National Technical University of Athens
School of Naval Architecture & Marine Engineering

Total Fuel Oil Consumption (TFOC) minimization using Reinforcement Learning (RL)

Submitted in partial fulfilment of the requirements for the
M.Sc School of Naval Architecture & Marine Engineering

Author
Konstantinos Mesolongitis

Supervisor
Associate Professor George Papalambrou

Committee members
Associate Professor Christos Papadopoulos
Assistant Professor Nikolaos Themelis

September 2022

Abstract

The purpose of this research was to discover if Reinforcement Learning (RL) could produce remarkable results in assisting an autonomous ship to minimize the Total Fuel Oil Consumption (TFOC). Furthermore, if the first goal was to be achieved, a second equally important objective was the discovery of the most suitable RL agent for this initiative.

In this particular case study, the environment consisted of the sea area - which can be explored by the ship - the existing boundaries, - such as islands, rocky islets and land where the access is forbidden - and the weather conditions.

The first part was related to the environment construction. A grid-world environment was selected and the testing route represented a short journey between two ports in the area of the Faroe Islands: Tórshavn and Krambatangi. There was no particular reason for choosing this exact course, other than the fact that this was the first constructed environment, and since it was capable of acting as proof of concept for the needs of the project, it was decided to be the final environment.

The boundaries were transformed according to a computationally cost-effective technique that was developed for the purpose of this task. In order to achieve this, some of the accuracy, as regarding the boundaries position, was sacrificed. This decision was made taking into consideration the massive amount of computations being executed during a Reinforcement Learning experiment. The aforementioned choice did not really substantially affect the problem at hand at all.

Since permitting an actual ship to explore its environment for the sake of learning is an unnecessary and costly enterprise (ship rental, fuel costs, crew related costs, ...), relevant simulations are implemented instead.

A reliable fuel consumption process was mandatory for the success of this project. To achieve that, data were obtained from a shipping company, which

were used to train an Artificial Neural Network (ANN). A Long Short Term Memory (LSTM) ANN was implemented as it has been proven to be superior at tracking a time-series outcome. Especially when the previous observations are not independent form the current ones. The following features were used as inputs; Speed overground, Significant Waves Height, Draught AFT, Draught FWD and Distance Overground.

A weather approximating function was designed in order to be applied into the environment. A storm center that interfered with the minimum distance route was chosen, in order to examine if the agent was capable of learning to choose the longer but cheaper route - the one where the minimum amount of fuel was consumed.

Since the environment's state space was really large, it appeared like a Deep Q-Network (DQN) agent would be the best option for this project. A value-approximating agent was required, in order to predict the values of states that had not been experienced before. However, the limits of the Q-learning agent were put to the test and the improved and more sophisticated rainbow agent was employed in order to detect the best available option.

The results revealed the dominance of the rainbow agent. Q-learning agent, as expected, was limited from the state's space size. While, on the other hand, the DQN agent was extremely unstable and sensitive to hyper-parameters tuning. This application, also demonstrates the feasibility of minimizing the Total Fuel Oil Consumption (TFOC) using Reinforcement Learning (RL).

Dedication

This study is wholeheartedly dedicated to my parents Konstantina and Vasileios, who have offered me more than I could possibly ever ask for. This is the ending chapter of a five year journey and countless opportunities lie ahead. I commit on always going after personal growth as it seems to be the only meaningful way to enjoy life and I sincerely hope that I make my parents proud along the way; not for my achievements but because of theirs, raising me to the best of their capabilities.

This thesis is also partially dedicated to my grandfather Nikolaos as he is a figure I truly admire both for his emotional strength and mental clarity.

Acknowledgements

This work was carried out at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture & Marine Engineering (SNAME) at National Technical University of Athens (NTUA).

I would like to thank Assistant Professor George Papalambrou of School of Naval Architecture & Marine Engineering (SNAME) at National Technical University of Athens (NTUA), Greece for the significant opportunity he offered me by suggesting to undertake a thesis on Reinforcement Learning.

I also want to deeply thank Orfeas Bourchas, PhD student at the Laboratory of Marine Engineering (LME) at National Technical University of Athens (NTUA) for sharing his thoughts and providing regular and substantial feedback.

I would like to sincerely thank Albin Heimerson, PhD student at the department of Automatic Control at Lund University, Sweden for the significant help he offered me, through his detailed replies, on a variety of issues related to the Julia Language.

I really need to thank the shipping company Minerva Marine Inc. for their tremendous generosity of sharing data from their ships' operation. This data was used to develop a predictive model for the fuel consumption. I would specifically like to thank Mr. Mike Servos, Head of Energy Efficiency Department at Minerva Marine Inc. for taking the initiative to support this study and Dr. Timoleon Plessas, Energy Engineer at Minerva Marine Inc. for being available for me whenever I needed him.

Finally, I would like to thank all of my friends for the emotional support throughout these exacting months and for their sharp judgement and feedback which I deeply appreciate. Kyriakos, Penny, Chris, Giannis, Chronis thank you!

I'm grateful. It really would not be possible without the help all of you offered me.

List of Figures

List of Tables

Contents

Glossary

AI Artificial Intelligence. 12, 13, 16

ANN Artificial Neural Network. 2, 7, 14, 16, 24, 27, 41, 42, 46–48, 56, 69

DP Dynamic Programming. 20

DQN Deep Q-Network. 2, 5–7, 13–15, 23–27, 38, 39, 49, 50, 63–65, 69

ETA Estimated Time of Arrival. 52, 70

HFO Heavy Fuel Oil. 13

LME Laboratory of Marine Engineering. 4, 69

LSTM Long Short Term Memory. 2, 5, 7, 14–16, 30, 31, 34, 46, 48, 54, 56, 69

MDP Markov Decision Process. 18, 19

ML Machine Learning. 13, 16

NTUA National Technical University of Athens. 4

PCA Principal Component Analysis. 5, 28, 43, 44, 55

RL Reinforcement Learning. 1, 2, 4, 5, 13, 14, 16–23, 26, 27, 33, 37, 39, 61, 65, 68

RNN Recurrent Neural Network. 30

SNAME School of Naval Architecture & Marine Engineering. 4

TD Temporal Difference. 22, 26, 61

TFOC Total Fuel Oil Consumption. 1, 2, 13–16, 68

Chapter 1

Statement of the Problem

1.1 Motivation

The urge of the climate crisis has led to truly significant work on the performance issues. Almost every business nowadays in the energy, industrial and agricultural sectors is investigating how to reduce the harmful emissions.

In shipping, which according to data from "Our World in Data" accounts for 1.7 % of the total CO_2 produced, emissions are caused from burning petrol and diesel [1]. As expected, the greatest proportion is caused for the needs of freight maritime trips.

The shipping industry, due to new regulations, appears to be making enormous efforts to reduce its carbon footprint. There have been advancements in internal combustion engines, in the direction of burning more environmentally friendly fuels: at first like LNG and soon like biomass from agricultural waste. New ships are designed with the goal of having no ballast system. Nowadays, all ships are equipped with Sulfur Scrubber Systems that lead to reduction up to 98 % of SO_x emissions. Noticeable work is being done on the efficiency of the propeller and rudder systems. Great care and thoughtfulness is apparent to the hull paint technology as well. Applying appropriate paint on the correct hull area results in reducing the frictional resistance of the ship, ensuing in 3-8 % of fuel savings. Exhaust Gas Recirculation and the use of water inside the fuel are tools that assure decreased NO_x emissions.

Right now one of the most promising technological sectors seems to be Artificial Intelligence (AI). The question that naturally arises is: can AI be of substantial use in order to achieve solid advancements on the aforementioned

environmental issues?

Recognizing these hard facts, an attempt was completed in researching if it is possible to train AI to make judgements for the course of a ship regarding its route and its speed. The final goal was to minimize the total amount of Heavy Fuel Oil (HFO) burnt. Of course, this approach could be expanded for any type of fuel.

Reinforcement learning is a type of Machine Learning and thereby also a branch of AI. It offers the opportunity to the machine and software agents to automatically determine the ideal behaviour within a specific context. The desired goal is achieved by maximizing its performance by strategically assigning rewards and punishments. Based on the algorithm, a reinforcement learning agent learns from the consequences of its actions, rather than from being explicitly taught.

In 2015, Google's Deepmind published the paper Human-level control through deep reinforcement learning in Nature journal [2]. It was a true breakthrough in the area. A Deep Q-Network was demonstrated that was capable of learning to control a policy, by directly using raw pixels data and the score of the games. Such experiments offered the evidence that this artificial network was truly able of producing better performance results compared to both previous existing algorithms and professional humans, across a set of 49 well-known games.

1.2 Objective

A Deep Q-Network (DQN) agent was utilized using the standard RL environment Grid World, in order to describe a real environment consisted of land, islands and sea water. Storms were included representing the changes in the weather. The goal was to design a single agent that was capable of learning to minimize the Total Fuel Oil Consumption (TFOC) by maximizing the future rewards.

As it is already described Reinforcement Learning (RL) is a computational procedure of an agent following a trial and error paradigm, the real-life example would be an agent taking decisions for a ship. Obviously this cannot be achieved due both to financial and practical reasons. Therefore what is needed instead, is an appropriate entity simulating all the necessary ship characteristics. The most important of which, is the fuel consumption. Of course, features like speed and loading conditions should be taken into

account, as they affect TFOC.

Total Fuel Oil Consumption can be predicted using empirical formulas or by implementing a simple predictive Artificial Neural Network (ANN). The most suitable result is definitely a method capable of producing trustworthy time-sequential results. The above mentioned desire led to choosing a type of ANN, that possesses memory of previous results, called Long Short Term Memory (LSTM) ANN.

1.3 Structure

This thesis starts by presenting all the necessary background information for reading it in Chapter 2. Reinforcement Learning (RL) is explained and all the necessary mathematical concepts are introduced. The various types of agents are demonstrated. The used feature reduction methods are displayed, while a proper initial report to the sophisticated architecture of Long Short Term Memory (LSTM) ANN is completed. Finally, there are a few words dedicated to the applied coding language; Julia.

In Chapter 3, related papers and literature in general are reviewed.

In Chapter 4, the grid world environment design and construction is discussed. This process includes choosing real routes and importing, in the appropriate (computationally cost-effective) form, all the existing boundaries such as lands, islands and shallow water areas. The weather modelling details are displayed. Last but not least, the three implemented agents are presented.

In Chapter 5, the Total Fuel Oil Consumption (TFOC) prediction process is unveiled. The first steps include the data pre-processing and feature reduction. Then, the LSTM data preparation, loss function, optimizer and the final architecture are displayed.

In Chapter 6, the challenges regarding the Reinforcement Learning (RL) are presented. More specifically, the local optimums, hyper-parameters sensitivity, rewarding strategy and the dense or sparse environment dilemma are settled.

In Chapter 7, the Long Short Term Memory (LSTM) ANN implementation is discussed. The over-fitting issue is the major concern. The final results are presented.

In Chapter 8, the comparison between the three types of agents (Q-learning, DQN and rainbow) comes a conclusion. The clear dominant agent

for this application is recognised as the results are displayed.

In Chapter 9, the summary and conclusions can be found.

In Chapter 10, there are recommendations for future work. These include the improvement of the weather modelling, the handling of the speed parameter, the insertion of engine data to the predictive model, a live action methodology, the manipulation of changes in draught and trim and last but not least, the hyper-parameters calibration.

1.4 Literature

The book [3] from Richard S. Sutton and Andrew G. Barto is considered a great starter to study Reinforcement Learning. The second edition does not include fresher ideas like DQNs.

The thesis [4] from Joan Petersen, provides excellent intuition on handling ship data.

The thesis [5] from Orfeas Bourchas on Neural Networks for the Prediction of Fuel Oil Consumption for Containerships, is the closest LSTM implementation for the exact same issue tackled in this thesis regarding the Total Fuel Oil Consumption (TFOC).

The article [6] from Christos Gkerekos, Iraklis Lazakis, Gerasimos Theodokatos presents a comprehensive comparison between various regression algorithms for fuel consumption prediction.

The article [7] from Lucia Moreira, Roberto Vettor and Carlos Guedes Soares, displayed explicitly the neural network approach for predicting fuel consumption. Interestingly, it seems that using engine data leads to almost perfect prediction while applying only weather data leads to relatively significant inaccuracy.

The technical report [8] from Markus Mathisen Johansen is an initial approach in using reinforcement learning for a ship. The environment approach was based on the Mountain Car Environment and the Q-learning agent was used. The final goal was achieving some ship manoeuvres in ports and the application was trivial.

Chapter 2

Background

Long Short Term Memory (LSTM) Artificial Neural Network (ANN) was implemented to obtain a reliable approximating (predictive) function for the Total Fuel Oil Consumption (TFOC). To train the aforementioned predictive model, appropriately processed data were required. Also, the necessary features had to be selected. Finally, an agent was mostly punished for the amount of fuel burnt throughout each journey (episode). Three different agents were put to the test.

2.1 Reinforcement Learning

Influence We as humans, more often than not, tend to learn by experimenting, observing the results and deciding whether we liked the outcome or not. From another perspective, we learn by interacting with the environment we live in, even from a very young age. Reinforcement Learning (RL) is a Machine Learning (ML) paradigm inspired by that same way we as humans learn; we try, we fail and then we try some more.

Conclusively, Reinforcement Learning is a Machine Learning training procedure based on rewarding or punishing behaviours. A RL agent explores a dynamic environment through a computational approach. It takes actions and according to them, it gradually learns by trial and error.

Its goal is to maximize the total reward. Despite the fact the designer sets the rewards and punishments, there are no hints whatsoever given to the Artificial Intelligence (AI) as to how to solve the problem at hand. It is up to the agent to figure out how to perform the desired task in order to

maximize the reward.

Machine Learning paradigms Reinforcement Learning is considered to be one of the three basic machine learning paradigms. The other two are supervised and unsupervised learning.

Supervised learning is using a training set, where the outcomes are known from an external supervisor. The goal here is the ability to later generalize to situations that are not present in the training procedure.

However, this paradigm can not be applied to situations where there are not known outcomes. The process of finding the hidden structure of situation like these is called unsupervised learning. Although, it seems like reinforcement learning and unsupervised learning are pretty much selfsame because both do not get a feedback in terms of corrected behaviour, they are not. And that is because Reinforcement Learning instead of trying to unveil a hidden structure, it tries to maximize the reward.

Architecture An agent is initially placed in a state inside an environment. It interacts with the environment by taking actions. Every time it completes one set of actions, this leads in changing states and receiving rewards. These rewards can either be good or bad (punishments).

The ultimate goal is to unveil a solution that maximizes the overall received reward. The reward signal is how it is communicated to the agent what the designer would like it to achieve, not how this will be accomplished.

