Åbo Akademi

FACULTY OF
SCIENCE AND ENGINEERING

MASTER'S THESIS

A simulator for evaluating machine-learning algorithms for autonomous ships

Author:
Kim HUPPONEN

Supervisor:
Ivan PORRES

2020

Abstract

In this thesis, a ship simulator capable of training agents in ship behavior is presented. The simulator can be used to train and simulate agents with different kinds of behaviors. The simulator supports ships and boats of different sizes. Users can input configuration files with agent and environment data to create their own scenarios. The agent data includes ship information such as the position, speed and heading of the vessel as well as the weights of the training data which define the behavior of the agent. The environment data allows the user add a map or obstacles as well as decide the size of the area the agents move on. The simulator is capable of training and simulating multiple agents simultaneously. The simulators time is divided into one second long time steps. The application does not work in parallel but instead handles all calculations between each time step. With various configuration options the simulator can be used to simulate a plethora of scenarios. New reinforcement learning methods and ship configurations can also easily be added. The simulator can be used in educational and industrial applications in reinforcement learning or autonomous surface vehicle navigation. The simulation is shown by a simple top down view. All the objects are two-dimensional polygons and therefore, the ships motions are composed of 3 degrees-of-freedom: surge, yaw and sway. The motion calculations are based on the models presented by Ueng et al. [1].

Keywords: simulator, machine-learning, ships, ship-physics, agents.

Contents

Preface

I want to thank Ivan Porres for his guidance and giving me the possibility of working on this project. I also want to thank Sebastian Penttinen who worked on many aspects of the simulator.

1. Introduction

Automated ships are a product that is heavily researched at this time. The idea to train vehicles with automated navigation using reinforcement learning methods is quite a new one. Figueiredo explains how it was only a few decades ago reinforcement learning started being used with ship maneuvering [2]. It is researched worldwide due to the freedom of the seas and the usefulness of unmanned vehicles in many fields of the industry [3]. While there are some ships that claim to be fully automated there is still a great deal of research and testing to be done before the technology becomes mainstream. Some successful projects do exist though. Rolls-Royce has the AAWA - The Advanced Autonomous Waterborne Applications Initiative - project which researches designs and other aspects used with autonomous ships [4]. The MUNIN project is establishing a concept for unmanned merchant ships [5]. Various university projects and even militaries uses smaller autonomous vessels such as catamarans successfully [6].

According to Kretschmann et al. [3], operating an unmanned bulk carrier for shipping would be 3.4 percent cheaper than via conventional methods. Using an autonomous unmanned bulk carrier for 25 years would save 4.3 million USD compared to a crewed ship. Having no crew is one of the main aspects that makes the voyaging cheaper. While the initial cost may be higher with the implementation of the additional technologies in the ship and in the shore control center, the voyage costs would be cheaper. Using an unmanned vehicle would also lead to new ship designs to emerge with the room used for the crew reused for other purposes. The outer design of the ship can be adjusted for better drag as the ship is not bound by the minimum sight restrictions from the bridge. The combined effect of decreased drag and fewer life-supporting systems provide new ways to reduce fuel consumption.

As with all automated vehicles safety is of the utmost importance. Vessels carrying valuable items and lives should be in safe hands. Replacing a fatigued crew with onboard automation and monitoring from the shore causes less accidents suggests Ramos et al. [7] as 62 percent of maritime accidents are caused by human error. To achieve a safe environment with no collisions, more testing and research is needed. While one could go and build an automated ship and start sailing with it and testing in real-life conditions the simpler solution would be to train and test the vessel using a simulator [8].

A simulator gives the opportunity to use objects with the same parameters as in real life to estimate their behavior in various situations. A simple problem could be solved analytically, but more complex problems may require the use of a simulator to calculate the end results. Using a simulator allows the user to configure and test different configurations for the vessel and make quick adjustments in its behavior algorithms. Using a simulator to train an autonomous vessel is also a safe way to train as no accidents can happen. Training with a simulator enables the user to train a vehicle more than if he would do it manually, with the risk of the simulator not behaving as expected. Nearly all fields use simulations to some extent to determine an acceptable model to use. Using a simulator allows the user test the vehicle in exceptional situations as well which may be hard to come by in real environments but still be accounted for. The process of training in a simulation and then exporting it to the real world is called sim-to-real [9]. The differences between the simulation and the real world is called the reality gap.

1.1 Goals

The goal of this project is to create a simulator that is capable to train, develop and evaluate autonomous navigational algorithms for agents. This simulator is created based on requirements that other simulators did not provide. Even though there are many simulators on the market, only a few can represent a marine environment and implementations of different kind of vessels with an inbuilt way to train navigational algorithms.

The main objective at the start was to create or find a simulator to develop and evaluate autonomous navigation algorithms. Agents controlling a ship and trained with the users choice of reinforcement learning method should be able to be trained and rendered through the simulator. A key element of the simulator is the performance of the simulation. The graphics need to be impressive as long as the simulator itself is capable of running simulations several times faster than in real-time. The simulator can be represented by a top-down 2-dimensional view but a 3-dimensional simulator is fine as well. The simulator should be able to handle multiple ships and their physics and provide a possibility for different agents to sense and direct different ships. These requirements are common and provided by most available simulators. The simulator should also collect and store information of the different scenarios. The collected information should include the time to complete a scenario, distance among ships, coast and destination, angle between ships, ship characteristics and the list of objects the ship can sense.

The simulator should be able to discern between terrain and ocean. Furthermore, the ships should be able to collide with other ships as well as terrain elements. A file format that lets the user define new scenarios with the terrain and the initial state of the ships should be implemented. The results should be collected with a possibility to be used in calculations and be compared with other results.

The simulator must have an interface in which the user can observe the decisions the agents have made. An additional desired feature would be to add the possibility to run the scenarios without the graphical interface to further accelerate the testing of the agents.

Adding new features to already existing simulators would be difficult if not impossible in many cases. Designing and creating our own simulator allows us to develop it with the requirements we have in mind and with the possibility to add more features as necessary.

2. Requirements

A robotic simulator is a simulator based on a vehicle or vessel of some sort and that usually follows the same kind of model. A vessel usually has a sensor that lets it observe the environment and an effector that makes it act on the environment [10]. The vessel also has a control system that decides how the vessel should act.

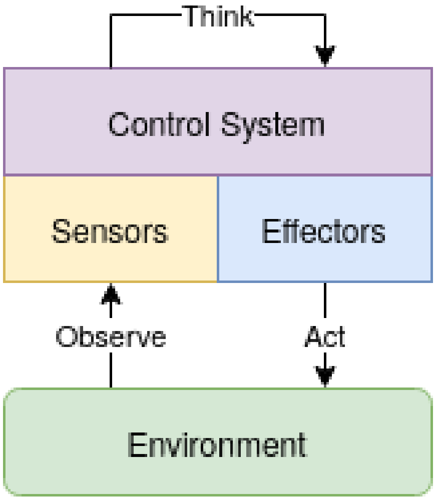


Figure 2.1: A simple figure of how a robot system usually looks[10].

Sensors tell the ships what they know of the outside world. A camera, for example, gives the ship eyes to know what is in front of it. An impact sensor can sense collisions and using GPS or AIS (Automatic identification system) can locate the ships position. A compass and an accelerometer can tell the direction and velocity of the vessel. Testing of all these sensors is often simulated, as real life testing can be expensive and time-consuming. The simulated versions of the sensors are often approximations of their real-world counterparts, trying to provide values as accurate as possible. The position, for example, could be acquired from a GPS sensor while the simulator gives the location outright since the simulator itself knows the values of all the objects being simulated.

There exists a plethora of different kinds of simulators with different types of applications and focuses. Depending on the target application, a simulator may focus more on some aspects and less on others. This paper will compare some aspects of different physics engines to our solution in the following areas: reproducibility, modularity, accuracy and the possibility to run the simulation in parallel, headless and accelerated mode.

2.1 Reproducibility

A simulator should always yield the same result with different executions of the same scenario. A common reason for different results is if randomness is incorporated in the calculations. In these cases, the randomness should use a seed to ensure that the same result will still be achieved on different executions of the same experiment. Another cause for different results may be different architecture between the computers running the simulator. This may happen when the physics engine and the users code communicate with each other using sockets with values that may reside in different threads and processes.[9]

2.2 Modularity

A good simulator should be built with modularity in mind. The simulator should be able to handle a varying degree of different agents that interact with the environment. Different vessels should be able to sense the world using different kind of sensors. An easy way to add or remove specific sensors and decide which are being used to sense the world around them is an important part of a good simulator. These sensors would represent their corresponding variant from the real-world. For example, a camera could serve as the eyes for the ship in a cone format in front of the ship or using a lidar to observe a 360-degree view around the ship. These could be represented with polygons drawn around the ships to show how far the ship can sense its surroundings. Most ships use ASI to obtain pivotal information of their surrounding ships and it would be a formidable addition to a vessel that tries to mimic a real world setting.

2.3 Accuracy

The simulator must provide results that resemble the results as the ones in the real world as close as possible. The accuracy of the simulator is one of the most important points when designing one. Can the results of the simulation be used in a real-time environment? The final goal when training autonomous vessels are to deploy them in the real world. Does the simulator support this? Simulating a ship is usually one of the first steps, along with designing algorithms, of the process of landing a ship on water. There are still many steps to be done after simulating a successful model such as making a real-life model ship with the simulated data. Measuring the behavior of a model ship provides valuable real-life results that may also help in finding problems in a simulated environment [11].

2.4 Parallel Simulation

Is the simulator capable of simulation multiple agents at a time? This may be achieved by running the simulator with parallel processes. Having a simulator be able to process multiple simulations simultaneously allows for more and therefore faster training to take place. If a simulator is not capable of parallel processes can it simulate multiple agents some other way? To only be able to simulate a single agent at a time is quite a limiting factor of a simulator and should be avoided.

2.5 Accelerated Simulation

Accelerated simulation means that the time in the simulation is accelerated and the objects move faster than in real time. Is the simulator able to increase the speed of the environment? In order to increase the amount of training of agents an accelerated simulation is required. The acceleration of time can be done in the training phase for the agents or during the rendering process. Having ships only move in real time is not feasible as bigger ships move quite slowly and take a long time. Having to wait the actual time a scenario takes to complete dismisses one of the key elements a simulator can provide which is to have more tests done in a shorter time frame.

2.6 Headless Simulation

Can the simulation be run without rendering? This would prompt the simulation to run much faster than if it were to be shown on screen, allowing more training to be taken place. As efficiency is often the most important part of a simulator, achieving faster training takes top priority. Having to watch the training take place is usually pointless as one is usually only interested in the end results of the training.

3. Background and related work

Before we could start implementing our ship simulation project, we needed to explore the market for already existing software. In this phase, we were mainly focusing on robotic simulators with capacity for extensions regarding ship physics and reinforcement learning. In this chapter, we compare and describe already existing boat simulators, engines and reinforcement learning solutions.

When comparing the simulator we will evaluate if they can fulfill the goals mentioned in Chapter 1. These requirements are quite specific and a simulator that fulfills all the requirements is going to be difficult to find. There are simulators that simulate ship behavior but they are not perfect in every aspect. As stated by Paravisi et al. [12], robotic simulators so far are not made for marine robotics and most solutions are still in development. There are no standards to follow with marine robotics which causes the research of unmanned surface vehicles to stall.

3.1 The Robot Operating System (ROS)

The Robot Operating System (ROS) is a framework for robot software [13]. It is a collection of libraries and conventions that has plugins for most robot simulators. ROS is a general framework, but with more and more specialized tools as it keeps expanding. It uses peer-to-peer technology to host and connect to the framework. It supports multiple programming languages by using a language-neutral interface definition language which then translates to the different programming languages. ROS is built from several different tools which can be thought of as nodes that are connected to each other.

ROS has been used in a few different autonomous surface vehicles such as the heron [14]. While ROS is a popular robotic framework with many tools, it is unsuitable for our simulator as it adds unnecessary complexity. Almost no maritime tools exist yet in the ROS library, and we would have to develop these tools ourselves. While the heron uses ROS for its sensors it is by far an outlier. The robotics ROS is focused on are land vessels, humanoid robots and conveyer belt robots. A plugin for ROS is a possible addition for the simulator in the future when the simulation results are applied to real-life vessels and the ROS library has expanded with more maritime tools.

3.2 Open Dynamics Engine (ODE)

The Open Dynamics Engine (ODE) is an open source physics engine. ODE is a physics package that can be used independent of the graphics library. ODE is used in various computer games and simulation tools. The core of the ODE is the Physics SDK, a development kit that simulates the behavior of objects in the simulation. The environment and the current state are sent to the SDK, and it calculates the next step which then can be represented graphically. Several of the simulators discussed in this chapter use the ODE physics engine such as Gazebo and Webots.[15]

3.3 OpenAI

OpenAI is the staple in artificial intelligence and is used by many reinforcement learning solutions [16]. Reinforcement learning is the part of machine learning which handles the decision-making. Our simulator needs some sort of intelligence to be able to handle collision avoidance feature. Using OpenAI ensures that the simulator will follow the industry standard of current reinforcement learning. Our simulator should support the way OpenAI uses environments to save collections of tasks.

The environment consists of steps where the agent takes an action and then receives an observation and reward depending on the result of the action. By giving different rewards for different observations the agent will learn to decide which actions it should take in different situations. This concept can be used to make the agents do what the user wants by giving appropriate rewards for different observations.

3.4 Gazebo

Gazebo is a 3D robot simulator that is widely used by roboticists. It uses an SDF (Simulation Description Format) format which is an XML format for easy creation of environments, robots and controls. It is an open source project capable of running on Linux systems and therefore keeps on evolving into different kind of packages depending on the users needs. Extensions exists which lets one use the OpenAI Gym and ROS on Gazebo [17]. Gazebo interacts with the world using bodies, sensors and joints. Joints are the moving force in the robots which move and interact with the bodies and the sensors read the data surrounding them. Gazebo does not come with fluid physics out of the box but some Gazebo-based simulators have successfully implemented them. [18]

3.4.1 Gym-Gazebo2

Gym-Gazebo2 is a robotic environment compatible with the OpenAI Gym and it uses the Gazebo simulator as its physics engine. The project uses a socket-based communication between the simulator, environment and the code causing the simulation not to behave identically on each run. It does not have any water physics and it is mainly focused on gripping objects. [19]

3.4.2 Gym-Ignition

The Gym-Ignition is a reinforcement learning framework that uses the Gazebo engine for simulation. It is targeted both for simulations and real-life settings [9]. The simulator has a feature which lets it integrate new physics on the fly during the simulation written in either C++ or Python. The simulator supports the OpenAI Gym interface which makes it compatible with most of the reinforcement learning projects. It uses a newer version of the Gazebo simulator called the Ignition Gazebo which is part of the Ignition Robotics suite[20].

Gym-Ignition has many of the features our project seek. It uses the common OpenAI gym interface which is almost a standard for reinforcement learning. The physics engine can run in accelerated mode with the possibility to run with parallelized threads for better performance. The main concern about using this framework for our project is that it is new and lacks thorough research. The focus of Gym-Ignition does not currently lie with ships. It does not include water physics. While it is capable of running multiple vessels, it can only operate on a single one. It is not currently capable of gathering information of sensors. The documentation is lacking as it is still in development as of the moment of writing.

3.4.3 USVSim

USVSim is one of the newest simulators in the field of unmanned surface vehicles (USV) [12]. Paravisi et al. concluded that there is adequate USV simulator currently on the market, but it would be unnecessary to create one from scratch. They built their simulator using Gazebo and adding features from other plugins for water and ship calculations. It uses UWSim which focuses on underwater vehicles for its water physics and underwater sensors [21]. The simulator supports water currents and waves and sensors for the ships. The simulator is still new and focuses on smaller vessels. Their focus lies in having the water physics be as close to reality as possible. The navigational aspects of maritime traffic have not been of much concern in this project, though.

3.5 Safety Gym

The Safety Gym environment focuses on the safety of the agents [22]. It separates the task performance and the safety so that both can be taken into accord. The most efficient way may not always be the safest option. The environment lets one measure and control these statistics to find a solution which works for the user. While our project focuses in having the agents avoid collisions, the Safety Gym does not provide any water physics and is still in its early stages of development.

3.6 Webots

Webots is a simulator created by Cyberbotics mainly focused on robotics. Webots provide the user with a prebuilt library with predefined actuators and sensors. The sensor library is vast, and each sensor can be finely tuned. Webots is built with the idea of transferring the code to real robots and has made it easy. Webots operates in accelerated time and the simulation may be speed up or slowed down. The Webots interface allows the user to interact with the simulation by clicking and dragging while the simulation is running. It is mainly used for development of land robots on feet or on wheels. [23]

3.7 Kelpie

Kelpie is a simulator for water surface and aerial vehicles [24]. The simulator is open source and developed as a part of the Riverwatch experiment for Echord, a project focusing on the development of robotics [25] for maritime use. The simulator uses an architecture resembling Gazebo but with the possibility to simulate water for 3D rendering and ROS for robotic frameworks and sensors. The physics are done via BulletPhysics library which is an open source physics engine. The simulations are loaded using the Simulation Description format (SDF) which describe the scene, robots and their sensors. The simulator supports waves and currents and works in six-degrees-of-freedom. While the simulator is great in various ways it is still in rapid development and as it is a custom build it does not have support for machine learning algorithms.

