YRKESHÖGSKOLAN
NOVIA

Deep Reinforcement Learning Implementation for Vessel Stability Calculations

Applying Machine Learning to Vessel Cargo Operations

Mateusz Szynalewski

Thesis for a Novia (UAS) – Master degree

Autonomous Maritime Operations

Warszawa, 2024

MASTERS THESIS

Author: Mateusz Szynalewski
Program and Campus: Autonomous Maritime Operations, Novia
Specialization:
Supervisor: Katarina Sandström
Title: Deep Reinforcement Learning Implementation for Vessel Stability Calculations

---

Date: 28.03.2024 Number of pages: 41 Appendices: 3

Abstract

This paper focused on combining ships stability calculations with deep reinforcement learning methods. For the sake of this thesis the author developed a proof of concept software that not only enables users to solve tasks from the field of vessel stability in manual mode but also allows the use of Proximal Policy Optimization (PPO) algorithm, which utilizes Python 3.11 programming language along with the Stable Baselines 3 package.

Deep reinforcement learning, which is an optimization problem framework for learning from experience that leads to refining a policy to maximize a future reward, was discussed in further detail. Moreover, attention was paid to a detailed explanation of the whole development process, neural networks architectures and hyperparameters defying PPO algorithm used in this project.

In the final chapters, the author presents the utility of the software in manual mode based on a few example tasks as well as the results achieved by the trained neural network which achieved an efficiency of 42 percent. Final conclusions were presented which emphasized the advantages and disadvantages of utilizing machine learning in the field of stability calculation. Finally, the potential for further development of this kind of software and autonomous maritime industry in general in the future was discussed.

Language: English
Key Words: vessel stability, deep reinforcement learning

Table of Contents

Appendices

List of Tables

List of Figures

1 Introduction

Newly built vessels are becoming increasingly technically advanced, which inevitably makes them more and more demanding to operate. The rising number of law regulations and requirements to be met by ships as well as crew members further complicates the matter. The complicated situation raises the question, what are the main advantages and disadvantages from deviating from manually writing instructions towards machine learning solutions? It has become almost impossible to manually perform all the necessary calculations related to loading and ballasting operations within a reasonable time frame in the context of real cargo loading and unloading scenarios. Additionally, the idea of automatization of those tasks is supported by the fact that many procedures related to the ship stability calculations are considered to be relatively difficult. Said complexity indicates a remarkably high risk of human error, which is the most prevalent cause for accidents in the maritime industry. Taking this into account, the obvious solution seems to be to narrow down the human role on board vessels, so that over time this role will be limited to supervising the work of automated solutions only. This role may even become obsolete in the future as fully autonomous ships gain popularity. However, it is still unclear if machine learning can be successfully applied to vessel stability calculations.

In this paper the author attempts to answer the question, whether commonly available deep learning tools are sufficient enough to implement them for stability calculation. To answer this question, an attempt to further the software development for ship stability calculations with the use of subfield of machine learning called deep reinforcement learning will be presented. The aim was to develop a software which, given the basic data regarding cargo, will be able to perform all necessary loading calculations. The user is expected to pass information to the software regarding sea water density, current state of ballast tanks and cargo holds as well as tonnage and stowage factor of the cargo to be loaded. For this research project, a bulk carrier vessel stability data is used as a model. It offers a lot of versatility regarding loading scenarios without the necessity to model multiple vessels. The end goal of the software is to load bulk cargo and ballast the vessel automatically to the desired draft and trim.

For the development of this application, Python 3.11 programming language along with Stable Baselines 3 library were chosen. These tools are broadly used by the machine learning community which is an undoubted advantage in terms of acquisition of the materials and

documentation as well as the further potential of development of the software in the future. For the sake of this thesis and the software assigned to it, solutions from ship stability field and from machine learning, including algorithms from the family of reinforcement learning connected with a deep neural network, were combined (for the methodology see Chapter 7).

2 Ship stability

Ship stability is an area of naval architecture and ship design that deals with ships' behaviour at sea, both in still water and in waves, whether intact or damaged. Stability calculations focus on establishing exact coordinates of the centres of gravity, centres and the metacentres of vessels, as well as the relations between them. Calculating ship stability takes into account centres of mass of all objects on the vessel, which are then computed to identify the centre of gravity of the vessel as a whole, while the centre of buoyancy of the hull is determined by taking into account variables such as draft and trim. A crucial aspect of ship stability is understanding how various factors, such as load, hull shape and weather conditions, affect its stability. Shipyards strive to optimize these parameters to ensure the vessel's safety and efficiency while underway. Optimal ship stability is essential for the safety of the crew, cargo and the ship itself (Kabaciński, 1993, Chapter 2; Rawson & Tupper, 2001, Chapter 4; Szozda, 2016, pp. 13-15).

While loading the cargo it is important to minimize the stresses on the ship's hull and to distribute them as evenly as possible. This is especially important in case of large vessels. All ships are designed with limitations imposed upon their operability in order to ensure that the structural integrity will be maintained. Therefore, exceeding these limitations may result in over-stressing of the ship's structure, which can further lead to catastrophic failure. We can distinguish two types of stability, that combined together create overall vessel stability - longitudinal and transverse stability (Barras & Derret, 2006, Chapter 6; Hughes, 1917, pp. 683-692; Szozda, 2016, Chapter 4).

2.1 Longitudinal stability

Longitudinal stability describes of stability describes the ability of a vessel to maintain balance along its longitudinal axis. The vessel should be able to counteract forces in balance resulting from backward and forward movement (Clark, 2002, Chapter 6).

2.2 Lateral stability

Lateral stability refers to ability of the vessel to maintain balance along its transverse axis. The vessel should stay stable during lateral movements such as rolling the ship from one side to another. (Miłobędzki, 1963, Chapter 2).

3 Origins of artificial intelligence and its evolution

Artificial intelligence is not an achievement of a single person, but rather the result of the combined efforts of many mathematicians, engineers and scientists over the years. The beginnings of artificial intelligence can be traced back to the 1950s when Alan Turing formulated the “Turing Test” to judge whether a machine shows signs of intelligence or not. He was not the only one laying the foundations for artificial intelligence at that time; however, due to his significant contribution and achievements in that field he is considered the father of artificial intelligence (Morales, 2020, pp. 15-16).

The first artificial intelligence projects dealt with relatively simple logical problems, puzzles and games, but with time, they gradually evolved and aimed to solve more sophisticated tasks. The breakthrough was in early 2000s, when rapid growth in development of graphic processing units (GPUs) resulted in increasing computing power, which allowed for more parallel calculations to be performed. That breakthrough turned out to be crucial for the further evolution of machine learning. The increased computational power opened a door to the effective use of neural networks – a tool already known to mathematicians, but too impractical to use until then. It was precisely the rapid evolution of graphics cards that unleashed the potential of neural networks.

In the relatively short timeline of rapid evolution of machine learning, a few breakthrough projects that singlehandedly progressed the entire field can be mentioned. One of them was definitely the AlexNet project that took place in 2012. The model achieved never-seen-before results in the image classification field. Since then, the development of deep learning based on increasingly larger neural networks with more and more parameters has gained momentum. Another milestone in the development path of this field was the AlphaGO project based on reinforcement learning technology created by the DeepMind studio in 2016. For a long time, the game of Go was considered to be too elusive and required the player's intuition rather than an algorithmic approach. The commonly accepted consensus at that time was, that computers were not yet able to compete with human professionals. Some experts doubted whether computer would ever be able to compete with professionals. Nevertheless,

the AlphaGO project was set to undermine this theory and did it so spectacularly that some strategies developed by the program are now being thought to humans.

Nowadays, machine learning has developed sufficiently enough to be used in fields such as medicine, engineering, design, finances and many other areas. At the time of writing this thesis, the technology is already well rooted in pop culture and easily accessible so anybody with internet access can benefit from projects such as Stable Diffusion for graphic generation or chatbots such as ChatGPT. In the era of the computer's hardware evolution, the existing computer technology self-accelerated it's further development. Similarly, we observe present artificial solutions being involved in creating new ones. This phenomenon seems to be only accelerating, considering the number of new projects being released nowadays.

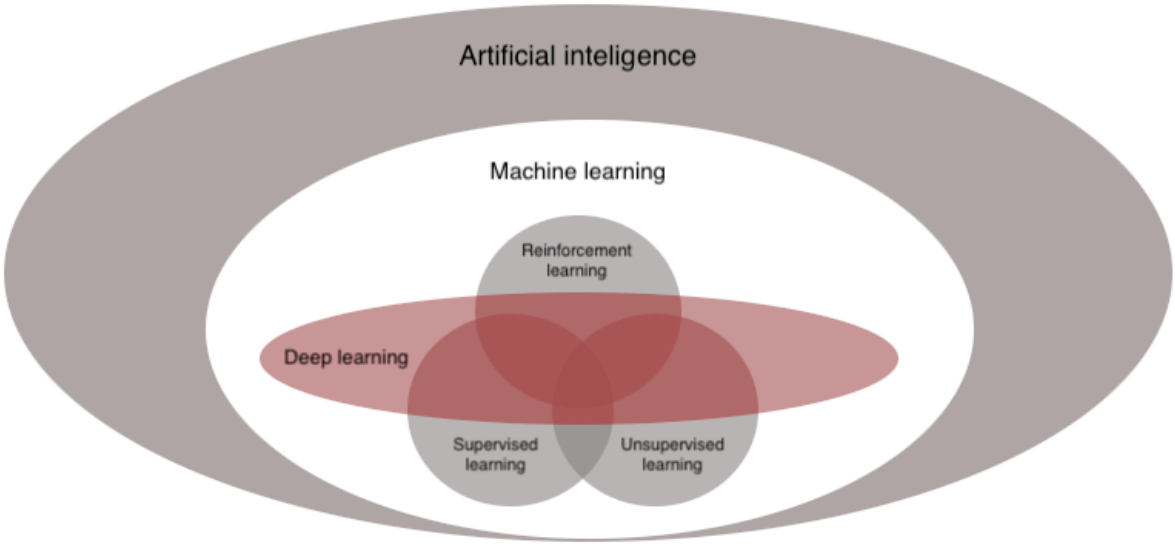

Figure 1. Subfields of artificial intelligence according to (Morales, M., 2020, Chapter 1)

4 Reinforcement learning

Reinforcement learning is one of the branches of machine learning, where the component responsible for making decisions and achieving the goal in a potentially complex environment is called an agent. The agent's role is to solve the task defined in program environment. Initially, a trial-and-error approach is used to search for the solution that later becomes increasingly refined. The end-goal of this task is defined through the reward signal which can be either dense or sparse. To make the software learn, the agent gets either rewards or penalties (rewards with a negative sign) for actions it takes. The software developer creates reward rules while not providing any suggestions on how to solve the given task. It

is the agent's goal to find an optimal strategy leading to the highest reward. The denser the reward signal, the more supervision the agent will have and the faster it will learn. Sparse signal however, allows for more flexibility in the process of solving the task, therefore increasing the likelihood of new unexpected behaviours of the agent that may lead to newbetter solutions. In practice the balance between the two must always be found. After the signal is set, it is up to the model to figure out how to perform the task to maximize the reward, starting with purely random behaviors, and finishing with efficient strategy, which is often beyond human intellect abilities (Mousavi et al., 2018; Nandy & Biswas, 2018, Chapter 1; Suttoon & Barto, 2018, Chapter 1; Trask, 2019, Chapter 1).

4.1 Agent

An agent is a component hat learns and makes decisions based on previous experiences. It takes actions by interacting with the environment and receives rewards based on the effects of its actions (Trask, 2019, pp. 6-7). We can distinguish two types of agents: single and multiple agent. As the names suggest, there are multiple agents in a multi-agent environment and only one in a single-agent environment. More than one agent approach is used particularly for complex tasks in stochastic environments, where the level of uncertainty is higher (Nandy & Biswas, 2018, pp. 16-18).

