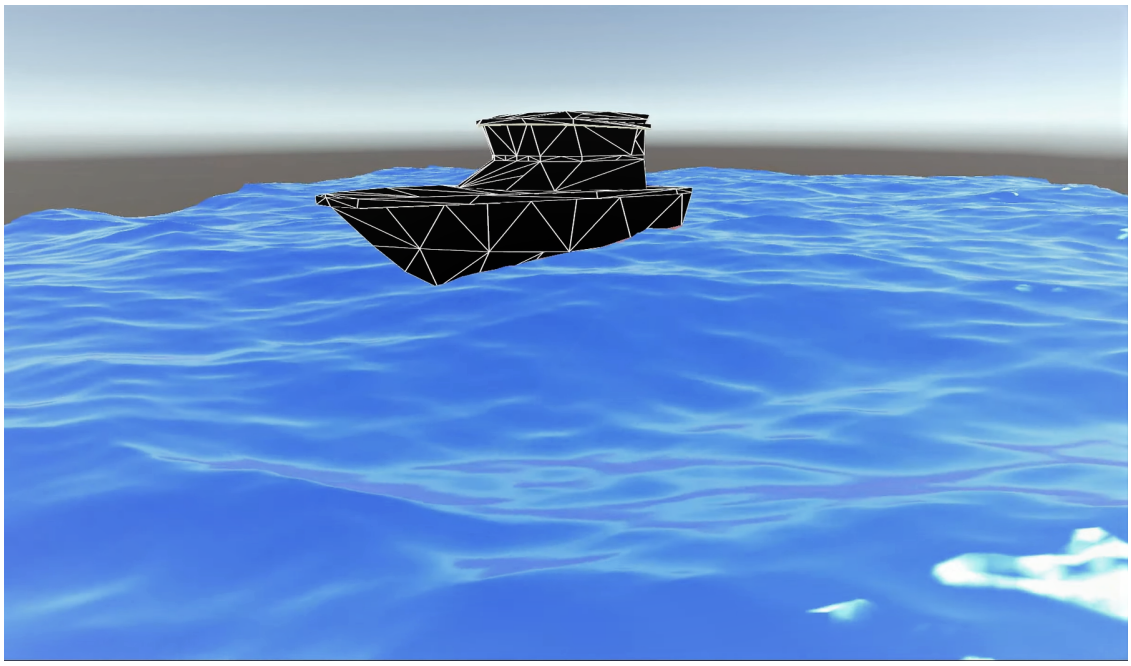


Artificial intelligence based marine autopilot

Trained using reinforcement learning in the Unity simulation environment

Master's thesis in Complex Adaptive Systems

Martin Asplund
David Näslund

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

Chalmers University of Technology
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022:37

Artificial intelligence based marine autopilot

Trained using reinforcement learning in the
Unity simulation environment

Martin Asplund
David Näslund

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Applied Artificial Intelligence Group
Chalmers University of Technology
Gothenburg, Sweden 2022

Artificial intelligence based marine autopilot
Trained using reinforcement learning in the Unity simulation environment
Martin Asplund and David Näslund

Supervisor: Mikael Ahlstedt, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2022:37
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Applied Artificial Intelligence Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simulation environment constructed in Unity showcasing a 3D boat model idling in the waves.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Artificial intelligence based marine autopilot
Trained using reinforcement learning in the Unity simulation environment
MARTIN ASPLUND and DAVID NÄSLUND
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

When driving a ship, especially long distances, an autopilot is usually deployed. Since the ship is subjected to disturbances from currents and wind alike, it is not easy to keep a steady course. Furthermore, the dynamic behaviour of waves striking the vessel, known as swaying, makes the task of keeping a straight line through the sea daunting. To assist, there exist commercial autopilots. However, most of these are subject to less than simple calibration, which also is hard to keep accurate throughout the lifespan of the boat due to wear, different load conditions, and other similar things. Also, there is generally no adaptability related to the autopilot, meaning a sudden change in engine performance will stop the autopilot from functioning. Further, the majority of today's commercial autopilots are designed to follow a course or a heading, known as *course-hold* and *heading-hold* autopilots. Hence, there exists a desire to develop a more adaptable *path-following* autopilot.

One way of solving the adaptability issue is to borrow the solution from the aircraft industry and use a control allocator. Given a set of global forces (usually F_x, F_y, and moment M_z) the control allocator tries to distribute these between a given set of actuators. Since the number of control signals usually is far less than the number of actuators, these systems are said to be over-actuated, and no unique solution exists.

This work aims at exploring a new way of constructing computationally efficient regulation and control allocation for vessels, in the form of a path-following autopilot. The hypothesis is that, by using a Neural Network as a control allocator, better performance and adaptability than offered by present solutions can be achieved.

Keywords: Unity, Reinforcement learning, Simulation, Autopilot, Marine, Artificial Intelligence, Neural Network, Control Allocation.

Artificial intelligence based marine autopilot
Trained using reinforcement learning in the Unity simulation environment
MARTIN ASPLUND and DAVID NÄSLUND
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Sammanfattning

När ett fartyg körs, särskilt under långa sträckor, används vanligtvis en autopilot. Eftersom fartyget utsätts för störningar från både strömmar och vind är det inte lätt att hålla en jämn kurs. Dessutom gör det dynamiska beteendet, från vågor som träffar fartyget, det svårt att hålla en rak kurs genom havet. Till hjälp finns det kommersiella autopiloter. De flesta av dessa kräver dock svåra och tidskrävande kalibreringar, vilka är svårt att hålla exakta genom båtens livslängd på grund av slitage, olika belastningsförhållanden och liknande. Dessutom finns det i allmänhet ingen anpassningsbarhet relaterad till autopiloten, vilket innebär att en plötsliga förändring i motorns prestanda kommer att få autopiloten att sluta fungera. Vidare, är majoriteten av dagens kommersiella autopiloter utformade för att följa en riktning eller en kurs, "*heading-hold*" och "*course-hold*". Därför finns det en marknad och efterfrågan för att utveckla en mer adapterbar "*path-following*" autopilot.

Ett sätt att lösa anpassningsbarheten är att låna lösningen från flygindustrin och använda en kontrollfördelare. Givet en uppsättning globala krafter (vanligtvis F_x, F_y och momentet M_z) försöker kontrollfördelaren fördela dessa mellan en given uppsättning ställdon. Eftersom antalet styrsignaler vanligtvis är mycket färre än antalet ställdon, sägs dessa system vara överaktiverade, och det finns därmed ingen unik lösning.

Detta arbete syftar på att utforska ett nytt sätt att konstruera beräkningseffektiv reglering och kontrollfördelning för fartyg, i form av en "*path-following*" autopilot. Hypotesen är att, genom att använda ett neuralt nätverk som kontrollfördelning kan bättre prestanda och anpassningsförmåga, än vad som erbjuds av nuvarande lösningar, uppnås.

Nyckelord: Unity, Reinforcement learning, Simulering, Autopilot, Marint, Artificiell Intelligens, Nueralt Nätverk, Kontrollfördelare.

Acknowledgements

We would like to show our deepest gratitude to everyone who has played a part in this project. If not for their enthusiasm and active support this project would not be possible. A big thanks goes to **Peter Forsberg**, our examiner at Chalmers, and **Mikael Ahlstedt** at CPAC for guiding and consulting us throughout the entirety of the project.

Furthermore, we would like to thank everyone else who has contributed with their time and knowledge. These people are:

Peter Forsberg – *Department of Mechanics and Maritime Sciences at Chalmers*
Mikael Ahlstedt – *R&D Manager at CPAC Systems AB*
Jon Wingren – *Naval architect at Volvo Penta*
David Wall – *Software developer at CPAC Systems AB*
Axel Måneskiöld – *Master Thesis Student at CPAC Systems AB*
Lukas Ljungquist – *Master Thesis Student at CPAC Systems AB*
Jonatan Nordh – *Perception Engineer at CPAC Systems AB*

Martin Asplund and David Näslund, Gothenburg, June 2022

Thesis advisor: Mikael Ahlstedt, CPAC Systems AB
Thesis examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

List of Acronyms

Here are every acronym that has been used throughout the report listed in alphabetical order:

AI	Artificial Intelligence
AP	Autopilot
API	Application Program Interface
BC	Behaviour Cloning
CAN	Controller Area Network
CTE	Cross Track Error
DOF	Degrees Of Freedom
FFT	Fast Fourier Transform
GAIL	Generative Adversarial Imitation Learning
MA-POCA	Multi-Agent POsthumous Credit Assignment
MAL	Multi-Agent Learning
ML	Machine Learning
NN	Neural Network
PID	Proportional-Integral-Derivatice controller
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAC	Soft Actor-Critic

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the topic of this thesis is laid out together with the appropriate background of autopilots, complete with what is available and missing on the market today. The main problem is explained followed by the purpose and the limitation of the project. Lastly, the general layout of the thesis is touched upon briefly.

Autopilots have been around for a long time [1]. They assist drivers of many different types of vehicles, ranging from cars and buses, to planes, trains, and boats. With the surge of computing power and the rise of artificial intelligence during the last years, the true potential of these autonomous systems are starting to take shape. It is therefore more relevant than ever to perform research in these areas and to further the understanding and capabilities of self-driving systems. The newfound computing power also helps unlocking faster and more accurate simulation environments and computer models. This is crucial since it enables engineers to artificially pre-train, test, and verify autonomous systems virtually before applying anything in the real world.

This thesis is therefore aimed at engineers, enthusiasts, and academics who wants to further their understanding of how to set up and train deep neural networks; with the goal of operating boats in physically accurate ocean environments. The networks are trained using reinforcement learning, and the simulation environments is built in Unity.

1.1 Background

An *Autopilot* (AP) is a system or module that is used to control parameters such as position, heading, or velocity for a marine vessel or other vehicle; without the need for human interaction. For example, cruise control is a simple kind of AP that regulates the power output of the engine to maintain a vehicle certain speed.

APs were first created in the early 20th century with the intention to help aircraft pilots. The primitive AP allowed the aircraft to fly straight at a fixed altitude on a compass course, allowing the pilot to keep track on other important readings while flying. APs were later introduced for marine vessels, namely oil tankers and cargo ships, and eventually even to land based vehicles, such as cars and tractors [1][2][3][4].

In Sweden, there has not been any attempts to create a commercial AP using *Reinforcement Learning* (RL) at the time of writing, to the best of the project members knowledge. Internationally there have been limited attempts, but they have been using very simplistic simulation environments [5]. For example, [6] shows the use of a simple grid-like model with wind and wave disturbances being uniformly distributed. This is usually not accurate enough to provide useful insight for potential real-world scenarios. Furthermore, a fair amount of academic research has been done around trajectory keeping using *Proportional-Integral-Derivatice controller* (PID) regulators and/or the 1^{st} and 2^{nd} order *Nomoto models* [7][8][9].

There has been a decent amount of academic research about marine APs based on RL and NN [10]. Although, the focus has been on deterministic station keeping [11][12] and slow movements for large marine vessels [13], such as oil tankers and cargo ships [14]. More often than not, these rely on commercial simulation programs with less capabilities to change things. These could include, parameters related to wind, waves, or forces. Furthermore, the simulation could be limited to 2D, thereby lacking the vital realistic 3D space [15]. Creating an environment in-house would therefor provide much greater freedom, control, accuracy, and could provide better performance.

To create a *virtual environment*, certain tools are needed. *Unity* is a game developing platform provided by the company *Unity Technologies*, that launched in 2005. According to their website [16], the company claims that it is available in over 190 countries, and that more than 50% of all games on mobile, PC, and console were made using their platform. It gives a good foundation to build simulated environments, is well documented, and has an active community with plenty of forums for additional aid[17].

1.1.1 CPAC

This thesis is carried out at the company *CPAC* during the spring of 2022, and is heavily influenced by one of their earlier projects. CPAC is a medium sized technical innovation-company with over 200 employees. The company is owned by Volvo Penta, which is included in the AB Volvo group. The company mainly works within the marine and construction fields and has its origin in Mölndal, Gothenburg.

The earlier project at CPAC was an in-house effort to create a *Heading-Hold Autopilot* using RL in Unity. The trained *Neural Network* (NN) was later used to control a boat in regards to rudder steering. The project utilized several python packages, most notably python-can for the communication to the onboard computers and ML-Agents to train the NN.

1.1.2 Autopilots on the market

Available brands include Garmin, Simrad, Kongsberg, Comnav and Furuno. One specially interesting aspect is that according to Comnav's website [18], they are using "Intelligent Steering Technology". They also state that using their "cutting edge technology", opens the possibility for new navigation solutions using *Artificiell Intelligens* (AI). With further investigation into the matter, i.e. contacting Comnav for a comment on their "cutting edge technology" it became apparent that their AP consist of a PID, and no *Machine Learning* (ML) at the moment of writing.

1.2 Problem Statement

The first problem with the APs available today, is that most of them are only focused on *Heading-Hold*, or *Course-Hold* applications. Heading-Hold will only try to "lock" the current compass-heading, for example, only keeping the heading north, no matter how much it veers off to the sides. Course-Hold, tries to make the boat follow a predetermined path, but has no active speed regulation. This means that if the speed is too high, it might fail, and would require manual speed-management.

The second problem is that these APs only use regulators to control the vessel, in forms of PID regulators or similar. Meaning that much time and effort is needed by a technician to fine-tune all the different parameters for each specific boat. This is a cost- and time-intensive process that needs to be performed at regular intervals, as wear and tear changes the properties of the boat over time. Another problem with using only regulators, is that, if independent control is in demand, there would have to be one regulator per driveline; making the entire process far more complex. That is why, often, the freedom of having independent-steering drivelines is overlooked.

The third problem is the lack of adaptability. If the capabilities of the actuators are changed, for example if a rudder gets stuck or drives with reduced speed, current APs have no way of compensating for these things. They are also not able to adapt to different boats with different physical properties.

1.2.1 Purpose and Research Questions

The purpose of this thesis is to investigate if these problems can be solved by adding NNs and ML to achieve greater independent control, and to provide adaptability. A possible solution is therefor to construct a two-part system, where, in addition to the commonly used regulator, a control allocator consisting of a NN is trained and implemented. To give the network the best chance of success, the training is taking place in realistic simulation conditions. The training environment is therefor constructed to replicate a real-world open ocean as closely as possible, while still maintaining manageable run times. Lastly, it is also investigated if this setup, with a smart system capable of controlling actuators individually, can prove beneficial when driving the vessel manually.

All of these points are summarized into one main research question and three sub-questions, which this thesis intends to answer.

- *Is it possible to create an adaptable and general two-part autopilot, consisting of a regulator and a neural network based control allocator, capable of following a path despite environmental disturbances?*
  - *Will the autopilot be able to continue following the path despite loss of capabilities of the actuators?*
  - *How well will the autopilot handle disturbances from the environment, such as wind and waves?*
  - *Will the control allocator prove beneficial for heuristic applications, where the boat is manually driven by a person?*

Even though the goal is to achieve a fully functional autopilot capable of following paths even under difficult environmental conditions, this thesis should only be seen as a proof of concept. A stepping stone to provide useful insight into how this goal might be accomplished in future works, research, and commercial products.

1.3 Delimitations

The project focuses exclusively on using RL to train a marine autopilot for personal use on privately owned boats. This is done in Unity utilizing various Python and C#-packages, where the creation of the simulation environment is also covered. In regards to implementing the AP, the work is limited to Volvo Penta boats and their already-in-place *Controller Area Network* (CAN) system. The trained AP is limited to 3 *Degrees Of Freedom* (DOF), namely *x*, *y* & *yaw*. Therefore, the AP only controls the position and heading of the boat, and does not regulate *roll*, *pitch* nor *heave*. The AP is trained on a *Test* boat, provided by Volvo Penta, that resembles an average boat on today's market. The various parameters related to the boat are located in Table 1.1 and are provided in part with Volvo Penta. A rendered version of the vessel is seen in Figure 1.1.

Table 1.1: Test boat data.

Length Overall (LOA)	15	[m]
Beam Overall (Boa)	5.173	[m]
Depth Overall (D)	2.658	[m]
Weight (m)	15000	[kgf]

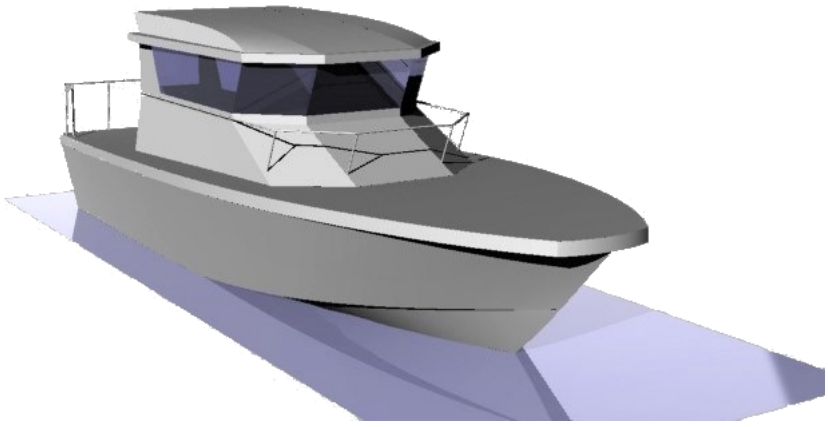


Figure 1.1: Rendered image of Test boat.

The main thrusters used are Azimuth thrusters from Volvo Penta, specifically the D6-IPS thrusters. No true propeller dynamics is implemented in the simulation model, the propellers are instead treated as small jet engines. Another limitation is that effects from propellers and boats on the surrounding water are not taken into consideration. Examples of these includes wakes, extra flow from propellers, and obstructed flow by the boat.

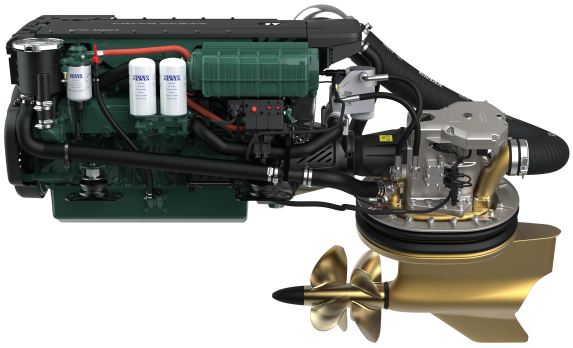


Figure 1.2: Volvo Penta Azimuth thruster D6-IPS.

1.4 Disposition

This thesis follows the standard disposition with chapters in the order of Introduction, Theory, Method, Results, Analysis, Discussion, and Conclusion. The Introduction provides a bit of background and defines the main problems, purpose, and delimitations of the project. Next, the Theory continues with outlining relevant theories and existing documentation needed to understand the topics of this thesis.

In Method, the methodology used to create and implement both the autopilot and the simulation environment is explained in detail. In the following chapter, Results, the performance of the system is tested and the outcome of how well it performs under different circumstances is showcased. The chapter in which the analysis takes place is called Evaluation, since the entire chapter is dedicated to evaluating the results from the autopilot. Finally, the last two chapters provides the Discussion and Conclusion to the entire thesis, explaining what could have been done differently, what should be done in the future, and if this is a viable solution to the outlined problems.

Chapter 2

Theory

In this chapter, relevant theory, previous research, and documentation is presented about the topics of this thesis. The main parts being, theory about reinforcement learning, documentation of ML-Agents, relevant equation of motion for marine vessels, and the theoretical modeling of ocean waves using the fast Fourier transformation.

2.1 Reinforcement Learning

RL is a type of machine learning where the goal is to train an *agent* to attain desired behaviors in an environment by maximizing a *reward*. A common structure of RL is depicted in Figure 2.1. The schematic consist of an agent and an environment that affect each other through *actions*, *states*, and the previously mentioned rewards (A_t, S_t, R_t). Subscript t indicates the current time step.

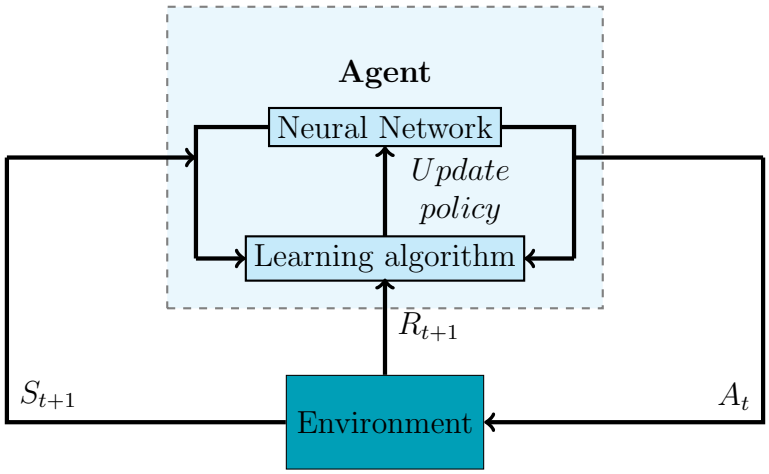


Figure 2.1: Training schematic of Reinforcement Learning with an agent and an environment.

First, the agent takes an action A_t, this could for example be *turn right* or *accelerate*. Next, the environment moves the agent accordingly, which then ends up in the new state S_{t+1}. A state is defined as a unique combinations of sensor observations, meaning that a single state can be seen multiple times throughout the training. The reward R_{t+1} is generated by the previous state and action, and defines how "good" the action A_t was in state S_t. Rewards are calculated in each time step based on a Reward Function, which governs the outcome of the training and eventually the behavior of the agent. The learning algorithm stores every observation, action and reward in order to update the agent based on the reward function.

When dealing with RL, the *policy*, usually in the form of a NN, is what determines the action given a certain state. An Artificial Neural Network is inspired by principles of the human brain, with many nodes connected through variables weights and biases. Figure 2.2 is visualizing the forward propagation of a NN.

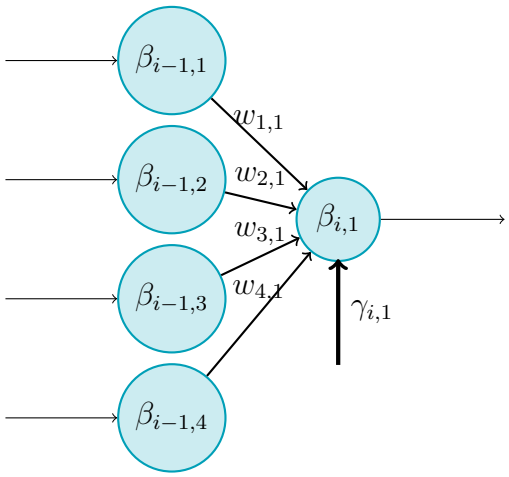


Figure 2.2: Section of a NN illustrating forward propagation.

The mathematical expression for the forward propagation in a general case can be defined as:

$$\beta_{i,j} = \gamma_{i,j} + \sum_{k=1}^{N} \beta_{i-1,k} \cdot w_{k,j} \tag{2.1}$$

In equation 2.1, given Figure 2.2, N is the number of neurons in the previous layer and would be equal to 4. Furthermore, i is the i-th layer and j is the j-th node in layer i, in this case j would be 1. Similarly, $\gamma_{i,j}$ is the bias at node j in layer i. Finally, $\beta_{i,j}$ is the sum of the $\{i, j\}$ node, and $w_{k,j}$ is the weigh connecting two nodes.