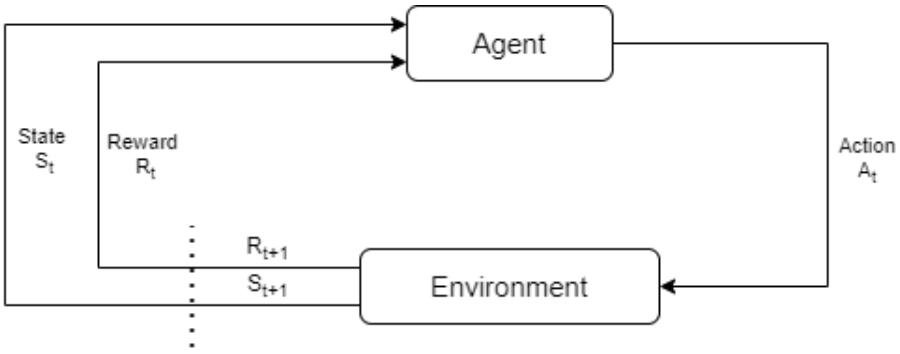

Figure 2.1: Reinforcement Learning (RL) architecture

Generally, it is desired to maximize the expected return. This is not the same with the sequence of returns described by the following equation.

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{2.1}$$

A well-known concept in RL is discounting. By using this approach, the agent is trying to select actions in order to maximize over time the discounted rewards.

$$\begin{aligned} G_t &= R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \\ &= R_{t+1} + \gamma \cdot G_{t+1}, \quad 0 \leq \gamma \leq 1 \end{aligned} \tag{2.2}$$

Delayed reward What is important to be noted is that despite the fact that the agent is mapping situations to actions and rewards, one particular action usually does not only affect the immediate reward but also many more rewards to come. This observation leads to another really crucial RL characteristic; the delayed reward. The agent really can not know what will be the gained total reward before the end of an episode. Especially if the focus is on episodic implementations like the one discussed here. Usually, the agents starts by experimenting with random actions until it eventually discovers really sophisticated solutions.

Exploration-Exploitation dilemma Last, one of the main challenges of Reinforcement Learning is the trade-off between exploitation and exploration. An agent desiring to instantly achieve the maximum reward, it has to exploit situations already experienced before. However, if it only exploits, then it will never explore options that will lead to a sequence of situations and actions with significantly greater total reward. This mathematical problem has been studied for decades. Still, however, there is no clear and definite answer.

Mathematical formalization The problem is formalized mathematically using some concepts from dynamical systems theory. The most useful tool of all is Markov Decision Process (MDP).

2.2 Markov Decision Process

Definition In mathematics, Markov Decision Process (MDP) is called a discrete time stochastic control process. It is actually a classic formalization of sequential decision. It works as a great framework for modelling such situations and is extremely helpful for solving optimization problems using dynamic programming.

RL implementation For Reinforcement Learning purposes, it is extremely helpful since it can model extremely well situations where actions do not only influence the immediate rewards, but also affect subsequent situations/states and subsequently future rewards [9].

What is normally done in MDPs, is calculating the value $q_*(s, \alpha)$ of taking action α while being in state s, or sometimes (depending on the project's needs) the value $v_*(s)$ of each state s, considering that optimal action selections are given.

An illustration of the sequence produced by MDP and the agent is the following:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, ... \tag{2.3}$$

In a finite MDP where there are finite number of elements regarding the states, actions and rewards, reward R_t and state S_t at time step t have easily defined discrete probability distributions on the preceding state and action.

Environment dynamics Of course, if we sum the above probability distributions for every action α and every state s, we get that,

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, \alpha) = 1, \ \forall s \in S, \ \alpha \in A(s) \tag{2.4}$$

The dynamics of the environment in a MDP are described by the above equations. Also, in order to have the famous "Markov Property", the state has to include information about all the aspects of the past interactions that could make a difference in the future.

2.3 Value Functions & Policies

In plain words, the values correspond to "how good" a situation is for the agent in terms of expected rewards. The action-state value functions $Q(\alpha|s)$ and the state value functions $V(s)$ are just functions estimating "how good" these situations are. On the other hand, policy is the probability of choosing action α if the agent is in state s. The notation used is $\pi(\alpha|s)$.

Definition The definition of the value function of a state s under policy π is the following,

$$\begin{aligned} \upsilon_\pi(s) &= \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s] \\ &= \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right], \forall s \in S \end{aligned} \tag{2.5}$$

The definition of the value function of taking action α while in a state s under policy π is the following,

$$\begin{aligned} q_\pi(s) &= \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s, A_t = \alpha] \\ &= \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = \alpha\right] \end{aligned} \tag{2.6}$$

Bellman equation Finally, by using the definition (2.5) of the value function and the recursive relationship displayed in (2.2), one of the most popular equations in Reinforcement Learning and Dynamic Programming (DP) pops up; the Bellman Equation.

$$\upsilon_\pi(s) = \sum_{\alpha}\pi(\alpha|s)\sum_{s',r}p(s',r|s,a)\cdot[r + \gamma\upsilon_\pi(s')],\ \forall s \in S \tag{2.7}$$

The Bellman Equation expresses the relationship between the value of current state and the values of the next states. It averages over all the possibilities by using as weights every probability of occurring.

Finally, it is important to define the optimal policy and the optimal value functions. A policy π is better than a policy π' ($\pi \geq \pi'$) if and only if $\upsilon_\pi(s) \geq \upsilon_{\pi'}(s), \forall s \in S$.

We denote the optimal policy as π_*. This policy corresponds to the optimal value function as following,

$$\upsilon_*(s) = max_\pi \upsilon_\pi(s),\ \forall s \in S \tag{2.8}$$

Similarly, for optimal action-value functions,

$$\begin{aligned} q_*(s,\alpha) &= max_\pi q_\pi(s,a) \\ &= \mathbb{E}[R_t + \gamma \upsilon_*(S_{t+1}) | S_t = s, A_t = \alpha] \end{aligned} \tag{2.9}$$

Value Iteration Value iteration algorithms combine policy evaluation and policy improvement. A simple update equation is the following,

$$\upsilon_{k+1} = max_\alpha \mathbb{E}[R_{t+1} + \gamma \upsilon_k(S_{t+1}) | S_t = s, A_t = \alpha],\ \forall s \in S \tag{2.10}$$

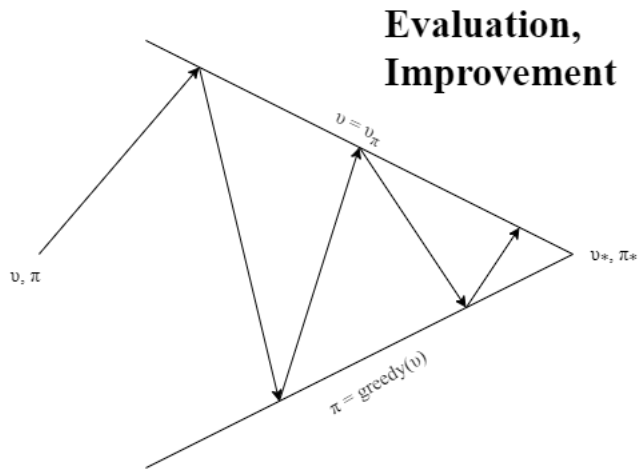

Figure 2.2: Reinforcement Learning (RL) policy iteration

2.4 Q-Learning

The most common learning algorithm in Reinforcement Learning is Q-learning [10]. A table which contains the values of all the states-actions pairs is constructed. Every time the agent completes some moves, the Q-learning table is updated using the Bellman equation.

$$Q(S_t, A_t) \leftarrow (1 - \alpha) \cdot Q(S_t, A_t) + \alpha \cdot (R_t + \gamma max_a Q(S_{t+1}, a)) \tag{2.11}$$

The working mechanism of Q-learning includes the following steps:

1. Initialize Q-table
2. Choose an action
3. Perform action
4. Measure reward
5. Update Q-table

After a lot of iterations a good enough Q-table is ready.

It is considered to be one of the very early breakthroughs in RL. It is an off-policy Temporal Difference (TD) control algorithm first introduced by Watkins in 1989.

This approach drastically simplified the algorithm's analysis and was the inspiration for the early convergence proofs.

2.5 The Deadly Triad

What is already known in Reinforcement Learning is the existence of a phenomenon called "The Deadly Triad". Every time a designer decides to simultaneously use the following three characteristics, the great danger of instability and divergence arises [11].

Function approximation It is a great tool for cases when the state space is larger than the available memory and computational resources. In such case, it is extremely useful to be able of predicting the value functions for states.

Bootstrapping This is the process of updating targets using existing estimates rather than using only actual rewards and complete sequence of returns.

Off-policy training In this type of training the algorithm is evaluating and improving a target policy that is different from the observational policy. With off-policy learning method, in order to update the policy, the best Q-value of the next state is used . The major issue with off-policy is the danger of not updating this value sufficiently often to prevent it from diverging[19].

The first RL architecture that solved the mentioned issue is the Deep Q-Network (DQN).

2.6 Deep Q-Network (DQN)

In order to tackle problems related with a large state-space, a neural network Q_ϕ is used for the estimation of Q^π, where ϕ is corresponding to the parameters.

DQN provides a set of three methods, discussed further below, to stabilize the learning process;

i clipping the TD-error,

ii using experience replay and

iii implementing two separate networks (the online network and the target network).

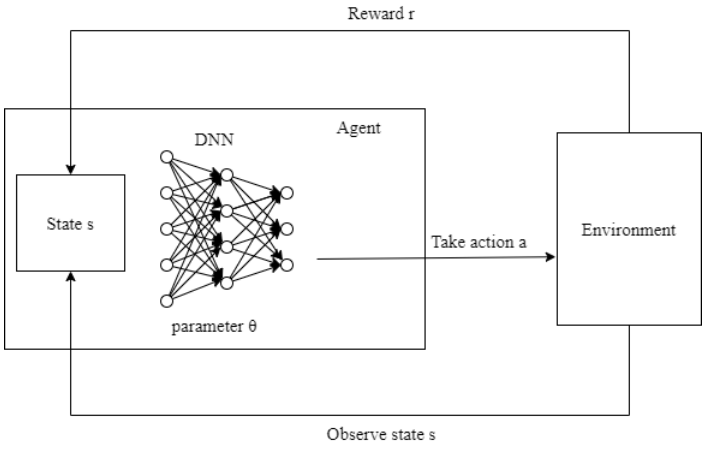

Figure 2.3: DQN and RL environment

Weights update The weights are updated according to the following relationship.

$$\theta_{k+1} \leftarrow \theta_k - \alpha \cdot \nabla_\theta \cdot \mathop{\mathbb{E}}_{s' \; p(s'|s,\alpha)} \left[(Q_\theta - target(s'))^2 \right] \Bigg|_{\theta=\theta_k} \tag{2.12}$$

Tackling deadly triad Deep Q-Networks refer to a combination of Q-learning and deep neural network function approximation. The major improvement of DQN over the Q-learning algorithms is the existence of a parallel target neural network, which is periodically synchronized with the main neural network [12]. The bootstrapped target value in the DQN is calculated in the target network. The periodical synchronization offers the advantage of providing more accurate and stable target values to the main network.

Mechanism For a great number of problems it is not ideal to represent the Q-function as a table that contains values for every combination of action α and state s.

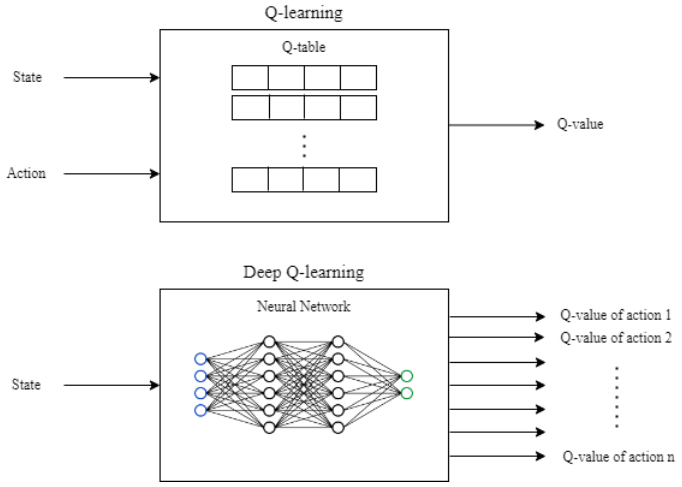

Figure 2.4: DQN and Q-learning comparison

Instead, it is preferred to train a function approximator, such as an ANN with parameters θ in order to estimate the Q-values.

$$Q(s, \alpha; \theta) \approx Q_*(s, \alpha) \tag{2.13}$$

The aforementioned is usually achieved by minimizing the following loss function,

$$L_i(\theta_i) = \mathbb{E}\left[(r + \gamma max_{\alpha'} Q(s', \alpha'; \theta_{i-1} - Q(s, \alpha; \theta_i))^2\right] \tag{2.14}$$

Experience replay If the loss is computed using just the last transition $\{s, \alpha, r, s'\}$ the equation above is reduced to standard Q-learning. At each time step, a data collection process is active. Therefore, the transitions are stored in a circular buffer called the replay buffer. Then, during the training procedure instead of just using the latest transition to compute the loss and its gradient, these are computed using a mini-batch of transitions sampled from the replay buffer.

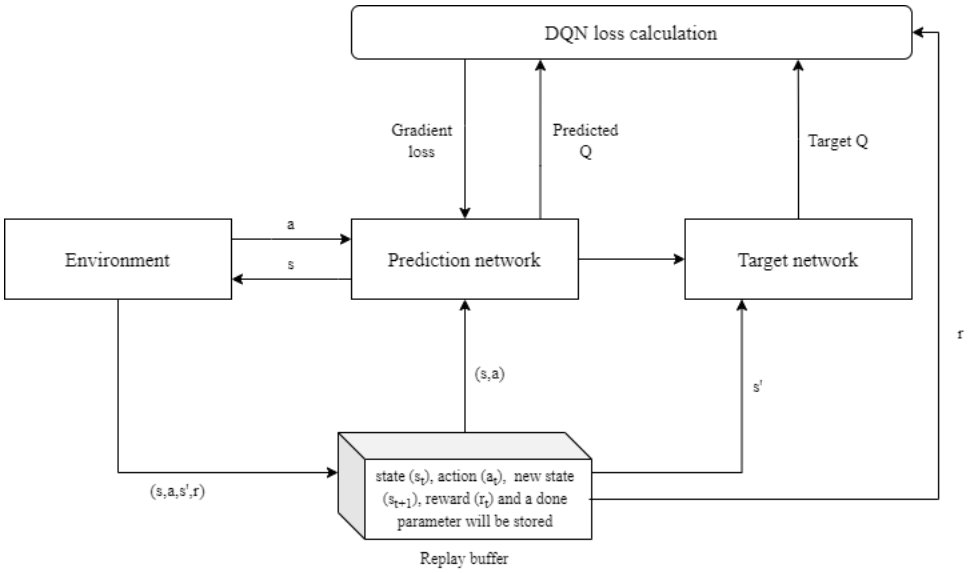

Figure 2.5: DQN internal structure

This methodology offers two great advantages,

- greater data efficiency because of reusing each transition in many updates
- and better stability due to use of uncorrelated transitions in a batch.