4.2 Environment

Environment is the outside world from the perspective of the agent. It is everything that agent interacts with. The following is a brief description of different types of environments.

4.2.1 Deterministic environment

A deterministic environment refers to the probability of of a successful execution of an action that is going to be taken by the agent is equal to 1 and the result of such an action is fully predictable as well. This means that the agent will always, and without fail, perform desired actions. For example, if agent decides to go forward, it will go forward. (Nandy & Biswas, 2018, p. 14; Hurbans, 2020, p. 9)

4.2.2 Stochastic environment

Stochastic environment refers to the probability of a successful execution of an action that is going to be taken by the agent is less than 1. This means that the agent may not successfully

perform desired action. Let us assume that the agent is walking on a frozen lake and would like to go forward, but because of a slippery surface the probability of going forward is not 100%; there might be 70% chance of going forward and 30% chance of going in any other direction. The agent has to keep in mind that not all of its actions will succeed. Hence, it has to deal with a greater level of uncertainty than in a deterministic environment (Hurbans, 2020, p. 9).

4.2.3 Fully observable environment

A fully observable environment occurs when an agent knows everything about the given environment. All data related with the agent's task is available to it. For example, in a chess game, the position of all the figures on the board are available for the player (Nandy & Biswas, 2018, p. 15).

4.2.4 Partially observable environment

In a partially observable environment, the agent has only constrained knowledge about the environment. For example, in a poker game, it has no access to the information regarding opponent's cards (Nandy & Biswas, 2018, p. 15).

4.2.5 Discrete environment

In a discrete environment, there is a finite number of actions for moving from one state to another. For example, in a chess game, there is only a finite set of moves available to a player as per (Nandy & Biswas, 2018, p. 16).

4.2.6 Continuous environment

In a continuous environment, there is an infinite number of actions available for moving from one state to another. For example, there are infinite routes available for traveling from one place to another, although any pair may vary from radically different (turning left vs right) to almost undistinguishably different (turning at an angle of 89.99° vs 90.00°) (Nandy & Biswas, 2018, p. 16).

4.3 States and observations

State of the environment is a full set of data describing the environment. Observation, on the other hand, is only a particular part of a state that the agent can observe. Quite often the agent

does not have access to the full knowledge about the environment and is forced to rely only on its observations while making decisions. Depending on the task, observations may include more or less information. Often, particularly in neural networks aimed at mimicking a person interacting with a computer (for example, playing a video-game or distinguishing an object in a picture), observation is identical to the general view of the screen that would be available for the user (see Figures 2 and 3; Morales, 202, pp. 8-9).

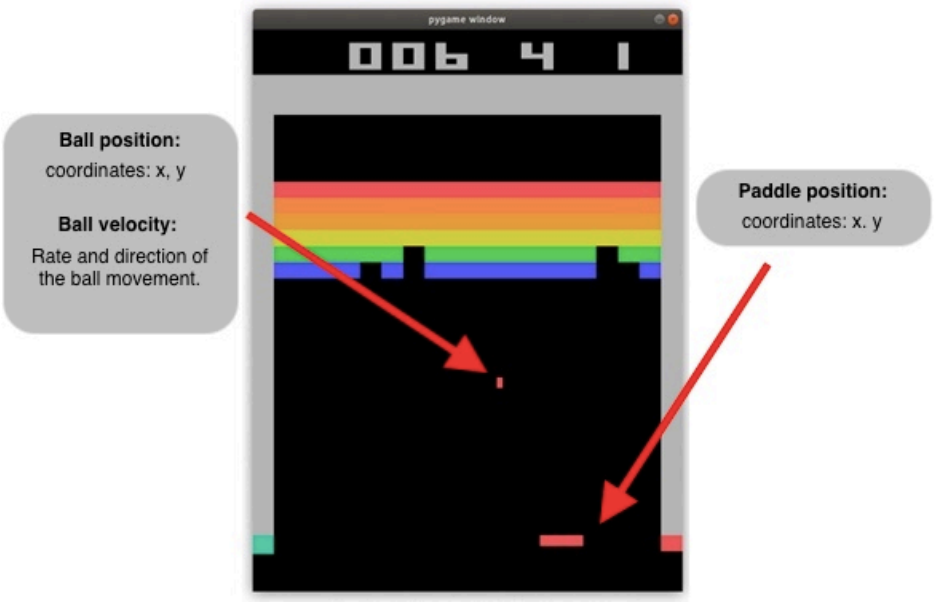

Figure 2. State

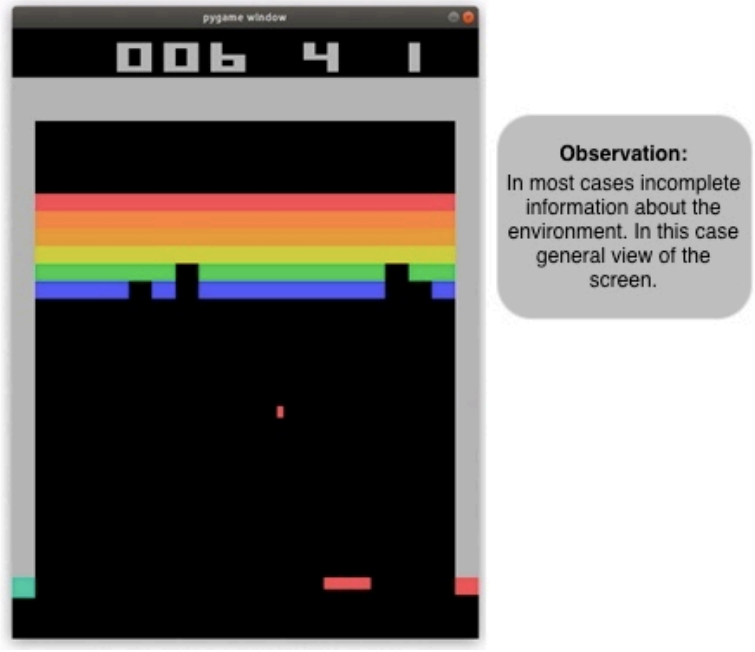

Figure 3. Observation

4.4 Reward

Reward is a temporary information available for the agent after every move. It changes as the agent proceeds with solving the task. Reward systems may be dense or sparse. Although, a sparse rewards approach is sample inefficient (agent has to run many examples to learn), an advantage of this approach is a high level of flexibility provided to the agent. The sparsest scenario would be one with a single reward granted at the very end of the task if the final goal has been achieved. This approach may lead to unexpected, "creative" solutions, that are often unimaginable for the programmer. Dense reward system means that there are many intermediate rewards on the way to the final reward at the end of the task. This approach allows for a quicker, more sample efficient results, but will never produce a surprisingly efficient outcome in the end.

Reward system tunes whether our machine learning approach falls more into supervised or unsupervised category. Reinforcement learning in general lays somewhere in between. The denser reward system is provided for the agent, the more supervision it has, and the closer to pure supervised machine learning it gets.

Although rewards are typically reset after every task, the long-term total reward made of the sum of rewards gained on the learning path is stored as a value called return (Morales, 2020, Chapter 1).

4.5 Markov decision processes

The theory of Markov Decision Processes is a mathematical framework for modelling decision making in situations where outcomes are partially random and partially under control of the decision maker. This framework allows for modelling sequential decision-making problem under uncertainty, in such a way that the agent can learn through experience. In reinforcement learning we assume that all environments have MDP working under the hood. Current states depend on decisions taken in the past according to (Morales, 2020, pp.45-53).

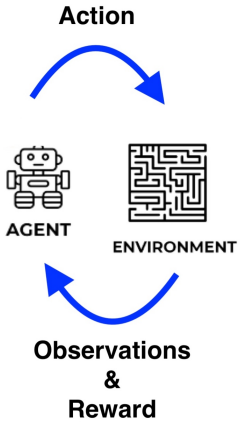

Figure 4. Reinforcement learning cycle

5 Deep reinforcement learning

Deep reinforcement learning combines artificial neural networks, which are multi-layered non-linear function approximations, with reinforcement learning. This combination allows for developing a sequence of decisions leading to a desired solution. Although it is possible to use “pure” reinforcement learning algorithms when dealing with small tasks in simple environments, the higher complexity of a problem and environment require the use of function approximators. That combination is exactly where reinforcement learning meets artificial neural networks (Mousavi et al., 2018).

Still, it should be emphasized, that deep reinforcement learning is far from being perfect. There are two main issues with this approach today. The first one is the amount of an input data machine learning algorithms require. In most problems, agents need millions of samples to learn well-performing policies. Humans, on the other hand, can successfully learn from only a few interactions. For example, to train a model to distinguish cat from a dog in pictures, we would have to provide hundreds of thousands of pictures of these animals to

properly feed the model. On the other hand, the average person would only need a few pictures in order to understand the differences between cats and dogs and to be able to classify them properly in the future unseen photos. Many teams around the world aim to resolve this issue by creating data-efficient deep learning algorithms which require less input data to train the model, as sample inefficiency is still probably one of the weakest parts of deep reinforcement learning nowadays.

The second main problem is computational power. Machine learning algorithms require immense computational power which in most cases is extremely expensive and impractical. This issue is also being researched with some promising results by Google.

These two problems, however, are intertwined. Resolving one should help resolving the other: if the input data is smaller the computational power to process it drops down as well. This mechanism also works the other way, as increasing the computational efficiency of our computers will allow for more input data. Considering that work on both these issues is carried out independently and in parallel, we can expect synergy in this field.

5.1 Artificial neural network

Neural network consists of three types of layers – input, hidden and output layers. That does necessarily mean that such a network consists of three layers only, as the hidden layer may consist of many individual layers that share the common trait of being inaccessible from the outside. The first layer receives raw input data, then hidden layers receive data as a result of data processing in the previous layer and the last layer produces an output. The distinguishing feature of this method is the possibility of solving practical problems without knowing their prior mathematical formalization. A further advantage is that there is no need to refer to any theoretical assumptions about the problem being solved when using the neural networks. The most significant feature of the neural network is its ability to learn from examples and the ability to automatically generalize the acquired knowledge. This makes artificial neural networks excellent function approximators (Hurbans, Chapter 9).

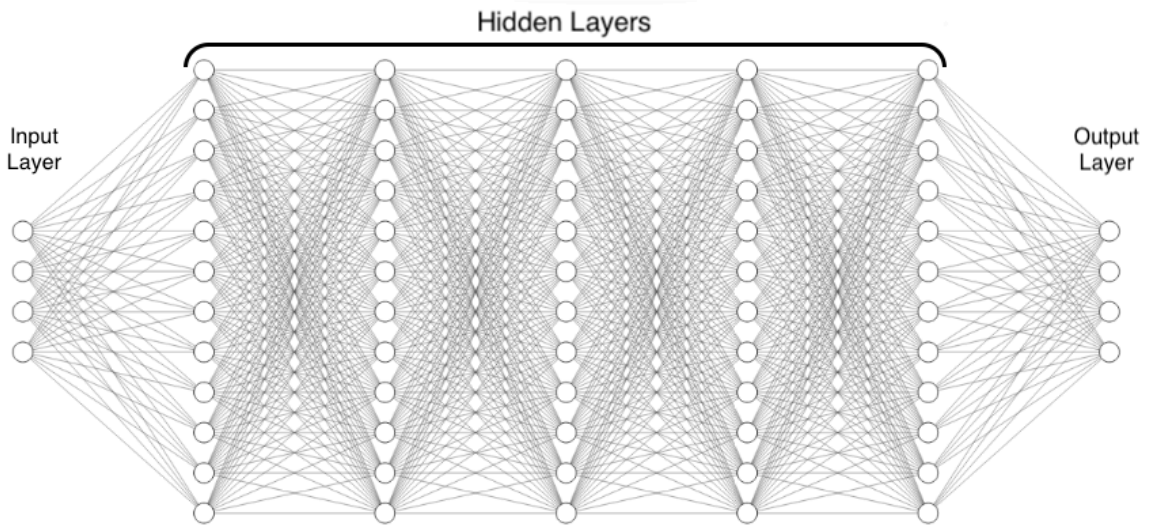

Figure 5. General architecture of artificial neural network

There are different architectures of artificial neural networks, each with their own strengths and weaknesses. The most commonly used types at the time of writing the thesis are as follow:

Feedforward Neural Network – This is the simplest type of neural network. The information flows only in one direction from input to output. Layers are fully connected, which means each neuron in a layer is connected to all neurons in the next layer (Theodoridis, 2015, Chapter 18.3).