2.2 ML-Agents

ML-Agents is an open-source toolkit that enables Unity simulations and games to serve as environments for training intelligent agents. It also provides a low level *Application Program Interface* (API) for Python developers that wants to directly interact with the Unity environment. In this section, the various features ML-Agent provide are explained, along with the possible benefits of implementing them.

2.2.1 Reward signals

There are different ways of rewarding the agent during training, these are: *Extrinsic*, *Curiosity*, and *Generative Adversarial Imitation Learning* (GAIL); and are referred to as reward signals. The extrinsic reward signal is the reward provided by the reward function, based on the simulation environment. Curiosity is a signal that rewards the agent for trying new actions, as described in [19]. Moreover, GAIL awards the agent for proving similar behavior to that of a provided *demonstration file*. This is explained more in-depth in section 2.2.5.

The reward does not need to be given to the agent at each time step t. The reward could be given at larger time steps or in a nonlinear manner. This is known as a *sparse reward function*, that only rewards the agent in specific states, whether they are based on time or position. For example, once the agent has achieved 100 steps, driven 50 meters, or reached a certain position.

2.2.2 Environmental parameters

Environmental parameters are variables that can scale or change value as the training progresses. In the ML-Agent documentation, this is referred to as *Curriculum learning*. This is often utilized to make the training environment progressively more difficult by changing training parameters over time. Whenever these changes occurs, the training is said to progress into a new *lesson*. According to the documentation, it is possible to advance to the next lesson by either reaching a certain training time or cumulative reward.

2.2.3 Multiple instances

To collect observations at a higher rate, and eventually train an agent faster, there can be multiple instances of the simulation environment running simultaneously. This is known as *Multi-Agent-Learning* (MAL), which is also used to explore cooperative behaviour in agents. For example, making several agents push an object into a goal or play a board game together. In the ML-Agents documentation, this is known as *MultiAgent POsthumous Credit Assignment* (MA-POCA).

2.2.4 Action masking

In the documentation for ML-Agents it is mentioned that *Action masking* is possible when using discrete actions. When discrete actions are used, the action space is usually small and user-defined. A discrete action is an integer value between 0 and the defined limit by the user. Contrary, when continuous actions are used, the action space is more or less infinite, since the action space is pre-defined to a floating point number.

The idea behind action masking, is to remove the possibility to perform actions which are infeasible in the current state. For example, trying to drive faster than top speed, or turning more than the maximum turning angle. By disabling invalid actions, the possible action space shrinks, which could help the agent to make better decisions.

2.2.5 GAIL

GAIL is used to potentially improve learning times by providing a rough demonstration of how the agent is expected to work. Demonstration files are obtained via the recorder component in the *ML-Agent Unity* package. As described in Jonathan Ho [20], GAIL is a framework to directly extract a policy, i.e. a neural network, from recorded data using inverse RL. According to the report, GAIL is based on imitation learning and Generative Adversarial Networks (GANs).

In the framework, the training now utilizes two neural networks, one named the discriminator and the regular policy. The discriminator learns to differentiate between actions/observations from the demonstration file and the ones from the agent itself. The reward is then based on how similar the new action/observation is to the one in the demonstration file. The agent always tries to maximize the reward given, and respectively the discriminator learns to better distinguish between the actions/observations from the agent and the demonstration. To summarize, as the agent learns to mimic the demonstrations, the discriminator increasingly is getting stricter.

According to the documentation for ML-Agents, this together with *Behaviour Cloning* (BC) can drastically improve the learning time of the agent. The documentation also recommends the demonstrations to include a variety of episode lengths for stability purposes.

BC is similar to GAIL in that it also requires at least one demonstration file. The main difference is that BC trains the policy of the agent to exactly mimic the one in the demonstration file. According to "Lecture 10: Imitation Learning" [21], the training will often perform poorly when the demonstration does not cover the entire state space. Unfortunately, this is the case for many complex simulations. The ML-Agent documentation states that BC is likely to work better in conjunction with GAIL or extrinsic reward. Further, the documentation claims that both BC and GAIL have support for multiple demonstration files during training. This could be beneficial to cover the whole state space by running multiple smaller recordings.

2.2.6 Self-play

Self-play is a method which may improve the stability of the training. It enables agents to play against each other and compete for the best policy. Self-play does not require the simulation to run MAL, a single agent can instead utilize self-play by competing against an older version of itself. However, it requires a specific extrinsic reward function that specifies which agent is currently winning and who is losing.

2.3 Marine vehicle equation of motion

Equations of motion are a set of equations that describe the motion of a physical system as mathematical functions of time [22]. According to Fossen [23], for a 6 DOF marine vessel in a body-fixed system, the equations that govern its motions can be written as:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau} \tag{2.2}$$

In equation (2.2), $\mathbf{M}$ and $\mathbf{C}(\boldsymbol{v})$ are the *Inertia matrix* and the *Coriolos and Centripetal matrix* of the ship, where added mass is taken into account [24]. Further, $\mathbf{D}(\boldsymbol{v})$ is the *Hydrodynamic dampening matrix* and $\mathbf{g}(\boldsymbol{\eta})$ is the vector containing the *Restoring forces*, that is due to weight and buoyancy. The vector $\boldsymbol{v}$ contains the 6 DOF, namely, $\{u, v, w, p, q, r\}^T$. These are depicted in Figure 2.3, and are the linear/angular -velocities. Moreover, $\boldsymbol{\tau}$ represents the resulting forces and moments acting on the ship, see equation (2.3).

$$\boldsymbol{\tau} = \tau_p + \tau_d + \tau_h + \tau_s \tag{2.3}$$

The different components of τ are: **p**ropulsion, **d**isturbance, **h**ydrodynamic & **s**urface. Further, the matrices $\mathbf{M}$ and $\mathbf{C}(\boldsymbol{v})$ are defined as:

$$\mathbf{M} \triangleq M_{RB} + M_A \tag{2.4}$$

$$\mathbf{C}(\boldsymbol{v}) \triangleq C_{RB}(\boldsymbol{v}) + C_A(\boldsymbol{v}) \tag{2.5}$$

Where the subscript "RB" stands for rigid body and "A" for added mass. The dampening matrix $\mathbf{D}(\boldsymbol{v})$, is constructed of multiple dampening components. In equation (2.6) , S is dampening due to skin effect, and W is for wave drift damping. Similarly, M is for damping due to vortex shedding and P for potential damping.

$$\mathbf{D}(\boldsymbol{v}) \triangleq D_P(\boldsymbol{v}) + D_S(\boldsymbol{v}) + D_W(\boldsymbol{v}) + D_M(\boldsymbol{v}) \tag{2.6}$$

When dealing with dynamic positioning of a marine vessel, it is common to simplify equation (2.2) to a system of 3 DOF, since many of these matrices are hard to define in 6 DOF [25][26]. To simplify the model, the global and local coordinate systems are defined as depicted in Figure 2.3. These are defined as $\vec{\boldsymbol{X}} = \{x, y, z\}^T$ for the global and $\vec{\boldsymbol{x}} = \{x_0, y_0, z_0\}^T$ for the local respectively. As the velocity of the boat is of great interest this is defined in the local scope as $\dot{\vec{\boldsymbol{x}}} = \{u, v, w\}^T$. Usually, the velocities in the local scope are named: *surge*, *sway* and *heave*. The angular velocities are also of great importance, especially r; or *yaw*, which is the rotation of the boat around its local y-axis. This is defined as $\vec{\boldsymbol{\phi}} = \{p, q, r\}^T$, these are referred to as; *roll*, *pitch* and as previously mentioned *yaw*. All the quantities are defined using the SNAME [27] notation. Transforming the system to 3 DOF, the dynamics associated with heave, roll and pitch are neglected ($p = q = w = 0$), this is especially true for large oil tankers or transport ships (large surface ships). The vector $\boldsymbol{v}$, then becomes $\{u, v, r\}^T$. Furthermore, this assumption also implies that the restoring forces, $\mathbf{g}(\boldsymbol{\eta})$, can be neglected in a 3 DOF system [23][28].

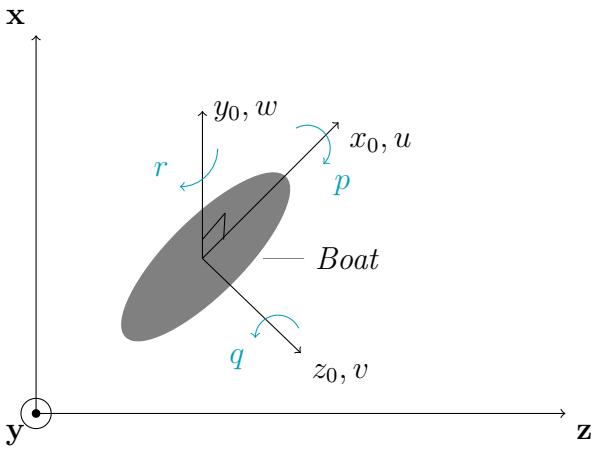


Figure 2.3: Definition of global $\{x, y, z\}$ and local $\{x_0, y_0, z_0\}$ coordinate system along with the linear $\{u, v, w\}$ and angular velocities $\{p, q, r\}$.

To further simplify the model, assume homogeneously distributed mass and xy-symmetry. Moreover, the coordinate system is placed such that its origin is located on the centerline of the ship, meaning $z_g = 0$. Here, z_g is the distance from the origin (centerline) to the center of gravity along the local z-axis. This is true for every ship that has Port-starboard symmetry; hence, the inertia components $I_{xz} = I_{yz} = 0$. These assumptions will give the simplified rigidbody matrices M_{RB} and C_{RB}:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & I_y \end{bmatrix}, \qquad C_{RB}(\boldsymbol{v}) = \begin{bmatrix} 0 & 0 & -m(x_g r + v) \\ 0 & 0 & mu \\ m(x_g r + v) & -mu & 0 \end{bmatrix} \tag{2.7}$$

This in turn, simplifies the inertia added mass matrix along with the added mass Coriolis and Centripetal matrix:

$$M_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Z_{\dot{v}} & -Z_{\dot{r}} \\ 0 & -N_{\dot{v}} & -N_{\dot{r}} \end{bmatrix}, \qquad C_A(\boldsymbol{v}) = \begin{bmatrix} 0 & 0 & Z_{\dot{v}} v + Z_{\dot{r}} r \\ 0 & 0 & -X_{\dot{u}} u \\ -Z_{\dot{v}} v - Z_{\dot{r}} r & X_{\dot{u}} u & 0 \end{bmatrix} \tag{2.8}$$

Moreover, if linear dampening can be assumed, the dampening matrix $\mathbf{D}(\boldsymbol{v})$ is transformed accordingly:

$$\mathbf{D} = \begin{bmatrix} -X_u & 0 & 0 \\ 0 & -Z_v & -Z_r \\ 0 & -N_v & -N_r \end{bmatrix} \tag{2.9}$$

By applying all of these assumptions and simplifications, equation (2.2) is transformed into a 3 DOF system that is suited for dynamic positioning [23]:

$$\mathbf{M}\dot{\boldsymbol{v}} + \mathbf{C}(\boldsymbol{v})\boldsymbol{v} + \mathbf{D}\boldsymbol{v} = \boldsymbol{\tau} \tag{2.10}$$

During station keeping, all the components of $\boldsymbol{v}$ are small, hence further simplifications can be made. Namely, the quadratic velocity terms in $\mathbf{C}(\boldsymbol{v})\boldsymbol{v}$ will be very small, hence $\mathbf{C}(\boldsymbol{v})\boldsymbol{v} \approx 0$ [29].

$$\mathbf{M}\dot{\boldsymbol{v}} + \mathbf{D}\boldsymbol{v} = \boldsymbol{\tau} \tag{2.11}$$

For the 3 DOF system, $\boldsymbol{\tau} = \{F_x, F_y, M_z\}^T$.

2.4 Mathematical ocean wave model

According to Fournier and Reeves [30] oceanic waves can be approximated using sums of *Gerstner waves*, where water particles are moving along circular or elliptical stationary orbits. These can be expressed as *Fourier series*, therefor it is possible to use *Fast Fourier Transformation* (FFT) algorithms to get an accurate and computationally efficient simulation environment.

2.4.1 The Gerstner wave

In the case of circular orbits, Gerstner waves can be described by *Eulers formula*, $\mathrm{e}^{i\phi} = \cos(\phi) + i\,\sin(\phi)$. When adding amplitude and angular velocity, A and ω, the expression (2.12) determines the motion of one water particle subjected to a single Gerstner wave. The imaginary part of h gives the height, and the real part gives the horizontal movement. The circles in figure 2.4 show how the orbits work.

$$h(t) = A\,\mathrm{e}^{i(\phi+\omega\,t)} \tag{2.12}$$

When considering a row of N water particles, their phases must be adjusted based on their positions for the wave to move along the surface. Otherwise, the entire plane would move up and down in unison. Let $0 \leq i_x < N$ be the index for each particle, such that their positions along the x-direction are described by (2.13). Here, L is the length of the surface. Also let $\Delta\phi = \frac{2\pi}{N}$ be the difference in phase between them, giving equation (2.14).

$$x = \frac{L}{N}\,i_x \tag{2.13}$$

$$h(i_x, t) = A\,\mathrm{e}^{i(\Delta\phi\,i_x+\omega\,t)} \quad , \text{ with } \quad \Delta\phi = \frac{2\pi}{N} \tag{2.14}$$

Since *Eulers unity* states that $\mathrm{e}^{i\,2\pi\,n} = 1$, for any integer n, equation (2.14) will take the same value for both $i_x = 0$ and $i_x = n\,N$. Rows can therefor be tiled together to simulate an infinitely long wave. The resulting wave can be seen in red and blue in figure 2.4.

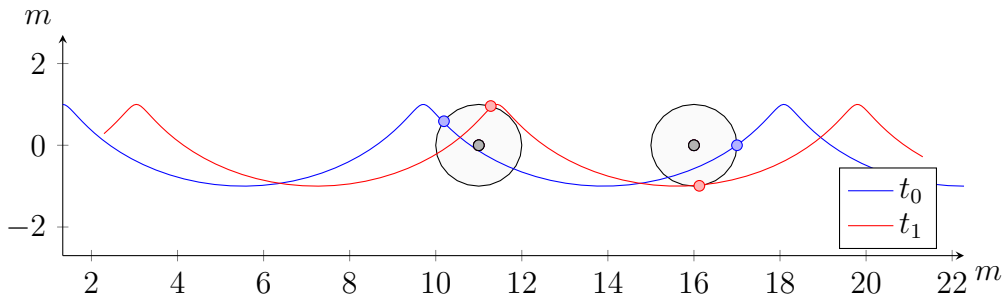


Figure 2.4: A Gerstner wave is shown at two different time steps. It starts at blue at time t_0, and then moves to red at time t_1. The two circles show two of the geostationary orbits that makes up the wave. Their centers must be located at the same height and their orbits must have the same amplitudes for them to create a single continuous wave.

2.4.2 Adding multiple waves

A Gerstner wave travelling along the surface only have one specific direction, amplitude, and angular velocity. The direction is determined by its *wave numbers*, k_x and k_z, which scales how much the wave moves in the x- and z-direction. These form a *wave vector*, $\mathbf{k}$, with magnitude $k = \sqrt{k_x^2 + k_z^2}$. The angular velocity, $\omega(k)$, depends on this magnitude, meaning that higher wave numbers gives faster angular velocities. For now, only the x-direction is considered, but the same principles can be applied in the z-direction as well.

Tessendorf *et al.* [31] describes the magnitude of the wave vector as being related to the length of the water plane, L, according to $k_x \propto \frac{2\pi}{L}$. The authors therefor suggest that, for K number of waves with indices $0 \leq j_x < K$, the different wave numbers can be expressed as (2.15).

$$k_x = \frac{2\pi}{L} j_x \tag{2.15}$$

Many different waves need to be added together to simulate ocean movements. They all need to be distributed according to equation (2.14), but with account to their angular velocities and wave numbers. Equation (2.15) says that for each increase of j_x, there will be an additional period (2π) over the length of L. The phase difference from (2.14), however, only gives one period over L, since it takes N particles to get the length L. This means $\Delta\phi$ needs to be multiplied by j_x to add additional periods for different wave numbers. The sum of the waves gives the motion of each particle, see equation (2.15).

$$h(i_x, t) = \sum_{j_x=0}^{K-1} A_{j_x} \, \mathrm{e}^{i(\Delta\phi \, i_x \, j_x + \omega(k)\, t)} \quad , \text{ with } \quad \Delta\phi = \frac{2\pi}{N} \tag{2.16}$$

A visual representation is shown in figure 2.5, where different Gerstner waves are added together, according to equation 2.16, and can be seen at different time steps. In the figure, the principles of 2.15 is used to produce the different waves such that the angular velocities are: $\omega_1 = \omega$, $\omega_2 = 2\,\omega$, and $\omega_3 = 3\,\omega$.

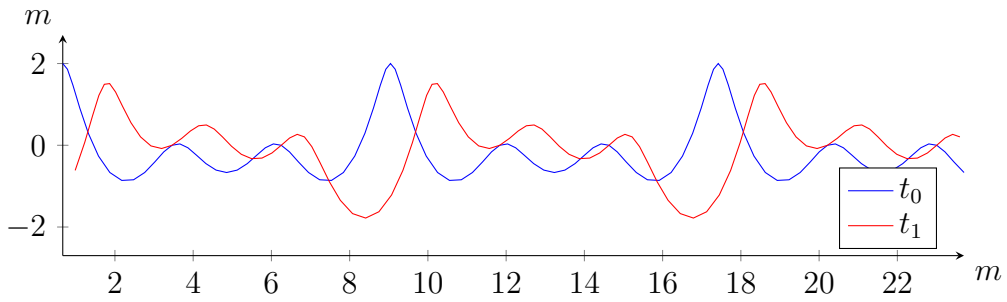


Figure 2.5: The result from adding together three Gerstner waves with different angular velocities. The blue shows the initial wave at time step t_0, and the red when it has moved to time step t_1. Note that the pattern repeats itself.

2.4.3 Expanding to 3D

Equation (2.16) can be expanded into waves with movements in the z-direction by the same logic as before, see equations (2.13)-(2.16). Assuming that for each wave in the x-direction, k_x, there also exists K number of waves in the z-direction, k_z. Each with indices $0 \leq j_z < K$. Turning (2.16) into (2.17), and the total number of waves in the sum to $K \times K$.

$$h(i_x, i_z, t) = \sum_{j_z=0}^{K-1} \sum_{j_x=0}^{K-1} A_{j_x,j_z} \, \mathrm{e}^{i\left(\frac{2\pi}{N} i_z j_z + \frac{2\pi}{N} i_x j_x + \omega(k)\, t\right)} \tag{2.17}$$

As can be seen from figure 2.6, the expansion turns the earlier mentioned row into a grid with size $N \times N$ in the xz-plane. The figure also shows how different wave numbers work, and their relation to the wave indices.

Although, since there are now horizontal movements in two directions, it is no longer possible to get these by taking the real part of h. Instead, the movements in the different directions, x and z, must be calculated separately. This is done by scaling h by its normalized wave numbers. See equation (2.18), where the horizontal movements in the x- and z-directions are given by the real part of x and z respectively. The height is still calculated by taking the imaginary part of h.

$$\begin{aligned} x(i_x, i_z, t) &= \sum_{j_z=0}^{K-1} \sum_{j_x=0}^{K-1} A_{j_x,j_z} \, \mathrm{e}^{i\left(\frac{2\pi}{N} i_z j_z + \frac{2\pi}{N} i_x j_x + \omega(k)\, t\right)} \frac{k_x}{k} \\ z(i_x, i_z, t) &= \sum_{j_z=0}^{K-1} \sum_{j_x=0}^{K-1} A_{j_x,j_z} \, \mathrm{e}^{i\left(\frac{2\pi}{N} i_z j_z + \frac{2\pi}{N} i_x j_x + \omega(k)\, t\right)} \frac{k_z}{k} \end{aligned} \tag{2.18}$$

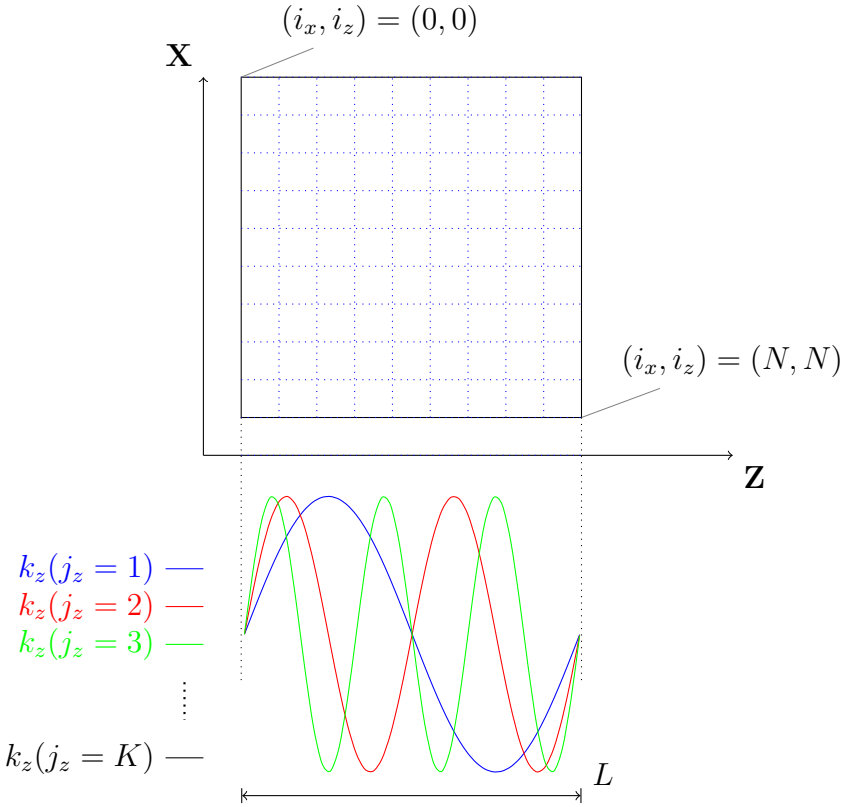


Figure 2.6: Definition of water plane indices, where i_x and i_z denotes the position of the grid while j_x and j_z denotes the different waves. The wave number k_z gets an additional period (2π) over L for every incremental increase of j_z.

2.4.4 Connect to existing models

In a paper by Flügge [32], the author explains that the height field of an ocean can be described by the following equation.