2.7 Rainbow Agent

The rainbow agent is actually an extension to the DQN agent. The exact structure of the rainbow agent can be found in the relevant article [13]. However, for the sake of completeness, the most important parts are discussed below.

Prioritized experience replay DQN samples uniformly from the memory buffer, which means that all samples are considered equal. However, not all data has the same amount of useful information. In order to improve this issue, the first proposal is the use of prioritized experience replay. Samples transitions according to the probability p_t which is relative to the last encountered absolute Temporal Difference (TD) error.

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \cdot \max_{a'} q_{\bar{\theta}}(S_{t+1}, \alpha') - q_\theta(S_t, A_t) \right|^\omega \tag{2.15}$$

Dueling networks It is specifically designed for the use of value based RL. It features two streams of computations named the value and the advantage stream which actually share a common encoder and are finally merged by a special aggregator. The above characteristics correspond to the following equation.

$$q_\theta(s, a) = \upsilon_\eta(f_\xi(s)) + \alpha_\psi(f_\xi(s), \alpha) - \frac{\sum_{a'} \alpha_\psi(f_\xi(s), a')}{N_{actions}} \tag{2.16}$$

where,

- $\xi,\ \eta,\ \psi$, are the parameters of the convolutional encoder.
- f_ξ, is the encoder
- υ_η, is the value stream
- α_ψ, is the advantage stream
- $\theta = \{\xi, \eta, \psi\}$, is their concatenation

Multi-step learning As already suggested by the book of Sutton and Barto, instead of using Q-learning which accumulates single rewards, a forward-view multi steps target can be implemented. The truncated n-step return from a given state S_t is given by the following equation.

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} \cdot R_{t+k+1} \tag{2.17}$$

Visibly, an alternative loss function must be defined.

Distributional RL It is possible to approximate the distribution of returns instead of approximating the expected return. In 2017 a model was proposed by Bellemare, Dabney and Munos that used a discrete support z in a vector with N_{atoms}, defined by:

$$z^i = \upsilon_{min} + (i-1) \cdot \frac{\upsilon_{max} - \upsilon_{min}}{N_{atoms} - 1},\ for\ i \in \{1, ..., N_{atoms}\} \tag{2.18}$$

The approximating distribution d_t at time t is defined on this support, with the probability mass $p_\theta^i(St, At)$ on each atom i, such that $d_t = (z, p_\theta(St, At))$. The goal is to update θ such that this distribution closely matches the actual distribution of returns.

A distributional variant of Q-learning is then derived by first constructing a new support for the target distribution and later minimizing the Kullbeck-Leibler divergence between the distribution d_t and the target distribution d_t'.

$$d_t' = (R_{t+1} + \gamma_{t+1} \cdot z,\ p_{\bar{\theta}}(S_{t+1}, \bar{\alpha}_{t+1}^*)),\ D_{KL}(\Phi_z d_t' || d_t) \tag{2.19}$$

where,

- Φ_z, is a L2 projection of the target distribution onto the fixed support z
- $\bar{\alpha}_{t+1}^* = argmax_a q_{\bar{\theta}}(S_{t+1}, a)$, is the greedy action
- $q_{\bar{\theta}}(S_{t+1}, a) = \mathbf{z}^T \mathbf{p}_\theta(S_{t+1}, a)$, are the mean action values in state S_{t+1}

In the exact same way that the parameters are frozen in the non-distributional DQN, they are frozen here too. The parameterized distribution can be represented by an ANN but with $N_{atoms} x N_{actions}$ outputs. A softmax function is applied in order to ensure the normalization process.

Noisy sets In order to ignore noisy streams, but at the same time be able to complete state-conditional exploration a noise layer transformation is being used.

$$y = (\mathbf{b} + \mathbf{W}\mathbf{x}) + (\mathbf{b_{noisy}} \odot \epsilon^{\mathbf{b}} + (\mathbf{W_{noisy}} \odot \epsilon^{\mathbf{w}})\mathbf{x}) \tag{2.20}$$

2.8 Feature Reduction

Principal Component Analysis (PCA) Principal Component Analysis is a feature reduction method that is used to reduce the dimensionality of large data sets by decreasing the number of variables used, ensuring that the smaller produced data set contains most of the information of the initial data set [14].

Minimizing the size of the data set results to an expense as regarding the accuracy, but contributes to overcoming issues related to over-fitting.

The PCA consists of the following steps:

1. Standardization
   This process is also known as Z Score Transformation. The purpose is to compare data for disparate distributions.

   $$Z = \frac{x - \mu}{\sigma} \tag{2.21}$$

   where,

   - Z, standard score
   - x, observed value
   - μ, mean of the sample
   - σ, standard deviation of the sample

2. Covariance matrix computation
   The goal of this step is to identify correlations between variables.

   For instance, the covariance matrix of a 3-dimensional data set is shown below:

   $$covariance\ matrix = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z \end{bmatrix} \tag{2.22}$$

3. Eigenvectors and eigenvalues of covariance matrix
   Through this process the principal components are identified. Principal components are new variables that are constructed as mixtures of the initial variables. These combinations are constructed in such way that the new variables are uncorrelated.

   Geometrically speaking, principal components represent the directions of data that correspond to the maximum amount of variance.

4. Principal components construction
   The first principal component accounts for the largest possible variance in the data set. The second principle component is calculated in the same way, but with the following condition: it must be uncorrelated with the first principal component. The procedure goes on.

Pearson's Correlation Coefficient Pearson's correlation coefficient is a measure of linear correlation between two sets of data.

The equation governing the results is the following.

$$r = \frac{\sum(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{2.23}$$

where,

- r, correlation coefficient
- x_i, values of the x-variable in sample
- y_i, values of the y-variable in sample
- $\bar{x}$, mean of the values of the x-variable
- $\bar{y}$, mean of the values of the y-variable

Spearman's Rank Correlation Coefficient Spearman's correlation measures the strength and direction of monotonic association between two variables. It is used complementary to Pearson's correlation.

The equation through which this coefficient is calculated is the following.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{2.24}$$

where,

- ρ, is the Spearman's rank correlation coefficient
- d_i, it the difference between the two ranks of each observation
- n, is the number of observations

2.9 Long Short Term Memory (LSTM) ANN

Goal Long Short Term Memory (LSTM) networks are a special category of Recurrent Neural Network (RNN) which are capable of learning long time dependencies [15]. Their greatest advantage is the ability of remembering information for long periods of time and are widely used when there is need for time-sequential prediction, or generally for time-series prediction.

Architecture In LSTMs the repeating module has four neural network layers interacting.

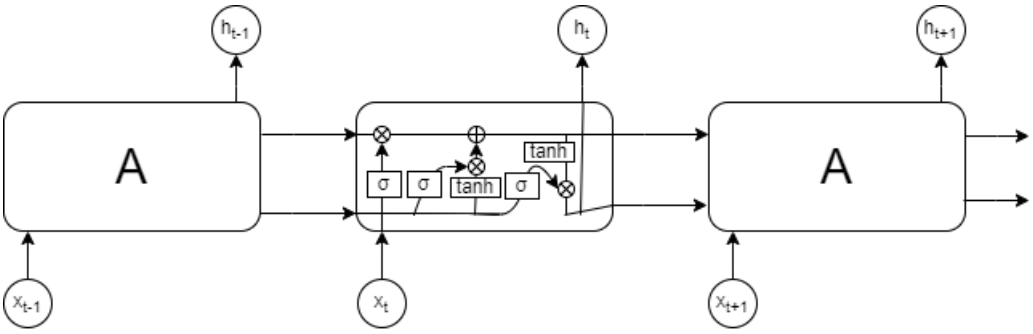

Figure 2.6: LSTM module

Equations The equations governing the LSTM are the following.

1. Forget gate layer
   This layer is responsible for deciding what information are going to be thrown away. It takes a look at inputs x_t and the input of the previous output h_{t-1} and outputs a number between 0 and 1.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.25}$$

2. Deciding which information will be stored
   This part is consisted of a sigmoid layer named "input gate layer" which decides which values will be updated. Next, there is a tanh layer responsible for creating a new vector of candidate values $\tilde{C}_t$ that could be added to the state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.26}$$
$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.27}$$

   The old state is multiplied by f_t in order to forget the things that were decided during the previous layer. Finally, these are the new candidate values transformed by how much it was decided to update each state value.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.28}$$

3. Deciding on the output
   Finally, there is a sigmoid layer responsible for deciding which parts of the inner state will be output. Then there is a tanh, that clips the data be between -1 and 1, multiplied by the sigmoid layer in order to maintain only the desired output.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.29}$$
$$h_t = o_t \cdot tanh(C_t) \tag{2.30}$$

Of course there are variants of the LSTM module, however, the aforementioned is all the background information needed for this thesis' demands.

2.10 Julia Coding Language

Julia Language was selected for coding. It first appeared on 2012 and it is steadily growing over the last couple of years. It is a high-level and high-performance dynamic programming language. It has been known for the excellent performance to lines of coding ratio and it is really promising for the future.

It is very well suited for computational science [16] and thus was the reason for being chosen for this application.

Some really useful packages were used. The most important of them for the completion of the coding part are mentioned below.

- Flux.jl
  A complete package for neural networks and training [17].

- ReinforcementLearning.jl
  Well designed package for handling all of the Reinforcement Learning procedures easily [18].

- DataFrames.jl
  A convenient package for handling large data sets.

- Hyperopt.jl
  A great package for tuning the hyper-parameters either by using the Random Sample procedure or Hyperband [19].

Chapter 3

Reinforcement Learning

3.1 Grid-world Environment

General In Reinforcement Learning, environment is the world where the agent lives and interacts. In this application it the sufficient simulation of the marine environment was demanded. This included the area where the ship (agent) was moving, the boundaries through which it was forbidden to pass and of course the weather conditions.

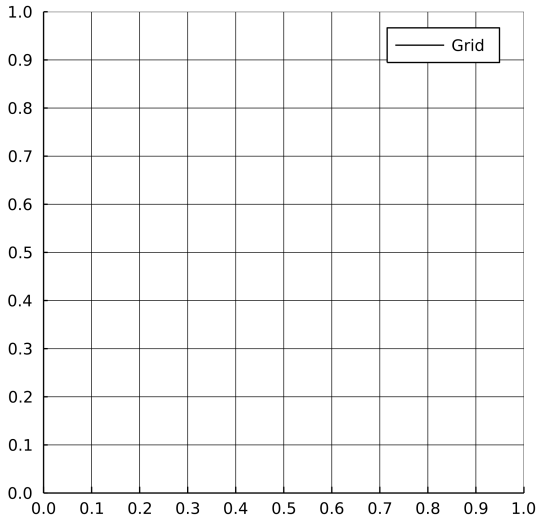

Figure 3.1: Squared grid-world

Limitation What is important to be stated is the fact that the grid-world's squares size was limited due to functional reasons, related with the consumption's predictive model reliability. Since, the LSTM model was trained using data points in 10-minute intervals, it was crucial to keep the time needed for each of the agent's moves approximately at the same duration - or even lower. Therefore, it was ensured that the predictive model produced accurate estimations.

Distance estimation To do so, the distances from one state to the other were calculated using the Haversine formula. This equation is considered common knowledge in navigation, to estimate great-circles distances between two points lying on a sphere, using their longitudes and latitudes.

$$d = 2 \cdot R \cdot atan2(\sqrt{a}, \sqrt{1-a}) \tag{3.1}$$

where,

$$a = sin^2\Big(\frac{lat_2 - lat_1}{2}\Big) + cos(lat_1) \cdot cos(lat_2) \cdot sin^2\Big(\frac{lon_2 - lon_1}{2}\Big) \tag{3.2}$$

Environment parameters In order for this environment to make sense, there were some parameters that were defined in an appropriate struct. These are presented in table 3.1.

Environment and ship related parameters		
Grid-world dimensions	50x50	
Available velocities	Minimum	6 knots
	Maximum	14 knots
Acceleration options	Deceleration	"-2 knots"
	Constant speed	
	Acceleration	"+2 knots"
Heading	all neighboring states	
Starting Point	(-6.733535,61.997345)	
Goal Point	(-6.691500,61.535580)	
ETA	6.5 hours	
Draught AFT	15.2 m	
Draught FWD	15.1 m	

Table 3.1: Environment and ship related parameters

3.2 Computationally Cost-effective Boundaries Construction

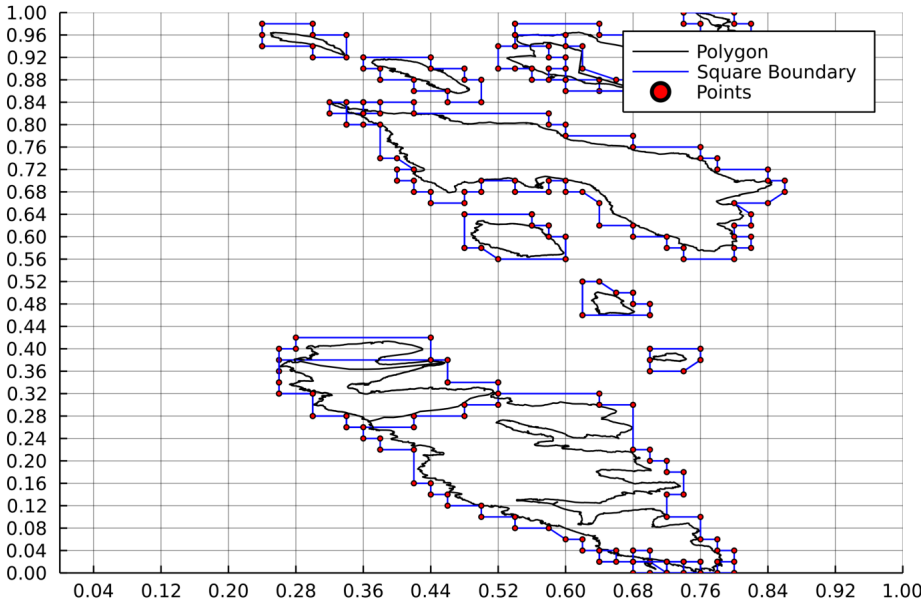

Figure 3.2: Polygon boundaries of reduced points

The goal was to check at every step if the agent was ending up inside any of the boundaries. Mathematically this meant checking if the final point was inside a polygon or not. It is clear that as the number of points of a polygon increased, the computations for the intersection check grew.