Recurrent Neural Networks – These types of neural networks have a component acting as a "memory" which allows information to flow in cycles through the neurons. This approach gives the network ability to process sequences of data which is extremely useful in time series or speech (Aggarwal, 2018, Chapter 1.6.4).

Convolutional Neural Networks – These networks are most commonly used to process data stored in form of images. Layers have convolutional layers which are responsible for detecting specific features in the data and pooling layers, which reduce spatial dimensions of data according to (Aggarwal, 2018, Chapter 1.6.5; Hurbans, 2020, Chapter 9).

Autoencoders – These networks have built-in encoder that reduces input data to lower-dimensional representation and decoder that brings it back to normal state (Aggarwal, 2018, Chapter 2.5.1).

Generative Adversarial Networks – These types of neural networks are used for generative modelling. They have two major components that define them: a generator which is responsible for generating new data samples and a discriminator that distinguishes between generated and real data (Aggarwal, 2018, Chapter 1.6.5; Hurbans, 2020, Chapter 9).

5.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) belongs, together with DQN, A2C and TRPO, to the most commonly used reinforcement learning algorithms that can handle action space of discrete type. The main idea behind this algorithm is to provide a high level of stability during the whole learning process by assuring balance between not deviating too far from the old policy yet allowing for substantial policy updates. According to Morales (2020, p. 398), the algorithm iteratively updates the policy based on experiences gained from interactions with the environment.

The following bullet points provide a general overview of the PPO algorithm workflow:

- The agent gathers data by interacting with the environment that incorporates information regarding states, activates and rewards.
- The agent approximates the benefits of selecting specific actions in particular states over other actions by utilizing the gathered data. These benefits reflect whether an action performs better or worse than the average action typically taken in that state. This process is called policy evaluation.
- The policy is updated with surrogate objective function.

The aim of each cycle is finding an updated policy that provides better performance with stochastic gradient descent. The steps are repeated during the whole learning process, gradually leading to policy improvement (Morales, 2020, Chapter 12).

5.3 Actor–critic algorithms

Proximal Policy Optimization (PPO) falls into an actor–critic category of deep reinforcement learning algorithms, which means it utilizes two artificial neural networks, one for the policy (actor) and one for the value function (critic) (Zai & Brown, 2020, Chapter 5).

5.3.1 Policy network - actor

The agent's behaviour in the environment is determined by the policy. The policy network is responsible for correlating observations from the environment with the probability distribution of success for each of the available actions (Figure 6). It describes the likelihood, that taking any given action in a particular state will eventually result in solving the task correctly. Therefore, it reflects agent's strategy for selecting actions depending on the current state. During the learning process this network is being updated leading to a better policy that over time leads to the highest expected cumulative reward. Optimal policy is the one that has the optimal value function (Morales, 2020, Chapters 11-12; Zai & Brown, 2020, Chapter 5).

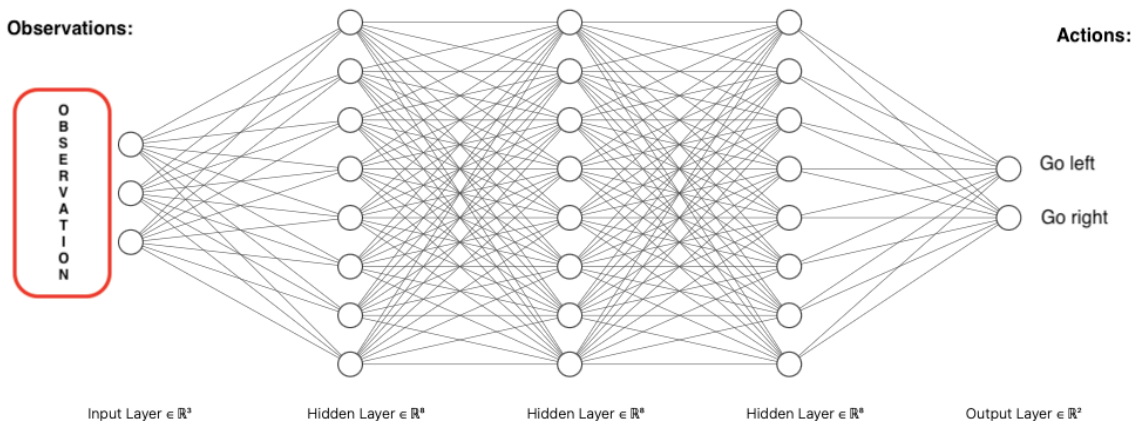

Figure 6. Policy network

5.3.2 Value network - critic

Value function network takes into account a certain state and calculates estimated cumulative reward at the end of the task (Figure 7). Its role is to score how good any given particular state is for the agent in terms of expected future rewards following the agent's policy thereafter. Assessment depends on what actions might be taken in the moment as well as in future states. Therefore, value functions are always based on policies making policies and value functions are inevitably intertwined (Morales, 2020, Chapters 11-12; Zai & Brown, 2020, Chapter 5).

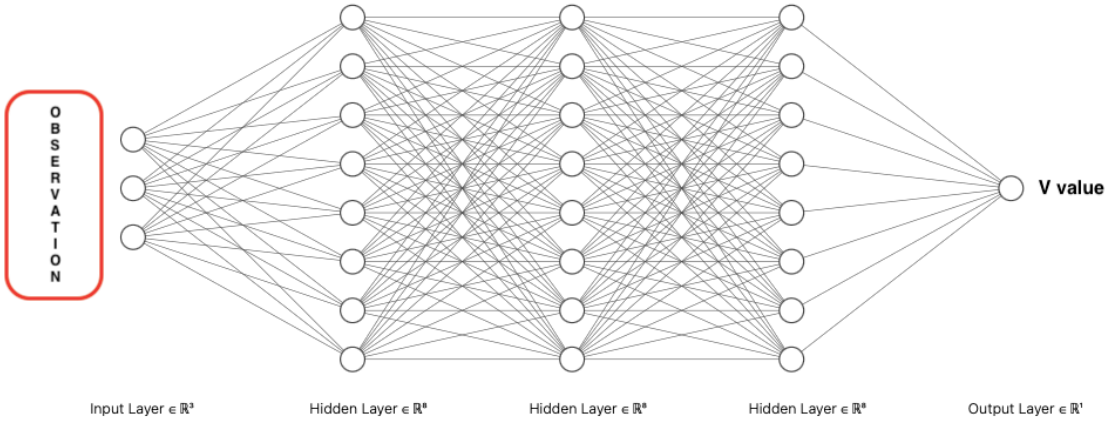

Figure 7. Value network

The idea behind combining these two components is to leverage the advantages of both policy-based and value-based methods. The actor's role involves exploration of the environment and acquiring the optimal policy, while the critic provides feedback guiding the learning process. Nature of actor-critic approach empowers balance between policy tuning and value estimation leading to more efficient and stable learning path according to (Morales, 2020, Chapters 11-12).

5.4 Hyperparameters

There are two types of components determining learning process in the field of machine learning, hyperparameters and parameters. Hyperparameters (external parameters) include for example but not limited to: topology of artificial neural network, values of learning rate or gamma. They are set by the software developer before initializing learning process and are not being further updated by the model during training. Nevertheless, they have a significant impact on the parameters which on the other hand are internal to the model and are subject to updates during learning process. The parameters include, among others, weights and biases of artificial neural network. At the beginning of training a model, parameters are initialized as random values and as training progresses these values are being updated (Auffarth, 2020, p. 33; Burkov, 2019, p. 12). The following are hyperparameters used in the project.

5.4.1 Artificial neural network

Two basic hyperparameters included in every artificial neural network are: the type of artificial neural network and its topology, meaning the number of layers and nodes per layer. They have been extensively described earlier in this paper (see chapter 5.1).

5.4.2 Learning rate

Learning rate α determinates the balance between the agent exploring the environment and using the knowledge it has already acquired. In other words, it is setting up a trade-off between exploration vs exploitation. How much effort will be applied into exploiting a strategy that the agent already has and how much into exploring new methods that may not work at all but may also bring better, more efficient solutions, hence higher rewards. Learning rate takes values between 0 and 1, where zero means that the agent does not explore at all and there is only exploitation process occurring, while learning rate equal to one would result with the agent only exploring the environment and trying new things but never taking advantage of the experience it has already gained. Common solution in machine learning community is defining learning rate as zero at the beginning of the learning process and successively increasing it while iterating the task so it converges to one. With this approach the agent extensively explores environment at the beginning of a given task, while changing the ratio of exploration to exploitation as the task progresses to the point where great majority of its actions are based on the knowledge it has gained according to (Hurbans, 2020, p. 318)

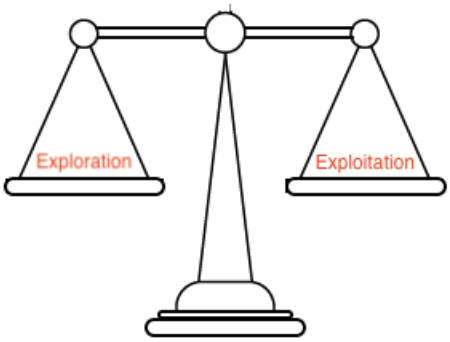

Figure 8. Exploration vs exploitation trade off

5.4.3 Number of steps

It determines number of interactions an agent has with an environment during its training process. It can be given as the total number of actions that the agent is able to undertake while solving a solving task or it can be measured by time units available.

5.4.4 Mini-batches

To improve stability of the learning process, common practice in machine learning field is using data in batches instead of single data point to update an artificial neural network. Minibatch size specifies number of data samples (steps or episodes) in a single update of artificial neural network. Most commonly this hyperparameter takes values which are power of 2, for example: 16, 64 or 256. Higher values of this hyperparameter may lead to more effective training but they significantly increase computing power consumption of the whole training process.

5.4.5 Number of epochs

Number of epochs defines how many times each mini-batch is used for policy and value updates during the training process. The number of epochs exerts crucial impact on how thoroughly the value and policy networks are tuned. A higher number of epochs tends to lead to more refined estimations; however, it also increases the demand for computing power extensively.

5.4.6 Gamma

Discount factor γ establishes hierarchy of the rewards. In other words, it determinates weather the rewards obtained in the short term or in the long term are more valuable for the agent. When gamma factor is equal to 0, the agent considers only current rewards, while discount factor equal to 1 makes the agent focus only on the long-term rewards. In practice, rewards that may be obtained in the short-term are typically more valuable, since future rewards contribution to total cumulative reward is never fully certain (Morales, 2020, Chapter 2; Nandy & Biswas, 2018, p. 10).