3.8 Transas

Transas is a company that was acquired by Wärtsilä in 2018 who specialize in ship and bridge simulations; AI-based decision-support tools as well as applications that handle real-time information [26]. While they are experts in their field, their navigational simulators are developed in-house and support only their own research and training. Their simulator cannot be used for development of our own algorithms and does not provide the freedom of using any reinforcement learning models the user wants to use.

3.9 Oceanic Corp

Oceanic Corp is a consulting company specialized in modeling ships [27]. They sell their benchmarked and validated simulators as services. As their simulators are not free they are not a candidate for our project. Their simulator does not have inbuilt support for reinforcement learning.

3.10 University projects

Hydronet is a project done by Wardt et. al [10] as part of their thesis. Their solution resembles the one we ended up creating as it is written from scratch and uses a QT interface with the difference that they programmed it in C++ instead of python. They built their physics through a model made with Simulink which is an environment for model-based simulation based upon Matlab. The project is not accessible to the public.

Sandaruwan et al. [28] made a simulator that works with six-degrees-of-freedom and is benchmarked from real ships and validated from real ships and models. As several of the other discussed simulators, Sandaruwan et al.:s is not freely distributed either.

4. Design

This chapter will explain different aspects about the simulator that was created, which fulfills the goals given in Chapter 1. While this chapter focuses on the design of the simulator, the evaluation of how well the simulator fulfills the requirements will be discussed in Chapter 5.

4.1 Our solution

Our solution was to create a simulator from scratch using Python. By not depending on a physics engine we had to create our own physics for the ships. These physics are further explained in the next section. To verify that the physics resemble real-life behavior we ran the ships through a couple of tests, which will be explained in Chapter 5. This simulator is viewed from a top-down 2D perspective. This version of the simulator is going to operate with three-degrees-of-freedom. This means that the ships can only move forward, backward and to the sides as well as rotate around the vertical axis. It will exclude external forces that would affect the ship.

These limitations will simplify the physics calculations as we do not need to take the ship's vertical movement and the rotation around the longitudinal and transverse axes into consideration. Having a simple graphical interface allows the simulator run with better performance. The positives of making this simulator from scratch is that it has the possibility to be further expanded unlike a third-party software. This has given us the possibility to have a surface vessel focused simulator with inbuilt capabilities to run reinforcement learning algorithms for navigating ships.

4.2 The configuration file

The user can use the configuration files to decide what kind of ships and what kind of environment a scenario consists of. The configuration file can be seen as a scenario where we decide which ships should go where and with what kind of obstacles. The configuration file uses the yaml format which is a human friendly data serialization standard resembling JSON [29]. There are three roots in the configuration file; the scenario, map information and eventual obstacles.

The first is the scenarios. The nodes under the root are the names of each different scenario. Each scenario has an amount of ships with their own plethora of configurations. Some of the settings are mandatory, so that the physics calculations can work as intended. The identification name of the ship is the first mandatory setting. This name is used through the application to identify a ship. It must not be the same as another ships identification name. The position of the ship is mandatory for placing of the ship on the map. The size of the ship consists of the ships length and width and its mass. These are used in calculations of the forward and turning accelerations and are thus mandatory. The maximum speed of the ship is mandatory so that the simulator knows when the ship has to stop accelerating. The minimum speed is optional, but if omitted the simulator assumes that it is zero. This will stop the ship from running backwards as a minimum speed is recognized as the maximum backward velocity a ship can achieve. Rudder information is mandatory for knowing the limits of the rudder. The simulator assumes that the maximum and minimum rudder angles are -35 and 35 to avoid stalling. The rudder settings that the configuration file needs are the rudder area, rudder coefficient and the time it takes for the rudder to turn the ship. The rudder area is the total area of all the rudders if a ship has multiple rudders. Next is the propeller data with the thrust coefficient of the propeller and the diameter of the propeller. These are used to calculate the forward force of the ship along with the maximum and minimum revolution speeds of the engine which are also mandatory. The last two mandatory settings are not physics related but important none the less. They are the destination coordinates and the autopilot. The destination coordinates tell the ship where it is supposed to go if the autopilot setting is on and the autopilot setting tells the simulator if the autopilot should be used or not.

Each ship has an agent field. This field dictates if the ship is controlled by an agent or not and what kind of agent. Leaving the field empty causes the ship to not be controlled by an agent. Another field is used to determine if the ship should be the one trained upon during training. The weights for the agents are optional since the application will provide a default value if it is missing. The weights are the results of the agents training and determine the actions the agents make in different situations.

The second root is the map root which determines the size and the obstacles of the map. None of the values are mandatory in the map root and are given default values if left empty. The map root consists of the path to a map image, scale, size and offset. The size of the map is given in pixels and is taken from the map image if provided. The scale and offset settings scale and change the position of the environment. The environment will be further explained in an upcoming section.

The third root is the land root and has the coordinates of possible obstacles. The user can add differently shaped and sized obstacles by adding a list of coordinates as the corners of the obstacle. These obstacles will cause the ships to collide with them if they touch each other. Obstacles are not mandatory and can be excluded if they are not needed.

4.3 Ship characteristics

The simulator supports a varying cast of ships. Tests have been done with smaller ships such as a lifeboat in Appendix A.2 and bigger ships such as the oil tanker in Appendix A.1.

4.3.1 Size

The configuration file provides the simulator with the knowledge of the ships parameters. The configuration file gives the position, target position, rudder angle, heading, current speed, maximum and minimum speed, target speed, mass, length, width, rudder area, rudder coefficient, rudder turn rate, propeller coefficient, propeller diameter and the maximum and minimum rounds per minute for the engine. While most of the information is mandatory some may be omitted and the simulator will use default values instead.

A common use for the simulator is to train ships to avoid collisions. This places the utmost importance on knowing the correct position of the exterior of the ship. A ship can be heading at an angle between 0 and 360 degrees and the pivot point of the ship is not in the middle of the ship. These conditions make us use the following calculations to correctly calculate the ships boundaries.

Calculating the corner γC:s coordinates Cx and Cy of a rectangle with an angle α is done by the following formula:

$$Cx = x + cos(\gamma C + \alpha) \cdot \sqrt{width^2 + height^2}/2$$

$$Cy = y + sin(\gamma C + \alpha) \cdot \sqrt{width^2 + height^2}/2$$

Where x and y is the middle point of the rectangle and $width$ and $height$ are the width and height of the rectangle. γC can be any of the four corners along with:

$$\gamma C_{1Front} = arctan2(-width, +height)$$

$$\gamma C_{2Front} = arctan2(+width, +height)$$

$$\gamma C_{3Back} = arctan2(+width, -height)$$

$$\gamma C_{4Back} = arctan2(-width, -height)$$

arctan2 is a function that takes into consideration the signs of its input and gives a result that always is between -π and π.

The formula needs to be changed a bit when the pivot point of the ship is not located in the center of the ship. The pivot point lies at one third of the ships length and as such we need to adjust the formula to take that into consideration. By adjusting the distance of a corner to the middle point we can ensure that the positions of the corners will be fixed. By changing the height for the front corners to:

$$height_{front} = 4/3 \cdot height$$

and the back corners to a third of the length:

$$height_{back} = 1/3 \cdot height$$

and change the height in their respective formulas to their new corresponding heights and we achieve our desired results.

To calculate the position of the area of the ship a python library named Shapely [30] is used. Shapely is a library with functions which help in calculating geometrical objects. Using the corners we obtained from our calculations we can calculate the exact and position of the ship area, and by using Shapely's functions to notice when polygons are touching each other we can accurately detect collisions. This is will be the impact sensor for the ships.

4.4 Ship motions

There are six different motions the ship can move in: surge, sway, yaw, heave, pitch and roll. Our simulator supports surge, sway and yaw for simplified navigational algorithms. Heave, pitch and roll are not used in the model our simulator provides. These would unnoticeable in a 2D environment as they do not move the ship from its location.

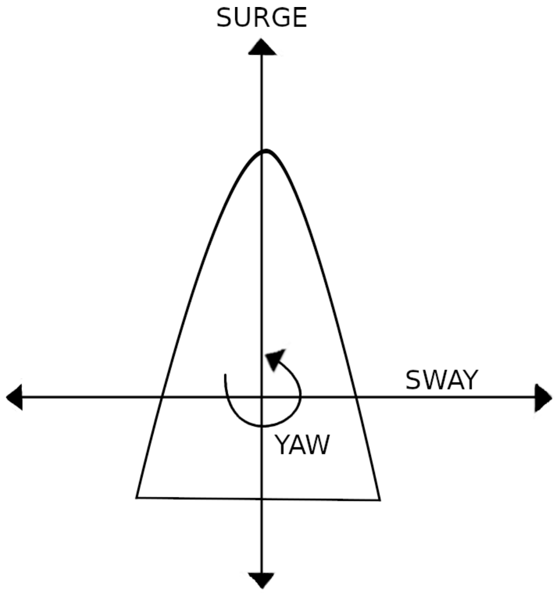


Figure 4.1: The three ship motions used in the simulator: surge, sway and yaw.

Heave is the ship's movement in the vertical axis a.k.a. up and down. Pitch is the rotation on the transverse axis and roll is the rotation around the longitudinal axis. While this simulator does not support these motions there is plenty of research on models using them such as Ibrahims [31], Khaleds [32], Rawsons [33] or Sandurawans [28, 34], but there are other successful models that only use three-degrees-of-freedom such as Mainals approach [35]. The motions not supported in this simulator work better with a three-dimensional solution where one can see the movements of a ship on waves. This version of the simulator does not include weather conditions such as wind, currents and waves.

The ship motions model our simulator uses are based on the ones created by Ueng et al. [1]. Their model is based on six-degrees-of-freedom using all the ships motions but we are only using three of them in our simulator. The simulator is built using a discrete method where each calculation is updated each step. Each step is small enough that we can use continuous calculations for our simulations. For future improvements one could include Lees et al. [36] model which also uses the propeller and rudder in their calculations but operates in six-degrees-of-freedom.

4.4.1 Surge

The surge is the forward and backward motion of a ship. It is the movement along the longitudinal axis. When no forces affect the surge motion the ship will remain still, and the velocity of the ship will be 0. Turning the ship on and starting the propellers of the ship will cause the ship to move forward. The water will work as a resistance for the ship and the ship is slowed down by waters drag force. The ship is given forward momentum from the propellers and is pushed back by the water [37]. This is the total force affecting the forward momentum

$$TotalSurgeForce = PropellerForce - WaterDragForce$$

Renamed as:

$$T_{fs} = F_p - R$$

The Drag force from the water can be formulated as:

$$R = b_d \cdot A \cdot v^2$$

Where R is the resistance (drag force), b_d is the drag force coefficient, v is the speed of the ship and A is the area of the ship hull under water. The drag force coefficient decided how long it takes for a ship to reach its maximum and minimum speed. A lower coefficient will have the ship accelerate and decelerate slower. We will replace the hull area under the water with the mass of the ship M as the area can be difficult to calculate. The mass or displacement of a ship is the amount of water that the hull displaces. The Drag force is then calculated by:

$$R = b_d \cdot M \cdot v^2.$$

The Propeller Force can be formulated as:

$$F_p = K_{pt} \cdot rpm^2 \cdot D_p^4$$

Where Fp is the force from the propeller, K_{pt} is the thrust coefficient of the propeller, rpm is the revolution speed of the propeller and D_p is the diameter of the propeller. We can then calculate the acceleration of the forward momentum by dividing the total surge force with the mass of the ship:

$$a_s(t) = T_{fs}/M = (F_p - R)/M$$

The speed for each time step can then be calculated with:

$$V_s(t) = V_s(t-1) + a_s(t) \cdot step$$

The calculations are based on continuous calculations while the simulator is using discrete time as each calculation is calculated for each step in the simulator. Having small enough steps will have the same result as a continuous approach.

4.4.2 Yaw

The Yaw is the rotating motion around the vertical axis of a ship. The ship turns by moving its rudder. The rudder can turn the ship by a maximum of 35 degrees without causing stalling. Stalling is caused if a ship would be turned too much at once and the ship would collapse to its side. As such 35 degrees as a maximum and minimum for a rudder is a standard with larger ships. The rudder causes a force to turn the ship into the desired direction but is also pushed back by the water resistance. The rudder movement affects the stability, maneuverability and fuel consumption of the ship. As the rudder causes more resistance for the ship if it is in an angle the fuel consumption rises accordingly. A ship with the rudder in a neutral position causes about a percentage increase in the resistance of the ship while an angled ship increases the resistance up to 6 percent [38]. While the shape of the rudder differs from vessel to vessel the differences are negligible with a few percent difference in turning rates and can be considered the same shape in our motion calculations [39]. The difference in shapes do affect the fuel consumption considerably though according to Lehmann [40].

There are many models to choose from when calculating the rudder motions. We will use Uengs et al. model but other models such as Lins, [41] who uses discrete calculations could be another option. The total force affecting the yaw is calculated by:

$$F_y = F_r - R_y$$

Where F_y is the total force affecting the yaw, F_r is the rudder force and R_y is the forces resisting the rotation. F_r can be formulated as:

$$F_r = K_{rd} \cdot A_{rd} \cdot V_w^2 \cdot O$$

Where K_{rd} is the rudder coefficient and A_{rd} is the area of the rudder. V_w^2 is the water speed. O is the angle of the rudder. R_y can be formulated as:

$$R_y = b_y \cdot I_y \cdot \omega_y$$

Where b_y is the resistance coefficient for yaw, I_y is the inertia moment for the yaw and ω_y is the angular velocity of the yaw. The resistance coefficient b_y affects the movement of the ship and a higher value would cause the ship to not move straight. The simulator has b_y thus as a small value so that the ship will move straight when the rudder is at 0 degrees. A higher value of b_y would simulate stronger weather conditions. The inertia I_y is calculated by:

$$I_y = M \cdot 2L/12$$

Where *M* is the mass and *L* is the length of the ship.

4.4.3 Sway

Sway is the movement on the transverse axis of a ship, usually caused by external forces. These external sources could be wind, currents or waves as an example. This simulator does not have any outwards forces that would affect the sway of the simulated ships. In a future version where weather conditions are added sway should be implemented.

4.5 Environment

The ships sail around in an environment built from a map or empty space with with obstacles. The environment is configured in the configuration file which includes all the relevant information for the simulation to create the scenario.

4.5.1 Map

The map data in the configuration files allows the users to import a path for a map, the scaling for the map and/or decide the size of the area the ships are moving in. If a map was inputted, the simulator will go through each pixel and refer the color of the pixel with a list of allowed colors. This will determine what is land and what is water. The allowed color palette is not configurable from the configuration file. If no values are inputted the simulator will make a area of the size of 800 x 600 pixels with a scaling of 0.1. This results in a map of the size of 8000 x 6000 pixels with plenty of room for ships to sail in. A pixel is 1 meter in the simulation.

As the simulator is built to find land based on the pixel colors on the map, the simulator has to do much work when starting up. As such the simulator only support maps that use the same color scheme as the default map. Larger maps will also cause a longer wait time when each pixel is calculated for and the simulator is currently more suitable on smaller scale scenarios. Inclusion of larger maps could be an improvement in the future.

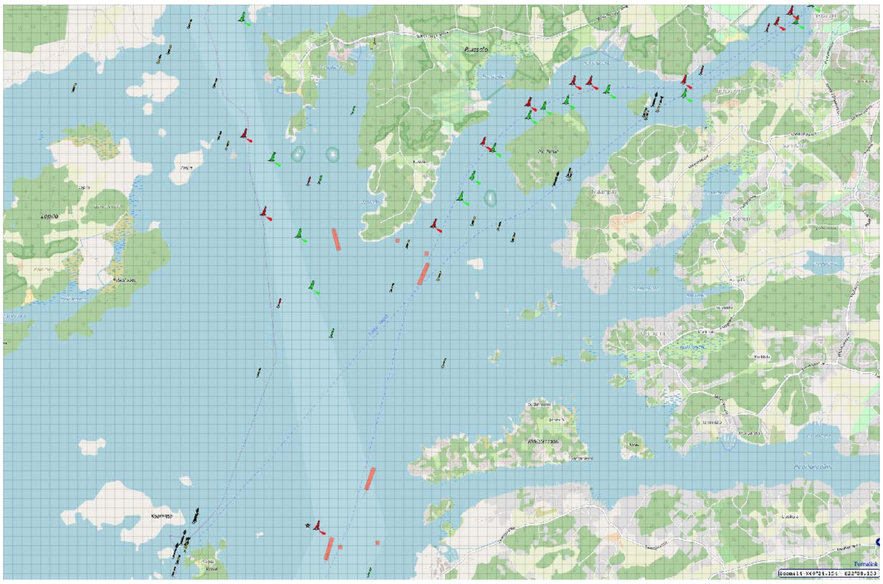


Figure 4.2: Example of an imported map.

