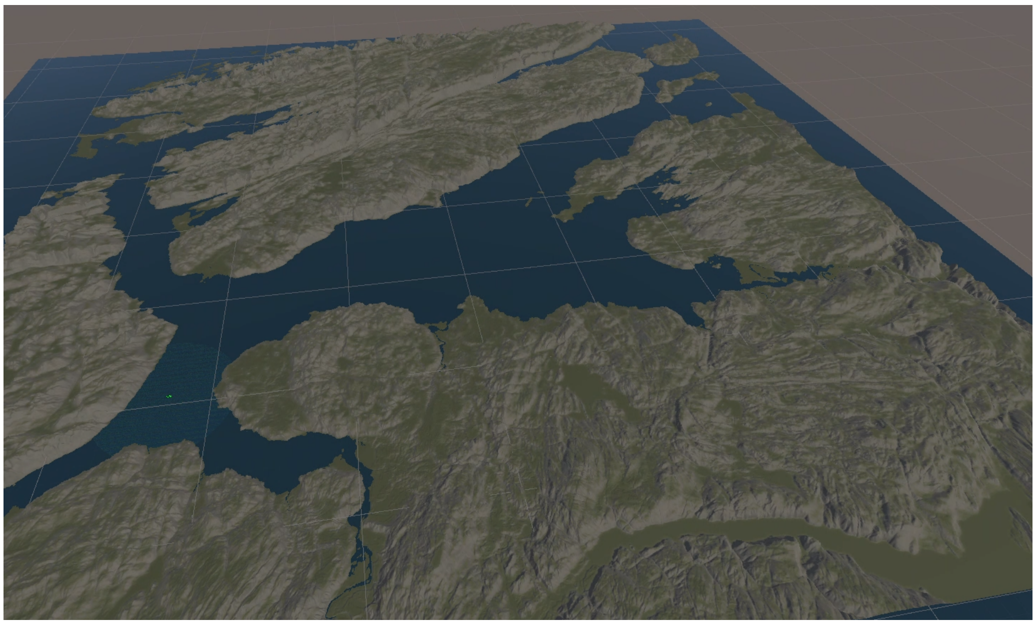


**Figure 3.5:** The new map

For importing the heightmap, the Gaia library for Unity[22] was used. This library is used by professional developers to develop good-looking maps in games and simulations. It was chosen because of its ability to handle other file types and for its texturing algorithm. Unity can only natively handle RAW files, which makes extremely large terrains like in this project unfeasible. Gaia fortunately is able to handle PNG files that are much easier to handle. Once imported, the image was imported by using the Gaia scanning tool. Because the data set here is DTM10, the terrain and the Gaia stamping tool has to be scaled by 10 for the pixels to represent 10 units in Unity. This is optional, but it is to make sure one unit in Unity is one meter. Applying the terrain can be quite difficult if one upscales an already large image. This project had a problem where the terrain itself was too large for the map stamper to handle because of Unity's use of 32bit float position. As the map had to be scaled 10x for each unit to represent one meter, the map ended up becoming a total of 81920x81920 units, which Unity was unable to handle out of the box. Everything thus had to be offset so the map's position also uses the negative values. In other words, the world origin has to be placed in the middle of the map for it to work properly when the map is too large. Texturing of the map is also handled by Gaia. Once configured properly, the map got a texture that looks much more realistic than the old project did in figure 3.3. The old map used a gradient, while Gaia's texturing algorithm takes into consideration the height and slope of the map. For water, a simple blue plane was used to represent water far away. A more detailed water provided by Gaia follows the Milliampere boat. It is purely visual. It was attempted to let the more detailed water stretch all across the map, but the increased amount of polygons decreased the performance.

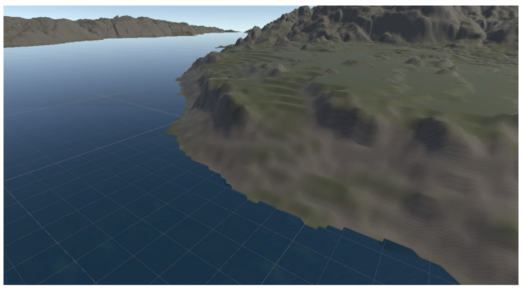


**Figure 3.6:** Closeup of the new map

Image 3.5 and 3.6 shows the new map. The map now looks significantly more realistic with proper environmental textures and smoother surfaces. The contours of the sea are still jagged like in 3.3, but that is mostly caused by the resolution of the heightmap data from Kartverket. Any higher resolution than the one provided here would be unfeasible at this scale because of the storage and computing power required. For smaller projects with a smaller area, a DTM1 dataset should be considered. This concludes the terrain generation section and its detailed guide for importing real world terrain in future projects.

## 3.3 Weather system

Wind is important to take into account when one wants to recreate the conditions at sea. A ship has to for example compensate for a strong side wind unless they want to be pushed sideways and potentially miss their destination. It is thus an important factor to add to an environment if one wants to develop a reinforcement algorithm that works in the real world. To add more realism, it has been decided to use real time data given by the Norwegian Meteorological Institute(MET)[23]. They provide weather data of the entire Scandinavia. This data is freely available for everyone and is updated about once per hour. The data sets used are the post-processed data called "MET Post-Processed Products". They are the ones recommended for this project's use-case and provide variables that are statistically adjusted and in a resolution of one square kilometer[24]. This dataset provides 12 variables which includes temperature, wind speed, wind direction, altitude, precipitation amount, cloud coverage, air pressure, and more. This project could have gone for the more advanced data set, but the data set's size of several gigabytes makes it not as feasible for a real-time system that needs to update several times a day. On the other hand, this should not be a problem provided one has a good internet connection and wishes to use this data for a more advanced weather model. The wind system will only use wind direction and wind speed, but using the rest of the variables provided by MET for further development should be a trivial task. One limitation to this approach is the fact that wind is not static, and the wind data received from MET only has a resolution of one hour. One can use the pre-processed data to create a proper wind model that realistically shows how the wind changes over time in the fjord, but that is beyond the scope of this project. One can also use historical data and create a Weibull distribution together with an Autoregressive Moving-Average model to estimate wind speeds between the datapoints[25]. This data will then unfortunately also be based on data that has a resolution of one hour, and data with a higher resolution is most likely required to create a proper model of the wind. A more simple approach is taken instead by keeping the original value and adding uniformly distributed noise to it. An uniform distribution means the values in the distribution have an equal chance of being selected randomly. In this case it is assumed a uniform distribution between -10% and 10% of the wind speed will make the wind speed dynamic *enough* for the simulated environment.

$$W = W_{data} + \mathcal{U}_{[-10\%,10\%]} \tag{3.6}$$

Equation 3.6 shows the equation that is used every five seconds to keep the wind speed random. The direction of the wind is kept noise-less and equal to the data provided by MET.

In figure 3.7, one can see streaks in the air. This is the visual representation of the wind data given by MET. The streaks of wind move in the correct direction and with correct speed.

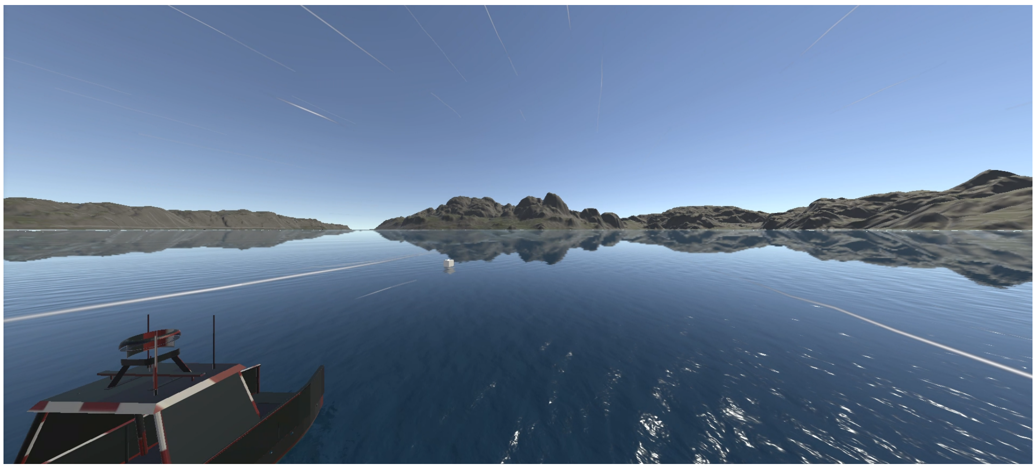


**Figure 3.7:** The wind is visualised with white moving streaks

The visual representation has no physical effect on any objects and are purely for looks and to make it easier for the user to see which direction the wind blows in.
In addition to the wind system, visualisation of the rain has been implemented as well. The data given by MET also provides precipitation in the desired area, which one can use to simulate rain. The data is sent into a particle system to control the amount of rain around the agent. While this data is used purely for aesthetic reasons at the moment, one can use this data for other purposes. Proposed uses are for example to make the amount of rain impact the dynamical model of the ship in some way, like weight increase or change in friction from the wind. This can for example be used in a digital twin approach as well, where this digital twin knows how the rain is impacting the dynamical behaviour of the ship and uses the information to feed it back to the physical system. The weather data also provides temperature, where one can determine when freezing rain can accumulate and impact the behaviour of the agent.

A day-night cycle was also implemented by simply comparing the current time of the day and adjusting the position in the sun accordingly. This is simply a calculation where one assumes the sun moves smoothly 360 degrees around the map. A more advanced system can be implemented if one desires to add a more realistic lighting system. A more realistic lighting system can be desirable if one for example would like to add solar panels on the boat and let the sun impact the behaviour in a digital twin approach.

## 3.4 Ray Perception

Detecting obstacles is crucial when it comes to navigating safely on the sea, as not being able to detect them will obviously lead to accidents on the sea. One can imagine that navigating without object detection is like if a captain would navigate by looking only at the ship's GPS and not at the view from the ship's cockpit. It is therefore obvious that an autonomous vessel needs something equivalent to our eyes to detect hazards. The feedforwad neural network developed in the specialisation project[3] had no obstacle detection and was purely developed with path following in mind. It had in other words no ability to detect hazards and no collision behaviour. To expand this NN, being able to detect obstacles must be implemented. To detect obstacles, a ray perception sensor was added to the RL system. The ray perception sensor is a part of the MLagents framework and is Unity's equivalent of a real life Light Detection And Ranging(LIDAR) sensor. The sensor emits vectors from a specified origin point and in a certain direction to then measure how far the vector is cast before it hits an obstacle. This is then repeated until the sensor has created an array of distance measurements. Unity

also supports sphere casting, a technique similar to ray casting where it checks if a sphere traveling along a vector hits any obstacles. This is useful for reducing observation space, but it will not be done in this project. The reason for this is that we would like the sensor in Unity to resemble the LIDAR on the Milliampere as closely as possible and therefore use normal ray casting. The perception sensor in the MLAgents library unfortunately does not support editing of the observed values it outputs. The sensor writes automatically the observed values to the observation vector, which is not the desired functionality. The sensor also does not support more than 100 ray casts from one module, one could add more sensors instead to compensate, but the main issue where the component writes to the observation is still there. One could edit the MLagents library to accommodate the features needed in this project, but then a custom version of the library would be needed in case this project needs to be used on a different machine. So instead it was determined to be much safer to instead make a new ray casting sensor from the ground up.
As one can see in image 3.8, the rays are emitted from the agent and displays all ray collisions

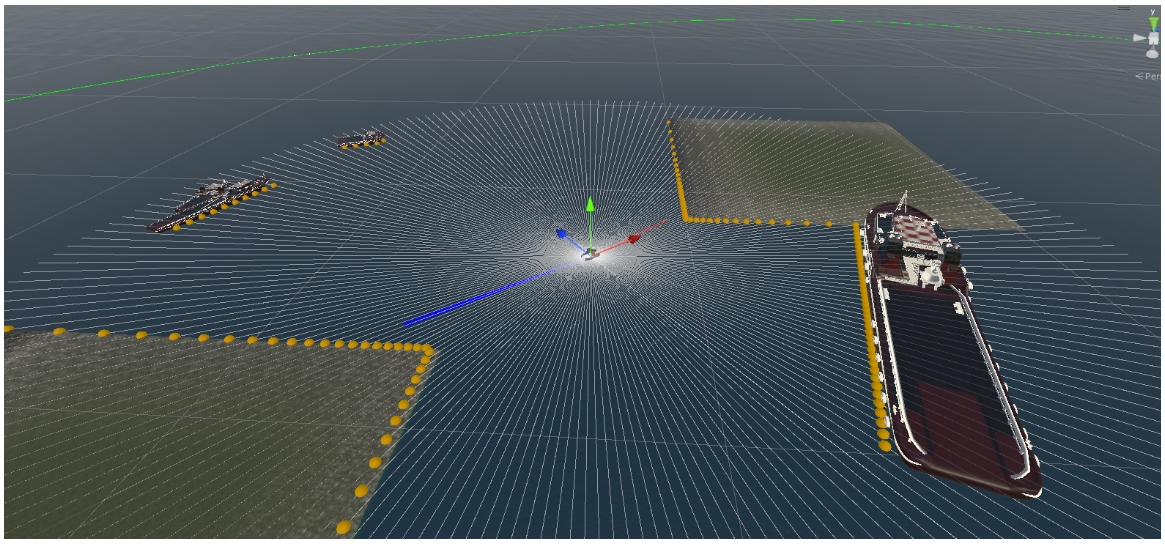


**Figure 3.8:** Screenshot of the ray perception sensor

with a sphere. These spheres are purely visual for the sake of debugging and for demonstrational purposes. These rays and spheres are in addition not visible if one for example uses Virtual Reality(VR). Displaying these rays are demanding for the program, so using them outside of demonstrations and debugging is discouraged. One challenge with using ray casting with a sufficient amount of resolution, is to handle the curse of dimensionality. As the amount of variables increase, the training time for reinforcement algorithms increase as well. To keep the observation vector length as short as possible, the rays are divided into sectors. By using Algorithm 3 from Article [26], one can check whether a sector is "feasible" or not for the given vessel. This is done by checking if the boat can fit between obstacles. If it is able to find a passage between obstacles, then the sector will show the value of the longest ray. If it does not find any fitting passage between the objects in a sector, the sector will instead report back the shortest ray length. By doing this instead of using the values of each individual ray vector, one can reduce the amount of variables. In this case the length of the observation vector from the ray sensor is reduced from 220 to 9, where 9 is the amount of sectors. The time it takes to train the neural network thus will be significantly reduced compared to using the ray casts directly.
The ray casting sensor was fully remade from the ground up and currently supports an infinite amount of rays for measuring distance. This is much more flexible than the MLagents ray perception sensor which only supports up to 100 rays. The ray cast indexing can also

be toggled to be either clockwise or counter-clockwise. This was done to provide additional flexibility to the sensor. Each ray is able to check if they collide with an object that is tagged with a specified tag. For this purpose the rays detect only objects with the tags "obstacles" and "land". The reason this is implemented is to make sure the rays do not collide with the agent the rays are emitted from. It in addition makes sure the rays do not detect objects that are unwanted for ray perception. There is also an angle offset for the ray casting to determine at what degrees the indexing should start at. The default ray perception sensor could do this by simply rotating the sensor component in the environment, but an additional offset option is provided in the custom sensor component just for convenience. Sectors are also implemented in the sensor, where one defines which angles each sector should contain. These sectors are then used to determine which rays are going to be grouped together and processed in the sector feasibility algorithm. At last, the processed values from the sector feasibility algorithm are sent to the MLagents interface's observation vector to be used by the neural network.