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\mathbf{k}\cdot\mathbf{x}} \tag{2.19}$$

By remembering that the expressions for position and wave number are $x = \frac{L}{N} i_x$ and $k_x = \frac{2\pi}{L} j_x$, it becomes clear that $k_x\, x = \frac{2\pi}{N} i_x\, j_x$. From this, equation (2.19) can be expanded to (2.20). This is equivalent to the 3D-expanded wave summation, (2.17), as long as $\tilde{h}(\mathbf{k}, t) = A_{j_x, j_z}\, \mathrm{e}^{i\omega t}$. The only difference is therefore that the time component is included in the amplitude, which becomes a complex number.

$$h(i_x, i_z, t) = \sum_{j_z} \mathrm{e}^{i\, \frac{2\pi}{N}\, j_z\, i_z} \sum_{j_x} \tilde{h}(j_x, j_z, t)\, \mathrm{e}^{i\, \frac{2\pi}{N}\, j_x\, i_x} \tag{2.20}$$

In the same paper, Flügge [32] explains that the amplitudes can be described by the following equation. Where $\xi_{r,i}$ are Gaussian distributed random numbers, with mean 0 and standard deviation 1, and $S(\mathbf{k})$ is a directional wave spectrum suitable for ocean waves.

$$\tilde{h}(\mathbf{k}, t) = (\xi_r + i\xi_i) \sqrt{\frac{S(\mathbf{k})}{2}}\, \mathrm{e}^{i\omega t} \tag{2.21}$$

2.4.5 Wave dispersion

To determine the wave spectrum and the angular velocity, a relationship between the traveling speed and the wavelength is needed. This relationship is called *Wave Dispersion*, and can, according to Horvath [33], be described by equation (2.22). Here, g is the gravitational constant, σ is the surface tension of water, ρ is water density, and d is water depth.

$$\omega(k,d)^2 = \left(g\,k + \frac{\sigma}{\rho}k^3\right)\tanh\left(k\,d\right) \tag{2.22}$$

In this project, as previously mentioned, a boat more than 10 meters long is simulated on the open ocean. Surface tension is too small to have an impact and d can be assumed to be large. Therefore, the equation can be simplified to

$$\omega(k)^2 = g\,k\,. \tag{2.23}$$

2.4.6 Directional spectrum

A directional wave spectrum, $S(\omega,\theta)$, is defined by Horvath [33] as a function that expresses the energy of an ocean, based on the wave's angular frequencies and direction relative to the wind. The spectrum can be written as the product of a non-directional wave spectrum, $S(\omega)$ and a directional spreading function, $D(\omega,\theta)$. Here, θ is the angle between the wave and the wind.

$$S(\omega,\theta) = S(\omega)\,D(\omega,\theta) \tag{2.24}$$

Horvath [33] says that, using the *Tessendorf Spectrum*, (2.24) can be rewritten as a function of wave numbers. Tessendorf scales the energy of the wave based on the gravitational constant g, magnitude of the wind speed $v = \sqrt{v_x^2 + v_y^2}$, and magnitude of the wave number vector k. He also adds a user-provided scaling factor, A_{Tess}, to determine the base amplitude, and therefor also the sharpness of the waves.

$$S(\omega,\theta) \approx S_{Tess}(\mathbf{k}) = A_{Tess}\frac{\exp\left(\frac{-g^2}{v^4}/k^2\right)}{k^4}\,D(\mathbf{k}) \tag{2.25}$$

There are a few different functions for *Directional Spread* proposed by Horvath [33], the main condition for these are that $\int_{-\pi}^{\pi} D(\omega,\theta)\,d\theta = 1$. One presented way is called the *Positive Cosine Squared* spreading function. This makes waves directly in line with, both along and against, the wind larger than those perpendicular to the wind. It is not the most accurate spreading function, but it is simple to calculate.

$$D(\omega,\theta) = \frac{\cos^2(\theta)}{\pi} \tag{2.26}$$

A benefit is that it can be written as the dot product between the unit vectors of $\mathbf{k}$ and $\mathbf{v}$, which are $\hat{\mathbf{k}} = \frac{\mathbf{k}}{k}$ and $\hat{\mathbf{v}} = \frac{\mathbf{v}}{v}$. Giving the entire spectra as (2.27). Note that the subscripts "*Tess*" are removed for simplicity, and that $1/\pi$ is a constant that can be included in A. This is the method used by Tessendorf *et al.* [31].

$$S(\mathbf{k}) = A\frac{\exp\left(\frac{-g^2}{v^4}/k^2\right)}{k^4}\,|\hat{\mathbf{k}}\cdot\hat{\mathbf{v}}|^2 \tag{2.27}$$

2.4.7 Completing the model

Combining equations (2.21), (2.23) and (2.27) gives the entire expression for the complex amplitude. Flügge [32] states that when using $D(\mathbf{k}) = |\hat{\mathbf{k}} \cdot \hat{\mathbf{v}}|^2$ it is possible to change the exponent to any positive number m. Higher m means waves will align more with the wind. A is a constant and can be brought outside of the square root.

$$\tilde{h}(\mathbf{k}, t) = (\xi_r + i\xi_i)\, A\, \frac{\exp\left(\frac{-g^2}{2\,v^4}/k^2\right)}{k^2}\, |\hat{\mathbf{k}} \cdot \hat{\mathbf{v}}|^m\, \mathrm{e}^{i\sqrt{g\,k}\,t} \tag{2.28}$$

For the FFT to work, the number of waves along one direction needs to equal the number of water particles along the same direction [31]. Meaning that $K = N$. All the wave indices are also shifted by $-\frac{N}{2}$, such that equation (2.15) turns into

$$k_x = \frac{2\pi}{L}\,(j_x - \frac{N}{2}) \quad . \tag{2.29}$$

This makes sure there are waves going in all directions, since there can only be waves going in the positive x-direction if k_x are strictly positive. These changes are applied to (2.20), giving (2.30).

$$\begin{aligned} h(i_x, i_z, t) &= \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,(j_z - \frac{N}{2})\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,(j_x - \frac{N}{2})\,i_x} \\ &= \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,(\frac{2\pi}{N}\,j_z - \pi)\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i(\frac{2\pi}{N}\,j_x - \pi)\,i_x} \end{aligned} \tag{2.30}$$

Since $\mathrm{e}^{i\,\pi} = -1$ and $\mathrm{e}^{i\,2\pi} = 1$, the $-\pi\, i_x$ and $-\pi\, i_z$ terms in the exponents will turn the sums positive or negative depending on if $i_{x,z}$ are even or not.

$$\begin{aligned} h(i_x, i_z, t) &= (-1)^{i_z} \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,j_z\,i_z}\, (-1)^{i_x} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,j_x\,i_x} \\ &= (-1)^{i_z + i_x} \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,j_z\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,j_x\,i_x} \end{aligned} \tag{2.31}$$

By adding an offset, $\mathbf{x_0} = (x_0, z_0)$, to the waves position, it is possible to move the water plane horizontally in the environment. These also need to be accounted for in the summation of the waves. Remembering from (2.19) that the phases of the waves depend on $\mathrm{e}^{i\mathbf{k}\cdot\mathbf{x}}$, means multiplying by $\mathrm{e}^{i\,k_x\,x_0}$ and $\mathrm{e}^{i\,k_z\,z_0}$ would shift the phases accordingly with the offset. This is shown in (2.32), included with the equations for the horizontal movements from (2.18).

$$
\begin{aligned}
h(i_x, i_z, t) &= (-1)^{i_z+i_x} \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,j_z\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,j_x\,i_x}\, \mathrm{e}^{i\mathbf{k}\cdot\mathbf{x_0}} \\
x(i_x, i_z, t) &= (-1)^{i_z+i_x} \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,j_z\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,j_x\,i_x}\, \mathrm{e}^{i\mathbf{k}\cdot\mathbf{x_0}}\, \frac{k_x}{k} + x_0 \\
z(i_x, i_z, t) &= (-1)^{i_z+i_x} \sum_{j_z=0}^{N-1} \mathrm{e}^{i\,\frac{2\pi}{N}\,j_z\,i_z} \sum_{j_z=0}^{N-1} \tilde{h}(\mathbf{k}, t)\, \mathrm{e}^{i\,\frac{2\pi}{N}\,j_x\,i_x}\, \mathrm{e}^{i\mathbf{k}\cdot\mathbf{x_0}}\, \frac{k_z}{k} + z_0
\end{aligned}
\tag{2.32}
$$

Together with (2.29), equations (2.28) and (2.32) describes the entire wave-system. Here it is possible to also shift the position indices $i_{x,z}$ by $-\frac{N}{2}$ if it is desired to keep the center of the water surface at index zero.

2.4.8 The Butterfly Algorithm RADIX-2

In total, there are $N \times N$ waves added together for every water particle. The FFT algorithm, *RADIX-2*, works by adding partial sums that can be reused multiple times at different positions.

Imagine having an array with size $N = 2^3$, and elements $[f_0, ..., f_{N-1}]^T$, with the goal of ending up with the sum of the array in all 8 spots. Let the results be called $[F_0, ..., F_{N-1}]^T$. If all additions where taken separately it would be $N \times N = 64$ additions. Instead, by adding together elements pairwise in *stages*, the number of additions becomes $N \log(N) = 8 \cdot 3 = 24$. This is the number of additions in figure 2.7, the blue elements show how the pairing should be done. As seen in the figure, each element is first paired up, and added, with its neighbor, elements (0,1). Then paired with one gap in between, elements (0,2). Lastly, they are paired with three gaps in between, (0,4). Note that if $N = 2^4 = 16$, then there would be a fourth stage with pairs (0,8). More generally, for $N = 2^n$, for any integer n, there will be n stages with pairs $(0,2^i)$ with $i = 1, 2, ..., n-1$.

$$
\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}
\Rightarrow
\begin{bmatrix} f_0 + f_1 = f_{01} \\ f_0 + f_1 = f_{01} \\ f_2 + f_3 = f_{23} \\ f_2 + f_3 = f_{23} \\ f_4 + f_5 = f_{45} \\ f_4 + f_5 = f_{45} \\ f_6 + f_7 = f_{67} \\ f_6 + f_7 = f_{67} \end{bmatrix}
\Rightarrow
\begin{bmatrix} f_{01} + f_{23} = f_{0123} \\ f_{01} + f_{23} = f_{0123} \\ f_{01} + f_{23} = f_{0123} \\ f_{01} + f_{23} = f_{0123} \\ f_{45} + f_{67} = f_{4567} \\ f_{45} + f_{67} = f_{4567} \\ f_{45} + f_{67} = f_{4567} \\ f_{45} + f_{67} = f_{4567} \end{bmatrix}
\Rightarrow
\begin{bmatrix} f_{0123} + f_{4567} = F_0 \\ f_{0123} + f_{4567} = F_1 \\ f_{0123} + f_{4567} = F_2 \\ f_{0123} + f_{4567} = F_3 \\ f_{0123} + f_{4567} = F_4 \\ f_{0123} + f_{4567} = F_5 \\ f_{0123} + f_{4567} = F_6 \\ f_{0123} + f_{4567} = F_7 \end{bmatrix}
$$

Figure 2.7: Simplified understanding of FFTs. Pairwise additions, according to the blue elements. The output array contains sums of all the input elements.

This is the core of the algorithm, but instead of only taking the sum of all elements, their phases are first shifted in each stage. Lets introduce a *Phase-shifting factor* according to (2.33), where N still is the number of points along one side of the water plane.

$$W_N = \mathrm{e}^{i\frac{2\pi}{N}} \tag{2.33}$$

The factor is a complex number, with amplitude equal to one, which is rotated $\frac{2\pi}{N}$ radians from its starting position in $W_N^0 = 1$. It has the properties that W_N^n will represent a rotation of $2\pi\frac{n}{N}$, for any real number n. Therefore, $W_N^N = 1$, $W_N^{N/2} = -1$, and $W_N^{N/4} = i$ describes one full rotation, one half, and one quarter respectively.

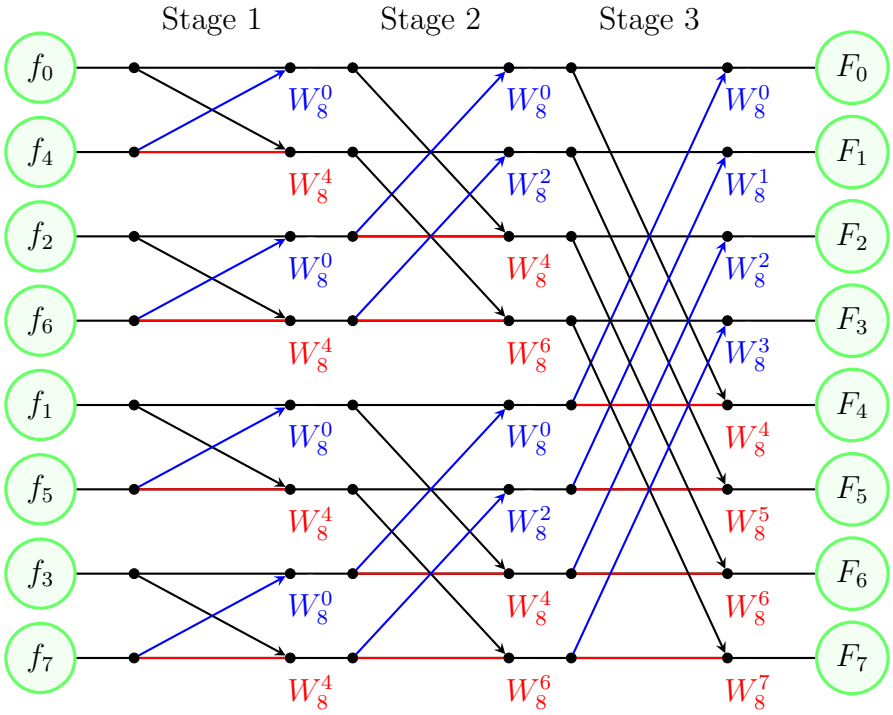


Figure 2.8: This describes the *Fast Fourier Transform* algorithm *RADIX-2*, also called the *Butterfly* algorithm. The input array is inserted in *bit-reversed* order and is outputted as *Fourier Sums*. In each stage, all elements are pairwise phase-shifted, by multiplying with $W_N^n = \mathrm{e}^{i\frac{2\pi}{N}n}$ (where $0 \leq n < N$ are integers), and then added together. Additions are read from left to right by following the lines. The lines are multiplied with the W_N^n with the corresponding color. Here, the array size $N = 2^3$, means there are 3 stages. More generally, there are m stages for $N = 2^m$.

Figure 2.8 shows the full RADIX-2 algorithm. The green circles on the left and right are inputs and outputs respectively. Each horizontal black line connecting them corresponds to a position in the array, and the black dots represent when values are copied from, or added to the element in the array. Note that the order of the input array is *bit-reversed* before it is inserted into the algorithm. The index 4 is 100 in binary, and will therefore become 001, which corresponds to index 1. This means elements 1 and 4 will switch places, other elements will switch places in the same way. This is done for the final outputted Fourier sums $[F_0, ..., F_{N-1}]^T$ to be in the correct order, without further modifications.

For each stage, there are $\frac{N}{2}$ number of *crosses*, pairs to be phase-shifted and added, and a differing amount of *sections*, non-overlapping crosses within one stage. Stage 1 has $\frac{N}{2}$ number of sections, this is halved for each stage until there is only one left. The number of crosses in each section starts at one, and is doubled for each stage until all $\frac{N}{2}$ crosses are included in one section. The first element in each cross is copied and added without any phase-shift, as indicated by the black lines between the dots. The second element is also copied, but each copy is shifted in phase by different amounts, as shown by the different colored lines and their corresponding phase-shifting factors.

In all of the sections, the first blue factor is always raised to the power of 0, and the first red is always raised to $\frac{N}{2}$. For the first stage, these are the only two factors present. For each successive stage, the difference in power between the crosses in one section can be described as $\Delta n = N/2^{\text{stage}}$. Meaning that $\Delta n = 2$ for stage 2, and $\Delta n = 1$ for stage 3.

2.4.9 Alternative Spreading Functions

According to Horvath [33], there are some improvements that can be made to Tessendorfs model, and other models which might be even more realistic. A few of the proposed models are outlined here in this section.

Mitsuyasu Directional Spreading

Mitsuyasu improved the cosine squared model, equation (2.26), by adding a shaping factor s. The factor was was tuned based on movements of a cloverleaf bouy [33]. To get the correct behavior, the shaping factor affects the equation through a function $Q(s)$, which uses the *gamma function* $\Gamma()$.

$$D(\omega, \theta) = Q(s)|\cos(\theta/2)|^{2s} \tag{2.34}$$

$$Q(s) = \frac{2^{2s-1}}{\pi} \frac{\Gamma(s+1)^2}{\Gamma(2s+1)} \tag{2.35}$$

The factor s is defined by (2.36), where ω_p is the *peak angular velocity*, and U is the average wind speed.

$$s = \begin{cases} 11.5(\frac{\omega_p U}{g})^{-2.5} (\frac{\omega}{\omega_p})^5 & \text{, for } \omega \leq \omega_p \\ 11.5(\frac{\omega_p U}{g})^{-2.5} (\frac{\omega}{\omega_p})^{-2.5} & \text{, for } \omega > \omega_p \end{cases} \tag{2.36}$$

Hasselmann Directional Spreading

Hasselmann modified the shaping factor further, by implementing a more complex exponent; to fit a larger dataset [33].

$$s = \begin{cases} 6.97\,(\frac{\omega}{\omega_p})^{4.06} & \text{, for} \quad \omega \leq \omega_p \\ 9.77\,(\frac{\omega}{\omega_p})^{s_h} & \text{, for} \quad \omega > \omega_p \end{cases} \tag{2.37}$$

$$s_h = -2.33 - 1.45\left(\frac{\omega_p U}{g} - 1.17\right) \tag{2.38}$$

Donelan-Banner Directional Spreading

Banner and Donelan proposed an alternative approach, in the form of equations (2.39) - (2.41), since they found Mitsuyasu/Hasselmann inadequate [33]. They introduced fitting parameters β_s and ϵ in order to get better results.

$$D(\omega, \theta) = \frac{\beta_s}{2\,\tanh(\beta_s\,\pi)}\,\mathrm{sech}(\beta_s\,\theta)^2 \tag{2.39}$$

$$\beta_s = \begin{cases} 2.61\,(\omega/\omega_p)^{1.3} & \text{, for} \quad 0.00 \leq \omega/\omega_p < 0.95 \\ 2.28\,(\omega/\omega_p)^{-1.3} & \text{, for} \quad 0.95 \leq \omega/\omega_p < 1.60 \\ 10^{\epsilon} & \text{, for} \quad 1.60 \leq \omega/\omega_p \end{cases} \tag{2.40}$$

$$\epsilon = -0.4 + 0.8393\,\exp\Big[-0.567\,\ln\big((\omega/\omega_p)^2\big)\Big] \tag{2.41}$$

Chapter 3

Method

This chapter contains the method, this includes: a list of every relevant program and procedure used, in depth description of system architecture, simulation environment, and training manners. Furthermore, an overview of the project and its major parts is presented; closing with a brief representation of the hardware.

3.1 Project overview

Figure 3.1, illustrates the overview of the project and the workflow, showing how the project is executed.

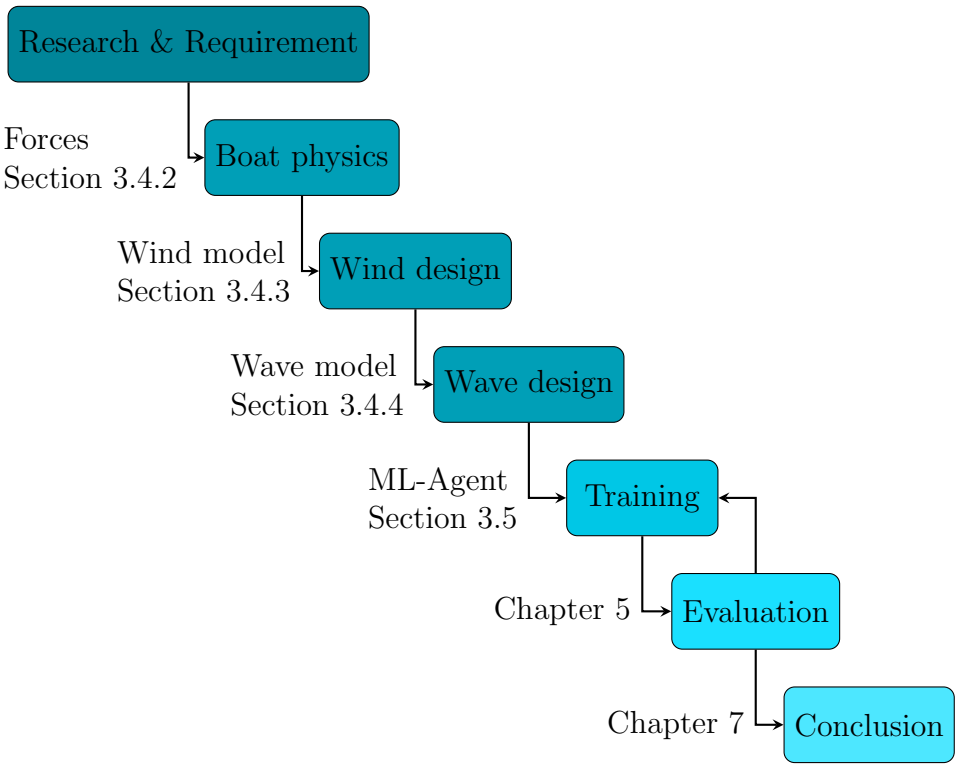


Figure 3.1: Waterfall model describing the project method.

3.2 Method definition

The simulations are carried out entirely in *Unity*, a real-time development platform, also known as a *game engine*. User-defined scripts that are added to Unity, are written in the object-oriented programming language C#. Hence, from here on, the words "method" and "function" are used interchangeably. In Unity, a class is usually constructed of three main methods: *Start*, *Update*, and *FixedUpdate*. The Start function is run once, as the name suggests when the program is launched. Update, is a frame-based method, meaning it runs once every new frame. Finally, FixedUpdate has an almost fixed run frequency (independent of FPS), that usually is much greater than the one of the Update function. This is utilized when computing physical calculations, such as applying a force or a torque to an object. As said, these are the main functions, meaning, there are additional built-in functions, but they are not used extensively in this report.

Solidworks and *Mesh-lab* are modeling programs used to modify 3D objects in virtual environments. In this project, they are mostly used for file conversions, and to change the complexity of the boat's geometries. This makes it possible to optimize between computing times and accuracy , and to completely remove unimportant parts like railings . The dimensions and properties of the boat, including driveline configuration and weight placements, is determined by meetings with engineers at Volvo Penta.

In order to train the AP in Unity, the Python package *ML-Agents* is used. According to its documentation, REF, an agent is defined as the one seeing observations and taking actions. In this case, the agent is by definition the same as the boat. However, if multiple boats are used, they can still count as a single agent, as long as they all have the same policy.

CPAC supplies the project with computers and together with Volvo Penta, data regarding various marine vessels. Unity licenses are carried out under the "Unity Student Plan".

3.2.1 Hardware

The computer used for running the simulations contains a Ryzen 7 3700X 8-Core Processor from AMD, 32 GB of RAM memory and an NVIDIA GeForce GTX 1660. Using this specific hardware, the computer managed to run three simulations with five instances at the same time, and throughout the project, approximately 65 simulation runs were performed, where one run took approximately 24 hours.

3.3 Procedure

The autopilot consists of two parts, a *Regulator* and a *Control Allocator*. An illustration of the system is seen in Figure 3.2, where the input of the regulator, e, is the error between the boat's current position, labeled as *pos*, and the desired *path*; which consists of *waypoints*. The control signal u, from the regulator, is then passed to the NN, which is the main part of the control allocator.

The control allocator consists of a NN, trained using RL, and a block called *Capabilities*; which holds information about all the actuators of the vessel, their RPM, and rotation. The output from the NN is the control vector $\vec{q}$, which contains the requested RPM and rotation for every actuator. These requested values are verified within the capabilities block and later sent to the *boat model*, seen as $\vec{y}$, which updates the actuators to the requested values.