Clearly, when the edges of a polygon that represented a physical boundary, like an island, were reduced, the accuracy was lowered too. However, since the agent corresponded to a ship, it did not truly matter, as long as the initial polygon was included in the constructed one.

Since near lands and islands, normally there are shallow waters a ship does not approach, the aforementioned modification was not an issue at all. Conclusively, there could be made some concession in accuracy without any substantial concerns for the final boundaries design.

Also, it should be pointed out that the new boundaries were designed on top of the states/squares boundaries as is already depicted in figure 3.2.

The algorithm followed for the construction of the computationally cost-effective new boundaries was comprised of the following steps.

1. Generate each point's square boundaries

   In this first step, it was detected the square/state in which each of the points belonged. For every single one of these points, there were four points generated equal to the four edges of the square/state.

2. Delete all points lying inside the initial polygon

   In this step, all the generated points from step 1 that were lying inside the initial polygon were deleted. As already mentioned, it was crucial to only retain the points outside of the polygon.

3. Eliminate all duplicate points

   Since the goal was to achieve the lowest amount of points, there was no use in retaining duplicate points.

4. Sort the boundary points

   When the polygon's points were not sorted in order, according to the minimum distance between them, a strange polygon with intersected lines was produced, which definitely did not reflect the actual desired output.

5. Delete consecutive collinear points

   Consecutive collinear points (especially on the horizontal and vertical axis) had nothing more to offer than costly computations. That is why, a germane function was implemented in order to get rid of all those useless points.

   For three random 2-D points (x_1, y_1), (x_2, y_2) and (x_3, y_3). If the determinant of these three is equal to zero, then they are collinear.

$$det = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1 \cdot (y_2 - y_3) + x_2 \cdot (y_3 - y_1) + x_3 \cdot (y_1 - y_2) \quad (3.3)$$

The figure 3.2 displays how the final result looked like. The red points were the final points left on the newly constructed polygons. The black line polygons were the initial ones and the blue polygons were the final ones, after points reduction.

3.3 Weather Modelling

As regarding the weather modelling, there is a wide margin for improvement. Since, however the main topic of this thesis was not the weather modelling, but validating that the RL approach was capable of yielding the desired results, the simplest weather modelling option was implemented.

As a result a circular storm was inserted. Anywhere outside the storm, the sea state was considered calm. However, inside this stormy area, the significant waves height parameter had much greater values. Consequently, when the ship passed through this surface it drastically increased the fuel oil consumption.

Weather parameters	
Storm center	(0.85,0.4)
Storm radius	0.07
Bad weather significant waves height (m)	3.5
Calm weather significant waves height (m)	0.7

Table 3.2: Storm details

3.4 Agents

All of the following agents were tested extensively. The given values of the parameters below correspond to stable learning for a multiple runs of experiments.

3.4.1 Q-learning

The Q-learning agent also operated as proof of concept of the software development, since it was one of the easiest to implement. When the Q-learning agent obtained the desired behaviour, most of the times meant that everything up to this point was okay.

Q-learning agent	
Learning rate	0.5
Initial ϵ	0.99
Warm-up steps	10000
Decay steps	5000
Final ϵ	0.1
Trajectory	State, action, reward, terminal

Table 3.3: Q-learning agent hyper-parameters

3.4.2 Deep Q-Network

Inputs The input of a Deep Q-Network (DQN) was the state the agent was at. The state included the following information:

- Position on x-axis
- Position on y-axis
- Current speed

Normalization Since it is widely known that the normalization process significantly increases the learning speed and stability of neural networks, all the inputs were normalized with the appropriate values.

In the table 3.4, are shown the values according to which the normalization process took place.

Input normalization		
State parameter	Minimum	Maximum
Position X	1	50
Position Y	1	50
Speed (kn)	6	14
Significant waves height (m)	0	4.1

Table 3.4: Normalization for the DQN inputs

Parameters After extensive search and hyper-parameters calibration, the following is a set of values that successively led to the desired behaviour.

Deep Q-Network parameters		
Parameter	Approximating network	Target network
h_1	20	20
h_2	30	30
h_3	20	20
Activation function	relu	
Learning rate	10^{-4}	10^{-4}
Update frequency	4	100
Batch size	64	64
Update horizon	5	5
Minimum replay history	500	
Trajectory capacity	1500	
Discount rate	0.99	
Warmup steps	300000	
Decay steps	100000	

Table 3.5: Deep Q-Network hyper-parameters

3.4.3 Rainbow

The inputs and normalization process was copied from the Deep Q-Network (DQN) agent. So the aforementioned information are true for Rainbow Agent too. The used parameters are depicted in table 3.6.

Rainbow agent hyper-parameters In order for the RL agent to be learning, a hyper-parameters optimization was mandatory. After completing this procedure, the results as regarding those parameters were the following.

Raibow agent parameters		
Parameter	Approximating network	Target network
h_1	40	40
h_2	35	35
h_3	25	25
Activation function	relu	
Learning rate	$15 \cdot 10^{-5}$	$15 \cdot 10^{-5}$
Update frequency	6	325
Batch size	150	150
Number of atoms	51	
Minimum replay history	500	
Trajectory capacity	1280	
Discount rate	0.99	
Warmup steps	30000	
Decay steps	10000	

Table 3.6: Rainbow agent hyper-parameters

Chapter 4

Total Fuel Oil Consumption (TFOC) prediction

4.1 Data Pre-processing

This is the pre-process procedure which was executed at least two times. Once for the feature reduction method and a second time for the final implementation of the ANN. For figuring out the details of this process, the article [20] was extremely helpful.

1. Complete missing values

   In the first step of the data pre-processing, an effort was made in order to recover a number of data points by appropriately completing some individual missing values. Namely, if among the time-sequential data provided by the shipping company there were one, two or three missing values among existing values, these were replaced by the arithmetic mean of the neighbour ones.

2. Transforming to step distance

   The distance in the data set was measured as nautical miles covered from an initial reference point. The appropriate modification was made to get distance per agent's step, which is equal to the distance difference of every two sequential data points.

3. Choose sub-data set and delete missing values

For the features which were selected, for each data point there was a missing value, the entire time-row was deleted.

4. Split data according to draught

   Since the ship, the one which the data was drawn from, is a tanker carrying oil from oil-producing countries to oil-importing countries, about half of the voyages were performed in full-load condition, while the rest of them were performed in ballast condition. Hence, there were really no intermediate draught values. In favour of helping out the ANN's training, the data were split into three categories.

   - all data combined
   - ballast condition data, $T_M < 12.5\ m$
   - loaded condition data, $T_M > 12.5\ m$

5. Statistical outliers rejection

   In last part, the following steps [21] were followed:

   (a) Speed was used as a primary parameter $p_{primary}$

   (b) The speed data was split in groups of values with range 1 knot.

   (c) The entire data was grouped according to the aforementioned split.

   (d) Distance and Fuel Consumption were used as secondary interconnected parameters.

   (e) The mean values and standard deviations of the secondary parameters were calculated in each group G_i

   (f) A threshold outlier factor k was chosen.

   Finally, if the following inequality was fulfilled, then the data point was deleted.

$$|p_{2ij} - m_{p_{2ij}}| > k \cdot \sigma_{p_{2ij}} \tag{4.1}$$

   where,

   - i, corresponds to the group number
   - j, corresponds to the number of element

4.2 Feature Reduction

In order to decide on the final features, a feature reduction process was implemented. The initial inputs were the following. As regarding the speed and the distance features the over-ground values were chosen, since these reflected the actual distances covered to reach the desired destination. However, it would make more sense in future work to use through-water values for predicting the fuel consumption.

- Speed (OG)
- Draught AFT
- Draught FWD
- Distance (OG)
- Significant waves height
- Mean wave period
- Wind speed relative
- Current speed relative
- Time passed from last propeller polishing
- Time passed from last hull cleaning

Principal Component Analysis (PCA) The Principal Component Analysis led to the conclusion that there was no purpose in using more than six input features. The initial experiments implemented six inputs. However, this choice led to extreme over-fitting. In an attempt to tackle this issue, the choice of five inputs was eventually made.

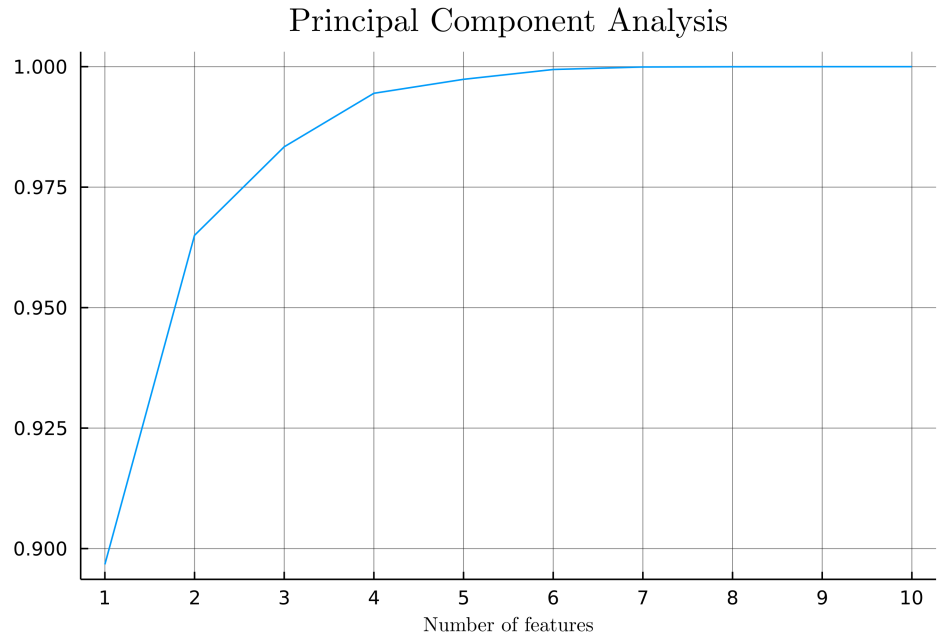

Figure 4.1: Principal Component Analysis

Correlation coefficients For the sake of discovering how the various variables were correlated one by one, it was extremely useful to calculate the following two metrics.

- Pearson's correlation coefficient
- Spearman's rank correlation coefficient

Hence, the linear and the monotonous relationships between the variables were clear.

A strong linear relationship was evident between the significant waves height, mean waves period and relative wind speed. Thus, only one from these three features was selected for the final model. Furthermore, there were some variables which seemed extremely important linear-wise for the fuel consumption prediction. These were; speed, draughts and significant waves height. So far, the results seemed in total harmony with the physical intuition.

There also existed a monotonous relationship between the before mentioned weather features. As regarding the importance for the fuel consumption prediction, according to this metric, the most important appeared to be; speed, distance, draughts and relative wind speed or significant waves height.

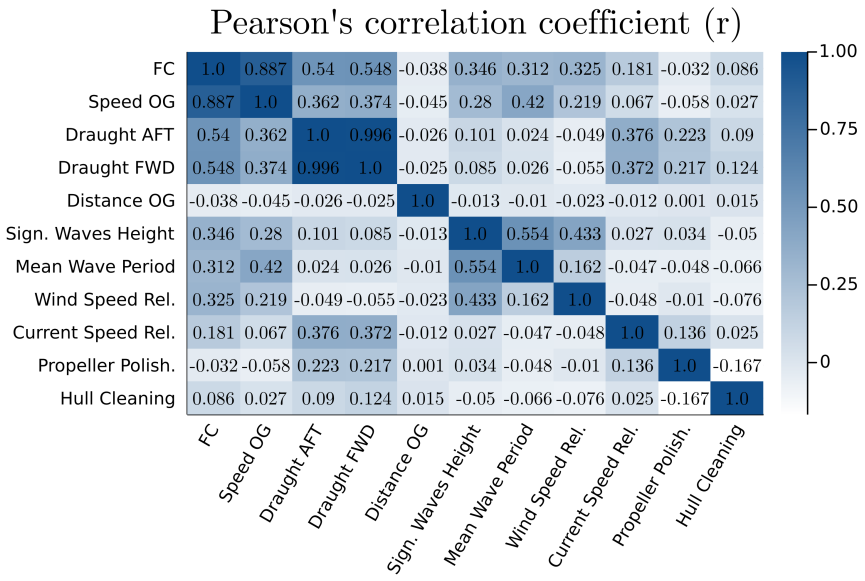

Figure 4.2: Pearson's correlation coefficient

By recognising the fact that significant waves height was a more important parameter than relative wind speed, both by the metrics produced from the data set on hand, and the physical reality.

Needless to say, extensive experiments were conducted. Ultimately, this process led to following the final selection:

- Speed OG
- Draught AFT
- Draught FWD
- Distance OG
- Significant waves height

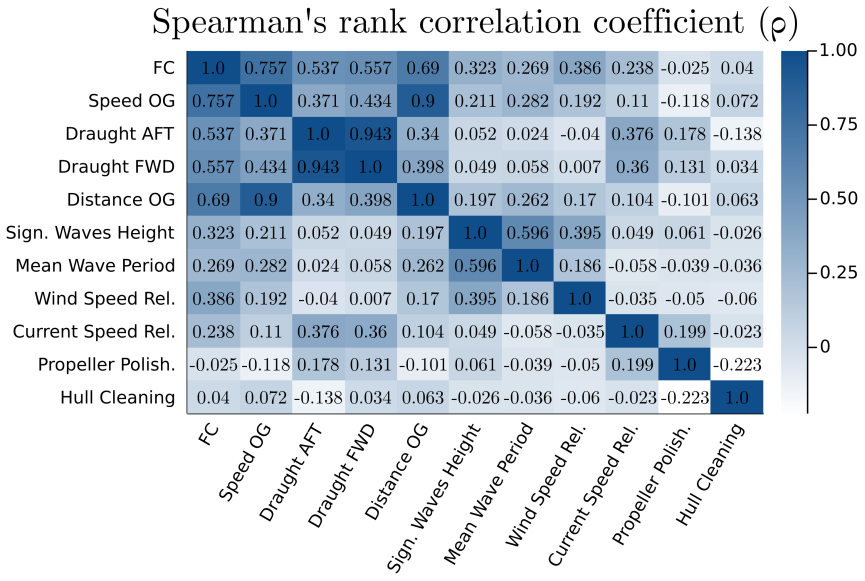

Figure 4.3: Spearman's rank correlation coefficient

4.3 Long Short Term Memory (LSTM) ANN Implementation

Final pre-processing procedure The pre-processing procedure was repeated and some more steps were implemented.