4.5.2 Obstacles

The Land parameter of the configuration file allows the user to fill in coordinates for obstacles for the ships. Multiple obstacles can be inputted. It is counted as an collision when a ship and an obstacle touch. Obstacles are inputted by giving a minimum of three corners for the simulator to calculate the polygon formed from the coordinates. The simulator will give an warning if the coordinates are outside the range of the map. Adding obstacles is not mandatory.

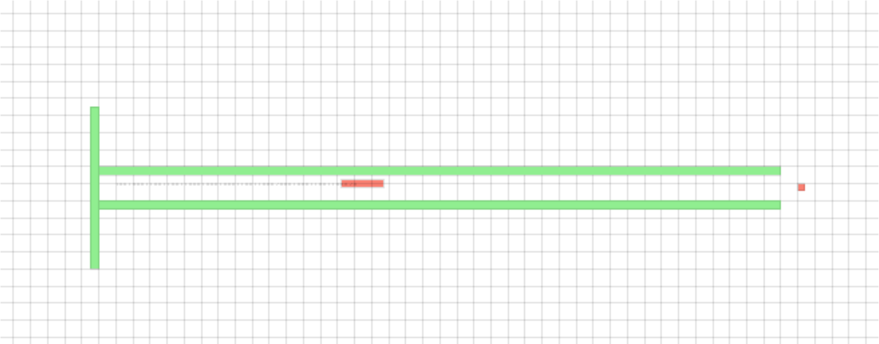


Figure 4.3: Example of three imported obstacles.

4.5.3 Land

Objects that are not ships are referred as "land" in the simulator. This includes the landmasses from the map and obstacles imported from the configuration file. While obstacles imported from the configuration file have automatically calculated borders for collision detection, the map has a list of coordinates with pixels it has determined is land. To calculate the border for each landmass from the list of pixels the simulation buffers up each land pixel by a couple of pixels so that the pixels closest to one another touch each other. The pixels form bigger landmasses with borders to the sea. The simulator can then calculate the outer border of the clumps of land and obtain a list of all the landmasses. The landmasses are not perfect as the outer pixels of each clump of land is also buffered resulting in that the land has swollen a few pixels. This also cause the land to have "soft" edges.

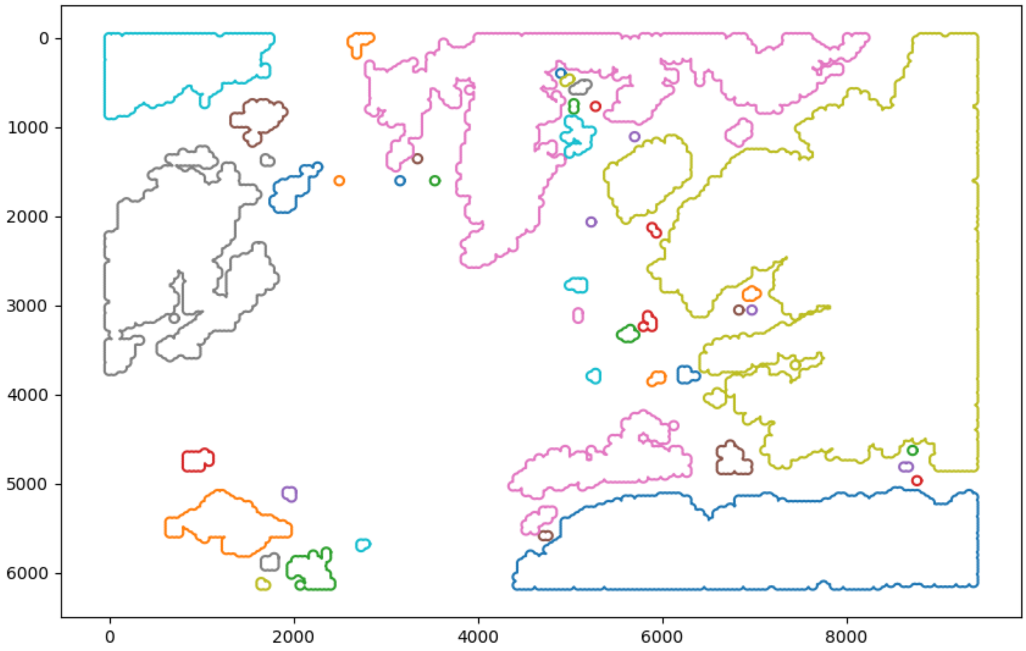


Figure 4.4: How the ship sees the map. Only the exterior of each landmass exists.

The landmasses have much more bounding pixels than an obstacle with a couple of corners. The simulator would slow down to a crawl if each ship would have to calculate the shortest distance to each bounding pixel for each landmass on the map to determine the closest land. The simulator has solved this problem by using the rtree model.

4.5.4 Rtree

Rtree is a height-balanced tree data structure [42]. The root nodes on the tree can be visualized as bounding boxes of the elements that are inside. A few big boxes will try to fit every object inside them. The children of the root nodes then work inside these big boxes and try to cram the elements in smaller boxes themselves. This continues until each element in the area has a bounding box around itself. These are the leaves of the tree and can be found in the lowest section in Figure 4.5.

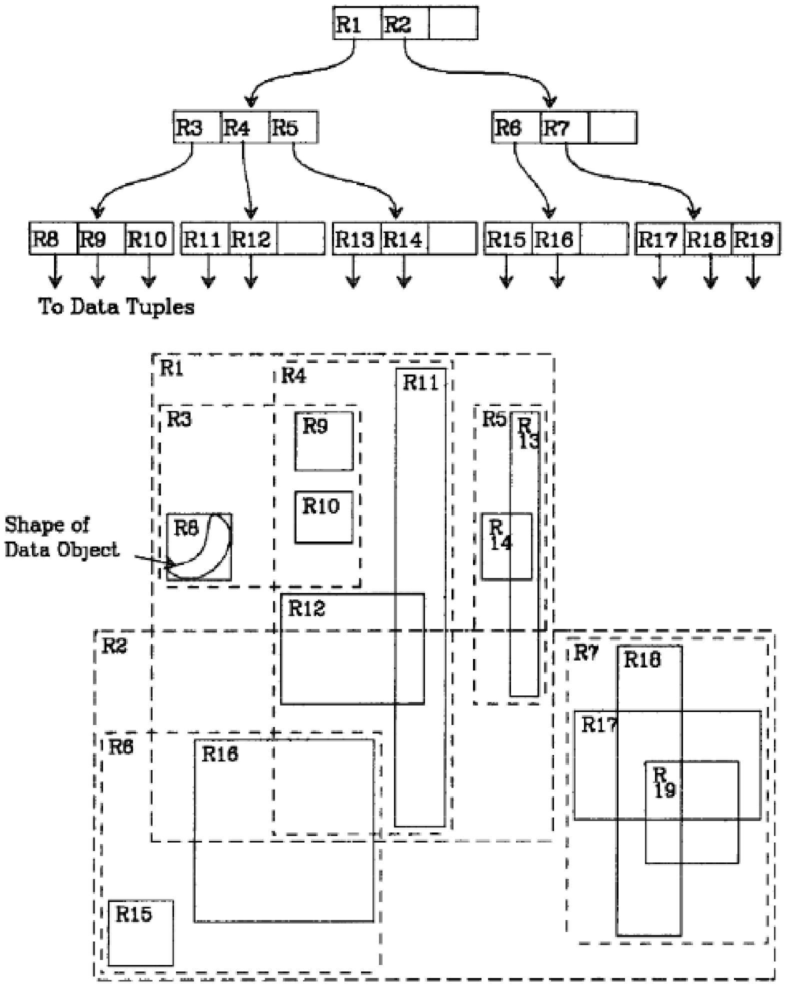


Figure 4.5: A visualization of the rtree data structure [42].

The simulator uses the rtree tree structure to help it calculate the distance from ships to land. If we start by assuming that we are only interested in knowing the distance from a ship to the closest land because we want to avoid a collision with it. In that case we do not need to know about the location of land when the ship is not close enough to care about land. If we would only care about finding the collision the solution would be to only check if the ship object is overlapping with a land object. Alas we want our ships to avoid land so the distance is crucial information for the agents to have. The simulator uses rtree to calculate the bounding boxes for each land piece.

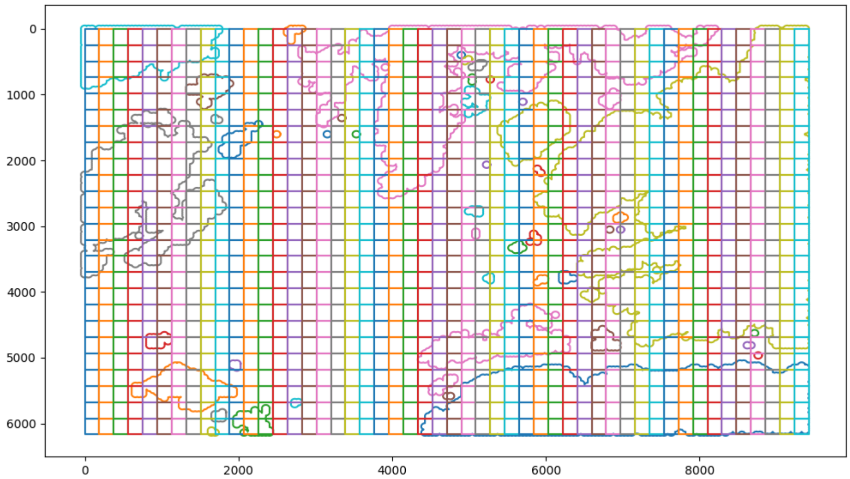


Figure 4.6: A grid for the map.

The map is first divided in a grid of rectangles of the same size. These are the bounds of the bounding boxes we will use. Without the grid each box would be as small as possible but still fit a land object. This way we gain some leeway and have some distance between the ships and land when the ships know that the land exists. This method is not perfect as it is still possible to have the bounding box land too close to the land and is something that could be improved upon in the future by giving the boundary a couple more cells to work with for example.

When a ship enters one of the bounding boxes the ship will know about the land that resides within it as seen in Figure 4.8. This way each ship only calculates the distance to the lands close to it that reside in the same box as itself. On the other hand this means that the ships have no idea if land is close by when sailing farther away from the shore. This will lessen the calculation times for the application as it does not need to calculate each pixel for each point of boundary of land.

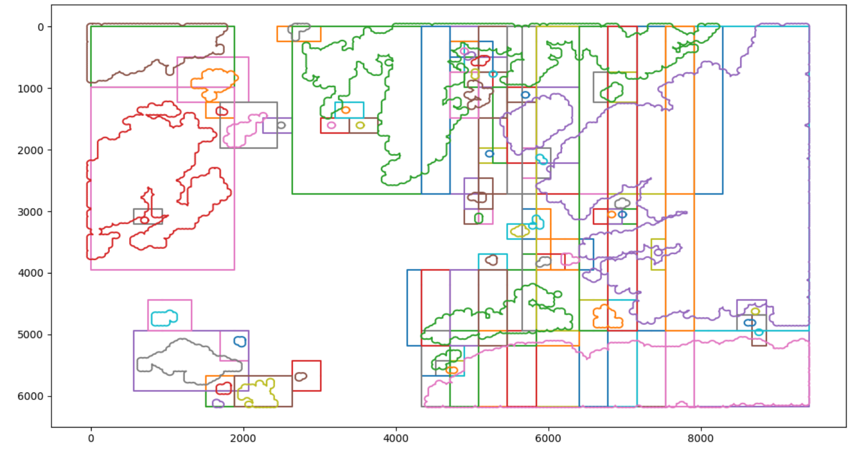


Figure 4.7: The land objects are inside slightly bigger bounding boxes.

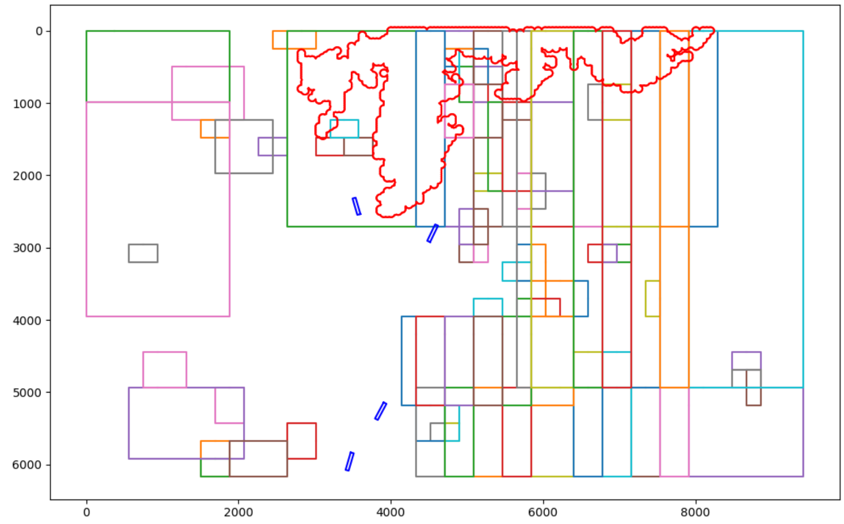


Figure 4.8: What the ship sees of the map. Only the closest ships know about land.

4.6 Agents

An agent is a ship with artificial intelligence. An agent decides which actions a ship should take in determined situations. An agent can follow simple orders or be trained by trial and error to handle more complex situations [22]. The simulator is built with training of autonomous ships in mind and supports saving the training results as weights. Weights can be distributed to various ships and allows them all use the same agent logic.

4.6.1 Existing Agents

One of the main purposes of this simulator is to train and develop agents to exceed in the tasks they are designed for. This simulator was developed in conjunction with Sebastian Penttinen who worked on training the agents and Ivan Porres as the project leader. Sebastians goal was to train agents to follow the Colreg rules a.k.a. to avoid collision with other ships while following the ship traffic laws. There are several ways an agent can be implemented. Statheros et al. [43] gives some examples on implementation strategies for avoidance navigation. The autonomous ship navigation can be divided into collision avoidance or track-keeping. The simulator supports varying tactics of avoidance. The agents Sebastian implemented uses the Deep Q-Network DQN [44]. DQN trains by running the same situation over and over again while determining which action gives the best results. Each action is given an amount of points as an reward and the network tries to increase the reward as much as possible. The simulator supports the training of several scenarios in one training session. Sebastian used this to train the ships with correct behavior when a ship passes from the right and left, when passing a ship and when a ship is sailing right at you. By having the simulator switch between different scenarios randomly it avoids over specializing on one scenario. With enough training the agents should be able to handle each situation with grace.

4.6.2 Creating new Agents

To create a new agent the user has to implement it in the codebase. An agent does not have to be trained and can simply do a designated action during each time-step. The user has to create a new class in the agents folder from which the configuration file knows to look for it. The configuration file will find the new agent by looking in the agent field for a name written as: "$NameOfAgentFile.NameOfAgentClass$".

The agent classes has some methods which are mandatory. The class needs the $load\_from\_file$- and $step$ methods to function. The $load\_from\_file$ method gives the user a place to load files for the agent to use. The $step$ method determines what actions the agent takes each time-step during a simulation. Both of these mandatory methods can be left empty by just calling *pass* in them, but they have to exist for the agent to work. The $step$ method takes two arguments: The ship which the agent is controlling and the environment. Both of these can be used when creating the actions the agent can make. There are two other methods which can help out with creating most agents. The __*init*__ method lets the user decide which actions should be made before starting a simulation. A great place when the user needs to prepare configurations or initialize variables and does not want them to run each step during a simulation. The __*str*__ method lets the user name the agent for what it is referred to during a simulation. This is done by returning the name you want the agent to have. Besides these methods, the user has to write their own methods to achieve their desired agent. For instance, if a user would want to create an agent that does random actions, he could have each step run a method which takes the ship as an argument and picks a random ship action from a list.

4.6.3 Training Agents

A complex agent which uses machine-learning to decide its actions, could use some of the following methods. The simulator has implementations of agents using the DQN- and the DDPG network [44][45] but other algorithms can be implemented as well. When the user wants to train the agent the *train* method is used. The *train* method takes an argument of the amount of steps the agent should be trained upon. The user can add the model on which the agent should be trained upon. The *train* method runs the amount of steps which were inputted and saves the results of the training in weights on a file.

The program will import the agent and execute the *train* method. The training is done via the console. To train an agent the user has to write *python train.py* − *p* ”*path*” ”*steps*” in the project folders terminal, where "*path*" is the path for the scenario for which the agent is training on and ”*steps*” is the amount of steps that should be trained. The simulator will start the training of the agent and continue for as many steps as given as an argument. The scenario that is trained upon has to have at least one of the ships assigned as an agent. This agent will be the ship that is being trained. The training is not rendered and is run several times faster than a rendered simulation.