## 3.5 Reward Function

A reward function is essential for an RL algorithm. Without a reward function adequately fit for the problem, the neural network might exploit the reward function and avoid solving the problem in the expected way. Because this project is about an autonomous boat, safety is of course important and must be reflected as such in the reward function. Properly defining the reward function to make the neural network function as safe and efficient as possible is therefore an important task.

In the specialisation project [3], the reward function 3.7 was used. The reward function was derived from the article by Larsen [6]. Do note that the specialisation project did not take collision into consideration, so the boat was only rewarded for following the path.

$$r^{(t)}_{\text{path}} = \underbrace{\frac{v(t)}{V_{\max}}}_{\text{Speed term}} \cdot \underbrace{\frac{1+\cos\left(\psi^{(t)}\right)}{2}}_{\text{Heading term}} \cdot \underbrace{\frac{1}{|\epsilon^{(t)}|+1}}_{\text{Distance term}} \tag{3.7}$$

$$r_{\text{exists}} = (2\alpha_r) \tag{3.8}$$

$$r^{(t)} = \begin{cases} r_{\text{coll}} \,, & \text{if collision} \\ r^{(t)}_{\text{path}} - r_{\text{exists}} \,, & \text{otherwise} \end{cases} \tag{3.9}$$

| Scaling parameter | Interpretation | Value |
|---|---|---|
| $\alpha_r$ | Zero-reward relative speed | $0.05$ |
| $r_{coll}$ | Collision reward | $-2000$ |
| $\epsilon^{(t)}$ | Distance from path | - |
| $v(t)$ | Tangent speed vector i.r.t. path | - |
| $\psi^{(t)}$ | Angle i.r.t. forward vector | - |

**Table 3.1:** Variables and constants used for calculating the reward function

$r_{coll}$ here is much higher than what can be observed in the paper by Larsen [6]. It is actually equal to the score required for the episode to end early. This is the case because it was observed that the agent would attempt to minimise the probability of colliding with moving obstacles rather than trying to avoid them as it follows the path. Thus the agent will focus on following the path until it manages to accumulate a score higher than the score required for an episode to end before the maximum amount of steps are reached.

$r_{path}$ in 3.7 is divided into three terms; speed term, heading term, and distance term. The speed term rewards the algorithm according to how high the measured speed of the agent is in the direction of the given path. This is in this project calculated by calculating the dot product of the speed vector and the normalised tangent vector on the closest point of the path. $V_{max}$ makes sure the speed term never goes above 1. The speed term is then multiplied with the heading term, which is a maximum of 1 when the agent's heading is parallel to the path. $\phi$ in the heading term is the angle between the velocity vector $v^{(t)}$ and the path. The distance term contains $\epsilon^t$, which is the normal vector between the agent and the closest point of the path. The distance term is maximised when the agent is directly on the path. Overall, the reward function will give a maximum score of 1 in each time step. This component of the reward function remains relatively unchanged in this project compared to the specialisation project [3]. The $r_{exists}$ in 3.9 is a constant punishment factor of the reward function which has a purpose of making the NN as effective as possible and to stop it from idling unnecessarily. This part of the reward function has been derived from Meyer[27].
The new addition in this project is $r_{avoid}$.

$$r_{avoid} = -\frac{1}{Max(0.001, \epsilon_{boat} - L_{boats} - C})Max(0, cos(v_{dir}))||v_{boat}|| \tag{3.10}$$

$$L_{Boats} = \frac{L_{NearestBoat} + L_{Agent}}{2} \tag{3.11}$$

$r_{avoid}$ in equation 3.10 is slightly different from the avoidance reward in Meyer's project [27]. While Meyer uses data from the ray perception sensor to determine how close the obstacles are, this project's collision avoidance function uses AIS data combined with Unity's powerful vector calculation toolbox to determine the distance between the objects and the heading. This project's reward function for avoiding boats takes the length of the nearest boat into consideration. As the distance between the boats are calculated from the origins and not from the closest points on the hulls, this is necessary. In addition, a long boat is more dangerous to move close to as a long boat can more easily collide with the agent if it turns. The reward function also got in addition $v_{boat}$, the constant $C$, $L_{boat}$ and $\epsilon_{boat}$. $v_{boat}$ is the relative velocity vector of the nearest boat compared to the agent. As a boat in motion is generally seen as a more hazardous obstacle, the punishment for being close to the boat must reflect this accordingly. As such, a boat approaching the agent with a low speed will punish less than if it approached with a high speed. Calculating by using the relative velocity means the reward function will also punish more harshly if the agent moves towards the boat at high speed. The constant $C$ represents an adjustable safety margin between the agent and the closest boat. If the agent moves within the set meters of the closest boat, the reward function will reach its asymptotic value and will give the agent the maximum punishment possible. $L_{boats}$ is a constant which contains the longest dimensions of the nearest boat and the agent, then divided by two as shown in equation 3.11. This constant is added in this equation because the position of the boats are calculated from their origin in Unity. Adding this constant will thus set the asymptotic value of the function to be at minimum the distance where a collision is possible. Dividing by zero is of course not a smart thing to do in a programming, so a $Max$ function has been added to make sure this never happens. While possible, dividing by zero would likely never happen regardless in this case. The float variable $\epsilon_{boat}$ would have to become exactly $-(L_{boats} + C)$. Setting a limit to the denominator is still desired as then the value of $r_{avoid}$ can be controlled easier. 3.10 $\epsilon_{boat}$ is simply the distance between the nearest boat and the agent. As one wants to avoid colliding, it thus makes perfect sense to use make the punishment for the collision avoidance proportional to the distance between the boats. The heading is also used in the function to make sure the agent is punished for moving towards danger. It calculates the cosine of the direction vector of the velocity. The value is clamped such that the value never goes below zero. This is to make sure to not give a positive score for moving away from the boat, but to instead turn the punishment for moving away to zero. So to simplify the purpose of $r_{avoid}$; the length of the closest boat, the relative velocity, and the safety margin creates an area around the boat that the neural network will try to avoid. Though, moving away from the area will make $r_{avoid}$ equal to zero. Combined, $r^(t)$ turns into the new reward function equation 3.12.

$$r^{(t)} = \begin{cases} r_{\text{coll}}\,, & \text{if collision} \\ r^{(t)}_{\text{path}} + r_{\text{avoid}} - r_{\text{exists}}\,, & \text{otherwise} \end{cases} \tag{3.12}$$

It was unfortunately observed that this part of the reward function will encourage the agent to avoid other boats instead of following the given path. This observation was also noted in Larsen's article [6], but confirmed during training in this project. According to Larsen, the agent's ability to observe the surrounding obstacles and make a decision accordingly makes more sense. The agent should be able to determine how to avoid collisions through the $r_coll$ reward and not by being in the vicinity of other boats.
A very common event that occurred during training was the fact that the agent often ended up in a local maximum where the best known solution to the problem was to spin fast around as fast as possible around a point. This unintended behaviour had to be deterred, which meant the reward variable $r_{idle}$ has to be introduced. At first only the fact that the agent remained in one spot was considered. The first idea tested was to introduce a trailing sphere to the project. This trailing sphere would record the position of the agent and then reenact the movement a few seconds behind. The idea behind this was that the ball would be sufficiently behind the agent when it was in motion, and would catch up to the agent if it remained in one area. The intention was that the agent would then learn to stop spinning if it wanted to avoid being punished. The equation for $r_{idle}$ thus became:

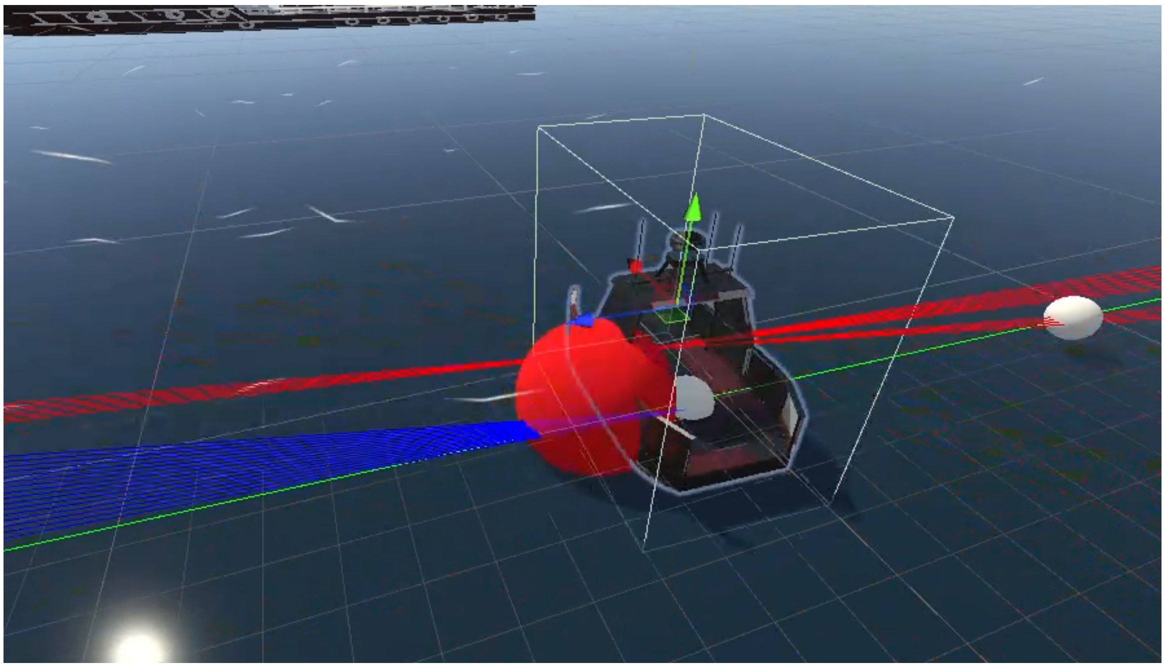


**Figure 3.9:** The trailing sphere catching up to the agent

$$r_{idle} = Min(0, \frac{\epsilon_{sphere} - r_{sphere}}{r_{sphere}}) \tag{3.13}$$

In equation 3.13, $\epsilon_{sphere}$ is the absolute distance between the origin of the sphere and the origin of the agent. $r_{sphere}$ is the radius of the sphere. The $r_{idle}$ function thus has a maximum reward of 0 and a minimum reward of -1. The reward function scales linearly proportionally to how close the agent is to the origin. This equation unfortunately was observed to not work under training, as the agent instead found a workaround and span fast like it did earlier while moving slightly faster. The agent thus moved slightly out of reach of the sphere and managed to avoid receiving a punishment from the sphere while still performing badly. The other unintended behaviour observed during training was that the agent would never attempt to return back to the path if it drifted away from it. This is likely because the trailing sphere would obviously trail behind it. This means the agent would be punished for attempting to return to where it came from. $r_{idle}$ had to be changed because of this.

$$r_{idle} = ||\omega_{agent}^2|| * \frac{1}{Max(0.001, ||v_{agent}||)} \tag{3.14}$$

Equation 3.14 was instead used. This one takes into consideration the agent's angular velocity in radians and its absolute speed. This version of $r_{idle}$ intends to punish the agent for spinning fast while at the same time not moving at a reasonable velocity. The angular velocity here is squared to make sure the reward function does not punish at lower angular velocities to any noticeable degree. To summarise, the final version of the reward function, as shown in Equation 3.15 does not feature the $r_{avoid}$ function because of the issues mentioned earlier. $r_{idle}$ is instead added to discourage idling. All this has turned equation 3.12 into equation 3.15.

$$r^{(t)} = \begin{cases} r_{\text{coll}}\,, & \text{if collision} \\ r_{\text{path}}^{(t)} + r_{\text{avoid}} - r_{idle} - r_{\text{exists}}\,, & \text{otherwise} \end{cases} \tag{3.15}$$

During the training of this problem, it was noticed that the neural network would not learn properly without multiplying $r_{\text{path}}^{(t)}$ with a constant higher than 1. The training speed improved significantly once $r_{\text{path}}^{(t)}$ was multiplied with $5$. The final reward function used for training thus becomes:

$$r^{(t)} = \begin{cases} r_{\text{coll}}\,, & \text{if collision} \\ 5r_{\text{path}}^{(t)} + r_{\text{avoid}} - r_{idle} - r_{\text{exists}}\,, & \text{otherwise} \end{cases} \tag{3.16}$$

## 3.6 Training of Feedforward Neural Network

To speed up training, one has three options; one can duplicate the environments such that one agent trains in each separate environment. The second option is to let one agent train alone, but speed up the timescale as much as it is feasible. The third option is to let all the agents train in the same environment and let each agent act as an obstacle to other agents. The first option sounds rather reasonable, but one unfortunately has to render each environment separately to make the ray perception sensor work. This obviously makes training in several environments rather taxing as one needs to render the entire map several times. Only the basic optimisation of the map has been implemented, like long distance culling, where the map has a much worse quality further away. Running several environments in parallel still is too taxing and rather difficult to set up compared to the other techniques. Option number two is very simple to implement and is the default and easiest of the techniques. The default timescale is 20x real time speed, which means the agent can gather information equal to 20 boats in real time. One issue with this technique is that the AIS boat data gathered is unfortunately live data and cannot be easily be adjusted to 20x real time speed. A different approach would then be needed specifically for the training. The third option avoids the issue of both option one and two by letting the environment only be rendered in one instance and