Moreover, the NN of the control allocator is fed with information about the current capacity of each actuator from the capabilities block. The stored capacity for the actuators can change due to wear and tear or user input. Finally, the boat's position is updated from the forces generated by the actuators and environmental disturbances. The cycle continues as long as there is a path available for the ship to traverse.

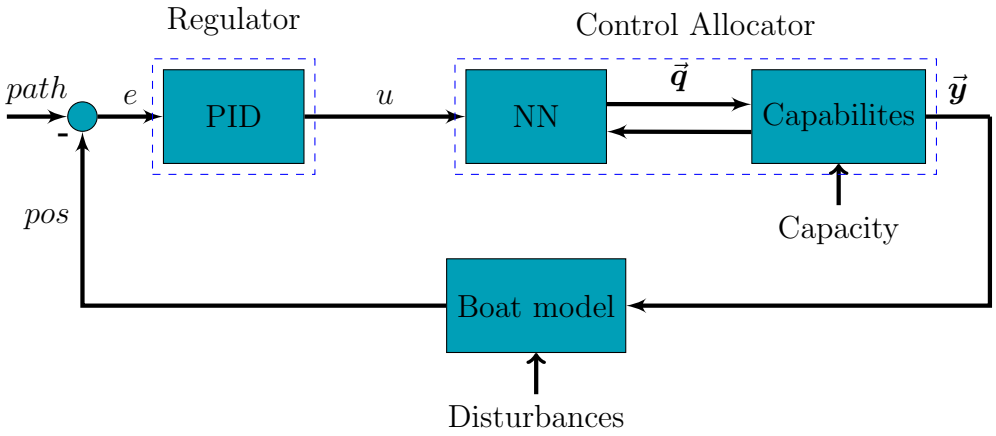


Figure 3.2: System architecture, highlighting the regulator and the control allocator that make up the autopilot.

3.3.1 Regulator

The part in charge of regulating the boat's position, i.e. the regulator, is a PID. Its input, e, is the *Cross Track Error* (CTE), the interpolation between CTE1 and CTE2, illustrated in Figure 3.3.

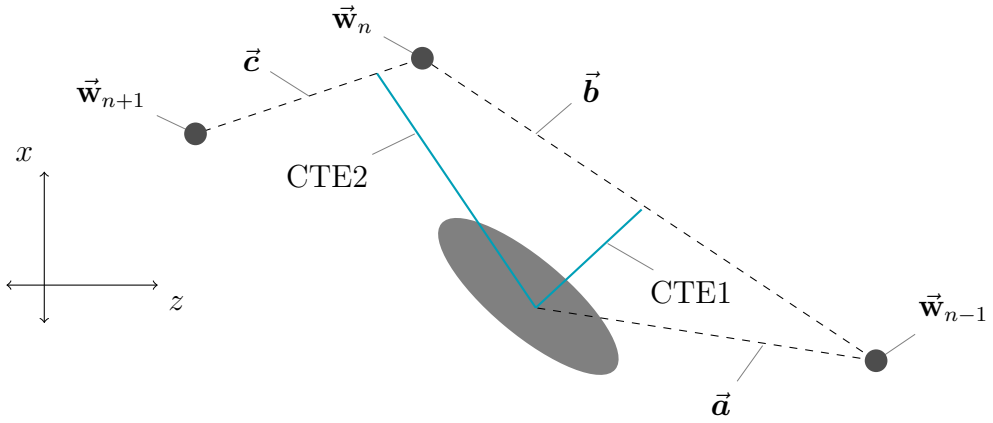


Figure 3.3: CTE1 and CTE2 are the perpendicular distances to the lines represented by vectors $\vec{b}$ and $\vec{c}$, which goes between the different waypoints $\vec{\mathbf{w}}$. CTE1 is calculated by projecting vector $\vec{a}$ onto $\vec{b}$. The same logic is used for CTE2.

As seen, CTE1 and CTE2 are the perpendicular distances from the boat to the vectors $\vec{b}$ and $\vec{c}$ respectively, see equation (3.1). The two CTEs are being interpolated according to Figure 3.4, $||\vec{b}||$ is the distance between waypoint $\vec{\boldsymbol{w}}_n$ and $\vec{\boldsymbol{w}}_{n-1}$.

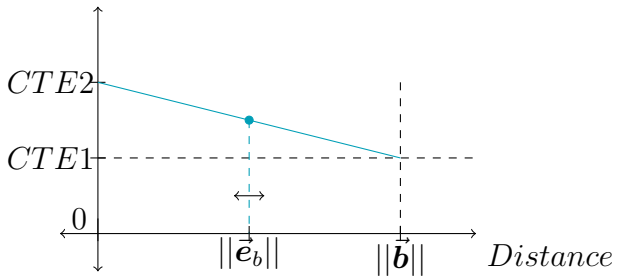


Figure 3.4: Graph showing how the CTE is interpolated between CTE1 and CTE2.

This whole process is known as *Vector projection*, temporary vectors are created to make the equations more readable:

$$\begin{aligned} \vec{a} &= Boat_{pos} - \vec{\boldsymbol{w}}_{n-1} \\ \vec{b} &= \vec{\boldsymbol{w}}_n - \vec{\boldsymbol{w}}_{n-1} \\ \vec{c} &= \vec{\boldsymbol{w}}_{n+1} - \vec{\boldsymbol{w}}_n \end{aligned} \tag{3.1}$$

These vectors can be seen in Figure 3.3, they represent the dashed lines, and are used to compute the progress; the projected magnitude of vector $\vec{a}$ onto vectors $\vec{b}$ and $\vec{c}$. In the current context, the progress can only be between 0 and 1 since otherwise the vessel would have passed the nearest waypoint.

$$\begin{aligned} prog_b &= \frac{a_x \cdot b_x + a_z \cdot b_z}{b_x^2 + b_z^2} \qquad 0 \leq prog_b \leq 1 \\ prog_c &= \frac{a_x \cdot c_x + a_z \cdot c_z}{c_x^2 + c_z^2} \qquad 0 \leq prog_c \leq 1 \end{aligned} \tag{3.2}$$

The progress is later used to calculate the *error point*, $\vec{e}$, from which the perpendicular vector shall begin. The progress is here seen as a scalar with vectors $\vec{b}$ and $\vec{c}$ as directional vectors. Further, it can also be said that the magnitude, the length of the error-point vector, will always be less or equal to the length of the dashed vector lines i.e. vector $\vec{b}$ and $\vec{c}$.

$$\begin{aligned} \vec{e}_b &= \vec{w}_{n-1} + prog_b \cdot \vec{b} \qquad 0 \leq ||\vec{e}_b|| \leq ||\vec{b}|| \\ \vec{e}_c &= \vec{w}_n + prog_c \cdot \vec{c} \qquad 0 \leq ||\vec{e}_c|| \leq ||\vec{c}|| \end{aligned} \tag{3.3}$$

Further, the CTE is defined as the magnitude of the vector between the boat's position and the error point.

$$\begin{aligned} CTE1 &= ||(\vec{e}_b - Boat_{pos})|| \\ CTE2 &= ||(\vec{e}_c - Boat_{pos})|| \end{aligned} \tag{3.4}$$

Moreover, the PID parameters for the proportional, integral, and derivative parts are tuned manually. This is done by controlling the boat and plotting the CTE as the boat traverses the path. The final settings for the gains are: $\{0.5, 1.4, 10\}$ for K_p, K_i, and K_d respectively. The output of the PID is seen as an average steering degree, which implies that there is a possibility of running the AP alone without the control allocator. The drawbacks are stability, lack of remapping, and lack of optimal steering.

3.3.2 Control Allocator

The control allocator, which decides how each actuator should operate [34], consists of a NN and a capabilities block, as described earlier in Figure 3.2. In section 3.5, the type of network is described along with how the network is trained. Different types of neural networks with different settings are explored and covered in the discussion.

3.4 Simulation Environment

As previously described, the simulation environment is created in Unity. The environment is used by ML-Agents as a training environment for the AP. Figure 3.5 illustrate the hierarchy of objects in the simulation environment. The parent node, *Environment*, controls the wind, waves, and the physics connected to each boat; described in sections 3.4.2-3.4.3. The image also depicts the ability to have multiple instances, as shown by the child object Sea_n. In the current setup, there can be a maximum of 20 different boats; since this is the number of free layers within Unity. Layers are a way to make objects non-interactive, this is utilized to make the different vessels not crash into each other.

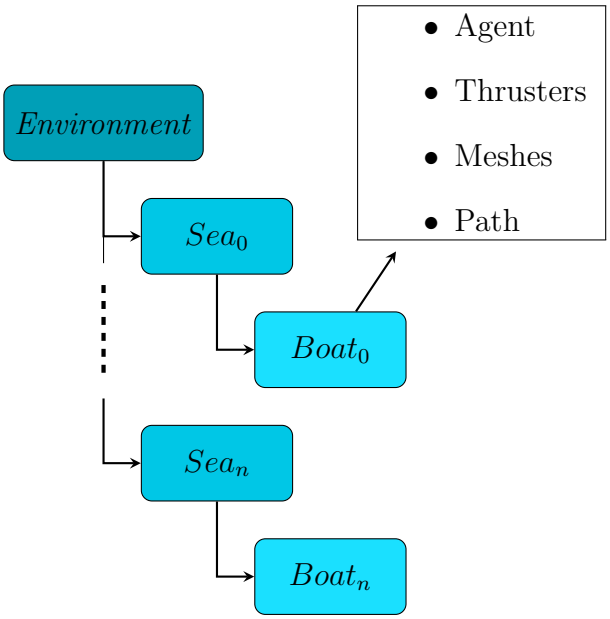


Figure 3.5: Hierarchy describing the simulation environment.

Each *Boat* object contains several minor parts, such as thrusters, the path, additional meshes, and the agent. As described, the boat contains two Azimuth thrusters. In Unity these are modeled as empty game objects, they therefor only contain a position and a direction (pointing outwards from the boat's stern). The path is made up of many waypoints, this is explained in section 3.4.5. When the boat is traversing the path, a new waypoint is spawned in such a fashion that there always are two points ahead of the boat. Further, each boat contains two additional meshes, these describe which parts of the boat are submerged and which are not. The method for extracting this information is touched upon in section 3.4.1. Finally, the boat object contains the agent that is trained with ML-Agents.

To test the simulation environment the boat is controlled manually using a keyboard, known as heuristic control. The wind speed is also able to be adjusted manually, to get a sense of how well the environment scales at greater disturbances.

3.4.1 Boat model

In Unity, the boat model consists of a *Rigidbody* and a *Mesh Renderer/Collider*. The rigidbody gives the boat its properties, such as mass, drag and angular drag. Several triangles and vertices make up the mesh and ultimately define the shape of the vessel. Figure 3.6 depict the boat model in Unity with wireframe rendering, showing the vertices and triangles. The image also shows submerged parts drawn in red and others yellow. Apart from the original mesh defining the body of the boat, the model contains two empty meshes that update every new frame, these meshes are yellow and red which is illustrated in Figure 3.6. This is to know which parts of the boat should be exposed to underwater forces, and similarly, forces generated above water.

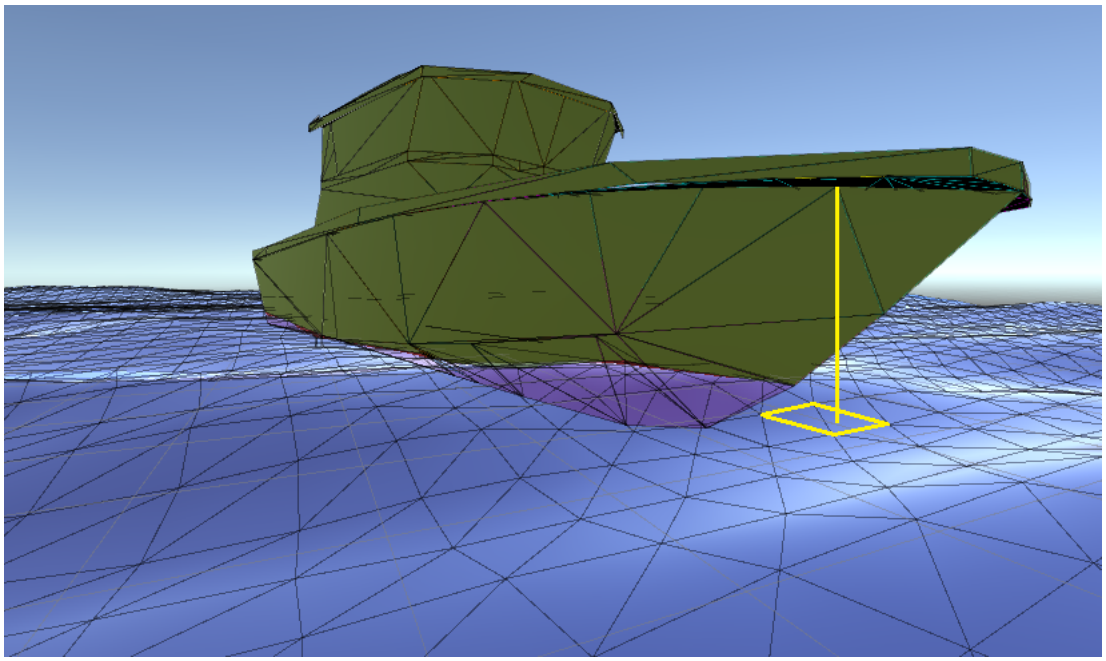


Figure 3.6: The boat model illustrates the vertices and triangles in the mesh and how the water plane is splitting the mesh into two parts. One of the boat's vertices is projected downwards onto the water plane, visualized by the yellow square.

To construct these meshes, each vertex in the original boat mesh is projected downwards to find the four closest surrounding points on the water plane. The water heights for these points are extracted and interpolated to find an approximation of the height of the water at the boat vertex. The height is then subtracted from the boat's vertex, to determine if it is submerged or not. Figure 3.6, is displaying this by the yellow square drawn on the water plane, along with the yellow line illustrating the boat's vertex being projected downwards.

Further, if all three vertices in a triangle on the boat mesh are below the water level, this is added to the red mesh. Otherwise, if all vertices are above the water plane it is added to the yellow mesh. From here on, these are referred to as underwater-mesh and abovewater-mesh. In the case, where one vertex is above the water (two below) or the opposite, two vertices are above (one below), the triangle is split into smaller ones and added to the respective mesh. This whole process is in Algorithm 1.

Moreover, the number of actuators used in the simulation model is two Azimuth thrusters, as described in section 1.3. No propeller dynamics are included, instead, the thrusters are modelled similar to a jet thruster. Meaning, that given a certain RPM, a certain propulsion force is produced; according to Figure 3.7. Each thruster also contains information about max allowed RPM and rudder angle. Here, these are set to $3500rpm$ and $\pm 30^{\circ}$ respectively.

Algorithm 1 Algorithm to generate under/above -water mesh

```
for each boat vertex do
    heights ← vertex_y − WaterHeight(vertex)
end for
for each triangle do
    (p1, p2, p3) ← heights                ▷ All vertices that make up the triangle
    if p1 < 0 & p2 < 0 & p3 < 0 then
        underMesh ← triangle              ▷ All vertices are below the water
    else if p1 > 0 & p2 > 0 & p3 > 0 then
        aboveMesh ← triangle              ▷ All vertices are above the water
    else
        SplitTriangle(triangle)           ▷ 1 or 2 vertices are above the water
    end if
end for
```

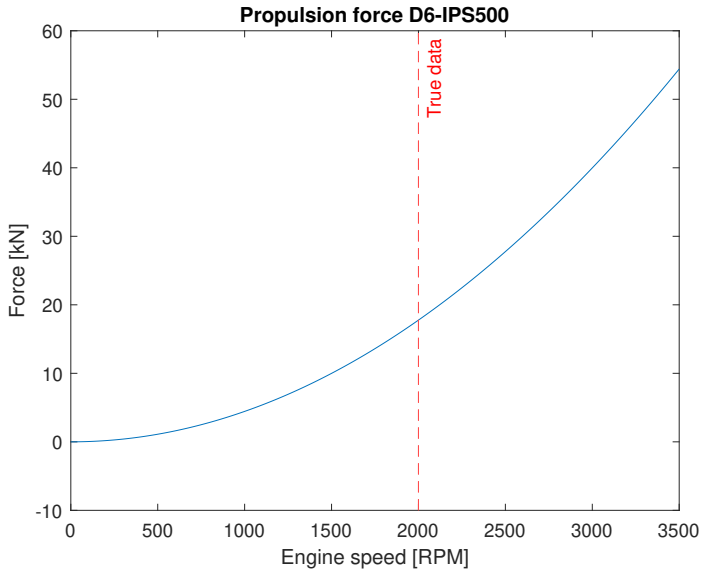


Figure 3.7: D6-IPS500 Engine propulsion force vs RPM, above 2000 RPM the propulsion force is estimated.

3.4.2 Additional forces from the environment

From the simulation environment in Unity, there are multiple forces that affect the boat model. They are all explained and defined in this section. Keep in mind that all these forces are calculated and applied for every triangle in the boat mesh, as explained in section 3.4.1. The forces are applied to the boat using the Unity function *AddForceAtPosition(force, triangle$_{center}$)*.

Apart from the wind force, explained in 3.4.3 , air resistance force is a force acting in the opposite direction of the ship's velocity. This force is applied to the part of the boat being above the water's surface.

$$\vec{\boldsymbol{F}}_{res.air} = -\frac{1}{2} \cdot \rho_{air} \cdot \vec{\boldsymbol{v}}^2 \cdot A_p \cdot C_{air} \qquad \text{(N)} \tag{3.5}$$

In equation (3.5), ρ_{air} is the density of air ($= 1.225\ kg/m^3$), v is the boat's velocity in m/s, A_p is the transverse projected area in m^2 and C_{air} is the drag coefficient of the boat ($= 0.85$). This specific value is appointed by Volvo Penta, referred to as forward drag.

Further, every force that follows is applied to the part of the boat being submerged, i.e. the underwater-mesh. Starting with buoyancy force, which is an upwards pointing force from the water surface that opposes the boat's gravitational force, no matter if the vessel is being partially or fully submerged. The horizontal components of the force cancel out, as shown by setting the x- and z-component to $0N$. The force is defined by Archimedes' principle.

$$\vec{\boldsymbol{F}}_{buoyancy} = \rho_{water} \cdot g \cdot V, \qquad F_{buoyancy}.x = F_{buoyancy}.z = 0 \qquad \text{(N)} \tag{3.6}$$

$$V = y \cdot A_{sur} \cdot \vec{\boldsymbol{N_{tri}}} \tag{3.7}$$

In equation (3.6), ρ_{water} is the water density ($= 1027\ kg/m^3$ for ocean water), g is the gravitational constant in m/s^2. In equation (3.7), y is the distance to the water surface in m, A_{sur} is the surface area of the triangle in m^2 and $\vec{\boldsymbol{N_{tri}}}$ is the normal vector of that triangle.

Very much like the air, the water is also resistant to change and thereby generates a force that works in the opposite direction of the flow, this is called Viscous water resistance force.

$$\vec{\boldsymbol{F}}_{res.water} = -\frac{1}{2} \cdot \rho_{water} \cdot \vec{\boldsymbol{v}}^2 \cdot A_{sur} \cdot C_f \qquad \text{(N)} \tag{3.8}$$

$$C_f = \frac{0.075}{(\log_{10}(R_n) - 2)^2}, \text{ where} \qquad R_n = \frac{u \cdot L}{v} \tag{3.9}$$

C_f is the coefficient for frictional resistance. R_n is the Reynolds number that is defined by the flow speed u $[m/s]$, length of the submerged body L $[m]$ and the kinematic viscosity of the fluid v $[m^2/s]$ [35]. The viscosity for sea water at 20°C is 10^{-6} $[m^2/s]$.

Moreover, pressure/suction-drag force is an approximate force that is based on a combination of resistive forces. According to Kerner [36], it is not uncommon for scientists to separate between different resistance forces, such as wave pattern resistance, spray resistance etc. The idea behind the pressure/suction-drag force is to account for all of these and in that sense function as the total resistance for the boat moving in the water.

The pressure and suction forces are separated based on the direction the water is traveling. If the water is moving towards the ship's hull, pressure drag force will arise. Otherwise, if the water is moving away from the hull, a suction drag force is created. As the previously cited article also mentioned, an alternative to this is to numerically solve the "Navier-Stokes equation", which explains the flow of fluids and gases [37].

$$\vec{\boldsymbol{F}}_{drag} = \begin{cases} -(C_{PD1} \cdot \vec{\boldsymbol{v_{tri}}} + C_{PD2} \cdot \vec{\boldsymbol{v_{tri}}}^2) \cdot A_{tri} \cdot (\vec{\boldsymbol{v_{tri}}} \cdot \vec{\boldsymbol{N_{tri}}})^{f_p} \cdot \vec{\boldsymbol{N_{tri}}} & \text{if } \vec{\boldsymbol{N_{tri}}} \cdot \vec{\boldsymbol{v_{tri}}} > 0 \\ (C_{SD1} \cdot \vec{\boldsymbol{v_{tri}}} + C_{SD2} \cdot \vec{\boldsymbol{v_{tri}}}^2) \cdot A_{tri} \cdot |\vec{\boldsymbol{v_{tri}}} \cdot \vec{\boldsymbol{N_{tri}}}|^{f_s} \cdot \vec{\boldsymbol{N_{tri}}} & \text{if } \vec{\boldsymbol{N_{tri}}} \cdot \vec{\boldsymbol{v_{tri}}} \leq 0 \end{cases} \tag{3.10}$$

Every C in the equation, along with $f_{p,s}$ are user-specified coefficients. A_{tri} is the area of the submerged triangle, similarly $\vec{\boldsymbol{N}}_{\boldsymbol{tri}}$ is the normal vector from the triangle and finally, $\vec{\boldsymbol{v}}_{\boldsymbol{tri}}$ is the velocity of the triangle.

In order for the ship model to drift along the waves, a wave drift force is derived from the direct pressure integration [38]. This force is an approximation, hence, the y-component of the force is fixed to $0N$ to avoid unexpected behaviour of the boat. The idea is that the buoyancy force will take care of the upwards-pointing component as the wave profile is traversing across the boat's hull.

$$\vec{\boldsymbol{F}}_{drift} = \frac{1}{2} \cdot \rho_{water} \cdot g \cdot A^2 \cdot \vec{\boldsymbol{N}}_{\boldsymbol{tri}}, \qquad F_{drift}.y = 0 \qquad \text{(N)} \tag{3.11}$$

A is the amplitude of the wave in m, and $\vec{\boldsymbol{N}}_{\boldsymbol{tri}}$ is the normal from the partial triangle area of the ship. ρ_{water} and g are, as explained earlier; the density of the water and respectively the gravitational constant.

Lastly, slamming is a force that works on the hull of the ship at the sea surface when the boat raises above the water and subsequently *slams* on it.

$$\vec{\boldsymbol{F}}_{slam} = \frac{2 \sum A_j^{sub} \cdot Cos(\theta_j)}{S_{tot}} \cdot m \cdot \vec{\boldsymbol{v}} \tag{3.12}$$

Where $A_j^{sub} \cdot Cos(\theta_j)$ is the submerged part of the triangle area being projected onto a plane that is perpendicular to the boat's velocity $\vec{\boldsymbol{v}}$. S_{tot} is the total surface area of the boat in m^2, m is the boat mass in kg.