4.7 Simulation

When the simulator is run, it starts by opening the user interface and importing the settings from the configuration files. The simulator makes the map and inserts the ships on it. A timer is started to keep track of the real time for the ships. A simulation is started by writing $python\ simulate.py\ -p$ "*path*" in the project folders terminal, where "*path*" is the path for the scenario's configuration file.

4.7.1 Steps

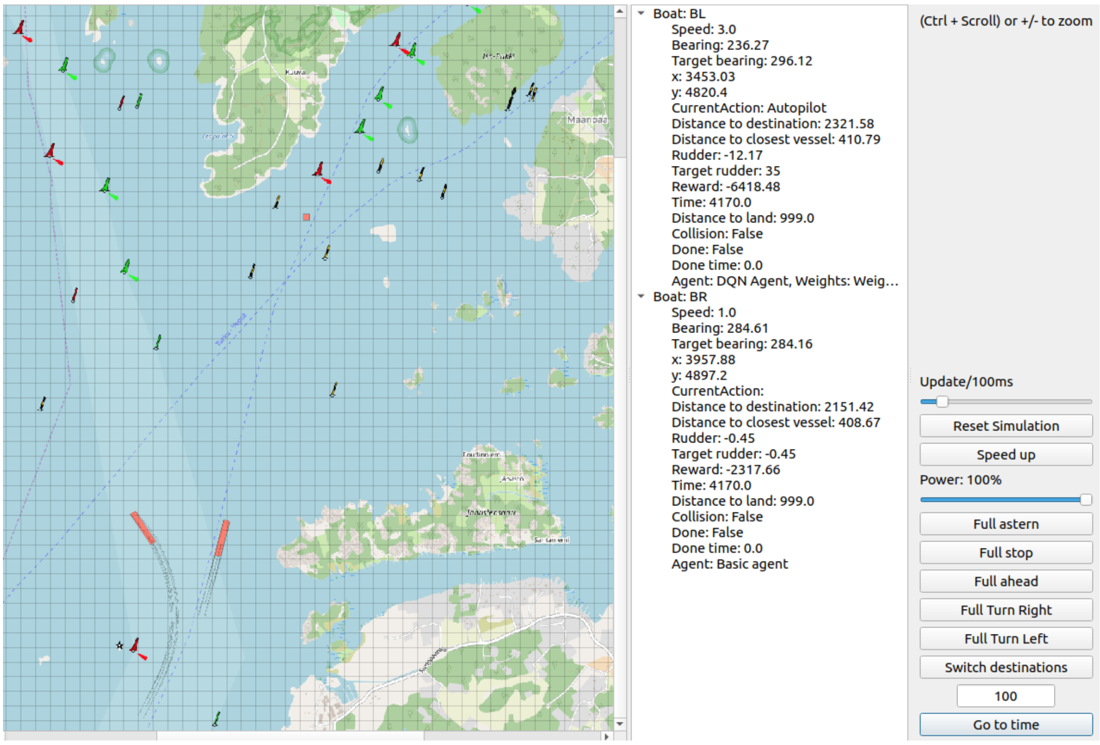
The time in the simulator is divided into steps. Each step is one second long. There are several methods which are triggered from each step. All of the agents determine the next action, ships calculate the physics for the current variables, the map displays the location of the ships, the log file saves the data and the simulator checks the state and distance for each object.

The world has a timer that counts to a designated amount, restarts, and starts counting again. Each time the timer restarts a step is triggered and the world will update. Having the timer reset after a count of 1000 milliseconds will result in a realtime simulation as the simulation counts in milliseconds. While the default setting is to wait for the timer to count to one second, the amount the timer has to count to before restarting can be altered. Changing the reset value is essentially changing the acceleration of the simulator. Having the timer reset at 100 milliseconds instead of 1000 causes the simulator to think a second has passed while to the user only a tenth of it has passed. After a second the simulation will be 10 seconds ahead of real time. The fastest time is 1000 times faster than realtime with a reset period of 1. This can be controlled by a speedup slider that has a minimum value of one and a maximum of 1000 that can freely be changed.

Each ship is updated each step. If a ship is an agent it will calculate its next move based on its configurations. Then if the ship is not in the goal or in a collision it will try to adjust its bearing towards its goal if the autopilot is online. After acquiring the new targets for speed and bearing the ship will try to reach these by adjusting its parameters. The motions of the ship will be updated with its current values and targets. These are explained earlier in Chapter 4. The simulator will then check the ship's distance to land and other ships and if a collision has happened. An collision can be set by having the ship move too close another object according to a threshold or when the ship actually touch the other object. The ship's information is then projected to the information panel and saved in to the log file.

4.7.2 User Interface

The user interface for the simulator has the map at the left side of the window, with a information panel on the right side of it followed by a list of buttons and slider at the far right. The user interface is made in pyside2, which is a python branch of the common QT interface tools.



```
▾ Boat: BL
    Speed: 3.0
    Bearing: 236.27
    Target bearing: 296.12
    x: 3453.03
    y: 4820.4
    CurrentAction: Autopilot
    Distance to destination: 2321.58
    Distance to closest vessel: 410.79
    Rudder: -12.17
    Target rudder: 35
    Reward: -6418.48
    Time: 4170.0
    Distance to land: 999.0
    Collision: False
    Done: False
    Done time: 0.0
    Agent: DQN Agent, Weights: Weig...
▾ Boat: BR
    Speed: 1.0
    Bearing: 284.61
    Target bearing: 284.16
    x: 3957.88
    y: 4897.2
    CurrentAction:
    Distance to destination: 2151.42
    Distance to closest vessel: 408.67
    Rudder: -0.45
    Target rudder: -0.45
    Reward: -2317.66
    Time: 4170.0
    Distance to land: 999.0
    Collision: False
    Done: False
    Done time: 0.0
    Agent: Basic agent
```

Figure 4.9: The whole user interface.

The map uses the path from the configuration files if a path to an image is provided. Otherwise an empty map will be displayed. The ships are salmon colored rectangles for easy recognition and with the correct dimensions which were imported from the configuration files. The ships pivot at one third of the length to resemble real ship behavior. A wake is made by drawing a dotted line from the rudder of the ship and from each corner. The map can be zoomed in and out by pressing the minus and plus keys on the keyboard, or if preferred, the Ctrl-key and using the mouse wheel.

The information panel has the current data for each ship. The information is built using a tree model and each root in the tree can be closed or opened. Clicking on a ship's information takes the map window to that ship to easily recognize which ship's information the user is following. The data presented is the following: Speed, Bearing, Position, Agent action, Distance to destination, Distance to closest ship and land, rudder angle and the state of the ship.

Lastly we have the input panel. Here we have different buttons and sliders that interact with the simulator. A speed slider that speeds up or slows the simulator time is at the top of the panel. The slider can go from a real time speed where one second is one second in the simulator and up to one second is on millisecond in the simulator. That is equivalent to 1000 times faster than real time. A button that toggles the time to 10 and 100 times as fast also exist. The reset button resets the simulator to its starting position. This is done by saving a copy of the original position of the ships when they are imported from the configuration files. The ships are then simply given the starting positions at the next step instead of a true reset of the simulation. A text box in which the user can input a time and a button which when pressed puts the ships at the position of that time can be found at the bottom of the panel. This is done by having a log file that saves each ships' position and other values for each time step. The button then takes those positions and values for the requested time and places the ships at those positions. The remaining buttons interact with the ships directly. These are probably not going to be in the final version of the simulator but are handy in testing the ships' maneuvering abilities.

4.7.3 Autopilot

The ships follow their autopilot if is turned on in the configuration files and the agents do not interfere. The purpose of the autopilot is to change the heading of the ship towards its goal [8]. This is done by measuring the angle between the ship and the goal and changing the rudder accordingly. The autopilot knows the angle of the ship to the goal and the heading of the ship. All it has to do is have them be the same. The heading will often overcompensate and change direction as the rudder is not instantaneous and changes with a delay and causing a small oscillating movement. This extra movement can be improved upon as unnecessary movement causes more fuel and time loss as well as rudder wear [8].

While the autopilot is only used for the ships to go from the starting point to the destination, it could also be used multiple times to replace the destination with a new one when it is reached. While the autopilot used in the simulator is quite capable it is still simple and could be improved such as with the model proposed by Khaled [46].

5. Evaluation

To confirm that the simulator works as a simulation of the real-world some verification is needed. This paper provides the results to tests where the simulation is tested in various aspects and if agents can be trained properly using the simulator. The various aspects a simulator can be evaluated on that was mentioned earlier in Chapter 2 will also be reviewed.

5.1 Reproducibility

The simulator should always strive to yield the same results independent of how many times or when a simulation is run. This can be tested by running the same simulation repeatedly or by pressing the reset button multiple times to reset the position and values of the agents.

As the simulator has no outer forces at this stage of development, the randomness is essentially eliminated from the simulator at this moment. While the simulator does not have any random aspects, the agents could be trained to behave randomly on each run of a scenario. This is something the designer of the machine learning algorithm must correct themselves if it is not the expected behavior of the agent. Each time the simulation is reset the simulation will reset the ships to their initial positions as stated by the configuration file. They will then produce the same results as they would do before resetting. Changing the weights the agent operates on will change the result as would be the case if some of the ships' configurations would be changed. No attribute is randomized in the simulator and thus no random seeds are needed. The simulation is also run locally so no packet loss or other problems that could occur on a cloud-based solutions can happen.

5.2 Modularity

While the simulator currently sports a limited amount of ways the agents can sense the world the application is built in a way to easily implement new features on top of the ships. New features can be added to the configuration files when incorporated as Boolean flags to turn them on and off. The simulator lets the agents sense the world in different ways using different sensors that resemble their real-world counterparts. A camera that sees in front of the ship is implemented by having a cone formed area that works as the line of sight for the ship. A lidar that sees the surroundings of the ships can be implemented by drawing a circle around the ship as the area the ship can see. The default setting is to have the agent know the position and conditions of all the ships around them resembling the data AIS would provide a real ship. Automatic identification system (AIS) is an automatic tracking system used to track maritime vessel traffic [47]. AIS provides the heading, speed and position of the ships which are the same parameter used in with our agents to perform avoidance maneuver. All of the simulators sensor counterparts will not work with real-life sensors out of the box as different sensors work in different ways, but by having the possibility to configure the application further it can always be changed for the present needs.

The simulator supports both big and small ships as seen in the accuracy section up ahead. The simulator only support ships using rudders and propellers to sail as our physics model is based on them. Sailing boats and rowing boats for example are not supported in the current version of the simulator.

5.3 Parallel Simulation

The simulator does not run on parallel threads but is capable of simulating multiple ships at the same time. Multiple agents can be active in the same scenario. As each time unit is split into steps where the agents calculate their next move the simulator can have all the agents calculate their moves before moving on to the next step. The time steps are small enough to essentially be considered a continuous approach. This leads to problems if the agents need to calculate too long and the simulator needs to wait for them to finish before moving on. This will slow down the simulator if the simulator is using accelerated update speeds. The simulation will still be reproducible and will give the same results as earlier but will run with a slower update speed. Normal calculations, such as retrieving the information of the closest ship, are not a problem for the performance but having to wait for an API or calculating for each pixel on the map could cause issues which then would cause slowdowns. The simulator is not capable of training or simulating multiple scenarios simultaneously.

Having the capability of running with parallel threads would give the simulator a performance boost for larger scenarios. The calculations could be divided to different threads in each step for faster calculations. This is a possible future improvement on the simulator.

5.4 Headless Simulation

The simulation trains the agents without rendering with enormous speed. As agents need to be trained thousands of times to learn the best solution to each scenario this will still take quite some time but is multiple times faster than if the training would be rendered. The simulation itself cannot be run without rendering, but the simulation is meant to be the result of the training. The agents are running the same simulation in their training which is run headless.

5.5 Motion accuracy

The end goal for the simulator is to train navigational algorithms that would later be used in real ships. To test if the physics are accurate to the real world, The ship motion will be compared to real ship data from a oil tanker pilot card (appendix A.1). The information acquired from the ships pilot card are adjusted to fit the parameters from the article by Ueng et.al [1].

The ships will be tested in three aspects; the Turning Circle Maneuver, the stopping test and the zigzag maneuver. These tests are common trials to measure the information that is put on the ship's pilot cards. These tests can be costly in real-life and need special environments [48]. These tests must be made in deep and calm waters without other traffic. No waves or wind may disrupt these tests. The simulator considers the

Table 5.1: A table of the different aspects of the ships.

	Oil tanker	Lifeboat
Length	235 m	16 m
Mass	64830 ton	36 ton
Rudder area	17.5 m	0.4 m
Rudder movement time	360 s	9.8 s
Rudder coefficient	1.1	0.35
Propeller diameter	7 m	0.95 m
Maximum propeller revolution	103.7 rpm	1049.7 rpm
Minimum propeller revolution	-88.2 rpm	-1050 rpm
Propeller thrust coefficient	0.882	0.85
Maximum speed	8.03 m/s	9.26 m/s
Minimum speed	-3.19 m/s	-4.52 m/s

ships to move in deep calm waters and does not take into consideration shallow water, waves, wind or currents. Sailing in shallow water would result in wildly different readings [49].

To evaluate the accuracy of the simulator I set up different scenarios for the ships to sail in. Each pixel represents 1 meter. Each square is 100 pixels wide and tall which represents 100-meter sized squares. These are used for distance calculations in the tests.

5.5.1 Turning Circle Maneuver

The turning test have a ship turn in a circle. First, the ship must move in a straight line with a constant speed. Then the rudder is turned to the maximum angle. This will start the first phase. The ship will start turning and when it has reached a 90-degree angle from the original heading it has started to reach its constant speeds. When the ship has reached an acceleration of zero and moves in a constant speed it has reached its steady turning circle aka phase three [48] . When the ship continues turning it will make a perfect circle from this point on. The test measures the time it takes to finish the circle and the measurements of the circle. The test measures the ship's advance, transfer and tactical diameter. The advance is the distance in the x-axis the ship has moved forward. The distance is counted to the utmost point of the circle where the ship is at a 90-degrees angle of its original heading. The transfer is the amount the ship has moved in the y-axis to this same 90-degrees point. Tactical diameter is the distance

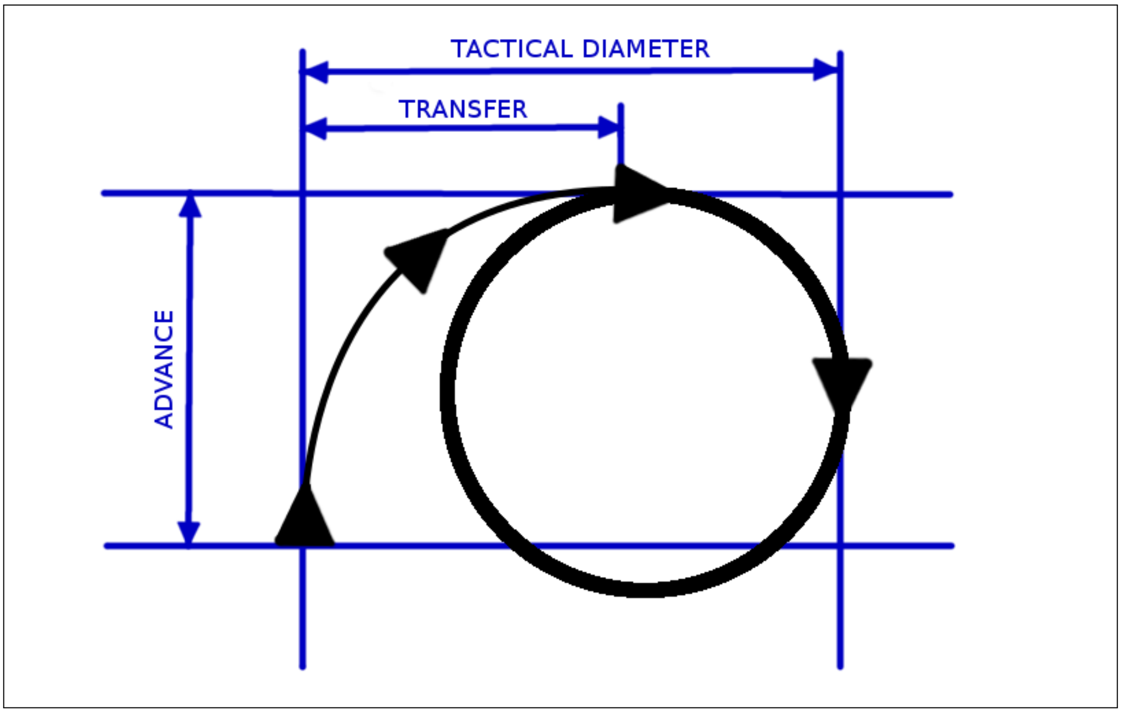


Figure 5.1: A turning circle diagram with all of the measured aspects [48].

from the starting x-axis to the point where the ship is at 180 degrees difference of the original heading.

The results will be compared to the pilot cards which always have the time it takes for a ship to make a full circle and the measurements of the circle. The test is made to confirm the physics affecting the turning of the ship. The rudder is the focus of this test.

5.5.2 Turning Circle Maneuver results

The circle test has the Oil tanker start by having a bearing of 0 degrees, maximum speed, and the rudder at 0 degrees. When the simulation starts the target angle of the rudder is set to 35 degrees to start turning the ship. The speed is lowered by the turn to the rate that is stated on the Oil tankers pilot card. When the ship has turned and moved past its starting positions x coordinate it has reached the goal. The time it took to sail a full circle is then compared to the time stated on the pilot card. The same is done with the size of the sailed circle. The simulator assumes to that the ships are sailing in deep water and uses the corresponding data from the pilot card.

The time it took to perform the test is 708 seconds for the Oil tanker, which is the

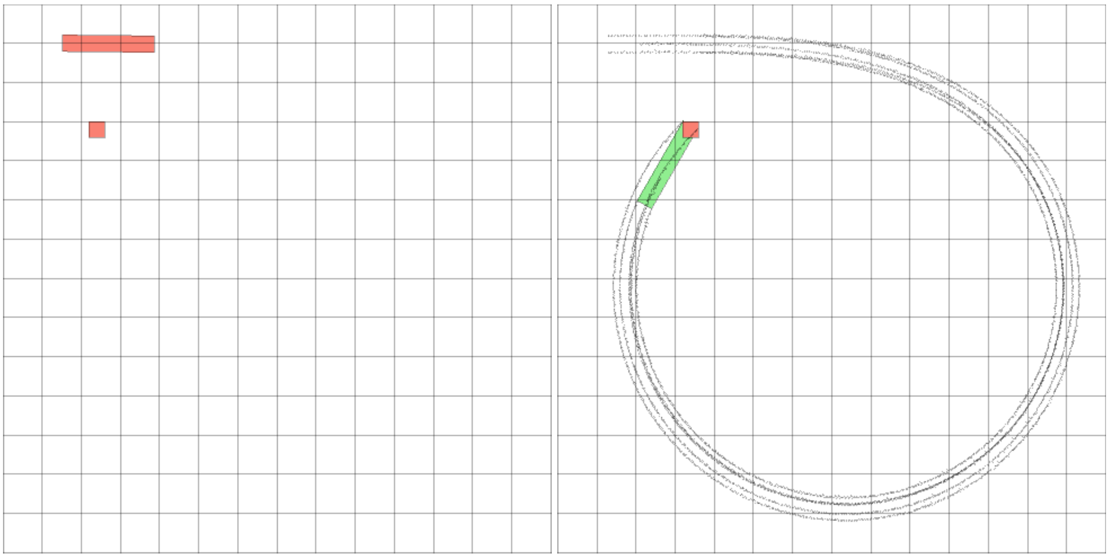


Figure 5.2: Start and end of the Turning Circle Maneuver.

same as in the pilot card of the Oil tanker in appendix A.1. The results can be observed from Figure 5.2. The tactical diameter is 1180 meters which is 2 a percent error from the value of 1200 meters on the pilot card. The advance is 850m which also is a 2 percent error of the value on the pilot card. The transfer is 500 meters which is an error of less than 1 percent of the value on the pilot card.

The results are acceptable with only a couple of percent error from the values on the pilot card. According to these results the simulator can be referred as accurate for testing of navigational algorithms.

5.5.3 Stopping ability methodology

The second test is to measure the time and distance it takes for a ship to stop after going from full ahead to full astern. Full ahead is when the ship moves with the maximum velocity forward. Full astern is when the ship travels in its maximum speed backwards. In other words: the test is to measure the time and distance it takes for a ship stop after going from full forward speed to full backwards speed. This test checks if the physics of changing speed and slowing down works. The focus in this test lies in the propellers and the power of the engine. We can compare these results with the information on the pilot card of the oil tanker in appendix A.1.

5.5.4 Stopping ability results

The test involves having the ship brake from the fastest speed by turning the power on reverse. We can measure the distance the ship travels and the time it takes when the ship reaches a velocity of zero.

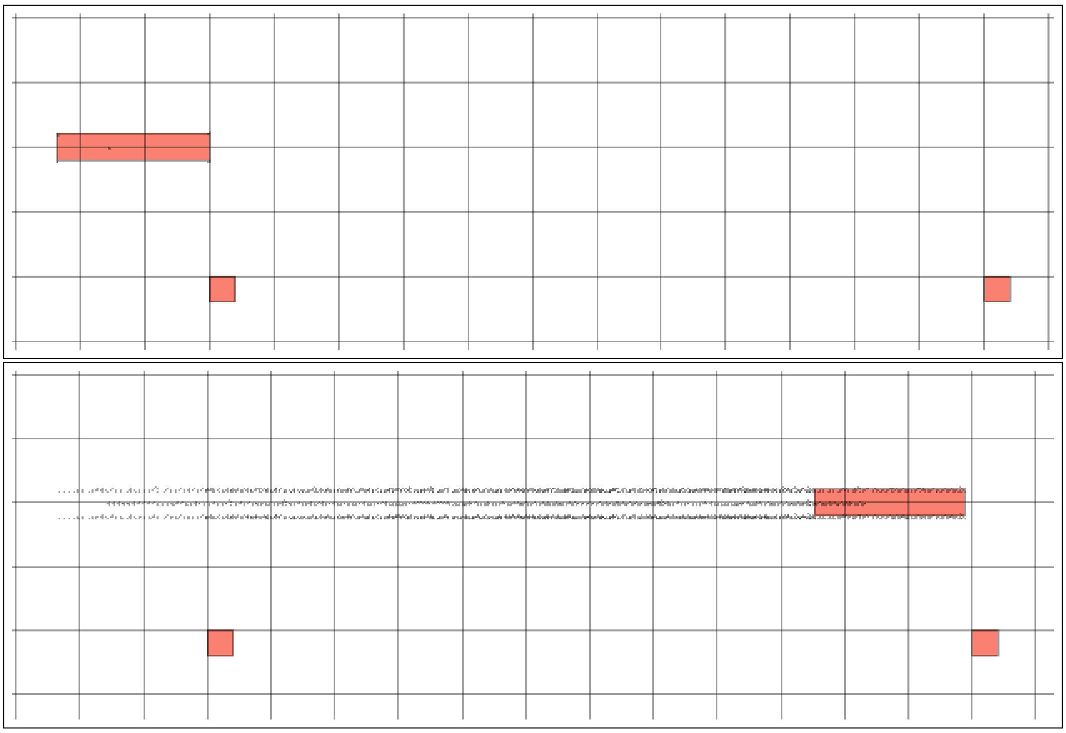


Figure 5.3: The Stopping ability test.

The Pilot card states that the oil tanker will stop after 355 seconds and reach 1193 meters. As one can see from Figure 5.3, the ship stopped at the correct distance. However the time it took to stop was 377 seconds which is an error of 6 percent. The results can still be deemed acceptable as the propeller coefficients, used to calculate the oil-tankers surge motion, are approximated.

5.5.5 Zigzag maneuver

The third test is the zigzag maneuver test also known as Z-maneuver or the Kempf Maneuver [50]. This test begins by having the ships move forward in a straight line and put the rudder at 20 degrees. This is called the ’First Execute’. The ship starts to turn and will reach a 20 degrees angle of difference from the original heading. After the ship has reached 20 degrees change it switches the rudder to -20 degrees and continue until the ship has reached 340 degrees of bearing. This test measures the turning of the ships and if it looks natural. In a real-life scenario rapid turning may be needed in rough weather or a heavily trafficked area. The focus in this test is in the rudder and the time it takes for a rudder to switch the angle of the ship.

5.5.6 Zigzag maneuver results

The zigzag test measures the movement of the ships while changing the rudder multiple times.

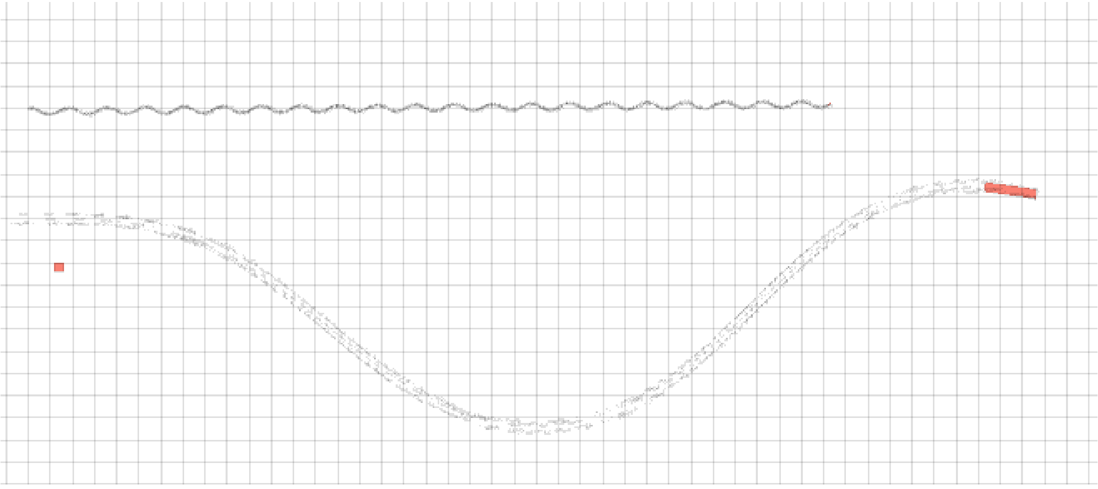


Figure 5.4: The zigzag test

At the top of the simulation we have a lifeboat with small and fast changes. At the bottom the oil tanker sails with long and slow changes of the heading. Looking at these results we can conclude that different ships move differently as they should and that they seem to move with natural movements without irregularities. As the pilot cards do not have any information that we could compare our results to we must be satisfied with having the ships' movement at least look natural.

5.5.7 Other maneuver tests

While we performed the before mentioned tests to verify the movements of the ships in our simulator, there are other tests that could also have been performed. These tests are for real-life ships to test if they are viable for travel, but we can use them to find any glaring problems in the physics of the simulator. Other ways to verify the results also exists such as comparing the data from a simulation with real ship's travel data.

The spiral maneuver is a test where a ship will increase the rudder angle at varying points in a circle making the ship move in a spiral-like movement. This test checks if the ship can handle sharp turns and continuous turns at a sharper angle. A reverse spiral maneuver is also a test where a ship makes less and less sharp turns and in the end sails out of a spiral. This test is similar to the original spiral test.

Another test is the pull-out test where it is tested if the ship behaves the same while turning left and right. The ship starts by being in a turn and after a while sets the rudder at 0 degrees. This will steady the ship and it will start moving forward. The test tests if the ship reaches the same spot if the rudder is released at the same spot both from starboard and port. An unstable result will have the ship be at different positions after the release. [50]

5.6 Simulator speedup

The simulator can run with an accelerated speed when running a simulation. The speedup can be set to update the simulation between once a millisecond and once a second. The training of the simulation can be run even faster as it uses headless simulation.

5.6.1 Methodology

As the simulator can be sped up to accelerate the time the simulation is performed in some proof of the accuracy is needed. To test the limits of the speedup function the earlier circle maneuver test is run in accelerated time with one to nine ships. The real-time time it takes for an oil-tanker to perform the circle maneuver is 708 seconds. The test will measure the simulated time and the real-time it takes to perform the maneuver. The application is set to write the time of completion of the test when the ship has reached its destination. These test result will be affected by the system the simulator is operating on. The tests were done on a computer with 16 Gb RAM and a 3.3 GHz quad-core processor.

5.6.2 Results

The test provided interesting results. The best result was a speedup of 540 times faster than real-time when simulating a single ship as can be seen in Figure 5.5. The simulated time provided the exact result on each accelerated speed and with any amount of ships. The time to perform the maneuver was, however, heavily dependent on the amount of ships in the test. The speed drop resembles a power function as can be seen by the dotted line in Figure 5.5. The simulator can handle up to eight ships without dropping under 100-times accelerated speed. As the results are system dependent, a more powerful computer would provide more speedup.

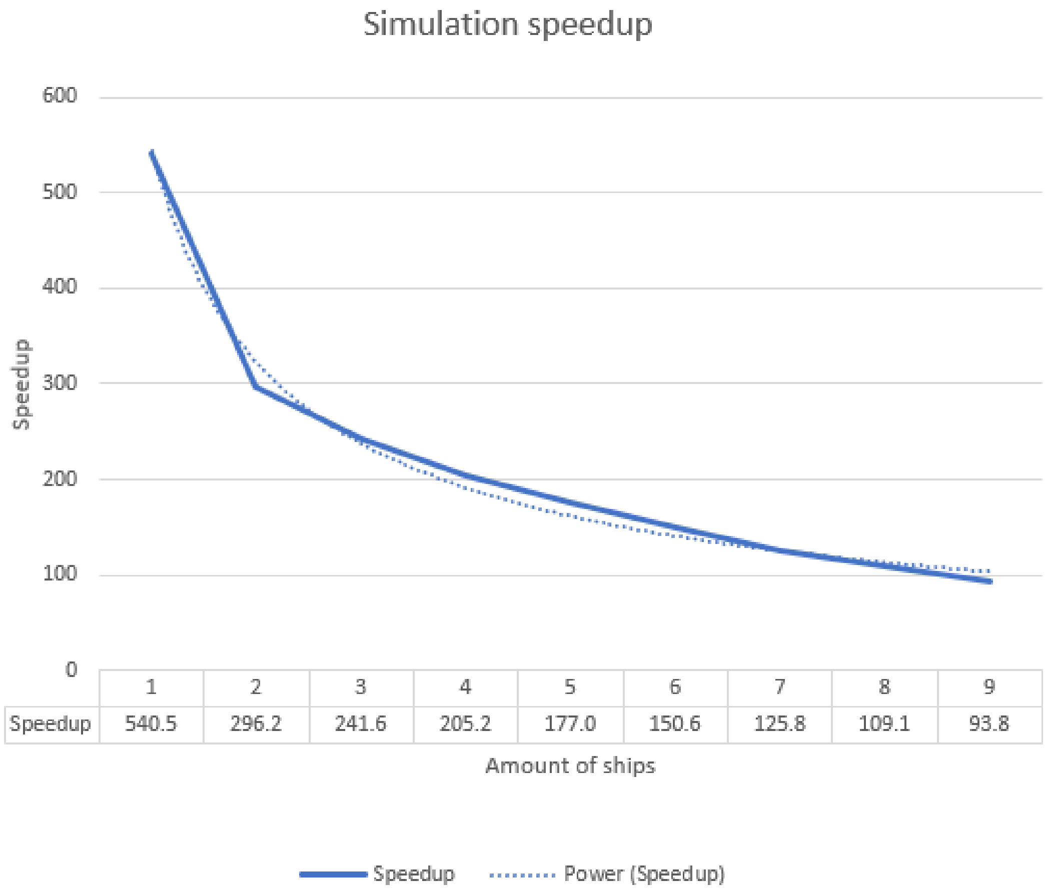


Figure 5.5: The speedup graph.

5.7 Agent evaluation

The simulator is mainly made to train agents to navigate ships. To test if the simulator works as intended, an agent who has achieved this goal is needed. To test this I will observe the results Sebastian Penttinen [51] has achieved with his agents using this simulator. Penttinen trained his agents with different scenarios to train them to follow the Colreg rules of ship traffic. Others such as Zaccone et al.[52] have also trained agents which follows the Colreg rules with varying results. He split the tests into four tests to train the agents for different scenarios. The first scenario is the scenario of crossing another ship from the left. In this scenario, a ship that is following the Colreg rules would avoid the ship coming from the left by sailing behind the incoming ship. The second test is similar to the first one as we have the same situation but with a ship coming from the right. The third test is when two ships are moving head on against each other. Both ships have the duty to avoid the other by turning to the right. The last test is overtaking where a ship that is traveling faster passes another ship from the left.

5.7.1 Overtaking test results

In the overtaking test, a faster ship wants to pass a slower one without colliding. The ship should overtake the ship in front of it from the left. The agent managed to do this successfully as can be seen in Figure 5.6. The ship the agent is controlling is an Oil tanker which has a slow turnrate. The ship has to do a big maneuvers to return on course. Reacting earlier on close by ship could provide a smaller avoidance maneuver.

5.7.2 Head on test results

The Head on test has the ships sailing toward each other. At least one of the ships must turn to the right to avoid collision. The agent succeeded in this test by barely avoiding the other ship. The settings for the agents could be adjusted to avoid sailing too close to another ship.