1. Z Score transformation

   This standardization procedure speeded up learning and led to faster convergence. This process constrained the inputs to almost the same range and distributed the values appropriately.

2. Time sequential data

   It was crucial to feed the LSTM ANN with time-sequential data. A relevant routine was implemented where the data was packaged in batches with guaranteed time continuity.

3. Moving average

   Last but not least, in addition to the outliers rejection procedure, a moving average function was performed to smooth out outlier values.

4. Final data discard

   All data points which size was less than the chosen sequence length of the ANN model were discarded.

5. Data split

   The data were split at 70 % of the data set. The majority of them was used for the training procedure while the rest 30 % was employed for testing.

6. Shuffling

   Finally the training and the testing data were shuffled.

Loss function The Huber loss function was implemented since it is less sensitive to outliers than squared loss functions. It is also appropriate for regression models.

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & if\ (y - \hat{y}) < \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta), & otherwise \end{cases} \tag{4.2}$$

Optimizer The ADAM optimizer was selected for this application because of the opportunity it offered for minimum oscillation, while at the same time it takes big-enough steps in order to avoid the local optimums along the way.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot \left(\frac{\partial L}{\partial w_t}\right) \cdot \upsilon_t = \beta_2 \cdot \upsilon_{t-1} + (1 - \beta_2) \cdot \left(\frac{\partial L}{\partial w_t}\right)^2 \tag{4.3}$$

where the parameters used were,

- β_1, β_2, decay rates of the average of gradients
- α, learning rate
- ϵ, small constant in order to avoid division by zero

Finally, the weights update was accomplished using the following formula:

$$w_{t+1} = w_t - \hat{m}_t \cdot \left(\frac{\alpha}{\sqrt{\hat{\upsilon}_t} + \epsilon}\right) \tag{4.4}$$

where,

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \cdot \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4.5}$$

ANN architecture The final architecture was an LSTM ANN compromised of 65 nodes connected with a dense layer consisted of 65 nodes leading to one predicting node.

Chapter 5

Challenges

5.1 Local Optimums

This was one of the most persistent issues faced. When the model was not capable of learning, it was because one of the two problematic cases. The first one corresponded to falling to a local optimum and sticking to it (Figure 5.3). The second one involved the extensive oscillation, not necessarily around the desired solution (Figures 5.4, 5.5).

The second issue occurred mostly due to the extreme DQN hyper-parameters sensitivity and will be discussed extensively in the next section.

The first issue, although, it partially arose due to poor hyper-parameters calibration as well, had a further explanation.

The initial approach as regarding the reward process design, was the following; the agent got a punishment at each step proportional to the total amount of fuel burnt. Moreover, when total time did surpass the estimated time of arrival, the agent would get a big punishment at the state it was in at the moment and the episode ended. However, this approach did not assist the agent to realize that being closer to the arrival port by the end of time, was a much better result than being far away.

By altering the reward approach, the system dynamics seemed to be more stable. Even when the agent fell into local optimums, these localities made more sense than the ones before. In the figure 5.1, a common local optimum fall result, before the changed reward approach, is displayed.

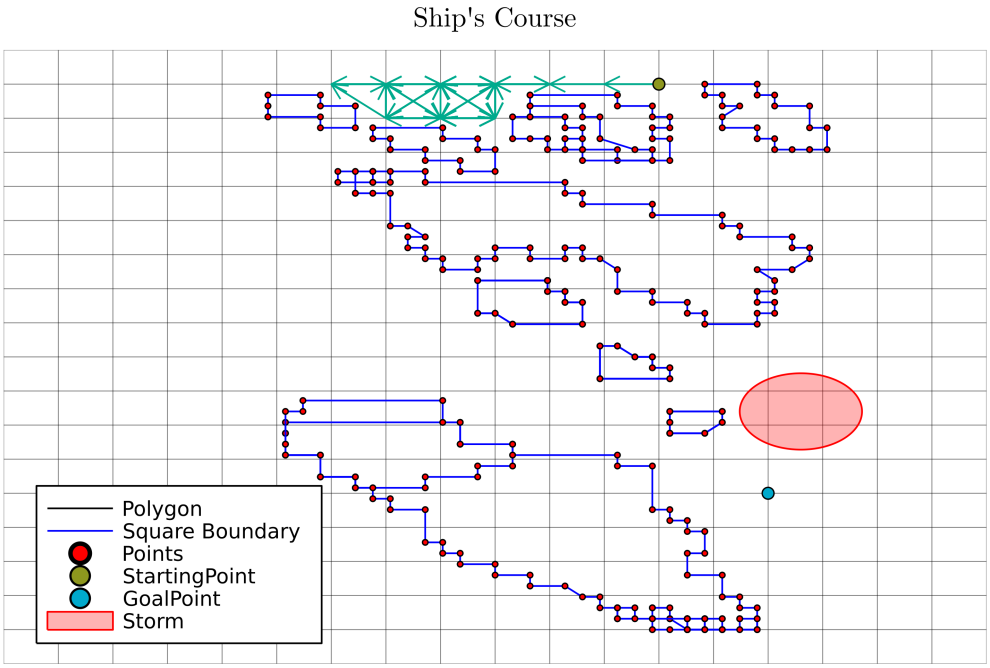

Figure 5.1: Local optimum before the changes in rewarding approach

The final reward approach The proposed and working solution was the agent to be punished, when the time ended, not by a constant number, but specifically by a reward proportional to the distance for the final destination. In figure 5.2 a common local optimum fall, after the aforementioned realization, is displayed.

5.2 DQN Hyper-parameters Sensitivity

One of the most confusing things to overcome during this thesis was the following issue; even in the simplest versions and environments, the DQN agent did not seem to be capable of learning. Until it was later realised, how sensitive it really was as regarding the hyper-parameters. If these parameters were not tuned in the exact needed range, the agent did not seem to learn.

The most common outcome during all these trials was the following. The agent in a condition where it was incapable of learning, kept falling into a situation where it did not ever manage to reach the arrival port. This

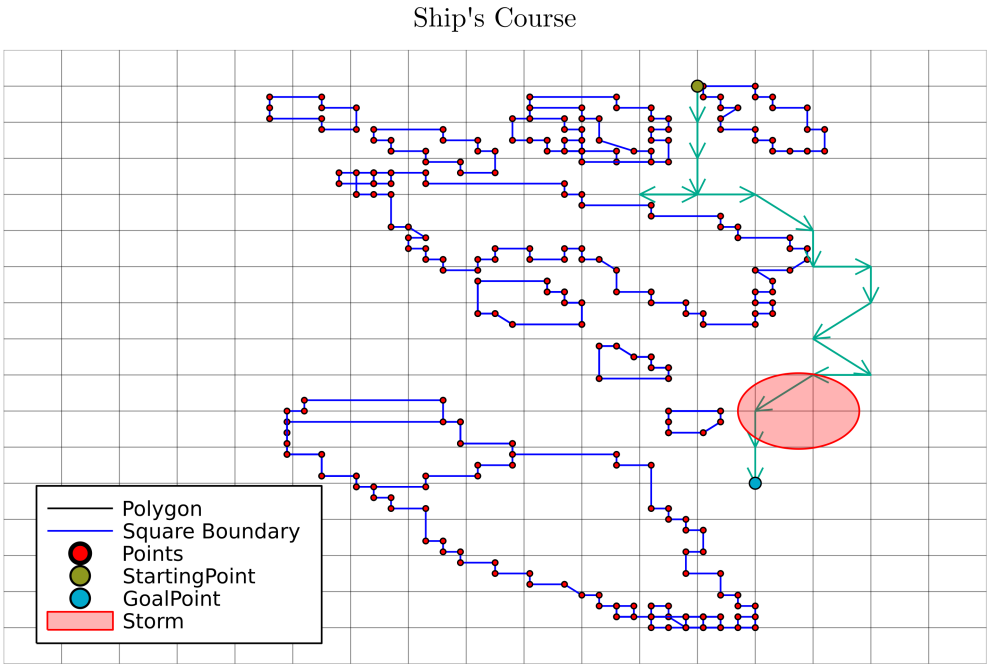

Figure 5.2: Local optimum after the changes in rewarding approach

occurred even when the agent was forced to discover that the best solutions were always those where the arrival to the goal point is achieved.

Also, the greater the state space was, the narrower the range of stable hyper-parameters got. This led to the need of extensive hyper-parameters calibration and finally to the introduction of the more sophisticated rainbow agent.

Intrinsic randomness Additionally, there was a significant number of times when the agent did seem to learn. However, if the process was restarted with the same hyper-parameters, the agent could not achieve the desired behaviour.

This phenomenon was the result of an unstable set of hyper-parameters along with the intrinsic randomness inserted in the model due to the initialization of weights and biases of the neural networks. In order to resolve those issues, an extensive hyper-parameters tuning was compulsory, ensuing to finding the most stable hyper-parameters.

In the figures 5.3, 5.4 and 5.5, there are examples where the above problems are depicted.

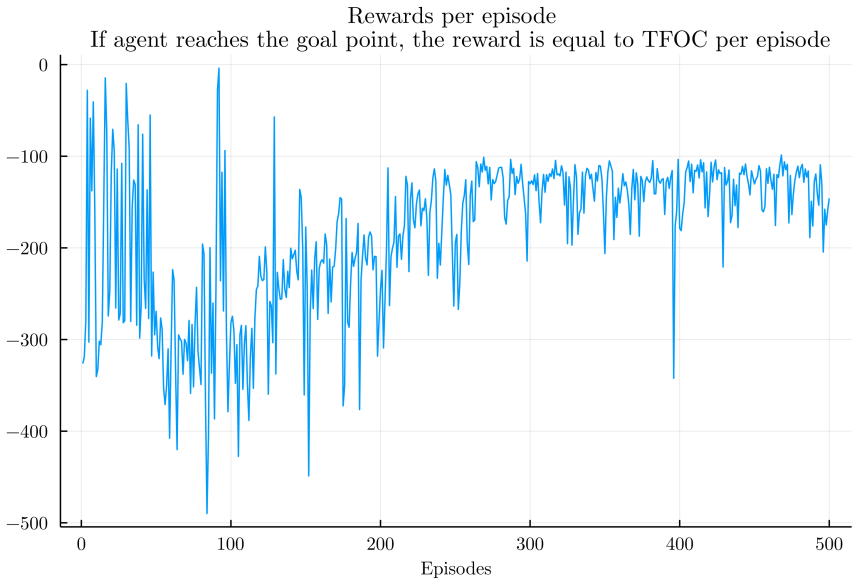

Figure 5.3: Failed hyper-parameters tuning attempt: Steady local optimum

5.3 Rewards of Different Magnitudes

This challenge was caused as a consequence of a miscalculation in the rewards distribution. The initial approach included a punishment at each step equal to the mass of fuel burnt (which was approximately 0.15), while a huge punishment was assigned every time the Estimated Time of Arrival (ETA) was surpassed. The different order of magnitude between these two punishments is clear.

As a result, this approach truly caused issues in the neural network updating procedure. It forced the agent to "ignore" its fuel consumption. It was indicated by the results that the agent was "caring", when stable hyper-parameters where used, only about the greater rewards of the two.

It became evident that the order of magnitude should be kept constant since a neural network's training procedure was implemented. The distributed rewards were clipped to the (-1,0) span.

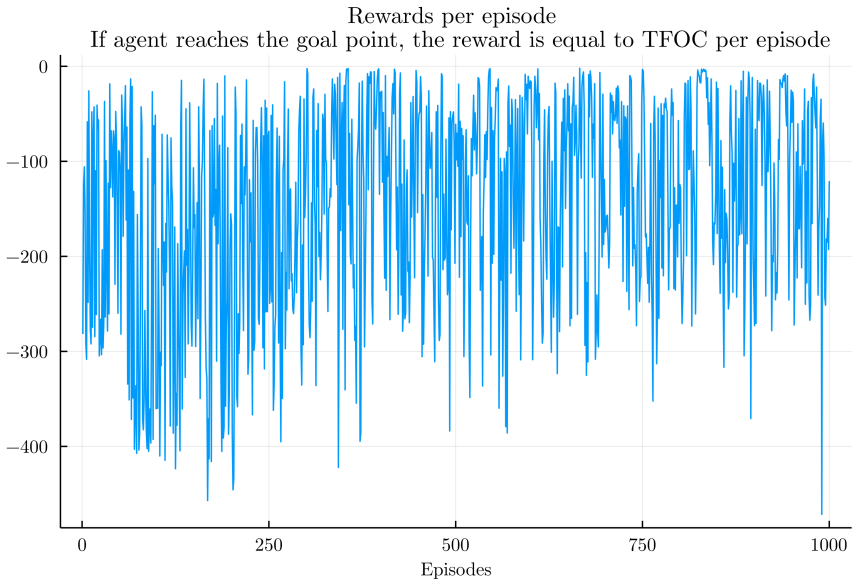

Figure 5.4: Failed hyper-parameters tuning attempt: Extensive oscillation

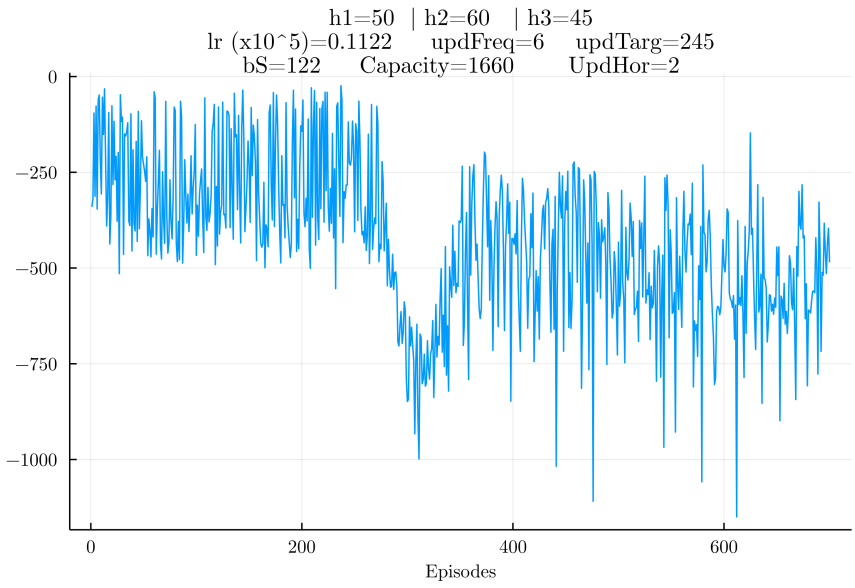

Figure 5.5: Failed hyper-parameters tuning attempt: "Ignoring" better results

5.4 Dense Environment

Since the LSTM model was responsible for predicting the reward values - which were equal to the amount of fuel burnt- each step should roughly correspond to a 10-minute long movement, in order to be accurate.