3.4.3 Wind model

The wind speed V_w is simulated using a low- and high-frequency component, x_1 and x_2, known as average wind speed and wind gust. In the initialization stage these are set to, 5 and 0.5 knots. The wind angle ψ_w is defined in a similar way using two components [23]. In Unity, the average wind speed and the wind gust are both user-defined variables; applied in the parent node "Environment". The wind angle ψ_w, is initialized as a random value; $0 \leq \psi_w \leq 359°$. The two components for the wind angle x_3, x_4, are defined as 90% and 10% respectively of the initial wind angle.

$$V_w(t) = x_1 + x_2, \qquad \psi_w(t) = x_3 + x_4, \qquad (\text{kn}, °) \tag{3.13}$$

Where $\dot{x}_i = w_i$ and $\dot{x}_j = -\frac{1}{T}(x_j - Kw_j)$ $(i = 1, 3,\ j = 2, 4)$, respectively, w are zero mean Gaussian white noise processes, with $std(w_{1,2}) = 0.5$ and $std(w_{3,4}) = 2$. T and K are the time and gain constants from the linear 1-st order approximation of the Harris spectrum [39], defined as:

$$T = \sqrt{286/V_w(0)}, \qquad K = \sqrt{5286 \cdot k \cdot V_w(0)} \tag{3.14}$$

In equation (3.14), k is the turbulence factor $(= 0.05)$, and $V_w(0)$ is the initial wind speed in knots. The relative wind speed V_R, relative to the speed of the vessel, is defined as:

$$V_R = \sqrt{u_R^2 + v_R^2}, \qquad \text{(kn)} \tag{3.15}$$

Where the x- and z-components of V_R, u_R and v_R, are:

$$\begin{aligned} u_R &= V_w(t)Cos(\gamma_R) - u + u_c \qquad \text{(kn)} \\ v_R &= V_w(t)Sin(\gamma_R) - v + v_c \qquad \text{(kn)} \end{aligned} \tag{3.16}$$

Here, in (3.16), γ_R is the relative wind angle, the angle between the ship's bow and wind direction, $\gamma_R = \psi_w - \psi$. (u, v) are the vessels velocity components and (u_c, v_c) are the currents velocity components. This is illustrated in Figure 3.8.

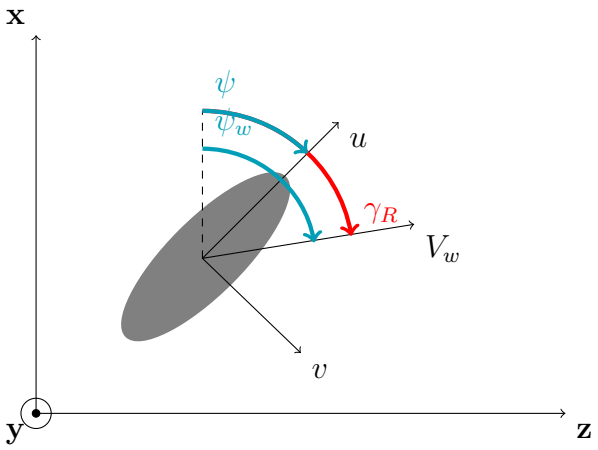


Figure 3.8: Illustration of wind angle and speed, γ_R is the relative angle between the boat's heading ψ, and the wind angle ψ_w. V_w is the wind speed and (u, v) are the boat's velocity components.

The resulting body-fixed forces and moment acting on the ship model are based on Isherwood [40], and later confirmed by Aage, Hvid, Hughes, *et al.* [41]. In Unity these are applied through *AddRelativeForce(force)*, and defined as:

$$\begin{aligned} \vec{\boldsymbol{F}}_x &= \frac{1}{2} \cdot C_X(\gamma_R) \cdot \rho_w \cdot V_R^2 \cdot A_T, \qquad \text{(N)} \\ \vec{\boldsymbol{F}}_z &= \frac{1}{2} \cdot C_Z(\gamma_R) \cdot \rho_w \cdot V_R^2 \cdot A_L, \qquad \text{(N)} \\ \overset{\frown}{\boldsymbol{M}}_y &= \frac{1}{2} \cdot C_N(\gamma_R) \cdot \rho_w \cdot V_R^2 \cdot A_L \cdot L, \qquad \text{(N m)} \end{aligned} \tag{3.17}$$

In equation (3.17), A_L, A_T and L are the lateral projected area of the boat above the water surface in m^2, transverse projected area in m^2, and overall length in m. C_X, C_Z and C_N are wind force/moment coefficients, ρ_w is the air density.

According to Isherwood, the coefficients are best fitted following these equations:

$$\begin{aligned} C_X &= A_0 + A_1\frac{2A_L}{L^2} + A_2\frac{2A_T}{B^2} + A_3\frac{L}{B} + A_4\frac{S}{L} + A_5\frac{C}{L} + A_6 M \\ C_Z &= B_0 + B_1\frac{2A_L}{L^2} + B_2\frac{2A_T}{B^2} + B_3\frac{L}{B} + B_4\frac{S}{L} + B_5\frac{C}{L} + B_6\frac{A_{SS}}{A_L} \\ C_N &= C_0 + C_1\frac{2A_L}{L^2} + C_2\frac{2A_T}{B^2} + C_3\frac{L}{B} + C_4\frac{S}{L} + C_5\frac{C}{L} \end{aligned} \tag{3.18}$$

Equation (3.18), contains A_i, B_i and C_j ($i = 0...6, j = 0...5$), which are tabulated variables based on γ_R in Isherwood [40]. The resulting coefficients, C_X, C_Z, and C_N, for the *test* boat are in Appendix A. Remaining parameters are:

B = beam
S = perimeter length of the entire lateral projected area
C = distance from the bow to the centroid of the lateral projected area
M = number of groups of masts seen in the lateral projection
A_{SS} = lateral projected area of the superstructure

In (3.18), A_i, B_i and C_j are only tabulated for $0 \leq \gamma_R \leq 180$ with increments of 10 °. Hence a 9-th order polynomial is created and fitted to the coefficients using *Matlab's polyfit()* function. The resulting functions can be found in Appendix A.

3.4.4 Wave model

The waves are modeled based on the theory in section 2.4. In Unity, all these calculations are implemented in a single C#-script that is applied to the "Environment" object, shown in figure 3.5. Initially, a standard 2D-plane gameobject, named Sea_0, is manually created. This plane is given a mesh, and a renderer/collider. Note that the script can be accessed and shared by all of the future water planes, since it is applied to their combined parent.

Global variables

Initially, the script starts with a few generic user defined global variables. The first three are set based on user preference, the last three are discussed in section 2.4.2 and in equation (2.27). Note that the condition $N = 2^n$, for any integer n, still must be true. Since N is the number of points in one direction, the total amount of points will be N^2.

n_{seas} = number of water planes
Δx_{start} = distance between planes in x-direction at the start
Δz_{start} = distance between planes in z-direction at the start
N = number of points along one side of each plane
L = length of each plane
A = modified Tessendorf amplitude

Three global matrices, $\boldsymbol{H}$, $\boldsymbol{X}$, and $\boldsymbol{Z}$, with sizes $N \times N$, are created as placeholders for the movements of these points in the corresponding directions. Two additional global matrices, $\boldsymbol{\xi_r}$ and $\boldsymbol{\xi_i}$, will hold the randomness parameters for all the Gerstner waves. These parameters are the same Gaussian distributed values seen in (2.28). By remembering that $K = N$, these additional matrices therefore also have sizes $N \times N$. Their values are generated at the start and are kept constant during the entire simulation. Lastly, a global list of meshes, with length n_{seas}, is created to store the meshes of all the water planes.

Start method

In the start-method, the number of vertices in the mesh is adjusted to fit N, and their initial positions are set as a flat grid with spacing L/N. This is done by looping over i_x and i_z in (3.19), taken from equation (2.13), for the entire grid in a type-writer manner.

$$\begin{aligned} x(i_x) &= \tfrac{L}{N} i_x \\ z(i_z) &= \tfrac{L}{N} i_z \end{aligned} \tag{3.19}$$

The vertices in the mesh are ordered with indices $0 \leq i_v < N^2$. Triangles are created by arranging the indices in groups of three in a list, where each group connects three points into a visible triangle in the mesh. See figure 3.6.

Sea_0 now has a visible mesh and can be duplicated and moved to new positions in the environment, using the *Instantiate()* function. New copies are instantiated as Sea_{old+1}, and connected to their respective mesh in the list by *GetComponent()*. The offsets (Δx_{start}, Δz_{start}) are added to their vertices in the x- and z-direction respectively, moving them to their new locations. Note that either Δx_{start} or Δz_{start} must be larger than L for the planes to not overlap at the start in the simulation.

Update method

In each time step, the time, wind speed, wind angle, and position of the underlying boat object are updated and stored in variables t, $V_w(t)$, $\psi_w(t)$, and (x_{boat}, z_{boat}). The wind parameters are taken directly from (3.13) and are used to calculate the x- and z-components of the wind in the global frame, according to (3.20). Note that $V_w(t)$ need to be converted to $[m/s]$ from $[kn]$.

$$\begin{aligned} v_x &= V_w(t) \cdot \cos\big(\psi_w(t)\big) \\ v_z &= V_w(t) \cdot \sin\big(\psi_w(t)\big) \end{aligned} \tag{3.20}$$

These components are used to generate an updated *height field* containing the amplitudes for all the Gerstner waves, and adding them together using FFT. See sections 2.4.7 and 2.4.8 for more detail.

By remembering that the wave amplitudes are governed by (2.28), the base for the height field becomes (3.21). The exponent is set to $m = 2$, V_w is inserted, and the spreading function is expanded to include the wind components v_x and v_z. Note that $\xi_r = \boldsymbol{\xi_r}(\boldsymbol{k})$ and $\xi_i = \boldsymbol{\xi_i}(\boldsymbol{k})$ are the generated random numbers discussed under "Global variables".

$$\tilde{h}^*(\mathbf{k}, t) = (\xi_r + i\xi_i)\, A\, \frac{\exp\left(\frac{-g^2}{2\,V_w^4}/k^2\right)}{k^2} \left| \frac{k_x\, v_x + k_z\, v_z}{k\, V_w} \right|^2 \mathrm{e}^{i\sqrt{g\,k}\,t} \tag{3.21}$$

To complete the field, the current position of the water plane must be taken into account by adding the factor $\mathrm{e}^{i\boldsymbol{k}\cdot\boldsymbol{x_0}}$, from (2.28). Since the only region of interest is around the boat, the current position is set to the position of the boat, meaning $\boldsymbol{x_0} = (x_{boat}, z_{boat})$. This modification is expressed in (3.22).

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}^*(\mathbf{k}, t)\, \mathrm{e}^{i(k_x\, x_{boat} + k_z\, z_{boat})} \tag{3.22}$$

To simplify the calculations, the x- and z-parts of $\tilde{h}$ is taken already in this step, on the same principle as in (2.18). Equation (3.23) shows how this is implemented.

$$\begin{aligned} \tilde{x}(\mathbf{k}, t) &= \tilde{h}(\mathbf{k}, t)\, \tfrac{k_x}{k} \\ \tilde{z}(\mathbf{k}, t) &= \tilde{h}(\mathbf{k}, t)\, \tfrac{k_z}{k} \end{aligned} \tag{3.23}$$

For the sake of completeness, the components and magnitude of the wave number vector, from (2.29), are included here. Since j_x and j_z are the indices used to loop through all the waves, and g is the gravitational constant, all the variables in (3.21)-(3.23) are therefore known and are discussed in this section.

$$\begin{aligned} k_x &= \tfrac{2\pi}{L}\,(j_x - N/2) \\ k_z &= \tfrac{2\pi}{L}\,(j_z - N/2) \\ k &= \sqrt{k_x^2 + k_z^2} \end{aligned} \tag{3.24}$$

Both the bit-reverse and the FFT are implemented fairly straight forward, as described in the theory under section 2.4.8. The only change is that instead of performing these for arrays of $\tilde{h}$, $\tilde{x}$, and $\tilde{z}$ separately, all three arrays are inserted at the same time.

The layout is constructed to allow for as much parallel computations as possible. When updating the height field, rows are generated in parallel and stored in local arrays, $\boldsymbol{h}$, $\boldsymbol{x}$, and $\boldsymbol{z}$. In the same parallel for-loop, these arrays are bit-reversed and then put through the FFT algorithm. The resulting arrays are stored in the global matrices $\boldsymbol{H}$, $\boldsymbol{X}$, and $\boldsymbol{Z}$ before exiting the loop. So far, only elements within each row are added together, meaning that another almost identical parallel for-loop must be implemented to add elements from different rows. This time, each column is extracted from $\boldsymbol{H}$, $\boldsymbol{X}$, and $\boldsymbol{Z}$ into the local arrays, which are also bit-reversed and put through the FFT algorithm. The resulting elements, now containing sums of all different waves, are finally stored back into the global matrices.

Before inserting the elements into the vertices of the meshes, they are first multiplied by either 1 or -1 depending on the $(-1)^{i_x+i_z}$ discussed in (2.30) and (2.31). They are then added with their initial positions in the grid, based on L/N as in equation (3.19), and the current position of the boat.

$$vertex(i_v) = \begin{bmatrix} \boldsymbol{X}(i_x, i_z) \cdot (-1)^{i_x+i_z} + \frac{L}{N} i_x + x_{boat} \\ \boldsymbol{H}(i_x, i_z) \cdot (-1)^{i_x+i_z} \\ \boldsymbol{Z}(i_x, i_z) \cdot (-1)^{i_x+i_z} + \frac{L}{N} i_z + y_{boat} \end{bmatrix} \quad , \text{ with } \quad i_v = N\, i_x + i_z \tag{3.25}$$

This entire procedure is performed for all the different planes, and the resulting waves are showcased in figure 3.9.

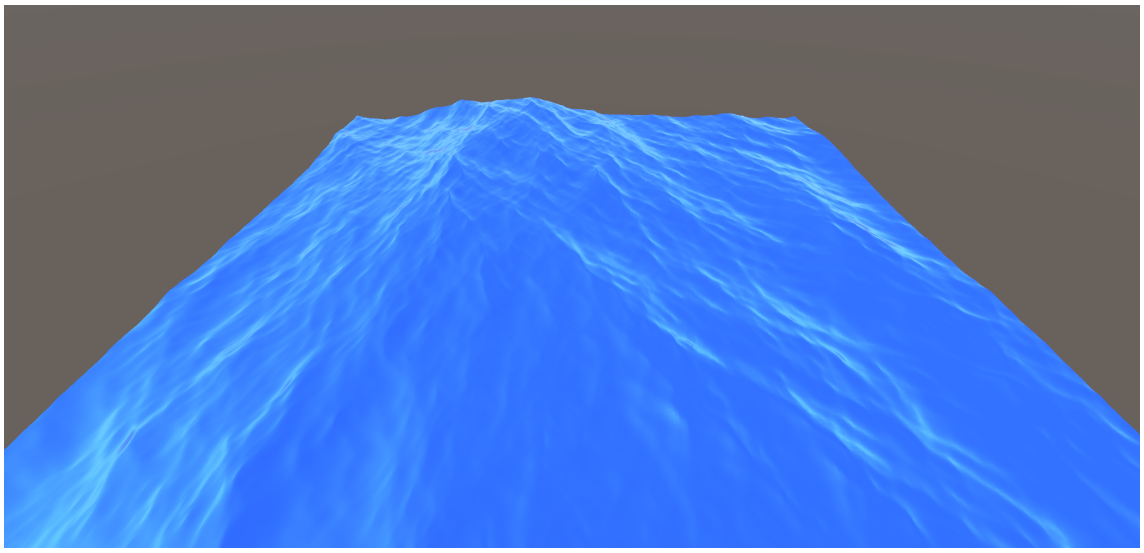


Figure 3.9: The resulting waves when 256×256 Gerstner waves are added together using Fast Fourier Transformations and parallel computations in Unity. The waves are subjected to winds by the Harris Spectrum and the Positive Cosine Squared spreading function.

3.4.5 Waypoint generation

The waypoints are generated randomly within certain constrains, except for the first two points which are always in the same position; straight in front of the boat. The argument for this, when generating a path, is that it should always originate from the boat's current position. To generate a new waypoint $\vec{\boldsymbol{w}}_{n+1}$, the vector connecting the previous two points $\vec{\boldsymbol{V}}$ is used.

$$\vec{\boldsymbol{V}} = \vec{\boldsymbol{w}}_{n-1} - \vec{\boldsymbol{w}}_n \tag{3.26}$$

Moreover, a random angle q is generated along with a distance d. The vector $\vec{\boldsymbol{V}}$ is rotated q degrees around its y-axis using a rotation matrix, and then normalized.

$$\hat{V} = \frac{\vec{\boldsymbol{V_1}}}{||\vec{\boldsymbol{V_1}}||}, \text{ where } \qquad \vec{\boldsymbol{V_1}} = \begin{pmatrix} Cos(q) & 0 & Sin(q) \\ 0 & 1 & 0 \\ -Sin(q) & 0 & Cos(q) \end{pmatrix} \cdot \vec{\boldsymbol{V}} \tag{3.27}$$

Figure 3.10 depicts the new waypoint $\vec{\boldsymbol{w}}_{n+1}$. Which is the sum of the last waypoint $\vec{\boldsymbol{w}}_n$ and the normalized vector $\hat{V}$ scaled by d.

$$\vec{\boldsymbol{w}}_{n+1} = \vec{\boldsymbol{w}}_n + d \cdot \hat{V} \tag{3.28}$$

Both q and d are variables defined within fixed limits determined by the authors. The limits for q is called *angle span* and can be modified to change the difficulty of the path. A specific difficulty is achieved through manually controlling the boat and observing how challenging the track is.

$$-30 \leq q \leq 30°, \qquad 15 \leq d \leq 30\text{m} \tag{3.29}$$

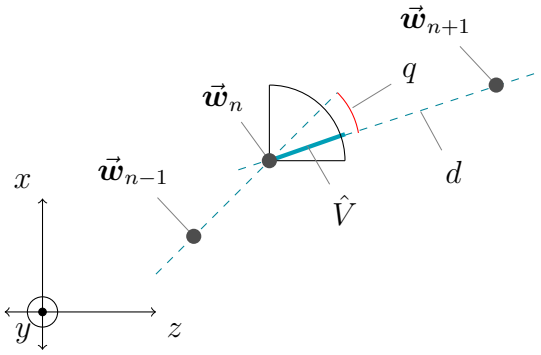


Figure 3.10: Paths are constructed through generation of new waypoints, $\vec{w}$. These spawn at an angle q, and a distance d from the previous point. Both q and d are random, within certain limits. $\hat{V}$ is the unit vector describing the angle q.

To determine if the boat has passed the waypoint or not, the *Progress* variable in equation (3.2) is used. If it is greater than 1, the boat has passed the current waypoint.

3.5 Training via ML-Agents

The training of the NN is done in Unity using the Python package "ML-Agents" [42]. The training is started by writing the following command in a command prompt and pressing *play* in Unity:

mlagents-learn **train.config** *--run-id* **ID** *--force/resume --time-scale* **x** *--capture-frame-rate* 0 *--base-port* **PORT** *--env* **Build/Program.exe**

The network used for the AP is a standard feed-forward NN, the parameters and size are determined by Unity as part of the training process. There is room for experimentation with other types of networks, like *Recurrent Neural Network* (RNN), as well as the use of different layers, such as pooling and normalization. The training algorithm uses *Proximal Policy Optimization* (PPO), ML-Agent also provides the possibility to train using *Soft Actor Critic* (SAC) or MA-POCA. This is specified in the **train.config** file, as seen in Appendix B. Additionally, **ID** defines a unique name of the training results, *force/resume* thereby indicates if a certain id should be resumed or overwritten. The **PORT** is a numerical number, indicating the port used by the python simulation and the coincident Unity instance. Lastly, the *env* option argument is the name, including the path, to the Unity executable; if no argument is passed here, the simulation runs in the Unity editor.

3.5.1 Training parameters

Within the ML-Agents package, multiple training parameters can be changed, far more than what this report will cover. That is why this thesis refers to the documentation for ML-Agents for more information [42], the specific training parameters used during the final training of the AP are listed in Appendix B.

Some of the more important parameters for the training outcome are *hidden_units* and *num_layers*, these variables define the configuration of the network, namely the number of layers and how many neurons those layers should include. The input and output layers are regulated using the number of observations and outputs (continuous and discrete) respectively, as seen in Figure 3.11. A discrete output can take integer values from 0 to $N-1$, where N is the size of the array or branch specified. A continuous output on the other hand can take float values from the range of -1 to 1. The size for the input/output -layer is specified in something called *Behavior Parameters* in Unity.

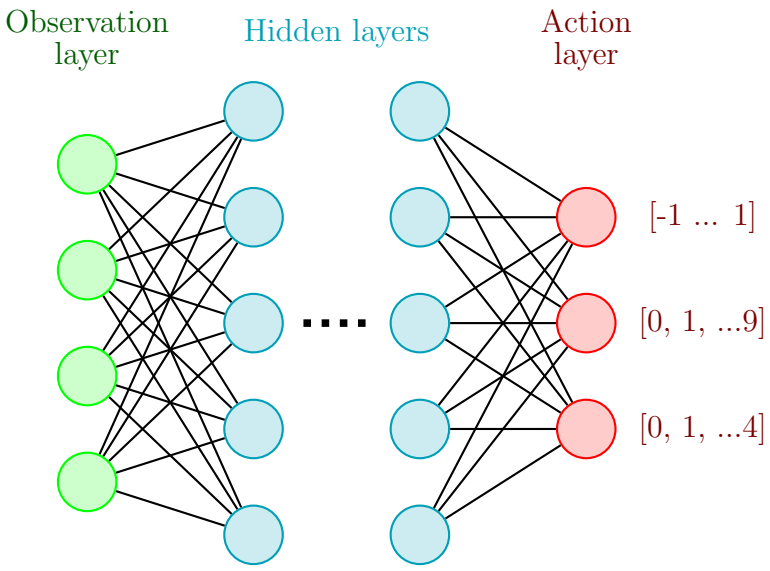


Figure 3.11: An example neural network control allocator, it illustrate the variable functionality of the hidden layers, further, the use of both discrete and continuous actions.

There is also the case of environmental parameters, that can alter values depending on the progress of the training. An example is the angle a new waypoint can spawn at described in section 3.4.5. Another is the maximum allowed wind speed, this will more or less control the magnitude of the disturbances applied to the boat since the wind is what generates the waves (as discussed in section 2.4).

There is also the ability to increase the timescale by running the ML-Agent training command with the option flag *–time-scale=X* where X is how the scale should be changed. *timeScale=2* makes the simulation run twice as fast and respectively *0.5* makes the simulation run twice as slow. This together with running multiple instances of the simulation are both techniques to make the agent take actions more frequent. Finally, *capture-frame-rate=0* is used to sync the call frequency of the Update and FixedUpdate methods.