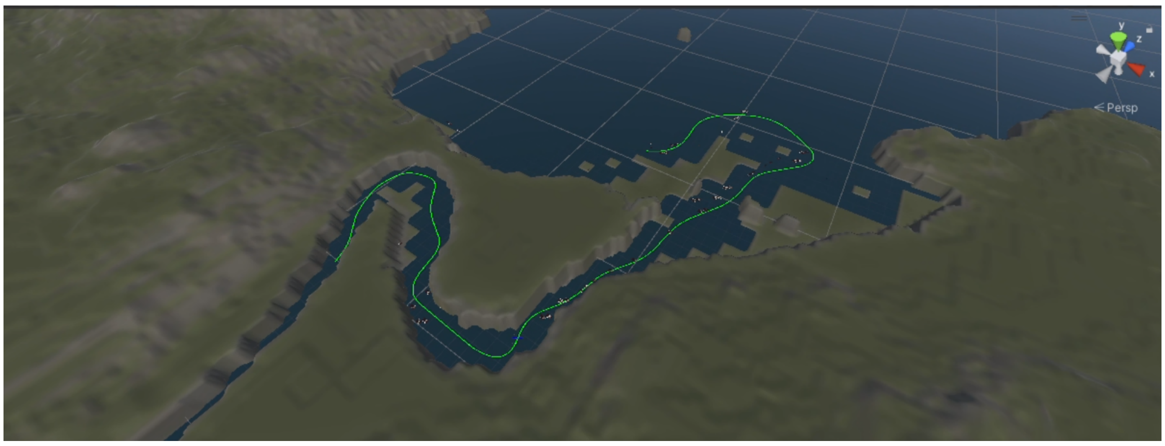


**Figure 3.10:** The training course for the agent

by making the other agents obstacles as well. This option unfortunately raises several issues that can occur. Should the agents follow the exact same path? If one provides each agent with their own path to follow, then the obstacles the AIS data provides could still be too sparse to train on not all paths can remain on the most trafficked areas. How would collision with other agents affect the path finding when they also can collide with each other? Will colliding with other agents affect training and create an unnatural bias?
These questions are hard to answer and could be worth exploring in the future, but option two seemed to be the best alternative for training the agent. Option two got fewer problems than both option one and three. The only problem that then needs to be solved is the AIS data being real time which conflicts with option two's 20x timescale.

The proposed solution is to simply spawn boats with an uniformly distributed random size heading in a random direction. These boats are spawned around the peripheral of the ray perception sensor as it is a reasonable distance and is excellent for a possible scenario where an unknown boat enters the agent's range without an AIS transponder. These boats travel in a random direction with a speed of between 0 and 6 $\frac{m}{s}$. The boats also spawn with a randomised width and length, where the boats can be between 5 and 50 meters long, and 3 to 25 meters wide. These numbers were arbitrarily chosen, but are supposed to give a decent representation of everything between the smallest boats to more normal-sized commercial ones. Do note that all of the random numbers are picked from uniform distributions, which means all values have an equal chance of being chosen. This provides the RL algorithm obstacles that might be more difficult to navigate than static ones, but will most likely train and generalise the agent faster.

As for training, it is very important to train the agent on a course that is both realistic and provides generalised results. In other words, the course needs to be made in such a manner that the agent will manage to navigate all possible paths. The course was because of this placed in the Trondheim's river, Nidelva. Nidelva has several curves and is narrow enough that the agent has to avoid colliding with the side of the river. A part of the path is also places in a more open area with little terrain nearby. Several stationary boats are placed and serve as obstacles in addition to the moving boats that approach the agent. In addition to this, the agent also starts each episode on a uniformly randomly chosen location on the curve between the start and 80% of the curve, and with a start orientation equal to the forward direction of the curve with a uniformly random value ranging from -45 to 45 degrees. Giving the agent a random starting location will in theory reduce the likelihood of overfitting, but keeping the agent's starting orientation constant relative to the path will likely increase the likelihood of overfitting. It was also observed during training that the agent had issues learning when the starting rotation was fully random. This is the reason why the range of randomness in the starting orientation is only $\pm 45\,\mathrm{deg}$.

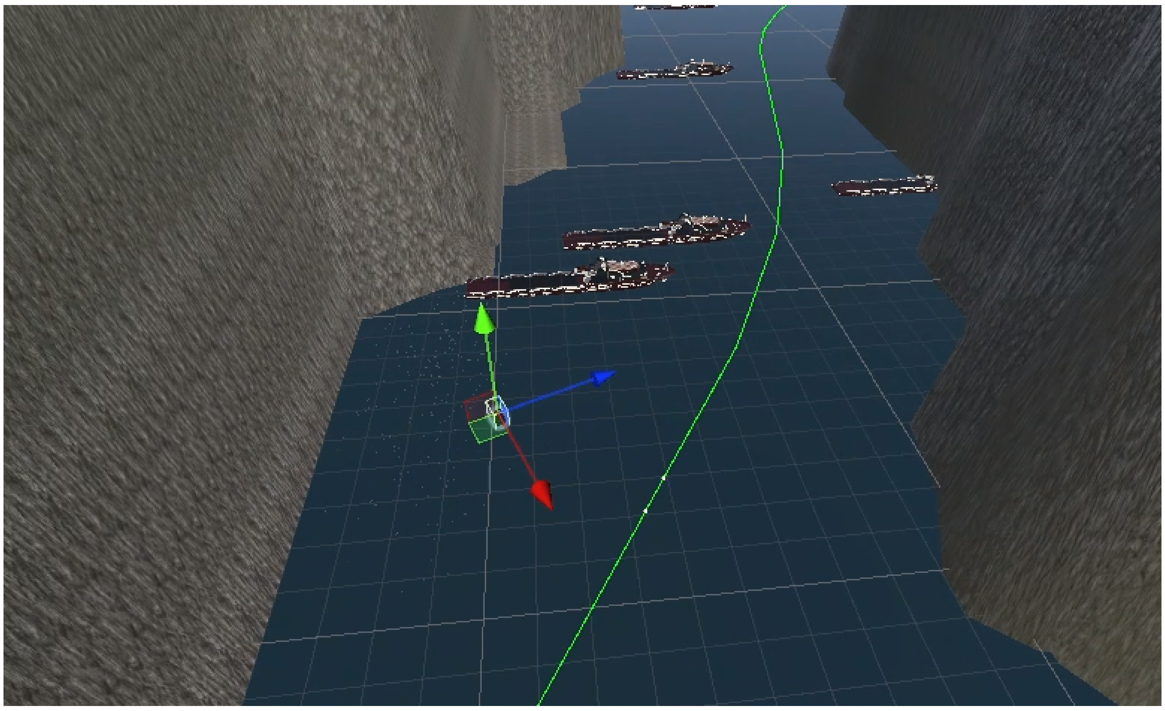


**Figure 3.11:** The agent during training

Figure 3.11 illustrates the agent training on the course. Do note that the edge of the terrain is exaggerated because of issue with the terrain generation where the lower values are improperly mapped. This can be solved with some simple adjustments, but has not been done at this stage. The obstacle boats have been placed by hand where some of them are obstructing the path. This is to make sure the agent learns to avoid the obstacles in addition to following the path. In addition to the placed static boats, random approaching boats are also spawned in. These moving boats ignore terrain and only spawn on the peripheral of the ray perception sensor. While the more realistic approach would be to make the moving boats follow the terrain in some way, this way of generating them is simpler and could perhaps make the agent take into consideration unexpected events, like when an object suddenly approaches from an area the agent previously thought only contained stationary obstacles. The randomly spawned boats were only enabled after the agent was able to follow the path. This was done because earlier iterations of the neural network likely trained poorly because of collisions in the beginning. The agent would simply spin or stand still to reduce the probability of it colliding with other boats.

The following parameters were used to train the agent:

These parameters were chosen to match the parameter values recommended by Unity's RL team [28], using the same values used in Larsen's paper [6], and by trial and error. While all of these parameters should work for any other similar project, one has to consider adjusting the time scale to a value that suits the specific project. When one trains with a too high time scale, the entire training can suffer from too large time steps, leading to simulation errors when the training is then run at a normal time scale. One also has to hide and deactivate every object and mesh that is unimportant for the training. Pointing the camera upwards and

| Parameter name | Explanation | Value |
|---|---|---|
| batch size | Amount of steps in each batch | 32 |
| buffer size | Amount of steps per model training | 1024 |
| learning rate | Gradient Descent step size | $2e^{-4}$ |
| $\beta$ | "Randomness" parameter | $1e^{-2}$ |
| $\epsilon$ | Allowed model change | 0.1 |
| $\lambda$ | Regularisation parameter | 0.99 |
| epoch | Iterations of gradient descent performed | 3 |
| hidden units | The width of each hidden layer in the NN | 64 |
| num layers | Amount of hidden layers in the NN | 2 |
| Stacked observations | How time steps used | 2 |
| Normalize | Normalise state values to be between -1 and 1 | true |
| Time Scale | How many times faster than real time | $5\times$ |

**Table 3.2:** Parameters used for training

away from most objects was discovered to lead to a large performance boost, as it would lead to objects being culled from rendering. These actions make sure the frame rate and thus the time steps in the training will remain high. For this project, a time scale of 5 combined with 140 frames per second was deemed to be enough for the training. When the agent performed well, the time scale was further reduced to 1 to make sure there would not be any difference at all between training and running in real-time.

The observation vector for the agent is as follows:

| Observation name | variable type |
|---|---|
| Signed distance between boat and path | float |
| Angle of node and boat | float |
| 2D velocity of boat i.r.t. nearest point | $2 \times 1$ float |
| Signed angle of velocity i.r.t nearest path node | float |
| Signed angle of velocity i.r.t. look-ahead node | float |
| Angular velocity of boat | $3 \times 1$ vector |
| Angle of thrusters | $2 \times 1$ vector |
| Rotational speed of thrusters | $2 \times 1$ vector |
| Ray casting sectors | $9 \times 1$ vector |

While the vector can be listed, it is hard to explain each element with one short sentence, so each element will be explained. The signed distance between the boat and nearest point of the path is interesting. While distance only makes sense to write as an absolute value, nothing stops it from containing additional info. One can in other words also add a sign to it to indicate whether the agent is on the right side or left side of the path. The angle of the boat is also important to add as an observation. It is desirable to let the agent travel forward facing the direction it should travel in. This also lets the agent eventually learn how it will move at specific angles. After that, the velocity of the agent in x and z are the next elements in the vector. These are specified in the coordinate frame of the path where $x$ is the forward direction and $z$ is the cross product of $x$ and the upwards vector. Because Unity works in a left-handed system, this means $z$ vector will point to the left. The following two signed angles of velocity are there to make sure the agent knows which direction it is traveling in relation to the nodes. Both the angles of velocity are derived from both path nodes to make sure the agent will gain information about incoming curves. At last the angular velocity of

the boat is given. This is of course to let the agent know how fast it is rotating. Then the angle and the rotational speed of each thruster are provided in the observation vector. These four values are added to the observation vector to make sure that the action space acts in a closed-loop manner and not in open loop. The background for this is that the neural network was earlier trained without observing the actual values of the thrusters, which was observed to be close to unstable. All these observations are then normalised to make sure the weights in the network do not "explode". This is fortunately dealt with in MLAgents where the mean and spread of the previous observations creates a normal distribution to normalise the values. Each episode is run for a maximum of 10 000 time steps or until the agent accumulates a total score below -2 000.
For training, this command gets used.:

```
mlagents-learn ParametersFile.yaml --run-id=NameOfRun --time-scale 5 --force
```

After the agent started to perform with satisfying results and was close to being done, the time scale was further reduced to real time to make sure the time scale would not affect the final results. In other words, the training was paused, and then resumed by using

```
mlagents-learn ParametersFile.yaml --run-id=NameOfRun --time-scale 1 --resume
```

It was then trained until the results would no longer improve. The –resume command is extremely important to use here, as using –force will delete all the training data and start the training all over again.

During training, .onnx files will be created at set intervals. .onnx files will also be created at the end of training. The Onnx file essentially represents the ML model and can be easily imported into other projects. The opposite is true too. An RL model can be imported from a different program and then used in Unity.

## 3.7 Virtual Reality

Digital projects are traditionally always displayed on a normal computer monitor. While this visualisation technique works, it could be more immersive. Virtual Reality, often abbreviated to VR, is a very sensible approach for making a visualisation more immersive. VR has been attempted for centuries. The first concept for VR can be traced back to 1838 when Charles Wetstone discovered that one could place two pictures next to each other from different perspectives to create a 3D effect [29]. This idea was then used later to create "virtual tourism" where one could visit famous spots all over the world by viewing stereoscopic images. Fast forward to recent times, modern computing power and technology makes it possible to wear a headset that enables the user to view, interact, and move in a digital environment. It makes sense to attempt to use this technology in this project to create a more immersive and appealing experience. It is also in addition a visualisation tool worth exploring as modern VR is now powerful and cheap enough to be used for scientific purposes. As seen in the article by Durukan [30], the majority of the scientific studies about VR that fulfills their criteria are less than ten years old. This indicates that the use of VR in science is still being explored and will further increase in popularity in scientific articles.

Unity has two libraries for VR support; one supports OpenXR and the other one supports OculusXR. OculusXR is the interface Meta's VR headsets use for interacting with applications, while OpenXR is the interface almost every other VR product uses. This project uses OpenXR, but it is trivial to add support to OculusXR as well. The VR headset being used is the HTC Vive Cosmos. This VR headset was chosen as it is the headset that is available for the developer of this project.

**Figure 3.12:** HTC Vive Cosmos. Image courtesy of HTC Corporation

For adding OpenXR to the project it was simply a matter of installing the OpenXR library by using Unity's asset manager and adding the OpenXR camera rig. For this project it was added in such a way that the user was standing on the boat.