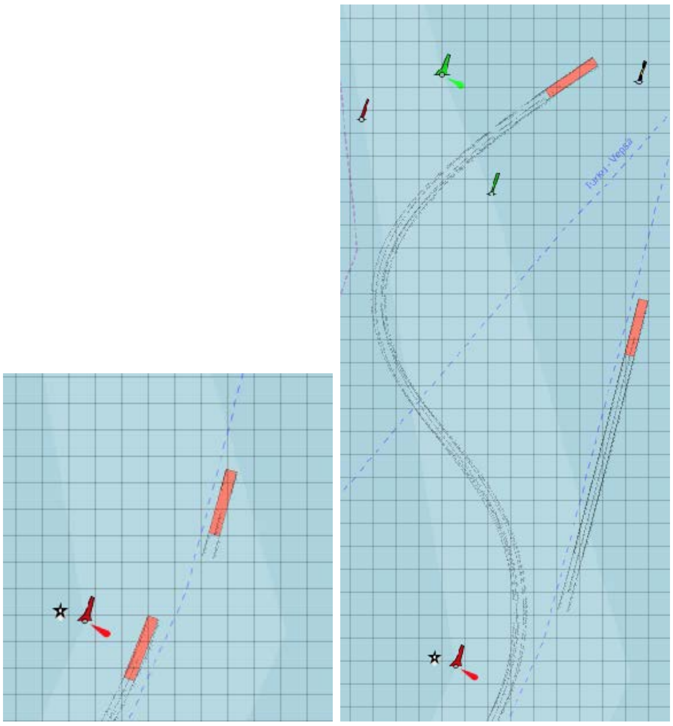


Figure 5.6: The overtaking test

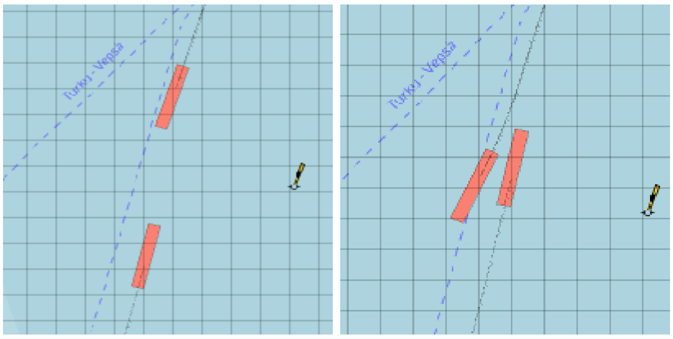


Figure 5.7: The head on test

5.7.3 Crossing test results

The Crossing tests have the agents avoid the other ship by sailing behind or in front of a ship depending on which side it approaches. The ship sailing from the right has the right of way, and the ship on its left has to sail behind it. The agents managed to succeed in both of the scenarios. While both crossing tests have the ship turning right, it is still a different kind of scenario as the agent calculates the differences in angle between the ships.

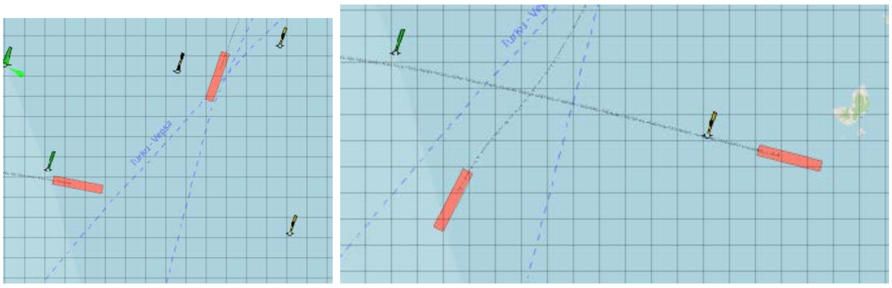


Figure 5.8: Crossing from left

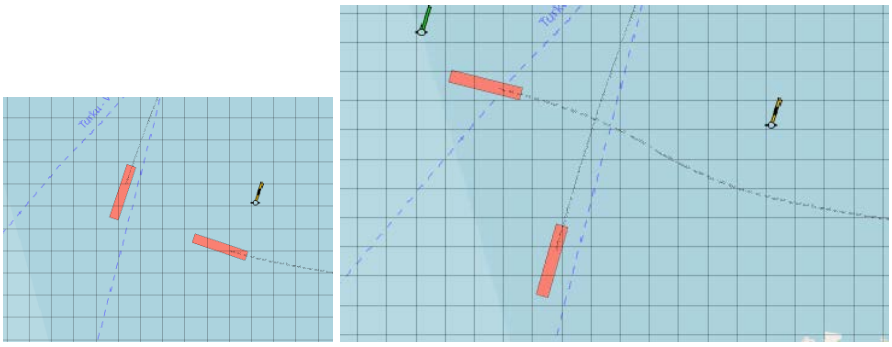


Figure 5.9: Crossing from right

6. Conclusion

The simulator was created to fulfill the need of a ship focused simulator for testing and designing navigational algorithms. The simulator had goals and requirements named in Chapters 1 and 2 which it needed to fulfill to be an acceptable product. The goals and requirements have been evaluated in Chapter 5 with acceptable results. The simulator is capable to train and evaluate the results of different navigational algorithms. It is able to run in accelerated time and is capable to simulate multiple agents' behavior. The simulator is ready to be used in further testing for academic purposes.

Several improvements can be made to the simulator. The simulator is operating in a three-degrees-of-freedom while a six-degrees-of-freedom approach would be provide more realistic results. This would also enable the incorporation of the ROS framework to import more realistic sensors to the simulator when they arrive. Alongside a three-dimensional approach weather conditions could be added for a more realistic approach.

The simulator does not support enormous maps as the loading times when starting the application would take a long time as it needs to calculate all the pixels that is considered land. The process could be accelerated by having the simulator support parallel processes and divide the work into different threads. This approach is something that also could be improved upon with a more dynamic solution. The rtree method to filter lands would not be needed if a solution with better performance would be incorporated.

In conclusion, the simulator has reached the goals set for it and is a viable product to test and evaluate navigational algorithms for ships.

7. Swedish abstract - Svensk sammanfattning

7.1 Inledning

Inom fartygsindustrin håller självstyrda fartyg på att ta stora steg framåt. Ett obemannat fartyg har inte samma krav som en traditionell, bemannat fartyg som styrs av manskap. Företag kan i framtiden spara pengar genom att använda självstyrda fartyg, eftersom fartyg då kan ta mera last då manskapen inte behöver utrymmen. För att ett fartyg ska bli självstyrd måste det ges en artificiell intelligens (AI). Denna AI måste sedan tränas på något sätt. Att använda en simulator är ett effektivt sätt att träna fartygets AI i en billigare och säkrare omgivning, eftersom det krävs många permutationer för att AI:t ska lära sig hur en situation bör hanteras. Användningen av en simulator möjliggör en snabbare träning tack vare den accelererade tiden i simulatorn. Simulatorer som fokuserar på träning av navigationsalgoritmer är sällsynta. Det finns flera simulatorer som fokuserar på realistisk hantering av robotar på land som rör sig med fötter eller på hjul, men det finns inte många simulatorer som hanterar fartygsfysik. På grund av att det inte finns en lämplig simulator har man nu utvecklat en ny som uppfyller många av de egenskaper som efterfrågas på marknaden. Denna avhandling går igenom egenskaperna hos denna nya simulator.

7.1.1 Krav

Målet med detta projekt är att skapa en simulator som kan träna, utveckla och utvärdera självstyrda fartygs navigationsalgoritmer. I början av projektet ställdes det upp några mål som simulatorn måste uppfylla. Simulatorn ska klara av att simulera flera fartyg samtidigt, av dessa kan några vara styrda av en AI. Dessa AI:er bör kunna känna in världen omkring sig, fatta beslut och vidta åtgärder. Simulatorn ska samla in information om de simulerade scenarierna såsom positionen av fartygen, distansen mellan fartygen och land, samt tiden det tog att köra scenariot. En av de viktigaste egenskaperna för simulatorn är att den bör fungera i accelererad tid. Detta innebär att användaren kan snabba upp tiden i simulatorn och således få mer träning gjort än vad man kan uppnå i realtid. Simulatorns utseende är däremot inte en prioritet, en enkel tvådimensionell vy uppifrån räcker för detta projekt. Simulatorn bör hantera land, vatten och båtfysik. Den skall också ha ett API (applikationsprogrammeringsgränssnitt) för att träna AI:n och ett sätt för användaren att mata in scenarier med land och fartyg. Simulatorn kan vara baserad på en existerande simulator eller skapas från början.

För att betygsätta en simulator i allmänhet kan man kontrollera om den uppfyller följande krav. En simulator bör alltid uppge samma resultat oberoende hur många gånger man upprepar samma scenario. Några orsaker till att simulatorn kan ge olika resultat är att simulatorn använder slumpmässiga tal eller att simulatorn beter sig olika med olika system. Simulatorn bör vara uppbyggd modulärt och ha möjlighet att senare inkludera nya sensorer och båttyper. Möjligheten att välja vilka sensorer ett fartyg kan använda via en konfigurationsfil rekommenderas. Resultaten som simulatorn ger ut bör överensstämma med verkligheten. Om simulatorn inte klarar av att simulera verklig fysik är simulatorn oanvändbar för akademiskt bruk. Parallell simulation är en användbar egenskap för en simulator som måste göra många samtidiga beräkningar. Accelererad simulation som tidigare nämnts, gör simulatorn mer effektiv då mer träning kan utföras på mindre tid. Om simulatorn kan träna utan ett grafiskt användargränssnitt kan simulatorn träna även snabbare då träningen inte behöver ritas upp.

7.2 Design

Simulatorn som har utvecklats är skapad från början och baserar sig inte på någon annan simulator. Programmet är skrivet i python och använder pyside2 som användargränssnitt. Simulatorns grafiska gränssnitt är tvådimensionellt med en vy uppifrån.

7.2.1 Scenarier

Simulatorn låter användaren mata in en så kallad scenariofil med information om fartygen, deras positioner och deras omgivning. Till fartygens variabler hör bl.a. deras storlek, massa, motorstyrka samt information om deras propeller och roder. Omgivningen kan tas från en kartbild av en karta eller lämnas tom med möjligheten att inkludera egna hinder. Fartygen kan ges ett mål de ska nå och information om huruvida de ska använda autopilot för att nå det. Autopiloten jämför vinkeln mellan målet och fartyget och ändrar rodret tills kursen pekar mot målet.

Användaren kan genom scenariofilerna bestämma ifall fartygen ska vara kontrollerade av en AI eller inte, samt vilken typ av AI ska användas i detta fall. Scenariofilerna används även vid träning av AI:n. Användaren kan bestämma vilket fartyg är tränat och i hurdan miljö träningen har gjorts i.

7.2.2 Fysikmodellen

Fartygen i simulatorn rör sig i tre grader av frihet. Detta innebär att fartygen kan röra sig längs längdaxeln, tväraxeln och rotera runt vertikalaxeln. Fysiken i simulatorn är baserad på Uengs et al. [1] modell. Uengs modell baserar sig på sex grader av frihet, vilket förutom de tre denna simulator använder även inkluderar rörelsen längs vertikalaxeln och rotationerna kring längd- och tväraxeln. Vår modell använder tre grader av frihet för att inte försvåra navigationsalgoritmerna mer än det krävs. De tre icke inkluderade rörelserna kan vid behov läggas till i framtiden. Än så länge stöder simulatorn inte förändringar i väderleken. Den antar också att alla fartyg seglar i djupt och stilla vatten.

7.2.3 Steg i simulatorn

Simulationstiden är uppdelad i en sekund långa steg. Under ett steg sköter simulationen alla beräkningar för varje fartyg och samtidigt bestämmer sig alla intelligenta fartygens AI:n för sin nästa åtgärd och tillståndet för varje fartyg sparas i en fil. Ett steg sker då en timer räknat till 1 000 millisekunder och startar då om. För att accelerera simulationen är det möjligt att sänka detta värde t.ex. till 100 millisekunder, vilket skulle göra simulationen 10 gånger snabbare. Tiden kan accelereras upp till 1 000 gånger snabbare. Simulatorn ger samma resultat även om tiden är accelererad och tidsaccelerationen kan påverkas med ett skjutreglage i programmet.

7.2.4 Träning av AI

Simulatorns huvudfunktion är att träna AI:er med navigationsalgoritmer. Detta görs genom att skriva in i konsolen vilken scenariofil AI:n ska träna på och i hur många iterationer. AI:n kör sedan scenariot flera gånger och väljer det bästa beteendet i varje situation. AI:n får respons av en så kallad prisfunktion, som belönar AI:n för bra beteende. Baserat på informationen ur de tidigare iterationerna försöker AI:n förbättra sitt resultat tills antalet givna iterationer är uppnått, och därefter borde AI:n vara vältränad och dess algoritmer väl justerade. Sebastian Penttinen var min kollega och jobbade med samma projekt, vilket hade som mål att programmera en AI som klarar av att följa de allmänna reglerna för sjötrafiken och testa den tillverkade simulatorn [51].

7.3 Utvärdering

Detta kapitel jämför simulatorns egenskaper med de mål som var givna i början av projektet. Simulatorn ger alltid samma resultat då ett scenario körs, vilket beror på att simulatorn inte har inkluderat slumpmässiga tal i dess beräkningar. Detta kan testas genom att köra samma scenario flera gånger eller genom att trycka på återställningsknappen i användargränssnittet. Simulatorn stöder en enkel tilläggning av nya sensorer genom modulära metoder för fartygsklassen i programmet. Fartygen tolkar omgivningen med hjälp av metoder som motsvarar verkliga sensorer. En kollision, som i verkligheten ger sig tillkänna av stötsensorer, hanteras i simulationen genom att simulatorn reagerar då ett fartygsobjekt träffar ett annat objekt. Fartygen kan se sin omgivning på flera olika sätt, de kan antingen se allt omkring sig eller välja att endast se vissa områden såsom i en kon eller i en cirkel runt omkring sig. Dessa former representeras av olika sensorer som fartyget kan ha, såsom en kamera eller en lidar.

Simulatorn kör på en process tråd och stöder inte parallella processer. Alla beräkningar körs mellan stegen i simulatorn och så länge det inte är för krävande beräkningar klarar simulatorn utan problem av att simulera scenarier. Om simulatorn inte klarar av att köra alla beräkningar mellan varje steg kommer den att köra långsammare. Detta påverkar dock inte slutresultatet. Simulatorn kör all träning via konsolen utan ett grafiskt användargränssnitt, vilket sker på en bråkdel av den tid som en grafisk simulation skulle ta.

För att utvärdera noggrannheten av simulationens fysikmodell har man gjort tester och jämfört dem med riktiga fartygs pilotkort. Pilotkorten innehåller information om fartygens storlek, hastighet och manövertider. Det första testet jämför tiden och storleken av det mönster som formas då ett fartyg kör i en cirkel, här mäter man cirkelns taktiska diameter, förflyttning och förskjutning. Simulationens resultat hade mindre än fem procents felmarginal jämfört med värden från ett existerande pilotkort, och låg således inom de acceptabla gränserna. Det andra testet görs för att se om fartyget klarar av snabba ändringar av rodervinkeln, fartyget kör framåt i ett sicksack-mönster för att se om rörelsen ser naturlig ut. Resultaten gav små och snabba rörelser för en liten båt och stora, långsamma svängningar för en stor oljetanker. Det sista testet jämförde tiden för ett fartyg att stanna upp från full fart i en simulation med tiden skriven i fartygets pilotkort. Slutresultatet låg inom 5 procents felmarginal.

För att simulatorn ska kunna användas till träning av navigationsalgoritmer bör det finnas något bevis på att simulatorn kan träna navigationsalgoritmer. Penttinen använde simulatorn för att åstadkomma en AI som klarar av att få fartyg att undvika kollisioner med varandra genom att följa de allmänna sjöfartstrafikreglerna. Han tränade AI:n att klara av tre olika situationer. AI:n klarade av att passera ett långsammare fartyg, väja till höger då två fartyg höll på att krocka och väja åt rätt håll då ett fartyg närmade sig från sidan. Detta bevisar att simulatorn går att använda för träning av navigationsalgoritmer.

Simulatorn har nått de mål som den blivit given och kan användas för akademisk forskning. Simulatorn är byggd för att möjliggöra enkel vidareutveckling av egenskaper och förbättringar. Möjliga förbättringar är integration av de saknade frihetsgraderna och väderförhållandena såsom vind och vågor. Simulatorn kunde ändras att fungera med parallella processer för att förbättra prestandan vid större scenarier.

List of Figures

References

[1] C.-H. L. Shyh-Kuang Ueng David Lin, “A ship motion simulation system,” 2008.

[2] J. Pereira and R. P. A. Rejaili, *Shipai*, `https://github.com/jmpf2018/ShipAI`, 2018.