5.4.7 Generalized Advantage Estimation Lambda

It controls the trade-off between bias and variance in estimating advantages during the training process. It improves the accuracy of advantage estimation of taking specific actions in certain states, which is essential for policy updates. The advantage represents how much better was the most recently taken action, compared to an average action taken in that particular state. Tuning this parameter to its optimal value for a given task leads to more stable learning process (Nandy & Biswas, 2018, p. 10).

5.4.8 Clipping range

There are two types of clipping ranges in PPO algorithm, one dedicated for policy function another for value function. This hyperparameter limits the size of updates during learning process. The main idea behind clipping range is to prevent extreme updates of policy as well as of value function, which in most cases lead to training instability. In other words, it specifies the maximum allowable change in the policy and value by constraining their updates. It prevents too drastic changes in a single training step. The biggest advantages of appropriately set clipping range are promoting stable learning, which helps avoiding erratic behaviour of the agent, as well as maintaining a balance between exploration and exploitation.

5.4.9 Entropy coefficient for the loss calculation

It determines how deterministic or random the agent's actions are. High entropy encourages the agent to explore more, while low entropy favours more deterministic actions.

5.5 PPO Learning cycle

As the agent starts interacting with the environment, at the beginning its actions are purely random and exploration/exploitation trade-off is set up in such a way that the agent is focused solely on the exploration of the environment. With each interaction the agent gains more experience and its understanding of the quality of any given state as well as strategy for choosing the best possible action to take improves. Exploration to exploitation ratio changes as well, resulting with agent being less and less focused on exploring the environment with every iteration, while paying more attention to knowledge it has already acquired (Figure 9). This strategy should lead the agent to developing a well refined policy according to (Morales, 2020, Chapters 11-12).

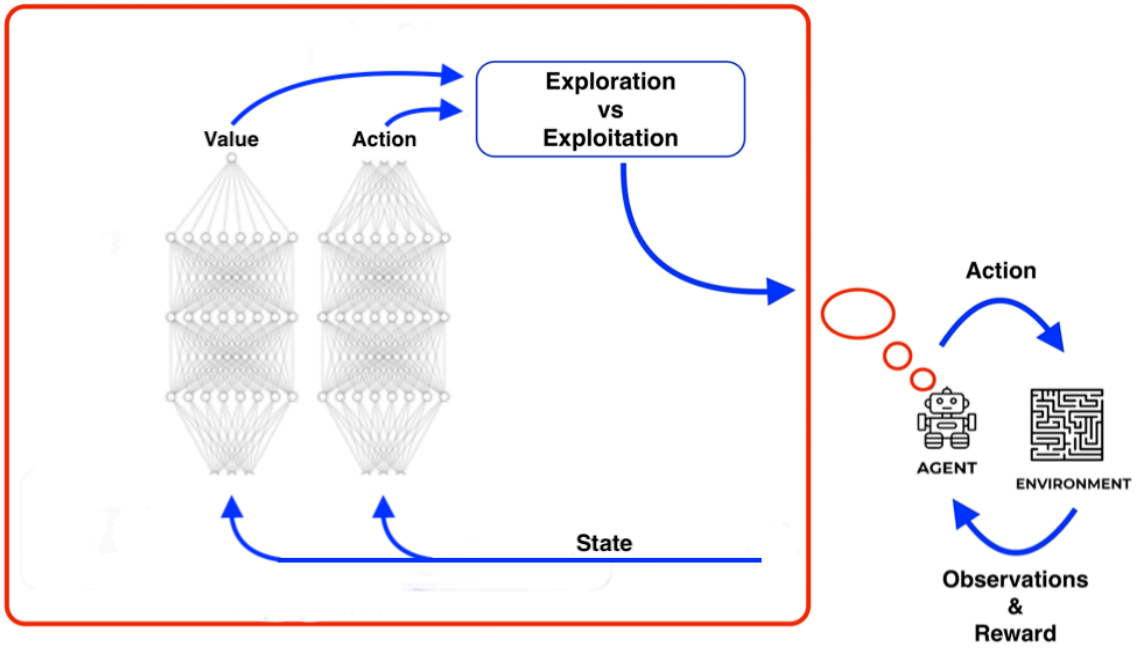

Figure 9. PPO Learning cycle

6 Stability software development

The aim of this project was to develop a program using Python programming language 3.11 which, given the vessel model along with a basic data regarding cargo, would be able to perform all necessary calculations to effectively load the vessel. A massive advantage of this approach would be the eventual departure from developing separate stability software for each vessel toward strategy where a single software is created for all possible vessels. This task, although quite futuristic, might be facilitated not so far in the future with software for specific type of vessels being developed. The vast differences between vessels and cargo specifications make this approach reasonable. Those could include: bulk carriers, tankers and container ships.

6.1 Python

Python is a high-level object-oriented programming language, that is commonly used in many fields by both beginners and advance programmers, due to its simplicity and versatility. It became one of the most popular programming languages thanks to its modularity as well as intuitive, easy to understand syntax. Those qualities allow developers to create libraries and packages that can be reused in different projects, making it one of the most user-friendly programming languages at the time of writing this thesis. Furthermore, Python is an open source programming language which means, it’s free of charge and users

can freely modify it depending on their needs. Moreover, there is an active community that constantly develops and improves packages, extensions, libraries as well as provides documentation along with learning materials. Not only does Python versatility allow for creating websites, games or desktop applications but also for data science and machine learning implementation.

As Python is a commonly chosen language for reinforcement learning and machine learning in general, with growing popularity of those fields many third-party libraries have been developed. That in turn significantly increased the effectiveness of research and development process in the field of deep reinforcement learning. Program developed for this thesis, like many similar projects focused on machine- and reinforcement learning, utilizes many of those libraries. Among them, a few are worth of an in-depth introduction.

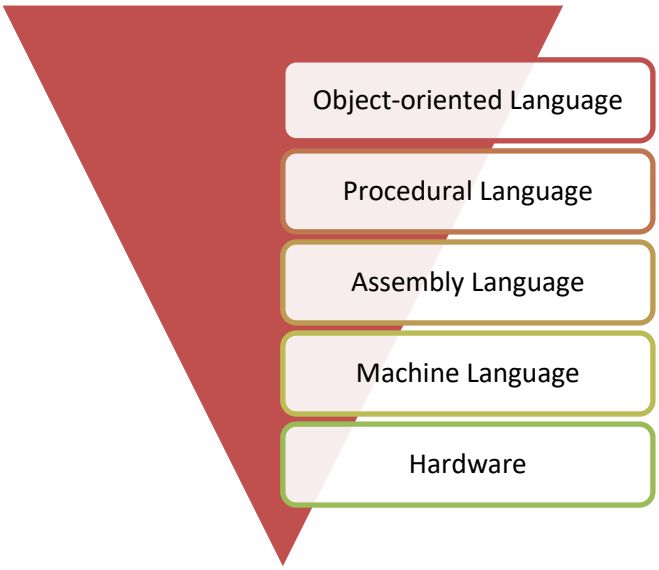

Figure 10. Programming languages classification – levels of abstraction

6.2 NumPy

NumPy is a package created for numerical calculations in Python, with a particular focus on effective operations on matrices which play crucial role in the world of machine learning. One of the biggest advantages of this package is high-level optimization, which is responsible for ensuring that mathematical calculations performed using this NumPy consume as little time as possible and require the least amount of computer processing power.

6.3 Pandas

Pandas is one of the largest, if not the largest, libraries for analyzing and processing data in Python. It is designed to work with data in tabular representation and to enable users to manipulate and process data stored in the form of tables in an optimal way. Data processing is efficient and effective thanks to data structures provided in the library, like DataFrame or Series. Furthermore, it enables working not only with numbers but also with strings, dates and other more unusual data types. Pandas offers many features for easy data manipulation, such as filtering, sorting, grouping and combining DataFrames. This library integrates very well with other popular Python libraries used in data science and machine learning, such as NumPy, Scikit-leran or Matplotlib, many of which the author of this thesis found extremely useful.

	T	V	D	TPC	ZM	MJ	XF	XS	ALFA	DELTA
0	4.00	7936.0	8159.0	22.8	11.19	15788.0	0.42	0.87	0.719	0.644
1	4.01	7958.0	8182.0	22.8	11.18	15795.0	0.42	0.87	0.719	0.644
2	4.02	7981.0	8205.0	22.8	11.16	15803.0	0.42	0.88	0.719	0.645
3	4.03	8003.0	8228.0	22.8	11.14	15810.0	0.42	0.88	0.719	0.645
4	4.04	8025.0	8250.0	22.8	11.13	15817.0	0.43	0.88	0.720	0.645
...	...	...	...	...	...	...	...	...	...	...
646	10.46	23736.0	24402.0	28.2	9.66	27728.0	-1.20	-5.07	0.891	0.737
647	10.47	23863.0	24430.0	28.2	9.67	27747.0	-1.20	-5.07	0.891	0.737
648	10.48	23790.0	24458.0	28.2	9.67	27765.0	-1.21	-5.07	0.891	0.737
649	10.49	23817.0	24486.0	28.2	9.67	27783.0	-1.21	-5.07	0.892	0.737
650	10.50	23845.0	24515.0	28.2	9.67	27801.0	-1.22	-5.07	0.892	0.737

651 rows × 10 columns

Figure 11. Table with B354 vessel hydrostatic data in DataFrame structure form

6.4 Scikit-learn

Scikit-learn is an extensive machine learning package in Python that provides solutions for data analysis, model building, feature selection and algorithm evaluation as well as optimization. The package offers a wide range of built-in machine learning algorithms including clustering, regression, classification, and more. The available algorithms include, among others, Support Vector Machines, K-Nearest Neighbours and Random Forests. Scikit-Learn offers very good integration with other most popular mathematics and machine learning packages, including aforementioned NumPy, which makes it a very versatile and

scalable tool. Watmore, similarly to other mentioned packages, Scikit-learn also has a large user-community which constantly improves it and updates its documentation. That makes it an excellent choice for people involved in machine learning and data science.

6.5 OpenAI Gym

OpenAI Gym is an open source library developed to provide unified structure of environments for testing and training machine learning algorithms, especially those from the branch of reinforcement learning. Not only does it offer tools for evaluating and monitoring performance but also an array of built-in premade environment-templates. Those features allow researchers to either test their algorithms directly, using the readily available templates, or build and then test in new, self-made environments based on those templates. The main idea behind developing this library was to provide a unified and easy-to-use tool enabling efficient testing and comparison of various algorithms from the reinforcement learning field. All OpenAI Gym environments are defined by a set of properties such as observation space, action space and rewards, which enables efficient algorithms testing in various reinforcement learning environments without the need to modify the tested algorithm each time. If the researched project was developed to accept and return values expected by one environment in OpenAI gym, it can be also tested on any other environment in the package.

This approach has more far-reaching consequences than it may seem. By ensuring all environments in this package are set on the same foundation, researchers can easily create new environments tailor made for their needs, thus expanding the already extensive set of environments provided by the OpenAI Gym library. This has greatly benefited the author of this work, who created a model of the B354 bulk carrier in such a way that it met the requirements of the OpenAI Gym environment, which significantly improved work on the project in its further stages of development.

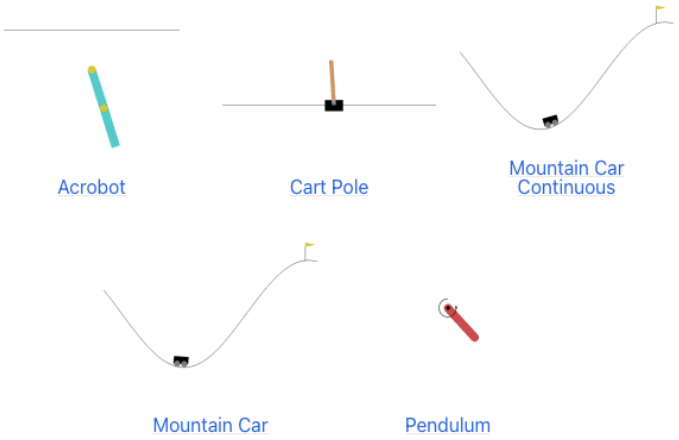

Figure 12. Examples of OpenAI Gym environments in classic control category

6.6 Stable Baselines 3

This library is designed purely for handling reinforcement learning tasks in OpenAI Gym environments, that, at the time of writing this thesis, does not seem to have any viable competition in a category of developing technologies in the field of deep reinforcement learning considering how versatile and user-friendly this tool is. Not only does the SB3 provide set of algorithms such as DQN, A3C, PPO and others, but also allows for easy customization and fine-tuning of the learning process, that may be easily adapted to suit specific environments and tasks. The library is designed in a modular way, which allows to modify its individual components by changing their parameters or even completely deleting or replacing them if needed. It supports various neural network architectures, which are all by default managed by the PyTorch package. Significant advantage of this library are the built-in tools that enable tracking the agent's progress during the model training process. This is remarkably useful for assessing agent’s effectiveness and tuning its settings during the development process.

6.7 PyTorch

PyTorch is a library used for machine learning projects, which has gained great popularity not only due to the simplicity of its syntax, which is very similar to pure Python, but also because of the flexibility and efficiency it offers in terms of designing and testing deep learning models. Said efficiency is related with the fact that it uses a dynamic computation graphs, instead of static ones, which means that the graph is created live, during code execution. Thanks to this approach, users can dynamically create and modify models during

training. Moreover, PyTorch allows for optimization of calculations performance with the use of graphics cards. This allows the users to significantly speed up the model training process. Just like most popular Python libraries, this one also has a substantial support from the machine learning and data science community that constantly creates new educational materials and code repositories.

7 Model development

For this project the highly supervised reinforcement learning type of machine learning has been chosen, as this approach seems to be a better fit from the perspective of maritime industry and everyday use. The loading operations require compliance with strictly defined rules in order to ensure safety not only during the passage of the loaded ship, but also in the harbour, during those operations. Theoretically, for the research purposes, unsupervised deep reinforcement learning could also be implemented to see what kind of creative and unusual solutions we could be obtained from such a model. Those solutions however, although interesting, may not be implemented in real-life scenarios without thorough testing, as they could potentially endanger safety of the vessel, cargo and workers.

7.1 General overview of the model

Digital model of B354 bulk carrier used in this project is based on paper version of stability booklet assigned to that ship. At the beginning of the software development presented in this thesis, the author migrated all relevant data regarding tanks, holds and stability from stability booklet into excel files. Next, the whole data has been pre-processed to match requirements of Pandas package for further handling data stored in the form of tables in Python. Once the whole necessary information regarding vessel has been transferred into Python, the author developed digital model of B354 vessel along with logic that enables execution of all necessary calculations from the field of ship's stability, including loading and unloading both cargo and ballast to desired spaces. Once this stage was completed, author created environment based on the B354 model that matches requirements of OpenAI Gym package where component responsible for making decisions (agent) can be trained. At that point certain restrictions were introduced so that the actions available for the agent are limited as compared to the normal user. That change was deemed necessary due to limited computing power available for training the model. Last stage was utilizing Stable Baselines 3 package to handle reinforcement learning algorithms along with neural networks architectures. The

path of searching for the best neural network architecture, hyperparameters and finetuning reward policy is described on the following pages of this thesis.

The foundation of this project is B354 vessel (Figure 13, Table 1 in Appendix 1), a bulk carrier vessel that can take containers on top its of hold hatches. Considering substantial differences between the two cargo types as well as the primary function of the vessel, in this project only the bulk cargo operations are being undertaken.

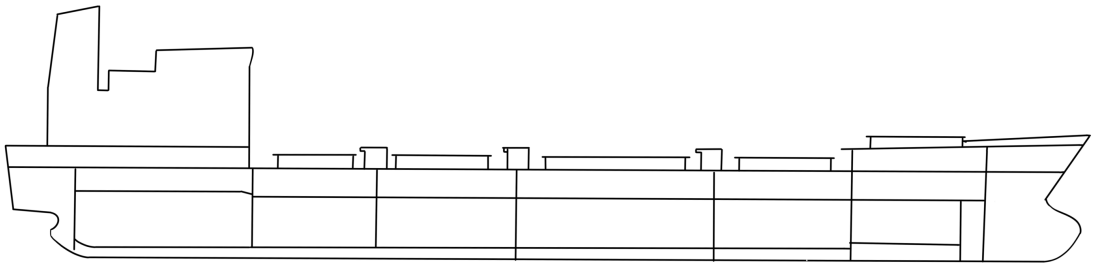

Figure 13. B354 overview

There are total of 5 holds in B354 vessel, one of them is designed for transportation refrigerated cargo and it is not taken under consideration in this project. The remaining four cargo holds are designated for storing bulk/general cargo, with the combined capacity of 8001.7m^3, available for loading cargo on B354 vessel (Table 2 in Appendix 1). Four of them are located forward from the midship and one is located aft from the midship (Figure 14). Therefore, most loading operations result in changing the trim to the bow. Of all five holds, only the two the smallest ones are not located on the longitudinal axis, what effectively eliminates the problem of unwanted listing during cargo operations.

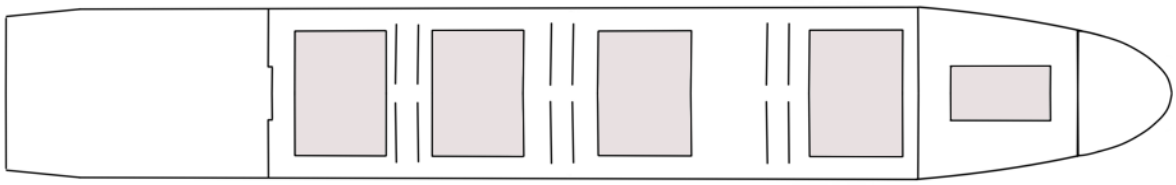

Figure 14. Holds arrangement – aerial view

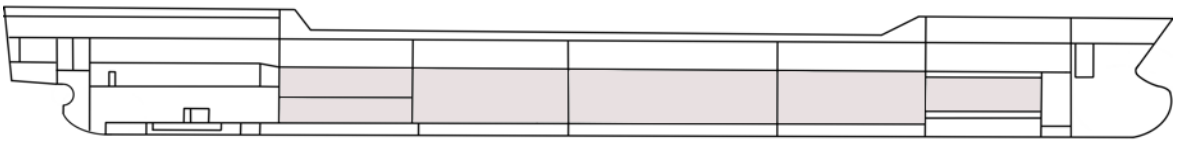

Figure 15. Holds arrangement – side view

The B354 vessel has 18 ballast tanks, with the combined capacity of 2597,2m^3 (Table 3 in Appendix 1). The majority of the tanks are located below the holds and machinery spaces, in the double bottom tanks, with an exception of 6 side-tanks being located laterally from the holds as well as 2 tanks – afterpeak and forepeak, located on the aftmost and foremost sides of the vessel respectively. Unlike cargo holds, almost all ballast tanks of this vessel are not located on the longitudinal axis, with the exemption of the afterpeak and the forepeak (Figures 16, 17).

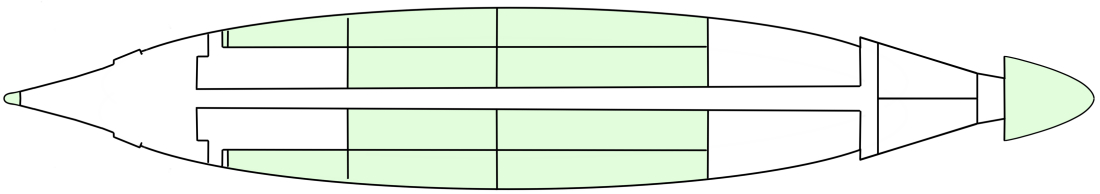

Figure 16. Double bottom tanks

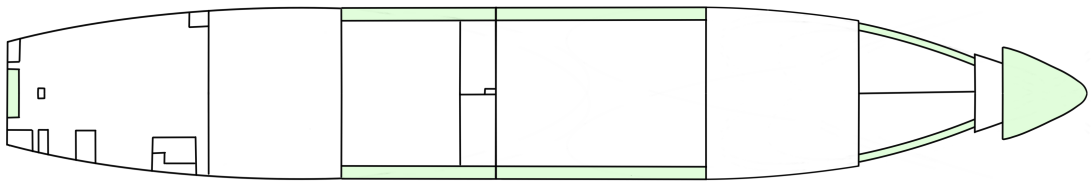

Figure 17. Tween-deck tanks

For the purpose of the program hydrostatic information regarding B354 vessel has been divided into 4 separate classes:

- Mass_table
- Ship

- Cargo_hold
- Ballast_tank

Each class contains functions and data structures responsible for handling all necessary stability calculations related to the given class. That approach facilitates handling calculations, especially regarding repetitive spaces such as holds and tanks, more effectively.

Few simplifications have been applied to the model to facilitate training of the agent, given computational power available for the project. Firstly, as possible listing resulting solely from cargo loading operations was deemed irrelevant, in the B354 vessel model all the paired spaces located on both sides of the vessel were merged, so that their centre of gravity would always be located on the ships longitudinal axis. Furthermore, the model developed for the purpose of this project does not consider other spaces, such as fuel tanks, sludges or machinery spaces in detail. They are all summarized together with the lightship as one. Vessel is always stocked with 100% provisions, including fuel and lubricating oils, while the sea water density is pre-set to 1.025, so that the task involves solely loading the cargo and ballasting the vessel. To further unify all tasks, after launching the software and at the beginning of each task all holds and ballast tanks are empty. Effectively the initial state of the model can be summarized with the fixed set of values (Table 4 in Appendix 2).

The model allows the user to manually solve desirable tasks from the vessel stability field in terms of loading, unloading and ballasting vessel B354. It is also capable of generating tasks for the agent, with each task consisting of two variables:

- Cargo mass (ctl) – takes values from 2000t to 8000t
- Stowage factor (sf) – takes values from 1.0 to 1.5

Variables that make up the task are randomly generated from the pre-set range to ensure, that every task will be solvable.

7.2 Model application for stability calculations

The model has been manually tested before introducing deep neural network. A few tasks are presented here as an example and detailed data regarding final results can be found in Appendix II.

7.2.1 Scenario 1

The task was to load 5570t of bulk cargo with stowage factor equal to 1. To mitigate the initial trim towards the aft, cargo was loaded primarily to the holds located forward from the midship (holds 2 and 3), yet to avoid creating to extensive trim towards bow, remaining cargo has been loaded into hold 4. At this point the vessel's trim was towards the bow. It was mitigated by fully filling up ballast tanks 5A PS/STB as well as afterpeak up to 23%.

7.2.2 Scenario 2

The task was to load 4550t of cargo with stowage factor equal to 1.35. Moreover, holds I PS/STB must stay empty due to ongoing maintenance and trim at the end of loading operations has to be equal to -0.5m. To meet these requirements holds 2 and 3 were fully loaded while hold 4 was loaded in approximately 32%. To obtain desired trim, ballast tanks 6 PS/STB were fully filled up while forepeak has been filled up to 45% which resulted with the desired value of trim.

7.2.3 Scenario 3

The task was to load 4420t of cargo with stowage factor of 1.35. At the end of loading operation vessel was expected to maintain small trim towards stern as well as small list to starboard side. Meeting these requirements was achieved by loading holds I PS/STB, hold II fully and hold III to 94% as well as filling up ballast tank 7STB to 37% and forepeak to 13% which resulted with trim equal to -0.03m and list of 2° to starboard side.

7.3 Combining vessel stability calculations with deep reinforcement learning

It is worth considering what type of the environment is the agent dealing with wile solving tasks concerning the B354 vessel model. The rules of the solving stability tasks are designed without any randomness and the agent can load and ballast the vessel with absolute certainty of the effectiveness of its decisions. The vessel's behaviour is easily predictable as well, since the all stability calculations are based on well-defined equations. Therefore, we can classify this environment as a deterministic. Moreover, the design environment is of fully observable type, as all data related with the agent's task is available to it, which means each agent's observation is equal to the actual state of the world. There is no hidden data regarding the task from the perspective of the agent. It has complete awareness of the environment at

all times. This environment is also discrete due to finite number of states and actions for moving from one state to another.

7.3.1 Constrains and limitations

Unlike the user, the agent has constraints in terms of loading, unloading and ballasting B354 which are mostly related to saving as much computing power as possible, hence solving vessel stability equations with artificial neural networks is already vastly computationally demanding at a very basic stage. The following is a list of optimization and simplification of the stability calculations for the sake of development a proof of concept software that involves deep reinforcement learning in vessel stability calculations:

- The agent is allowed only to load cargo, unloading is unavailable
- The agent can only add ballast water to the ballast tanks. Discharging is unavailable
- While loading the agent must always use the full capacity of the hold or all remaining cargo, depending on which of the two values is lower.
- The agent adds ballast in batches of 2.5 tones. If the remaining available ballast tank volume does not allow for it, the agent fills the remaining tank volume.
- The agent has a limit of 100 steps to solve a given task.
- The agent cannot load cargo or ballast spaces, which centre of gravity is not located on the longitudinal axis of the ship – those spaces have been paired up to avoid creating list

To properly train the agent a function that generates tasks to be solved during the whole training process was developed. The agent's goal is to load cargo provided by the user and then ballast the vessel so that there is no list or trim (even keel, upright). In terms of stability calculations, it has to be noted whether the set of tasks provided to the agent is actually solvable. For instance, a task where the agent is asked to load 7000 tons of bulk cargo of stowage factor of 1.5 to B354 is impossible, because it will exceed available volume in cargo holds. On the other hand, a task of loading less than 2000 tons of goods is unacceptable as well, as there is insufficient amount of ballast tanks to provide desired trim of 0 with such a little amount of cargo. Hence, an `excercise_generator()` function has been developed in order to generate solvable tasks for the agent, where the stowage factor is in the range

between 1.0 and 1.5 and the mass of cargo between 2000 and 8000 tons. Moreover, there is a limit of 100 steps given to an agent to solve each task, hence tasks are designed to be solvable within fifty steps. If the agent has not finished a given task in 100 steps, the task is considered to be failed, the whole training environment is being reset and a new task begins. Considering limitations listed above, agent has very little error margin and, in some cases, it may even not have it at all.

7.3.2 Reward policy

Reward policy takes crucial role in terms of agent's performance. In the presented project the reward system is unquestionably of the dense type, which means that the agent is evaluated after each step and intermediate rewards and punishments (rewards with negative sign) are granted on its way to the final evaluation at the end of each task. The agent is rewarded according to the following policy:

- The agent gets a reward of +2 each time the trim of the vessel is reduced
- The agent gets a reward of +5 each time it loads cargo successfully
- The agent gets a reward of +2 once there is no remaining cargo, granted after each step since the cargo is fully loaded
- The agent gets a reward of +1000 for solved task, all cargo is loaded and trim is in range of -0.05m and 0.00m.
- The agent gets a reward of -10 (punishment) with each taken step
- The agent gets a reward of -10 (punishment) each time it decides to make a move that is impossible to execute, for example:
    - loading cargo to a fully loaded cargo hold
    - adding ballast to a ballast tank that has no available space
    - loading cargo when there is no cargo left to be loaded

The whole idea behind shaping the reward policy that way is to point the agent into a set direction. The component responsible for making decisions is expected to develop strategy where cargo is being loaded first and then the vessel trim is taken care of, not the other way around, while maintaining the lowest possible number of steps.

7.3.3 Environment, state, observation and actions

In real life the data regarding vessel characteristics is developed during the build process of the vessel in the form of ship's stability booklet. This publication is compulsory for all commercial vessels and must be provided by the shipyard. Information regarding ballast tanks and cargo holds states is provided by sensors that are commonly used onboard vessels. Here the environment is the B354 vessel itself. Information regarding cargo is provided on the basis of a cargo manifest. This data is going to act as an input for the model and due to the nature of its source it can be firmly stated that it is of high quality, reliable and without any noise before any pre-processing being applied to it, which is not common phenomena in the world of data science and machine learning.

In this project state and observations are the same, as the agent has full knowledge regarding B354 credentials at all times and those terms can be used interchangeably in regard to the model. Data regarding vessel's state is stored in an array of 16 elements, each state data point is in the range between 0 and 1 and the whole set is normalized. A complete list with description of each element in the array is available in Appendix III, (Table 14).

There are 14 different actions of discrete type that the agent can execute, the author created an `action_selection()` function that takes one parameter which is an integer number from 0 to 15. Each number has an assigned separate function that triggers desired activity. Available actions with a description can be found in Appendix III, (Table 15).

8 Results

In this chapter author presents the process of training the agent. As a measurement of progress, the mean reward (the higher the better), the mean length of episode (the lower the better), the highest reward, and solved tasks per 1000 tasks were chosen. The agent has been trained utilizing default hyperparameters settings from Stable Baselines 3 package, except neural network architecture which was set to 5 fully connected layers with 32 neurons each.

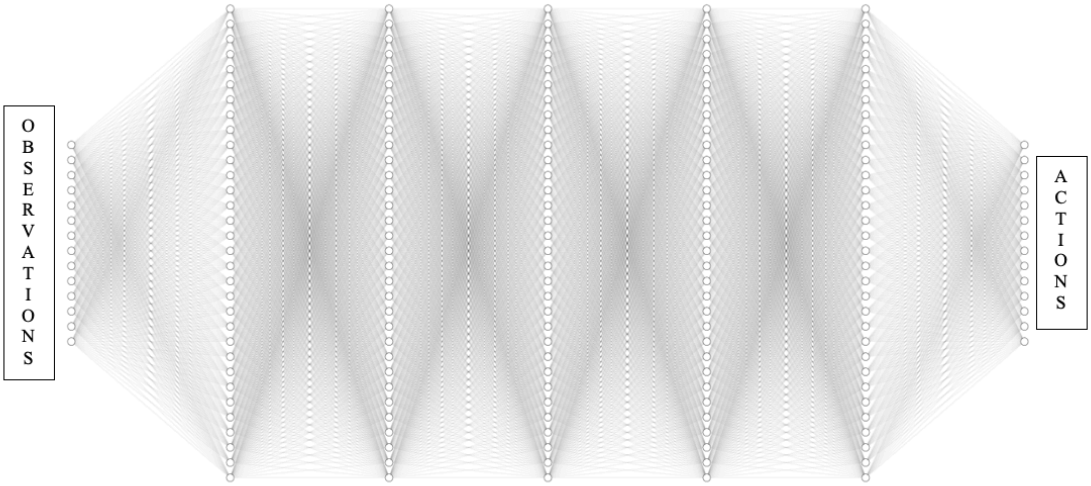

Figure 18. Neural network architecture

The entire training process was stable, consisting of increasing the average reward value and reducing the average number of steps per task. The author decided to end the training upon noticing that the agent stopped making noticeable progress which occurred at the stage of 40M*2048 steps in total, which means the component responsible for making decisions and learning, solved over 819200000 tasks in total. The following is a chart of mean reward evaluated every 20.000 steps, excluding few outliners marked on the chart with red circles, the whole process was stable which is confirmed by low value of mean square error equal to 1.513745e+02. Linear regression applied to that chart clearly proves improvement of agent's performance through the whole learning process with the mean reward steadily rising.

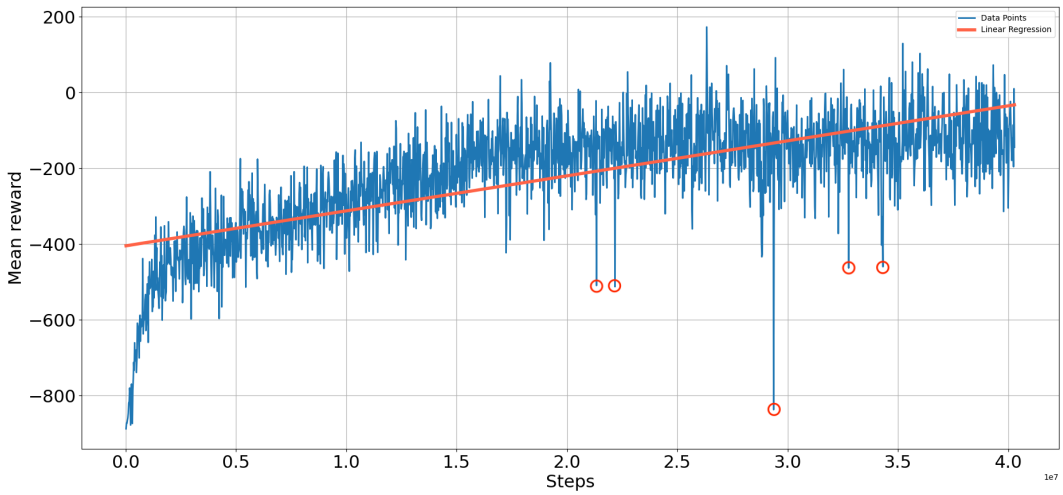

Figure 19. Distribution of average reward during the learning process

We can observe that mean number of steps required to finish given task was steadily descending as well, with mean square error of 5.922784e+00. General trend of decreasing number of steps necessary to solve task is well presented by linear regression on the chart below.

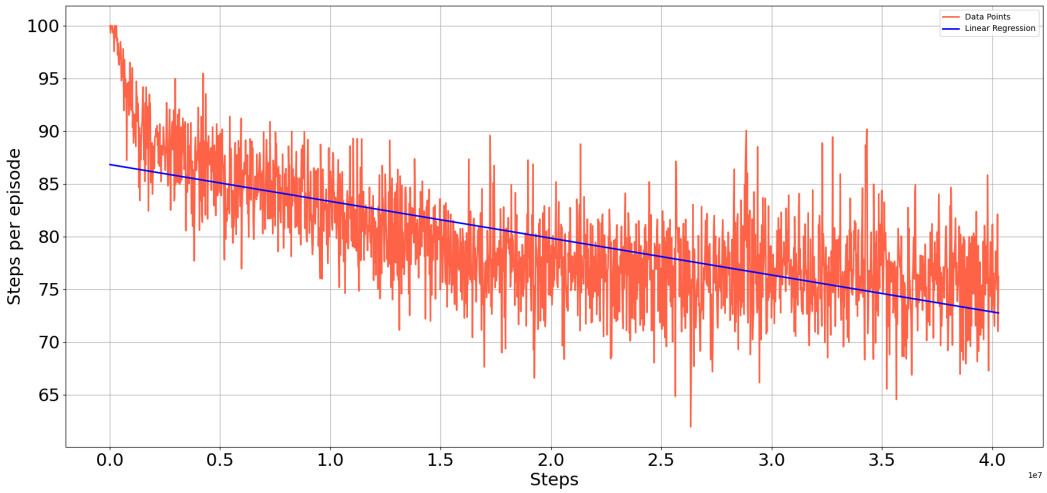

Figure 20. Distribution of average numbers of steps during the learning process

At the end of the training process author tested the agent's performance at various stages of its development starting with agent trained to 1milion steps (*2048) and finishing with agent trained to 40 million steps with intervals of one million steps in-between. The agent at each level of development was given 1000 task to solve. The agent achieved final efficiency of 42%.

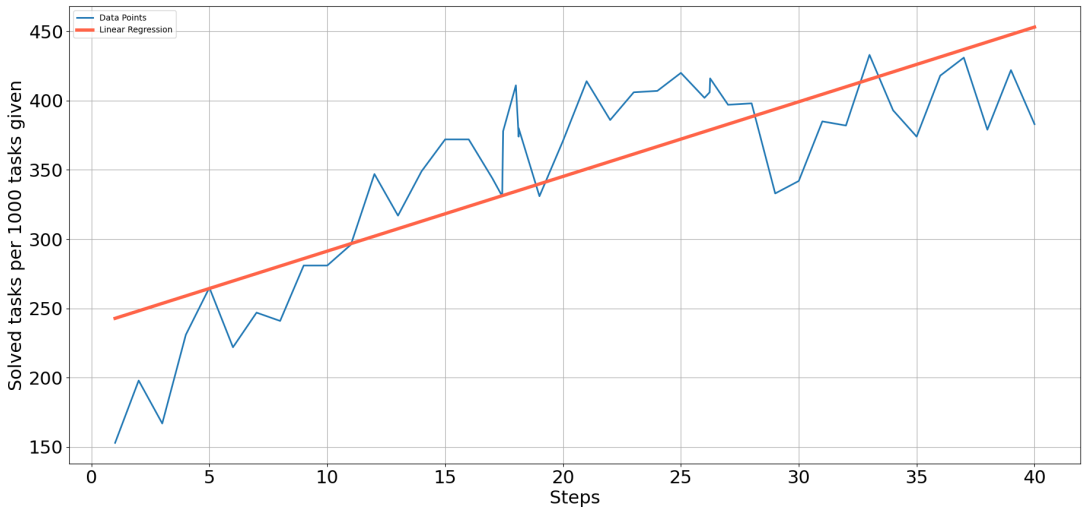

Figure 21. Agent's efficiency during the learning process

The final efficiency of 42% has been reached by implementing default PPO Stable Baselines 3 settings with neural network consisting of 5 fully connected layers with 32 neurons each. Although the final parameters were chosen through trial and error approach, by experimentation with various neural networks architectures and settings as well as reward policies, the in-depth analysis of more models has been unobtainable due to the limit of available computation power. Therefore, given more time and computational power, it might be possible to reach higher efficiency. Further development of the program and implementation of more variables (such as listing, or whether conditions) in each task may even serve as a stepping stone into development of a stability software applicable on board of vessels.

9 Potential further development of the deep reinforcement learning stability software

In this paper the author managed to create model of B354 bulk carrier vessel in such a way that it meets requirements of OpenAi Gym environment which allowed to use the Stable Baselines 3 package to handle deep reinforcement learning algorithms. There were many simplifications along the way to accommodate for restrictions regarding computation power available. Nevertheless, the process of developing deep reinforcement learning vessel stability calculations software presented in the paper, aimed to be a proof of concept rather than a ready-to-use program, was successful. The Autor hopes that this project can be used as a starting point and solid foundation for future development of this kind of tools. It seems to be almost certain that for the right research team with proper resources, combining deep reinforcement learning with other, computational demanding mathematical tools like for example Monte Carlo tree search or Bayesian statistics will lead to higher effectiveness of this kind of software.

9.1 Monte Carlo method and Monte Carlo tree search

The Monte Carlo method is a computational method that is based on random sampling to obtain numerical results for problems that are essentially deterministic. It is particularly useful for solving complex problems with many variables. The core idea behind Monte Carlo method is utilizing random sampling to estimate results. In practice a large number of random samples are generated by running simulation to obtain approximate solution which becomes more accurate the more simulations are executed. (Graham & Talay, 2013, Chapter 1). Monte Carlo Tree Search is a particular application of the Monte Carlo method commonly used in decision-making processes, especially in the field of artificial intelligence and game algorithms. It involves creating tree structure which represents various possible sequences of actions, which is then being explored by the algorithm by choosing actions leading to desirable outcomes. Monte Carlo Tree Search demonstrates its effectiveness in scenarios where exploring all possible actions is nearly impossible. This method gained popularity thanks to DeepMind, an artificial intelligence research company, that utilized it in their AlphaGo and AlphaZero projects (Mnih et al., 2015; Silver et al., 2016; Silver et al., 2017).

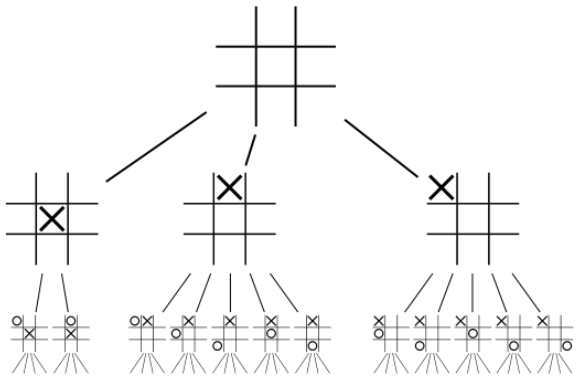

Figure 22. Applying MCTS for Tic-Tac-Toe game

9.2 Bayesian statistic

Bayesian statistics is a subfield of statistics that is based on Bayes' theorem. Unlike classical statistics (frequentist statistics), where probabilities are treated as constants, and the probability of hypothesis is not calculated, Bayesian statistics considers hypothesis as variables, which are subjects to updates in the light of new information (Gelman et al., 2014, p. 6; Koski & Noble, 2009, pp. 12-13). Paraphrasing, in Bayesian statistics, probability is being updated every time a new piece of data is collected, leading to updated probability that takes into account new information. Therefore, Bayesian statistics is extremely useful mathematics tool in a field of data analysis that utilizes prior beliefs as well as current evidence (Bolstad, 2007, Chapter 6.; Downey, 2013, Chapter 1; Gemerman & Lopes, 2006, Chapter 2). Thanks to this approach it is possible to draw conclusions from approximations, missing data or even inadequate information, as long as we keep updating our beliefs based on the inflow of new data. With this in mind, it can be noted that this method fits quite well into core of reinforcement learning, where the agent makes decisions based on feedback from the environment by interacting with it.

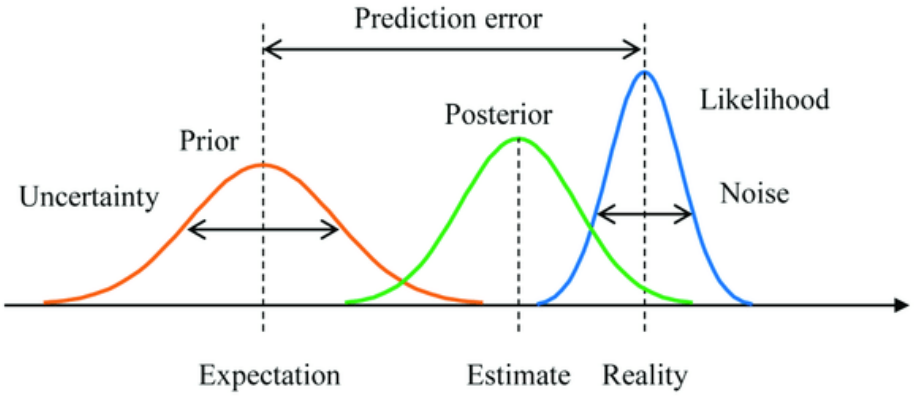

Figure 23. Bayesian Interference

10 Final conclusions

Machine learning is being developed rapidly all over the world and there is a lot of research going on in this field, yet there is still considerable room for improvement as well as further potential to be revealed. It is already widely accepted that deep reinforcement learning can be used to tackle complex problems that cannot be solved with conventional methods. Furthermore, deep reinforcement learning can provide entirely new solutions both in areas well known to humans, as well as in fields where humankind has not reached a high level of understanding yet. For the sake of this paper an attempt of utilizing deep reinforcement learning to perform stability calculations has been made. The results, although imperfect, suggest that further development of that technology may bring implementable automated solutions on board real-life vessels.

Looking even further into the future, development of an efficient solution in the field of stability branch for one type of cargo could be the first step in the development of tools of this kind which would then naturally lead to a universal solution that works on different ships with different cargo types in the future. For the best results we should focus on improving all the small pieces that together create maritime industry by providing remote and autonomous solutions. Once autonomous vessels are fully developed and the entire maritime industry takes advantage of artificial intelligence as well as remote solutions, the savings associated with those new tools will be significant. The human factor working at sea will be reduced to a minimum, maybe even to zero, and the efficiency of the whole industry will probably increase by several orders of magnitude. To sum up, it is hard to anticipate how long it will take for our civilization to reach such a level development. However, shipping will inevitably become fully autonomous in the next 1000 years, if not in the next 10 years.

As promising as they may seem, development and implementation of autonomous solutions come with an array of difficulties that all have to be tackled in order to ensure safety of the vessels. The first challenge is common for all new branches of science and industry, and it is the lack of specialized staff. We can create research teams made of engineers and seafarers, but the best solution would be having people with experience from sea, who are highly educated in engineering and computer science fields as well. Autonomous maritime industry combines highly digitalized solutions and applies them to a quite conservative maritime industry. Providing procedures and contingency plans for new potential risks, threats and emergencies is another challenge that has to be faced. New risks related to autonomous vessels will inevitably arise, and they have to be noticed beforehand, so adequate prevention plans must be implemented to avoid potential financial losses, ecological disaster or loss of life. Another important matter will be providing a high level of reliability through redundancy of implemented systems as well as standard operation procedures and contingency plans in case of emergency. Adapting port facilities to fully utilize autonomous ships will also undoubtedly present many challenges.

Other aspects that have to be taken under consideration are the security and safety of the entire autonomous solutions. When speaking of security in terms of automation in maritime industry, we have to divide security into cybersecurity and physical security of the ship and the facilities it cooperates with. At first the focus should be paid to providing high level of physical safety and security aspects, assuring that vessels and port facilities meet requirements of ISPS Code. Taking care of physical safety and security of the ship is a major, yet easily overlooked component. Cybersecurity matter is far more complicated, as there are already more digital solutions, software and wireless communication devices on board vessels than ever before and this trend is only accelerating. Those developments strongly rely on connectivity via Bluetooth and Internet which increases potential vulnerabilities and risks. International Maritime Organization (IMO) created Guidelines on Maritime Cyber Management, focused on cybersecurity. These guidelines aim to improve safety and security of people working at sea, as well as personnel being involved in shipping industry from the shoreside perspective (Boyes, 2021). Nevertheless, cybersecurity is a highly sophisticated, developed and demanding field, mostly due to the nature of computer science and IT technologies that it belongs to. Therefore, it grows exponentially both in size and advancement. Solutions that provide a safe working environment in the digital world today most likely will be outdated in the near future. Hence, once created and implemented,

cybersecurity measures have to be maintained and updated frequently. Maintaining cybersecurity at a high standard requires a lot of effort and know-how.

11 Reference list

Aggarwal, C.C. (2018). Neural networks and deep learning: a textbook. New York City: Springer

Auffarth, B. (2020). Artificial Intelligence with Python: cookbook. Birmingham – Mumbai: Packt

Barras, B., & Derret D. (2006). Ship Stability for Masters and Mates. Amsterdam: Elsevier

Bolstad, M.W., (2007). Introduction to Bayesian Statistics. New Jersey: John Wiley & Sons

Burkov, A. (2019). The hundred-page machine learning book, Andriy Burkov

Clark, C.I., (2002). The management of ship stability, trim and strength. London: Nautical institute

Downey, B.A., (2013). Think Bayes. Sebastopol: O'Reilly

Gelman, A., Carlin, J.B., Stern, H,S., Dunson D.B., Vehtari, A., Rubin, B,D., (2014). Bayesian data analysis. Boca Raton: CRC Press

Gemerman, D., Lopes, H., (2006). Markov Chain Monte Carlo - Stochastic Simulation for Bayesian Inference. Boca Raton: CRC Press

Graham, C., Talay, D., (2013). Stochastic simulation and Monte Carlo methods. New York: Springer

Hastie,T., James,G., Tibshirani,R & Witten,D. (2017). An Introduction to Statistical Learning with application in R. New York: Springer

Hughes, C.H. (1917). Handbook of ships calculations, construction and operation. New York: D.Appleton and company

Hurbans, R. (2020). Grokking artificial intelligence algorithms. Shelter Island: Manning

Kabaciński, J., (1993). Stateczność i niezatapialność statku. Szczecin: Akademia morska w Szczecinie

Koski, T., Noble, M.J., (2009). Bayesian networks an introduction. West Sussex: Wiley Publisher

Miłobędzki, J., (1963). Stateczność morskich statków handlowych. Warszawa: Wydawnictwo komunikacyjne

Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

Molland, F.A., (2008). The Maritime Engineering Reference Book - A Guide to Ship Design, Construction and Operation. Oxford: Elsiver

Morales, M. (2020). Grokking deep reinforcement learning. Shelter Island: Manning Publisher

Mousavi, S.S., Schukat, M., Howley, E,. (2018). Deep Reinforcement Learning: An Overview. Lecture Notes in Networks and Systems LNNS, volume 16), 426–440, 10.1007/978-3-319-56991-8_32.

Nandy, A., & Biswas M. (2018). Reinforcement Learning with Open Ai TensorFlow and Keras using Python.

Rawson, K. J., Tupper, E. C. (2001). Basic ship theory volume 1 - Hydrostatics and Strength. Oxford: Butterworth Heinemann

R. I. Hugh Boyes, "Code of Practice Cyber Security for Ships," IET Standards - Department for Transport, Accessed: Oct. 16, 2021. [Online]. Available: www.gov.uk/dft

Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). https://doi.org/10.1038/nature24270

Suttoon, R., & Barto A. (2018). Reinforcement Learning: an introduction. London: The MIT Press

Szozda, Z. (2016). Stateczność statku morskiego. Szczecin: Akademia Morska

Theodoridis, S. (2015). Machine learning: a bayesian and optimization perspective. Amsterdam: Elsevier

Trask, A.W. (2019). Grokking Deep Learning. Shelter Island: Manning Publisher

Zai, A., & Brown B. (2020). Deep reinforcement learning in action. Shelter Island: Manning Publisher

Appendix I

Vessel B354 basic information:

Table 1. B354 – basic data

Length overall	149m
Length between perpendiculars	140m
Breath	20m
Displacement(max)	20767t
Deadweight(max)	13593t
Gross Tonnage	11573
Net Tonnage	6179

Table 2. B354 Cargo holds for bulk/general cargo

	Name	Volume [m^3]
1	Hold I PS	303.7
2	Hold I STB	303.7
3	Hold II	2363.3
4	Hold III	3196.4
5	Hold IV	1834.6

Table 3. Ballast Tanks

	Name	Volume [m^3]
1	BALLAST TANK 3 PS	194.2
2	BALLAST TANK 3 STB	194.2
3	BALLAST TANK 3A PS	135.9
4	BALLAST TANK 3A STB	135.9
5	BALLAST TANK 4 PS	141.2
6	BALLAST TANK 4 STB	141.2
7	BALLAST TANK 4A PS	105.9
8	BALLAST TANK 4A STB	105.9
9	BALLAST TANK 5A PS	57.1
10	BALLAST TANK 5A STB	57.1
11	BALLAST TANK 6 PS	116.9
12	BALLAST TANK 6 STB	116.9
13	BALLAST TANK 7 PS	188.0

14	BALLAST TANK 7 STB	188.0
15	BALLAST TANK 8 PS	166.2
16	BALLAST TANK 8 STB	166.2
17	BALLAST TANK AFTERPEAK	43.2
18	BALLAST TANK FOREPEAK	343.2

Appendix II

Vessel state at the beginning of each task

Table 4. Set of values at the beginning of each task

D = 8990
X = 57.05
Y = 0.0
Z = 9.09
fsm = 1809.0
GMp = 1.37
T = 4.36
t = -7.0
T aft = 7.81
T fwd = 0.82
ZGp = 9.29
Sea_water_density: 1.025

Solutions to examples

The following are solutions to examples from chapter 6.

Solution to Scenario I:

Table 5. Scenario I - stability data at the end of the task

D = 14685
X = 69.59
Y = 0.0
Z = 7.38
fsm = 1946.7
GMp = 1.67
T = 6.78
t = 0.00
T aft = 6.78
T fwd = 6.78
ZGp = 7.51

Table 6. Scenario I – holds utilization

	Hold	Available volume [m^3]	Cargo [t]
1	Hold I PS	303.7	0
2	Hold I STB	303.7	0
3	Hold II	0	2363.3
4	Hold III	0	3196.4
5	Hold IV	1823.3	10.3

Table 7. Scenario I – ballast tanks utilization

	Ballast tank	Available volume [m^3]	Ballast water [t]
1	BALLAST TANK 3 PS	194.2	0
2	BALLAST TANK 3 STB	194.2	0
3	BALLAST TANK 3A PS	135.9	0
4	BALLAST TANK 3A STB	135.9	0
5	BALLAST TANK 4 PS	141.2	0
6	BALLAST TANK 4 STB	141.2	0
7	BALLAST TANK 4A PS	105.9	0
8	BALLAST TANK 4A STB	105.9	0
9	BALLAST TANK 5A PS	1	57.1
10	BALLAST TANK 5A STB	1	57.1
11	BALLAST TANK 6 PS	116.9	0
12	BALLAST TANK 6 STB	116.9	0
13	BALLAST TANK 7 PS	188.0	0
14	BALLAST TANK 7 STB	188.0	0
15	BALLAST TANK 8 PS	166.2	0
16	BALLAST TANK 8 STB	166.2	0
17	BALLAST TANK AFTERPEAK	33.4	10
18	BALLAST TANK FOREPEAK	343.2	0

Solution to scenario II

Table 8. Scenario II - stability data at the end of the task

D = 13932.5
X = 68.90
Y = 0.0
Z = 7.46
fsm = 1894.1

GMp = 1.63
T = 6.47
t = -0.5
T aft = 6.72
T fwd = 6.22
ZGp = 7.60

Table 9. Scenario II – holds utilization

	Hold	Available volume [m^3]	Cargo [t]
1	Hold I PS	303.7	0
2	Hold I STB	303.7	0
3	Hold II	0	1750.6
4	Hold III	0	2367.7
5	Hold IV	1251.8	431.7

Table 10. Scenario II – ballast tanks utilization

	Ballast tank	Available volume [m^3]	Ballast water [t]
1	BALLAST TANK 3 PS	194.2	0
2	BALLAST TANK 3 STB	194.2	0
3	BALLAST TANK 3A PS	135.9	0
4	BALLAST TANK 3A STB	135.9	0
5	BALLAST TANK 4 PS	141.2	0
6	BALLAST TANK 4 STB	141.2	0
7	BALLAST TANK 4A PS	105.9	0
8	BALLAST TANK 4A STB	105.9	0
9	BALLAST TANK 5A PS	57.1	0
10	BALLAST TANK 5A STB	57.1	0
11	BALLAST TANK 6 PS	2.66	177.1
12	BALLAST TANK 6 STB	2.66	177.1
13	BALLAST TANK 7 PS	188.0	0
14	BALLAST TANK 7 STB	188.0	0
15	BALLAST TANK 8 PS	166.2	0
16	BALLAST TANK 8 STB	166.2	0
17	BALLAST TANK AFTERPEAK	43.2	0
18	BALLAST TANK FOREPEAK	189.5	157.5

Solution to scenario III

Table 11. Scenario III - stability data at the end of the task

D = 13527.500000000002
X = 69.4783349007851
Y = 0.0546128257253742
Z = 7.617024059357824
fsm = 1833.879825
GMp = 1.52
T = 6.3
t = -0.03
T aft = 6.32
T fwd = 6.28
ZGp = 7.76
Sea_water_density: 1.025
Cargo_to_load: 0
sf_cargo: 1.35

Table 12. Scenario III – holds utilization

	Hold	Available volume [m^3]	Cargo [t]
1	Hold I PS	0	225
2	Hold I STB	0	225
3	Hold II	0	1750.5
4	Hold III	200	2219.5
5	Hold IV	1834.6	0

Table 13. Scenario III – ballast tanks utilization

	Ballast tank	Available volume [m^3]	Ballast water [t]
1	BALLAST TANK 3 PS	194.2	0
2	BALLAST TANK 3 STB	194.2	0
3	BALLAST TANK 3A PS	135.9	0
4	BALLAST TANK 3A STB	135.9	0
5	BALLAST TANK 4 PS	141.2	0
6	BALLAST TANK 4 STB	141.2	0
7	BALLAST TANK 4A PS	105.9	0
8	BALLAST TANK 4A STB	105.9	0
9	BALLAST TANK 5A PS	57.1	0

10	BALLAST TANK 5A STB	57.1	0
11	BALLAST TANK 6 PS	116.9	0
12	BALLAST TANK 6 STB	116.9	0
13	BALLAST TANK 7 PS	188.0	0
14	BALLAST TANK 7 STB	117.2	72.5
15	BALLAST TANK 8 PS	166.2	0
16	BALLAST TANK 8 STB	166.2	0
17	BALLAST TANK AFTERPEAK	43.2	0
18	BALLAST TANK FOREPEAK	299.2	45.1

Appendix III

State, observation and action space

Table 14. State Array

[0]	represents available volume in hold_I_Ps+Stb
[1]	represents available volume in hold_II
[2]	represents available volume in hold_III
[3]	represents available volume in hold_IV
[4]	represents available volume in ballast_tk3_Ps+Stb
[5]	represents available volume in ballast_tk3_A_Ps+Stb
[6]	represents available volume in ballast_tk4_Ps+Stb
[7]	represents available volume in ballast_tk4_A_Ps+Stb
[8]	represents available volume in ballast_tk5_A_Ps+Stb
[9]	represents available volume in ballast_tk6_Ps+Stb
[10]	represents available volume in ballast_tk7_Ps+Stb
[11]	represents available volume in ballast_tk8_Ps+Stb
[12]	represents available volume in ballast_tk_afterpeak
[13]	represents available volume in ballast_tk_forepeak
[14]	trim
[15]	cargo volume

Table 15. Action space

0	Load cargo to hold no.1 PS+STB
1	Load cargo to hold no.2
2	Load cargo to hold no.3
3	Load cargo to hold no.4
4	Ballast 3 PS+STB
5	Ballast 3A PS+STB
6	Ballast 4 PS+STB
7	Ballast 4a PS+STB
8	Ballast 5A PS+STB
9	Ballast 6 PS+STB
10	Ballast 7 PS+STB
11	Ballast 8 PS+STB
12	Ballast afterpeak
13	Ballast forepeak

Table 16. Agent hyperparameters

Hyperparameters type	Description
Artificial neural network	5 Layers, 32 neurons each, MlpPolicy
Learning rate	0.0003
Number of steps	40M x 2048
Mini-batches	64
Number of epochs	10
Gamma	0.99
Generalized Advantage Estimation Lambda	0.95
Clipping range	0.2