The user can freely move around on the boat and receive a nice impression of being a passenger on the boat. During testing it was noticed that the user can be afflicted by a long-lasting motion sickness. This could be happening because the vessel moves on its own, but could also be caused by tracking issues during testing. Moving the user in VR outside of normal user movement should generally be avoided as it makes the visual input of the user not correspond with what the user is actually feeling in real life. Shorter sessions of a couple of minutes gave no ill effects to the user except for some slight discomfort for a few seconds after removing the VR headset. Apart from the motion sickness, adding VR made the user feel like a passenger of the boat, while viewing on a monitor did not. When viewing from the monitor, the experience was more like viewing through a camera. Using VR is therefore seen as a great addition to the visualisation and should be considered for similar projects.

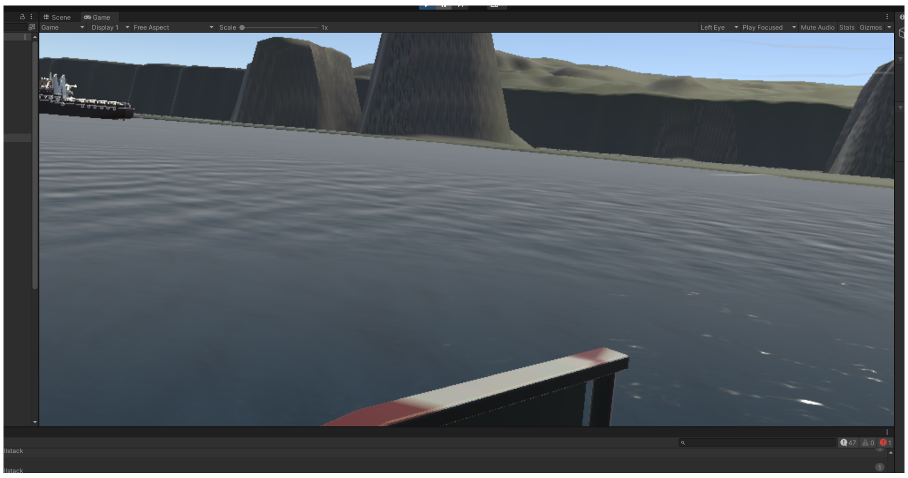


**Figure 3.13:** Screenshot of the visual output going to the VR headset.

The VR module was later removed to reduce potential conflicts between the VR camera and the normal camera. As said earlier, adding the VR module back to the project is trivial and can be easily done in the future.

## 3.8 Floating Point Error

In the specialisation project [3], it was observed that the Unity engine contained limitations in positional accuracy. Unity stores the position of objects in a 32 bit floating number. As one moves far away from the origin point of the world, the floating number has to contain fewer digits behind the decimal point to store larger numbers. This is usually not noticeable in the majority of cases where one does not let an object stay far away from the origin, but this project's map is tens of kilometers wide. As a result of this, the agent had issues maintaining an acceptable accuracy at positions far away from the origin while traveling. The most noticeable part of this inaccuracy was the vessel's inability to move at a low velocity. While staying 5000 units away from the origin, it would simply not move at all while moving lower than 0.05 units per second. While this sounds insignificant, it was quite noticeable. As the position is stored in a vector containing three elements, each element could have different amounts of positional accuracy depending on the agent's position relative to the origin. Say the agent's position was [10,0,5000], then the agent would seemingly move just fine in the x axis, but would seemingly refuse to move in the z axis until it had gained enough velocity in the z direction. This lead to the agent moving in straight lines when it was obviously supposed to be moving diagonally. This could be fixed by increasing the size of the time step between each physics update, but this would affect accuracy yet again. In the specialisation project, this issue was avoided by simply letting the agent train close to the origin point. This was possible because collision was not implemented and thus the map simply did not matter.
The floating point error issue cannot be ignored in this project unfortunately. Several solutions are proposed; one can split up the map into several smaller tiles and only load a small number of them. Once the agent moves too far away from the origin, a new set of tiles are loaded and everything is moved closer to the origin. The second option is to keep the entire map as one complete map like it is right now and move the agent to the origin once it moves too far away. One can then compensate for this by shifting all of the world's objects to a new position. One can also rewrite parts of the physics engine to allow 64 bit positioning, which also fixes this problem. Out of all of these proposed solutions, the second solution was chosen. Tiling real map data was attempted and discovered to be time consuming. Tiling also requires one to "stitch" the tiles together in a seamless way, and no good solution for this issue was found. The reason why the third option was not chosen was because editing a physics engine is a rather daunting task and it is unknown how much time needs to be spent on implementing 64 bit floating points. It is also unknown as to which degree this affects the performance of the application. This could possibly be explored in a future project. The second solution instead is a rather simple solution, but a messy one. It involved restructuring the entire project so every object except for the agent became a child of a single object that in this project is called "World Anchor". Once the agent moves 5000 units away from origin, the position of the agent is set to zero and the World Anchor's position is set to be the old position vector of the agent, but with the opposite sign. Thus the relative position of everything has not changed, yet the agent will remain around the origin and maintain its positional accuracy that is needed. This will come at the cost of the positional accuracy of other objects. Objects moving far away from the agent can experience the same inaccuracy as the agent did, but the objects will be too far away from the agent to be noticeable. The floating point fix could be changed to instead change the position of every object relative to the camera, but this again could affect the accuracy of the agent. The floating point fix remained to only care about the accuracy of the agent.

# 4 Results and Discussions

While training, several iterations were made of the RL model. LSTM was at first attempted. The reason why LSTM was attempted to be incorporated into the neural network was mainly out of curiosity and to see if it would significantly improve the performance of the agent. The plan was to use LSTM to observe the past states and thus make a better decisions based on this. The idea behind this was that the RL model could use the past states to gain more information like speed, heading and interactions between the action space and observation space over time. Unfortunately it was observed that training with LSTM took a significantly longer time than by simply using a normal feedforward neural network. After a week of training, the RL model with LSTM did not perform better than what a normal feedforward neural network could achieve after a few hours of training. There are several reasons for why this could be the case, but there are two main potential culprits, one general one and one specific for this problem; a NN with LSTM simply takes significantly longer time to train because the RL model in addition to training the feedforward NN also has to train the parameters that control which past states to remember and forget. The one issue specific to this problem is the vector of values gained from the ray perception sensor or LIDAR. The LSTM is known to have issues with values suddenly changing position in the observation vector. This happens in this case when the agent turns and the obstacles observed by the ray perception sensor suddenly appear in different sectors and thus in different spots in the observation vector. This creates an issue in the LSTM where it simply does not know which elements in the past states to remember. A workaround to this issue would be to let the LIDAR maintain a constant angle relative to the world. This would make sure that the obstacles observed in the LIDAR will not change position in the observation vector anymore. Unfortunately this would create yet another issue where one cannot use differently sized sectors in the LIDAR pooling algorithm. As keeping differently sized sectors is very handy for when you want to maintain a high perception resolution in the most important direction while having lower resolutions at less important directions. If one instead has fixed sector sizes, one has to either sacrifice perception resolution by increasing the sector angles, or training speed by increasing the sector width. Sacrificing either of those is not viable in this problem, as the training speed is already rather slow and having a low perception resolution can be dangerous in the real world. It was thus decided that LSTM is not viable for this problem.
Normal feedforward neural networks were attempted again after this. MLAgents supports stacked vectors, which was determined to come in handy in this project. The stacked vectors use observations from past steps, almost like in LSTM and added them into the observation vector. The intent of this was to gain interactions between each state to provide the neural network with velocity and other time-based information. A stacked vector of three was at first attempted, but the training again was slow and the results were lackluster. A stacked vector of two, which means an observation vector with information from current step and last step were instead used. This provided a higher training speed and thus a better performance. Unfortunately the agent would most of the time "wiggle" back and forth in a borderline unstable manner. The theory behind this is that the action vector of the NN only provides the "desired" value of the angle and speed of each thruster. The NN on the other hand does not provide the actual values. From control theory, one can label this part of the NN as open-looped. The algorithm merely controls the desired value of the thrusters, but has no information of the actual state of the thrusters. It's because of this the actual angle and rotational speed is fed back into the controller through the observation vector. In addition to the open-loop issue, an issue with one of the observation vector elements was spotted and was quickly fixed. A stacked observation vector of two, which means current observation and the one previous observation, were used. As a result of this, the performance improved significantly.

## 4.1 Training Data

The training of the NN was done quite smoothly after some trial and error. As one can see in figure 4.1, the cumulative reward decreased at some points during training. Significant changes to the training were done at these times. At 7.7 million time steps, the performance of the agent was determined to be good enough. Up until this point, the agent had only been training with static objects. This is because it was observed in earlier iterations that colliding with moving objects would severely impact the training and lead to the agent reaching a local maximum where it would spin in one spot to avoid the moving boats. Moving boat obstacles were added after this point. This can be seen by the sudden drop in cumulative score in Figure 4.1. After 9 million time steps, the performance of the agent was determined to be good enough and the time scale was decreased to 1 to make the training more realistic. As a result of this, the cumulative reward for the agent increased suddenly before decreasing again. This is likely because the time steps at this point are shorter which will increase the time steps in each episode and an increase in reward. The agent also seemed to be impacted by the better frame rate of the simulation. This was discovered to be likely caused by the "max timestep" setting not being set high enough. Unity will skip physics time steps if the frame rate is too low. This will hinder the program in outputting even less frames at the cost of physics accuracy. This is a reasonable default feature to have in Unity as the main focus of the engine is for running games. Games usually do not care too much about accurate physics and can in most cases ignore physics accuracy to make sure the game runs smoothly. This feature was not discovered until after the training was done, but it has been adjusted afterwards to have a maximum time step of 1 second instead of 0.1 seconds. Future training sessions will then most likely run much more accurately. At 11 million time steps an error in the observation vector was discovered which likely impacted the behaviour. This was fixed and the agent was trained some more to make sure the fix would impact the NN. The change as seen in Figure 4.1 at 11 million time steps, lead to a sudden increase in performance. As the cumulative is seemingly increasing, it is reasonable to let it train until the score plateaus, but the training at this point in time had been significantly slowed down by setting the time scale to 1. What would take a few hours to train at a time scale of 5, now would take several days. An early stop was thus done.

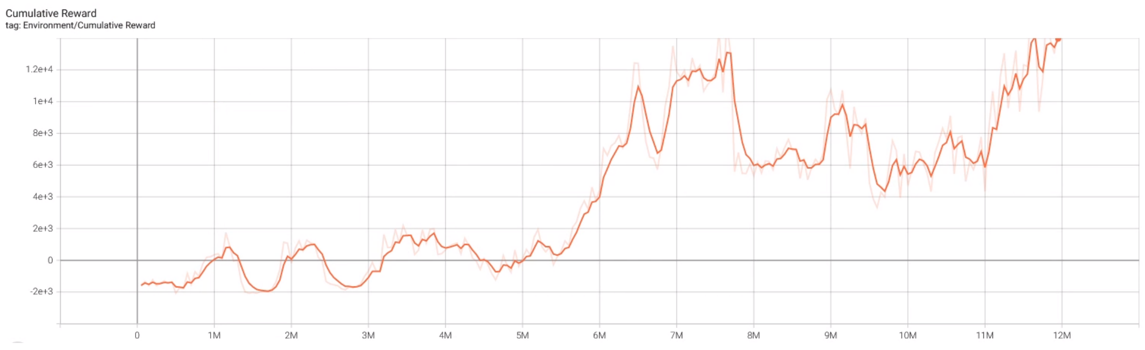


**Figure 4.1:** The cumulative reward graph provided by Tensorboard

As seen in 4.2, the addition of moving boats seemed to not have been impacting the length of each episode by much. The training would have been shorter and more efficient if the error found at 11 million time steps was found earlier, but it is unsure how impacting the error was. The error turned the element in the observation vector responsible for the agent's angle into an element where two different angles would become the same value. This happened because of a misplacement of a cosine function and would make the agent's position not fully defined.

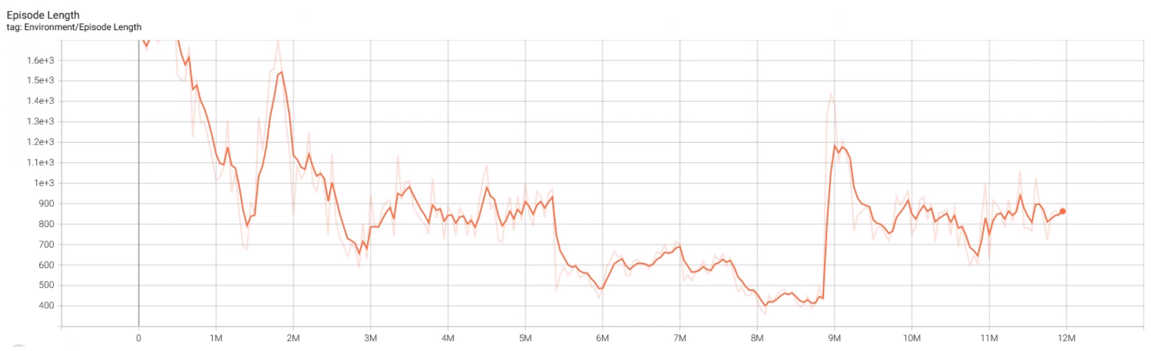


**Figure 4.2:** The episode length graph provided by Tensorboard

## 4.2 Testing of the Neural Network

A very important factor within RL is to check if the NN is generalised enough to be useful for any situation. To test this, several test courses of increasing difficulty were proposed. The agent then got at least 100 attempts to complete each course, where the score, time spent, and success rate were calculated. The reason for this was to determine the performance and the safety of the agent. A high score obviously indicates that the agent performs in an effective and safe manner. It can be argumented to be safe because the reward function sets the score to -2000 and ends the training at failed attempts. Thus being close to other vessels and acting unsafely will increase the risk of gaining a bad score, which the agent should avoid. The success rate of the agent was crucial in this case as it indicated if the proposed scenario was feasible for the agent to complete. In the real world, a failed attempt at completing a scenario might lead to hurt passengers, damages to vehicles, and possibly passengers being stuck on the vessel and being unable to reach land. These test courses were rather similar to the ones from the specialisation project [3], where they the paths had a similar shape. The length of the new paths were much longer than the ones presented in the specialisation project. This made comparisons between the new and old project possible. Unfortunately, little data from the old project was recorded. No data from the old paths were recorded. Only the actual behaviour of the agent was observed and described. The following paths were used for testing in this project; a straight path without any obstacles, a straight path with randomly placed static obstacles, a path with several turns, and finally an advanced path with several turns and both dynamic and static obstacles. Each of the obstacles are presented below in the same order as mentioned. Optimal weather conditions were provided in each environment, which means the wind speed is zero. The rain was also set to be zero to keep the frame rate at maximum. The score, time steps spent on successful attempts, success, and collisions were recorded in each test run, where the mean, standard deviance of the score and time spent, as well as the success rate and collision rate were calculated and displayed in a table. Because it was desirable to test it in a realistic environment, a time scale of 1 was used in all test environments.