The initial viewpoint was the implementation of "if" statements, which indicated when the 10 minutes had passed. However, it was clear later on that a movement from one state to another, should not approximately take 10 minutes, but much lower. Hence, in the final approach a denser state space was constructed; each step was taking approximately one minute. This resulted in attaining greater run speed; for one simulation. Furthermore, the LSTM estimations were monitored to be closer to 10 minutes and thus the predictions were more accurate. Finally, it allowed for a greater variety of moves that a ship may execute during a 10-minute period.

Chapter 6

LSTM Performance

6.1 LSTM Over-fitting

In order to tackle this issue of over-fitting and also manage to generalize appropriately, it was required to reduce the input features to the bare minimum, while at the same time choosing the best ones.

Physical understanding of the problem, is not always the way to go in such models where statistical dependencies are more important. The feature inputs that were finally chosen are the following.

- Speed OG
- Draught AFT
- Draught FWD
- Distance OG
- Significant waves height

Since the measurements' frequency was 10 minutes and normally ships during voyages keep a constant and steady speed, a logic hypothesis would be that the distance is irrelevant. Since most of the times distance can be calculated by the speed scaled with time. The Principal Component Analysis indicates the use of five features and thus another weather feature could be used. However, this selection provided really poor results as regarding the performance on unseen data.

So contrary to common sense, choosing the distance as input feature provided much better results.

6.2 Final Prediction Results

Training parameters selection The training hyper-parameters used for the ANN model were the following. The split point refers to the percentage of data points used for training.

Training parameters	
Sequence length	3
Split point	70 %
Batch size	12
LSTM nodes	65
Dense nodes	20
ϵ (learning rate)	10^{-4}
Epochs	1000

Table 6.1: Parameters for LSTM ANN training

Error The average error was kept under 5 %, and the maximum error did not exceed 80 %.

Tracking A better understanding of the results can be achieved through the actual plotting of the tracking.

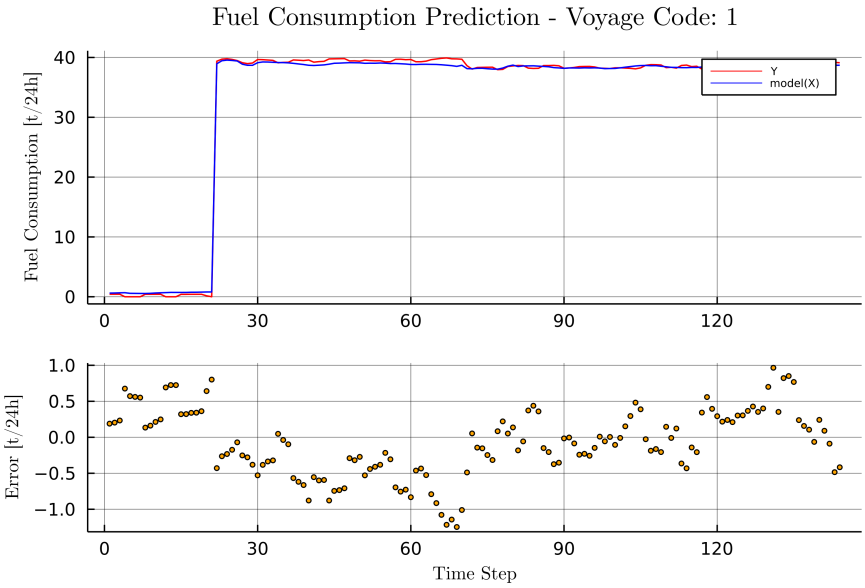

Figure 6.1: Prediction time-series for voyage 1 (training data)

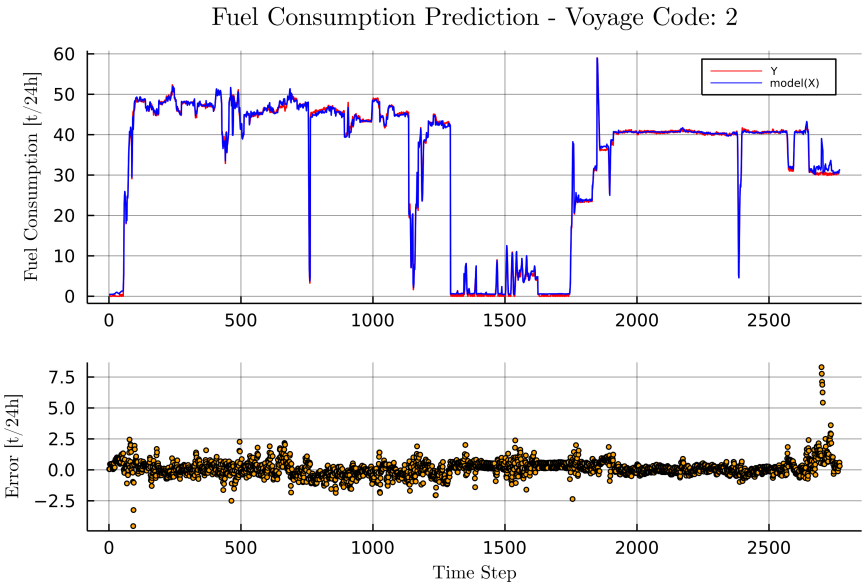

Figure 6.2: Prediction time-series for voyage 2 (training data)

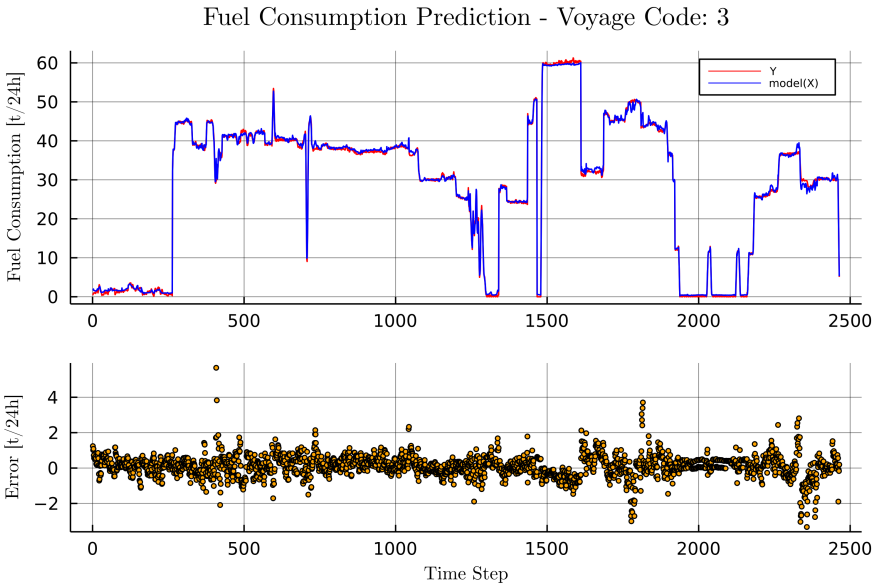

Figure 6.3: Prediction time-series for voyage 3 (training data)

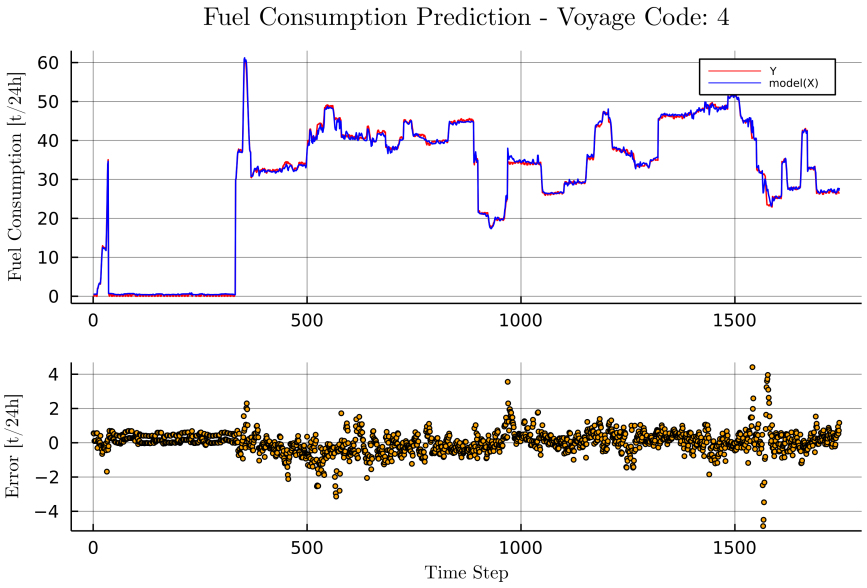

Figure 6.4: Prediction time-series for voyage 4 (training data)

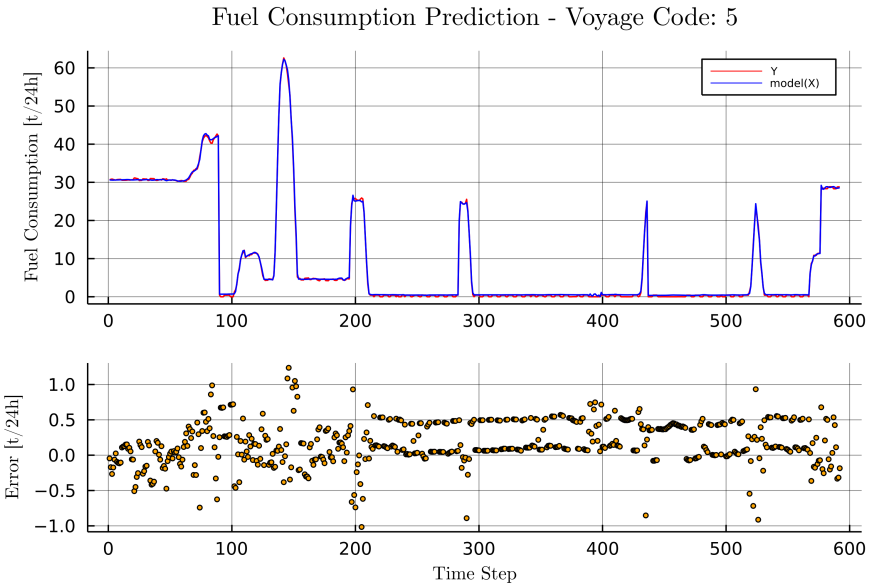

Figure 6.5: Prediction time-series for voyage 5 (training data)

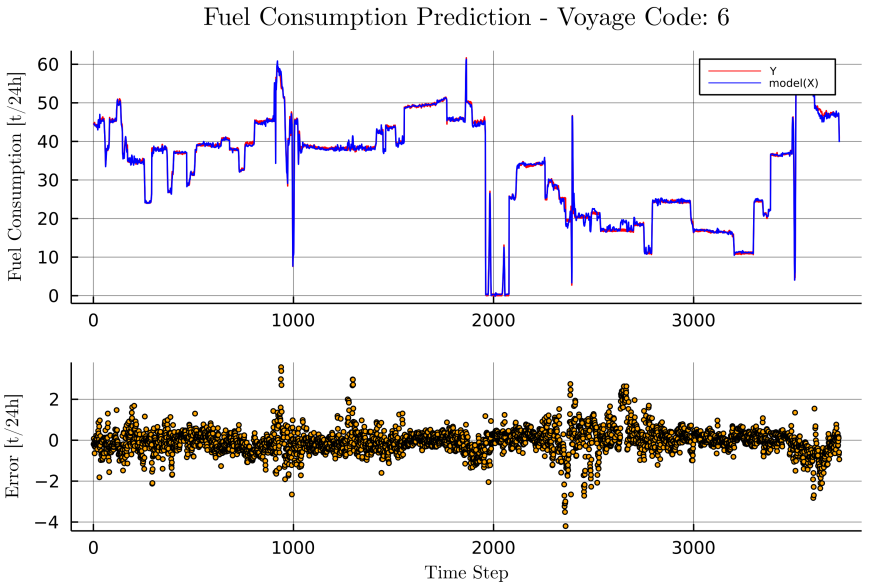

Figure 6.6: Prediction time-series for voyage 6 (test data)

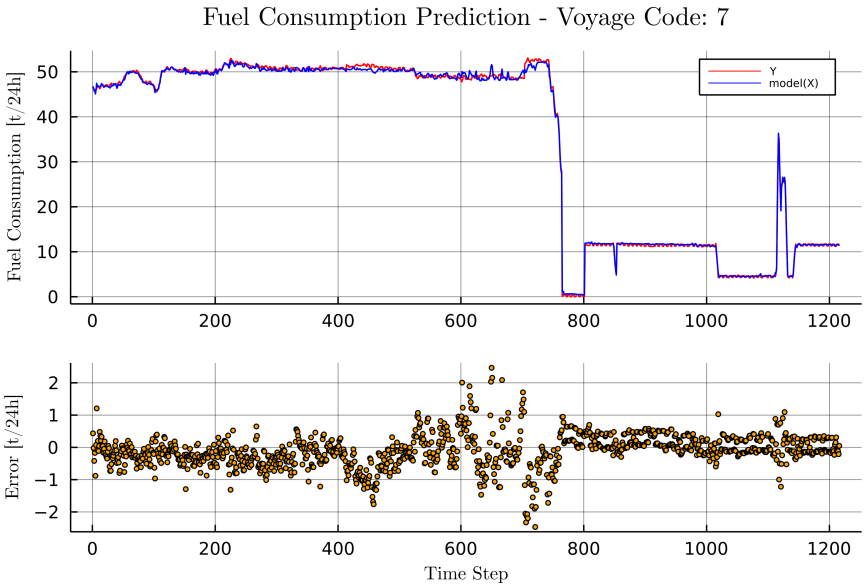

Figure 6.7: Prediction time-series for voyage 7 (test data)

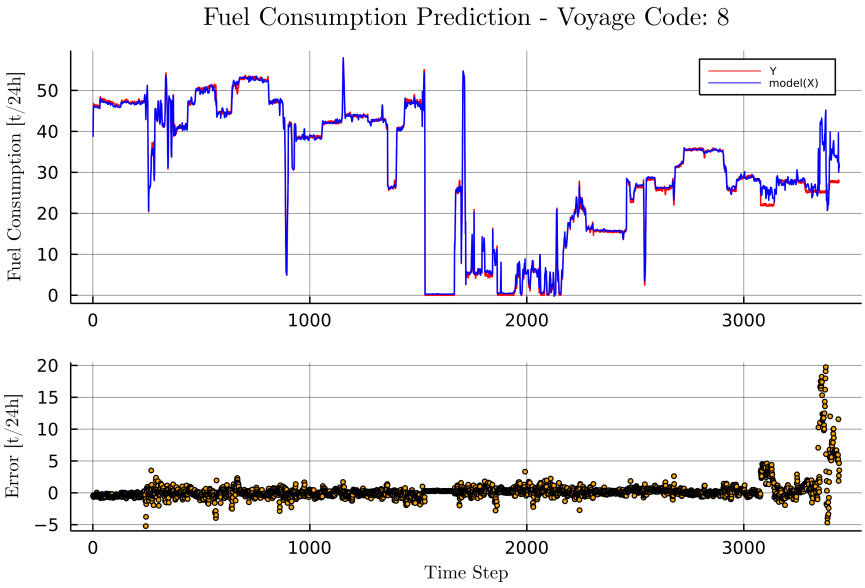

Figure 6.8: Prediction time-series for voyage 8 (test data)

Chapter 7

Comparison

7.1 Q-learning Agent

The Q-learning agent was the simplest implementation of a RL agent. It offered speed, especially if it was combined with a learning option like Temporal Difference (TD). However, it was not the best option when a large state space was required. That was because in order to evaluate the different state-action pairs, it was mandatory to experience them all at least once.

At experiments where the state space was kept small enough, the Q-learning agent was clearly superior to the other two. It was both faster and more stable. However, when the state space got increased it seemed, as expected, to lose its powers.

The simplistic but powerful approach the Q-learning offers, makes it seem like an undeniable choice when the problem's requirements allow it. In this certain problem, especially if at some point scales to environments of greater area, a dense state space is required. Hence, overall, definitely this agent is not the best alternative.

Next, there are a typical plot of the rewards compared to the number of episodes and the desired output produced by the agent. Also, a velocity profile for one of the best solutions is provided.

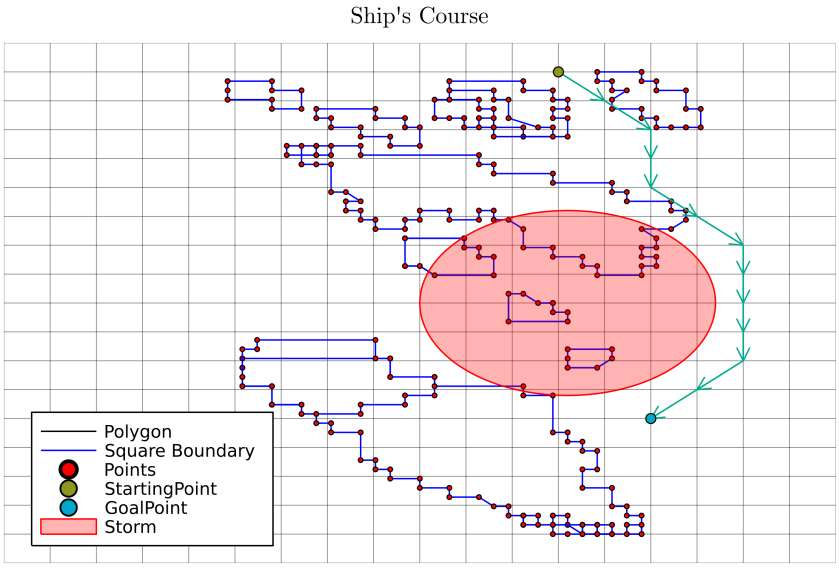

Figure 7.1: Route followed by Q-learning agent after training

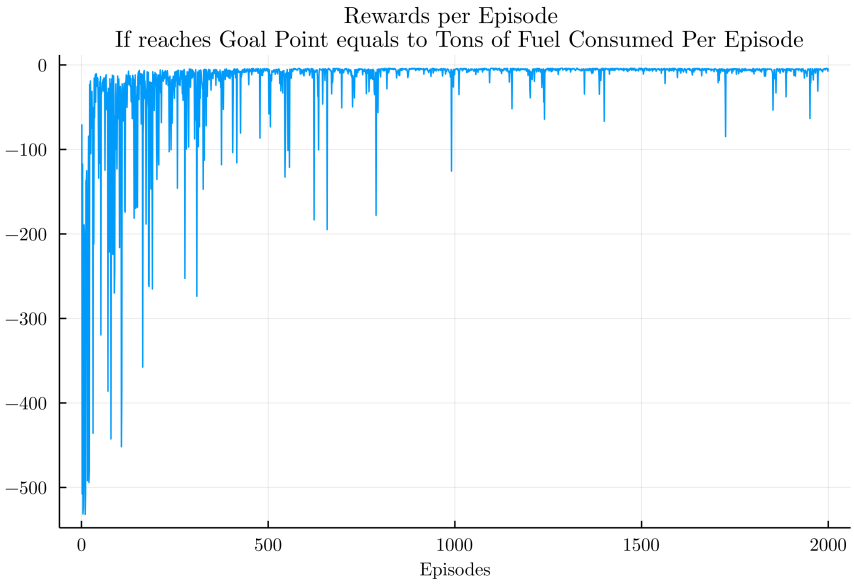

Figure 7.2: Rewards achieved by Q-learning agent

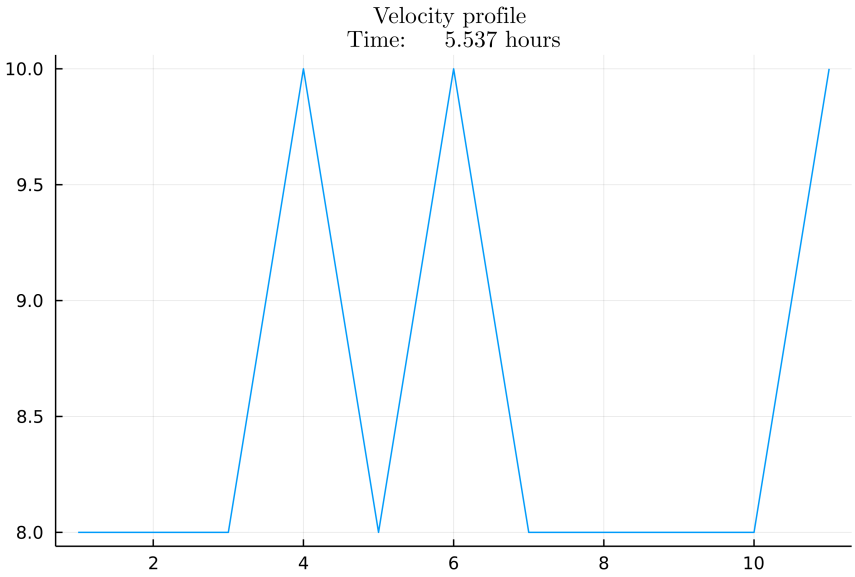

Figure 7.3: Velocity profile by Q-learning agent

7.2 Deep Q-Network (DQN) Agent

This is the indicated agent when a large state space is required. Although initially seemed the one method to go, however it was discovered that it was extremely sensitive to the hyper-parameters. Every time the state space grew larger, the hyper-parameters range where the learning is achieved seemed to become narrower.

Definitely, a more sophisticated method than random sampling was required for hyper-parameters tuning. However, the DQN agent's learning process seemed to be a lot more stable than the Q-learning's for large state spaces.

Below, there are a plot of rewards compared to number of episodes, the best solution of the learning's procedure and a velocity profile.

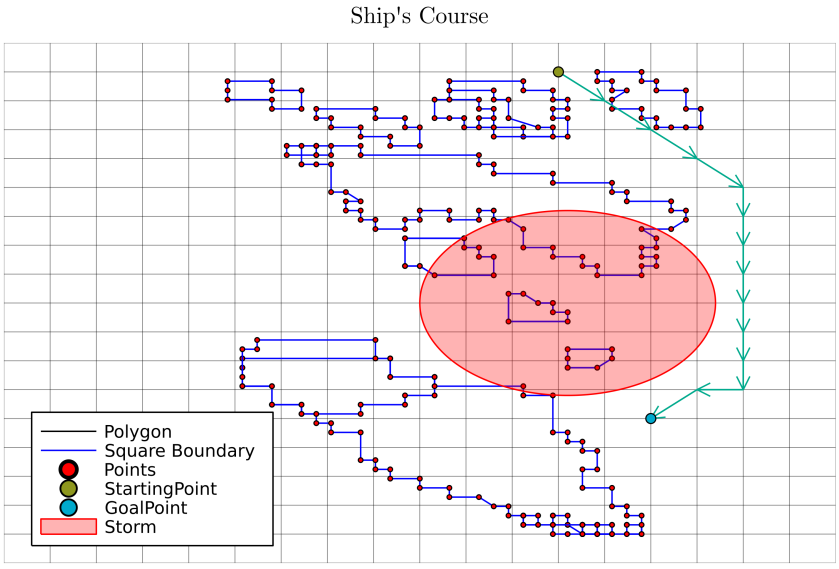

Figure 7.4: Route followed by DQN agent after training

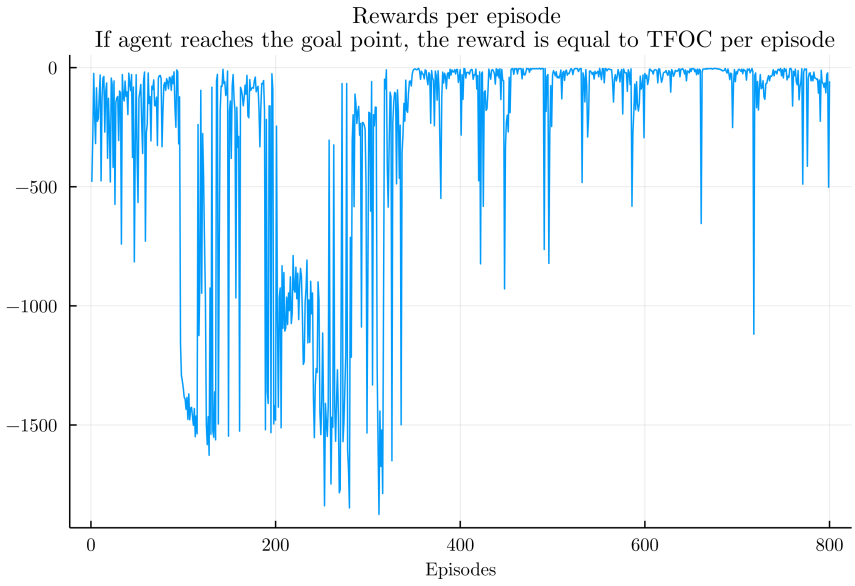

Figure 7.5: Rewards gathered by DQN agent

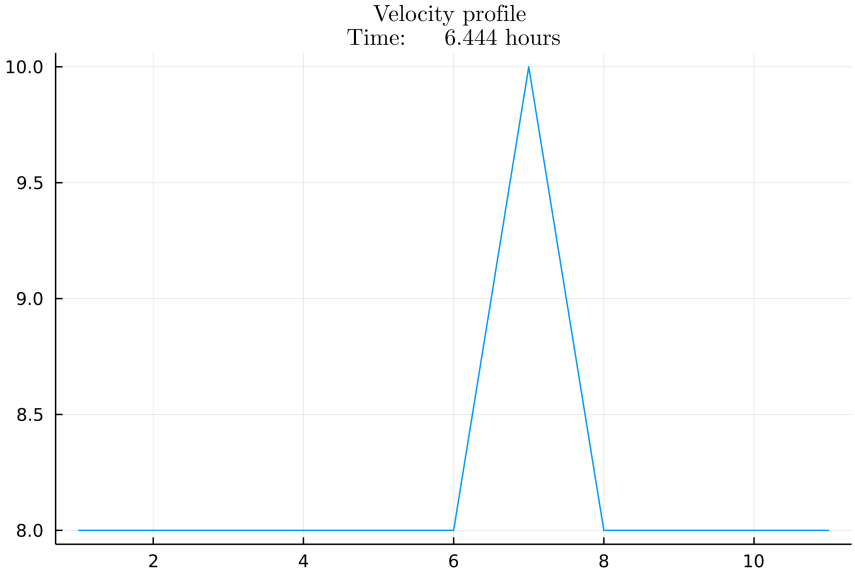

Figure 7.6: Velocity profile by DQN agent

7.3 Rainbow Agent

The rainbow agent includes all the latest improvements on the field of RL agents. Instead of approximating expected returns, it approximates probability distributions. It is certainly more sophisticated.

To all intents and purposes, it offered a more stable learning procedure compared to the DQN agent. The range of stable hyper-parameters seemed to be greater than this of the latter. Also, after many experiments it was realized that the oscillations were not as intense as the DQN agent's.

Overall, it definitely proved to be the best option for this application while at the same time maintains all the pros of the DQN agent.

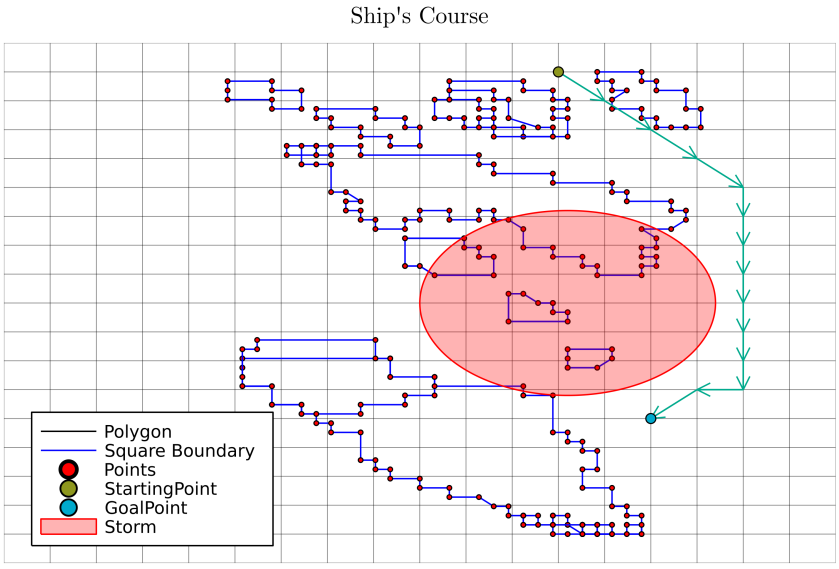

Figure 7.7: Route followed by rainbow agent after training

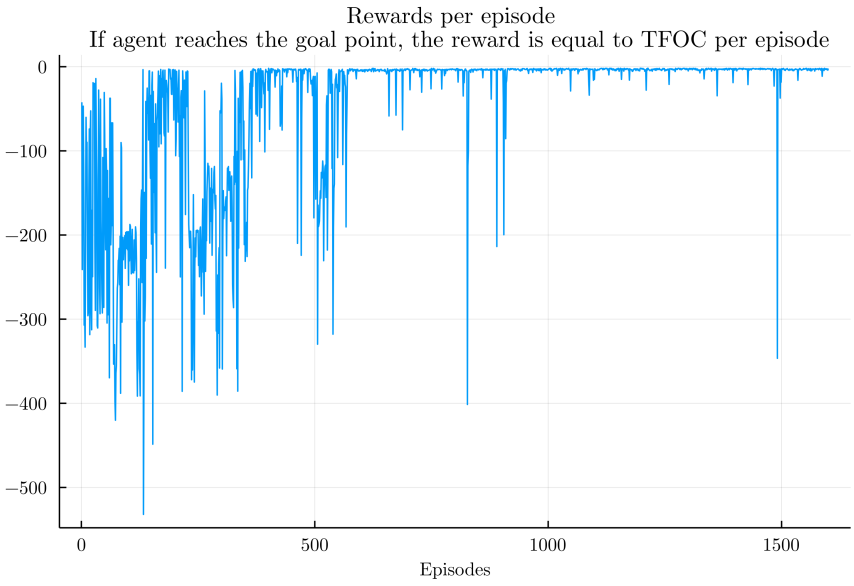

Figure 7.8: Rewards gathered by rainbow agent

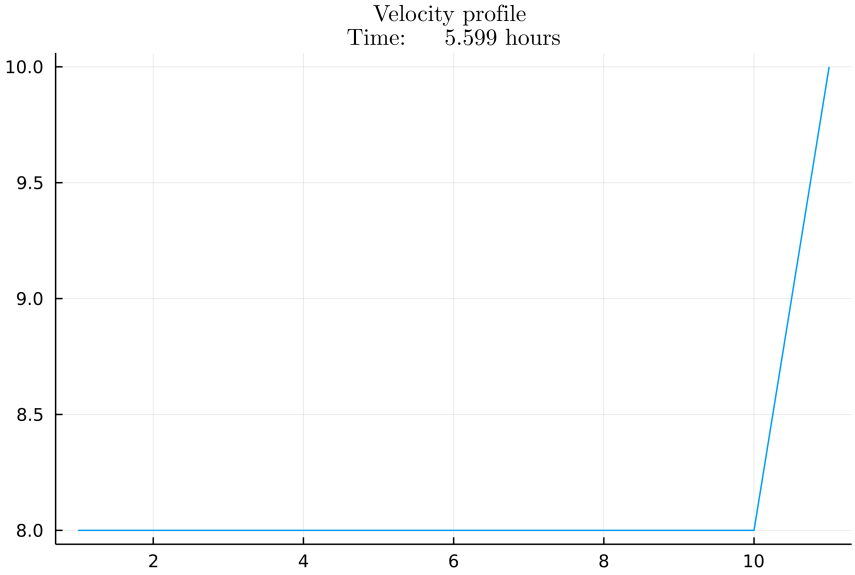

Figure 7.9: Velocity profile by rainbow agent

Chapter 8

Summary, Conclusions & Recommendations

8.1 Summary

Overall, if this project could be characterized for what it is, it is probably a solid start for an overall project of minimizing TFOC using Reinforcement Learning. There are various things that have to be included, which are all mentioned in the next chapter as recommendations. This work aims to work as a foundation for future research projects on this subject.

Agents The use and comparison of different agents seems to have declared a clear winner between those three for the requirements of this project. Rainbow agent seemed to be both the most stable and the most efficient in handling large state-spaces. Q-learning would be the definite answer only in the case where a small-sized grid is chosen.

Julia language The Julia Lang was a great opportunity since it is really promising for the future. There is a great community, with people really eager to help and grow together. However, the language itself right now has some corners that are hard to avoid and can cause a lot of discomfort in an attempt to develop new work. Since this research project was something fresh, on terms of coding this language proved to be both a curse and a gift. The running time was significantly faster and the coding significantly shorter than other language; such as Python.

8.2 Conclusions

LSTM implementation The LSTM implementation looking back was not necessary for the length and scope of this approach. It could be done, with a simple Artificial Neural Network (ANN) which would be easier to implement. However, the research project was from the start faced with a product-oriented perspective, without any prior knowledge as to how far it would get. In the end, I personally am really glad for all the knowledge I got from this ambitious approach and I really enjoyed it, despite the adversities caused by thinking it as a whole product. However, it needs to be stated, that if anyone just tries to replicate the results up to this point, it is possible to do so with a simple ANN or just some simplified functions punishing appropriately for each parameter (speed, distance, draught, weather).

Also, what needs to be mentioned is that since ANN are somewhat statistical models, it is common to get unreliable results for some value ranges.

What made great difference The computationally cost-effective approach as regarding the boundaries really made the runtime a lot faster, without sacrificing anything of value. Also, clipping the reward values for the DQN agent's loss function was key to stabilizing the results. This whole project, could not be done without the use of powerful GPUs, provided from the Laboratory of Marine Engineering (LME) and the excellent Julia package for CUDA [22].

8.3 Recommendations

8.3.1 Weather Model Improvement

The weather model could be further improved step-by-step in order to reflect realistically the sea conditions. The immediate next step could be adding weather storms with Gaussian distribution models. The final step is using live weather data and storm forecasts.

As regarding the available models, live data from a well-known provider seems to be the best and most trustworthy source. However, there is also the option, at least as an intermediate step of using a neural network in order to predict the weather forecasts. This might be an attractive approach, depending on the goals of the researcher.

8.3.2 Speed Parameter

Since the agent is taking decisions no different than those a ship's captain would take, it is crucial to handle the same parameters as a human would. It is not possible to choose the speed over-ground a ship will cover. What the captain really handles usually is the instructed speed. Therefore, the best option would be the development of a model which predicts the speed over ground using as inputs the instructed speed and the weather conditions. The exact same procedure is recommended to be applied for the distance over ground.

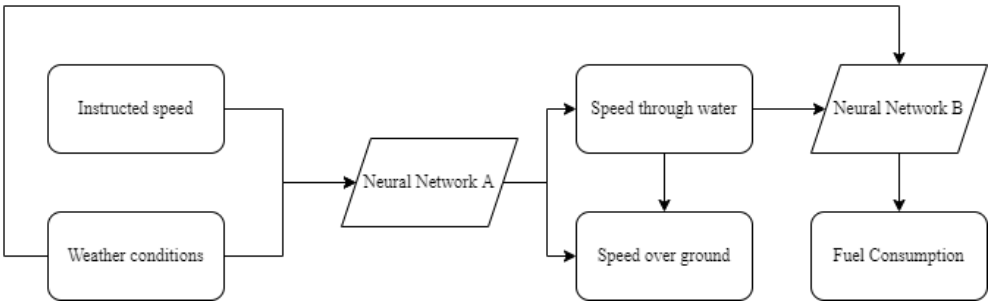

Figure 8.1: A schematic for improved implementation of speed

Also, what is needed to be explored is the velocity control which is dependent on the Estimated Time of Arrival (ETA) in order to minimize the fuel consumption. In this thesis, since changing velocities between the states due to the simplified weather model is not required, the ship is constantly using the average needed speed in order to achieve the objective.

8.3.3 Engine Data for the Prediction Process

All the available models using as inputs weather data, without any clue about the engine data seem be inaccurate up to 10 %. In an attempt to tackle this issue, it could be examined, if the addition of engine data into the regression models for the fuel consumption offers greater accuracy. It is not an unusual policy for ship captains to keep steady values for the revolutions of the engine throughout different segments of a voyage.

8.3.4 Live Action Methodology

When a long voyage is to be made, lasting three days or even more, it is not possible to decide on a route beforehand. That is because of the change in the weather conditions. So the recommended procedure would be to decide on a route beforehand and re-training the agent every x hours in order to achieve a close to optimal result.

For instance, in the example in figure 8.2, the route is decided beforehand when there is information for a storm in the first part. Ship follows this route, but after a while information for a new storm at the second part arise. The agent is being retrained-specifically for the remaining part. It learns to go around the storm and thus achieve the minimum fuel oil consumption.

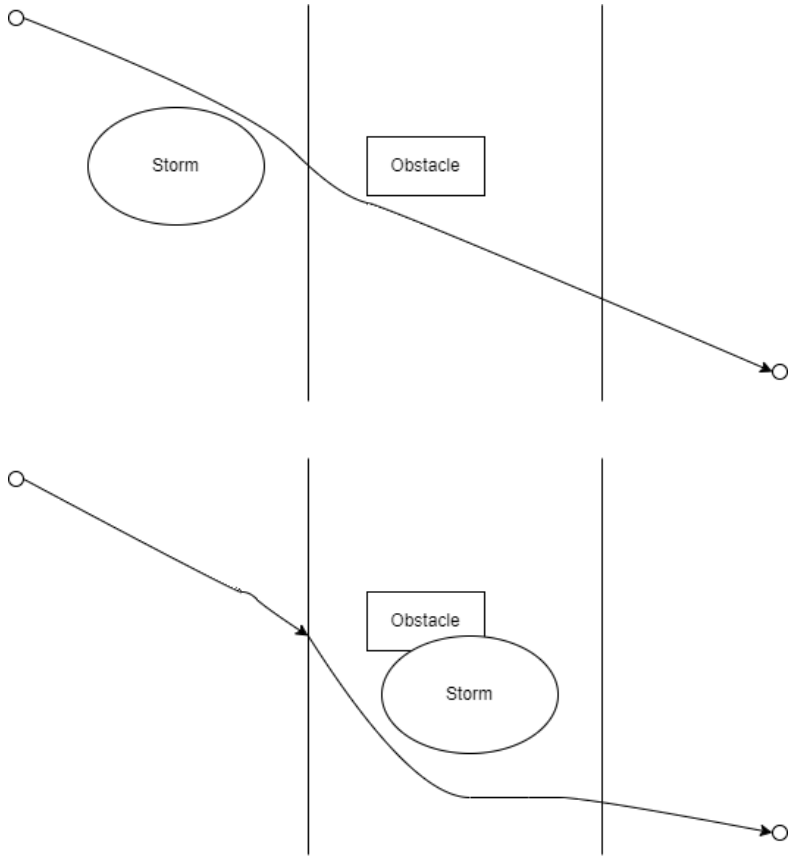

Figure 8.2: An example of live action methodology

8.3.5 Dynamic Change of Draughts and Trim

Throughout a ship's voyage there is change both in draughts and trim. These changes account up to 1-2 % of the total ship's displacement, which is not a negligible amount. This is partially because great amounts of fuel are being burnt but also due to other, less significant, factors as well. This phenomenon results in differences that do alter the consumption and should be taken into account. What is really important to be considered is the fact that the resistance ships are experiencing, can be greatly reduced if they are sailing very close to the optimized trim values for each draught/displacement value. Thus, the trim parameter, can really be used in the final version as a control parameter.

8.3.6 DQN Hyper-parameters tuning

The hyper-parameters tuning was completed using the really painful procedure of random sampling. If the hyper-parameter tuning procedure is implemented using more sophisticated techniques that would reduce the total amount of time needed to find the correct values. This can really contribute into making real progress in this project. Some recommendations are the use of genetic algorithms and grid search. Also, an interesting consideration may be the use of the convergence speed as evaluation metric. Right now, the available packages in Julia are a bit restricting, however in the next few months this job will be surely easier to do.

Bibliography

[1] Hannah Ritchie, Max Roser, and Pablo Rosado. "CO_2 and greenhouse gas emissions". In: *Our world in data* (2020).

[2] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[3] Andrew Barto & Richard S. Sutton. *Reinforcement Learning: An Introduction*. 1992.

[4] Joan P Petersen, Ole Winther, and Daniel J Jacobsen. "A machine-learning approach to predict main energy consumption under realistic operational conditions". In: *Ship Technology Research* 59.1 (2012), pp. 64–72.

[5] Bourchas Orfeas. "Neural networks for the prediction of fuel oil consumption for containerships". 2021. URL: `https://dspace.lib.ntua.gr/xmlui/handle/123456789/52927`.

[6] Christos Gkerekos, Iraklis Lazakis, and Gerasimos Theotokatos. "Machine learning models for predicting ship main engine Fuel Oil Consumption: A comparative study". In: *Ocean Engineering* 188 (2019), p. 106282.

[7] Lúcia Moreira, Roberto Vettor, and Carlos Guedes Soares. "Neural network approach for predicting ship speed and fuel consumption". In: *Journal of Marine Science and Engineering* 9.2 (2021), p. 119.

[8] Markus M Johansen. "MACHINE LEARNING FOR SHIP AUTONOMY". PhD thesis. Norwegian University of Science and Technology.

[9] Martijn van Otterlo and Marco Wiering. "Reinforcement learning and markov decision processes". In: *Reinforcement learning*. Springer, 2012, pp. 3–42.

[10] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[11] Quentin Delfosse. *Why is there a Deadly Triad issue and how to handle it ?* 2020. URL: https://k4ntz.medium.com/why-is-there-a-deadly-triad-issue-and-how-to-handle-it-e1839b1a8b40.

[12] Shangtong Zhang, Hengshuai Yao, and Shimon Whiteson. "Breaking the deadly triad with a target network". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12621–12631.

[13] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[14] L.J. Abdi H. & Williams. *Principal component analysis*. 2010.

[15] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[16] Kaifeng Gao et al. "Julia language in machine learning: Algorithms, applications, and open issues". In: *Computer Science Review* 37 (2020), p. 100254.

[17] Mike Innes. "Flux: Elegant machine learning with Julia". In: *Journal of Open Source Software* 3.25 (2018), p. 602.

[18] Jun Tian and other contributors. *ReinforcementLearning.jl: A Reinforcement Learning Package for the Julia Language*. 2020. URL: https://github.com/JuliaReinforcementLearning/ReinforcementLearning.jl.

[19] Fredrik Bagge Carlson. "Hyperopt. jl: Hyperparameter optimization in Julia." In: (2018).

[20] A Famili et al. "Data preprocessing and intelligent data analysis". In: *Intelligent data analysis* 1.1 (1997), pp. 3–23.

[21] Pavlos Karagiannidis et al. "Ship fuel consumption prediction using artificial neural networks". In: *Proceedings of the Annual meeting of marine technology conference proceedings, Athens, Greece*. 2019, pp. 46–51.

[22] Tim Besard, Christophe Foket, and Bjorn De Sutter. "Effective extensible programming: unleashing Julia on GPUs". In: *IEEE Transactions on Parallel and Distributed Systems* 30.4 (2018), pp. 827–841.