3.5.2 Observations

Figure 3.2 depicts the agent having a variety of observations. Namely, the PID output, and the capacity of each driveline, rotation and RPM. Each input to the NN is normalized, as recommended by the ML-Agents documentation. The inputs are illustrated as a vector, where the subscript $1, 2$ indicate the number of each driveline.

$$\{\frac{u}{30^\circ}, \frac{RPM_1}{3500rpm}, \frac{Rotation_1}{30^\circ}, \frac{RPM_2}{3500rpm}, \frac{Rotation_2}{30^\circ}\}$$

Many iterations of the input set made up the final set of variables used. After trying to steer the boat manually, while only looking at the observations, it became evident that the variables seen in the input are the important ones. Other input sets are touched upon in the discussion section of the report. The aim of the input set is twofold, to keep the NN updated with the current state of the actuators, and, given the control signal, to let it know how to steer accordingly.

3.5.3 Action space

The agent only uses discrete actions, both for the Azimuth and the bow thruster. Moreover, the action space is very small, and only contains actions to increase or decrease the RPM/Rotation. The action space for each driveline looks like this, $A_n = \{0, 1, 2\}$. Where 0 is the default value and does not change any of the current parameters for the driveline, 1 increases and 2 decreases the RPM/rotation.

3.5.4 Reward function

As briefly discussed in section 2.1, the reward function determines if an action is good or bad, giving a positive or negative reward. This process is performed over many iterations so that patterns are identified and the NN can be adjusted accordingly, to maximize the accumulated reward. If the agent is getting consistently high rewards at the end of a training session, the policy might have converged into a local or global maximum of optimal behavior. A reward function is therefor tested multiple times to hopefully explore different local maxima, and avoid getting stuck in a single one. The final reward function used in this thesis is expressed in (3.30).

$$R(t) = R_0 + \left(1 - \tanh\left(\frac{CTE}{5}\right) + 1.5\,\frac{a_{head} - a_{headMin}}{1 - a_{headMin}} + A_a + 10\,wp_{pass}\right) A_{RF} \quad (3.30)$$

Here, $R(t)$ is the reward given to the agent in any timestep where an action is taken. It should be confined to $|R(t)| < 1$ for optimal learning, according to Juliani, Berges, Vckay, *et al.* [42]. A reward is connected to, and assumed to be influenced by, a set number of previous actions. In this case, it is chosen to be 64, see the previous reference for more detail. R_0 is a small negative reward with two parts, shown in (3.31). The first one being a constant to punish standing still or taking no relevant action. This is meant as a motivation to learn quicker. The second is a small punishment to purely sideways movements, which is proportional to the speed

of the boat in its local z_0-direction. The variable v is the lateral speed of the boat, see figure 2.3.

$$R_0 = -\frac{1}{2000} - \frac{|v|}{200} \tag{3.31}$$

The first two terms within the parenthesis in (3.30) maps the CTE into a gradually changing function between 1 and 0. It becomes 1 when CTE is small, and 0 when CTE is high. The next term depends on a_{head} and $a_{headMin}$, where a will be defined as the dot product between two unit vectors. This means a will be equal to cosine of the angle between two unit vectors, $\alpha_{1,2}$, as seen in (3.32).

$$a = vector_{1x} \cdot vector_{2x} + vector_{1z} \cdot vector_{2z} = \cos(\alpha_{1,2}) \tag{3.32}$$

If the two vectors align, $a = 1$, and if they are pointing in the opposite direction, $a = -1$. The term a_{head} is the cosine between the heading of the boat and an interpolation between the two next way-points, representing the point where the boat should be heading. $a_{headMin}$ is the breaking point after which the reward should turn negative. This entire term is multiplied by 1.5, other a-terms are combined into A_a, see (3.33).

$$A_a = a_{vel} + a_{path} + a_{velHead} + a_{rudder} + a_{rpm} \tag{3.33}$$

The first part in A_a, a_{vel}, is very similar to a_{head}. They use same interpolated heading-point, but the boat's velocity-vector is used instead of its heading-vector. This means the velocity of the boat should be pointing towards the heading-point. Next, a_{path} and $a_{velHead}$ represents how much the boat heading aligns with the path, and with the boat's velocity respectively. Lastly, a_{rudder} and a_{rpm} give rewards based on how parallel the rudders are, and how similar the rpm of the propellers are. These last two have slightly different equations allowing them to give maximum reward within a certain span, to not punish small variations.

$$a_{rudder} = \min\left(1,\ c - (c+1) \cdot \left(\frac{rudderDiff}{2 \cdot maxRudderAngle}\right)^3\right) \tag{3.34}$$

The expression for a_{rudder} can be seen in (3.34), the exact same concept is also used for a_{rpm}. Here, c is a constant, *rudderDiff* is the difference between the angles of the rudders, and *maxRudderAngle* is the maximum allowed angle for one rudder. When the rudders are parallel, the whole expression becomes either 1 or c depending on if $c > 1$ or not. By choosing different c, it is therefor possible to tune how much the rudders are allowed to differ before the agent gets punished. The maximum difference occurs when *rudderDiff*$= 2 \cdot$*maxRudderAngle*, which turns the entire expression equal to -1.

The last term inside the parenthesis in (3.30) depends on wp_{pass}. This is a Boolean that only becomes 1 when the boat passes a waypoint. It is multiplied with a relatively large number, 10, to make sure the agent understands that it is beneficial to make progress along the path. The last variable left in the equation is A_{RF}. This is a factor that scales the reward based on the current speed of the boat, a few of the different a-terms from before, and a user-provided constant, see (3.35).

$$A_{RF} = \min\left(1,\ \frac{|v_{boat}|}{v_{boatMax}} \cdot \frac{a_{head} - a_{headMin}}{1 - a_{headMin}} \cdot a_{path} \cdot \max\left(0.15, a_{rudder}\right)\right) K_{RF} \tag{3.35}$$

Since the previously accumulated reward can be negative, this factor must be strictly positive as to not change the sign of the previous reward. For the speed and a_{rudder}, this is done by taking the absolute value and with a hard cap. For a_{head} and a_{path}, conditions must be implemented to keep these from becoming negative, but that will be included in the "End Conditions", explained in the following section, which is 3.5.5. The constant, K_{RF}, is chosen such that the accumulated reward is roughly equal to 1 for each waypoint passed. This happens at close to $K_{RF} = 1/700$. Note that all factors inside the parenthesis can take values between [0,1], but since the wind can push the boat faster than $v_{boatMax}$, this need to be capped at values ≤ 1.

Finally, there is on last additional reward that can affect the final score, the reward for completing the entire path, R_{clear}. To minimize the time spent on the track, the reward needs to be calculated in a way that give higher rewards when the final waypoint is reached earlier. This is achieved by dividing the maximum allowed number of steps, *MaxStep*, with the number of steps needed to complete the path, *StepCount*. Therefor, the reward would be equal to 1 when $StepCount = MaxStep$, and equal to 2 when $StepCount = MaxStep/2$. The resulting term can be seen in equation 3.36.

$$R_{clear} = \begin{cases} 0 & \text{, if the end is not reached} \\ \frac{MaxStep}{StepCount} & \text{, if the end is reached} \end{cases} \tag{3.36}$$

3.5.5 End conditions

There are multiple conditions that can terminate a training episode, in the current training configuration there are five, listed in Table 3.5.5. These exist to prematurely finish the current episode, where the agent has executed a unwanted behavior.

Table 3.1: End conditions used during training.

Condition	Reward	Comment
$n > 40$	Maxstep/StepCount	Path finished
$boat_x < -40$	-0.005	Reversed in the beginning
$boat_y < -10$	0	Sunk
$\|CTE\| > 10$	-0.005	Drow to far away from the path
$Dot(heading, \vec{\boldsymbol{d}}) < 0.7$	-0.005	Not aiming at the waypoint
$\|Dot(heading, \vec{\boldsymbol{a}})\| < 0.56$	-0.005	Not traveling in the same direction as the path

Figure 3.12, depicts what these conditions are, related to the boat and the path.

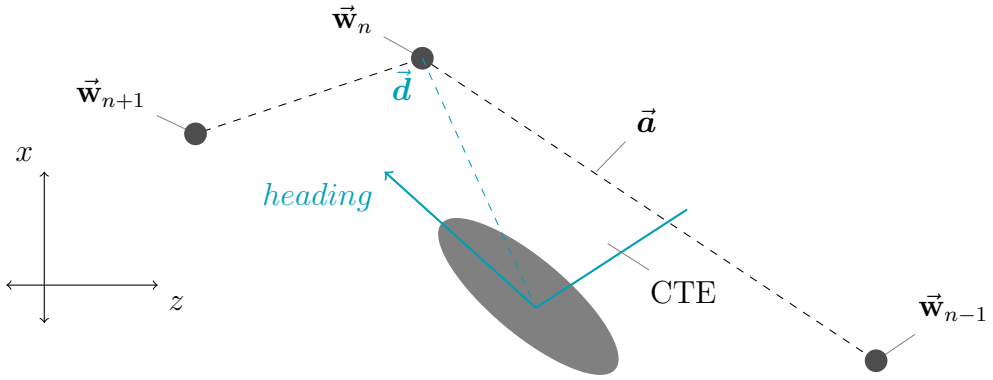


Figure 3.12: End condition visualization.

When an end condition is met the boat resets, meaning, the boat, along with related parts, such as the PID regulator, are reset to their original values. First, all the environmental forces for the boat are turned off. Using the Unity API, the *kinematic* setting is turned on for the boat. This implies that neither forces, collisions nor joints can affect the boat. Only then, can the boat be moved back to its original position and rotation safely. The velocity for the boat is set to a forward speed, more specifically, the RPM for each driveline is set to 2000. Moreover, the PID, described in section 3.3, is reset. Namely, the stored values for the integral and derivate parts.

Various parts explained in section 3.4 are also reset, namely, the under/above-water meshes, the relative wind speed, and the path.

Chapter 4

Results

This chapter presents the results from when the AP is driving the boat on a set of test tracks, in different wind conditions. Moreover, a structural analysis on the reliability of the AP is presented.

In the following sections, the level of disturbance is separated between *High*, *Medium* and *Low* disturbances. From here on, High disturbance is referred to a simulation done with $V_w(t) \leq 10$ kn and $V_w(0) = 10$ kn. Similarly, Medium is $V_w(t) \leq 10$ kn and $V_w(0) = 5$ kn, and Low disturbance is a simulation done where $V_w(t) \leq 1$ kn with $V_w(0) = 1$ kn.

4.1 Path driving

The most important aspect of the AP developed in this project is its ability to follow a pre-determined path. A series of tests are therefor designed and conducted to measure its performance for different types of paths and environmental conditions. Three categories of tests are set up, each with their own test-tracks. The first one uses a fairly straight path to test basic control and small course adjustments. The second requires the AP to perform slow to medium turns, and the last one checks its ability to handle fast and sharp turns. Each of these are performed with both high and low disturbances, from the wind and waves. Keep in mind that these results are in comparison to human drivers, also called heuristic. Later, in section 4.2, the AP uses the same tracks when driving with losses of capabilities.

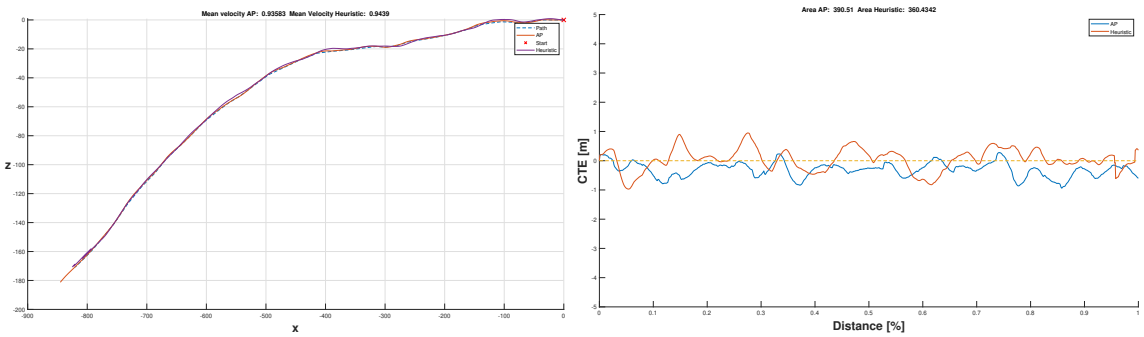
Based on the principles in section 3.4.5, the different tracks are built by using the same logic. Their angle spans are adjusted according to the level of difficulty each test are supposed to achieve. Key-metrics includes plots of trajectories, plots of CTE values, mean velocities, and the mean and max of their CTE values.

4.1.1 Straight line

The fairly straight track is made out of waypoints with angle spans of 12 degrees. Each waypoint could therefor at most have angles of $\pm 6^\circ$ relative to each other. The low disturbance test should only be seen as a proof of concept for the AP. There is however a real use-case for keeping a course though harsh environmental conditions, the high disturbance test is therefor the first relevant test.

Low disturbance

The first test is a low disturbance run on the fairly straight track. As can be seen from figures 4.1a and 4.1b, its clear that the AP did follow the path and that their CTEs remained stable. They kept within 1 from the center throughout the run, and ended up at a mean of 0.328 m for the AP and 0.298 m for heuristic. Mean velocities at 93.58% and 94.39% of the maximum speed are also maintained for the AP and heuristic respectively.

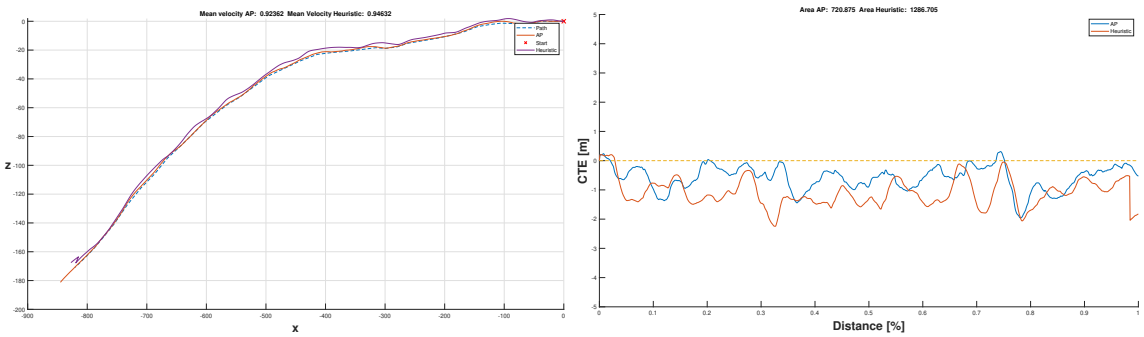


(a) Trajectories **(b)** CTE

Figure 4.1: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 6° turns, subjected to small winds and waves.

High disturbance

In this test, 4.2a and 4.2b shows that the AP once again did follow the path and maintained stable CTE-values, within 2 m. The heuristic stayed within 2.3 m. Their mean velocities and CTE values are 92.36% of max speed and 0.594 m for the AP, and 94.63% and 1.099 m for the heuristic.



(a) Trajectory **(b)** CTE

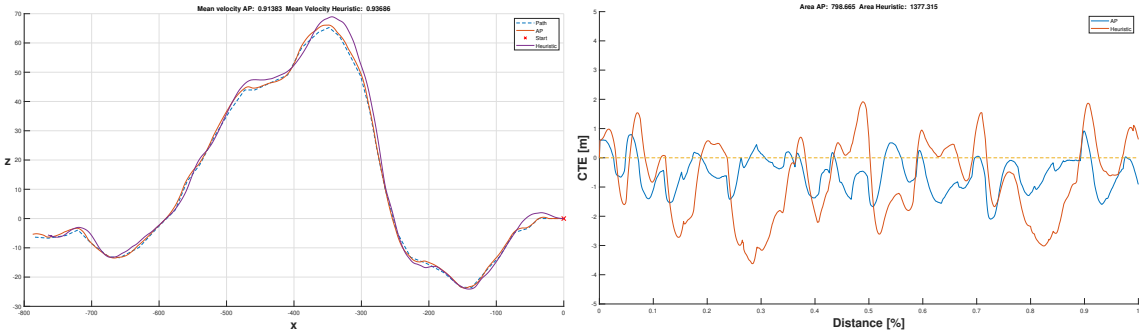
Figure 4.2: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 6° turns, subjected to large winds and waves.

4.1.2 Slow turn

The track with slow to medium turns is made out of waypoints with angle spans of 38 degrees, meaning that the relative angles between each point could therefor at most be ± 19 degrees.

Low disturbance

Figures 4.3a and 4.3b shows that the AP still followed the path. Their mean velocities and CTE values are 91.38% of max speed and 0.661 m for the AP, and 93.68% and 1.229 m for the heuristic. Maximum CTEs are 2.10 and 3.62 m.

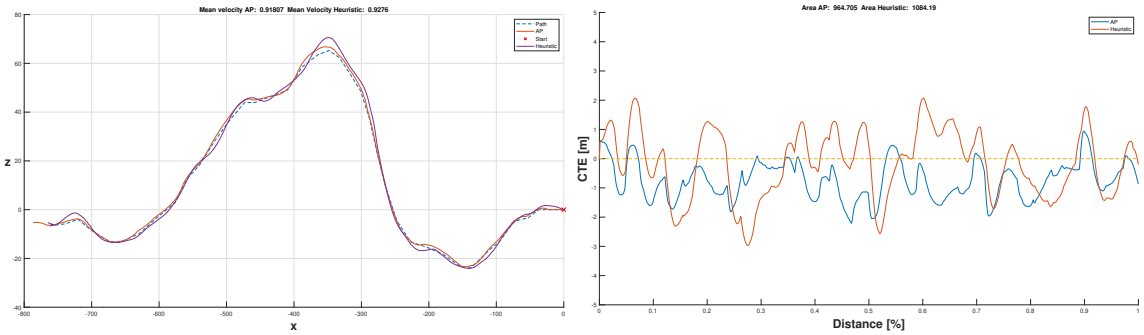


(a) Trajectory **(b)** CTE

Figure 4.3: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 19° turns, subjected to small winds and waves.

High disturbance

According to 4.4a and 4.4b, the AP did follow the path. The mean velocities and CTE values are 91.80% of max speed and 0.832 m for the AP, and 92.76% and 0.959 m for the heuristic. Maximum CTEs are 2.21 and 2.97 m.



(a) Trajectory **(b)** CTE

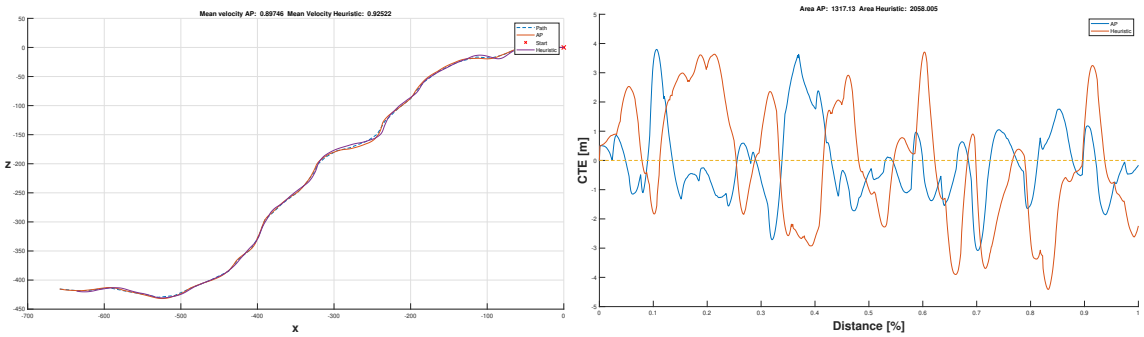
Figure 4.4: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 19° turns, subjected to large winds and waves.

4.1.3 Fast turn

The final track, with fast turns, is made out of waypoints with angle spans of 52 degrees, meaning that the relative angles between each point, at most, could be up to ± 26 degrees.

Low disturbance

Figures 4.5a and 4.5b shows that the AP followed this path as well. The mean velocities and CTE values are 89.74% of max speed and 1.007 m for the AP, and 92.52% and 1.766 m for the heuristic. Their maximum CTEs are 3.80 and 4.41 m.

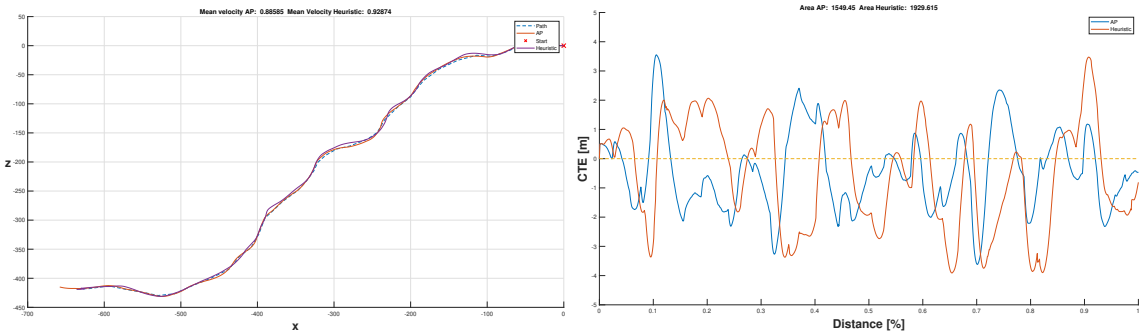


(a) Trajectory **(b)** CTE

Figure 4.5: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 26° turns, subjected to small winds and waves.

High disturbance

Finally, 4.5a and 4.5b shows that the AP still managed to follow the path. The mean velocities and CTE values are 88.58% of max speed and 1.157 m for the AP, and 92.87% and 1.594 m for the heuristic. The maximum CTEs are 3.62 and 3.91 m.



(a) Trajectory **(b)** CTE

Figure 4.6: Comparison between trajectories and cross track errors of a boat driven by the autopilot, or heuristically along a path with 26° turns, subjected to large winds and waves.

4.1.4 Summary of path driving results

All the results from the different tests in section 4.1 are summarized in table 4.1. The data is sorted based on the driver, and then on the track. Velocity % means speed relative to its max speed, and mean CTE is the mean absolute values from the graphs in figures 4.1 - 4.5. Max CTE are the maximum values from the same graphs.

Table 4.1: Summary of the tests where the AP, and the heuristic, are driven on three different tracks, with turning-angles between 6 - 26°, in different environmental conditions.