### 4.2.1 Straight line with no obstacles

An approximately 500 meter long straight line with no obstacles was used for the first test.

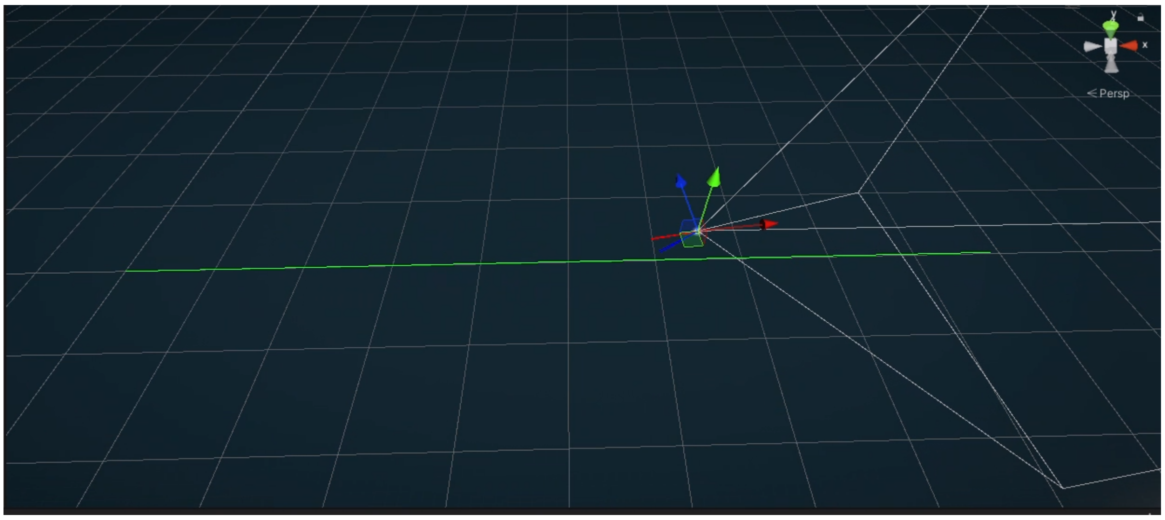


**Figure 4.3:** A straight 500 meter long path

| Score | Time Spent | Success Rate | Collision Rate |
|---|---|---|---|
| $5585{,}1291[\mu,\sigma]$ | $3970{,}606[\mu,\sigma]$ | 1 | 0 |

**Table 4.1:** Results from straight path experiment without obstacles

### 4.2.2 Straight line with obstacles

For the second test path, the same path was used, but with uniformly randomly placed obstacles. As the results from the previous experiment showed promising results, it makes sense to use the same path but with obstacles in addition so one can see how adding obstacles affect the outcome of the experiment. A random amount between 10 and 15 obstacles were placed in each episode. The placement and size of each obstacle is random.

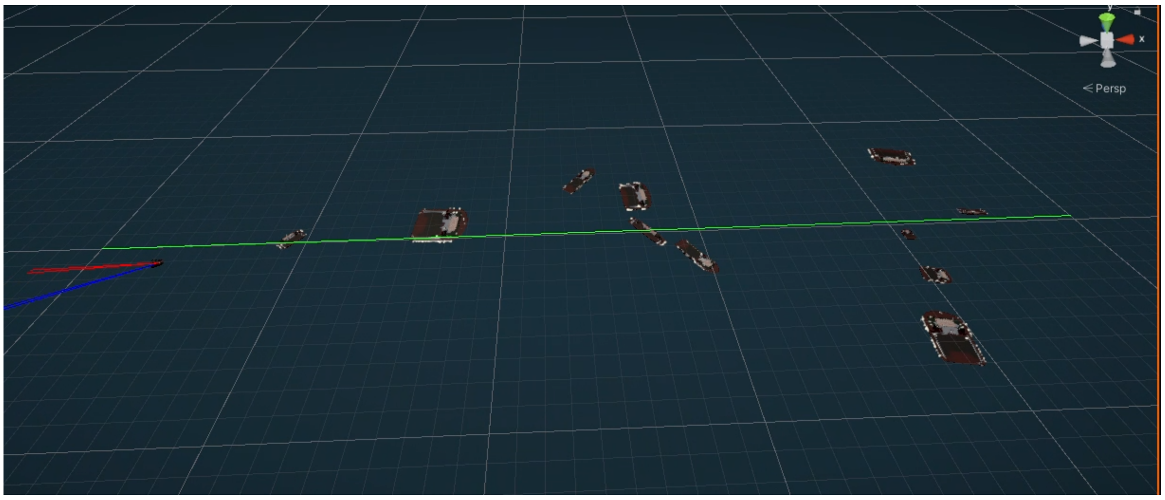


**Figure 4.4:** A straight 500m long path

107 tests are performed for this test course.

As one can see, the NN struggles to finish this course. A success rate of 17.2% and a collision rate of 82.8% means this task is too difficult for the agent.

| Score | Time Spent | Success Rate | Collision Rate |
|---|---|---|---|
| 1209,2114$[\mu, \sigma]$ | 3922,552$[\mu, \sigma]$ | 0.172 | 0.828 |

**Table 4.2:** Results from straight path experiment with obstacles

### 4.2.3 Curved path with no obstacles

This path tests the performance of the agent by only considering it has to follow a curved path. Although the performance in experiment 4.2.2 showed subpar results, it is still worth checking out if the agent will perform just as well on a path featuring multiple curves as it did in the first experiment 4.2.1. Although the distance between the start and end of the path is still 500 meters, the distance traveled by the agent is about 1 km because of the curved path. The agent here is expected to spend more time on the map. This will most likely also affect the score on each successful episode as the map is longer and will thus keep on rewarding the agent for longer. 192 episodes were performed on this specific path.

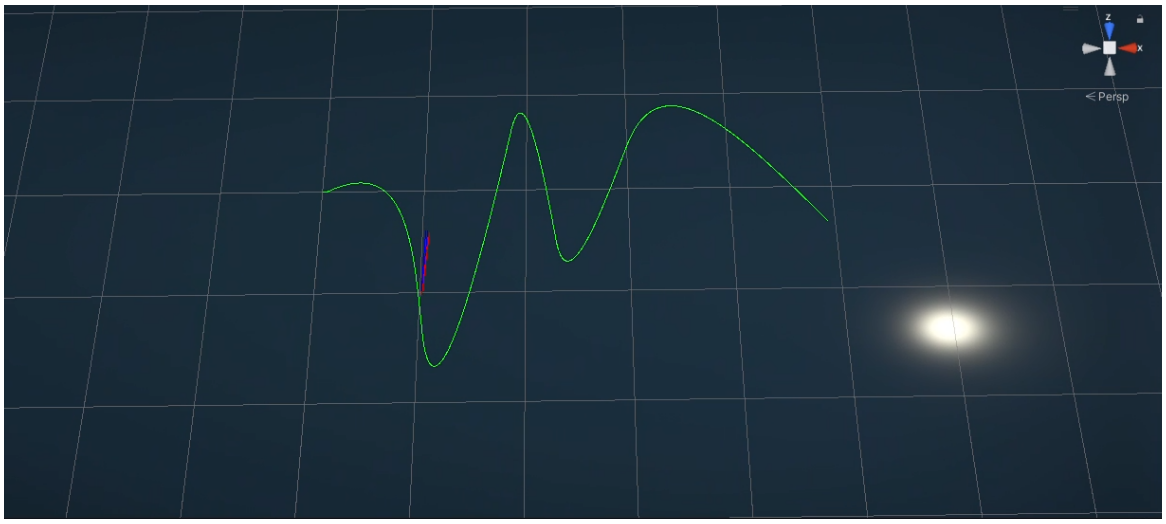


**Figure 4.5:** A curved path seen from above

| Score | Time Spent | Success Rate | Collision Rate |
|---|---|---|---|
| 7775,2733$[\mu, \sigma]$ | 8780,854$[\mu, \sigma]$ | 1 | 0 |

**Table 4.3:** Results from curved path experiment with no obstacles

The performance in this experiment shows excellent results. Although the time spent is significantly longer than in the first experiment 4.2.1, this still is acceptable because of the curve's longer and more difficult path.

### 4.2.4 Curved path with dynamic and static obstacles

The same curved path was used yet another time for the next experiment. The maximum amount of possible static vessels in each episode were reduced to 20 and the size of each were reduced slightly as well. This was done because the vessels at maximum size would in certain cases block the path severely and make the path impossible to complete.

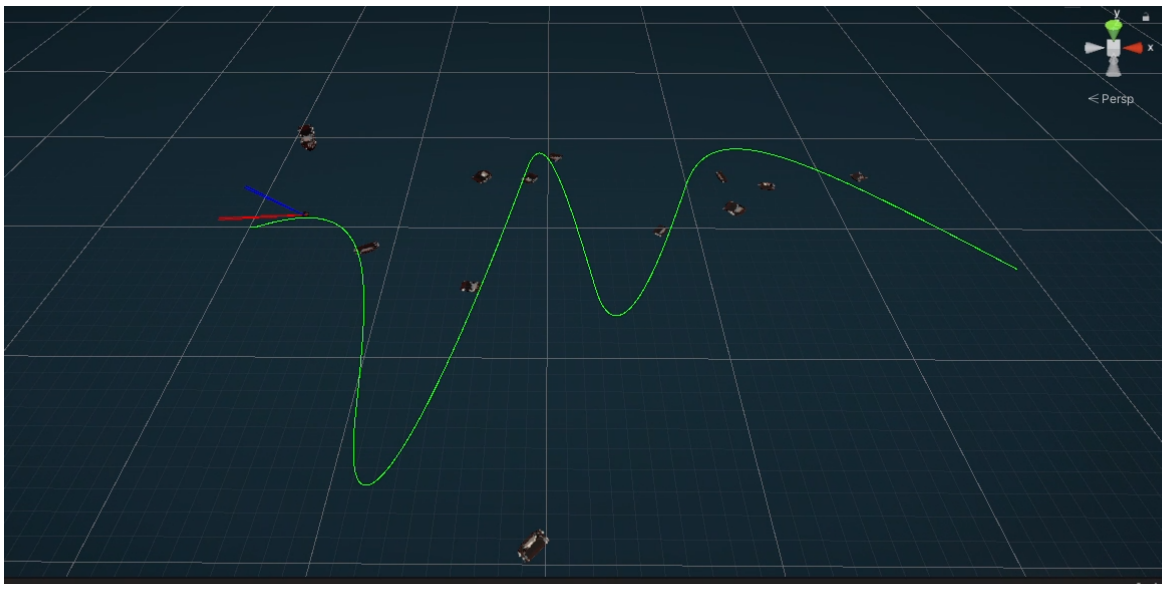


**Figure 4.6:** The same curved path as figure 4.5, but with obstacles

| Score | Time Spent | Success Rate | Collision Rate |
|---|---|---|---|
| 5173,4728$[\mu,\sigma]$ | 3106,2026$[\mu,\sigma]$ | 0.17 | 0.83 |

**Table 4.4:** Results from curved path with obstacles experiment

## 4.3 Discussion

As one can see from the results, the agent was able to follow the path quite nicely regardless of difficulty provided. The mean time spent traversing the more difficult path without obstacles is 4858 more time steps than what it would spend traversing the straight line. This is a significant increase in time steps, but an acceptable one considering the increased length the path gets from being curved. A success rate of 1 in both cases without obstacles while running over 100 tests on each is a strong indication of its capability of following paths. The performance of the old paths in the specialisation project on the other hand struggled in certain maneuvers and would often end up spinning if it deviated slightly from the path. The success rate of the specialisation project's agent was not recorded, but one can safely assume it was lower than 1.
Meanwhile the experiments with obstacles indicate that more work is required to make the agent work with obstacles. A success rate of 0.17 in both obstacle experiments shows that the current iteration of the RL model is not safe enough for being capable of transporting pedestrians in the real world. According to Kujala [31], the possibility of a collision happening when a different ship crosses a ship's path is at $5 \cdot 10^{-4}$ per encounter. This number is based on AIS data in an area outside of Finland and the encounters happening between two large ships. To compare the agent's performance, the following formula for finding the probability of a collision occurring per encounter is done.

$$1 - P = \sum_{n=0}^{x-2} p(1-p)^n \tag{4.1}$$

Here in equation 4.1, $P$ represents the probability of a test run succeeding at least once. $p$ on the other hand represents the probability of an encounter leading to a collision. This probability calculation is a series, as each test run ends if the encounter leads to a collision. The probability series is one unit shorter than the amount of possible collisions, because The data from the experiment featured in Section 4.2.4 is used in these calculations. First the amount of ship encounters are calculated. We take the the best-case estimate and assume every spawned ship crosses the agent's path. The mean spawn time is at 37.5 seconds, the fixed time step is 0.05 seconds, and the mean time spent each test run is 3106 steps. Each test run here lasted for 62 seconds, which means about 2 boats were spawned on average. In addition, static boats were placed, where the average placed was 18 boats, resulting in a total of 20 possible collisions. As each run had a 0.17% success rate, one can calculate the actual probability of a collision. Plugging the numbers into equation 4.1, one gets:

$$0.83 = \sum_{n=0}^{19} p^n(1-p) \tag{4.2}$$

Computing for $p$ by using a solver yields:

$$p = 0.085 \tag{4.3}$$

This in other words means the probability of any encounter from the experiment in Section 4.6 leading to a collision is at 0.085. This is as mentioned earlier a best-case estimate, but the estimate shows the collision rate is still magnitudes higher than the estimated collision rate shown in Kujala's works [31]. These experiments were also performed at optimal conditions, which means the wind speed was zero during the entire testing. This means the results would likely be even worse if the agent was affected by strong winds during the experiments. Its performance during training was acceptable and indicated good behaviour for evading obstacles. This could mean the training environment was not varied enough, or that the test course was too difficult compared to the training environment. These results can unfortunately not be compared with the specialisation project as obstacles were not introduced at that point.
If the error in the observation vector was detected earlier, then the performance of the agent during testing would likely be better. While the error should not have an impact on the detection of an obstacle that is either static or moving, there is a possibility that the error could have an impact on the agent's response. Because the error caused the agent's position to not be fully defined, it is likely the agent would not respond to an approaching collision in an appropriate manner. If one compares the results gained here versus the results gained by Larsen [6], the results seem to be different. In Larsen's paper, the agents would adhere nicely to the path when only static obstacles were introduced, but would start to deviate once dynamic objects were introduced. The agent in this paper was observed to be follow the path closer and more accurately compared to the PPO algorithm in Larsen's paper, but it would also not care about most of the obstacles before colliding became inevitable. As it stands now, one would have to implement a "safety filter" as seen in [32]. The safety filter checks whether the agent's expected course will be safe or not. If not, then an algorithm made by hand will take over and make sure the agent avoids dangers and can safely return to normal behaviour. It was observed during testing that the continuous action space of the neural network acted more like a discrete action space. In other words, the desired thruster force for instance was either at maximum forward thrust or maximum backward thrust. This was not the case during the first million time steps of training, where the action space was fully continuous. This means the discrete action space behaviour has been gained during training and thus determined by the NN to likely be the most optimal way of using the action space. To maintain a specific thrust force, it would instead rapidly switch between each extremes. As the agent was fully able to follow any path given, a discrete action space should be considered instead of a continuous action space to reduce the training time.

### 4.3.1 Unity Engine Discussion

This entire project was implemented in Unity Game Engine with C#. It currently will work fine with externally trained NNs as long as they are configured the way Unity currently expects them to be. This means that models can be trained in environments that are more efficient than Unity or preferred by each developer, and then exported to Unity where the environment is more visually appealing. One can also train a model in Unity directly as all of the groundwork for it has been made already. The physics calculations done in this program should also be quite accurate, but one needs to watch out when setting the timescale above 5 as the physics seem to deviate slightly from what can be seen in real time. The accuracy of the physics seem to match closely with the real world as seen in Pedersen's article [4]. With an accurate dynamical model and with capabilities of communicating with external files, means this program is well-suited for the implementation of a digital twin. The program also fetches the amount of rain, wind speed and wind direction. Other variables are also readily available to be used, like air temperature and cloud coverage, and can be used for creating more advanced weather.
The AIS representations that are used in the program right now work fine, but the AIS boats currently only travel in straight lines, where the position gets updated every to the new position in a jarring manner. This works fine for most vessels, but can be quite inaccurate if the GPS signal from the boat transponder is inaccurate or has a slow update rate. It was also noticed that the boat movements would often be quite unnatural during turning.
The ray perception sensor implemented here was implemented in a flexible and future-focused manner that makes sure alternative changes can be easily implemented, which means one can easily change the order of the rays, the amount of rays, size of different sectors, ray lengths and different sector algorithms.
The terrain for the program provides a mostly accurate representation of the real world terrain, but it does not feature any representation of artificial infrastructures or buildings. It is definitely a big improvement over the previous iteration, but there is still room for improvement, like adding a representation of cities, making certain areas more detailed and so on. As it stands right now, it is capable of being used mostly for visualisation purposes, but additional work can be done to make it more accurate. This would be at a cost of map size because of engine limitations and practicality, as an equally large map with more details would take up a lot more space and memory than what most modern computer is capable of handling. The agent should also behave the same regardless of how large the map is or where it is located now. The floating point error fix solved the issues that has been with the project since its inception.
VR was also implemented and tested. The VR visualisation was then removed after testing, but should be trivial to implement again. VR should was discovered to be a very reasonable visualisation, especially for showing people without any background knowledge in the project. This is because one becomes more immersed in the world by standing inside of it instead of looking at it through a screen.

# 5 Conclusion and future work

## 5.1 Conclusion

To conclude this project, one may look back at the objectives as shown in section 1.2.

Primary Objective:

- Lay the foundation of a digital twin with access to real world values.

Secondary Objectives:

- To develop an expandable digital framework for autonomous ship navigation
- Improve the visual appeal and accuracy of the terrain
- Fetch real world data and utilise them for calculations and visualisation
- Implement VR visualisation

One will first consider the secondary objectives in order. A more accurate dynamical model of the Milliampere boat was implemented in the project with more accurately modeled thrusters as well. The positioning and dynamics of these thrusters can be easily changed to accommodate other configurations. A flexible digital LIDAR was also created such that one can test different types of ray pooling algorithms. To test the capability of the autonomous ship navigation, a model trained using RL was used. It was tested and was proven to be able to follow any path provided according to the tests, with a completion rate of 100%. The model was also shown to have issues avoiding obstacles, like static obstacles and other boats, where the completion rate is reduced to 17% in both test paths with obstacles.
The visual appeal of the environment was also significantly improved by using a proper terrain creator, but there is still room for improvement.
Real-time data provided by BarentsWatch and Norwegian Meteorological Institute are used to generate a weather system which replicates the wind and rain occurring real time in the Trondheim Fjord. AIS data provided by BarentsWatch lets real boats be represented in the simulation. The wind currently affects the agent, while the rain is purely visual. Other weather variables and AIS data can be used for further expansion.
VR was briefly attempted and found to work fine straight out of the box. Additional functionality should be doable.
As the secondary objectives were all completed, the primary objective has been fulfilled as well. While the objectives were completed, there will always be ways to improve it. Suggested improvements have been listed in Section 5.2.

First, how usable is Unity Game Engine for simulating a realistic RL environment? Unity is generally usable for simulating a realistic RL environment. Dynamical models that are scientifically proven to be accurate are able to be easily calculated in Unity. As long as the time steps are small enough and the engine does not skip any time step, the engine will remain accurate. The repercussion of skipping frames was explained in Section 4.1.

The second question was related to the safety aspect of the trained RL agent trained in a Unity environment and implemented in the real world. One can, from the results, conclude that the current algorithm is not safe enough. Some tiny adjustments should make the model safer, but as the model is a black box, it cannot be guaranteed to be safe. Therefore, additional work is required to make the algorithm safer.

## 5.2 Further work

There are several directions to expand this project in. Some of these proposals contain new directions for the current project, or further improvement of the current systems. Remember that this is only a list of proposed tasks and is non-exhaustive. The tasks can be changed, or new ones can be added. The proposed future tasks for this project is as follows:

This project laid the foundation for a digital twin, but did not fuse together the digital twin and the physical object. Thus the most interesting and preferred direction of this project is to combine this project with a physical autonomous vehicle and use the project as a digital twin. Unity should be capable of interfacing with an external program where it can both send and receive data. This could in return be extremely useful for the Autoferry project, where data provided by the physical object can be processed and used for a digital visualisation and for detecting possible hazards that would be impossible without a digital twin. One can look at similar digital twin projects, like the one by Matulis [33], where a digital twin for a robot arm is created and used for improving its real behaviour.

Norwegian Meteorological Institute has a freely open dataset of weather data that is updated hourly with a grid size of 1x1 square kilometers. Thus the spatio-temporal resolution might be too poor. A more advanced weather system can then be implemented and used to create an even more accurate digital twin. The data used in this project has also so far used only a few of the variables the data provide. Utilising more of the data provided might turn the weather more realistic. The AIS data given by BarentsWatch is excellent for representing

boats, but the way the data provided is used can be improved. Right now the position and rotation of the boats can jump around in an unnatural way every time there is an update. The solution for this in the project was to do a linear interpolation (lerp), which hides this data update slightly. The interpolation hides the turning of boats poorly and needs to be improved. It has been suggested to use historical data to predict the movement of the boat [34].

The current map is affected by the cities not being represented on the map. As a result of this, the map resolution around cities can be quite poor. Using map data from Kartverket or OpenStreetMap can be used to generate a 3D model of the cities and towns. The maps can then be combined to create a hybrid map. The map can also be improved in other ways, like for example by developing an algorithm that smoothes the mesh so the shoreline does not look at "blocky". The current map also does not feature a proper ocean. Combining the cur-

rent height map with an ocean depth map can open up a new direction of the project where one can simulate underwater vehicles and equipment. These devices can then be used in a digital twin approach where the program fetch real time data and improves the behaviour of the underwater devices. This can for example be to use ocean current data for improved pathing both over and under water. The behaviour of the boats created by the AIS data can

be changed. The current AIS boats do not react to the agent because the agent does not exist in the real world. Turning the boats into a pseudo-real representation of the boats could be interesting and better for predicting the real world behaviour of the boats. By for example making the AIS boats yield for approaching boats could likely improve realism. The behaviour of AIS boats can then be changed to follow boating rules whenever they are near the agent.

As it currently stands, the model has issues avoiding obstacles. One can either improve the current model, or implement a safety check to override its behaviour when the model is acting dangerously. Minor changes to the training of the model as mentioned in Section 4.1 should make it behave safer, but as the model is essentially a black box, one cannot be fully certain that it will behave well in all situations. A safety check algorithm thus has to be added in addition to make sure it acts in a safe manner in all possible scenarios.

# Bibliography

[1] *Autoferry - ntnu*, `https://www.ntnu.edu/autoferry`, Accessed:2022.02.14.

[2] J. Sánchez-Beaskoetxea, I. Basterretxea-Iribar, I. Sotés, and M. d. l. M. M. Machado, "Human error in marine accidents: Is the crew normally to blame?", *Maritime Transport Research*, vol. 2, p. 100 016, 2021.

[3] S. M. Sætre, *Machine learning in unity*, Dec. 2021.

[4] A. A. Pedersen, *Optimization based system identification for the milliampere ferry*, 2019. [Online]. Available: `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2625699`, (accessed: 17.09.2021).

[5] In, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley and Sons, Ltd, 2011, ISBN: 9781119994138. DOI: `https://doi.org/10.1002/9781119994138.refs`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.refs`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.refs`.

[6] T. L. Thomas Nakken Karlsen Halvor Ødegård Teigen *et al.*, *Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters*, 2021. [Online]. Available: `https://www.frontiersin.org/articles/10.3389/frobt.2021.738113/full`, (accessed: 17.09.2021).

[7] O. San, A. Rasheed, and T. Kvamsdal, "Hybrid analysis and modeling, eclecticism, and multifidelity computing toward digital twin revolution", *GAMM-Mitteilungen*, vol. 44, no. 2, e202100007, 2021. DOI: `https://doi.org/10.1002/gamm.202100007`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/gamm.202100007`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.202100007`.

[8] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective", *IEEE Access*, vol. 8, pp. 21 980–22 012, 2020. DOI: `10.1109/ACCESS.2020.2970143`.

[9] T. Torben, A. H. Brodtkorb, and A. J. Sørensen, "Control allocation for double-ended ferries with full-scale experimental results", 2020.

[10] "Environmental forces and moments", in *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, 2011, ch. 8, pp. 187–225, ISBN: 9781119994138. DOI: `https://doi.org/10.1002/9781119994138.ch8`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119994138.ch8`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch8`.

[11] U. Technologies, *Unity machine learning agents*, `https://unity.com/products/machine-learning-agents`, Accessed: 2022.01.21.

[12] ——, *Ml-agents toolkit overview*. [Online]. Available: `https://github.com/Unity-Technologies/ml-agents/blob/release_19_docs/docs/ML-Agents-Overview.md#running-example-training-npc-behaviors`, (Accessed:05.06.2022).

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: `1707.06347 [cs.LG]`.

[14] *Faulty reward functions in the wild*, Accessed:2022.03.14, 2016. [Online]. Available: `%5Curl%7Bhttps://openai.com/blog/faulty-reward-functions/%7D`.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: `10.1162/neco.1997.9.8.1735`.

[16] *Barentswatch*, https://www.barentswatch.no/om/apnedata/, Accessed: 2022.01.16.

[17] *Evised guidelines for the onboard operational use of shipborne automatic identification systems (ais)*, https://wwwcdn.imo.org/localresources/en/OurWork/Safety/Documents/AIS/Resolution%20A.1106(29).pdf, Accessed: 2022.01.16.

[18] P. Silveira, A. Teixeira, and C. Guedes Soares, "Assessment of ship collision estimation methods using ais data", Oct. 2014, ISBN: 978-1-138-02727-5. DOI: 10.1201/b17494-27.

[19] Kartverket, *Map of norway*, 2021. [Online]. Available: https://hoydedata.no/LaserInnsyn/, (accessed: 25.08.2021).

[20] *Qgis*, https://www.qgis.org/en/site/, Accessed:2022.02.09.

[21] F. W. E. Rouault *et al.*, *Gdal - gdal documentation*, 2021. [Online]. Available: https://gdal.org/download.html#windows, (accessed: 31.08.2021).

[22] P. Worlds, *Gaia 2021*, https://assetstore.unity.com/packages/tools/terrain/gaia-2021-terrain-scene-generator-193509, Accessed: 2022.01.21.

[23] *Meteorologisk institutt*, https://thredds.met.no/thredds/catalog.html, Accessed:2022.01.29.

[24] *Meteorologisk institutt github*, https://github.com/metno/NWPdocs/wiki, Accessed:2022.01.29.

[25] R. Billinton, H. Chen, and R. Ghajar, "Time-series models for reliability evaluation of power systems including wind energy", *Microelectronics Reliability*, vol. 36, no. 9, pp. 1253–1261, 1996, ISSN: 0026-2714. DOI: https://doi.org/10.1016/0026-2714(95)00154-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0026271495001549.

[26] E. Meyer, H. Robinson, A. Rasheed, and O. San, "Taming an autonomous surface vehicle for path following and collision avoidance using deep reinforcement learning", *IEEE Access*, vol. 8, pp. 41 466–41 481, 2020. DOI: 10.1109/ACCESS.2020.2976586.

[27] E. Meyer, "On course towards model-free guidance: A self-learning approach to dynamic collision avoidance for autonomous surface vehicles", M.S. thesis, NTNU, 2020. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2780874.

[28] U. Technologies, *Unity technologies ml agents hyperparameters*, 2021. [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md, (accessed: 09.11.2021).

[29] *History of virtual reality*, Accessed:11.03.2022. [Online]. Available: %5Curl%7Bhttps://www.vrs.org.uk/virtual-reality/history.html%7D.

[30] A. Durukan, H. Artun, and A. Temur, "Virtual reality in science education: A descriptive review.", *Journal of science learning*, vol. 3, no. 3, pp. 132–142, 2020.

[31] P. Kujala, M. Hänninen, T. Arola, and J. Ylitalo, "Analysis of the marine traffic safety in the gulf of finland", *Reliability Engineering and System Safety*, vol. 94, no. 8, pp. 1349–1357, 2009, ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2009.02.028. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832009000568.

[32] K. P. Wabersich and M. N. Zeilinger, “A predictive safety filter for learning-based control of constrained nonlinear dynamical systems”, *Automatica*, vol. 129, p. 109 597, 2021, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2021.109597. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109821001175.

[33] M. Matulis and C. Harvey, “A robot arm digital twin utilising reinforcement learning”, *Computers and Graphics*, vol. 95, pp. 106–114, 2021, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2021.01.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009784932100011X.

[34] S. Hexeberg, A. L. Flåten, B. H. Eriksen, and E. F. Brekke, “Ais-based vessel trajectory prediction”, in *2017 20th International Conference on Information Fusion (Fusion)*, 2017, pp. 1–8. DOI: 10.23919/ICIF.2017.8009762.

# 6 Appendix

## 6.1 Straight Line no obstacles test data

These are the test data from each run in the straight line test.

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 7926 | 3850 | 1 | 0 |
| 5465 | 3718 | 1 | 0 |
| 6603 | 3732 | 1 | 0 |
| 5151 | 3733 | 1 | 0 |
| 5443 | 3726 | 1 | 0 |
| 5811 | 3715 | 1 | 0 |
| 5187 | 3732 | 1 | 0 |
| 5066 | 3925 | 1 | 0 |
| 4748 | 3724 | 1 | 0 |
| 4783 | 3732 | 1 | 0 |
| 5644 | 3707 | 1 | 0 |
| 5553 | 3715 | 1 | 0 |
| 6132 | 3704 | 1 | 0 |
| 5919 | 3736 | 1 | 0 |
| 5094 | 3726 | 1 | 0 |
| 5794 | 3718 | 1 | 0 |
| 6385 | 3733 | 1 | 0 |
| 5660 | 3718 | 1 | 0 |
| 4446 | 4625 | 1 | 0 |
| 3678 | 5554 | 1 | 0 |
| 2384 | 5501 | 1 | 0 |
| 2877 | 5651 | 1 | 0 |
| 7149 | 5697 | 1 | 0 |
| 8612 | 5960 | 1 | 0 |
| 12604 | 5510 | 1 | 0 |
| 7377 | 5327 | 1 | 0 |
| 5545 | 4618 | 1 | 0 |
| 5673 | 3736 | 1 | 0 |
| 6087 | 3717 | 1 | 0 |
| 5401 | 3707 | 1 | 0 |
| 6314 | 3710 | 1 | 0 |
| 5070 | 3729 | 1 | 0 |
| 5622 | 3721 | 1 | 0 |
| 5395 | 3717 | 1 | 0 |
| 5528 | 3726 | 1 | 0 |
| 6356 | 3727 | 1 | 0 |
| 5509 | 3731 | 1 | 0 |
| 4916 | 3721 | 1 | 0 |
| 5306 | 3723 | 1 | 0 |
| 5134 | 3737 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 5654 | 3715 | 1 | 0 |
| 5262 | 3740 | 1 | 0 |
| 4804 | 3729 | 1 | 0 |
| 5398 | 3727 | 1 | 0 |
| 5243 | 3727 | 1 | 0 |
| 5151 | 3745 | 1 | 0 |
| 5178 | 3731 | 1 | 0 |
| 5175 | 3721 | 1 | 0 |
| 4704 | 3736 | 1 | 0 |
| 5332 | 3730 | 1 | 0 |
| 5140 | 3723 | 1 | 0 |
| 5480 | 3707 | 1 | 0 |
| 5918 | 3727 | 1 | 0 |
| 5563 | 3727 | 1 | 0 |
| 5518 | 3713 | 1 | 0 |
| 4398 | 3746 | 1 | 0 |
| 5318 | 3735 | 1 | 0 |
| 5457 | 3715 | 1 | 0 |
| 6031 | 3716 | 1 | 0 |
| 5387 | 3731 | 1 | 0 |
| 5329 | 3729 | 1 | 0 |
| 5494 | 3719 | 1 | 0 |
| 5819 | 3726 | 1 | 0 |
| 5797 | 3717 | 1 | 0 |
| 6472 | 3727 | 1 | 0 |
| 6491 | 3751 | 1 | 0 |
| 5122 | 3709 | 1 | 0 |
| 5515 | 3731 | 1 | 0 |
| 6325 | 3723 | 1 | 0 |
| 5828 | 3714 | 1 | 0 |
| 5014 | 3754 | 1 | 0 |
| 5019 | 3714 | 1 | 0 |
| 5636 | 3721 | 1 | 0 |
| 5237 | 3704 | 1 | 0 |
| 6134 | 3726 | 1 | 0 |
| 5070 | 3726 | 1 | 0 |
| 4915 | 3740 | 1 | 0 |
| 4874 | 3719 | 1 | 0 |
| 4920 | 3737 | 1 | 0 |
| 5392 | 3719 | 1 | 0 |
| 5350 | 3736 | 1 | 0 |
| 5608 | 3727 | 1 | 0 |
| 5106 | 3723 | 1 | 0 |
| 4904 | 3728 | 1 | 0 |
| 6174 | 3723 | 1 | 0 |
| 5015 | 3740 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 5186 | 3738 | 1 | 0 |
| 6405 | 3701 | 1 | 0 |
| 6399 | 3718 | 1 | 0 |
| 5269 | 3763 | 1 | 0 |
| 5547 | 3738 | 1 | 0 |
| 5907 | 3721 | 1 | 0 |
| 4997 | 3737 | 1 | 0 |
| 5544 | 3721 | 1 | 0 |
| 4765 | 3711 | 1 | 0 |

## 6.2 Straight line with obstacles test data

The data from the straight line test with obstacles. A total of 116 runs.

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 821 | 904 | 0 | 1 |
| 546 | 951 | 0 | 1 |
| 1794 | 2556 | 0 | 1 |
| 493 | 2631 | 0 | 1 |
| -523 | 3039 | 0 | 1 |
| 2415 | 1463 | 0 | 1 |
| 2788 | 2725 | 0 | 1 |
| 5418 | 4722 | 1 | 0 |
| 401 | 1317 | 0 | 1 |
| 3570 | 3109 | 0 | 1 |
| 218 | 1565 | 0 | 1 |
| 4573 | 3738 | 1 | 0 |
| 1206 | 2628 | 0 | 1 |
| -36 | 2692 | 0 | 1 |
| 3482 | 6077 | 1 | 0 |
| 347 | 3176 | 0 | 1 |
| 3239 | 3226 | 0 | 1 |
| 5202 | 2921 | 0 | 1 |
| 303 | 1100 | 0 | 1 |
| -495 | 965 | 0 | 1 |
| 4658 | 3727 | 1 | 0 |
| 6224 | 3771 | 1 | 0 |
| -656 | 637 | 0 | 1 |
| -196 | 1365 | 0 | 1 |
| 235 | 1398 | 0 | 1 |
| 5979 | 3727 | 1 | 0 |
| 328 | 1426 | 0 | 1 |
| -63 | 1540 | 0 | 1 |
| 585 | 2509 | 0 | 1 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| -60 | 1245 | 0 | 1 |
| -60 | 1083 | 0 | 1 |
| 5306 | 3785 | 1 | 0 |
| 306 | 1958 | 0 | 1 |
| -564 | 761 | 0 | 1 |
| 617 | 2563 | 0 | 1 |
| 598 | 1817 | 0 | 1 |
| -155 | 1694 | 0 | 1 |
| 146 | 2062 | 0 | 1 |
| 2583 | 3145 | 0 | 1 |
| 4651 | 3501 | 0 | 1 |
| 4847 | 3747 | 1 | 0 |
| 151 | 1906 | 0 | 1 |
| -3 | 1044 | 0 | 1 |
| -114 | 2215 | 0 | 1 |
| 2639 | 2885 | 0 | 1 |
| 3419 | 3308 | 0 | 1 |
| 284 | 1079 | 0 | 1 |
| -58 | 1604 | 0 | 1 |
| -68 | 2046 | 0 | 1 |
| -98 | 833 | 0 | 1 |
| 4942 | 3739 | 1 | 0 |
| -565 | 607 | 0 | 1 |
| -223 | 530 | 0 | 1 |
| -248 | 466 | 0 | 1 |
| -50 | 1949 | 0 | 1 |
| -672 | 524 | 0 | 1 |
| 5297 | 3744 | 1 | 0 |
| 631 | 1232 | 0 | 1 |
| 99 | 1069 | 0 | 1 |
| -366 | 1104 | 0 | 1 |
| -249 | 640 | 0 | 1 |
| -619 | 1549 | 0 | 1 |
| 315 | 2418 | 0 | 1 |
| -193 | 696 | 0 | 1 |
| -292 | 634 | 0 | 1 |
| 252 | 1640 | 0 | 1 |
| -805 | 465 | 0 | 1 |
| -60 | 1632 | 0 | 1 |
| -486 | 662 | 0 | 1 |
| 4911 | 3734 | 1 | 0 |
| -55 | 1073 | 0 | 1 |
| -543 | 900 | 0 | 1 |
| -68 | 1371 | 0 | 1 |
| -350 | 1077 | 0 | 1 |
| -419 | 876 | 0 | 1 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| -427 | 780 | 0 | 1 |
| 5490 | 3737 | 1 | 0 |
| -132 | 616 | 0 | 1 |
| 205 | 1074 | 0 | 1 |
| 6600 | 3736 | 1 | 0 |
| -584 | 826 | 0 | 1 |
| 176 | 2503 | 0 | 1 |
| -463 | 745 | 0 | 1 |
| 3289 | 3076 | 0 | 1 |
| 392 | 1294 | 0 | 1 |
| 4601 | 3762 | 1 | 0 |
| 210 | 2212 | 0 | 1 |
| 5467 | 3734 | 1 | 0 |
| -419 | 564 | 0 | 1 |
| -34 | 1222 | 0 | 1 |
| -82 | 854 | 0 | 1 |
| -227 | 1251 | 0 | 1 |
| 2433 | 3149 | 0 | 1 |
| 290 | 792 | 0 | 1 |
| -616 | 580 | 0 | 1 |
| 4413 | 3772 | 1 | 0 |
| 4635 | 3740 | 1 | 0 |
| -145 | 1164 | 0 | 1 |
| 1626 | 2688 | 0 | 1 |
| 353 | 1168 | 0 | 1 |
| 4903 | 3732 | 1 | 0 |
| 694 | 2402 | 0 | 1 |
| -534 | 730 | 0 | 1 |
| -237 | 1429 | 0 | 1 |
| -233 | 710 | 0 | 1 |
| -145 | 717 | 0 | 1 |
| -36 | 856 | 0 | 1 |
| 32 | 2360 | 0 | 1 |
| 6244 | 3782 | 1 | 0 |
| -419 | 592 | 0 | 1 |
| 447 | 2757 | 0 | 1 |
| 557 | 2099 | 0 | 1 |
| -215 | 1333 | 0 | 1 |
| -281 | 886 | 0 | 1 |
| 5095 | 3758 | 1 | 0 |
| -3 | 1601 | 0 | 1 |

## 6.3 Curved path with no obstacles test data

This is data from the third experiment 4.2.3.

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 7779 | 8685 | 1 | 0 |
| 4225 | 9063 | 1 | 0 |
| 4111 | 7924 | 1 | 0 |
| 6213 | 8890 | 1 | 0 |
| 7886 | 7621 | 1 | 0 |
| 9617 | 8314 | 1 | 0 |
| 5962 | 8965 | 1 | 0 |
| 10427 | 8977 | 1 | 0 |
| 7965 | 8187 | 1 | 0 |
| 6037 | 8232 | 1 | 0 |
| 8691 | 9615 | 1 | 0 |
| 7540 | 7575 | 1 | 0 |
| 6543 | 8845 | 1 | 0 |
| 11438 | 10166 | 1 | 0 |
| 11684 | 8199 | 1 | 0 |
| 6477 | 9431 | 1 | 0 |
| 5368 | 7937 | 1 | 0 |
| 5938 | 8875 | 1 | 0 |
| 4597 | 8926 | 1 | 0 |
| 5462 | 8454 | 1 | 0 |
| 8314 | 8259 | 1 | 0 |
| 5654 | 8197 | 1 | 0 |
| 15561 | 9606 | 1 | 0 |
| 11768 | 10895 | 1 | 0 |
| 7322 | 8520 | 1 | 0 |
| 9236 | 8008 | 1 | 0 |
| 10808 | 8575 | 1 | 0 |
| 6257 | 9686 | 1 | 0 |
| 2326 | 8198 | 1 | 0 |
| 12085 | 9381 | 1 | 0 |
| 8135 | 8119 | 1 | 0 |
| 4353 | 9348 | 1 | 0 |
| 8487 | 8735 | 1 | 0 |
| 9916 | 8543 | 1 | 0 |
| 9729 | 8548 | 1 | 0 |
| 5077 | 8738 | 1 | 0 |
| 7750 | 8996 | 1 | 0 |
| 10367 | 8097 | 1 | 0 |
| 4794 | 8391 | 1 | 0 |
| 5739 | 9080 | 1 | 0 |
| 5321 | 9442 | 1 | 0 |
| 10593 | 8231 | 1 | 0 |
| 3338 | 9303 | 1 | 0 |
| 6964 | 9063 | 1 | 0 |
| 7414 | 7787 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 9926 | 8555 | 1 | 0 |
| 9287 | 10413 | 1 | 0 |
| 11763 | 10163 | 1 | 0 |
| 4500 | 8345 | 1 | 0 |
| 11451 | 8886 | 1 | 0 |
| 4702 | 8015 | 1 | 0 |
| 9360 | 10734 | 1 | 0 |
| 7980 | 13169 | 1 | 0 |
| 4110 | 9821 | 1 | 0 |
| 10741 | 11181 | 1 | 0 |
| 5790 | 10164 | 1 | 0 |
| 7763 | 8615 | 1 | 0 |
| 4816 | 7972 | 1 | 0 |
| 11146 | 8180 | 1 | 0 |
| 5753 | 8694 | 1 | 0 |
| 5679 | 8661 | 1 | 0 |
| 2891 | 8442 | 1 | 0 |
| 8694 | 9288 | 1 | 0 |
| 8330 | 8127 | 1 | 0 |
| 11726 | 7492 | 1 | 0 |
| 8160 | 8343 | 1 | 0 |
| 12125 | 8823 | 1 | 0 |
| 8895 | 9305 | 1 | 0 |
| 4449 | 8073 | 1 | 0 |
| 5529 | 8041 | 1 | 0 |
| 9021 | 8296 | 1 | 0 |
| 6573 | 8583 | 1 | 0 |
| 5469 | 8870 | 1 | 0 |
| 5901 | 7403 | 1 | 0 |
| 9204 | 8334 | 1 | 0 |
| 6289 | 8056 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 4628 | 8407 | 1 | 0 |
| 8499 | 8985 | 1 | 0 |
| 7281 | 9287 | 1 | 0 |
| 6351 | 8430 | 1 | 0 |
| 9773 | 9442 | 1 | 0 |
| 8086 | 8125 | 1 | 0 |
| 6029 | 8152 | 1 | 0 |
| 6742 | 9283 | 1 | 0 |
| 5887 | 8261 | 1 | 0 |
| 7298 | 7817 | 1 | 0 |
| 5344 | 8695 | 1 | 0 |
| 3847 | 8156 | 1 | 0 |
| 12395 | 8947 | 1 | 0 |
| 7953 | 9233 | 1 | 0 |
| 4663 | 8087 | 1 | 0 |
| 12162 | 7709 | 1 | 0 |
| 11367 | 8758 | 1 | 0 |
| 10176 | 9199 | 1 | 0 |
| 6903 | 9391 | 1 | 0 |
| 7461 | 8073 | 1 | 0 |
| 5996 | 8897 | 1 | 0 |
| 14940 | 8660 | 1 | 0 |
| 8536 | 8876 | 1 | 0 |
| 12098 | 8849 | 1 | 0 |
| 6599 | 9057 | 1 | 0 |
| 12503 | 10026 | 1 | 0 |
| 6504 | 8425 | 1 | 0 |
| 11575 | 8317 | 1 | 0 |
| 6158 | 9236 | 1 | 0 |
| 6164 | 8286 | 1 | 0 |
| 7385 | 7986 | 1 | 0 |
| 6373 | 7616 | 1 | 0 |
| 5495 | 8349 | 1 | 0 |
| 12918 | 9144 | 1 | 0 |
| 6228 | 8281 | 1 | 0 |
| 10779 | 9213 | 1 | 0 |
| 7694 | 7857 | 1 | 0 |
| 13641 | 9136 | 1 | 0 |
| 7997 | 8460 | 1 | 0 |
| 3071 | 7804 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 4769 | 9009 | 1 | 0 |
| 6246 | 8487 | 1 | 0 |
| 10955 | 8915 | 1 | 0 |
| 5402 | 9905 | 1 | 0 |
| 9710 | 6653 | 1 | 0 |
| 7062 | 8338 | 1 | 0 |
| 5961 | 9611 | 1 | 0 |
| 12377 | 10122 | 1 | 0 |
| 10945 | 9965 | 1 | 0 |
| 6596 | 8211 | 1 | 0 |
| 7847 | 8106 | 1 | 0 |
| 4775 | 8612 | 1 | 0 |
| 6354 | 10371 | 1 | 0 |
| 7989 | 8705 | 1 | 0 |
| 9903 | 8373 | 1 | 0 |
| 8348 | 10074 | 1 | 0 |
| 6716 | 9640 | 1 | 0 |
| 6779 | 8724 | 1 | 0 |
| 10136 | 9339 | 1 | 0 |
| 7653 | 8330 | 1 | 0 |
| 12526 | 9302 | 1 | 0 |
| 6266 | 7474 | 1 | 0 |
| 12998 | 8740 | 1 | 0 |
| 2289 | 7852 | 1 | 0 |
| 11109 | 8370 | 1 | 0 |
| 13802 | 7809 | 1 | 0 |
| 7159 | 7691 | 1 | 0 |
| 6033 | 8365 | 1 | 0 |
| 10486 | 10186 | 1 | 0 |
| 12279 | 8759 | 1 | 0 |
| 7409 | 8094 | 1 | 0 |
| 4850 | 10035 | 1 | 0 |
| 7361 | 8388 | 1 | 0 |
| 7102 | 9216 | 1 | 0 |
| 5830 | 7995 | 1 | 0 |
| 12293 | 11002 | 1 | 0 |
| 4350 | 8880 | 1 | 0 |
| 5109 | 8960 | 1 | 0 |
| 6920 | 8367 | 1 | 0 |
| 10974 | 7566 | 1 | 0 |
| 2904 | 8032 | 1 | 0 |
| 7507 | 9026 | 1 | 0 |
| 9290 | 10230 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 11898 | 8866 | 1 | 0 |
| 9440 | 8962 | 1 | 0 |
| 4374 | 8343 | 1 | 0 |
| 8574 | 9218 | 1 | 0 |
| 11359 | 9524 | 1 | 0 |
| 5057 | 11264 | 1 | 0 |
| 8911 | 7723 | 1 | 0 |
| 10358 | 8861 | 1 | 0 |
| 5651 | 7238 | 1 | 0 |
| 5886 | 9173 | 1 | 0 |
| 8204 | 9100 | 1 | 0 |
| 4329 | 8375 | 1 | 0 |
| 6922 | 8949 | 1 | 0 |
| 5102 | 7829 | 1 | 0 |
| 4936 | 10058 | 1 | 0 |
| 9801 | 9084 | 1 | 0 |
| 7734 | 8771 | 1 | 0 |
| 2771 | 8367 | 1 | 0 |
| 6398 | 8377 | 1 | 0 |
| 8271 | 8581 | 1 | 0 |
| 8464 | 10140 | 1 | 0 |
| 4747 | 7652 | 1 | 0 |
| 12645 | 9878 | 1 | 0 |
| 5449 | 10331 | 1 | 0 |
| 7296 | 7782 | 1 | 0 |
| 8653 | 9044 | 1 | 0 |
| 3827 | 8412 | 1 | 0 |
| 11954 | 9141 | 1 | 0 |
| 8281 | 8273 | 1 | 0 |
| 8689 | 8679 | 1 | 0 |
| 5602 | 7440 | 1 | 0 |
| 9029 | 9327 | 1 | 0 |
| 4063 | 8573 | 1 | 0 |

## 6.4 Curved path with dynamic and static obstacles test data

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 1546 | 837 | 0 | 1 |
| 992 | 1081 | 0 | 1 |
| 10304 | 4712 | 0 | 1 |
| 8417 | 2958 | 0 | 1 |
| 14525 | 6337 | 1 | 0 |
| 4223 | 2586 | 0 | 1 |
| 3945 | 3493 | 0 | 1 |
| 7513 | 6193 | 1 | 0 |
| 1363 | 1258 | 0 | 1 |
| 7924 | 6576 | 1 | 0 |
| 18594 | 6773 | 1 | 0 |
| 5866 | 2791 | 0 | 1 |
| 5594 | 6384 | 0 | 1 |
| 9604 | 5640 | 1 | 0 |
| 2925 | 2209 | 0 | 1 |
| 4815 | 3919 | 0 | 1 |
| 8071 | 4070 | 0 | 1 |
| 3256 | 1372 | 0 | 1 |
| 19936 | 7048 | 1 | 0 |
| 2139 | 1080 | 0 | 1 |
| 3656 | 2652 | 0 | 1 |
| 6525 | 3664 | 0 | 1 |
| 2419 | 1436 | 0 | 1 |
| 4082 | 3280 | 0 | 1 |
| 3477 | 2462 | 0 | 1 |
| 1577 | 946 | 0 | 1 |
| 8665 | 5821 | 0 | 1 |
| 3178 | 1346 | 0 | 1 |
| 2009 | 2516 | 0 | 1 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 5061 | 2568 | 0 | 1 |
| 2237 | 2368 | 0 | 1 |
| 3122 | 1708 | 0 | 1 |
| 3101 | 1439 | 0 | 1 |
| 3343 | 4170 | 0 | 1 |
| 7768 | 4399 | 0 | 1 |
| 1549 | 1367 | 0 | 1 |
| 2064 | 2280 | 0 | 1 |
| 6892 | 3154 | 0 | 1 |
| 1132 | 1052 | 0 | 1 |
| 6805 | 5962 | 1 | 0 |
| 3799 | 1265 | 0 | 1 |
| 1957 | 2812 | 0 | 1 |
| 3452 | 1149 | 0 | 1 |
| 7069 | 4654 | 0 | 1 |
| 2788 | 1457 | 0 | 1 |
| 2402 | 2199 | 0 | 1 |
| 4239 | 1847 | 0 | 1 |
| 4342 | 3019 | 0 | 1 |
| 1778 | 1232 | 0 | 1 |
| 3111 | 778 | 0 | 1 |
| 2840 | 2882 | 0 | 1 |
| 1724 | 2266 | 0 | 1 |
| 3389 | 709 | 0 | 1 |
| 19497 | 6956 | 1 | 0 |
| 4261 | 3810 | 0 | 1 |
| 3079 | 1983 | 0 | 1 |
| 15645 | 6924 | 0 | 1 |
| 3328 | 1409 | 0 | 1 |
| 4216 | 2022 | 0 | 1 |
| 19616 | 7482 | 1 | 0 |
| 4311 | 1895 | 0 | 1 |
| 15727 | 6430 | 1 | 0 |
| 6761 | 5455 | 1 | 0 |
| -1757 | 146 | 0 | 1 |
| -2003 | 13 | 0 | 1 |
| 13340 | 6503 | 0 | 1 |
| 4175 | 1114 | 0 | 1 |
| 8639 | 5626 | 1 | 0 |
| 5458 | 5795 | 1 | 0 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| 2535 | 1259 | 0 | 1 |
| 1654 | 1532 | 0 | 1 |
| -2000 | 1 | 0 | 1 |
| 10130 | 4040 | 0 | 1 |
| 6615 | 4650 | 0 | 1 |
| 3675 | 3552 | 0 | 1 |
| 9800 | 6589 | 1 | 0 |
| 1560 | 1716 | 0 | 1 |
| 6108 | 2402 | 0 | 1 |
| 11019 | 5371 | 0 | 1 |
| 3436 | 2118 | 0 | 1 |
| 1939 | 1032 | 0 | 1 |
| 2695 | 2382 | 0 | 1 |
| 1589 | 858 | 0 | 1 |
| 10678 | 6698 | 1 | 0 |
| 2261 | 2048 | 0 | 1 |
| 1200 | 782 | 0 | 1 |
| 5174 | 3730 | 0 | 1 |
| 2803 | 1402 | 0 | 1 |
| 6095 | 4553 | 0 | 1 |
| 7419 | 5128 | 0 | 1 |
| 2224 | 1868 | 0 | 1 |
| 6298 | 2615 | 0 | 1 |
| 3325 | 1243 | 0 | 1 |
| 14960 | 6762 | 1 | 0 |
| 11593 | 5578 | 0 | 1 |
| 8010 | 3174 | 0 | 1 |
| 27852 | 6805 | 1 | 0 |
| 3529 | 1558 | 0 | 1 |
| 1833 | 1805 | 0 | 1 |
| 2313 | 2409 | 0 | 1 |
| 5810 | 5129 | 1 | 0 |
| 1407 | 983 | 0 | 1 |
| 3215 | 1546 | 0 | 1 |
| 7698 | 3635 | 0 | 1 |

| Score | Time steps spent | Did it complete? | Did it collide? |
|---|---|---|---|
| -311 | 385 | 0 | 1 |
| -404 | 346 | 0 | 1 |
| -449 | 346 | 0 | 1 |
| 8115 | 6555 | 1 | 0 |
| 10494 | 5027 | 0 | 1 |
| 739 | 1112 | 0 | 1 |
| 5940 | 6147 | 1 | 0 |
| 3995 | 3858 | 0 | 1 |
| 2684 | 1263 | 0 | 1 |
| 5306 | 3712 | 0 | 1 |
| 3450 | 2777 | 0 | 1 |
| 4000 | 3254 | 0 | 1 |
| 4935 | 4003 | 0 | 1 |
| 6460 | 4767 | 0 | 1 |
| 2038 | 1313 | 0 | 1 |
| 1259 | 1341 | 0 | 1 |
| 3142 | 4458 | 0 | 1 |
| 3807 | 2178 | 0 | 1 |
| 3498 | 1891 | 0 | 1 |
| 2710 | 1786 | 0 | 1 |
| 2711 | 3397 | 0 | 1 |
| 3742 | 4625 | 0 | 1 |
| 10302 | 6378 | 1 | 0 |
| 900 | 1000 | 0 | 1 |
| 4282 | 1372 | 0 | 1 |
| 316 | 896 | 0 | 1 |
| 16107 | 6864 | 1 | 0 |
| 5946 | 1920 | 0 | 1 |
| 3723 | 2713 | 0 | 1 |
| 5470 | 3808 | 0 | 1 |
| 6252 | 2961 | 0 | 1 |
| 3343 | 1394 | 0 | 1 |
| 3286 | 3309 | 0 | 1 |
| 2022 | 1802 | 0 | 1 |
| 9935 | 6533 | 1 | 0 |
| 3350 | 1593 | 0 | 1 |
| 8683 | 6191 | 1 | 0 |
| 2094 | 788 | 0 | 1 |
| 1841 | 2324 | 0 | 1 |
| 16293 | 6738 | 1 | 0 |
| 1741 | 2078 | 0 | 1 |
| 2886 | 4568 | 0 | 1 |
| 3019 | 1553 | 0 | 1 |
| 11609 | 6638 | 1 | 0 |
| 3415 | 2141 | 0 | 1 |
| 4180 | 3808 | 0 | 1 |
| 2762 | 2438 | 0 | 1 |
| -1949 | 73 | 0 | 1 |
| -2009 | 167 | 0 | 1 |

NTNU

Norwegian University of
Science and Technology