[3] L. Kretschmann, H.-C. Burmeister, and C. Jahn, “Analyzing the economic benefit of unmanned autonomous ships: An exploratory cost-comparison between an autonomous and a conventional bulk carrier,” *Research in Transportation Business & Management*, vol. 25, pp. 76–86, 2017.

[4] *Aawa project introduces the project’s first commercial ship operators*, https://www.rolls-royce.com/media/press-releases/2016/pr-12-04-2016-aawa-project-introduces-projects-first-commercial-operators.aspx.

[5] *Maritime unmanned navigation through intelligence in networks*, http://www.unmanned-ship.org/munin/.

[6] M. Caccia, M. Bibuli, R. Bono, and G. Bruzzone, “Basic navigation, guidance and control of an unmanned surface vehicle,” *Autonomous Robots*, vol. 25, no. 4, pp. 349–365, 2008.

[7] M. Ramos, I. B. Utne, J. E. Vinnem, and A. Mosleh, “Accounting for human failure in autonomous ships operations,” *Safety and Reliability-Safe Societies in a Changing World ESREL 2018*, pp. 355–63, 2018.

[8] A. W. Browning, “A mathematical model to simulate small boat behaviour,” *SIMULATION*, vol. 56, no. 5, pp. 329–336, 1991.

[9] D. Ferigo, S. Traversaro, and D. Pucci, *Gym-ignition: Reproducible robotic simulations for reinforcement learning*, 2019. arXiv: `1911.01715 [cs.RO]`.

[10] J. Van Der Wardt, “A graphical simulator for the dynamics of marine autonomous surface robots,” PhD thesis, Universita Di Pisa, 2011.

[11] *Twenty-First Symposium on Naval Hydrodynamics*. National Academies Press, 1997.

[12] M. Paravisi, D. H. Santos, V. Jorge, G. Heck, L. Gonçalves, and A. Amory, “Unmanned surface vehicle simulator with realistic environmental disturbances,” *Sensors*, vol. 19, no. 5, p. 1068, 2019.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: An open-source robot operating system,” in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[14] *Heron usv*, https://clearpathrobotics.com/heron-unmanned-surface-vessel/.

[15] R. Smith, *Open dynamics engine (ode)*, http://www.ode.org/.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[17] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, “Extending the openai gym for robotics: A toolkit for reinforcement learning using ros and gazebo,” *arXiv preprint arXiv:1608.05742*, 2016.

[18] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, IEEE.

[19] N. G. Lopez, Y. L. E. Nuin, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, and R. Kojcev, “Gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo,” Mar. 14, 2019. arXiv: 1903.06278v2 [cs.RO].

[20] *Ignition robotics*, Ignition Robotics. [Online]. Available: https://ignitionrobotics.org.

[21] M. Prats, J. Perez, J. J. Fernandez, and P. J. Sanz, “An open source tool for simulation and supervision of underwater intervention missions,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012.

[22] A. Ray, J. Achiam, and D. Amodei, “Benchmarking Safe Exploration in Deep Reinforcement Learning,” 2019.

[23] *Webots reference manual*, release 4.0.27, Cyberbotics Ltd., https://www.csie.ntu.edu.tw/ b91 2004.

[24] R. Mendonca, P. Santana, F. Marques, A. Lourenco, J. Silva, and J. Barata, "Kelpie: A ROS-based multi-robot simulator for water surface and aerial vehicles," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2013.

[25] *Riverwatch - cooperating robots for monitoring of riverine environments*, http://www.echord

[26] *Transas*, https://www.wartsila.com/transas.

[27] *Oceanic corp*, https://www.oceaniccorp.com/.

[28] D. Sandaruwan, N. Kodikara, R. Rosa, and C. Keppitiyagama, "Modeling and simulation of environmental disturbances for six degrees of freedom ocean surface vehicle," *Sri Lankan Journal of Physics*, vol. 10, 2009.

[29] *Yaml, a human friendly data serialization standard for all programming languages*, https://yaml.org/.

[30] S. Gillies, *The shapely user manual*, 1.7.0, https://shapely.readthedocs.io/en/stable/manual.h 2020.

[31] R. A. Ibrahim and I. M. Grace, "Modeling of ship roll dynamics and its coupling with heave and pitch," *Mathematical Problems in Engineering*, vol. 2010, pp. 1–32, 2010.

[32] N. Khaled and N. Chalhoub, "A dynamic model and a robust controller for a fully-actuated marine surface vessel," *Journal of Vibration and Control*, vol. 17, no. 6, pp. 801–812, 2010.

[33] K. Rawson, *Basic Ship Theory Volume 1*. Butterworth-Heinemann, 2001, ISBN: 0-7506-5396-5.

[34] D. Sandurawan, N. Kodikara, C. Keppitiyagama, and R. Rosa, "A six degrees of freedom ship simulation system for maritime education," *International Journal on Advances in ICT for Emerging Regions (ICTer)*, vol. 3, no. 2, p. 34, 2011.

[35] M. R. Mainal and M. S. Kamil, "Estimation of ship manoeuvring characteristics in the conceptual design stage," 1996.

[36] J.-H. Lee and Y. Kim, “Study on steady flow approximation in turning simulation of ship in waves,” *Ocean Engineering*, vol. 195, p. 106 645, 2020.

[37] A. F. Molland, S. R. Turnock, and A. Hudson Dominic, *Ship Resistance and Propulsion*. Cambridge University Pr., Aug. 1, 2017, ISBN: 1107142067.

[38] R Alte and M. Baur, “Propulsion. handbuch der werften,” *Hansa*, vol. 18, no. S 132, 1986.

[39] J. Liu and R. Hekkenberg, “Sixty years of research on ship rudders: Effects of design choices on rudder performance,” *Ships and Offshore Structures*, vol. 12, no. 4, pp. 495–512, 2016.

[40] D. Lehmann, “Improved propulsion with tuned rudder systems,” in *1st International Conference on Ship Efficiency*, 2007.

[41] R.-Q. Lin, M. Hughes, and T. Smith, “Prediction of ship steering capabilities with a fully nonlinear ship motion model. part 1: Maneuvering in calm water,” *Journal of Marine Science and Technology*, vol. 15, no. 2, pp. 131–142, 2010.

[42] M. Hadjieleftheriou, Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras, “R-trees: A dynamic index structure for spatial searching,” in *Encyclopedia of GIS*, Springer International Publishing, 2017, pp. 1805–1817.

[43] T. Statheros, G. Howells, and K. M. Maier, “Autonomous ship collision avoidance navigation concepts, technologies and techniques,” *The Journal of Navigation*, vol. 61, no. 1, pp. 129–142, 2008.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[45] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” Sep. 9, 2015. arXiv: `1509.02971v6 [cs.LG]`.

[46] N. Khaled and N. G. Chalhoub, "A self-tuning guidance and control system for marine surface vessels," *Nonlinear Dynamics*, vol. 73, no. 1-2, pp. 897–906, 2013.

[47] *Ais, marinetraffic*, https://www.marinetraffic.com/.

[48] S. Ghosh, "Understanding different types of manoeuvres of a vessel," *Marine insight*, 2019.

[49] V. N. Drachev, "Calculating wheel-over point," *Asia-Pacific Journal of Marine Science&Education*, vol. 2, no. 1, pp. 27–46, 2012.

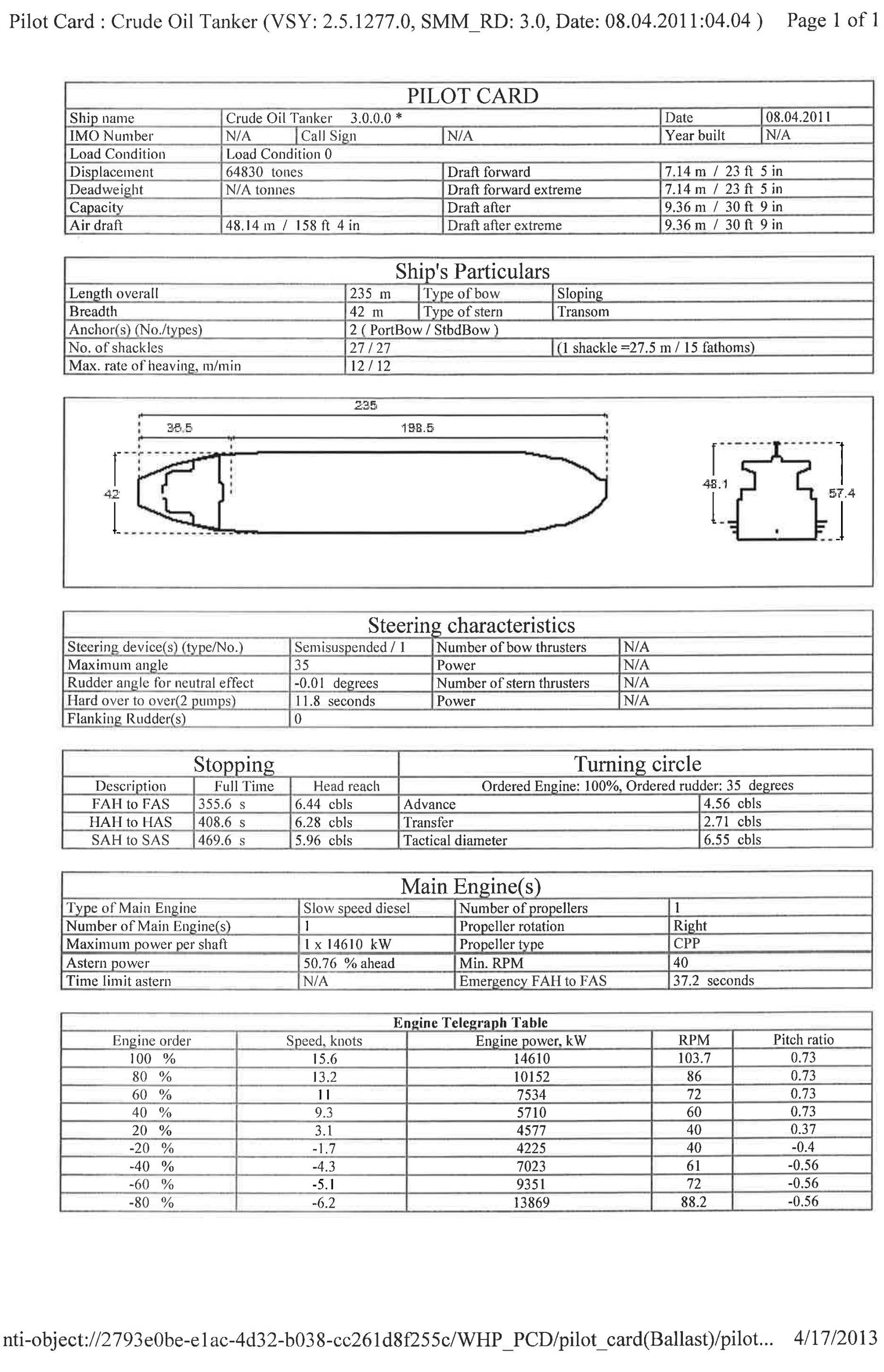
[50] S. Ghosh, "Understanding different types of manoeuvres of a vessel -part 2," *Marine insight*, 2019.

[51] S. Penttinen, *Colreg compliant collision avoidance using reinforcement learning*, 2020.

[52] R. Zaccone, M. Martelli, and M. Figari, "A COLREG-compliant ship collision avoidance algorithm," in *2019 18th European Control Conference (ECC)*, IEEE, 2019.

A. Appendix

A.1 Oil Tanker Pilot Card



Pilot Card : Crude Oil Tanker (VSY: 2.5.1277.0, SMM_RD: 3.0, Date: 08.04.2011:04.04) Page 1 of 1

PILOT CARD					
Ship name	Crude Oil Tanker 3.0.0.0 *			Date	08.04.2011
IMO Number	N/A	Call Sign	N/A	Year built	N/A
Load Condition	Load Condition 0				
Displacement	64830 tones		Draft forward	7.14 m / 23 ft 5 in	
Deadweight	N/A tonnes		Draft forward extreme	7.14 m / 23 ft 5 in	
Capacity			Draft after	9.36 m / 30 ft 9 in	
Air draft	48.14 m / 158 ft 4 in		Draft after extreme	9.36 m / 30 ft 9 in	

Ship's Particulars			
Length overall	235 m	Type of bow	Sloping
Breadth	42 m	Type of stern	Transom
Anchor(s) (No./types)	2 (PortBow / StbdBow)		
No. of shackles	27 / 27		(1 shackle =27.5 m / 15 fathoms)
Max. rate of heaving, m/min	12 / 12		

Steering characteristics			
Steering device(s) (type/No.)	Semisuspended / 1	Number of bow thrusters	N/A
Maximum angle	35	Power	N/A
Rudder angle for neutral effect	-0.01 degrees	Number of stern thrusters	N/A
Hard over to over(2 pumps)	11.8 seconds	Power	N/A
Flanking Rudder(s)	0		

Stopping			Turning circle	
Description	Full Time	Head reach	Ordered Engine: 100%, Ordered rudder: 35 degrees	
FAH to FAS	355.6 s	6.44 cbls	Advance	4.56 cbls
HAH to HAS	408.6 s	6.28 cbls	Transfer	2.71 cbls
SAH to SAS	469.6 s	5.96 cbls	Tactical diameter	6.55 cbls

Main Engine(s)			
Type of Main Engine	Slow speed diesel	Number of propellers	1
Number of Main Engine(s)	1	Propeller rotation	Right
Maximum power per shaft	1 x 14610 kW	Propeller type	CPP
Astern power	50.76 % ahead	Min. RPM	40
Time limit astern	N/A	Emergency FAH to FAS	37.2 seconds

Engine Telegraph Table

Engine order	Speed, knots	Engine power, kW	RPM	Pitch ratio
100 %	15.6	14610	103.7	0.73
80 %	13.2	10152	86	0.73
60 %	11	7534	72	0.73
40 %	9.3	5710	60	0.73
20 %	3.1	4577	40	0.37
-20 %	-1.7	4225	40	-0.4
-40 %	-4.3	7023	61	-0.56
-60 %	-5.1	9351	72	-0.56
-80 %	-6.2	13869	88.2	-0.56

nti-object://2793e0be-e1ac-4d32-b038-cc261d8f255c/WHP_PCD/pilot_card(Ballast)/pilot... 4/17/2013

WHEELHOUSE POSTER

Ship's name Crude Oil Tanker 3.0.0.0 , Call sign N/A ,
Gross tonnage N/A , Net tonnage N/A , Load Condition Load Condition 0 , Displacement 64830 tones , Deadweight N/A tones

DRAFTS IN PRESENT CONDITION	
Forward	7.14 m
Forward extreme	7.14 m
After	9.36 m
After extreme	9.36 m

STEERING PARTICULARS	
Type of rudder	Semisuspended
Maximum rudder angle	35 degrees
Hard-over to hard-over(1/2 pumps)	23.6 sec/11.8 sec
Neutral effect angle	-0.01 degrees
Flanking Rudders	0

ANCHORS INFO	
Anchor(s) (No./types)	2 (PortBow / StbdBow)
No. of shackles	27 / 27
Max. rate of heaving, m/min	12 / 12

PROPULSION PARTICULARS			
Type of Main Engine	Slow speed diesel	Number of propellers	1
No. of Main Engines	1	Propeller rotation	Right
Max. power per shaft	1 x 14610 kW	Propeller type	CPP
Astern power	50.76 % ahead	Min. RPM	40
Time limit astern	N/A	Emergency FAH to FAS	37.2 seconds

Engine Telegraph Table

Engine order	Speed, knots	Engine power, kW	RPM	Pitch ratio
100 %	15.6	14610	103.7	0.73
80 %	13.2	10152	86	0.73
60 %	11	7534	72	0.73
40 %	9.3	5710	60	0.73
20 %	3.1	4577	40	0.37
-20 %	-1.7	4225	40	-0.4
-40 %	-4.3	7023	61	-0.56
-60 %	-5.1	9351	72	-0.56
-80 %	-6.2	13869	88.2	-0.56

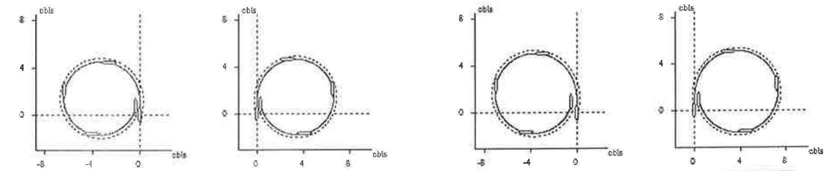
THRUSTER EFFECT

Thruster (s)	No. of units	Power (kW)	Time delay for full thrust(s)	Turning rate at zero speed(degrees/min)	Time delay to reverse full thrust(s)	Not effective above speed (knots)
Bow	N/A					
Stern	N/A					
Combined	N/A					

DRAFT INCREASE IN PRESENT CONDITION

Squat effect				Heel effect	
Under keel clearance	Ship's speed	Bow squat	Stern squat	Heel angle	Draft increase
3m	14.96 knots	-0.01 m	0.04 m	2 deg	0.57 m
	13.11 knots	-0.01 m	0.03 m	4 deg	1.1 m
	11.14 knots	-0.01 m	0.02 m	8 deg	2.11 m
2 m	14.93 knots	-0.01 m	0.07 m	12 deg	3.03 m
	13.09 knots	-0.01 m	0.05 m	16 deg	3.86 m

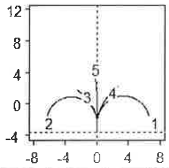
Deep Water TURNING CIRCLES Shallow Water*



Eng.	Rudd.	Advance	Transfer	Tact. D	Final RoT	Final speed	Final time
100	35	4.57 cbls	2.72 cbls	6.56 cbls	28 deg/min	9 knots	708.6 s
100	-35	4.45 cbls	-2.59 cbls	-6.33 cbls	-29 deg/min	9 knots	690.6 s

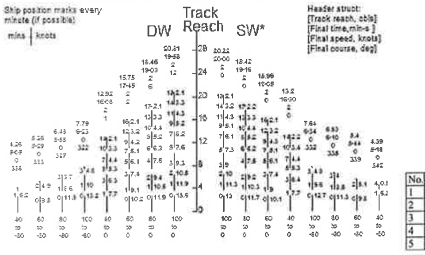
Eng.	Rudd.	Advance	Transfer	Tact. D	Final RoT	Final speed	Final time
100	35	5.07 cbls	3.03 cbls	7.22 cbls	26 deg/min	10 knots	777.6 s
100	-35	4.97 cbls	-2.96 cbls	-7.01 cbls	-27 deg/min	9 knots	762.6 s

Emergency Manoeuvers(DW)

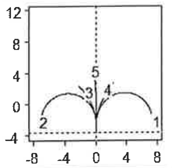


No	Rudd	Eng	Full_time	Head_reach	Side_reach
1	35	100	333.6 s	2.05 cbls	6.56 cbls
2	-35	100	323.6 s	2.06 cbls	-6.33 cbls
3	35	-80	335.6 s	5.48 cbls	-2.51 cbls
4	-35	-80	378.6 s	5.96 cbls	2.66 cbls
5	0	-80	383.6 s	7.78 cbls	-0.47 cbls

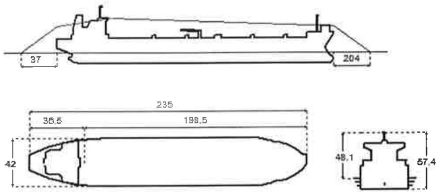
STOPPING CHARACTERISTICS



Emergency Manoeuvers(SW*)



No.	Rudd.	Eng.	Full_time	Head_reach	Side_reach
1	35	100	372.6 s	2.26 cbls	7.22 cbls
2	-35	100	364.6 s	2.29 cbls	-7.01 cbls
3	35	-80	360.6 s	6.08 cbls	-2.23 cbls
4	-35	-80	392.6 s	6.48 cbls	2.23 cbls
5	0	-80	395.6 s	7.64 cbls	-0.3 cbls



MAN OVERBOARD
RESCUE MANOEUVRE

SEQUENCE OF ACTION TO BE TAKEN:

- TO CAST A BUOY
- TO GIVE THE HELM ORDER
- TO SOUND THE ALARM
- TO KEEP THE LOOK OUT

Approximate Maneuver Program

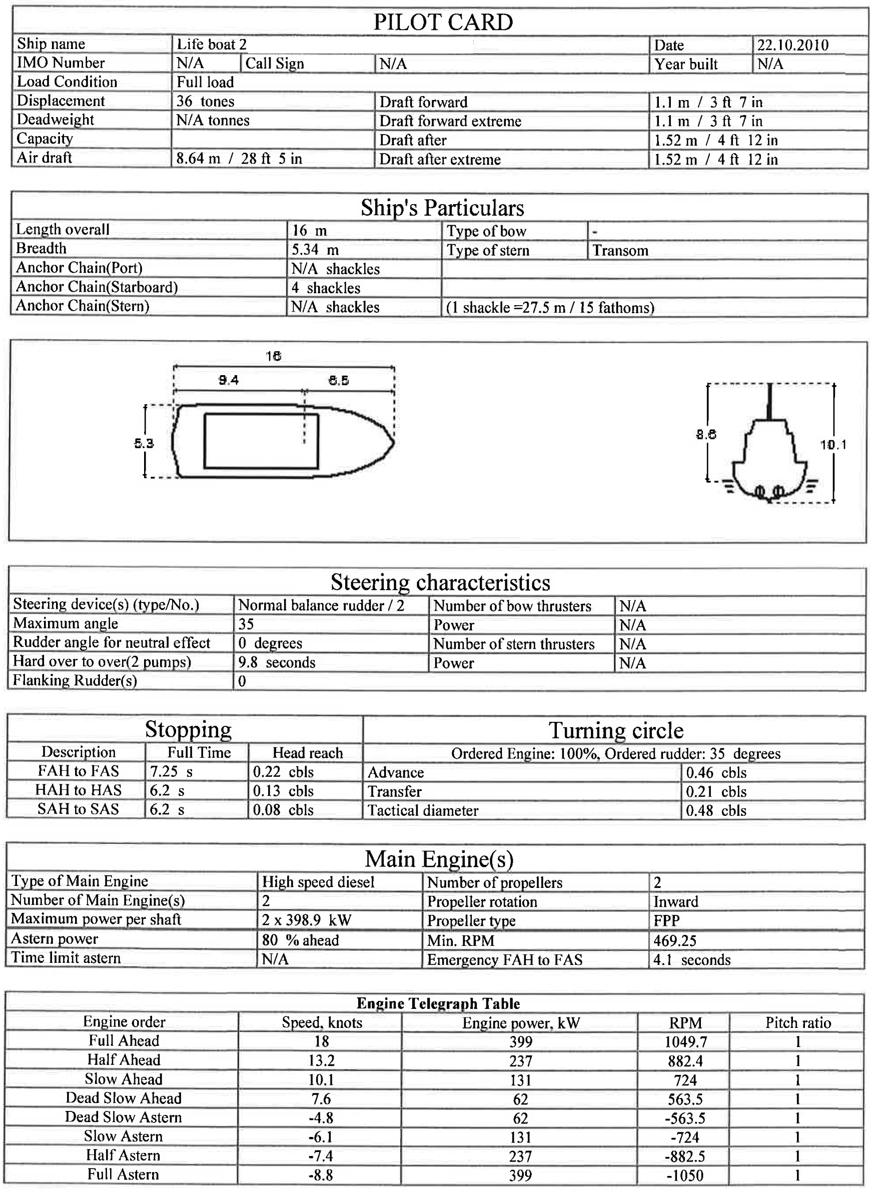
Time	Action
0 s	Set rudder 35 STBD. Wait till ship course altered to 67.5 degrees from initial.
118 s	Set rudder 35 PORT. Wait till course altered to -170 degrees from initial.
613 s	Turn AP on. The difference between AP course and initial course must be 180 degrees.

Bridge To Stern(A)	36.5 m	Length of Midbody(D)	176.25 m	Air Draft(G)	48.14 m / 158 ft 4 in
Bridge To Bow(B)	198.5 m	Length Overall(E)	235 m	Forward Blind Zone(I)	204 m
Breadth(C)	42 m	Height(F)	56.39 m	Backward Blind Zone(J)	37 m

* Shallow Water: depth is equal 1.2 Draft ** Model: 2.41.732.18; VSY02: 2.5.1277.0;

PERFORMANCE MAY DIFFER FROM THIS RECORD DUE TO ENVIRONMENT, HULL AND LOADING CONDITION

A.2 Lifeboat Pilot Card



PILOT CARD

Ship name	Life boat 2			Date	22.10.2010
IMO Number	N/A	Call Sign	N/A	Year built	N/A
Load Condition	Full load				
Displacement	36 tones		Draft forward	1.1 m / 3 ft 7 in	
Deadweight	N/A tonnes		Draft forward extreme	1.1 m / 3 ft 7 in	
Capacity			Draft after	1.52 m / 4 ft 12 in	
Air draft	8.64 m / 28 ft 5 in		Draft after extreme	1.52 m / 4 ft 12 in	

Ship's Particulars

Length overall	16 m	Type of bow	-
Breadth	5.34 m	Type of stern	Transom
Anchor Chain(Port)	N/A shackles		
Anchor Chain(Starboard)	4 shackles		
Anchor Chain(Stern)	N/A shackles	(1 shackle =27.5 m / 15 fathoms)	

Steering characteristics

Steering device(s) (type/No.)	Normal balance rudder / 2	Number of bow thrusters	N/A
Maximum angle	35	Power	N/A
Rudder angle for neutral effect	0 degrees	Number of stern thrusters	N/A
Hard over to over(2 pumps)	9.8 seconds	Power	N/A
Flanking Rudder(s)	0		

Stopping			Turning circle	
Description	Full Time	Head reach	Ordered Engine: 100%, Ordered rudder: 35 degrees	
FAH to FAS	7.25 s	0.22 cbls	Advance	0.46 cbls
HAH to HAS	6.2 s	0.13 cbls	Transfer	0.21 cbls
SAH to SAS	6.2 s	0.08 cbls	Tactical diameter	0.48 cbls

Main Engine(s)

Type of Main Engine	High speed diesel	Number of propellers	2
Number of Main Engine(s)	2	Propeller rotation	Inward
Maximum power per shaft	2 x 398.9 kW	Propeller type	FPP
Astern power	80 % ahead	Min. RPM	469.25
Time limit astern	N/A	Emergency FAH to FAS	4.1 seconds

Engine Telegraph Table

Engine order	Speed, knots	Engine power, kW	RPM	Pitch ratio
Full Ahead	18	399	1049.7	1
Half Ahead	13.2	237	882.4	1
Slow Ahead	10.1	131	724	1
Dead Slow Ahead	7.6	62	563.5	1
Dead Slow Astern	-4.8	62	-563.5	1
Slow Astern	-6.1	131	-724	1
Half Astern	-7.4	237	-882.5	1
Full Astern	-8.8	399	-1050	1

WHEELHOUSE POSTER

Ship's name Life boat , Call sign N/A ,
Gross tonnage N/A , Net tonnage N/A , Load Condition Full load , Displacement 36 tones , Deadweight N/A tones

DRAFTS IN PRESENT CONDITION	
Forward	1.1 m
Forward extreme	1.1 m
After	1.52 m
After extreme	1.52 m

STEERING PARTICULARS	
Type of rudder	Normal balance rudder
Maximum rudder angle	35 degrees
Hard-over to hard-over(1/2 pumps)	19.6 sec/9.8 sec
Neutral effect angle	0 degrees
Flanking Rudders	0

ANCHOR CHAIN		
	No. of shackles	Max. rate of heaving
Port	N/A shackles	N/A m/min
Starboard	4 shackles	10.02 m/min
Stern	N/A shackles	N/A m/min
(1 shackle = 27.5 m /15 fathoms)		

PROPULSION PARTICULARS			
Type of Main Engine	High speed diesel	Number of propellers	2
No. of Main Engines	2	Propeller rotation	Inward
Max. power per shaft	2 x 398.9 kW	Propeller type	FPP
Astern power	80 % ahead	Min. RPM	469.25
Time limit astern	N/A	Emergency FAH to FAS	4.1 seconds

Engine Telegraph Table

Engine order	Speed, knots	Engine power, kW	RPM	Pitch ratio
Full Ahead	18	399	1049.7	1
Half Ahead	13.2	237	882.4	1
Slow Ahead	10.1	131	724	1
Dead Slow Ahead	7.6	62	563.5	1
Dead Slow Astern	-4.8	62	-563.5	1
Slow Astern	-6.1	131	-724	1
Half Astern	-7.4	237	-882.5	1
Full Astern	-8.8	399	-1050	1

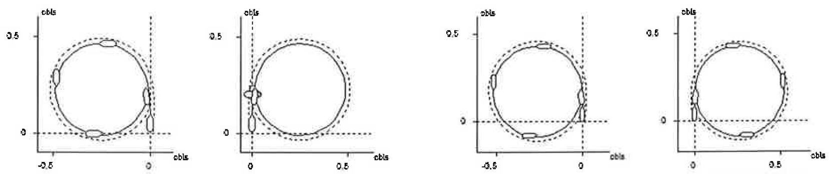
THRUSTER EFFECT

Thruster (s)	No. of units	Power (kW)	Time delay for full thrust(s)	Turning rate at zero speed(degrees/min)	Time delay to reverse full thrust(s)	Not effective above speed (knots)
Bow	N/A					
Stern	N/A					
Combined	N/A					

DRAFT INCREASE IN PRESENT CONDITION

Squat effect				Heel effect	
Under keel clearance	Ship's speed	Bow squat	Stern squat	Heel angle	Draft increase
3m	14.7 knots	-0.76 m	0.4 m	2 deg	0.07 m
	14.7 knots	-0.76 m	0.4 m	4 deg	0.14 m
	11.71 knots	-0.25 m	0.38 m	8 deg	0.26 m
2 m	14.48 knots	-0.75 m	0.41 m	12 deg	0.38 m
	14.48 knots	-0.75 m	0.41 m	16 deg	0.48 m

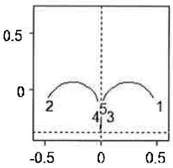
Deep Water TURNING CIRCLES Shallow Water*



Eng.	Rudd.	Advance	Transfer	Tact. D	Final RoT	Final speed	Final time
100	35	0.46 cbls	0.21 cbls	0.48 cbls	463 deg/min	12 knots	49.1 s
100	-35	0.46 cbls	-0.22 cbls	-0.49 cbls	-463 deg/min	12 knots	49.1 s

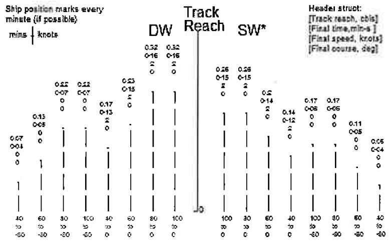
Eng.	Rudd.	Advance	Transfer	Tact. D	Final RoT	Final speed	Final time
100	35	0.43 cbls	0.22 cbls	0.51 cbls	404 deg/min	11 knots	56.45 s
100	-35	0.43 cbls	-0.23 cbls	-0.52 cbls	-404 deg/min	11 knots	56.45 s

Emergency Manoeuvers(DW)

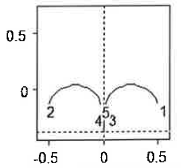


No.	Rudd	Eng	Full_time	Head_reach	Side_reach
1	35	100	25.25 s	0.28 cbls	0.48 cbls
2	-35	100	25.25 s	0.28 cbls	-0.49 cbls
3	35	-80	7.25 s	0.21 cbls	-0.01 cbls
4	-35	-80	7.25 s	0.21 cbls	0 cbls
5	0	-80	7.25 s	0.22 cbls	-0.01 cbls

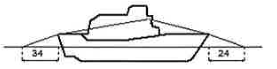
STOPPING CHARACTERISTICS

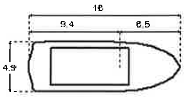


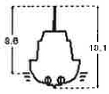
Emergency Manoeuvers(SW*)



No.	Rudd	Eng	Full_time	Head_reach	Side_reach
1	35	100	29.25 s	0.22 cbls	0.51 cbls
2	-35	100	29.25 s	0.22 cbls	-0.52 cbls
3	35	-80	7.25 s	0.17 cbls	0 cbls
4	-35	-80	7.25 s	0.17 cbls	-0.01 cbls
5	0	-80	7.25 s	0.17 cbls	-0.01 cbls







MAN OVERBOARD
RESCUE MANOEUVRE

SEQUENCE OF ACTION TO BE TAKEN:

- TO CAST A BUOY
- TO GIVE THE HELM ORDER
- TO SOUND THE ALARM
- TO KEEP THE LOOK OUT

Approximate Maneuver Program

Time	Action
0 s	Set rudder 35 STBD. Wait till ship course altered to 15 degrees from initial.
50 s	Set rudder 35 PORT. Wait till course altered to -170 degrees from initial.
97 s	Turn AP on. The difference between AP course and initial course must be 180 degrees.

Bridge To Stern(A)	9.46 m	Length of Midbody(D)	12 m	Air Draft(G)	8.64 m / 28 ft 5 in
Bridge To Bow(B)	6.54 m	Length Overall(E)	16 m	Forward Blind Zone(I)	24 m
Breadth(C)	4.9 m	Height(F)	9.95 m	Backward Blind Zone(J)	34 m

* Shallow Water: depth is equal 3 Draft ** Model: 2.32.536.19 ; VSY02: 2.4.972.0;

PERFORMANCE MAY DIFFER FROM THIS RECORD DUE TO ENVIRONMENT, HULL AND LOADING CONDITION