Driver	Angle °	Disturbance	Velocity %	Mean |CTE| m	Max |CTE| m
Autopilot	6	low	93.58	0.328	0.94
	6	high	92.36	0.594	1.96
	19	low	91.38	0.661	2.10
	19	high	91.80	0.832	2.21
	26	low	89.74	1.007	3.80
	26	high	88.58	1.157	3.62
Heuristic	6	low	94.39	0.298	0.97
	6	high	94.63	1.099	2.25
	19	low	93.68	1.229	3.62
	19	high	92.76	0.959	2.97
	26	low	92.52	1.766	4.41
	26	high	92.87	1.594	3.91

4.2 Loss of Capabilities

To test the adaptability of the AP, it is put through scenarios where its capabilities are altered to simulate problems with its actuators. The altered properties are maximum rpm of the drivelines and the maximum rotation angles of the rudders. These are set to either 50% or 0% for one of the metrics, or 50% for both of them, and added randomly to one of the two drivelines with a 50% probability. This turned into 5 test-cases:

$L1$: 50% max velocity on one driveline
$L2$: 0% max velocity on one driveline
$L3$: 50% max rudder-angle on one driveline
$L4$: 0% max rudder-angle on one driveline
$L5$: 50% max velocity and rudder-angle on any driveline

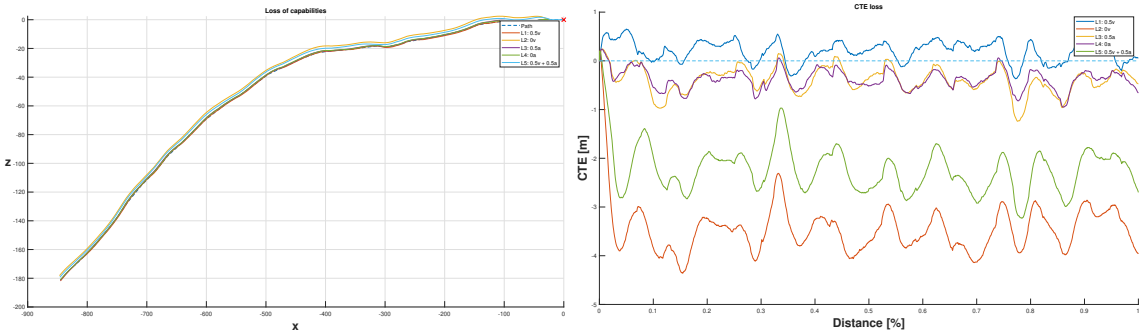
Maximum velocity and rudder-angle are denoted by v and a, and the tracks used for these tests are the same as in the previous tests; described in section 4.1.

Track 1

For the first track, figure 4.7 shows that the AP handled all losses and got CTE values mostly well within 1 m for L1, L3, and L4, while L5 and L2 are almost kept within 3.2 and 4.3 m respectively. Except for L1, all the CTE values are almost constantly negative. Their mean velocities, mean CTEs, and maximum CTEs can be seen in table 4.2.

Table 4.2: Result from AP, driven on a track with turning-angles of 6°, influenced by different losses of capabilities.

Property\Driver	L1	L2	L3	L4	L5
Velocity %	61.14	48.37	94.51	94.38	60.20
Mean CTE m	0.247	3.451	0.400	0.384	2.229
Max CTE m	0.65	4.36	1.24	0.95	3.23



(a) Trajectory **(b)** CTE

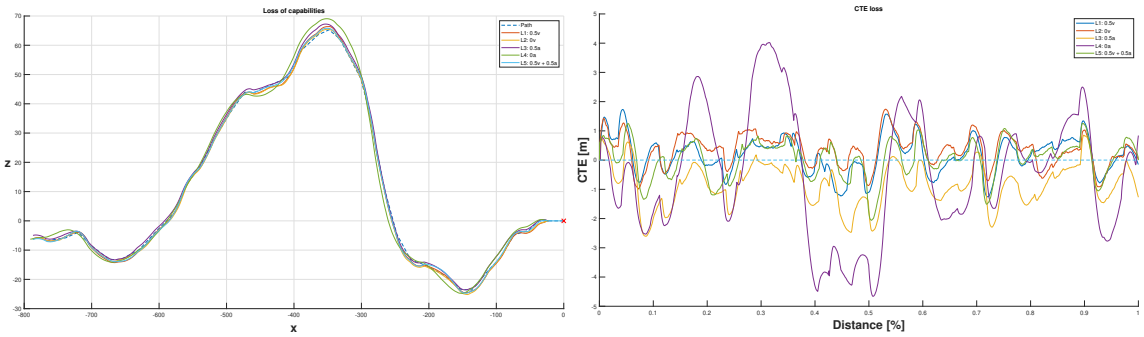
Figure 4.7: Trajectories and cross track errors of a boat driven by the autopilot along a path with 6° turns, subjected to medium winds and waves, and varying degrees of loss of capabilities. In the legends, a stands for rudder-angle, and v for rpm (velocity) of the thruster, 0.5 means 50% capacity.

Track 2

For the second track, figure 4.8 shows that the AP handled all losses and got CTE values mostly well within 2-3 m for the majority of the cases. However, L4 went up to almost 5 m, but still managed to keep below this number. Their mean velocities, mean CTEs, and maximum CTEs can be seen in table 4.3.

Table 4.3: Result from AP, driven on a track with turning-angles of 19°, influenced by different losses of capabilities.

Property\Driver	L1	L2	L3	L4	L5
Velocity %	59.62	48.16	92.11	92.37	58.39
Mean CTE m	0.527	0.554	0.905	1.753	0.51893
Max CTE m	1.73	1.74	2.61	4.66	2.06



(a) Trajectory **(b)** CTE

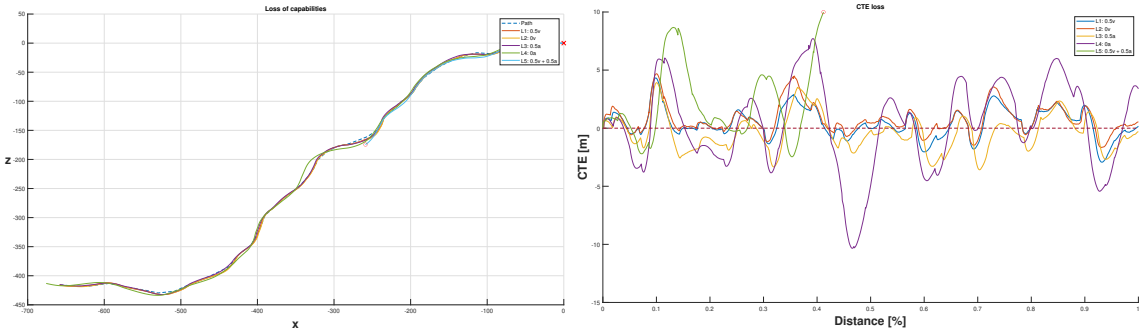
Figure 4.8: Trajectories and cross track errors of a boat driven by the autopilot along a path with 19° turns, subjected to medium winds and waves, and varying degrees of loss of capabilities. In the legends, a stands for rudder-angle, and v for rpm (velocity) of the thruster, 0.5 means 50% capacity.

Track 3

For the third track, figure 4.9 shows that the AP handled most of the cases, except for L4 and L5, where the CTE values got too high, in regards of the end conditions. Both of these reached 10 m, but the others were keeping their CTEs below 5 m. Their mean velocities, mean CTEs, and maximum CTEs are seen in table 4.4.

Table 4.4: Result from AP, driven on a track with turning-angles of 26°, influenced by different losses of capabilities.

Property\Driver	L1	L2	L3	L4	L5
Velocity %	59.62	48.16	92.11	92.37	58.39
Mean CTE m	1.007	1.049	1.299	3.040	2.717
Max CTE m	4.32	4.68	3.92	10.34	9.99



(a) Trajectory **(b)** CTE

Figure 4.9: Trajectories and cross track errors of a boat driven by the autopilot along a path with 26° turns, subjected to medium winds and waves, and varying degrees of loss of capabilities. In the legends, a stands for rudder-angle, and v for rpm (velocity) of the thruster, 0.5 means 50% capacity.

4.2.1 Summary of capability-loss results

The results from the capability-loss tests are summarized in table 4.5. Only two of the test cases failed, marked red, due to CTE values being over 10 m.

Table 4.5: Summary of the tests where the AP is driven on three different tracks, with turning-angles between 6 - 26°, influenced by different losses of capabilities.

Driver	Angle °	Disturbance	Velocity %	Mean |CTE| m	Max |CTE| m
L1: 0.5v	6	medium	61.14	0.247	0.65
	19	medium	59.62	0.527	1.73
	26	medium	58.35	1.007	4.32
L2: 0.0v	6	medium	48.37	3.451	4.36
	19	medium	48.16	0.554	1.74
	26	medium	47.08	1.049	4.68
L3: 0.5a	6	medium	94.51	0.400	1.24
	19	medium	92.11	0.905	2.61
	26	medium	90.12	1.299	3.92
L4: 0.0a	6	medium	94.38	0.384	0.95
	19	medium	92.37	1.753	4.66
	26	medium	88.83	3.040	10.34
L5: 0.5v+0.5a	6	medium	60.20	2.229	3.23
	19	medium	58.39	0.519	2.06
	26	medium	55.81	2.717	9.99

4.3 Reliability

The result related to reliability of the AP consists of a sample of 12471 random path runs, with varying disturbance magnitude, and with angle spans of 60°. Meaning that waypoints has maximum relative angles of 30° between them. Out of these almost 12500 runs, 12425 finished the path, and 46 failed, from CTE being too high. This gives a success rate of over 99.63%. Every end condition explained in Section 3.5.5 are used when performing the runs.

Chapter 5

Evaluation

In this chapter, the results presented in the previous chapter are evaluated, for path driving, losses of capabilities, and reliability. The AP is also receiving an evaluation score, and is assigned a tier, based on a newly formed standard created in this project.

5.1 Driving better than expected

It is possible to draw a few conclusions about the general capabilities of the AP from table 4.1. The first thing to note is that all of the tracks are completed without any issues in all test-cases. This was a bit surprising, since many of the previous versions of the AP did not handle the more difficult cases at all. As expected, the mean CTE increases and the velocity decreases when angles and disturbances gets higher. However, the heuristic results shows that the mean CTE decreases when disturbances are high for the last two tracks. This could be due to a low sample size, where the driver might have been lucky with wind orientations; or other random factors that could have been solved with more rigorous testing.

From table 4.1, it can be seen that the AP kept a lower speed on average compared to heuristic runs. It also decreased the speed further when faced with more difficult turns, which might be why it outperformed human drivers, in terms of CTE values, in these situations. The improvements were not small either, ranging between 13.2 - 46.2% on the last two tracks. On the straight track, however, it performed equally well during low disturbances, only a difference of 0.03 m, and significantly better during high, 46% improvement. This was also a bit surprising since previous versions easily beat the heuristic in both these tests, but were much less robust on the more difficult tracks. Despite performing slightly worse on track 1, it still performed very well, with an mean CTE of just 0.328 m, keeping within 1 meter from the center.

Environmental disturbances had less of an impact than expected, only adding between 0.150 - 0.266 m to the APs mean CTE for all the tracks. This might seem like a lot since the value increases by 81% on the first track, but compared to the other values, 0.594 m is still a rather small number. Furthermore, the disturbances mostly affects the mean CTE values rather than the max. This may not be obvious from the graphs, but since the mean values always goes up, while the max sometimes even decreases with higher disturbances, it is the logical conclusion. Small consistent errors are much more preferable than larger spikes. The smaller errors might still be within the "safety margins" while larger ones could lead to crashes or other unwanted situations.

Even though these results are better than expected, it might have been better to put more effort into lowering the CTE at the expense of higher speeds. These tracks with angles of 26° would not occur in reality unless to navigate between shallow or obstructed areas. Therefor, keeping a low CTE should be even more important.

5.2 Managing most capability-losses

From table 4.5 it is clear that L1 and L3 have the best results regarding CTE values overall. They have similar results to without losses, with mean and max values being within 0.29 m and 0.52m respectively. Althogh, L1 gets slightly lower, and L3 slightly higher values. L3 keeps a surprisingly high speed, even gaining up to 1% higher than without losses, while L1 lost over 30%. However, this reduction in speed is, as expected, proportional to the loss of max-speed capacity. Driving at a reduced speed inherently gives higher CTE since it stays longer at the track, but it also gets more time to react. Therefore, L1 must have driven closer to the center, in order to have gotten good CTE values despite the reduced speed.

L1 and L3 are considered the easiest of the tests, and the ones with the highest chance of success. Both performs exceptionally well, they even beat the CTE scores of the heuristic runs in 5 out of the 6 test cases. Where the human driver just barely gets 0.10 m lower mean CTE than L3 on track 1, see tables 4.1 and 4.5. This good performance can also be seen in figures 4.2 - 4.4, where they keep within 1.25 m CTE for track 1, 2.62 m for track 2, and 4.33 m for track 3. The AP can therefor, theoretically, drive with these issues through a tight, fairly straight, canal without danger of hitting the sides.

L2 also completes its runs, and does fairly well on the last two tracks; where its mean and max CTE only increases with less than 0.11 m and 0.88 m in the worst case. On the first track however, it somewhat unexpectedly get the highest recorded mean CTE value of any of the tests, over 3.45 m. That is an increase of over 3.1 m, which corresponds to being over 952% larger than without losses. Figure 4.2 reveals that it slowly and steadily drives to the left side, between 3 - 4 m from the center almost for the entire duration. It therefor seems like at least some rpm is needed from both propellers to manage small adjustments to the course, but not as much for larger changes. Which is also why the max CTE still never gets above 4.36 m. This means it actually performs better on more difficult paths, as confirmed by figures 4.3 and 4.4, where the CTE stays within 2 and 4.7 meters respectively. Note that the speed is still proportional to the loss of RPM.

For L4, the results are mixed. The speeds are almost unchanged for all the tracks, only differing with about 1%. On track 1, the loss does not seem to impact the result much compared to without losses. The mean and max CTE values on track 2 increases by over 1.75 m and 4.66 m, which is worse than most previous results. However, on track 3 the mean and max CTE goes up to over 3.04 m and 10 m, meaning that it even fails. This is not good, indicating a lot of unnecessary movements. Figures 4.3 and 4.4 confirms that L4 indeed crosses back and forth over the center, up to 5 and 10 meters for the last two tracks. This suggests that both rudders must be able to turn for the AP to follow a path through quick turns; but that it handles small changes and can maintain a fairly straight course with only one rudder.

The last test, L5, has the worst and strangest results. It has speeds similar to L1, and shows the same problems on track 1 as L2, but to a lesser extent. On track 3 it even fails after roughly 40% of the path, as can be seen in figure 4.4. However, on track 2, it somehow performs just as well as L1, which is not at all expected. The reasonable thing would be to assume a worse result than both L1 and L3, since L5 is a combination of the two. This is mostly true, but on the second track the lower speed may actually help the AP get CTE values closer to L1. The higher turn-angles may also help against the effect seen by L2 on track 1, but seems to become too much to handle on track 3, with all the losses.

Overall this section shows that the AP is both very adaptable to many situations, and robust to a lot of problems that might occur. It only fails 2 out of 15 tests, and only goes above 5 meters from the center of the path in the same 2 tests; which are hardest cases on the most difficult track.

5.3 Evaluation of reliability

The results from the reliability tests are that it managed to complete 12425 out of 12471 test runs, with randomly generated test-tracks. Each track has 40 way-points spawning at 15-30 meters apart, meaning that the mean path-length is 900 meters. This only considers straight lines, when bends are accounted for, the number probably rises by at least 3-5%. This means, since 12425 runs were completed, that the AP managed to safely drive the boat over 243 km without failing, over 250 km when adding the extra 3%. The reliability tests are performed under medium disturbances and with angle-differences of 30°. These tracks are therefor more challenging than the other tests, to discover the limits of the AP.

One thing to note regarding these tests is that the failures occurred very infrequently during the testing, and remained unchanged for hundreds of runs. This might indicate that a much higher number of runs is needed to get the real statistical success rate, but the current one, at oven 99.63%, could serve as a baseline for future evaluations.

5.4 Creating a new standard

There is no official standard for testing and evaluating APs of the sort developed in this project. The group members therefor decided to create a new standard based on the tests and findings of this thesis.

An evaluation standard for this purpose should use metrics that are not dependent on the duration of the evaluations, number of tests, or lengths of the paths used during testing. Based on the results in tables 4.1 and 4.5, the mean speed, together with mean and max values of the CTE are good choices. They are easy to compare and the values have meanings that are easy to understand from the perspective of a sailor. The tests should be performed at multiple set target speeds, this way, they can be run on multiple boats to see how well an AP performs on them. This might also be useful to test adaptability of an AP or to match different APs to different types of boats.

The standard is structured in five different tiers, where tier 5 is the best and tier 1 is the worst. An additional tier 0 will only be assigned if the absolute minimum requirements are not met, APs in this category will not be considered complete products. The tier is decided by a score, where table 5.1 shows the score needed for each tier.

Table 5.1: Showing the score needed for each tier in the new standard for APs.

Tier 0	Tier 1	Tier 2	Tier 3	Tier 4	Tier 5
0	< 25	< 65	< 85	< 95	$95 <$

Scores are calculated, according to equation (5.1), based on three different test scenarios, tracks with 6, 19, and 26°. v_{mean} is the mean velocity of the boat from the tests an v_{target} is the desired velocity. When these terms are equal, and both CTE_{mean} and CTE_{max} are equal to zero, the resulting score becomes two times the angle. If the same thing happened for all three tracks, the final score would reach its theoretical maximum of $2 \cdot (6 + 19 + 26) \cdot 1.045 = 106.59$. The reason for this equation is that it becomes $99.94 \approx 100$ when all $CTE_{mean} = 0.1$, $CTE_{max} = 0.15$, and all v_{mean} only differs 3 % from the target. This is considered the perfect score.

$$score = 2 \cdot angle \cdot \frac{5 - CTE_{mean}}{5} \cdot \frac{10 - CTE_{max}}{10} \cdot \left(1.045 - \left|1 - \frac{v_{mean}}{v_{target}}\right|\right) \tag{5.1}$$

This is almost impossible, and most likely, the score will not reach higher than 100. To keep the score from becoming negative, it is forced to zero if either of the conditions in (5.2) are met.

$$\begin{aligned} CTE_{mean} &\geq 5 \\ CTE_{max} &\geq 10 \\ \left|1 - \tfrac{v_{mean}}{v_{target}}\right| &\geq 0.5 \end{aligned} \tag{5.2}$$

When this standard is applied to the developed AP, with the values from chapter 4 and v_{target} equal to the max speed of the boat, the AP gets the score of 55.73. This makes it an AP of tier 2 based on the testing so far. Note that an AP can get different scores for different targeted speeds, meaning that it will only be certified within a certain range of speeds.

Chapter 6

Discussion

In this chapter, a discussion about the project as a whole is held; this includes a review of data, safety aspects along with our own experience. Moreover, other reward functions that was tried during the course of the project is also discussed.

One thing we made clear quite early in the project was, that despite the results from the AP, we wanted to keep the simulation environment as physically correct as possible; even if it signified worse results. Because of this, the computational performance during training suffered. To combat this, a higher time-scale was used, as described in section 3.5.1. What was found during the training and later confirmed was that the physical background processes did not perform the same when increasing the time-scale. This was due to something called *maximum allowed timestep*, which controls how long FixedUpdate should run in a low frame-rate worst-case scenario. Lowering this value seemed to solve the problem.

Moreover, we tried to keep the input space as small as possible, also only containing natural sensor values. As of now, the only sensor needed is a global navigation satellite system (GNSS). Further, we tried to add wind-related data to the input, such as wind direction and speed; to get the AP to better cope with disturbances. However, without major improvements to the training results.

As mentioned in section 1.2.1, we wanted to test if the control allocator would prove beneficial while driving the boat manually. Initially, this should be tested using a joystick but due to lack of time and hardware, we eventually scrapped this test. We tried to get it working with a USB joystick, commonly used for flight simulators, but unfortunately, the interface between the joystick and Unity was not straightforward. Further, the input space was not suited for joystick-driving, which meant that the mapping of the joystick and the actions the boat should perform became much more deterministic.

6.1 Other reward functions

Many different reward functions were tried before ending up with the current one. The stepping stones along the way are outlined here together with the problems and difficulties they brought with them. Other setups following different types of reward functions are also discussed in the later section.

6.1.1 Getting to the final setup

At first, the only important metric used was CTE, and a small reward for passing waypoints. The problem was the lack of punishment for standing still. It could therefor generate much more reward by standing still, as close to the center of the path as possible, until the maximum allowed amount of time steps had passed. A fix was to scale the reward with the speed, and to add a small negative reward in each step to motivate the AP to keep moving.

New problems occurred, since there was no incentive for the AP to face the boat in the same direction as the path. It often moved sideways or even backwards while trying to maintain some speed and keep the CTE low. Another reward was therefor introduced based on cosine of the angle between the boat heading and the direction of the path. This was somewhat successful, however, it lead to other difficulties. When it did veer of the path, it was now punished when it tried to correct itself, since the new heading would deviate from the angle of the path. To counteract this, an additional term based on the cosine of the angle between the boat heading and the desired heading was implemented. The idea was for these two terms to balance each other to achieve a smooth and robust path-following ability.

Even though the AP got much better, there were still some issues with the boat driving slightly sideways. One last similar heading-term was introduced based on the heading and the velocity vector of boat. This was to motivate the boat to align its heading and velocity in the same direction. The problem with following the path was now solved, but other unwanted behaviors still remained.

The AP had figured out that the easiest ways of controlling the boat was to turn each rudder to its maximum in opposite directions, and to manage the turns by adjusting the rpm of the drivelines. In reality, this would be an inefficient and wasteful way of driving the vessel. Two last terms were therefor implemented to keep rudder-angles and rpm at similar values. To allow some individual movements, there was a cap keeping the reward at max for values within a certain span from one another.

All the included terms in the reward function took values between [-1,1] or [0,1]. To get a tighter reward, all these terms were first added together, then also became individual factors that were all multiplied with the initial sum. As described in section 3.5.4, see (3.35). Here, all the factors needed to be positive to not turn the entire reward negative or positive incorrectly. This is why some factors were left out, capped, and conditioned in the final version of the reward function.

6.1.2 Seemingly promising setups

The final version let the reward be applied in each time step, where the episode was terminated whenever an end condition was met. There was also extensive testing on different setups, using the same basic reward functions. One promising setup was to use sparse rewards and keep the same track until it was completed, or the time ran out. When an end condition was met, the boat was reset to a previous waypoint, which it had already passed, but with 5% lower speed. The idea was for the network to learn at what speed it could manage the critical part where it had failed. The sparse rewards meant that rewards were only applied when a new waypoint was passed. This way it would not continuously gain rewards from resetting.

The setup seemed promising, but was prone to getting stuck in edge-cases, unable to move, which had the risk of erasing any progress made during training. Another problem was that it did most training by looping over small difficult snippets of the path and never got to see many of the scenarios it would otherwise have seen. Also, it did not often run the track in one go. It learned that one speed was okay to have when going into a corner, and that it, in an unrealistic manner could get another going out of it. This meant it was not good at handling corners overall, and that this was a failed setup.

Other failed setups included extremely complicated reward functions that performed terribly overall. These were third degree polynomials of all the included terms and factors from the final version of the reward function. The resulting graphs were visualized, with sliders for all inputs, to get positive or negative rewards for certain combinations of input values. The idea was to create a function where only good combinations of inputs should give positive rewards, and where all bad combinations should give negative rewards. Third degree polynomials were chosen to get three distinct areas on the graph, one with high positive rewards, one with roughly zero reward, and one with negative rewards.

All these graphs layered on top of one another meant a lot of user decided coefficients and parameters. These included plenty of limitations within certain ranges to get the correct behavior. The whole setup was difficult to balance, often leading to very high negative rewards. This made the AP actively try to reach an end condition prematurely, to minimize the negative rewards. It might have been possible to pursue this concept further and get it to work properly, but this idea was eventually scrapped. Keeping things fairly simple is, apparently, very important when designing a system like this.

6.2 Significance of PID-parameters

For a long time, the AP did not perform very well. It had troubles keeping up with the path and was always lagging behind. After some extensive empirical parameter searches, it was found that the issues came from two sources: the PID-parameters, and something that will be called *CTE-smoothing*. The CTE-smoothing was a function that set the CTE to a trailing average of a few previous values, this was removed due to making the AP less responsive.

The PID-parameters were a bit more interesting. Since the NN would scale the input to its liking, the only important thing would be to get good ratios between the P, I, and D parameters. Initially, they took values (0.5, 0.6, 0.7), which made the AP follow the path poorly. The first thing tried was to raise I to 1.4 to hopefully give it more stability, but it accumulated large errors which made it veer more to the sides. Instead, D was raised to give it more responsiveness. This worked better the higher it went, up to a breaking point at over 30, where it started becoming uncontrollable. Now I was once again raised to 1.4 to regain some stability, which also worked, pushing the breaking point of D closer to between 40 - 50.

Good and stable behavior could then be found at around (0.5, 1.4, 35). However, this was with a NN that had trained with (0.5, 0.6, 0.7). Therefor the decision was made to train the AP using (0.5, 1.4, 10), and to run it with D set to 15 after the training. The goal was to teach it to become more responsive, but also to give it a boost afterwards. This worked well, and was implemented in the final version of the AP. It might have been possible to train the AP using D set to 15 or 20, and to perhaps go even further afterwards. This was not done due to risks of unstable behavior.

6.3 Sustainable development of the product

In this section, the entirety of the product and the project is analyzed through different aspects.

Data revision

As described earlier in section 3.4.1, the data for the drivelines were not 100% accurate. Moreover, as seen in the Results section, there were a few outliers in some of the tests. All of them occurred during the loss of capabilities tests, where the mean CTE of L1 went to 3.5m on track 1, and L4 and L5 failed on track 3. Other that that, there were no outliers throughout the project. However, as seen in Figure 4.9b and also in Table 4.5, L4 reaches a value (-10.34) that is prohibited by the end condition; $|CTE| > 10$. Somehow the end condition is not activated and the run therefore continues. The CTE quickly decreases to a level within the limits of the end conditions. This phenomenon is the first spotted and is most likely due to the rate at which the agent take an action and checks the end conditions. Moreover, the frequency at which data is being logged is likely to be higher than the rate at which the agent takes action.

Economical

A positive thing about this project was that the purchase of any software was not necessary. Instead, we could utilize student licences, for example, the "Unity student plan". We also believe that CPAC can find further usefulness in the simulation environment we created, and in that sense saved CPAC the cost of buying another. This way, the only hypothetical cost for the thesis was the work hours spent.

Environmental

In regards to the project's impact on the environment, we find many advantages to implementing such a system. Especially in combination with a fuel-efficient path generating software.

Furthermore, computers are capable of finer controlled actions than human drivers. This could be optimized to improve efficiency and therefore fuel consumption. With neural networks, it is also possible to continuously improve these networks over time as they gather more and more data through training and real-world usage. Although, the AP developed in this project was not optimized for fuel efficiency, or even to drive smoothly, and might therefore even worsen the fuel consumption significantly.

On the other hand, the energy usage of the computer performing the calculations is insignificant in comparison to the power needed to propel the ship. However, the concept of autonomous vehicles makes it possible to have more boats in use simultaneously. This can lead to many more boats being produced and utilized, which could have a strong negative impact on the environment.

Safety

There are definitely issues with autonomous ships running around, if there are not enough safety measures in place. Even if it follows the path perfectly it can still crash into small boats, get tangled in fishing nets, or run onto shallow rocks during low tides. To fully utilize an autopilot, all of these problems must be dealt with, usually through various sensors. Redundancy is also an extremely important aspect, especially when the system is running in real-time.

Further, for the launch of a commercial marine AP several safety International Organization of Standardization (ISO) and American National Standards Institute (ANSI) -tests must be passed. Further, the AP must also follow the COLREG rules. At the time writing this, no ISO-tests that are highly related to this problem statement could be found. Although, there exists ISO-tests for so called heading-hold and course-hold autopilots.

Integrity

Before we started the project at CPAC, a non-disclosure agreement was signed, and any files that were created during the project were strictly not to be distributed via other means outside of CPAC. Towards the end of the project, the report was checked and possible revisions were made to ensure the report would not include anything that could potentially harm or leak information about the company.

Our experience

We both enjoyed our time at CPAC and the project as a whole. Other than a few setbacks regarding getting the FFT to work properly, it was fairly straight forward setting up the virtual environment and the training in Unity. The real difficulty was trying to understand how to optimize all the training parameters, and most importantly, the reward function.

This was not easy, especially since one training session often lasted over 24 hours, before it was possible to tell if it had truly converged or not. Which meant, there was plenty of time for research and tweaking, but almost no time to actually test new implementations. Eventually, the only option was to have at least three computers running training sessions simultaneously.

Simulation improvements

Some of the major improvements included: re-meshing the boat model to a less detailed one, containing fewer vertices. This made all the for/while-loops related to vertices and triangles in the boat model take significant less time.

As already mentioned the waves are generated using FFT which massively improves computing efficiency and therefor performance. Other performance improvements included utilizing Microsoft's parallel for-loops. Even though Unity's API is deemed not *thread safe*, and can therefor not utilize the performance boost of parallelization, standard C#-operations can still do. Analyzing and improving the usage of garbage collector/allocation also proved very beneficial to find hidden performance. This is done utilizing the built-in *Unity Analyzer* tool.

Chapter 7

Conclusions

This chapter contains the conclusions of the Evaluation and Result chapters. Moreover, a brief discussion about future work; both for the autopilot and the simulation environment.

Before stating the conclusions from the project lets recall the problem statement:

> *"Is it possible to create an adaptable and general two-part autopilot, consisting of a regulator and a neural network based control allocator, capable of following a path despite environmental disturbances?"*

This is the main question the report aims to answer, to give a definitive answer all of the three in-going questions need to be evaluated.

- *Will the autopilot be able to continue following the path despite loss of capabilities of the actuators?*

As section 5.2 describes, the AP handles most of the losses with ease. Comparing the result in Table 4.1 and 4.5, it can be said that even though the results are fundamentally different, the total outcome is still that the AP handles even the harshness of losses on the hardest paths. To answer the question at hand, Yes, from the results it is evident that the AP manages, in most cases, to follow the path despite having a loss of capacity.

- *How well will the autopilot handle disturbances from the environment, such as wind and waves?*

By assessing the sections related to path driving in various conditions under the Evaluation chapter, section 5.1. It is apparent that the AP has no problem dealing with higher levels of disturbances of wind and waves. Strengthening this argument is Table 4.1, where comparing the AP to the heuristic data shows that not only does the AP handle the higher degree of disturbances, it more often than not manages to outperform the human driver. Once again, proving the level of robustness of the AP.

– *Will the control allocator prove beneficial for heuristic applications, where the boat is manually driven by a person?*

As stated in chapter 6, due to time and hardware limitations, there is no definitive way of answering this question. Further, as a consequence of how the training is performed, the mapping of the joystick is not straight-forward.

To connect back to the original question, did the AP perform to the level of greatness specified by the goal statement? Yes it did, even so that, according to the standard proposed by the thesis workers, the resulting AP got a 2/5 grade. Furthermore, as Section 5.3 suggests, the reliability of the AP is very good.

7.1 Future work

The first thing is really to evaluate the AP connected to a new boat model in the simulation environment. Next, in a real environment, only then can flaws be found and further improved upon. As the result shows, the AP is working well; within the limits of the simulation environment. Furthermore, it would be interesting to add a bow thruster to the boat model and mask the actions so that it only applies during low speed conditions. Also, to rework the current propulsion force produced by the thrusters; since it is an approximation. Another detail that might improve the overall functionality of the AP, would be to make the **d** factor of the PID adaptive based on the complexity of the path. Changing the **d** term to cope with the path is something that has been seen throughout the project to give good results.

As for the simulation environment, adding the possibility to connect a rig would further improve the area of use. This would perhaps strengthen the argument for why the simulation is a useful tool in today's marine development. Moreover, as the simulation environment, today, is only intended for training purposes, obstacles such as rocks beaches etc. are neglected, and could be further improved upon.

Worth noting is that, right now, the boat is not affecting the waves; adding wake patterns might be an additional area of improvement. Further, adding dynamics related to changing the trim level of the boat might also be something to look into. Also, during the course of the project, changing the spreading function of the waves was discussed. This involved the use of a more robust version, such as either the *Mitsuyasu*, or the *Donelan-Banner* spreading functions, as proposed by Horvath [33]. Donelan-Banner seems more promising, since it is supposedly more physically correct than Mitsuyasu's model. These are discussed further in the theory, see section 2.4.9.

Bibliography

[1] C. E. Billings, "Toward a human-centered aircraft automation philosophy," *The International journal of aviation psychology*, vol. 1, no. 4, pp. 261–270, 1991.

[2] R. Sutton, *Marine control systems-introduction*, 1997.

[3] S. Hardman, "Drivers of partially automated vehicles are making more trips and traveling longer distances," 2020.

[4] S. Berglund and R. Buick, "Guidance and automated steering drive resurgence in precision farming," *Precision Agriculture*, vol. 5, pp. 39–45, 2005.

[5] C. Liu, T. Sun, and Q. Hu, "Synchronization control of dynamic positioning ships using model predictive control," *Journal of Marine Science and Engineering*, vol. 9, no. 11, p. 1239, 2021.

[6] Y. Cui, S. Osaki, and T. Matsubara, "Reinforcement learning boat autopilot: A sample-efficient and model predictive control based approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 2868–2875.

[7] J. Pomirski, L. Morawski, and A. Rak, "Trajectory tracking control system for ship," *IFAC Proceedings Volumes*, vol. 37, no. 10, pp. 251–255, 2004.

[8] P. Mishra, S. Panigrah, and S. Das, "Ships steering autopilot design by nomoto model," *International Journal on Mechanical Engineering and Robotic*, vol. 3, no. 3, pp. 37–41, 2015.

[9] I. Pavić, S. Vukša, and M. Laušić, "Analysis of the nomoto ship model response to course changes using pid controller in matlab/simulink,"

[10] H. Bjering Strand, "Autonomous docking control system for the otter usv: A machine learning approach," M.S. thesis, NTNU, 2020.

[11] M. Tomera and K. Podgórski, "Control of dynamic positioning system with disturbance observer for autonomous marine surface vessels," *Sensors*, vol. 21, no. 20, p. 6723, 2021.

[12] J. Du, Y. Yang, D. Wang, and C. Guo, "A robust adaptive neural networks controller for maritime dynamic positioning system," *Neurocomputing*, vol. 110, pp. 128–136, 2013.

[13] T. I. Fossen and Ø. N. Smogeli, "Nonlinear time-domain strip theory formulation for low-speed manoeuvering and station-keeping," 2004.

[14] L. Morawski, J. Pomirski, and A. Rak, "Design of the ship course control system," in *DS 36: Proceedings DESIGN 2006, the 9th International Design Conference, Dubrovnik, Croatia*, 2006.

[15] Y. Cao and T.-h. Lee, "Maneuvering of surface vessels using a fuzzy logic controller," *Journal of ship research*, vol. 47, no. 02, pp. 101–130, 2003.

[16] Unity Technologies, *Welcome to unity*, 2022. [Online]. Available: `https://unity.com/our-company` (visited on 06/01/2022).

[17] J. Hanski and K. B. Biçak, *An evaluation of the unity machine learning agents toolkit in dense and sparse reward video game environments*, 2021.

[18] ComNav, *Autopilot*, Online; accessed 06-June-2022, 2022. [Online]. Available: `https://comnav.com/autopilot/`.

[19] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning*, `https://arxiv.org/pdf/1705.05363.pdf`, PMLR, 2017, pp. 2778–2787.

[20] S. E. Jonathan Ho, "Generative adversarial imitation learning," 2016, `https://arxiv.org/abs/1606.03476`.

[21] "Lecture 10: Imitation learning," `https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_10111213.pdf`, 2021.

[22] R. G. Lerner and G. L. Trigg, *Encyclopedia of physics, 2 volumes*. 2005.

[23] T. I. Fossen, "Guidance and control of ocean vehicles," pp. 48–54, 76–84, 1994, `http://kashti.ir/files/ENBOOKS/Guidance%20and%20Control%20of%20Ocean%20Vehicles%20Fossen.pdf`.

[24] D. Sen and T. Vinh, "Determination of added mass and inertia moment of marine ships moving in 6 degrees of freedom," *International Journal of Transportation Engineering and Technology*, vol. 2, no. 1, pp. 8–14, 2016.

[25] A. Witkowska and T. Niksa-Rynkiewicz, "Dynamically positioned ship steering making use of backstepping method and artificial neural networks," *Polish Maritime Research*, vol. 25, pp. 5–12, 2018.

[26] S. Inoue, M. Hirano, and K. Kijima, "Hydrodynamic derivatives on ship manoeuvring," *International Shipbuilding Progress*, vol. 28, no. 321, pp. 112–125, 1981.

[27] T. SNAME, "Nomenclature for treating the motion of a submerged body through a fluid.," in *The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin*, 1950, pp. 1–5.

[28] Y. Yoshimura, "Mathematical model for manoeuvring ship motion (mmg model)," in *Workshop on Mathematical Models for Operations involving Ship-Ship Interaction*, 2005, pp. 1–6.

[29] J. Alme, "Autotuned dynamic positioning for marine surface vessels," M.S. thesis, Institutt for teknisk kybernetikk, 2008.

[30] A. Fournier and W. T. Reeves, "A simple model of ocean waves," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 75–84.

[31] J. Tessendorf *et al.*, "Simulating ocean water," *Simulating nature: realistic and interactive techniques. SIGGRAPH*, vol. 1, no. 2, p. 5, 2001.

[32] F.-J. Flügge, "Realtime gpgpu fft ocean water simulation," Tech. Rep., 2017.

[33] C. J. Horvath, "Empirical directional wave spectra for computer graphics," in *Proceedings of the 2015 Symposium on Digital Production*, 2015, pp. 29–39.

[34] B. Källstrand, "Control allocation for vehicle motion control-maximizing traction and steering capabilities under different road conditions," M.S. thesis, 2017.

[35] "Chapter 7: Resistance and powering of ships," `https://www.usna.edu/NAOE/_files/documents/Courses/EN400/02.07%20Chapter%207.pdf`, Online; accessed 25-April-2022.

[36] J. Kerner, "Water interaction model for boats in video games: Part 2," 2016. [Online]. Available: `https://www.gamedeveloper.com/programming/water-interaction-model-for-boats-in-video-games-part-2`.

[37] P. Constantin and C. Foias, *Navier-stokes equations*. University of Chicago Press, 2020.

[38] wikiwaves, *Wave drift forces*, Online; accessed 06-June-2022, 2010. [Online]. Available: `https://wikiwaves.org/Wave_Drift_Forces`.

[39] R. I. Harris, "The nature of the wind. in," *Proceedings of the Seminar on the Modern Design of Wind Sensitive Structures*, (pp. 29–55), 1971.

[40] R. Isherwood, "Wind resistance of merchant ships," 1972, `http://www.cembercikutuphanesi.biz.tr/hull/displacement/wind/RINA-isherwood-1972.pdf`.

[41] C. Aage, S. L. Hvid, P. H. Hughes, and M. Leer-Andersen, "Wind loads on ships and offshore structures estimated by cfd," in *8th International Conference on the Behaviour of Offshore Structures, BOSS'97*, Elsevier Science Ltd., Pergamon, 1997.

[42] A. Juliani, V. Berges, E. Vckay, *et al.*, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018, `https://github.com/Unity-Technologies/ml-agents`. arXiv: `1809.02627`. [Online]. Available: `http://arxiv.org/abs/1809.02627`.

Appendix A

Wind coefficients

The function of the force/moment coefficients takes the form of a 9-th order polynomial as:

$$C_{X,Y,N}(\gamma_R) = p_1 * \gamma_R^9 + p_2 * \gamma_R^8 + p_3 * \gamma_R^7 + p_4 * \gamma_R^6 + p_5 * \gamma_R^5 + p_6 * \gamma_R^4 + p_7 * \gamma_R^3 + p_8 * \gamma_R^2 + p_9 * \gamma_R + p_{10} \tag{A.1}$$

p_i $(i = 1...10)$ are shown in Table A.1 for C_j $(j = X, Y, N)$. C_Y and C_N are also forced to 0 if $\gamma_R > 170°$ or $\gamma_R < 10°$, since according to [23] there are no tabulated data for those angles.

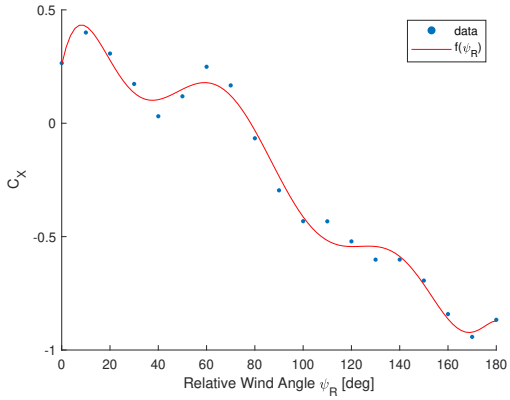


Figure A.1: C_X wind force coefficient vs relative wind angle.

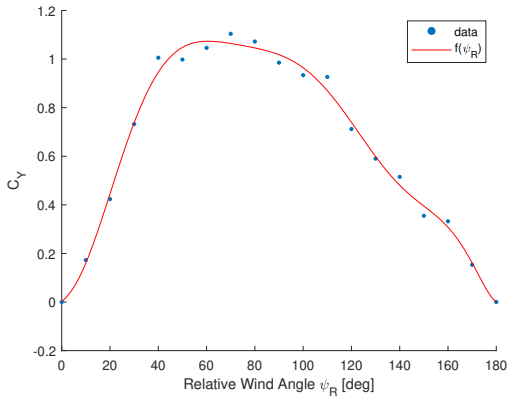


Figure A.2: C_Y wind force coefficient vs relative wind angle.

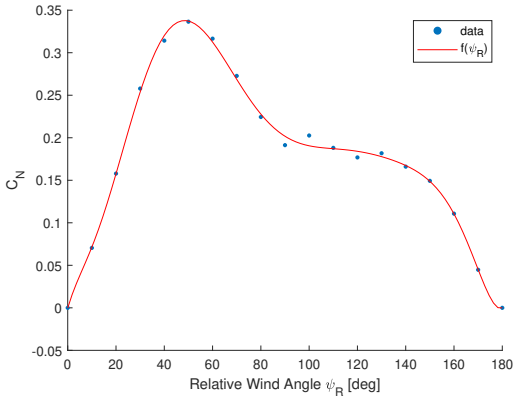


Figure A.3: C_N wind moment coefficient vs relative wind angle.

Table A.1: Force/Moment coefficients of fitted function.

	C_X	C_Y	C_N
p_1	-1.64882367716472e-17	1.21149434755140e-17	5.49108052515805e-18
p_2	1.22916417745904e-14	-9.50518372731261e-15	-4.53194159810532e-15
p_3	-3.61488114298683e-12	3.04837551595768e-12	1.55811005495267e-12
p_4	5.16612271798030e-10	-5.10377316842586e-10	-2.87052554715277e-10
p_5	-3.32200462754575e-08	4.67775061186077e-08	3.02063438699443e-08
p_6	1.43546889713605e-07	-2.16628885201390e-06	-1.77388229844598e-06
p_7	9.15407197493763e-05	3.08962520011244e-05	5.17410391200879e-05
p_8	-0.00407842083170941	0.000660920447195799	-0.000611673305111313
p_9	0.0486513349761720	0.00787254304240812	0.00961157461297032
p_{10}	0.258987460484858	0.00215461443447406	-0.000239606630111893

Appendix B

ML-Agents training parameters

```yaml
behaviors:
    BoatBehavior:
        trainer_type: ppo
        hyperparameters:
        # Hyperparameters common to PPO and SAC
        batch_size: 256
        buffer_size: 10240
        learning_rate: 3.0e-4
        learning_rate_schedule: linear
        # PPO-specific hyperparameters
        beta: 5.0e-4 #only used for discrete actions
        epsilon: 0.2
        lambd: 0.95
        num_epoch: 5

        network_settings:
            # Normalize the observations based on running average
            normalize: false
            hidden_units: 196
            num_layers: 3
            memory:
                memory_size: 128
                sequence_length: 64

        reward_signals:
            extrinsic:
                gamma: 0.995
                strength: 1.0
            curiosity:
                strength: 0.02
                gamma: 0.995
                encoding_size: 256
```

```yaml

        max_steps: 10000000
        time_horizon: 64
        summary_freq: 500
        threaded: false

torch_settings:
    device: cuda

environment_parameters:
    max_wind_speed:
        curriculum:
            # The '-' is important as this is a list
            - name: LowWinds1
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.05
                value: 2.0
            - name: LowWinds2 # This is the start of lesson 2
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.10
                    require_reset: false
                value: 4.0
            - name: LowWinds3 # This is the start of lesson 3
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.15
                    require_reset: false
                value: 5.0
            - name: LowWinds4 # This is the start of lesson 4
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.20
                    require_reset: false
                value: 6.0
            - name: MediumWinds1 # This is the start of lesson 5
                completion_criteria:
```

```
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.25
                    require_reset: false
                value: 6.5
            - name: MediumWinds2 # This is the start of lesson 6
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.30
                    require_reset: false
                value: 7.0
            - name: MediumWinds3 # This is the start of lesson 7
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.35
                    require_reset: false
                value: 7.5
            - name: MediumWinds4 # This is the start of lesson 8
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.40
                    require_reset: false
                value: 8.0
            - name: HighWinds1 # This is the start of lesson 9
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.45
                    require_reset: false
                value: 9.0
            - name: HighWinds2 # This is the start of lesson 10
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.50
                    require_reset: false
                value: 10.0
```

```
            - name: HighWinds3 # This is the start of lesson 11
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.55
                    require_reset: false
                value: 11.0
             - name: HighWinds4 # This is the start of lesson 12
                value: 12.0

    spawn_angle_max:
        curriculum:
            - name: SmallBends1 # First lesson
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.05
                value: 6.0
            - name: SmallBends2 # This is the start of lesson 2
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.10
                    require_reset: false
                value: 10.0
            - name: SmallBends3 # This is the start of lesson 3
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.15
                    require_reset: false
                value: 14.0
            - name: SmallBends4 # This is the start of lesson 4
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.20
                    require_reset: false
                value: 18.0
            - name: MediumBends1 # This is the start of lesson 5
                completion_criteria:
```

```
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.25
                    require_reset: false
                value: 22.0
            - name: MediumBends2 # This is the start of lesson 6
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.30
                    require_reset: false
                value: 26.0
            - name: MediumBends3 # This is the start of lesson 7
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.35
                    require_reset: false
                value: 30.0
            - name: MediumBends4 # This is the start of lesson 8
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.40
                    require_reset: false
                value: 34.0
            - name: HighBends1 # This is the start of lesson 9
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.45
                    require_reset: false
                value: 40.0
            - name: HighBends2 # This is the start of lesson 10
                completion_criteria:
                    measure: progress
                    behavior: BoatBehavior
                    signal_smoothing: true
                    threshold: 0.50
                    require_reset: false
                value: 46.0
```

```
217             - name: HighBends3 # This is the start of lesson 11
218               completion_criteria:
219                 measure: progress
220                 behavior: BoatBehavior
221                 signal_smoothing: true
222                 threshold: 0.55
223                 require_reset: false
224               value: 52.0
225             - name: HighBends4 # This is the start of lesson 12
226               value: 60.0
```

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY