NTNU

Norwegian University of
Science and Technology

Deep Reinforcement Learning based tracking behavior for underwater vehicles

Abir Khan

Marine Technology
Submission date: June 2018
Supervisor: Ingrid Schjølberg, IMT
Co-supervisor: Anastasios Lekkas, IMT

Norwegian University of Science and Technology
Department of Marine Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER THESIS IN MARINE CYBERNETICS

Name of the candidate: Abir Khan

Field of study: Marine Technology

Thesis title: Deep Reinforcement Learning based tracking behavior for underwater vehicles

Background

With the industry is pushing further offshore to more complex and dangerous maritime conditions, this begs the questions of cost efficiency, health, safety and environment. A central solution to these questions resides in the increased focus on autonomy, especially underwater missions related to underwater vehicles.

Today, the industry traditionally employs human operators for control of both Remotely Operated Vehicles (ROVs) and Autonomous Underwater Vehicles (AUVs) in underwater operations. Not only is this practice costly, the mission also relies fully on the availability of these human operators. Most importantly, involving more operators on underwater missions in challenging maritime conditions is a high risk, which is crucial to minimize. By introducing increased autonomy within the underwater operations, more specifically render ROVs/AUVs fully autonomous, one has effectively reduced the need of human operators.

One way to increase autonomy in the underwater missions is by utilizing autonomous behavior for underwater vehicles. This thesis puts forward a deep reinforcement learning based tracking behavior for underwater vehicles. This property promotes increased autonomy in the sense of not being reliant on an extra operator for that specific underwater vehicle during a mission.

The use of AI, specifically machine learning, has been increasingly popular in the late years due to increased computer power. Especially does reinforcement learning hold an enormous potential as they are capable of exceeding human performance, which is important to highlight in complex tasks such as underwater missions. In this thesis a suitable reinforcement learning algorithm will be implemented to solve the tracking problem for a ROV. The implementation will be verified for performance in simulations. Ultimately, the implementation will be tested for performance in physical experiments conducted in the test basin at Marine Cybernetics Laboratory at Tyholt, NTNU.

Work description

1. Literature study on reinforcement learning with emphasis on the following
    - Basic concepts and challenges within reinforcement learning in general applications
    - State of the art reinforcement learning methods in light of robotic systems
2. Develop a reinforcement learning algorithm for a ROV system based on existing implementation for control tasks
3. Train the tracking behavior using the developed reinforcement learning algorithm on a simulator
4. Use the simulator results to verify and validate the tracking behavior

5. Test and verify the tracking behavior module through physical laboratory experiments at a test basin
6. Discuss and compare the results obtained from simulation and physical experiments
7. Write thesis a thesis on the simulation and experimental testing, along with a conclusion and suggestions for further work

Specifications
The thesis is written in English, where a preface is included, followed by a summary in both Norwegian and English, which is in accordance to the guidelines for a masters thesis from NTNU. In the introduction of the report the background and motivation for the thesis are further detailed. A small-scale review on related work on the topic is provided. The outline of the rest of the thesis is also included in the introduction. The main section of the thesis comprise of literature review, description of mathematical models, presentation of reinforcement learning implementation and simulator setup, presentation of simulation results and experimental results. Finally, discussion and conclusion is in order, with suggestions for further work provided. Source codes and a PDF version of the report shall be included electronically along with the submission.

Start date: January 15th, 2018 **Due date:** June 25th, 2018

Supervisor: Ingrid Schjølberg

Preface

This report is a result of a master thesis in Marine Technology with specialization within Marine Cybernetics at the Norwegian University of Science and Technology. The main motivation for this work is to investigate the possibilities for Deep Reinforcement Learning as a basis for control systems design in underwater vehicles. The control sequence that is specifically to be designed is a autonomous tracking behavior for underwater vehicles with the use of Computer Vision technology as a key feature. This approach has the potential to, not only reduce the severe engineering workload of manually designing such control systems, but also increase autonomy in underwater operations. This concept is also easily importable to other types of vehicular systems, such as cars, autonomous ships and aerial drones.

This thesis is an continuation of a Project Thesis carried out in autumn 2017, and the period of work ranges from January to June 2018.

The assumed background of the reader should be an education within control engineering, marine hydrodynamics, and basic knowledge in Machine Learning.

Acknowledgement

The work is accomplished with the support from my supervisor Ingrid Schjølberg. I would like to thank Ingrid for her optimism and help. I am grateful to Ph.D candidate Bent O. Arnesen and researcher Tore Mo-Bjørkelund for introducing me to the BlueROV2, and spending a lot of their time on debugging softwares with me. Also, a special thanks to postdoc Mikkel Cornelius Nielsen for his assistance in the MC-lab, guidance in the thesis work and simulator development.

Lastly, my kindest thanks to my family and friends which has been encouraging me throughout my years of study.

Abir Khan

Trondheim, June 20, 2018

Summary

This thesis introduces the use of Machine Learning, specifically Reinforcement Learning, to create a model-free tracking property for Remotely Operated Vehicles (ROV). In detail, the ROV is trained by a RL algorithm to track an aruco marker, using online implementation of a Computer Vision (CV) algorithm as a detection property. The main motivation behind this enterprise is the contribution to increased autonomy in underwater operations, by introducing model-free autonomous tracking behavior to underwater vehicles. This approach of implementation requires minimal human intervention during operation, while significantly reducing prior human control programming effort. Firstly, a simulator based tracking behavior training of the ROV was done prior to conducting physical experiments with a real ROV in the MC-laboratory at NTNU. The ROV used for the experimental tests is a BlueROV2, which is highly customizable and fitting for R&D purposes.

The theory presented in this thesis lays the groundwork for the many reasonings done in this project's course, including the choice of RL method. The RL algorithm chosen for training the tracking behavior is a online Python implementation of the type Proximal Policy Optimization (PPO) algorithm. The tracking behavior is trained on a simulator, which is a Python script based on typical OpenAI's simulator architecture. The resulting tracking performance is then evaluated by studying the evolution of accumulated rewards and ROV's trajectory plots. While the resulting performance did show to have some weak sides, it was, however, feasible enough to test the trained model in a real-world setting.

However, the real-world experiments did not yield positive tracking results, considering the ROV performed in a random manner instead of favorably moving towards the aruco marker. Several challenges described in the theory-section proved to be prevalent during the lab experiments, which caused the disruption in the real-world tracking performance. Nonetheless, based on experience gained from both the simulations and real-world experiments, various proposals for further work was devised and highlighted. Especially, is the importance of appropriate reward function design underlined.

Sammendrag

Denne oppgaven introduserer maskinlæring, spesifikt forsterket læring (Reinforcement Learning, les: RL), til å utvikle sporingsadferd hos undervannsfarkoster (ROV). Med andre ord, ROV'en trenes opp av en RL-algoritme for å spore en aruco-kode under vann ved hjelp av en maskinvisjon-basert deteksjonsalgoritme. Hovedmotivasjonen bak dette prosjektet er å bidra til økt autonomi i undervannsoperasjoner ved å introdusere modell-fri autonom sporingsadferd hos undervannsfarkoster. På denne måten reduseres menneskelig intervensjon hos undervannsfarkosten i undervannsoperasjoner. I tillegg reduseres arbeidsbelastningen for mennesker da en slipper manuelt å utvikle et slikt kontrollsystem. Til å begynne med ble ROV'en trent opp i en simulator på forhånd før den ble testet i virkelige verden i MC-laboratoriet ved NTNU. ROV-typen som er brukt til laboratorietestene er BlueROV2 som er enkelt å modifisere. Dette gjør den utmerket til forskningsprosjekter slik som denne masteroppgaven.

Den omfattende teorien som er tatt for seg i denne oppgaven er grobunnen til mange av beslutningene gjort i dette prosjektets forløp. Blant annet er valget av RL-algoritme basert på argumenter tat for seg i teoridelen av oppgaven. Den valgte RL-algoritmen er en Python-basert implementasjon anskaffet fra internettet og er av typen Proximal Policy Optimization (PPO). Sporingsadferden ble trent opp i en simulator som er et Python-basert skript. Simulatoren baserer seg på en standard simulator-arkitektur fra OpenAI. Sporingsadferden fra simulasjonsopplæringen ble vurdert i ettertid ved å studere utviklingen av akkumulert belønning og plottene av ROV'ens baneløp. Selv om den resulterende opplærte sporingsadferden hadde noen svakheter så var den egnet til å testes hos en ekte ROV i laboratorieforsøk.

Sporingsadferden under laboratorieforsøkene ble derimot ikke vellykket siden ingen lovende opptreden ble observert. Isteden for å bevege seg mot aruco-koden som ønsket oppførte ROV'en seg vilkårlig under eksperimentet. Utfordringer som var beskrevet i teoridelen av oppgaven ble derimot støtet på under laboratorieforsøkene, noe førte til begrenset mulighet for optimal sporingsadferd. Likevel ved hjelp av praktisk erfaring ervervet fra simulasjoner og laboratorieforsøkene ble det utarbeidet forslag til videre arbeid. Spesielt er viktigheten av riktig type belønningsfunksjon (reward function) understreket.

Contents

List of Figures

List of Tables

1 Introduction

A Remotely Operated Vehicle (**ROV**) is an unmanned vehicle that is utilized in deep subsea environments. Typically, ROVs are tethered by an umbilical which supplies fiber optics and electrical power rendering a feasible communication line and data transfer possible to the operator. Hence, the ROV is controlled by the operator from either a surface-vessel or inland control room. The ROV types are divided into classes which specifies the amount of supporting equipment it can be equipped with. Apparatus such as sensors and mechanical tools and the physical ROV size play a major factor in defining which type of class it belongs to. For high working class ROVs, manipulators are normally applied for physical work during subsea operations. While as for the lighter classes, commonly a smaller ROV size with fewer mechanical capabilities is typical.

An Autonomous Underwater Vehicle (**AUV**) is an untethered unmanned vehicle that is also operated in deep subsea environments. AUVs are, like ROVs, operated from a surface vessel or an inland control room. However, since AUVs are not supported by umbilicals, a sufficient battery is utilized for power supplies. In addition, transponders are used for communication and navigation purposes. Unlike ROVs, AUVs are not constrained by the length of a tether, but only by the length of battery life. This renders surveillance operations a typical field of application for AUVs as it is less power demanding, in contrast to the power-draining intervention tasks ROVs normally conduct.

1.1 Subsea underwater robotic operations

The World's ocean covers three fourth of Earth's surface and is essential field of utilization for commercial purposes, scientific research and resource extraction. Specifically, Norway is in the possession of high technological activities within naval and ocean engineering while exploiting the said fields heavily on offshore industries such as oil and gas production, and aquaculture and fisheries.

1.1.1 Underwater robotic operations in oil and gas industry

The continental shelf outside of the Norwegian coast is rich in natural gas and oil, but is, however, situated below the seabed in the violent North Sea. Thus, a feasible solution is to transfer the oil and gas production to the seabed where the

subsea installations are placed. Thereby, severely reducing the production cost as an alternative to a more expensive surface production facility on land.

Subsequently, as the oil and gas production facility is moved to the seabed, the need for underwater robotic vehicles has grown exponentially. Nowadays, the oil and gas industry is reliant on ROVs for numerous IMR operations, which is a common term for inspection, repair and maintenance (IMR) operations. Typical ROV tasks in a oil and gas facility are subsea construction and maintenance, surveillance of subsea site, pipeline inspection for fault location, drilling support, and other various types of operations.

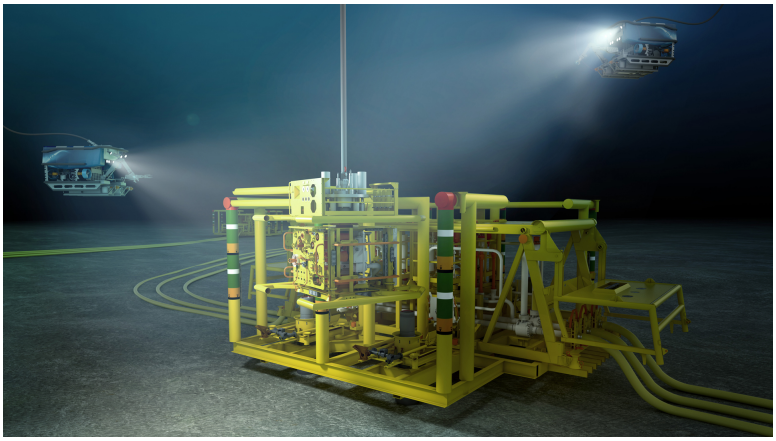


Figure 1: ROVs during an IMR operation. Credit: Aker Solutions.

Now that the oil and gas activity is expected to be pushed further offshore in the near future, the need for increased autonomy in subsea operations is essential for saving cost in future endeavors.

1.1.2 Underwater robotic operations related to aquaculture

(Autonomous Underwater Vehicle) The aquaculture and fishing industry is becoming important because fish farming is expanding rapidly as world's seafood supplier. In this case, employing ROV's and AUV's as a production support for the fish farms and active surveillance tool can lead to cost reduction and increased quality of the product. Application of underwater vehicles in aquaculture means that inspection of fish cages can be preformed both systematically and more autonomously, without the need for human divers.

Small ROV's or AUV's are typically employed for assuring the integrity of mooring

lines, detecting net holes in fish cages, surveillance of fish behaviour and other aquaculture infrastructure. The increasing aquaculture industry is leading to increased demand for autonomous underwater operations related to the usage of underwater vehicles. Hence, rendering an improvement in Health, Safety and Environment (HSE), and severe cost reduction by being less dependant on expensive operators.

1.2 Underwater robotic control

1.2.1 State of underwater vehicle operations

The control related to underwater robotics encompasses both underwater vehicles or with manipulators attached, and is considered to be a challenging issue due to unpredictable external disturbances and non-linearity. There has been developed control methods for various motion sequences for unmanned vehicle such as sliding mode control (SMC) [6], Proportional-Integral Derivative controller (PID), robust/optimal control and many more have been proposed and applied for underwater robots.

Today most of the state of the art ROVs are controlled manually by an operator, where only a few or none automatic control actions are utilized. Subsequently, the ROV operation performance is heavily dependant on the operator expertise. This renders the long term effects of repeatedly employing ROVs for essential underwater operations, e.g. IMR operations, to accumulate a huge expense for the industry. Hence, the need for increased ROV autonomy in the industry is desired leading to a reduction in workload for the operator. Moreover, increasing autonomy in ROV motion control will minimize human errors and increase operation efficiency. Autonomy, in this sense, specifies the ROV's ability to execute actions independently by utilizing available information during a mission. According til Schjølberg and Utne (2015) [62], autonomy can be regarded as one of the key factors that will optimize IMR operations in oil and gas industry. Current ROV models retain a large potential for increased autonomy which, if remedied, will increase their potential in other offshore industries also.

AUVs on the other hand, are typically by default more autonomous than ROVs. The former is normally assigned a task beforehand which it is to conduct. Typically, this would be travelling in a specified pattern (e.g. lawnmower pattern) within an area in which the deployed acoustic transponders encloses. For example, behavior such as object avoidance is extensively utilized in cases where the AUV is operating in

autopilot. This is an essential feature as it is critical for danger avoidance, and thus saving the industry considerable amount of effort and resources. Other autonomous functions such as Dynamic Positioning (DP), velocity control and path tracking are also being further developed and improved, which are utilized for both ROVs and AUVs. These functions grant the operator the possibility to emphasize more on data extracted by the cameras and sensors.

However, the main challenge concerning such autonomous functions is to make them reliable enough such that they can be applicable for motion sensitive operations, for example IMR operations for ROVs. Improvements are constantly accomplished and new methods are developed as an increased step towards autonomy in underwater operations. Naturally, allowing reduced effort for the operator so that he/she can focus more on important tasks.

Traditionally, an autonomous functionality is based on feedback-loop controller functions, that requires accurate state estimation to act upon. Figure 2

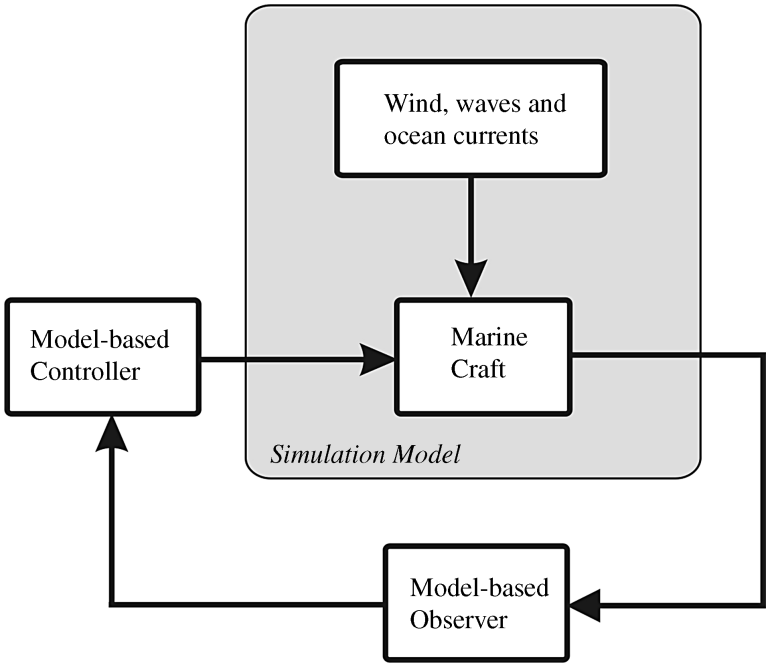


Figure 2: Traditional feed-back loop. Credit: Fossen (2011) [25].

Such control methods requires an accurate model of the underwater vehicle dynamics, that is kinetics and kinematics. Which can be costly to design.

1.2.2 Introducing Machine Learning in robotic control

Lately, there has been an increased number of applied autonomous robots in the industry that aims to operate in unknown and unstructured environments. Recent advances done in technology allows for actuators, sensors and mechanical devices to be applied in autonomous systems platforms that can sense and act in environments such as air, ground, underwater and space [17].

As modern robotic systems are assigned to increasingly complicated tasks, both onshore and offshore, a rapid development in robotic techniques needs to be carried out in order to meet the required brainpower. Manually designing a reliable control system for autonomous robotic tasks is considered a complicated process, even for people specialized in robot programming, as stated by Xia and El Kamel (2016) [79]. The sheer number of situations with high uncertainty a robot must encounter represents a spectrum of well-defined behaviour it has to perform. Consequently, the job of programming such an advanced robot is considered difficult, time-consuming and expensive for the experts. Hence, instead of manually programming all the behaviours on a robot, it could be rather advantageous if the robot could learn the tasks by itself.

Recent researches have introduced Machine Learning (ML) into the robotics community in order to improve the robots' autonomous capabilities based on acquired experience. Reinforcement Learning (RL), which is one of ML methods, has especially increased in interest [45]. This is due to its abilities to exceed human performance when provided enough training session, and properly designed. Basically, the RL system learns by interacting with the environment, then changes its policy regarding action selection by evaluating its feedback signals, thus, converging towards its desired behaviour. According to Lin et al. (2010) [45], a RL algorithm, namely Q-learning, is favorably applied in many robotic systems. RL algorithms have yielded numerous successes in performance, proving the potential for usage of ML in robotic system. This paves the way into a different approach to the robotic problem at hand, providing a huge space for searching for the optimal ML algorithm in robotic systems.

1.2.3 Related work

Recent advances have been made in ML based underwater vehicle control systems as well. This subsection covers the latest developments in control systems for un-

manned vehicles that are both fully and partially based on ML algorithms.

Carreras et al. (2005) [16] presented a behavior-based scheme using high-level RL based control of autonomous underwater vehicles (AUVs). The paper puts forward two main features approaches, which are hybrid coordination that includes the benefits of both competitive and cooperative approaches, and the utilization of Semi-Online Neural-Q-learning (SONQL). The latter feature is a RL approach which combines Q-learning algorithm with a multilayered neural network to learn specific behaviours state-action mapping online. Among the predefined generic behaviours for the AUV were transition from point to point, strolling, navigation, hovering, global-path/way-point generation, and obstacle avoidance. The paper concluded positively that the experimental results showed the feasibility of this approach for AUVs.

Ahmadzadeh et al. (2013) [2] presents a ML algorithm that is specified for an autonomous valve turning task for AUVs that renders the control system partially ML based. To remedy this challenge, the paper proposes a three-layer hierarchical scheme, where each layer is responsible for specific sub-tasks to improve the autonomy of the system. The first layer acquires and teaches the robotics skills of approaching and grasping the valve based on kinesthetic form of learning. The second layer devices a Reactive Fuzzy Decision Maker (RFDM) which takes the relative movement between the valve and AUV into account. Third layer implements apprenticeship learning method, which uses expert knowledge to tune the RFDM. The paper successfully applied the proposed approach to the task of valve turning yielding a promising feature to the AUV.

A fully ML based underwater vehicle controller is put forward in El-Fakdi and Carreras (2013) [24] for a cable tracking task. The ML approach is a RL control system, specifically, a Natural Actor–Critic (NAC) algorithm used for solving the action-selection problem of a cabled AUV. The visual based cable tracking task is a learning process that is split into two steps. The first step is where the policy is computed by the means of simulation environment, in which a hydrodynamic model of the AUV is to learn the task of following the cable. Once the simulated results converges to be accurate enough, the second step is set to motion where the simulation-learned policy is transferred to the real AUV. Hence, continuing the learning process in a real environment and improving the previously learned policy.

Lin et al. (2010) [45] proposes a SNQL algorithm as a motion controller for a bionic underwater robot. The main motivation for the paper is to successfully carry out experiments of bionic robot swimming straight forward using SNQL algorithm.

Instead of being propelled by thrusters, two undulating fins are utilized, resembling a robotic fish in motion. Subsequently, to speed up the learning process and increase safety, supervised control was introduced in the earlier stages of learning.

Several other notable researches that are recognized are done by Andrew Ng at Stanford University. Along with his apprentices, a series of RL control system are developed for Remotely Controlled (RC) helicopter control. Abbeel et al. (2007) [1], in collaboration with Andrew Ng, used a pilot to fly the helicopter in order to help find the helicopter dynamics, then utilized a RL algorithm to find a controller that is used to optimize the behavior of the resulting model.

1.3 Thesis objective

This thesis investigate the use of ML as an apparatus for increased autonomy in underwater robotics, by putting forward autonomous tracking behavior on a ROV based on RL and computer vision technology. Specifically, the object of interest is an aruco marker that the ROV is tasked to track. This thesis focus on the implementation of the RL algorithm, training and verification of the tracking performance through simulations, and lastly, conduct real-world testing of the trained model with BlueROV2 and use of computer vision technology.

1.4 Overview of the thesis

The thesis content and structure is given as the following.

Chapter 2 is committed to the literature review. This chapter covers the fundamental basics of ML, with emphasis on RL, to more in-depth details on RL algorithms that are widely used in the industry. In addition, RL algorithms that are highly relevant to underwater vehicles application are also presented as a proposition for the curious reader. Lastly, a set of relatable RL challenges to this thesis (i.e. continuous robotic systems) are highlighted. This chapter contains knowledge in which later chapters lay the groundwork for line of thought and reasonings for pathways, especially during the implementation of RL algorithm.

Chapter 3 provides an overview of mathematical modeling of the ROV. Both kinematics and kinetics are highlighted in this chapter and are essential for understanding the motion of the ROV during the simulations.

Chapter 4 presents the implementation of the RL algorithm and the simulator. This chapter explains the composition of the simulator and its settings in terms of optimal training of the ROV tracking behavior. The architecture of the RL implementation is also detailed, and the reward function design is also presented.

Chapter 5 presents the simulation training results. They are further discussed.

Chapter 6 is committed to the real-world experiments. Here, the main components essential for the physical tests in the MC-laboratory are presented. An overview of relevant software frameworks are also put forward. A brief description of the experimental cases performed during the physical tests are also in place.

Chapter 7 covers the results from the real-world experiments. A discussion surrounding the real-world experiments is covered, along with a comparison to the simulation results.

Chapter 8 draws a conclusion based on the results and discussions, and proposes further work.

2 Machine Learning: A Literature Review

ML is currently applied on multiple fields, such as robotics [38], processing natural languages [81], finance [23], and most notably internet advertisement [72]. To fathom the line of thought in this thesis, one must grasp the fundamentals of ML. This section introduces the theoretical concepts and mathematical aspect of some of the fundamentals in ML and key algorithms that are cutting-edge in terms of performance for use in robotic control. Thereby, laying the basis for understanding the line of thought when implementing the ML control system for the ROV in chapter 4 in this thesis.

The concept of ML can be divided into three subgroups, and they are namely *supervised learning*, *unsupervised learning* and *reinforcement learning*. Where supervised learning and reinforcement learning are considered *feedback based* ML. Feedback in this sense is understood as a reward the robotic system receives as a result after a single action taken or as a result of the system's performance as a whole.

The main goal of ML is to learn the mapping between the situations of the robotic system and the correct action in a given situation. In the sense of ML, a common term for the learning robotic system is *agent*, which is prompted to optimize its behavior to a desirable performance. To guide the learning process, a so called *reward function* is introduced to punish and award the agent's behaviour. This way, the best actions will lead to a maximum accumulation of rewards.

In **supervised learning** (SL) the agent is to learn the function mapping between the state-observations and actions based on knowledge of an already known set of observation-action pairs. After an extensive training session with feedback received from a human supervisor, the agent can then, without any further feedback from the supervisor, autonomously execute the right action for each observed state resulting in a desired performance.

In **unsupervised learning** (UL) the agent does not collect any explicit feedback, nonetheless, it learns the pattern in the data-inputs. A widely used form of unsupervised learning is *clustering* [56], which observes the most potential useful clusters of the input examples. Specifically, clustering is favorable in cases such as shopping websites utilizing this method to classify their potential customers based on their internet browsing history. Consequently, clustering will arrange customized ads, or targeted offers, in order to bait the customer to purchase from the store.

In **reinforcement learning** (RL) the agent receives reinforcement, that is reward

or punishment, during the operation. This helps express that the agent has done something erroneous when it receives punishment after executing an action in a given state. An example could be that a robotic manipulator, which is due to grab an object, instead grabs free air. Hence, a punishment will decrease the chance for the agent to repeat such mistake in the future, and vice versa for the case of receiving a reward where the agent will be most likely to repeat such rewarding action again for that given state. RL has shown to be very feasible for applications in real-world robotic systems such as self-driving Google Car [18], RL based control system for helicopters by Kim et al. (2004, supervised by Andrew Ng) [36], and lately by Wu et al. (2017) [78], a RL based depth controller of AUV. RL is also a powerful tool as it is shown to outperform even human skills when properly designed.

In this thesis a RL algorithm is implemented as the ML based tracking algorithm, subsequently, this chapter will emphasize the literature study on RL.

The literature on RL is vast and growing in a continuously. Here an effort is made to present the fundamental theoretical background in RL, and the most recent developments and state of the art algorithms that are suitable for continuous vehicular locomotion systems, for example ROVs. This chapter concludes with presenting the challenges related to continuous RL robotic systems, such as *Curse of Dimensionality*, *Exploration vs Exploitation dilemma* and *Curse of real-world samples*. These challenges are highly imperative to understand when encountering issues during the implementation, training and real-world experimenting phases in this thesis.

2.1 An intuitive approach to Reinforcement Learning

As previously stated, in ML the goal is to learn the mapping between situations and correct actions for the situations, and RL is no different in this case. The agent starts off with a blank memory, thus no knowledge on which steps to take, but must by itself discover which actions that needs to be taken in order to maximize its rewards. RL learning can be intuitively understood by explaining it through an analogy where a dog is learning to fetch a ball.

Consider a dog learning the process of fetching a ball.
1. The dog has to notice how the humans fetches the ball properly.
2. A challenge that comes along with the process of fetching the ball is to first be able to **run**. This skill has to be learned and improved so that the dog may run after the ball.

3. When step two is remedied, the dog actually needs to **pick up the ball** with its mouth when it finally reaches it.
4. The last step is to return with the ball in its mouth to the owner. However, in this process there are many things to consider. That is maintaining the ball in the mouth, maintaining a fitting running speed and form, upkeep a well-behaved motor skill set, and so forth.

The **agent** in this sense would be the dog with the **goal** of fetching the ball. The agent endeavors to maneuver from one **state** to another by exploiting the **environment**, that is the surface on which the dog is so eagerly running on. For each sub-task the dog completes, it receives a **reward** as a form of feedback. Thus, bad performance will naturally result in scolding or no reward at all. Subsequently, the dog would know when to minimize the chance of repeating a certain behavior or not.

This analogy helps understand the line of logic in a RL process. For humans, many details and sub-processes that constitute the overall process of fetching a ball are so deeply embedded in the subconsciousness that they are neglected. However, for an inexperienced dog these sub-processes and details are heavily focused on when the aforementioned steps are confronted.

Sutton and Barto (1998) [68] illustrates a simplified model of a RL process in the following figure.

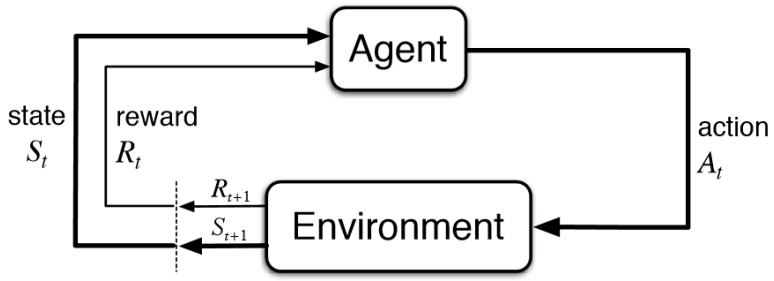


Figure 3: RL process overview (simplified). Sutton and Barto (1998) [68]

2.2 Markov Decision Process

A Markov Decision Process (MDP) is discrete-time state-transition methodology, and formalizes the backbone of a typical RL problem. Specifically, MDPs specifies how an agent plans and acts in face of uncertainty [52]. An MDP consists of five

main parameters that describes the central aspects of a RL problem, and are as following.

- S - The **state** in which the agent inhabits. For example a ROV may be in a certain position in the pool with a certain velocity and orientation.
- A - Set of **actions** the agent is able to execute. For example assign thrust force such that the ROV may turn or change velocity.
- R - Represents a **reward** function which maps each state to a reward.
- P - Corresponds to the probabilities of **transition** between states. Intuitively, what the chances are for doing such maneuver in a given state and obtain the expected result.
- $\gamma \in [0, 1)$ is the discount factor.

A MDP process may be better grasped by an example, where an agent starts off in a state s_t and performs an action a_t. Consequently, the agent will transition to state s_{t+1}, with the probability denoted by P_{s_t,a_t}. Thus, the agent receives a reward from the reward function R accordingly. However, to check that the agent does not over-complicate a task by performing unnecessary set of actions, a discount factor γ is introduced. The discount factor determines how much each reward is weighted based on the time it was received. A γ close to zero assigns early rewards to higher value, while γ close to one means that the time it was awarded is almost irrelevant.

An essential assumption that lays the foundation for a MDP is that the environment satisfies the *Markov property*, which means that the state s_{t+1} is only dependant on the action a_t taken in previous state s_t. In other words, the next state is only dependant on the current action and state while previous actions in the past are neglected. Subsequently, it is possible to estimate the next state and rewards based on the current state. Sutton and Barto (1998) [68] argues that methods of RL based on MDP may be applied even if this property is approximated.

The parameter that decides which action should be executed in what state is denoted as **policy** π, which is a function that maps each state $s \in S$ to an action $a \in A$. Formally, $\pi : S \to A$. A policy is considered good if the action selection yields highest total accumulated rewards, and vice versa for bad policies. Thus, in order to separate the good policies from the bad policies, a **state value function** V^π, or an **state-action value function** Q^π, is introduced. The state-value function V^π is the expected reward the agent receives by following a policy π from a

state s_t. This is mathematically represented in equation 2.1.

$$V^{\pi}(s) := E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_t = s] \tag{2.1}$$

The action-state value function Q^{π} is defined correspondingly by considering initial action a_t. In other words, what the expected return will be when starting in state s_t, taking an action a_t and then follow a policy π. Formally, it is presented in the following equation as defined by Sutton and Barto (1998).

$$Q^{\pi}(s, a) := E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_t = s, a_t = a] \tag{2.2}$$

As previously underlined, the main goal of RL is to maximize the accumulated reward over time. This means finding such policy that maximizes the value functions. This makes policy a key factor in a successful RL implementation. Sutton and Barto (1998) [68] proves that the maximization process can be done by exploiting a recursive relationship of V^{π}, as shown in following equation 2.3.

$$V^{\pi}(s) = \sum_{a \in A} \pi \sum_{s' \in S, r} P(s', r | s, a)[R(s, a, s') + \gamma V^{\pi}(s')] \tag{2.3}$$

Equation 2.3 is known as the *Bellman equation*. It highlights how the value of a state is related to the value of the succeeding state. The value function V^{π} is the unique solution to the Bellman equation, yet several policies may posses the same V^{π}. Hence, this allows for equation 2.3 to form the basis of solving the value function V^{π} in multiple ways.

A policy that yields maximum accumulated reward is denoted as **optimal policy** π^*, such that $V^{\pi^*(s)} \geq V^{\pi}(s)$ for all policies π and all states $s \in S$. Furthermore, it can be proven that optimal state value function, V^*, is equal to the largest state-action value function with respect to optimal policy π^*. This is shown in the following equation 2.4.

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s, a) \\ &= \max_{a \in A(s)} \sum_{s', r} P(s', r | s, a)[R + \gamma V^*(s')] \end{aligned} \tag{2.4}$$

This is known as the *Bellman optimally equation* for V^*, and can also be defined for the state-action value function as shown in equation 2.5.

$$Q^*(s,a) = \sum_{s',r} P(s',r|s,a)[R + \gamma \max_{a'} Q^*(s',a')] \tag{2.5}$$

However, solving a MDP can be ambiguous and problematic in terms of learning efficiency and time. In the next subsection, some common tabular methods are presented, that stores the state- and state-action values in tables.

2.3 Tabular approaches of solving MDPs

Storing the learning data in tabular form has the fundamental assumption that the state-action space is deemed small, normally discrete. Hence, the terminology **finite MDP** is used for such cases, and their approximation to the value function can be stored in the tables, which in many cases makes it possible to find the optimal function.

2.3.1 Dynamic Programming Methods

The idea behind Dynamic Programming methods is to organize the search for optimal policy by utilizing the value function. This method assumes a perfect model of the environment. Hence, known as *model-based method* which will be explained in subsection 2.5.2. The principle in Dynamic Programming method is used in *policy iteration* and *value iteration* algorithms.

Policy iteration method is made up by two parts, policy evaluation and policy improvement. Both of which are carefully presented in Sutton and Barto (1998) [68]. This algorithm initializes a policy for the agent, so that the policy evaluation determines a value function for the policy using the Bellman equation. Each state is then explored so that the value function of the state is updated based on the value of its succeeding states, along with the transition probabilities and the policy itself. Subsequently, this process will repeat until the value function converges to a fixed value. Thus, the policy improvement is initialized, which will then select the best action, known as *greedy selection*, in each state, hence, creating a new fixed policy.

Value iteration method, on the contrary, does not wait until the value function converges before computing the value function. This method rather improves the policy on the go. The value iteration method shortens the evaluation part of the policy iteration to a single step. Consequently, integrating the policy improvement in each iteration, emphasizing only on estimating the value function directly.

The major disadvantage with Dynamic Programming methods is the requirement of exploring the entirety of the MDP state space, which is typical for model-based methods. Thus, convergence is hard to achieve if the state space grows large, especially large enough to be considered continuous state space. However, *Asynchronous Dynamic Programming* is introduced to counteract this drawback. In contrast to the systematic approach of the Dynamic Programming methods, policy and value iteration, Asynchronous Dynamic Programming backs up the value in random order by using the available values at that time instant. Thus, some values may be backed up multiple times while other values are not backed up yet. A criteria for the algorithm to fully converge is that all states must be backed up after some time. The benefit with Asynchronous Dynamic Programming is the possibility for the agent to back up selected states which will most likely lead to an optimal policy.

2.3.2 Temporal Difference Methods

Unlike Dynamic Programming methods, Temporal Difference (TD) methods is a *model free method*, and does not require an explicit model of the environment. However, similar to Dynamic Programming method, this method also estimates the values of the states based on estimates of the succeeding states, known as *bootstrapping*. The most elementary TD method is TD(0) which is not used in control, but lays the foundation for RL-based control algorithms. The mathematical representation of TD(0) is defined in equation 2.6. Where α is the learning rate and γ is the discount factor.

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{2.6}$$

This method examines the transition between the states, and learns the value of each state. Equation 2.6 presents how the value of current state, $V(s_t)$, is updated using the difference between the estimated value function of the next state. The current state and the reward r_{t+1} is given in the next state. This term is denoted as TD-error, and can be thought of as the "wake-up call" the agent experiences when

it moves to a new state and realizes the large gap between its estimate value against the actual value of the state. The TD-error can be formalized as follows:

$$\delta_t := r_{t+1} + \gamma(V(s_{t+1}) - V(s_t)) \tag{2.7}$$

TD methods are particularly used in two well known control algorithms which are the on-policy method SARSA, and the off-policy method Q-learning. An off-policy approach is known as updating the state-action value, $Q(s_t, a_t)$ using the state-action value of the next state while retaining a greedy action selection policy. As for the on-policy approach, $Q(s_t, a_t)$ is updated using the value of $Q(s_{t+1}, a_{t+1})$ and following the current policy for action selection.

SARSA is built on the generalized policy iteration, presented in Sutton and Barto (1998) [68]. However, for the policy evaluation a TD-method is used. It separates itself from TD(0) method by taking the transitions between state-action pairs into account, and learns the values of these. Otherwise, the methods are identical. SARSA state-action value update law is given as follows.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.8}$$

Similar to the update law for TD(0) in 2.6, the state-action value are updated after each state transition. However, if S_{t+1} is a terminal state, $Q(s_{t+1}, a_{t+1})$ is equal to zero. A terminal state is understood as the agent's final desired state which yields reward equal to one, while other states yield zero reward.

Watkins (1989) [77] introduced Q-learning, which uses $Q(s_t, a_t)$ directly to approximate the optimal state-action value $Q(s_t, a_t)*$ without regarding for which policy is being followed. In the case of discrete state space, Q-learning algorithms will converge if all states and actions are tried. The state-action value update law is defined in equation 2.9.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)) \tag{2.9}$$

2.4 Function approximation approaches for solving MDPs

Approaches for solving MDPs in the previous section assumes a relatively small state space which allows for storing the results in a tabular representation. However, for larger and continuous state spaces this quickly becomes impractical. For example, discretizing a 2D ROV state space which consists of three DOFs, where both the position and velocity must be accounted for. This yields a total of six unique parameters describing a single ROV state. Even using coarse discretizing dimensions in a small spatial area, the total number of unique states could rapidly reach 50 million different states. ROVs are typically equipped with multiple actuators, thus, a combination of the actuator effort represent an action. Thereby, when including the ROV set of actions into the state space, rendering it to action-state space, the table grows exponentially larger. Dealing with such large tables requires large amount of storage and computational capacity to search, compute and process all the stored data. This phenomena in ML is known as *Curse of dimensionality* and will be further detailed in section 2.7.2.

The core obstacle with discretization is the inability to generalize states. In other words, each state is treated as a complete distinction, which leads to that near identical states are assumed unknown until explored. However, *function approximation* remedies this issue which generalizes the continuous state space, rendering it possible for the agent to operate in a large state spaces.

A function approximator uses values from the full state space to design a function which returns approximately that of the original function. Subsequently, function approximators can be compared to SL where a function is used to map between the inputs and outputs as introduced in this section. Function approximators can be applied in state value functions V^{π} or state-action value functions Q^{π}, the policy π, or the model itself which represents the reward functions and state transitions. Function approximators can be classified in two sub-categories, linear and non-linear function approximators, which will be detailed further in the following subsections.

2.4.1 Linear Function Approximation

A linear function approximator performs the generalization by projecting the full set of states to a lower dimensional space where the original function can be approximated by a linear function. In this case the function that is to be approximated is the state-value function $V(s)$ and state-action value function $Q(s, a)$. However,

in this section only $V(s)$ is considered as the approximation function of interest. The following equation represents the linear function approximator which can be described by a feature vector ϕ, and a weight vector θ.

$$f(x) = \theta^T \phi(x) \tag{2.10}$$

The weights have to be tuned to minimize the error between the function approximator and the original function. Typically, the performance is measured by utilizing the Mean Squared Error over an arbitrary distribution μ of the inputs. Where μ indicates how much error in each state is valued. The mathematical representation is shown in equation 2.11.

$$MSVE(\theta) = \sum_{s \in S} \mu(s)[V^{\pi}(s) - \hat{V}(s, \theta)]^2 \tag{2.11}$$

To maximize the performance, the local minimum of MSVE remains to be found. This can be remedied by introducing *gradient decent methods* which are widely utilized in domain of RL. Especially, Stochastic Gradient Decent (SGD) method is prevalent and is thoroughly introduced in Sutton et al. (2009) [70] and Mnih et al. (2013) [49]. For more complex approximators, convergence typically takes place towards a local minimum. However, for simple linear function approximators it might be possible to obtain a global minimum.

A widely used linear function approximator is the Cerebellar Model Arithmetic Computers (CMAC), introduced by Miller et al. (1990) [47]. CMACs, also known as *tile coding*, can be viewed as a complex table look-up tool. Figure 4 shows the nature of tiling structure in CMACs.

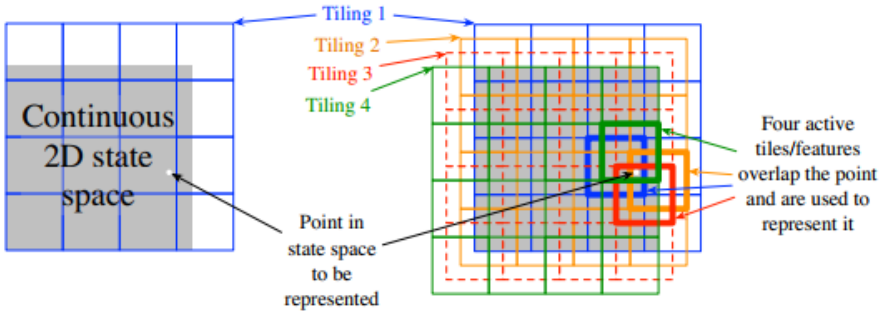


Figure 4: Overview of tile coding. Sutton and Barto (2012) [69].

This methodology divides the state space into multiple overlapping tilings that are slightly displaced from each other while the tilings are divided into tiles. The tiles contains a binary feature such that the overlapping tiles can create a feature vector. These feature vector are utilized to define the value of each point on the overlapping tiles. CMAC as a function approximator has been extensively used by prominent figures in ML, amongst whom Sutton et al. in (1996) [67] and (1997) [58]. Timmer et al. (2007) [73] utilized a modified CMAC function approximator, namely A-CMAC, which proved convergence guarantee if used along with Q-learning. However, this approach also resulted in reduced performance.

An implementation tool that is well known for its usage with function approximators is the Radial Basis Function (RBF). It allow for continuous valued features in the range $[0, 1]$. The RBF feature transforms the tile representation from binary value to a response in the form of Gaussian distribution in accordance to equation 2.12.

$$x_i(s) := exp(-\frac{||(s - c_i)||}{2\sigma_i^2}) \tag{2.12}$$

The feature width is defined by σ, and $||(s - c_i)||$ is the distance between the state s and the feature's center state c_i, which can be selected by the designer. The RBF implementation yields a smooth and differentiable function approximator, but computational cost is immensely increased. Sutton and Barto (2012) [69] showed a decrease in performance in environments with more than two dimensions, making it less feasible for robotic implementations.

2.4.2 Nonlinear Function Approximation

Linear function are utilized for their stability properties and good predictive behavior. However, when the state space becomes more complex as in robotic environments, a linear function approximator may not be feasible due to its inability to grasp the desired approximation function. Subsequently, this would yield a divergent result. On the other hand, nonlinear function approximators such as Artificial Neural Network (ANN), and its extension to Deep Neural Network (DNN), are possible options. Yet, as discussed in Tsitsiklis et al. (1996) [75], both ANNs and DNNs have recently been struggling with divergent results. Since the use of both ANNs and DNNs have been the basis for many recent developments in RL, they will both be emphasized in this section.

The ANNs, inspired by the structure of human brain, consist of a network of a

interconnected *artificial neurons*. The structure of ANNs consists of an input layer, a set of hidden layers and an output layer. Figure 5 illustrates an example of three-layered ANN structure, and the arrows between each layers are called *links*, which represents a real-valued weight that is multiplied with the output of the neuron. A network with one hidden layer is known as a *shallow network*, while network with more than one hidden layers is known as *deep neural network*.

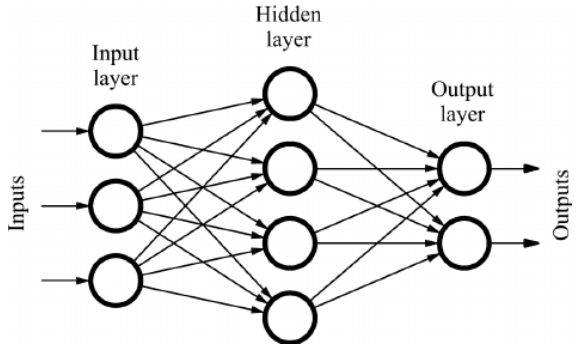


Figure 5: Feedforward Neural Network, extracted from Boyd et al. (2011) [11]

The neuron calculates a weighted sum of their inputs and the values is then injected in the *activation function*. Each neuron is equipped with an activation function which is used to render the calculated the weighted sum of neuron inputs into a range from zero to one. Basically, the introduction of activation functions serves as a measure of how positive the relevant weighted sum is. A common activation function used in the ANN is the S-shaped sigmoid function, and is prevalent in use for SL algorithms. The simgoid activation function is formulated in equation 2.13.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

However, a recent trend discovered by Sze et al. (2017) [71] shows that Rectified Linear Unit (ReLu) has an increasing popularity due to its simplistic usage and fast training abilities. In addition, it also makes it possible to utilize more than one activation function in ANNs. Thereby, this allows for different activation functions per layer.

Before ANN is utilized, it must be trained in advance, that is updating the weights. Typically, stochastic gradient method (SGD) is used for learning the ANN, where each weight is updated with respect to optimizing the performance assessment. In such cases, the measurement of performance can be in the form of TD-error or the expected reward.

Bengio et al. (2007) [7] showed with experimental results that deep supervised neural network with gradient-based methods, such as backpropagation algorithm, tend to converge towards local minimums. Bengio et al. (2009) [8] further supported this statement by proving that DNNs with more than two hidden layers seems to preform less better than DNNs with fewer hidden layers. Yet, recent research done by Mnih et al. (2013) [49] proved that the DQN algorithm contradicts the experimental results of Bengio and his associates. This resulted in a large scale increase in newly developed ML methods using DNNs.

As stated, DNNs are defined by ANNs with more than one hidden layer. Cybenko (1989) [20] put forward that a single-layered ANN with a finite amount of neurons, each using sigmoid activation function, is able to represent any continuous function on a compact region of the network's input. This begs the question of why it would be necessary to apply DNNs as it may possess instabilities. The answer to this question is explained in Dellalleau and Bengio (2011) [22] and Liang et al. (2016) [42], where the former used deep sum-product networks while the latter used neural networks. Both papers proved that shallow networks required an exponentially larger amount of neurons than DNNs in order to approximate the same function. Thus, by utilizing DNNs, a less significant amount of data is required than a large single-layered shallow neural network, making the computational workload much less wearing. Consequently, this renders DNNs to a key player in ML as a function approximator for continuous environments, even though it suffers from instabilities as discussed in Tsitsiklis et al. (1996) [75].

However, the introduction of DQN by Mnih et al. (2013) [49] has remedied the many stability- and divergence related issues when using DNNs. An addition to DNNs is the utilization of the concept named *experience replay* originally proposed by Lin (1993) [44]. This concept reuses previous experience when learning while at the same time breaking the similarity the training samples. This leads to a reduction in probability of statistical *overfitting* and correlation of experiences. Overfitting will be further discussed in chapter 4. In light of robotic usage, when the agent interacts with the environment, each experience is stored as a tuple of parameters (s_t, a_t, r_t, s_t) to a replay memory. During a training session, a minibatch of these tuples are randomly extracted from the replay buffer and employed to train the neural network instead of the current experience. The concept of experience replay has been further developed, such as the Prioritized Experience Replay proposed by Schaul et al. (2015) [61]. This rates sample transition in the replay memory based on TD-error, and replay important transitions more regularly. This approach outperformed the uniform replay in 41 out of 49 games when using the DQN algorithm and Atari

as benchmark. Hindsight Experience Replay was introduced newly introduced in 2017 by Andrychowicz et al. [3]. This method incorporates goals to the experience replay, which considerably assists the progress in environments with sparse or binary rewards, hence bypassing the need for complicated reward engineering. Even though experience replays has been beneficial against stabilization problems, Zhang et al. (2017) [82] discussed that experience replays used inaccurately may lead to serious decline in performance. Subsequently, experience replays must be considered as a hyper-parameter which requires careful tuning.

Ioffe et al. (2015) [33] introduced the *batch normalization* as an improvement to DNNs. In a nutshell, this improvement provides solution to the changes in distribution which takes place in each of the layers in DNN while training. Ioffe et al. coined this phenomenon as *internal co-variate shift*. Batch normalization ensures that each layer receives whitened input which has also been used on DDPG algorithm, which is introduced in subsection 2.6.2. This concept increases the learning rate for the agent, thereby leading to a reduction in training time.

2.5 Reinforcement Learning in Large State and Action Spaces

This section puts forward state of the art RL algorithms that are designed for robotic systems operating in continuous, or near-continuous, action-state spaces. For such cases, a RL based ROV control system fits the profile perfectly. By employing the fundamental methods presented in sections 2.2−2.3 and the function approximation techniques in section 2.4.2, it forms the principals for highly complex RL control systems. These are heavily applied on robotic systems which yields good performance.

2.5.1 Policy search method

In contrast to value function based methods as discussed in previous sections, in policy search method, the policy is the most important formulation instead of value. The policy search based approaches find the optimal solution by searching for the optimal policy. This approach yields many features in robotic applications while allowing for an integration of highly trained knowledge, by e.g. policy structure and initialization. Kirk et al. (1970) [37] has proven that local policy searches can result in a good performance with hill-climbing approaches. Policy search has a good ability to easily include constraints, for example the possibility to regularize

the change appearing in path distributions. For this reason, policy search methods fit naturally in robotic applications.

However, Kober et al. (2013) [34] asserts that policy search methods is deemed to be the harder problem to solve. The paper argued that this is due to the inability for the maximum return function 2.14 to provide the optimal solution directly.

$$max_{\pi} J(\pi) = \sum_{s,a} \mu^{\pi}(s)\pi(s,a)R(s,a) \tag{2.14}$$

Here J represents the return, which combines the reward $R(s,a)$, policy $\pi(s)$ and state distribution in order to obtain the optimal behaviour of the system. The state distribution keeps track of the states and ensures that the states are well-defined. Equation 2.14 is also widely known as the *optimization problem*. In value function based methods, the optimization problem is remedied by using the Lagrangian approach, ending up with the value function $V(s)$. This is where the drawbacks with policy search methods may occur, that is if there cannot be obtained any suitable optimization method for the problem. Hence, leading to a worst case scenario where an comprehensive search for the optimal policy have to be conducted. Bosinou et al. (2010) [14] compared such a search to a Q-learning algorithm which resulted in 50 times higher computational cost for the policy search.

However, policy search methods possesses much better scalability capabilities than value function based methods as proven by Kober et al. (2013) [34] and Nagabandi et al. (2017) [51]. Scalability property is understood as the extent the algorithm performs feasibly when the number of state-action space grows in scale (e.g. scale to continuous state-action space). This is one of the key reasons to why policy search has grown to be competitive alternative to value based methods. Yet research conducted by Li et al. (2017) [41] has shown that value function based deep RL methods prove to operate with feasible performance in very large state spaces. Even close to continuous state spaces. This research contested any perception of value function based methods not being able to operate in very large state spaces.

Generally, most policy search methods seem to optimize locally, surrounding existing policies π, which are parametrized by a set of policy parameter θ_i in a policy parameter vector $\boldsymbol{\theta}$. A change in policy parameter $\Delta\theta_i$, which is calculated, increases the expected return $E\{J\}$ and the following update in policy has the following iterative form.

$$\theta_{i+1} = \theta_i + \Delta\theta_i \tag{2.15}$$

Equation 2.15 represents a crucial step in the policy search method, that is the policy update computation. A great variety of policy update computation proposals has been put forward such as pairwise comparison by Strens et al. (2002) [66], gradient estimation using finite policy differences in Buesing et al. (2016) [13] (and Roberts et al. (2016)), and general stochastic optimization methods in Deisenroth et al. (2013) [21].

Another promising discovery by Li et al. (2016) [40] is an algorithm developed based on off-the-shelf guided policy search RL, which allows for learning an optimization. Thus, making it possible for algorithms to autonomously discover algorithms that will improve the performance. With regard to speed of convergence and the final objective, as a result this approach outperforms any hand-engineered algorithms.

2.5.2 Model-free and Model-based Policy Search

Within policy search methods, the two classes of both model-free and model-based methods uses the concept of *sample trajectory* which basically initiates a policy π in a given state 1, denoted $s1$. That will cause an action execution, and in turn yield a reward. This reward will be used to update the value function $V_k(s1)$ at point k. This update form is known as *temporal difference update* which is also extensively used for updating value function.

Model-free policy search methods directly learn their policy from sampling trajectories that are based on stochastic trajectory generation. Specifically, the trajectories $p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$ are generated by sampling from the agent, e.g. robotic system, and the policy π. Thus, a previous learned model is not required as the agent is able to acquire it through experience. However, in the case of model-based policy search methods, a sampled trajectory is not acquired from the agent itself, but from previously learned environment. Which can be obtained, for example from a computer simulated trajectory learning, where the environment is fully explored. Then the sample trajectory is imported to the agent and the model of the state dynamics for the robotic system is built, and ready for use.

Both **stochastic** and **deterministic approaches** are suitable for model-based methods. In RL sense, a deterministic view on an action will imply that the performed action will affect the state space the same way each time. Thus, is deterministic approach suitable for software based RL algorithm such as computer simulations. For the stochastic approach, physical control systems for robotic agents are suitable as the real-life outcome of an action is not guaranteed to be the exact same

each time. The stochastic approach is applicable in model-free methods since the learned models are used as simulators for the sampling trajectories, and the robot can be replaced with the learned model with similar robotic dynamics. However, in the case of deterministic approach, the trajectories are not sampled, but analytically predicted and the trajectory distribution $\mathbf{p}_\theta(\tau)$ is obtained. Deisenroth et al. (2013) [21] further adds that even though the deterministic approach for predicting trajectories requires more computation than its stochastic counterpart, the policy gradient can easily be acquired. Since the policy gradient can only be approximated in stochastic approach, the deterministic approach is at an advantage as it can find it directly.

Model-based methods remedy the issue of sample inefficiencies through instead learning the samples by learning a complete model of the environment, typically on a simulator. When the environment is learned, a RL algorithm is then applied on the agent to find the most optimal solution in the pre-learned environment before even initiating its first action. It can be viewed as the RL problem is transformed to a path planning problem. As such, the robotic system quickly acquires the optimal solution which outperforms any other solutions existing in the environment, while saving time as fewer real-world interactions is required for training.

Despite for all the advantages, model-based methods still faces a weighty set of challenges that remains to be remedied. Specifically for cases with continuous domains, which often requires simulator based model pre-learning, the biggest issue is that the simulator-learned model is not identical to the real-world environment. Despite how much effort is put to design a realistic simulator, it remains rather an approximation. This challenge will be further detailed in section 2.7.4. Subsequently, utilization of models in continuous domain RL has been limited because inaccurate environment may be learned from the simulation-learned models. In which case physical parameters may have been assigned dubious values. While as for ROV's, faulty physical parameters such as negative friction, coefficients and masses may be derived. This results in a poor learned policy as the policy search algorithms will exploit the inaccurate parameters.

2.5.3 Actor-Critic Methods

Actor-critic methods are designed such that the policy is explicitly represented independently from the value function. In this algorithmic structure, the critic's role is to evaluate the action taken by the actor. Such learning form is classed as

on-policy algorithm since the critic learns on-the-go about the current policy while evaluating the actor. The TD-error is fundamental for all of the learning processes taking place in the actor-critic methods as the critic is based upon it. The role of TD-error is depicted in Figure 6 which illustrates the process of an actor-critic algorithm.

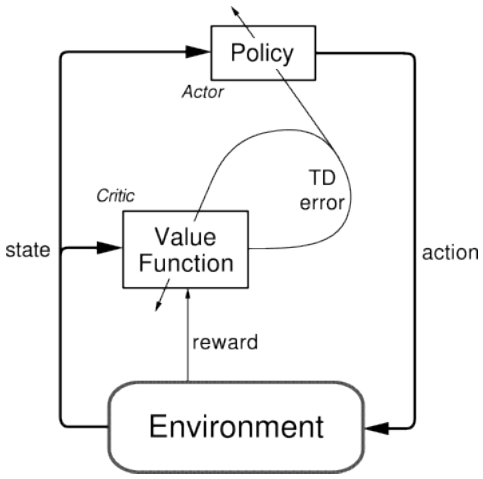


Figure 6: Illustrative overview of actor-critic architecture. Sutton and Barto (1998) [68].

From the figure, it can be noticed how the value function bases on the TD-error feedback, which is estimated by the critic's value function and current reward. The TD-error is delivered to the actor which is prompted to update its estimates on the most rewarding actions in the current residing state.

The purpose of the TD-error is to specifically evaluate action a_t executed in state s_t. If the TD-error turn out to be positive, the probability of choosing a_t in the future is more probable. Should the TD-error be negative, however, the tendency of choosing a_t will be decreased in the future. Value function update based on TD-error is formulated in following equation as presented in Bhatnagar et al. (2008) [10].

$$V(s_t) = V(s_t) + \alpha\delta \tag{2.16}$$

α is the learning rate of the algorithm. Equation 2.16 is basically a combination of equation 2.6 and 2.7. While the actor can be, among many alternatives, presented as the Gaussian probability distribution with stochastic parameters managed by a mean μ and variance σ.

$$\pi(s,a) = \frac{1}{\sqrt{2\pi}\sigma(s)} e^{-\frac{(a-\mu(s))^2}{2\sigma(s)^2}} \tag{2.17}$$

Parameters μ and σ are functions of state s, thereby the parameters will change over time as the policy changes.

While another widely used stochastic actor representation is *Gibbs Softmax* method as shown in equation 2.18.

$$\pi_t(s,a) = P_r(a_t = a | s_r = s) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}} \tag{2.18}$$

The parameter $p(s,a)$ indicates the tendency of choosing action a in state s, and are the values of the actor's policy. After an action is initiated, $p(s,a)$ will then be updated in the following manner.

$$p(s_t, a_t) = p(s_t, a_t) + \beta\delta_t \tag{2.19}$$

With β being a positive step-size parameter. While the critic is employed to observe and evaluate the actor's performance, it also determines whether the policy needs to be updated or not. The policy update is done using the update law in equation 2.19. For instance, the update step can be based on the *policy gradient method*. This method assumes that there exists differentiable presentations of a predefined class of stochastic policy, and thus ascend the measurement of the policy's performance. For instance, policy gradient formulation can be presented by the following formula obtained from Heidrich-Meisner et al. (2007) [30].

$$\rho(\theta) = \sum_{s,s' \in \mathcal{S}, a \in \mathcal{A}} d^\pi(s)\pi(s,a)\mathcal{P}^a_{s,s'}\mathcal{R}^a_{s,s'} \tag{2.20}$$

The policy gradient method assumes that a stationary state distribution exists, and is represented as $d^\pi(s) = \lim_{t\to\infty} Pr(s_t = s | s_0, \pi)$ in the equation. However, the performance gradient $\nabla_\theta \rho(\pi)$ is expressed by the policy parameters θ, which in turn is estimated by interacting with the environment. The *policy gradient theorem*, that is equation 2.21, can be derived based on Q and d^π, and determines the policy gradient.

$$\nabla_\theta J^\theta = E\left\{\nabla_\theta \rho\right\} = \left\{\sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta(s,a) Q^\pi(s,a)\right\} \tag{2.21}$$

Action-state value $Q(s, a)$ is unknown and needs to be estimated using, for example by using a function estimator $f_{\mathbf{w}}$ as discussed in Heidrich-Meisner (2007) [30].

Peters and Schaal (2008 c,b) [55] introduced a **natural policy gradient** by modifying the policy gradient approach using the Fisher information matrix $F(\theta)$. This method yields faster convergence resulting in reduced learning time and reduced required real-world interaction. Other advances done in policy gradient based actor-critic methods is DDPG [43], DQN [49], asynchronous advantage actor-critic AC3 [48] and NFAC [85].

2.6 Suitable methods for robotic navigation control

Since the real-world environment is a continuous state space, the robotic controller must also operate in continuous state spaces. While in some cases action-state spaces depends on the robot dynamics. As previously stated, coping with continuous environments is one of the main areas of challenge in RL. While recently, several highly promising developments in ML have shown that RL algorithms have proven to be suitable for robotic control, even surpassing traditional feedback-based control. This subsection introduces the RL algorithms which have been proven to be feasible in robotic control, or that is deemed arguably promising as a robotic controller.

2.6.1 Partially Observable Markov Decision Processes

A elementary assumption in a MDP (see section 2.2) is that the environment is entirely observable. In other words, the agent has full overview of its current state, without any uncertainties. Therefore, the optimal policy is only dependant on the current state. As for the cases where the environment is **partially observable**, such as real-world robotic computer vision interaction, the situation is more blurred. In Partially Observable Markov Decision Processes (POMDPs) cases, the agent does not possess complete information about its current state, and for that reason, it may find a hard time executing any policy $\pi(s)$. Not only does the policy in POMDPs depend on the state s, but also on *how much the agent knows* when it is in state s. This framework renders POMDPs much more complex than MDPs, and is widely applied in RL robots operating in continuous real-world environments. Specifically, is POMDPs prominent in robotic navigational tasks since the environment is expected to change randomly for each time-step. For instance, a ROV cruising through rocky underwater environment, or self-driving car are prime exam-

ples where the environment would only be partially observable through computer vision cameras and positioning sensors installed on the vehicles.

When it comes to parameters, the POMDP approach has the same parameters as MDP, which are the set of actions $A(s)$, the transition model $P(s'|s,a)$, and a reward function $R(s)$. However, in addition to these aforementioned MDP-parameters, POMDP also includes a *sensor model* $P(e|s)$ which represents the uncertainty related to the sensors. The latter parameter indicate how much the agent knows when it is in state s. In the sensor model $P(e|s)$, the parameter e indicates the probability of perceiving evidence in state s. In this formulation, the noisy measurements and uncertainty linked to sensor perception influence the probabilistic values of evidence e in a given state s, i.e. $P(e|s)$.

The POMDP has another property which is the **belief state** b. The belief state b is probability distribution which covers all possible states in which the agent might be in, given a current state s. Therefore, is the belief $b(s)$ probability an expression of state s. The belief state is calculated by filtering methods, also known as "state estimation", where the objective is to filter out noise from the sensors by estimating the underlying properties. Sardag et al. (2006) [59] proved that a Kalman filter, which is a commonly used filtering technique, has proven to work with POMDPs. This renders the agent to perform a reliable belief state estimation. Equation 2.22 shows how the basic recursive nature of filtering is used to calculate the belief state b.

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= \alpha P(e_{t+1}|X_{t+1} \sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1_t}) \\ &= \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t}) \end{aligned} \tag{2.22}$$

However, in cases for POMDPs, actions needs to be taken into consideration in the calculations while obtaining the same results. Considering that $b(s)$ is a previous belief state, and the agent executes an action a and perceives an evidence e, then the new belief state $b'(s')$ is calculated as follows.

$$b'(s') = \alpha P(e|s') \sum_{s} P(s'|s,a) b(s) \tag{2.23}$$

Where α is a normalizing constant which renders the probability function of the belief state sum to 1.

Russell et al. (2002) [56] emphasizes that the fundamental understanding in POMDPs is that *the optimal policy is only dependant on the agent's current belief state.* Subsequently, the agent does not depend on the actual state s itself, and that the

optimal policy $\pi^*(s)$ is directly mapped from belief states $b(s)$ to actions a. As a result, the decision cycle for a POMDP agent can be understood as:
1. Given a current belief state $b(s)$, execute an action a.
2. Assess the perception evidence e.
3. Update the current belief state using equation 2.23 and repeat.

Considering the case of calculating the probability of an agent residing in belief state b, which then takes action a and shifts to next belief state b'. The following perception evidence e in the next belief state b' is currently unknown. Therefore, the agent may appear in various possible belief states that are dependant on the registered perception evidence e. Provided an action a is performed in a belief state b, the perception evidence e probability $P(e|a,b)$ is given by summing over all probabilities of the actual states s' the agent might enter. The following mathematical formulations are thoroughly described in Russel et al. (2002) and are presented in equation 2.24 and 2.25.

$$\begin{aligned} P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e|s')P(s'|a,b) \\ &= \sum_{s'} P(e|s') \sum_{s} P(s'|a,b)b(s) \end{aligned} \tag{2.24}$$

As for the case, where action a and current belief state b are given, the probability of reaching the next belief state b' is then denoted as $P(b'|b,a)$.

$$\begin{aligned} P(b'|b,a) = P(b'|a,b) &= \sum_{e} P(b'|e,a,b)P(e|a,b) \\ &= \sum_{e} P(b'|e,a,b) \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)b(s) \end{aligned} \tag{2.25}$$

Equation 2.25 can be viewed as a transition model between current belief state b and the next belief state b'. Moreover, a belief state has its corresponding reward function which is defined in the following equation.

$$\rho(b) = \sum_{s} b(s)R(s) \tag{2.26}$$

Both equations 2.25 and 2.26 together characterize "observable" MDP on the belief state space. Russel et al. (2002) also proves that an optimal policy $\pi^*(s)$ for "observable" MDP is also an optimal policy for the corresponding POMDP. Principally, a POMDP problem in physical state space can be scaled down to solving a MDP problem in belief state space.

2.6.2 DDPG - Deep Deterministic Policy Gradient

Silver et al. (2014) [65] introduced the Deterministic Policy Gradient (DPG) algorithm and which was extended to Deep Deterministic Policy Gradient (DDPG) algorithm by Lillicrap et al (2015) [43]. In Sivler et al. (2014) it was argued how DPG could be estimated more efficiently than the stochastic policy gradient. Since the latter integrates over both action and state spaces while DPG only accounts for the action space. The researchers then employed an off-policy algorithm which learns the deterministic desired policy from an exploratory policy behaviour. Subsequently, compared to the stochastic policy gradients, Silver et al. (2014) were able to show remarkable behaviour in both high- and low dimensional tasks with DPG.

In 2015 Lillicrap et al. extended the DPG algorithm utilizing the benefits from DQN algorithm introduced by Mnih et al. (2013), and batch normalization by Ioffe et al. (2015). Lillicrap et al. (2015) put forward DDPG, which is an off-policy, model-free actor-critic algorithm. DDPG is desgined to be applicable in continuous state-action spaces while using deep neural networks as function approximators. Specifically, in order to render the deep neural networks stable and robust, the two benefits from DQN that remedies this are experience replay and use of fixed target networks. In addition, it utilizes the batch normalization as further stabilizing mechanism. These benefits were previously discussed in section 2.4.2.

In relation to underwater vehicle (UV) control, Wu et al. (2017) [78] applied a DPG algorithm where a state-feedback controller for an AUV's depth was learned. Specifically, the applied RL algorithm applied was neural network DPG (NNDPG), which trained on sample trajectories of the AUV. The NNDPG is almost equivalent to the DDPG algorithm since both uses neural networks as function approximators, specifically, four hidden-layered deep ANN. NNDPG utilizes a modified experience replay and batch normalization. The paper compares the performance of the NNDPG algorithm to those of linear quadratic integral (LQI) controller and Nonlinear Model Predictive Control (NMPC). The LQI controller uses a linearized model of the AUV while the NMPC is derived from the exact nonlinear AUV dynamics. The experiment resulted in the NNDPG performance closely challenging the NMPC method, while exceeding the LQI method. However, it should be known that the paper performed all experiments in a simulator, hence, the results from the simulations may differ from those of real-world applications. Andrychowiz et al. (2017) [3] evaluated an algorithm which combined Hindsight Experienced Replay with DDPG on a 7 DOF manipulator on a sparse reward environment. The

training was conducted on a simulator. However, when the model was transferred to a real-world manipulator, the resulting performance proved to be satisfactory, which verified the trained policy. As a result, the manipulator was able to complete the task without fine tuning, and succeeded the performance of the original DDPG algorithm.

2.6.3 Covariance Matrix Adaption - Evolutionary Strategy

Within the model-free policy search approach, Covariance Matrix Adaption - Evolutionary Strategy (CMA-ES) is a unique policy update method which allows for use in continuous environment. It is based on the stochastic optimization approach, which is episode-based algorithm that model *upper level* policy $\pi_\omega(\theta)$. This creates samples in the policy parameter space $\boldsymbol{\theta}$ which is then evaluated on the agent's performance. The term "upper level policy", coined by Deisenroth et al. (2013) [21], differs from the *lower level* policy by $\pi_\theta(s)$ by being typically modeled as a Gaussian distribution, while the latter being typically of deterministic form. The parameter ω represents policy parameter θ distribution, which is beneficial for the ability of exploring policy parameters space as discussed in the report. The derived stochastic optimization for learning for upper-level policies becomes as follows.

$$J_\omega = \int_\theta \pi_\omega(\theta) \int_\tau p(\tau|\theta) R(\tau) d\tau d\omega = \int_\theta \pi_\omega(\theta) R(\theta) d\theta \tag{2.27}$$

Deisenroth et al. (2013, 2014-updated) asserted that the CMA-ES is considered to be the state of the art policy search method for stochastic optimization as stated in Hansen et al. (2003) [28]. The works reported in Heidenreich et al. (2009) [29] confirms the feasibility of CMA-ES method as a robotic controller. This was supported by a cart-pole system with the assigned task of balancing the pole yielded satisfying results. CMA-ES method maintains the Gaussian distribution $\pi_\omega(\theta)$ over θ while utilizing a data set D_{ep} for policy updates and weight d_i for each samples. The weights d_i are obtained through non-provable approaches, that is empirical approaches known as *heuristics* in algorithm-terminology. CMA-ES estimates the weights d_i by first sorting the policy parameters samples θ_i by returning R_i, and then compute the weights of n best samples by the following equation.

$$d_i = log(1 + n) - log(i) \tag{2.28}$$

Consequently, the new mean μ of the policy distribution $\pi(\theta)$ is found by weighing the average of the data points from data set D_i. Deisenroth et al. (2013) argues that the main advantage with this approach is that the covariance matrix is only

dependant on the current sample θ_i. Thereby, only depended on few samples. Nevertheless, while being simple to utilize, and being of stochastic form, CMA-ES has a disadvantage of failing to generalize the upper level of policy $\pi_\omega(\theta)$ in various contexts. Additionally, if the sample return R_i proves to be noisy, then the sample trajectory needs to be evaluated.

Yet, CMA-ES proves its potential as a viable control system for robotic systems, which may be of interest for agents like ROV/AUV's and other vehicular robots. Ahmahdzadeh et al. (2013) [2] proposed CMA-ES algorithm as a part of AUV control, designed for underwater valve-turning operations. The authors highlighted that the relative motion between a manipulator-equipped AUV and the valve is a challenging task which required a high-precision controller. The input parameter to CMA-ES algorithm is dependent on relative motion between the AUV and the valve, so that the a suitable tuning parameter for the RL AUV controller can be found. At first glance, this may seem like a time demanding task for the AUV to learn the underwater robot dynamics. However, the principal knowledge of the relative movements between the manipulator end effector and the valve is imported through expert demonstrations, on an imitation algorithm. Which is also based in Dynamic Movement Primitive. The latter is, as described by Ijspeert et al. (2003) [32] and Schaal et al. (2005) [60], a common time-dependent policy representation employed extensively in robotic systems. As a result, much of the underwater dynamics knowledge is obtained beforehand, thus, reducing learning time significantly. Ahmadzadeh et al. (2013) [2] further added that CMA-ES was the fastest algorithm in terms of computation time. Additionally, it required a small number of initial parameter settings. Consequently, the algorithm yielded feasible results in tuning the Reactive Fuzzy Decision Maker (RFDM), that is the AUV-manipulator-valve dynamic system the RL controller is based upon.

This development has proved to be promising for further developments in designing a CMA-ES based ROV RL controller. Specifically, since the CMA-ES approach has low computation time, it may more smoothly process the sheer amount of data stream between the top-side PC and the ROV.

2.6.4 Model Predictive Control with Guided Policy Search

In Zhang et al. (2016) [83] a survey was done on Model Predictive Control (MPC) in combination with guided policy as a control system on a quadcopter, which yielded positive results. Naturally, underwater vehicles operates under similar dy-

namic conditions, compared to aerial vehicles in terms of relevant DOFs and the unstructured complex environment. Hence, this approach should be considered for autonomous ROV/AUV control.

Mayne et al. (2005) [46] argued that a standalone MPC proved to be both reliable and effective control system for robotic systems. This statement was proved by Todorov et al. (2012) [74] when the paper experiments showed the MPC's ability to work with complex objectives. Mayne et al. (2005) also argued that MPCs posses a simplistic design and has the ability to control model errors. However, a major obstacle with MPSc applied on complex robotic systems is that estimation of the state is required. For example using Kalman Filter as state estimator. This is a challenging task for complex environments where ROVs and AUVs operates in. RL itself should render state estimation negligible by learning a policy that maps sensor readings to actions. However, the MPC approach is done by trial and error, and employing a partially trained policy on an unstable, fragile robotic system such as AUVs, ROVs and UAVs can lead to hazardous situations and costly damages.

However, Zhang et al. (2016) countermeasured these mentioned issues regarding standalone MPCs by introducing off-policy guided policy search. Specifically, is the high computation power remedied, along with hazards related to RL training process. The authors showed that the resulting computational cost was only a fraction of a standalone MPC approach. Additionally, the guided policy feature transforms the algorithm from from RL method to SL approach. Consequently, the policy training process is administered by a supervisor, i.e. a human operator or expert, by initially steering the UAV to avoid dangers.

While guided policy search offer such improvements to standalone MPCs, Zhang et al. (2016) asserts that this method still assumes a previously known model dynamics during test flights. Otherwise, the results would most likely be a failure as both known and learned model would be inaccurate. Hence, when the test flight is due, a DNN is used with guided policy since the former represents a wide-ranged complex behaviours. This approach provides two great advantages. The first being that DNN can restrict its observations from sensors to only those equipped on the UAV, for example Inertial Measurement Unit (IMU) inputs and camera vision feed. Thus, since the policy is represented by neural network, it can learn policies directly from, e.g. sensor cameras. The second advantage this approach introduces is heavily reducing computational cost compared to standalone MPC. Specifically, this applies to faster processing sensor observation data. This results in a robotic system with a satisfactory performance based on neural network, which is simply

trained with supervised learning.

2.6.5 Asynchronous Off-Policy Updates - A3C

Gu et al. (2016) [27] presented in their paper the benefits of employing Asynchronous Off-Policy Updates (also known as A3C) on robotic manipulation systems. In this sense, state of the art deep RL algorithms that are based on deep Q-learning functions, were scaled to perform complex 3D manipulation tasks. Additionally, the authors demonstrated that the A3C approach efficiently learns policies from DNNs, so that the simulation based training phase is negligible and directly training the robot in real-world environment is possible. The report also proves reduction in learning time as multiple robots are training simultaneously by parallelizing the algorithm across these robots. Subsequently, policy updates on multiple training robots takes place asynchronously, hence the name. This report confirms the efficient learning from this approach as a variety of 3D manipulation tasks was learned without any prior expert demonstrations.

2.6.6 CACLA - Continuous Actor Critic Learning Automation

Hasselt and Wiering (2007) [76] presented both Actor-Critic Learning Automation (ACLA), and its extension CACLA (Continuous Actor-Critic Learning Automation). The former algorithm bases itself on tabular representations, where one is for storing values and the other is for storing probabilities for performing each action in all states. ACLA assumes that the states and actions are in discrete form, and the values can be updated using for example TD-learning update. Given a state s_t, if the resulting action performance for a value is a positive change ($\delta_t > 0$), then the corresponding action a_t is considered to potentially lead to a higher future discounted reward. Thus, ending up with a improved policy $\pi(s_t, a_t)$. In other words, the probability of selecting a_t is increased and probability of taking other actions is reduced accordingly. ACLA approach differs from conventional actor-critic approach by only using the sign of the TD-error to update the actor, while the latter approach uses the exact value of TD-error for the value update process.

ACLA value update has its advantages as it is easily extendable to continuous domain, rendering the method to CACLA. The extension into continuous state space can be done by replacing the tabular representation with function approximator. In which case takes state s_t as an input and outputs an action a_t. In Hasselt and Wier-

ing (2007) the action output of the function approximator as time t is denoted as $Ac_t(s_t)$, and its corresponding parameter vector is θ^{Ac}. Hence, the report describes the parameter update as following.

$$\text{IF} \qquad \delta_t > 0 \ : \qquad \theta^{Ac}_{i,t+1} = \theta^{Ac}_{i,t} + \alpha(a_t - Ac_t(s_t))\frac{\partial Ac_t(s_t)}{\partial \theta^{Ac}_{i,t}}$$

As stated in the logic above, given that the value has increased, that is positive TD-error, this results in that the actor learns to perform more similar actions to a_t in state s_t. Otherwise, if the value is not increased ($\delta_t < 0$) the actor will not change and the value is not further updated. This algorithmic structure informs the agent that the latest action is more sufficient if there is no positive change in value. Subsequently, avoiding faulty value updates towards some random action that leads to a worsened present approximation of the optimal action.

It is also possible to extend the vector parameter update so that the effects of actions are stressed to further improve the value. This can be achieved by storing an average variance of TD-error and use this variance to determine whether the action was particularly good or not. The number of updates towards an action is related to the number of standard deviations between the target value and the old value. This approach results in more updates than the standard CACLA, which updates more than once, and is referred to as CACLA+Var.

2.6.7 Trust Region Policy Optimization - TRPO

Schulman et al. (2015) [63] put forward TRPO algorithm which utilizes the policy gradient methodology that adopts the actor-critic architecture. In line with DDPG, TRPO also allows for agents operating in continuous action-state space while maintaining a decent performance. However, a major difference from the DDPG method is that TRPO uses a different approach to update its policy parameters of the actor. TRPO introduces the advantage element $\eta(\tilde{\pi})$, which is considered as the expected return of the new policy $\tilde{\pi}$ with respect to the advantage over old policy π. In Schulman et al. (2015), the advantage is formally presented in the following equation.

$$\eta(\tilde{\pi}) = \eta(\pi) + E_{s_0,a_0,..\tilde{\pi}}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)] \tag{2.29}$$

Intuitively, the advantage module can be viewed as measurement of how good the new policy is compared to the average performance of the old policy. As shown in

Schulman et al. (2015) the advantage η of the new policy can be rewritten into the following equation.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s,a) \tag{2.30}$$

Where ρ is discounted visitation frequencies.

$$\rho_{\pi}(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + ... \tag{2.31}$$

Nonetheless, equation 2.30 has proven to be hard to optimize since ρ is greatly dependant on the new policy $\tilde{\pi}$. Hence, to counter this issue, the paper introduced an approximation to $\eta(\tilde{\pi})$, which is denoted as $L_{\pi}(\tilde{\pi})$.

$$L_{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \pi(a|s) A_{\pi}(s,a) \tag{2.32}$$

The difference between equation 2.30 and 2.32 is that $\rho_{\tilde{\pi}}$ is replaced by ρ_{π}. This approach anticipates that state visitation for the old and new policies is not too different. This equation is then combined with the favourable policy update method shown below.

$$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha\pi'(a|s) \tag{2.33}$$

Where π_{old} is the current policy used, and $\pi' = argmax_{\pi}, L_{\pi_{old}}(\pi')$ is the argument max of the policy that maximizes $L_{\pi_{old}}$. Hence, the following theorem is obtained.

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{KL}^{max}(\pi, \tilde{\pi}) \tag{2.34}$$

C is the penalty coefficient, whereas D_{KL}^{max} correspond to the maximum information loss (namely KL divergence) of the two parameters for each of the states. Simply put, the last term in general represents how different the parameters π and π' are. Schulman et al (2015) proved in their paper that equation 2.34 can be viewed as the expected long term reward η is monotonically improving, given that the right-hand-side of the inequality is maximized. All in all, the problem is boiled down to the objective function of maximizing the right-hand-side of the equation in terms of policy parameter θ.

$$maximize_{\theta}[L_{\theta_{old}}(\theta) - CD_{KL}^{max}(\theta_{old}, \theta)] \tag{2.35}$$

In practice, when introducing a penalty coefficient in the objective function, the step size becomes very small, which will lead to long training time. However, constraining the KL divergence allows for a larger step while guaranteeing good performance. The KL divergence constraint is imposed on every state s in the state space by controlling that the maximum of which should be smaller than, or equal to, a small number δ.

$$D_{KL}^{max}(\theta_{old}, \theta) \leq \delta \tag{2.36}$$

Since, TRPO assumes a continuous state, there is an infinite large amount of states, which in turn would have yielded infinite large amount of constraints. To remedy this, the paper proposed a solution based on heuristic approximation with the expected KL divergence, instead of finding the maximum KL divergence. Here, the term *heuristics* is understood as a technique for finding an approximate solution where classical methods fails to find the exact solution. In often cases, heuristic approach trades completeness, accuracy and precision for speed, which renders this method as a shortcut.

Consequently, the paper shows that the objective functions is rendered to the following resulting problem.

$$\begin{aligned} &maximize_{\theta} E_{s\pi_{\theta_{old}}, a\pi_{\theta_{old}}}[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\theta_{old}}(s,a)] \\ &subject\ to\ E_{s\pi_{\theta_{old}}}[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_{\theta}(\cdot|s))] \leq \delta \end{aligned} \tag{2.37}$$

This objective function is also named "surrogate" function since it consist of a probability ratio between the current policy and the next policy. TRPO prosperously addresses the issue introduced by DDPG, where the performance does not improve monotonically. While the change in policy is fairly small, the approximation is not much different from the original true objective function. Hence, a new policy parameter θ is chosen which maximizes the expectation subject to the KL divergence constraint. Following, a lower bound of the expected long-term reward η is assured, which implies that the step size in TRPO is of little significance.

2.6.8 Proximal Policy Optimization - PPO, OpenAI version

Although TRPO proves to be very prominent in its performance, the computational power and implementation of this algorithms is immensely complicated. This problem has its roots from the constraint imposed on the surrogate objective function (i.e. equation 2.37), which is the KL divergence between π_{old} and π_{new}.

To approximate the KL term, a second-order derivative of KL term is used, which is the Fisher Information Matrix. This process requires large amount computation capacity as it requires calculating several second-order matrices. In the TRPO paper by Schulman et al. (2015) [63], this problem is addressed by introducing Conjugate Gradient (CG) algorithm to solve the constrained optimization problem in order to bypass the explicit computation of Fisher Information Matrix. However, the introduction of CG renders the TRPO implementation severely complicated.

However, Schulman et al (2017) [64] later introduced PPO that avoids the computation step due to the constrained optimization problem. This paper put forward a clipped surrogate objective function. The main intention of TRPO's constraint is to forbid the policy to change too much. Hence, instead of adding a constraint like TRPO, PPO rather slightly adjust TRPO's objective function with a penalty for policy updates that are too large. Subsequently, the new surrogate objective function becomes as following.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.38}$$

Where $r_t(\theta)$ is the probability ratio between the new and the old policy. In equation 2.38 is it shown how $r_t(\theta)$ is clipped between $[1-\epsilon, 1+\epsilon]$. This implies that if $r_t(\theta)$ prompts the objective function to increase to a predefined limit, its efficiency will, that is be clipped. This algorithm design renders to main cases, that is when the advantage $\hat{A}_t$ is larger than 0, and when $\hat{A}_t$ is smaller than 0.

When $\hat{A}_t$ is greater than 0, it means that the action taken is better than the average of all the actions that state. Hence, is the $r_t(\theta)$ for that action increased in order to encourage such action. This way any promising action has a higher chance to be adopted by the agent. This is due to since the denominator of $r_t(\theta)$ is constant, i.e. $\pi_{\theta_{old}}(s, a)$, only the $\pi_{\theta_{new}}(s, a)$ is increased. In other words, increase the chance for taking that action a in state s while the clip functions limits $r_t(\theta)$ to grow larger than $1+\epsilon$. On the other hand, should $\hat{A}_t$ shrink smaller than 0, then that action is discouraged by the algorithm. Subsequently, $r_t(\theta)$ is decreased, and in contrast to the other case, $r_t(\theta)$ will not decrease to a smaller value than $1-\epsilon$, due to the clip function. Fundamentally, the clip function restricts the range in which the new policy can vary from the old policy, hence, discouraging the probability ratio $r_t(\theta)$ to step outside the defined interval.

Schulman et al (2017) argues in their paper that when comparing the objective function used in TRPO (i.e. equation 2.37) to the objective function used in PPO (equation 2.38), the latter is in fact a lower bound of the former one. This means

that a "pessimistic" estimate of the policy's performance is weighed. It also removes the heavy computational nature of KL divergence constraint, thus, rendering the optimization process for a PPO objective function much easier than that of a TRPO's. In addition, PPO has empirically proven to outperform TRPO.

2.7 Challenges related to Reinforcement Learning in robotic systems

Although utilizing RL in robotic systems increases the autonomy in the operator's perspective, it is, however, far from a smooth process in order to obtain a reliable RL based robotic control system. Real-world robots operates in the continuous domain, and thus, it is crucial to carefully design the RL system to obtain the optimal behaviour.

This section presents the most renowned challenges in applied RL systems in general basis, and related to real-world robotic systems. In addition, recent developments that are dedicated to remedy these issues are also presented.

2.7.1 Exploration-Exploitation dilemma

In contrast to SL, in RL, the agent must first discover its environment. This is fundamental for the agent to obtain experience about the reward and general system behaviour. Hence, it is required for the agent to take actions that are rarely used, or completely unused, which has high statistical uncertainty. Therefore, it comes down to whether to play it safe and embrace the current working policy that guarantees adequate rewards, or to step out of the comfort zone and try new strategies which may yield exceeding rewards. This is a dilemma that nearly all RL systems must face, and is known as *exploration-exploitation dilemma*.

The term "exploitation" term means that the agent only chooses the optimal action known to the current policy. However, a pure exploitation behavior renders the agent to probe within a limited region of search space. This leads to only optimizing its behaviour within the current area, while neglecting a possibly more optimal solution in other state spaces. A policy gradient policy search method is viewed as an exploitative algorithm, since it only utilizes its current policy and its surrounding area only to gradually improve performance. According to Kober et al. (2013) [34], this kind of method only results in local optima being obtained, while possibly

leaving out a more rewarding optima.

As for the term "exploration", which refers to nature of selecting sub-optimal actions in the hope of acquiring more rewarding policies than the currently pursued. Yet, this may result in the agent probing in a much larger state space area in order to search for other more optimal solutions. As a result, methods that are purely build on explorative behavior requires vast amount of training time during the learning phase before converging towards a solution.

Consequently, it is essential for algorithms to balance the frequency of both exploiting and exploring actions so that the global optima is obtained within a reasonable amount of time.

One commonly used algorithm that handles the exploration-exploitation trade-off is the $\epsilon - greedy$ strategy. This algorithm aims at choosing a greedy action which maximizes $Q(s, a)$ with a probability of $(1 - \epsilon)$. Where the parameter ϵ has the range $0 < \epsilon < 1$, and defines a randomly selected action with probability ϵ. Thus, by decreasing ϵ, the level of explorative behavior is correspondingly decreased, and vice versa for exploitative behavior. Poole et al. (2010) unveiled a downside with the $\epsilon - greedy$ strategy, which is that it treats all actions equally, aside from the seemingly optimal solution. For instance, when ending up with two apparent feasible actions, while the rest of the actions are deemed less rewarding, it may be more beneficial to explore the two promising actions rather than the rest.

As a countermeasure to this problem, an alternative action-selection approach can be employed, such as Gibbs Softmax action-selection. This approach chooses action a with respect to the value of $Q(s, a)$. A favourable action-selection distribution utilized in Softmax method is *Gibbs-* or *Boltzmann-distribution*. This renders the probability of selecting action a in state s proportional to $e^{Q(s,a)\tau}$. Where τ is a parameter that determines the randomness of choosing a value. If τ is high, the actions are selected on equal ground, and if τ is converging towards zero, the supposed optimal action is repeatedly chosen.

2.7.2 Curse of Dimensionality

Bellman (1957) formulated the term *curse of dimensionality* after studying optimal control in discrete high-dimensional state spaces. It was observed how the amount of data and computational effort increased exponentially as the number of dimensions grew. For instance a discretized state space, where each dimensions dimension is

divided into 10 levels, then there are 10 states for one-dimensional state spaces, while as for 3D state spaces that renders $10^3 = 1000$ different states. Hence, this can in general be formalized as such, for n-D state spaces there is 10^n different states. Even in discrete cases, the state space may quickly grow large and become impractical to cope with.

This is a challenge that often limits the capabilities in robotic systems operating in continuous state spaces. Yamaguchi et al. (1997) [80] highlights how the curse of dimensionality is prominent in anthropomorphic (i.e. human-like) robotic systems with antagonistic driven joints that resembles human tendon-muscle system.

Traditionally, in RL based robotic operations, the engineer normally decomposes the task into hierarchies. This is to shift complexity towards a lower layer of physical functionality. For instance, RL approaches often perceive the world as grid-based, which is in ML terminology known as *grid-world*, where states and actions are expressed in discrete form. For instance, if using this grid-world perception on a navigational task on robot, a potential command would be "move to the right hand cell". In such cases a low-level controller is employed that handles the movement dynamics, bein0g acceleration and stopping. However, Schaal et al. (2002) and Peters et al. (2010b) argued that the grid-world representation renders the dynamic capabilities for the robots considerably limited, unveiling a major downside in this type of state-dimensionality reduction.

A common way to cope with curse of dimensionality is by using ML tools such as appropriate function approximators, as presented in section 2.4.2. It can also be solved by employing macro-actions, which is comprehensively introduced in Barto et al. (2003) [5].

2.7.3 Curse of Real-World Samples

Other than issues in algorithmic-level related to robotic control, there are also real-life factors needed to consider during robotic systems design. For example, a practical case of hardware wear and tear is a major threat, and requires constant care to avoid such physical damage. Should any damage occur, the resulting maintenance labor would have been both quite costly and cause unnecessary stagnation in development. Subsequently, safe exploration is naturally a key factor to consider in order to avoid unintentionally hazards for the robot. Yet, self-exploration process in the real-world is a central part of the learning process, from which a large amount of dangers are expected. A approach to remedy this hazardous phase is

by switching between the underlying controllers in the robot, hence, preserving an abundance of safety and performance guarantees. This approach was introduced by Perkins et al. (2002) [54], which in their work based their RL method on Lyapunov functions.

A relatable real-world challenge is the use of computer visualization technology. In this thesis, for example, a ROV using camera for object detection may suffer from faulty image processing such as non-ideal light conditions, particle reflections or image distortion. Hence, should the visual performance be negatively affected, it will then negatively affect the robot's sate perception, resulting in misguided performance. As argued by Kober et al. (2013) [34], such cases requires human supervision in order to avoid dangerous situations for the robot itself, and in some cases the environment itself, specifically for self-driving cars systems. Subsequently, real-world challenges can arise in terms of time consuming training phases, human workload, and result in growing expenses. Thus, Kober et al. (2013) argue that in many robotic learning situations, limiting the real-world interaction time is often preferred instead of limiting the computational memory and complexity.

Time-discretization is also a case of real-world problem since the RL algorithms are normally implemented on a computer, which is a digital platform. Event though the physical system is continuous-time domain, time-discretization is still inevitable. This induces inaccuracies, such as time-delays during data processing from sensors to initiating actions, or physical delays stemming from joint motion and actuation. As a result, action executed may not have immediate physical effect on the RL robotic system, but only observable first some time-steps later. Time-delays due to time-discretization can be remedied by increasing the duration of time steps, however, at the cost of more unreliable robotic controller.

2.7.4 Curse of Under-Modeling and Model Uncertainty

In RL it is a common practise to pre-learn the robotic system, specifically the interaction with the environment, as it reduces required learning time. However, some physical environments prove hard to model with decent accuracy, e.g. an ROV operating in underwater environment. Physical states and environments of an underwater ROV robotic system requires careful design and verification before use. Otherwise, any small parameter deviations may accumulate to significant erroneous real-life performance.

Importing policies from simulator trained model into real-world robotic systems can

be done smoothly if the task, or the physical environment, is self-stabilizing. While in the case of ROVs operating in underwater environment is not self-stabilizing as it constantly relies on active control to maintain a safe state. Naturally, active control is needed to avoid hazardous situations like collisions or drifting. However, should the task be stable by nature, it is then more suitable to assume that the model obtained from the simulations would operate feasibly in the real-world environment. However, when it comes to critical physical parameters, such in the case of ROVs, frictional and 2^{nd} order hydrodynamical forces are the key parameters to be learned. Typically, they are hard to learn precisely through simulations, and are learned to a certain extend, in order to be fully mastered later when imported to real-world environment and continue the remaining training phase.

2.7.5 Curse of Goal Specification

The reward function is the only element that provides feedback to the RL performance in terms of reaching the optimal behaviour, and is therefore a key element in RL. Designing reward functions may prove to be troublesome, especially for high-dimensional state spaces. For instance, if the reward function is binary, the same return will be collected. Consequently, the agent will not be able to distinguish if its policy is improving and/or converging towards the optima. Such challenge is classified as *Curse of Goal Specification*, which is thoroughly represented in Kober et al. (2013) [34].

Binary reward functions can be convenient in software-based low-dimensional discrete board games, in which the reward is provided only after winning a match. While as for robotic control systems, a more intricate reward function is required in order to receive corrects reward signals for each action-state pairs. Laud et al. (2004) [39] introduced *reward shaping*, which is initiating rewards midway for agents, operating in high-dimensional state spaces, along the way to guide the learning process to a solution.

Reward shaping shortens the problem horizon by introducing midway rewards. However, this may result in trade-off in several performance aspects. For instance, a ROV moving to a desired point at highest possible speed would have yielded high reward, but is likely that the actuators would experience wear and tear. As such, by behaving in nonsensical ways, RL algorithms may very well exploit the reward functions so that high rewards are obtained, while still achieve solution. Another example is robotic ball-paddling RL system, where the distance between ball and

the highest defined desired is a parameter in the reward function. Thus, the manipulator may find the local optimal solution by easily moving the racket, with the ball on it, to the desired point and hold it there. Reward shaping allows for an optimal, or desired, behaviour for high-dimensional systems instead of coping with ultimate success or failure scenarios. This approach renders the RL problem manageable.

Curs of goal specification can also be remedied by employing *Inverse Reinforcement Learning* methods, which is presented in detail by Ng et al. (2000) [53]. This method, also known as "inverse optimal control", defines the reward function for the agent by being initialized with a policy. This policy can be, for instance, be defined by human-supervised demonstrations. The resulting constructed reward function may not automatically be optimal, nevertheless, it may be considerably better than that of a typically manually designed reward function. Ziebart et al. (2008) [84] describes how, over time, inverse learning has been elaborated with various applied techniques. Inverse learning has proven by Bagnell et al. (2007) [4] to be efficient tool for robotic systems such as manipulators, outdoor robotic manipulation and robotic legged locomotion.

3 Modeling ROV dynamics

As explained in subsection 2.5.2, in model-free ML cases, the environment dynamics are not taken into consideration because it is expected to be learned when the agent interacts with the environment. While, in this case, since the training phase of the ROV will take place in a simulator, the environment dynamics are needed to simulate the real-world underwater effects adequately. Specifically, the hydrodynamic coefficients and equations of motions play a vital role in rendering a reliable simulator.

This section aims to present the mathematical model of the ROV that describes its dynamics and used in the simulation training in this thesis. The theory is mainly based on Fossen (2011) [25], and thereby, the mathematical notation is also adopted from the author.

3.1 Mathematical model of a ROV

Fossen (2011) divides mathematical models of aerial- and underwater vehicles in two main parts, that is kinematics and kinetics. The former describes the geometrical approach of motion while the latter comprises the forces and moments acting on the ROV body generate motion. The kinematic aspect does not consider the cause of the motion, while kinetic aspect specifically explains the motion.

In this section, the ROV's kinematic model is described, which encompasses the equations of motions including its physical parameters. The relations between the kinematic and kinetics are explained, and how those together induces ROV motions. In addition the concept of reference frames are fundamental for understanding the current state of the ROV in terms of positioning, velocity and orientation.

3.1.1 ROV kinematics

In this thesis, the ROV kinematics are introduced to define the motions in different reference frames. Here, the position, orientation and motion are presented by vectors and generalized coordinates.

Table 1 below shows the notations for 6 DOFs representation of a vessel.

Table 1: Overview of the 6 DOFs used for marine vessels and aircrafts.

Kinematic direction	**Positions and Euler angles**	**Linear and angular velocities**	**Forces and moments**
x-direction (surge)	x	u	X
y-direction (sway)	y	v	Y
z-direction (heave)	z	w	Z
Rotation about x-axis (roll)	ϕ	p	K
Rotation about y-axis (pitch)	θ	q	N
Rotation about z-axis (yaw)	ψ	r	M

Reference frames

Reference frames are utilized to represent the position, orientation and motion of the vessel. The two most used reference frames are the North-East-Down frame (NED) and the body-fixed frame . Typically, those frames are used in conjunction to determine the relative global motion of a marine vessel.

NED frame, denoted with a "n"-subscript, is defined with the $[x_n, y_n, z_n]$ axes. The frame's origin is fixed on the Earth's surface while the frame is tangential to the spherical nature of the Earth's surface. x_n aims towards the true north, y_n towards the true East, while z_n points downwards normal to the frame plane and to the Earth's core origin. According to Fossen (2011), a vehicle, within a small area, which travels with a relatively low speed renders the longitude and latitude to be assumed constant. This means that the NED frame can be deemed inertial (i.e. non-accelerating) and the Earth's angular rates are then negligible such that Newton's laws are valid.

BODY frame, denoted with a "b"-subscript, is defined with the $[x_n, y_n, z_n]$ axes. The frame's origin is fixed on an arbitrary point on the vessel, however, it is normally placed on the vessel's centerline or at intersections of symmetry planes. The body-fixed origin is denoted as Center of Origin (CO). BODY frame is used along with an inertial frame (e.g. NED frame) to determine its position and orientation relative to the inertial frame, while the linear and angular velocities are purely defined in the BODY frame.

Figure 7 illustrates how BODY and NED frames are fixed at their respective placed in the environment.

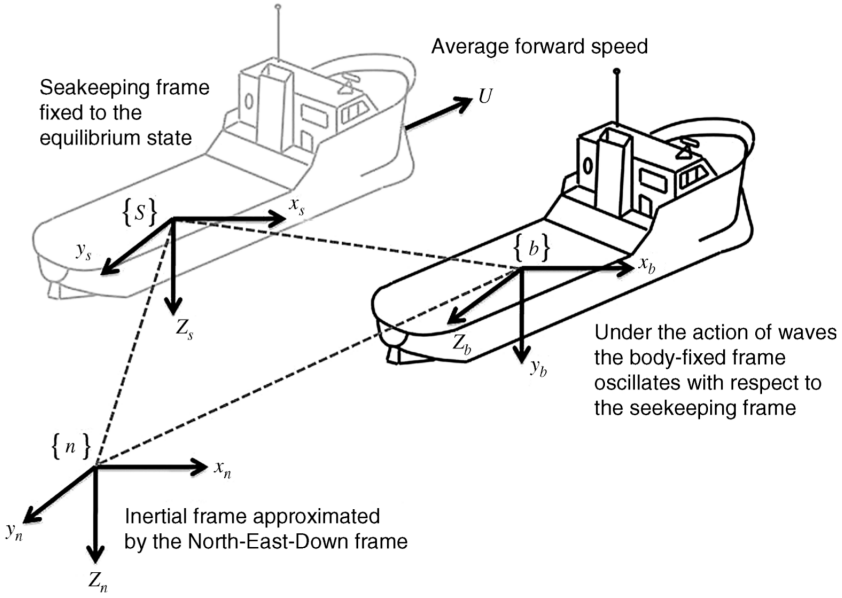


Figure 7: Overview of relation between BODY frame and NED frame. Fossen (2011) [25].

The reference frame used in the test basin at MC-laboratory, where the practical experiments are tested, is defined by the Qualisys Motion Tracking system (QMT) . QTM will be further detailed in section 6.2. While the BlueROV2 has its predefined conventional BODY frame, QMT system is an underwater positioning system which is used as NED frame during experiments. The origin in the basin is situated at the middle in the bottom of the tank.

3.1.2 ROV kinetics

The mathematical model of the ROV is based on Newtons 2$^{\text{nd}}$ law. This approach assumes a perfect model when all the forces acting upon the ROV are accounted for. However, obtaining an accurate detail on each of the forces is a challenging task, and is consequently remedied by making some simplifications. One of which is that the ROV is considered to be a rigid body, thus no bending and geometrical deformations are assumed. By this reasoning, the front part, can not rotate or move faster or slower than its counter part, and vice versa. The equations of motion implemented in the simulator are based on Fossen (2011, p. 110). The original equations of motions are shown in equation 3.1.

$$\boldsymbol{M\dot{\nu}_r + C(\nu_r)\nu_r + D(\nu_r)v_r + g(\eta) + g_0 = \tau_{thrust} + \tau_{wind} + \tau_{wave}} \quad (3.1)$$

While in the test basin, both wave and wind disturbances are negligible as they are non-existent there. Since the ROV will be operating in 2D plane in this thesis, only the xy-coordinates are of interest and the parameters concerning any other DOFs that are not surge, sway and yaw are negligible in equation 3.1. As a result, the restoring forces $\boldsymbol{g(\eta)}$ and $\boldsymbol{g_0}$ are thereby eliminated as they only affect the DOFs of heave, roll and pitch, and have no effect in a planar environment. The subsequent equations of motion which are utilized in the simulation becomes as following.

$$\boldsymbol{M\dot{\nu}_r + C(\nu)_r\nu_r + D(\nu_r)\nu_r = \tau_{thrust}} \quad (3.2)$$

The **mass matrix** $\boldsymbol{M = M_{RB} + M_A}$ is the system inertia matrix included with the added mass matrix. $\boldsymbol{M_{RB}}$ is the rigid body mass matrix of the ROV in air, in addition to the moments and products of inertia. This matrix is highly dependant on the ROV's mass and is subject to change if equipment on the ROV is replaced or removed. While $\boldsymbol{M_A}$ is the added mass matrix that represents the resistance force which is proportional to the ROV's acceleration. Depending om the vessel's shape, the added mass matrix can be assumed to be constant.

The **Coriolis** and **centripetal matrix** $\boldsymbol{C = C_{RB} + C_A}$ includes the Coriolis effect into the ROV's equations of motion. They are fictional forces that accounts for the Earth's rotational nature. Myrhaug (2006) [50] highlights that the deflection of an object, as it moves in a rotating reference frame, causes the Coriolis effect. Hence, there is no real force affecting the ROV, however, since the reference frame is rotating relative to the moving object, it suggests that the object deviates from its path. As Newtons 2$^{\text{nd}}$ law is only applicable in inertial reference frames, the Coriolis and centripetal forces must be included.

$\boldsymbol{C_{RB}(\nu)}$ is the Coriolis and centripetal matrix for the ROV's rigid body. This term considers the fictional force which causes the relative movement between the ROV and NED reference frame due to Earth's rotation. $\boldsymbol{C_{RB}(\nu)}$ is dependant on the ROV's added mass and is a function of the relative velocity to the NED reference frame.

$\boldsymbol{C_A(\nu_r)}$ is the Coriolis and centripetal matrix for the ROV's added mass. This term is also dependant on the ROV's added mass and its relative velocity to the NED frame's rotation while $\boldsymbol{C_A(\nu)}$ takes the current effects into consideration.

$\boldsymbol{D(\nu_r)\nu_r = D_L\nu_r + D_Q(\nu_r)|\nu_r|}$ is the damping matrix where $\boldsymbol{D_L\nu_r}$ is the linear vi damping coefficient. While $D_Q(\nu_r)|\nu_r|$ correspond to the nonlinear viscous damping force in water stemming from the the ROV's velocity. The damping matrix is an estimated parameter found by model testing.

All of the matrices are listed in the appendix and defined in line with the formulations described in Fossen (2011). The exact hydrodynamic coefficient values are used from Stian Sandøy (2016) [57]. While those coefficients are specified for the BlueROV1, it is in this thesis with large certainty applied on the BlueROV2 as well since it is very comparable in design.

The thrust force $\boldsymbol{\tau}$ is the control input in this simulator, which directly assigns the velocity of the ROV. Reformulating equation 3.2, equation 3.3 is obtained, which correspond to the relation between the ROV acceleration $\boldsymbol{\dot{\nu}}$ and the thrust force input.

$$\boldsymbol{\dot{\nu} = M^{-1}[C(\nu_r)\nu_r + D(\nu_r)\nu_r + \tau_{thrust}]} \tag{3.3}$$

From the acceleration vector, the velocity in x- and y-direction is obtained respectively through time-step integration. Since the calculation procedures are taking place on a digital platform (i.e. computer), the equation 3.3 must presented in discrete form with respect to time-step t_k. The resulting equation of motion is formulated as following.

$$\boldsymbol{\dot{\nu}(t_k) = M^{-1}[C(\nu_r)(t_k)\nu_r(t_k) + D(\nu_r(t_k))\nu_r(t_k) + \tau_{thrust}(t_k)]} \tag{3.4}$$

The simulator estimates the ROV's velocity in the BODY reference frame from the acceleration attained by equation 3.4. The BODY velocity is then converted to NED reference frame, hence, the resulting position is obtained by time-step integration. The integration method is of Runge-Kutta approach and will be further detailed in subsection 4.1.2.

4 Implementation of RL algorithm and simulator

The RL algorithm utilized for training the BlueROV2 is PPO. This recent algorithm, which was discussed in subsection 2.6.8, has a large variety of publicly available implementations which was easily accessible online, and ready for modification before implementation in this thesis. As stated in subsection 2.6.8, while being state-of-the-art within RL, PPO is also impeccable for usage in continuous robotic systems such as vehicular locomotion. By this reasoning, the PPO algorithm is an ideal match for ROV control. This line of thought was also followed by Henderson et al. (2017) [31] where the authors used publicly available implementations in their work, proving such common practise.

In this thesis, the PPO algorithm was implemented in Python programming language with **Keras** as ML-tool using **Tensorflow** as backend. A simulator was developed in the same programming language to verify the performance of the RL algorithms before conducting experiments in the real-world test basin. Several experiments were performed with various reward functions as an effort to obtain the most promising tracking performance. Subsequently, in this chapter, the importance of reward functions is also highlighted.

This chapter aims to present an overview of the simulator composition and the structurization of the employed RL algorithm. The reward functions of interest are presented along with an explanation of the hyperparameters related to the PPO algorithm.

4.1 Simulator configuration

As explained in chapter 2, the simulation process is a key element to speed the learning process prior to applying it on the real-world test basin. When the model is trained to a desired level in the simulator, it is then exported to the BlueROV2 to be applied in the MC-lab test basin.

The simulation setup used in this thesis is a non-graphical Python software based on a **OpenAI** architecture [12]. The latter is a non-profit AI organization which promotes research and development in AI technology. OpenAI aspire to collaborate with other institutions and researchers by making its patents open to the public. Most notable founders are Elon Musk, which is the founder of Tesla and SpaceX, and Sam Altman which is a renowned entrepreneur. OpenAI provides various agents

operating in various environments so that researchers can access a large range of different testing ground. The environments stretches from basic discrete systems, such as an inverted pendulum, to complex anthropomorphic systems, such as a half cheetah. Many of the aforementioned articles in section 2.6 have been using OpenAI environments as testing ground for their works.

The simulator utilizes the implemented kinematic and kinetic ROV models as presented in chapter 3. The parameters used in the models were specified for BlueROV2 and can be found in appendix B. A designed 2D area for test basin was defined with the dimensions of 40 m and 6.45 m in x- and y-directions respectively. The aruco marker's position is used as reference coordinates, which is situated in the basin's origin at $(x, y) = (0, 0)$. As the aruco marker is fixed at the basin's origin, the simulator-ROV is tasked to travel to the aruco marker's location based on it's relative state feedback. Figure 8 shows the simulator environment, that is the test basin along with its dimensions. Positive x-direction points toward right, while positive y-direction is downwards in figure 8.

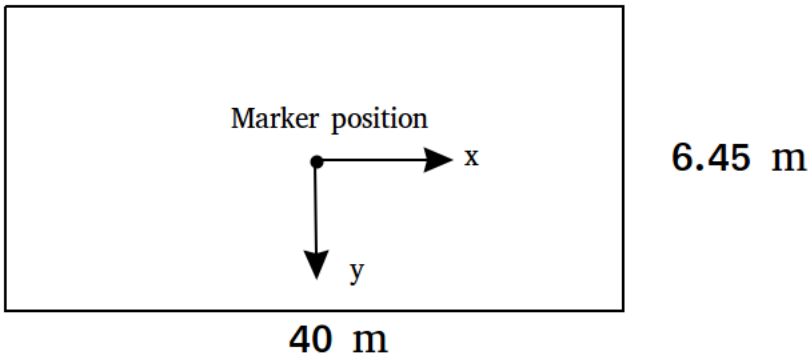


Figure 8: Test basin as defined in simulator.

At the very beginning of each simulation episode, initial ROV deployment position is determined randomly by the simulator. In such case, it is important to avoid defining a too large area for initial ROV states. This is because it would require a longer training time before a convergence in performance would appear. Hence, the curse of dimensionality becomes an issue since the ROV is prompted to probe a large area for solutions. Yet, a too small deployment area for the ROV would render the trained model to be an optimized at that specific small area, while being unskilled when deployed outside that area during real-life experiments. Evidently, determining the initializing area requires accounting for both training time during simulations and real-world performance.

In addition, it is also essential to mind the practical aspect of state initialization as the computer vision performance can only be reliable in a certain range of distance and orientation. This was found out to be maximum of 5 meters away from the aruco marker and the limit for relative yaw was found to be 90 degrees. For practical purposes, the real-life aruco marker should maintain a "safety-zone" of 1 meter radius. Subsequently, a suitable configuration for initial ROV deployment for the simulator would be $-5 \leq x \leq -1.5$ meters, and $-3 \leq y \leq 3$. The deployment area is visualized in figure 9 and its parameters are summarized in table 2.

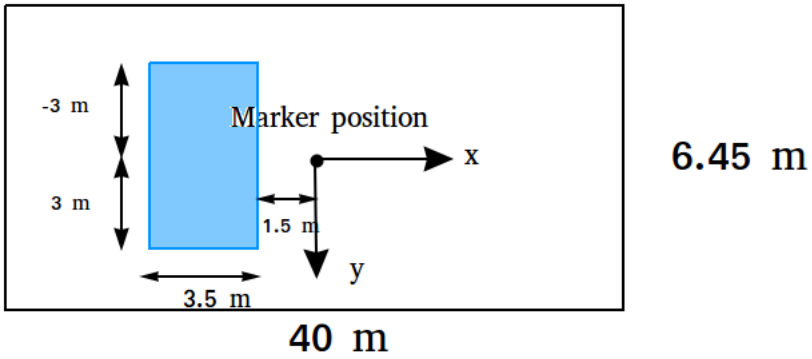


Figure 9: Area of random ROV deployment (blue) in the simulator.

Table 2: Configurations for initial random ROV deployment per episode.

Along x-axis	$-5 \leq x \leq -1.5$ m
Along y-axis	$-3 \leq y \leq 3$ m
Heading	$-20 \leq \psi \leq 20$ degrees

Since the simulator is based on OpenAI's architecture, it is thereby built in an object-oriented manner. This means that logical subgroups are implemented as classes and objects. Such design approach renders it simpler to structurize the programming of such a large application. This approach of programming paves way for easier further modification and software error handling.

Figure 10 is an effort to visualize a simulation time-step procedure. In this setup the agent receives observation and calculated reward from the simulator followed by an action initiation, and the process repeats itself for the next time-step.

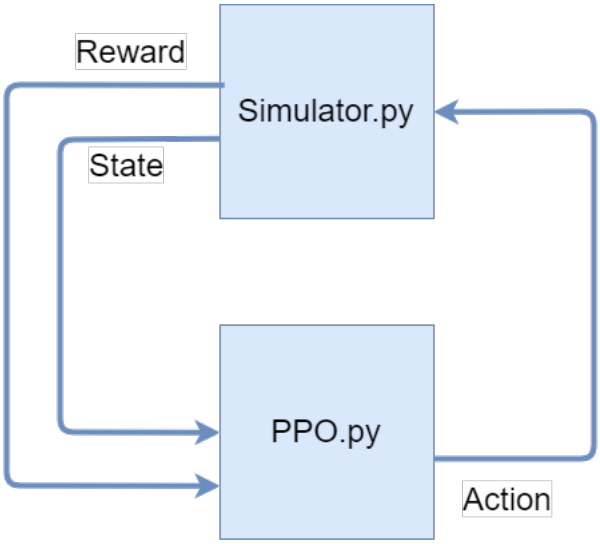


Figure 10: Simulator-Agent interaction.

4.1.1 ROV state space

While there exists a global NED frame and a BODY frame on the ROV, in this simulator however, a local frame is defined where a kinematic relation between the ROV and aruco marker is defined. The local frame is based on the aruco marker's relative yaw orientation and relative position in x- and y-directions to the ROV itself. As a results, the state feedback are relative pose of the aruco marker (i.e. x_r, y_r and ψ_r). This is illustrated in figure 11 while the relevant states, which are also denoted as "observations", are listed in the following table 3.

Table 3: Relevant ROV states.

States	
Relative position in x-direction	x_r
Relative position in y-direction	y_r
Velocity in x-direction	v_x
Velocity in y-direction	v_y
Relative orientation in yaw	ψ_r

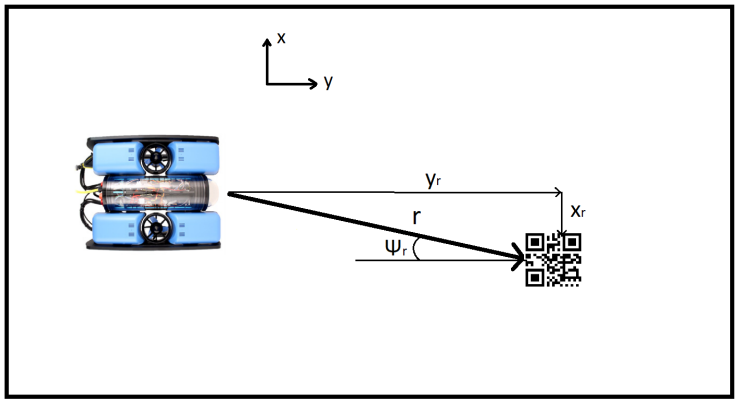


Figure 11: Overview of the local reference frame. Credit: BlueLink.

While the essential translational state feedback is in the form of relative distance in x- and y-directions, the simulator however uses the **radial distance** r when calculating the reward. The calculation is shown in the following formula.

$$r = \sqrt{x_r^2 + y_r^2} \tag{4.1}$$

The radial distance can be useful when applying constraints on the ROV. For example, in this thesis, an ideal area for the ROV to dwell is defined to be at an arbitrary radial distance between 1 to 2 m away from the aruco marker. This makes it easier to shape the ROV's tracking behavior in such a way that it will most likely not drift away from the aruco marker while avoid collision.

Use of such local frame relies on that the relative states are directly measurable, which is where the ROV's computer vision feature comes in to play during the physical experiments. The main advantage with using this local frame is that a lot of extra calculation steps are bypassed should an alternative NED-BODY based frame be used instead.

4.1.2 ROV action space

When it comes to the action space, it is essential to mind the curse of dimensionality as explained in section 2.7.2. This means that the action space needs to be carefully designed so that it is not unnecessary large. This could potentially slow the learning time significantly. However, since the ROV is tasked to track the aruco marker, the set of actions is comprise of planar BODY velocities, that is v_x and v_y, and relative yaw between the ROV and the aruco marker itself, which is ψ_r. This way the ROV

may freely move in the environment while being able to control the relative yaw to be within the visibility range, which is $-90 \leq \psi_r \leq 90$. However, x_r and y_r could also be included as desired setpoints, yet this would expand the action-space, resulting in increased training time. The actions are presented in table 4 below.

Table 4: ROV actions set.

Actions	
Velocity in x-direction	v_x
Velocity in y-direction	v_y
Desired body fixed yaw orientation	ψ_r

Although the reward function, which will be discussed later, is reliant on relative states, i.e. r, it is also reliant on the body-fixed ROV velocities v_x and v_y. In reality, the BlueROV2 is designed to travel up to 2 m/s, though in the simulator, the maximum velocities is limited to 1 m/s. This was done to minimize the curse of real-world samples (see section 2.7.3), hazardous behavior when the trained model is exported for real-world lab experiments. Furthermore, it is normally harder to cope with due to real-world time-delays. The velocities are approximated using an imported Python library function called *scipy.integrate.ode.set_integrator('dopri5')*, which is an explicit Runge-Kutta approximation method of 5^{th} order put forward by Calvo et al. (1990) [15]. Similar to the radial distance, the radial velocity is used to determine the reward and is calculated in the same manner.

$$v = \sqrt{v_x^2 + v_y^2} \tag{4.2}$$

The radial velocity is taken into account because it is preferred that the ROV minimizes its velocity when it is in close proximity to the aruco marker. That is when $2 \leq r \leq 0$ m, the radial velocity should be $-0.1 \leq v \leq 0.1$ m/s. Thus, the ROV has a feasible time to react favourably should any urgent change in the current state be desired, e.g. braking without time-delay effects.

4.1.3 Simulator annotations

Since this is a software-environment, an ensuing **simplification** exists, where the simulator assumes that the ROV has continuous observability of the aruco marker. In other words, the ROV in the simulation-environment knows about the aruco marker's whereabouts at all times, and the ROV enjoys continuous updates of its relative position and orientation to the aruco marker at every time-step. This is

in heavy contrast to real-world case, where the state feedback is only accessible whenever the aruco marker is in the line of sight of the ROV's built-in camera. Subsequently, it is expected that state feedback signals dropouts will occur due to poor camera vision, light reflections causing disturbances on the aruco marker, or when the ROV loose the sight of aruco marker due to large yaw movement.

While the simulator utilizes a very accurately measured parameter setting specified for the BlueROV2 dynamics, it is important, as discussed in section 2.7.4, to consider the challenges related to under-modeling and model uncertainties. For example, a too large predefined time-step will render the system's state calculations inaccurate due to time-integration. Also, the simulator does not take the real-world time-delays into account during computer simulations. Subsequently, the ROV's behavior is optimized without this essential physical property. Subsequently, when conducting real-world experiments, the timing of a desired action execution given a state is expected to be slightly delayed, and is executed in a different state where a different suitable action is rather preferred. In worst case, this may cause chain of faulty action selections leading to poor performance.

4.2 PPO implementation

Python supports a large variety of ML libraries, which amongst other **PyTorch**, **Caffe**, **scikit-learn**, **Theano**, **Tensorflow** and **Keras**. Subsequently, this renders Python as one the most favored programming language used for data science and ML purposes, while being largely supported by softwares that are designed for use with robots.

A previous PPO implementation was found online on GitHub repository [19], which is based on Keras library with Tensorflow backend. This implementation was originally designed for use with an OpenAI environment based on an Atari game called **Breakout-ram**. The agent in this environment is trained to deflect incoming ball toward the upper multi-layered wall (see figure 12), with the end-goal of breaking through the wall.

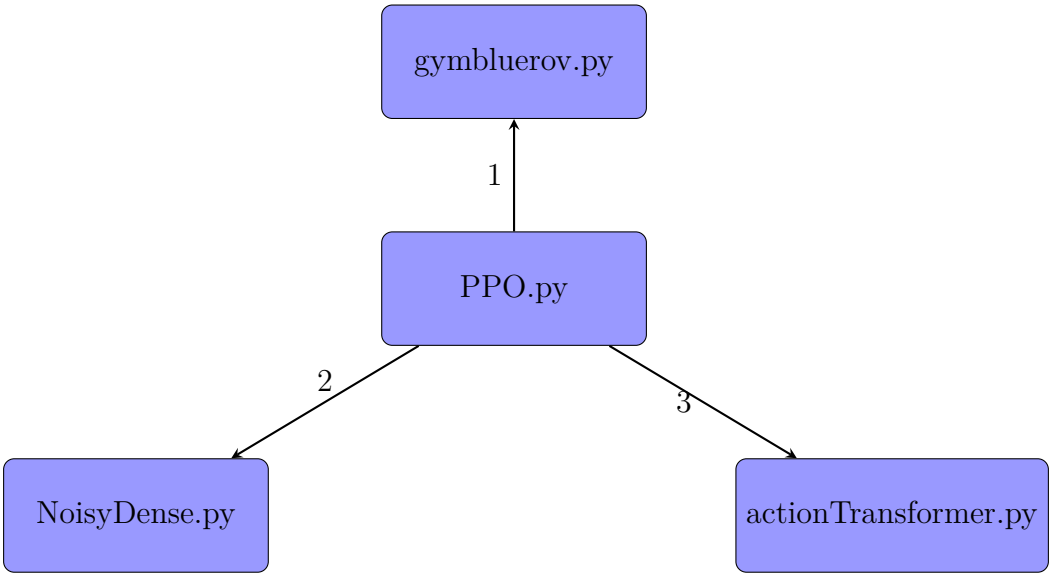
Figure 12: Breakout-ram environment in which the original PPO implementation was applied on. Credit: OpenAI.

In the GitHub repository, the author of the original PPO implementation stated that the code is based on the works of OpenAI's version of PPO by Schulman et al. (2017) [64], which is presented in section 2.6.8. Before modification was done for application in this thesis, it was made sure that the original implementation was tested for performance, which proved to be satisfying.

However, the original code was designed to be operate in discrete action space as the agent (i.e. the racket) took a single predefined action value for each time-step. Consequently, the implemented PPO algorithm was customized according to the original implementation, and was also designed to operate in a discrete action space. Hence, the ROV agent initiates one set of predefined action command per time-step. However, this required that action space being discretized into a total of 30 different actions to choose from. Each action set in thrust in x- and y-direction, and desired yaw orientation, ranging from -1 at minimum to 1 at maximum, is divided into 12 different intensities increasing with $+0.1666...$ from lowest action choice. While, not being fully continuous, the action space configuration yielded a convergence in performance, which will be further detailed in section 5.

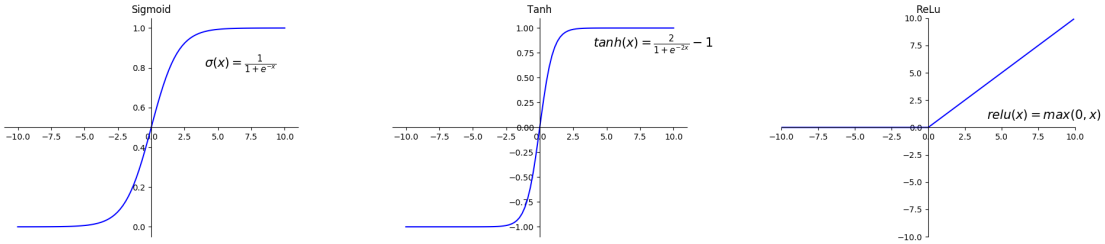
4.2.1 System architecture

In order to acquire deeper understand of the mechanism behind the model training procedure, each algorithms employed in the simulator will be further elaborated in this subsection. The following flowchart shows all the code files that comprise the simulator, and how it is arranged according to each other.



The ROV's properties, along with the environment dynamics and its physical boundaries, are defined in *gymbluerov.py*. This algorithm defines the **BlueROVEnv**-class, which bases its design using OpenAI's environment template. The BlueROVEnv-class specifies the conditions for terminating episodes, calculates the next ROV state based on the current state, applying action commands and specify the reward. All of these functions are called upon in the *step*-method which in turn outputs the calculated reward, next state of the ROV and any termination command. The step-function manages the time-step motion in the simulator, and represents the ROV's feedback to the PPO algorithm defined in *PPO.py*.

The *PPO.py* file contains the RL mechanism in the architecture. Figure 14 shows the classes used in *PPO.py* along with their respective member functions. The *Agent*-class initiates the PPO algorithm by building the actor and critic models separately. They are each based on two hidden-layered DNNs, and are activated by *Rectified Linear Unit* (**ReLu**) activation functions. The reasoning behind the choice of this activation function is due to it being less computational expensive, which is critical for the training time. Another point worth mentioning is that the other activation functions which were considered, that is sigmoid and tanh, are better suited for classification problems. This means that they are often used in supervised learning tasks as they have a convergence in both ends of their respective graphs (see figure 13). This renders the gradient region to be small, hence, the network either refuses to learn any further or becomes a very slow learner.



(a) (b) (c)

Figure 13: Plots of sigmoid, tanh and ReLu activation functions.

Since this is a RL problem, it is therefore favorable to apply an unbounded activation function such as ReLu. However, it should also be asserted that playing with the different activation function during training sessions did not affect the training time significantly. Each session lasted in approximately 20 minutes mostly depending on the arbitrary choice of actions the ROV agent made during the sessions. However, it is imaginable that for agent that requires a large number of training episodes will shorten its training time significantly with ReLu as opposed to sigmoid or tanh activation functions.

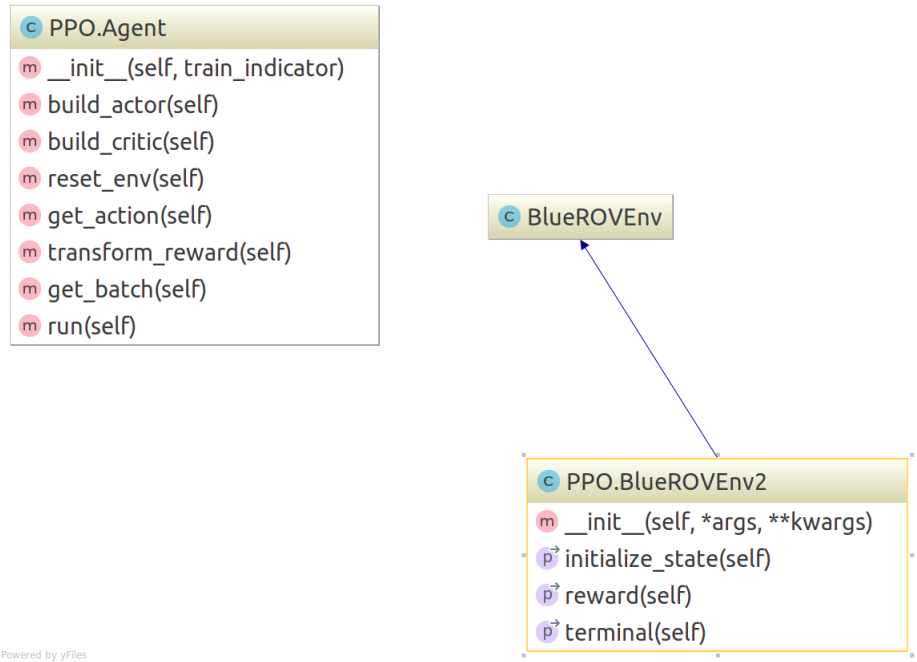


Figure 14: Class diagram of the PPO algorithm file.

In the *Agent*-class, the member function *get_action()* extracts the output action-

number from the actor-model in each time-step. The action-number, ranging from 1 to 30 represents the 30 discrete predefined action commands specified in the *action-Transformer.py*. This function takes in the action-number, which the *get_action()*-function passes down, and return an array of action commands of 3 elements. Each elements in the array represents the actions in the action-space.

A fundamental challenge in ML is to make the algorithm to perform adequately, not just on the training data, but also on new inputs during model testing phase. Many strategies have been applied to reduce the test-error, maybe at the expense of the training-error. These strategies are collectively known as *regularization* strategies. Goodfellow et al. (2016) [26] argues that regularization strategies have been major research endeavors in ML, emphasizing its importance. Adding noise to the DNN is a type of regularization strategy, which is what *NoisyDense.py* does in the actor-model. By adding noise to the DNN in the actor, it remedies the overfitting problem and improve generalization properties for the RL. Overfitting is, as described by Goodfellow et al. (2016), when the gap between the training-error and test-error is too large. Subsequently, the trained model will be robust to other different, or unaccustomed, data inputs. In physical aspect, this means that the trained ROV will have an increased chance of taking the right action when in an seemingly unknown state.

4.2.2 Reward function design

Reward function design has proven to be a significant factor when it comes to the overall ROV performance. The process of designing a reward function proved to be both challenging and a time consuming task. In order to see if the ROV learns further, i.e. improves its policy, the evolution of the accumulated reward per training episode needs to be studied. If it converges, this indicates that the ROV does not learn anymore and has found a seemingly optimal solution according to the reward function. If the reward does not converge over the training episodes, it may indicate that the ROV still resides in exploring phase or that the reward function is poorly designed as it does not properly guide the agent towards the desired behavior.

Even though, the convergence-check is the primary indicator for improvement in the agent's policy. It does not however guarantee that the agent has reached a feasible practical solution. Essentially, curse of goal specification (see section 2.7.5) is very relevant in this case, which may lead the agent astray as it will exploit the poorly

designed reward function by taking nonsensical actions. Consequently, in order to verify that the ROV's solution is rational with respect to tracking behavior, a trajectory plot is made to visualize the ROV's motion over time.

The following subsections proposes two reward functions that are considered promising in terms of its potentials and resulting simulator performance. The first proposal did not show a stable tracking behaviour in the simulator. Yet, it is nevertheless included in this thesis as it can be improved by tuning. The second proposal, however, yielded a satisfying tracking behavior, and its resulting trained model is taken to the physical experiments.

4.2.3 First proposed reward function

$$R = -\psi_r^2 - r^2 \tag{4.3}$$

Since equation 4.3 is negative with an upper boundary at ≤ 0, the ROV is fated to be punished with negative rewards. The severity of the punishment is dependant on the error on the relative state. The first term in the equation indicate that the ROV will be punished quadratically with the relative yaw orientation to the aruco marker. Likewise, the second term punishes the ROV quadratically with relative distance to the aruco marker. Hence, with respect to the reward function, the most ideal state for the ROV would be to reside as close to the aruco marker while pointing straight towards it. However, the following if-conditions 1 accompanies the reward function 4.3, which yields positive rewards in specified cases to hopefully guide the ROV to a tracking behavior.

Algorithm 1 Accompanying reward conditions.

1: **if** $\psi_r \geq \frac{\pi}{4}$ or $\psi_r \leq -\frac{\pi}{4}$ **then**
2: Let Reward = -100
3: **end if**
4: **if** $\psi_r \leq 0.17$ and $\psi \geq -0.17$ **then**
5: Let Reward = Reward + 100
6: **end if**
7: **if** $r \geq r_d$ and $r \leq 2.5$ **then**
8: Let Reward = Reward + 100
9: **if** $v \leq 0.2$ and $v \geq -0.2$ **then**
10: Let Reward = Reward + 100
11: **end if**
12: **end if**
13: **if** Traveling out of boundaries in the basin **then**
14: Let Reward = -1000
15: And reset the environment and terminate the episode
16: **end if**

Algorithm 1 was implemented for reward shaping purposes. Lines 1-3 specify whenever the ROV has a ψ_r larger than 45 degrees, the ROV is punished by -100 reward. This way the ROV is trained to maintain a line of sight towards the aruco marker, which is a favorable practice when applied in real-world. Lines 4-6 highlight that while maintaining a relative yaw displacement between 10 and -10 degrees the ROV will be rewarded with 100 extra reward points. Lines 8-11 encourages the ROV to position itself within an ideal range of distance, which is defined to between r_d and 2.5 m. In this case r_d is set to 1 m. In addition, while maintaining an ideal distance to the aruco marker, the ROV will be rewarded with extra 100 reward by keeping a velocity between -0.2 and 0.2 m/s. This if-condition is introduced so that the ROV is encouraged to slow down its pace when in close proximity to the aruco marker. This is to increase the ROV's maneuverability when close to the aruco marker in case of sudden emergency action is in order. The last if-conditions in the algorithm punishes the ROV severely when it travels out of boundaries. Subsequently, the ROV is trained to operate within the defined MC-lab boundaries. Physically, this ensures that the ROV has a significantly lower chance for colliding with the edges of the test basin, thus, avoiding hazardous situations.

4.2.4 Second proposed reward function

In contrast to the previous proposal, this proposal, however, is not based on a singe reward function accompanied by guiding if-conditions. It is rather defined with a mesh-grid based rewarding mapped on the 2D plane. In other words a 2D grid-world divided is mapped into the test basin, which is divided in smaller mesh areas. Each mesh area in the plane is assigned a reward value, which is defined by radial distance r. Figure 15 visualize an example of how each position in the basin can be assigned a reward with grid approach.

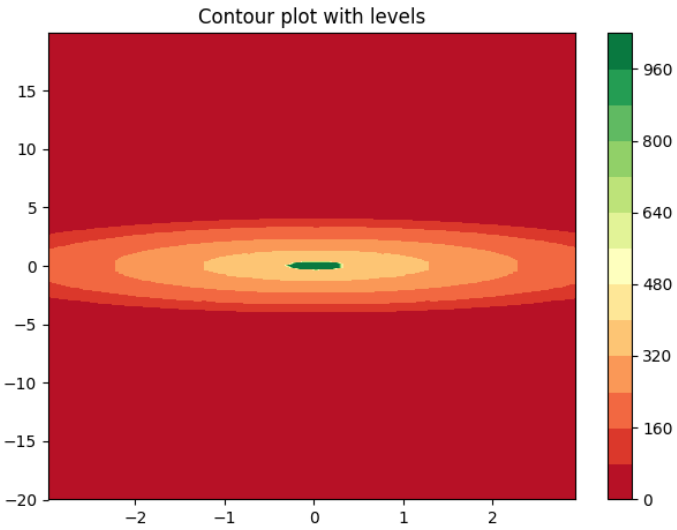


Figure 15: Example mesh-grid reward. Red indicate zero reward, while dark green is the highest reward.

In this example the rewards are purely based on predefined radial distances in the mesh-grid environment. Here, the closest interval yields highest reward (green), while the further the position is from the origin, that is the aruco marker's placement, the lower the returned reward is.

However, in the case of ROV training, separate mesh-grid rewards were specified when the ROV velocity is smaller than or equal to zero ($v \leq 0$), and when the velocity is larger than zero ($v > 0$). The setup is shown in the following algorithm.

Algorithm 2 Mesh-grid based reward function

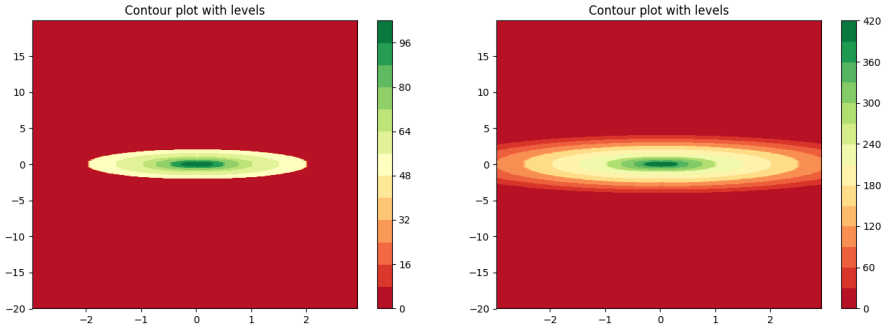
1: **if** $v \leq 0$ **then**
2: $r < 0.3 \rightarrow$ Reward $= 100$
3: $r > 0.3$ and $r < 0.5 \rightarrow$ Reward $= 90$
4: $r > 0.5$ and $r < 0.75 \rightarrow$ Reward $= 80$
5: $r > 0.75$ and $r < 1 \rightarrow$ Reward $= 70$
6: $r > 1$ and $r < 1.5 \rightarrow$ Reward $= 60$
7: $r > 1.5$ and $r < 2 \rightarrow$ Reward $= 50$
8: $r > 2$ and $r < 2.5 \rightarrow$ Reward $= 0$
9: $r > 2.5$ and $r < 3 \rightarrow$ Reward $= 0$
10: $r > 3$ and $r < 3.5 \rightarrow$ Reward $= 0$
11: $r > 3.5$ and $r < 4 \rightarrow$ Reward $= 0$
12: $r > 4 \rightarrow$ Reward $= 0$
13: **end if**
14: **if** $v > 0$ **then**
15: $r < 0.3 \rightarrow$ Reward $= 400$
16: $r > 0.3$ and $r < 0.5 \rightarrow$ Reward $= 360$
17: $r > 0.5$ and $r < 0.75 \rightarrow$ Reward $= 320$
18: $r > 0.75$ and $r < 1 \rightarrow$ Reward $= 280$
19: $r > 1$ and $r < 1.5 \rightarrow$ Reward $= 240$
20: $r > 1.5$ and $r < 2 \rightarrow$ Reward $= 200$
21: $r > 2$ and $r < 2.5 \rightarrow$ Reward $= 160$
22: $r > 2.5$ and $r < 3 \rightarrow$ Reward $= 120$
23: $r > 3$ and $r < 3.5 \rightarrow$ Reward $= 80$
24: $r > 3.5$ and $r < 4 \rightarrow$ Reward $= 40$
25: $r > 4 \rightarrow$ Reward $= 10$
26: **end if**

Also in this case, a set of accompanying if-condition guides the ROV during the training sessions.

Algorithm 3 Accompanying reward conditions.

1: **if** $\psi_r \geq \frac{\pi}{4}$ or $\psi_r \leq -\frac{\pi}{4}$ **then**
2: Let Reward = -1000
3: **end if**
4: **if** Traveling out of boundaries in the basin **then**
5: Let Reward = -1000
6: And reset the environment and terminate the episode
7: **end if**

The the if-condition above is expressed with mesh-grids, and the resulting plots shows the subsequent reward configurations for the ROV in the 2D simulator environment.



(a) (b)

Figure 16: Mesh-grid reward when $v \leq 0$ (a), and when $v < 0$ (b).

From figure 16 is it notable that the rewarding area of the test basin is much higher while the ROV has a velocity higher than zero. This is so that the ROV is encouraged to maintain a velocity rather than standing still far away from the aruco marker, leading to an increased explorative behavior. As such, this increases the chance for the ROV to probe in areas near the optimal solution, which in this case is the green midpoint surrounding the aruco marker. By assigning a high contrast in reward between the zero-reward area (i.e. $r \geq 4$) and rewarding area ($r < 4$), thwarts any chances for the ROV to get stuck in a poor local minimum. For example, in earlier design attempts, the ROV would be rewarded even if it was distant positioned (i.e. $r \geq 4$). This would render the ROV to be stuck in that position without moving as it would accumulate rewards anyways.

An important aspect worth noticing in this proposal, is that there are lesser effort on guiding the relative yaw orientation, and desired interval of radial distance between

the ROV and aruco marker, as it is on the previous rewarding proposal. Algorithm 3 only specifies severe punishment (i.e. Reward = -1000) is specified whenever the ROV looses its line of sight of the aruco marker, and likewise punishment if the ROV should travel out of the basin's physical boundaries. While no desired radial distance is specified. This means that the ROV is freely to "collide" with the aruco marker, which is, in real-world application, a possibly dangerous behaviour. However, it can easily be remedied with a kill-switch logic designed to stop the ROV whenever a dangerous situation occurs.

4.2.5 Hyperparameters

To successfully train a RL model, it often involves tuning the training hyperparameters of the RL algorithm. In this sense, the hyperparameters is understood as the predefined parameters which manages the physical configuration of the RL algorithm. This subsection introduces some well-renowned practices for tuning hyperparameters for the PPO algorithm.

Tuning the hyperparameters are typically performed when the default configurations does not seems to yield the desired level of performance. However, the hyperparameter configurations in this thesis was used as default values originally defined by the author of the implementation. However, for further works or later improvements, knowledge of each of the hyperparameters physical influence on the training may come in handy.

Gamma (γ) is the discount factor for future rewards. This can be viewed as how far the agent should care about further rewards. If the agent ought to be prepared for rewards in distant future by acting in the present, gamma should be high. While in situations where rewards are more immediate, this value can be smaller. Typical range is $0.8 - 0.99$, and in this implementation $\gamma = 0.99$ is used.

Batch size is the amount of experiences (i.e. obtained agent observations, actions and rewards) is used in one iteration of a gradient decent update. The batch size should always be a fraction of the *buffer size*. For continuous spaces, this value should be large and in the order of thousands, typical range is $512 - 5120$. While for discrete spaces the value is far smaller, in the order of tens, with typical range being $32 - 512$. For this implementation a batch size of 512 is used.

Buffer size coincides with the amount of experiences the agent should collect before updating the model and learning is done. This is defined as a multiple of

the batch size. Typically, buffer size ranges between $2048 - 409600$, while in this implementation a size of 2048 is used which is exactly four times larger than batch size.

Number of epochs is the number of entire dataset that passes through the experience buffer in the actor-critic model during a gradient iteration step. A larger batch size allows for larger epoch number. Decreasing the epoch number will lead to a stable updates, hence, stable learning process, but at the cost of learning speed rendering it to a matter of trade-off balance. Typically range is $3 - 10$, and in this implementation 10 is used.

Learning rate is related to the significance of the each gradient descent step. Typically, this should be lowered if training is unstable, that is no convergence in sight, and the reward does not consistently decrease. Typical range is $10^{-5} - 10^{-3}$. In this thesis both the actor model and critic model uses learning rate at 10^{-4}.

Loss clipping (epsilon) ϵ is the tolerable threshold for divergence between the old and new policies during gradient descent updates. Small epsilon values results in stable updates while at the expense of training speed. Hence, this parameter is also a trade-off configuration. Typical range is $0.1 - 0.3$, and in this thesis $\epsilon = 0.2$ is used.

5 Simulation results

In this section, the simulation results with both reward functions and trajectory plots are presented. In addition, a discussion highlighting the performances of both simulation instances is also covered.

The optimal number of training episodes is found be around 850 or so episodes, while in this thesis 870 episodes was chosen. The reason for this choice of number is because if ROV is trained with more than 900 or so episodes, it tend to exploit the reward function and reach a poor solution while acquiring rewards. Typical poor solution observed is when the ROV drifts away from the aruco marker rather than converge toward it, or for instance maintain a fixed position while rotating with a constant yaw rate. Hence, at the episodes range of 850-900, the ROV is considered as it is "onto something" in terms of desired tracking behavior, and from then it looses its track when it takes advantage of the reward function.

Results with reward proposal 1

Figure 17 shows the resulting evolution of rewards accumulated per episode when employing the first proposed reward as described in subsection 4.2.3.

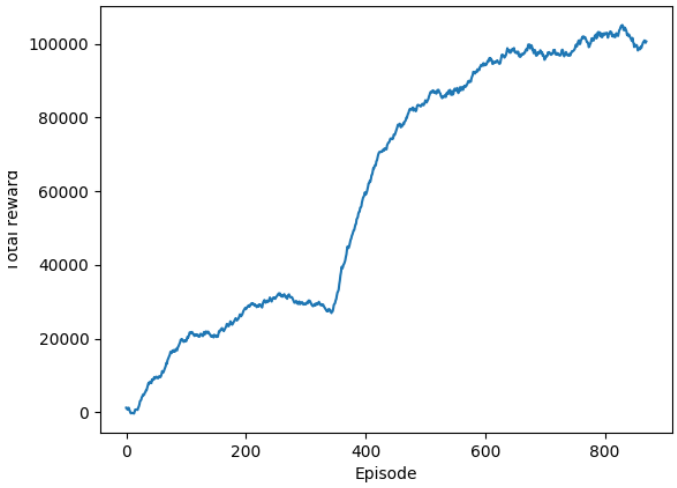


Figure 17: Growth of reward in throughout the episodes.

The following figures shows arbitrary trajectory plots per episode.

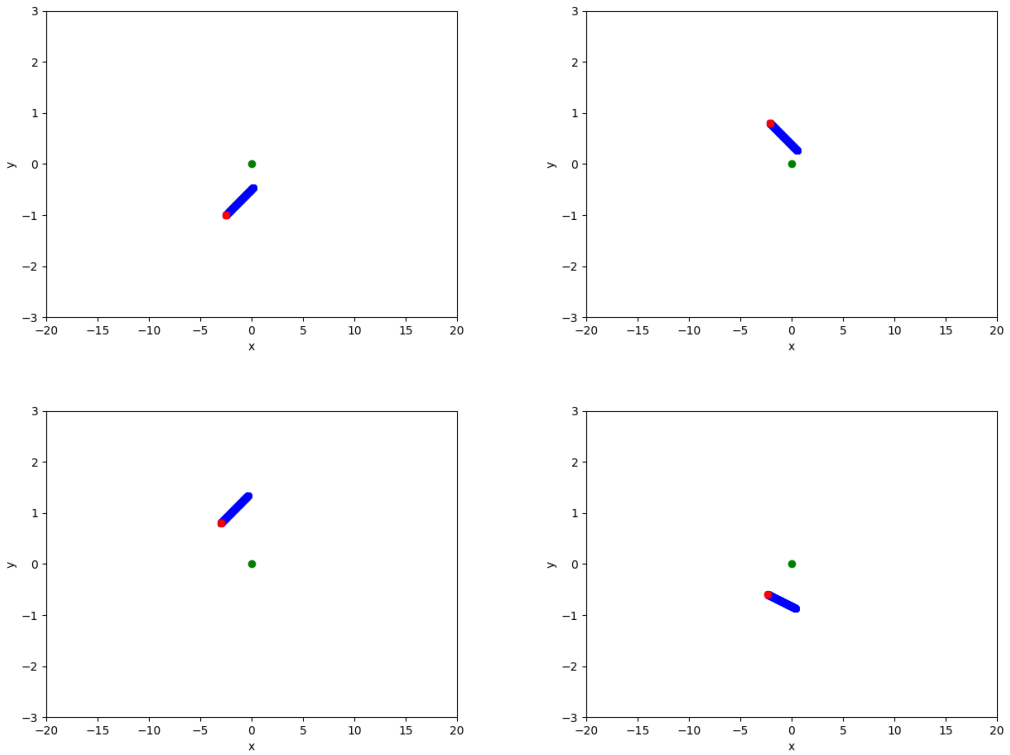


Figure 18: Trajectory plots of the ROV. Red dot is the starting point, green dot is the aruco marker's position, and blue is the ROV's trajectory.

Results with reward proposal 2

The evolution of accumulated reward when using the mesh-grid rewarding approach is shown in the following figure.

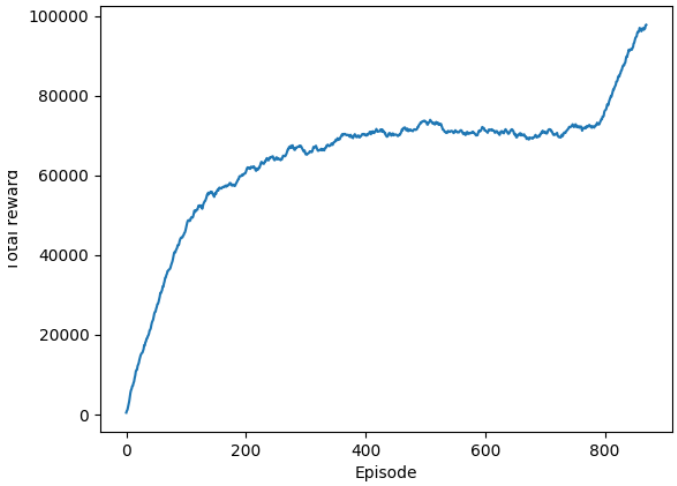


Figure 19: Growth of reward in throughout the episodes.

Following figures are arbitrarily selected trajectory plots for an episode.

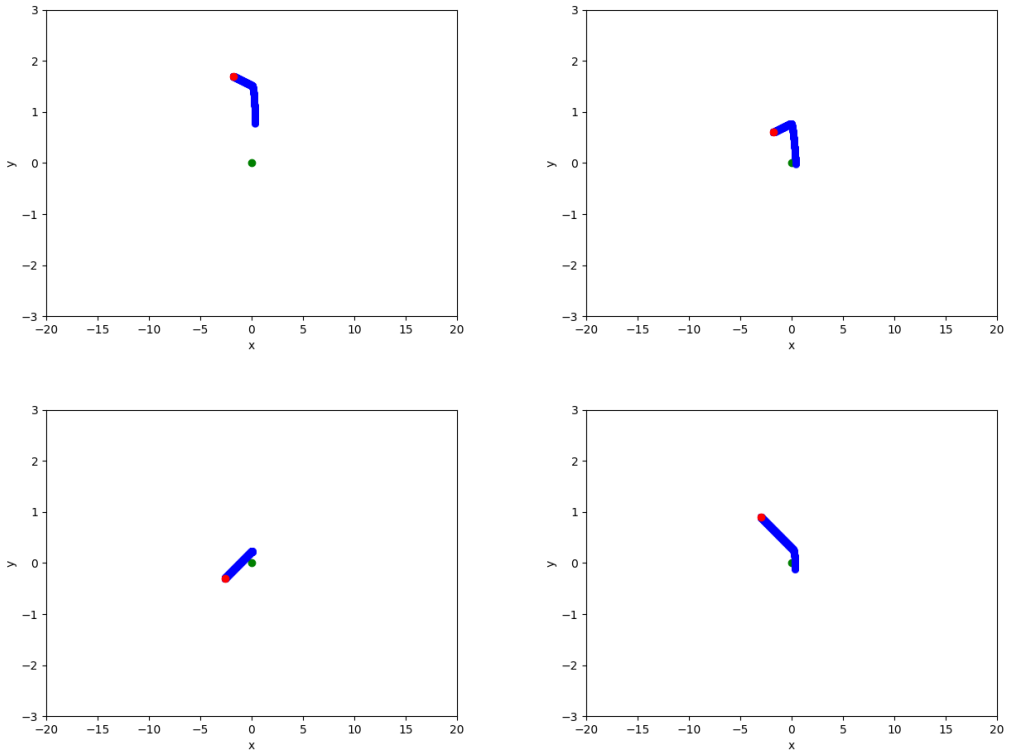


Figure 20: Trajectory plots of ROV performance.

5.1 Discussion of simulation results

Each training sessions resulted in a great variety of quality of the tracking behavior. Subsequently, an extensive number of simulations had to be done with each proposed reward functions before retiring with the best obtained models with each respective reward functions. More precisely, up to 60 trials were run. The evolution of rewards in both simulation cases indicate that the agent's policy is improving in terms of reward accumulation.

Reward function proposal 1

In the case of reward proposal 1, it is noticeable in figure 17 that the reward evolution graph is steep. This indicate that the learning speed is high and the agent updates its policy continuously as new and seemingly better solutions are found. This implies that the nature of the reward function results in a high explorative behavior. Yet, the resulting trajectory plots in figure 18 shows that the ROV roughly travels with 50 percent chance in the opposite direction of the aruco marker. A reason behind such performance is that it is misguided to resolve that the most subtask to optimize is to maintain a line of sight heading towards the aruco marker. While at the same time neglect the other important subtask, which is reducing the radial distance to the aruco marker. This explains why the ROV have a constant heading orientation towards the aruco marker during the episodes, as it has learned not to risk punishment from relative yaw error in function 4.3, hence, travelling in straight line only.

Reward function proposal 2

While as for the second simulation case, the reward evolution graph resembles a logarithmic function with a sudden increase at the of the training session. This can be viewed as the agent has an extensive learning rate in the beginning of the session, while reaching a long lasting convergence before suddenly obtaining an even better solution. In contrast to the reward evolution in the previous case, this reward function renders a far less explorative behavior with a greedy actions selection policy. However, even though both reward functions yields substantially high reward (about 100000), the performance of the second proposed reward function is much more stable. The trajectory plots in figure 20 shows how the heading motion is not constrained to a constant value during the run, but is used reasonably. This

allows for a more flexible motion approaches which renders it easier for the ROV to encroach the aruco marker from various positions. While being more reliable in performance compared to the previous proposed reward function, the trajectory in this case is not completely flawless. The figure shows that the ROV has a tendency to take a irrational detour before converging towards the aruco marker. Subsequently, it is expected that in the real-world experiments the ROV will perform such mistakes at times.

However, it should be specified that despite the latter reward function yielded a much more promising performance, the ROV, however, showed to utilize only two action commands. Which implies that the policy is stuck in a poor local minimum. In detail, those actions are 0.33 m/s desired velocity in x-direction v_x, and -1 m/s desired velocity in y-direction v_y. This implies that the ROV has a significantly limited maneuverability as only two out of 30 action commands are used, and may lead to poor real-world performance. This also explains the L-shaped form of the trajectory plot, as it is able to travel only in positive x- and y-directions in BODY frame. Figure 21 shows the velocity evolution in x- and y-direction during a simulation run.

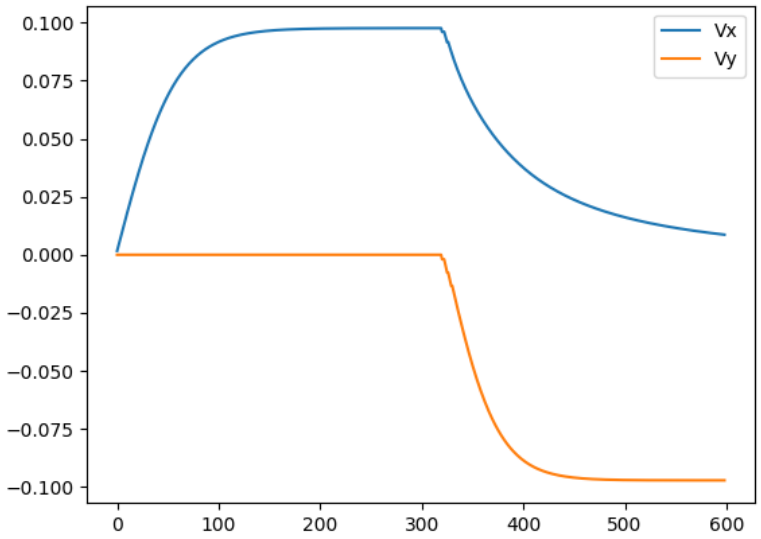


Figure 21: The x-axis show the instantaneous velocities, while y-axis records the time in seconds.

Settling down with only 870 training episodes will yield a policy that is a local solution, even if it still yields a feasible behavior. Hence, the resulting policy is not

robust to unexpected situations for the agent, as it is seen in the experiment results where unwanted real-world issues come into play. However, while training with more than 2000 episodes, and yet yield an even inferior policy solution indicates that the problem is a matter of curse of goal specification.

6 Outline of the real-world experiments

To verify the trained model a real-world test was performed with a ROV to assess its convergence property towards a fixed aruco marker. This is a simple method to examine whether the tracking behavior is achievable if the aruco marker was in fact moving.

All real-world experiments were conducted in the MC-lab at NTNU, which supports underwater experiments. In this thesis, the experiments were executed with a ROV of the type BlueROV2, which will be further detailed in the following subsection.

This chapter specifies the most essential lab components and how they were interconnected during the physical experiments. The configurations on the BlueROV2 used in the experiments are detailed and the positioning systems used for obtaining the ROV's states. The use of computer vision technology is also highlighted and its challenges considering real-world performance. The MOOS-IvP framework is used for ROV application is detailed, followed by an overview of the complete laboratory setup.

6.1 BlueROV2

Figure 22 depicts the BlueROV2 which is developed by Blue Robotics, and is a rather small research-class ROV with the main purpose of scientific use at NTNU, Departement of Marine Technology (IMT). It's thruster configuration allows for motion in surge, sway, heave and yaw. It is also equipped with a small on-board processing unit, Raspberry Pi 3 (RPI3). While being equipped with a full HD resolution camera for observation purposes, the BlueROV2 can also be armed with a manipulator for intervention tasks. The ROV contains a sensor suite of 4 units which are gyroscope, accelerometer, magnetometer, and a pressure sensor. It can also be manually steered by a Xbox controller or a topside PC, while in the experiments, it is primarily used as a kill-switch whenever the ROV comes close to a hazardous situation.

Figure 22: BlueROV2. Credit: Blue Robotics.

While in this thesis the experiments are taking place on a 2D plane, the ROV is thereby required to maintain a fixed position in heave. This was achieved by implementing an active heave controller. While desired depth was regulated by placing four 2.5 kg weights symmetrically on the ROV, weighting it down to a certain depth where ROV surfacing is infrequent.

6.2 Qualisys Motion Tracking system

Qualisys Motion Tracking (QMT) is a motion observation system used in the MC-lab, to obtain position and Euler orientation of an object of interest. This tracking technology is set up by 6 high-speed infrared cameras fixed in the test basin which detects markers mounted on a body in motion. In the experiments, four markers was used on the ROV to uniquely determine the position, which also was the minimum requirement. The camera recognizes the markers underwater and uses triangulation principles to calculate position and orientation, which is then communicated trough Wi-Fi in the MC-lab. The cameras are configured to produce measurements at a frequency range of $100 - 150$ Hz.

A NED frame is defined by Qualisys based on the area covered by the 6 surrounding cameras. While determining the orientation of the ROV, Qualisys also defines a body-fixed frame on the detected body, which operates within the NED frame. Figure 23 visualizes the lab environment in a simplistic manner.

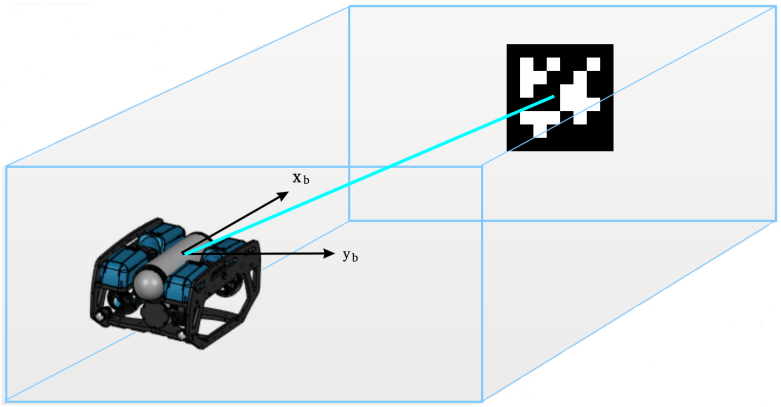


Figure 23: BlueROV2 2D BODY frame in a 3D space. The local reference frame are the same as shown in figure 11.

Since QMT position estimates are deemed accurate, these were then used as the true ROV position in the test basin. The linear velocities were estimated by time derivation with the position measurements as basis. However, since the position estimates contained noise, this rendered the velocity measurements noisy and therefore Kalman filter was employed. As a result, this caused a minor time-delay on the reference, nevertheless, this trade-off for accurately measuring the velocity proved to be favorable during experiment sessions.

6.3 OpenCV and Computer Vision functionality

Open Computer Vision) (OpenCV is an open source programming library developed for use in applications that utilizes computer vision technology. Originally created by Inter, but is currently developed at *Willow Garage* under a BSD licence (as of 2018). Its primary interface is based on C++ programming language, however, an equally prominent Python wrapper is supported and used in this thesis. Other programming languages are also supported such as Java and MATLAB.

OpenCV supports a large variety of functions for the capture, analysis and manipulation of visual data. In addition, the library is widely favored by programmers as it is stable in use and vigorously supported by its developers. A large portion of the library also provides user interface and pattern recognition functions. The latter is extensively used to detect aruco marker and calculate its relative pose to the camera's position and orientation. One of the most powerful functions the library maintain is the many built-in functions designed for real-time image processing. This renders OpenCV ready for use in self-piloting vehicles and other forms

of innovative digital applications.

In the physical experiments, an already available computer vision implementation from GitHub [35] was employed with slight adjustments to render it ready for use with BlueROV2. The computer vision implementation is an OpenCV based script written in Python specifically designed to detect and calculate the 6 DOF pose of predefined aruco markers. The original implementation was designed to be used with the PC's built-in web-camera or an external camera, and was tested for performance and accuracy in measurements prior to being applied in this thesis.

While there was possibilities to apply the computer vision functionality directly on the BlueROV2's RPI3 computer, this would however render the RPI3 slow as image processing tasks requires high CPU power. Consequently, leading to an unstable communication line between the topside PC and the BlueROV2 itself. Thus, the script was implemented on the topside PC instead, which is typically more powerful. Then a pipeline functionality was used to stream the camera feed from the BlueROV2 directly to the topside PC, such that the image processing tasks could be conducted in the latter device. Figure 24 shows the in-action image of the computer vision functionality implemented in the BlueROV2.

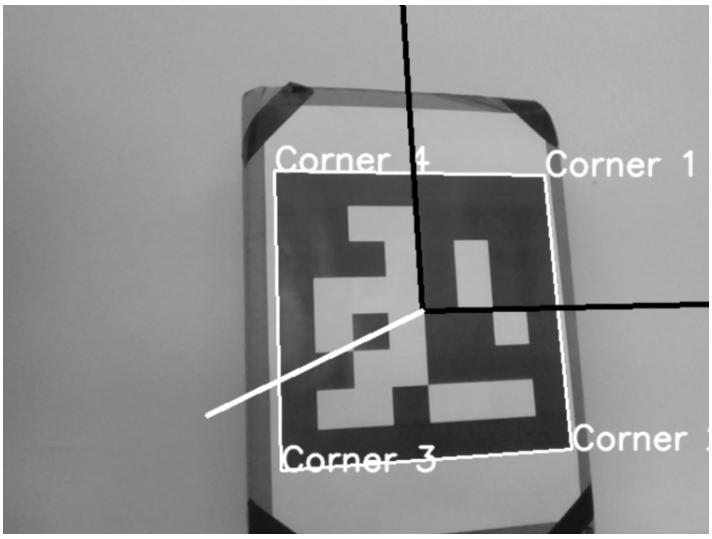


Figure 24: BlueROV2's computer vision apparatus.

Despite yielding quite favorable measurements in both orientations and translations, this implementation does, however, sometimes struggle with sunlight reflections. Subsequently, this whitens parts of the the aruco marker, rendering it undetectable for the computer vision apparatus. In practical sense, this means that the BlueROV2 occasionally looses sight of the aruco marker, and will lead to decreased

real-life performance as compared to simulator performance. Hence, the light intensity in the environment must be taken into consideration. However, this issue was remedied by only using the BlueROV2's headlights, while the rest of the test basin facility was darkened. Hence, there is minimal chance of light reflections caused by the ceiling lights.

6.4 MOOS-IvP framework

MOOS-IvP is an open-source project designed and developed for use with autonomous unmanned marine vehicles. The project remedies and supports increased mission complexity and duration, increased capability in on-board sensors processing and computing power. In addition, it also supports a larger number of users and owners of unmanned vehicles. The MOOS-IvP framework is an implementation of an autonomous helm (known as steering wheel of a ship) with considerable support applications that aspires to administer a feasible autonomy system.

MOOS-IvP comprises of two distinct software modules, that is MOOS (Mission Oriented Operating Suite), and IvP Helm. The former is a work of University of Oxford which furnishes core middleware capabilities in a publish-subscribe software architecture, including various applications commonly used on unmanned marine robotic and land robotic applications using MOOS. The IvP (Interval Programming) Helm module is a MOOS application that is, along with other MOOS applications, available in the MOOS-IvP project. While the IvP part specifies the multiobjective optimization method used by IvP Helm for intervening between clashing behaviour signals in its behavior-based architecture.

Over the course of its history, MOOS-IvP has been refined towards a less defined mission structure with respect to sequence of tasks, but rather as a set of complete autonomy modes. These modes are accompanied with conditions, field commands and events that defines transition between the modes. Originally developed on a C++ back-end, MOOS-IvP supports a Python wrapper and is therefore feasible to employ in the experiments in this thesis. In Benjamin et al. (2010) [9], a detailed explanation of how this framework is implemented for use in underwater robotics.

6.5 Lab setup

The diagram in figure 25, which is based on a similar diagram found in Sandøy [57], highlights the essential components used in the lab and their respective communication lines.

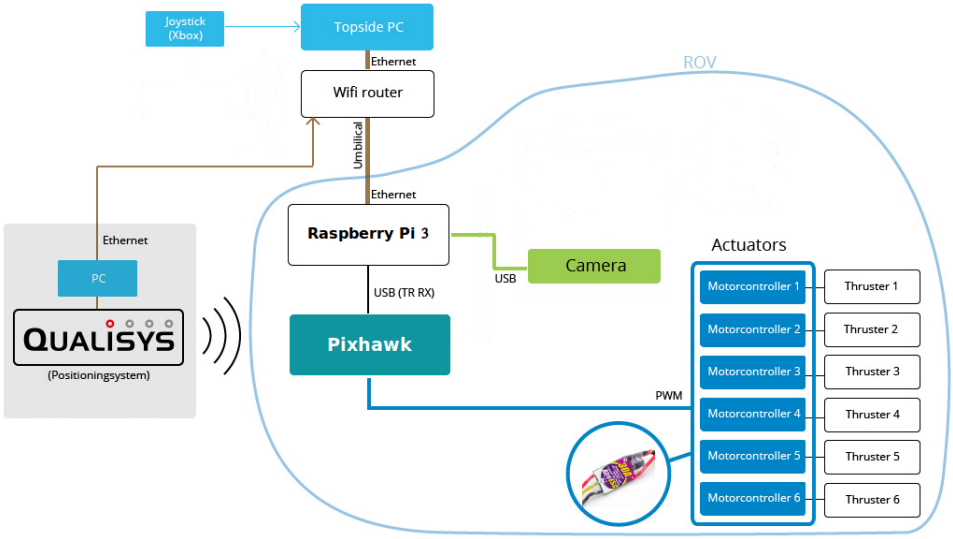


Figure 25: Diagram showing the different communication lines between the components. Courtesy of Sandøy [57].

The RPI3 on BlueROV2 runs Ubuntu 14.04 with the framework MOOS-IvP, while the topside PC also runs MOOS-IvP with Ubuntu 16.04 as operating system. Pixhawk is another on-board circuit on the BlueROV2 which handles the signaling to the motor controllers, that is the six thrusters. The communication line between the BlueROV2 and the topside PC is a 40 m umbilical cable with ethernet connection. The BlueROV2 is powered by a rechargeable battery shown in figure 25, where the main cause of power drainage is the thruster workload. The thrust signals obtained from Pixhawk are passed on to a component named Pulse Width Modulation (PWM), which maps the signals to desired thrust values for the motor controllers.

7 Real-world experiment results

7.1 Experiment remarks

The experiments conducted was mainly to evaluate the ROV's ability to converge towards the aruco marker. Such arrangement is a good way to examine the ROV's tracking capabilities in case of a moving aruco marker. Figure 26 shows the in-action picture of an experiment procedure.

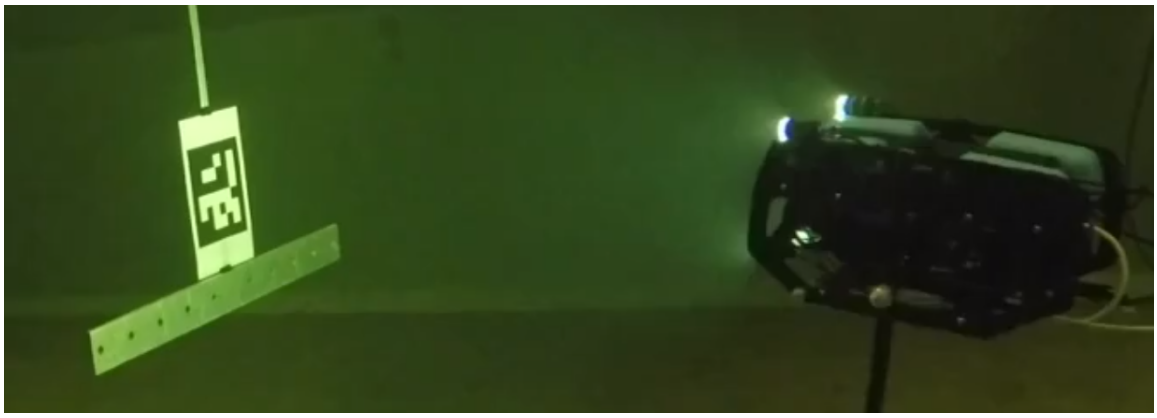


Figure 26: Experiment case of ROV convergence test.

This picture was captured with a GoPro camera mounted on the side of the test basin. From the figure, it can be noticed that the underwater visibility is significantly reduced by the seemingly green contamination. This is a result of lack of chlorine in the test basin. As a consequence, this tainted test basin rendered Qualisys inapplicable because its positioning cameras could not detect the underwater markers. Subsequently, no logging data was obtained of the ROV's position and velocity estimation, hence, no direct trajectory plots were obtained from experiment data.

However, since the ROV was trained to exploit only two action commands, as discussed in subsection 5.1, the subsequent travel path of the ROV was easily surveyed. As an effort to provide an impression on the ROV's typical performance, handcrafted plots were created based on various trajectory samples observed during the experiments, and presented in the next section.

7.2 Experiment results

The GoPro camera mounted on the bridge in test basin, and at the side edge of the basin, provided essential visualized information on the ROV's behavior. Based on this, a trajectory plot was carefully made. The following plots are some of the example trajectories which the ROV executed during the experiments, and are accurately representing the performance.

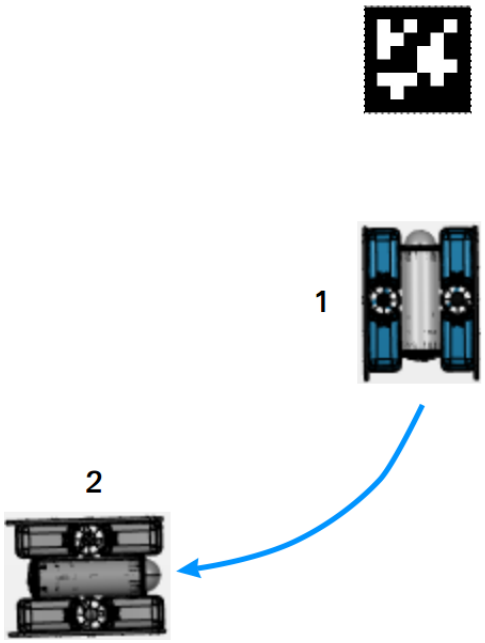


Figure 27: Sample 1: The ROV diverges backwards to the left.

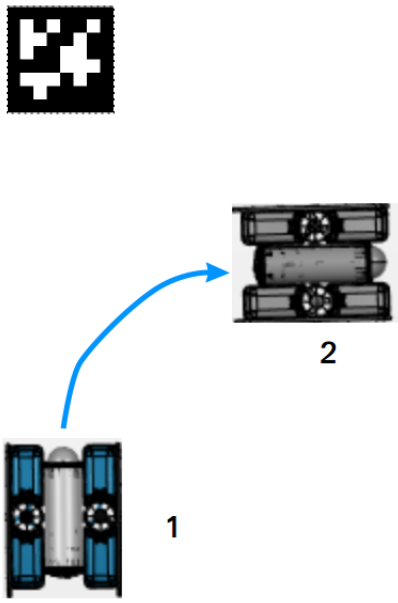


Figure 28: Sample 2: The ROV moves forward while diverging towards right.

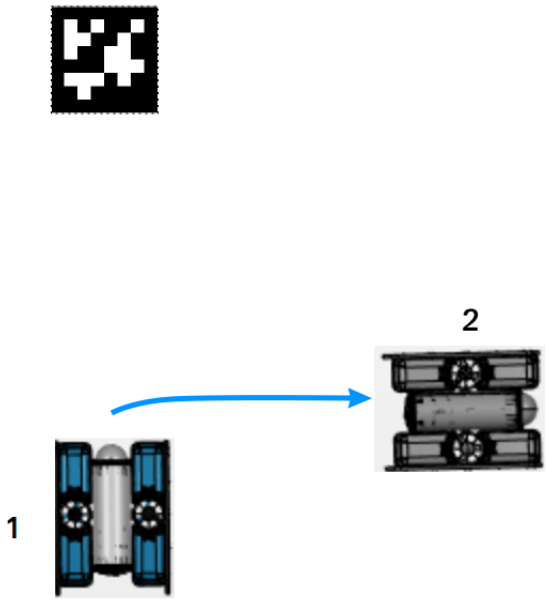


Figure 29: Sample 3: The ROV diverges directly towards the right.

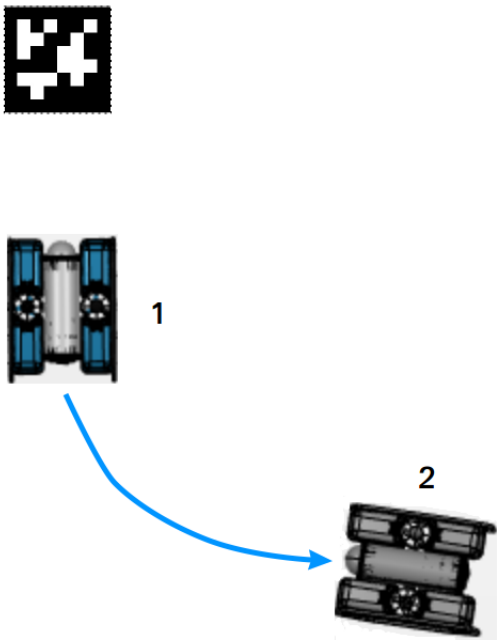


Figure 30: Sample 4: The ROV diverges backwards to the right.

7.3 Discussion of the experiment results

From figures 27 - 30 it is apparent that the ROV struggles to travel to the aruco marker. Similar to the simulation results, its trajectory path has a L-shaped form. However, in this case the ROV turns away from the aruco marker, resembling an avoidance behavior rather than a tracking behavior. In addition, there was no pattern found at which cases the ROV would turn to a particular direction given a state. Subsequently, the ROV behavior seemed unpredictable.

This behavior is to some extent as expected since the simulation-trained model performance was not optimal in terms of action selection. In section 5.1 it was explained that the maneuverability of the ROV was severely limited due to its policy being stuck in a poor local minimum because only two action commands were used. Consequently, this issue, which is related to curse of goal specification, affects the real-world performance accordingly.

In addition, the issue of limited observability in real-world scenarios had a clear presence in the experiment sessions, and was also discussed in subsection 4.1.3. Considering that the ROV in the simulator assumes that the aruco marker's whereabouts is known at all times, this is however not the case in real-life as the line-of-sight issue is an essential constraint. From figures 27 - 30 it is evident that the

ROV looses its sight on the aruco marker once it turns away and the relative yaw exceeds 90 degrees. Hence, the ROV ceases to receive the relative state feedback it requires to navigate itself towards the aruco marker. Subsequently, this issue renders it harder for the ROV to recover to path when it first looses its sight of the aruco marker.

Another noteworthy challenge was the BlueROV2's excessive pitch motion following an initial forward acceleration motion. This resulted in the ROV's camera-end pointing upwards and above the aruco marker, missing the sight. This was attempted to be remedied by placing weights on the ROV symmetrically in both ends, such that a heavier ROV would be more stable in terms of motion in pitch. Subsequently, the pitch orientation was to some extent stabilized, yet not fully desirable stable. As a result of acceleration done by the ROV, an initial exceeding jump in pitch motion did occur, which resulted in inadequate relative state observability, and is a significant contributing factor in the ROV's poor performance.

A minor issue encountered was light reflection occurring on the aruco marker as a result of powerful ceiling-lights in the laboratory. However, this issue was neglected as it only resulted in short-lived signal dropout in relative states to the aruco marker. Hence, no significantly negative impact on the performance of the ROV was noticed. However, this could be attended by lowering the ceiling lights' intensity or turning them of, while conducting the experiments with the ROV's head lights only.

8 Conclusion and further work

8.1 Conclusion

The objective of this thesis was to design a model-free RL based tracking behavior for underwater vehicles, with computer vision technology as a key feature. This was carried out by creating a RL algorithm based on an existing online implementation. The RL method chosen for tracking task is a PPO algorithm, which was suitable for training a vehicular robotic system like a ROV. The ROV chosen for the physical experiment is the BlueROV2 due to its easy modification properties.

A simulator was created for model training and performance verification purposes. An approximate underwater environment for a BlueROV2 was implemented in the simulator such that the model training is close as possible to the real-world dynamics. The training of the ROV was done with the PPO algorithm where the hydrodynamic environment is based on the BlueROV2's model. The hydrodynamic parameters were acquired from Sandøy (2016), which are originally obtained from BlueROV1. However the latter's hydrodynamic properties are approximately the same as for BlueROV2 and any difference in physical parameters is thus trivial. During the training phase, different variants of reward functions were investigated, where two of them were highlighted in this thesis. While only the mesh-grid based reward function was further employed in the real-world experiments. To verify the performance of the model, the evolution of accumulated rewards in each training session, and trajectory plots of the ROV, were evaluated. The challenge related to exploration-exploitation during the simulation based training were highlighted along with the importance of a proper reward function design. While the resulting policy did prove to yield a tracking behavior, it did not, however, succeed to utilize all of the available action space in the robotic system.

In preparation for real-world experiments, a computer vision module was employed for detecting aruco markers with the BlueROV2's RPI-camera. This was also based on an online implementation, but slightly modified for use with BlueROV2. This algorithm takes use of the popular OpenCV module for the computer vision process. This implementation also calculates the aruco marker's relative pose to the camera, which proved to be convenient for use in the real-world experiments since the ROV is trained with the relative states as principal parameters. A MOOS-IvP framework is used for rendering an eazy communication line between the topside PC and BlueROV2 during the experiments.

In comparison to simulation results shown in section 5, the experimental results evidently shows an inferior tracking behavior. There are clear signs of challenges related to real-world issues and goal specification, while these two issues in combination would naturally render a more troublesome resulting tracking performance. Due to the scope of this thesis, a limited amount of reward shaping effort was available. A more focus on further reward design should be done in further works, as reward shaping proved to be a key factor for successful simulation results. The real-world issues in the experiments are mainly due to poor physical configurations on the BlueROV2.

8.2 Further work

Both the simulation and real-world experiment results indicates that there is room for future work. The simulation results did not yield an optimal behavior which is confirmed by the trajectory plots in chapter 5. Essentially, the reward function could be further improved due to its significant impact on the agent's intended behavior as explained in section 4.2.2. It was further deduced in chapter 5, that the current reward function used in this thesis suffers from curse of goal specification. A proposed way to counter such problem is by reducing the action space even further. This could be done by, for example, only defining relative yaw in the action space, or with only Vx and Vy comprising the action space. Subsequently, a smaller action space would render the exploration-exploitation issue a less significant concern, and hopefully result in a rewarding pattern that would be less complicated for the agent to grasp.

Another approach that is yet to be attempted, is to tune the hyperparameters, and examine the subsequent training approaches. Especially is the exploration-exploitation dilemma important to keep track of. In this case, a training approach with increased exploration would be desirable as it would increase the chance of finding a global optima for the policy.

While the simulator defines a steady-state aruco marker in the environment, a more tracking-specific training environment for the ROV could be a slightly moving aruco marker. This would be more inline with the real-world situation as the aruco marker is continuously in motion, resulting a harder training objective, but more robust tracking properties.

In section 4.2 it was explained that the action space was discretized into 30 different

action commands. A further improvement here could be to maintain a continuous action space. This would, however, increase the action space significantly, but also increase the much needed maneuverability for the ROV. Consequently, unlike in this thesis, there is less likelihood of ending up with a policy solution that only utilizes only two predefined action commands. Hence, a foreseeable ROV behavior in the simulator performance test would not be limited to a L-shaped movement like in the case in this thesis.

In this case, further work on the ROV's design itself would also be reasonable to conduct Especially, the initial overshoot in pitch as a result of sudden acceleration. This could be remedied by either experimenting with various weight placement on the ROV, or by implementing an algorithm such that the acceleration gradually increases to its desired level. Consequently, it will reduce the real-world challenge of pitch overshoot, and ideally render a stable planar environment for the ROV to operate a tracking task without loosing the sight of the aruco marker.

While RL is a powerful tool which has the potential of outperforming human expertise, it is however, a time demanding endeavor to successfully design a working model. In the case of underwater tracking behavior, SL is worth exploring. With the DNN as a basis, SL could be an equally powerful tool which has the human supervision as groundwork during its training phase. Other vehicular systems, such as Google's self-driving cars [18], have achieved successful results. Consequently, it should be fully possible to render a tracking system for underwater vehicles in the same manner.

Bibliography

[1] Abbeel, P., Coates, A., Quigley, M. and Ng, A. Y. [2007], An application of reinforcement learning to aerobatic helicopter flight, *in* 'Advances in neural information processing systems', pp. 1–8.

[2] Ahmadzadeh, S. R., Kormushev, P. and Caldwell, D. G. [2013], Autonomous robotic valve turning: A hierarchical learning approach, *in* 'Robotics and Automation (ICRA), 2013 IEEE International Conference on', IEEE, pp. 4629–4634.

[3] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P. and Zaremba, W. [2017], Hindsight experience replay, *in* 'Advances in Neural Information Processing Systems', pp. 5053–5063.

[4] Bagnell, J., Chestnutt, J., Bradley, D. M. and Ratliff, N. D. [2007], Boosting structured prediction for imitation learning, *in* 'Advances in Neural Information Processing Systems', pp. 1153–1160.

[5] Barto, A. G. and Mahadevan, S. [2003], 'Recent advances in hierarchical reinforcement learning', *Discrete Event Dynamic Systems* **13**(4), 341–379.

[6] Beard, R. W. and McLain, T. W. [2012], *Small unmanned aircraft: Theory and practice*, Princeton university press.

[7] Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. [2007], Greedy layer-wise training of deep networks, *in* 'Advances in neural information processing systems', pp. 153–160.

[8] Bengio, Y. et al. [2009], 'Learning deep architectures for ai', *Foundations and trends® in Machine Learning* **2**(1), 1–127.

[9] Benjamin, M. R., Schmidt, H., Newman, P. M. and Leonard, J. J. [2010], 'Nested autonomy for unmanned marine vehicles with moos-ivp', *Journal of Field Robotics* **27**(6), 834–875.

[10] Bhatnagar, S., Ghavamzadeh, M., Lee, M. and Sutton, R. S. [2008], Incremental natural actor-critic algorithms, *in* J. C. Platt, D. Koller, Y. Singer and S. T. Roweis, eds, 'Advances in Neural Information Processing Systems 20', Curran Associates, Inc., pp. 105–112.

[11] Boyd, S., Parikh, N., Chu, E., Peleato, B. and Eckstein, J. [2011], 'Distributed

optimization and statistical learning via the alternating direction method of multipliers', *Foundations and Trends® in Machine Learning* **3**(1), 1–122.

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

[13] Buesing, L., Weber, T. and Mohamed, S. [2016], 'Stochastic gradient estimation with finite differences'.

[14] Busoniu, L., Babuska, R., De Schutter, B. and Ernst, D. [2010], *Reinforcement learning and dynamic programming using function approximators*, Vol. 39, CRC press.

[15] Calvo, M., Montijano, J. and Randez, L. [1990], 'A fifth-order interpolant for the dormand and prince runge-kutta method', *Journal of computational and applied mathematics* **29**(1), 91–100.

[16] Carreras, M., Yuh, J., Batlle, J. and Ridao, P. [2005], 'A behavior-based scheme using reinforcement learning for autonomous underwater vehicles', *IEEE Journal of Oceanic Engineering* **30**(2), 416–427.

[17] Carreras, M., Yuh, J., Batlle, J. and Ridao, P. [2007], 'Application of sonql for real-time learning of robot behaviors', *Robotics and Autonomous Systems* **55**(8), 628–642.

[18] Cheruvu, R. [n.d.], 'Google self-driving car'.

[19] Clouâtre, L. [2018], 'Octthe16th/ppo-keras'.
URL: *https://github.com/OctThe16th/PPO-Keras*

[20] Cybenko, G. [1989], 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals, and Systems (MCSS)* **2**(4), 303–314.

[21] Deisenroth, M. P., Neumann, G., Peters, J. et al. [2013], 'A survey on policy search for robotics', *Foundations and Trends® in Robotics* **2**(1–2), 1–142.

[22] Delalleau, O. and Bengio, Y. [2011], Shallow vs. deep sum-product networks, *in* 'Advances in Neural Information Processing Systems', pp. 666–674.

[23] Deng, Y., Bao, F., Kong, Y., Ren, Z. and Dai, Q. [2017], 'Deep direct reinforcement learning for financial signal representation and trading', *IEEE transactions on neural networks and learning systems* **28**(3), 653–664.

[24] El-Fakdi, A. and Carreras, M. [2013], 'Two-step gradient-based reinforcement learning for underwater robotics behavior learning', *Robotics and Autonomous Systems* **61**(3), 271–282.

[25] Fossen, T. I. [2011], *Handbook of marine craft hydrodynamics and motion control*, John Wiley & Sons.

[26] Goodfellow, I., Bengio, Y. and Courville, A. [2016], *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

[27] Gu, S., Holly, E., Lillicrap, T. and Levine, S. [2016], 'Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates', *arXiv preprint arXiv:1610.00633* .

[28] Hansen, N., Müller, S. D. and Koumoutsakos, P. [2003], 'Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)', *Evolutionary computation* **11**(1), 1–18.

[29] Heidrich-Meisner, V. and Igel, C. [2009], 'Neuroevolution strategies for episodic reinforcement learning', *Journal of Algorithms* **64**(4), 152–168.

[30] Heidrich-Meisner, V., Lauer, M., Igel, C. and Riedmiller, M. A. [2007], Reinforcement learning in a nutshell., *in* 'ESANN', pp. 277–288.

[31] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D. [2017], 'Deep reinforcement learning that matters', *arXiv preprint arXiv:1709.06560* .

[32] Ijspeert, A. J., Nakanishi, J. and Schaal, S. [2003], Learning attractor landscapes for learning motor primitives, *in* 'Advances in neural information processing systems', pp. 1547–1554.

[33] Ioffe, S. and Szegedy, C. [2015], Batch normalization: Accelerating deep network training by reducing internal covariate shift, *in* 'International Conference on Machine Learning', pp. 448–456.

[34] Jens Kober, J. Andrew Bagnell, J. P. [2013], 'Reinforcement learning in robtics: A survey', *The International Journal of Robotics Research* .

[35] Khosla, S. [2018], 'sahilkhoslaa/augmentedreality-arucoposeestimation'. **URL:** *https://github.com/sahilkhoslaa/AugmentedReality-ArucoPoseEstimation*

[36] Kim, H. J., Jordan, M. I., Sastry, S. and Ng, A. Y. [2004], Autonomous helicopter flight via reinforcement learning, *in* 'Advances in neural information processing systems', pp. 799–806.

[37] Kirk, D. [1970], 'Optimal control theory, englewood clifs'.

[38] Kober, J., Bagnell, J. A. and Peters, J. [2013], 'Reinforcement learning

in robotics: A survey', *The International Journal of Robotics Research* **32**(11), 1238–1274.

[39] Laud, A. D. [2004], Theory and application of reward shaping in reinforcement learning, Technical report.

[40] Li, K. and Malik, J. [2016], 'Learning to optimize', *arXiv preprint arXiv:1606.01885* .

[41] Li, Y. [2017], 'Deep reinforcement learning: An overview', *arXiv preprint arXiv:1701.07274* .

[42] Liang, S. and Srikant, R. [2016], 'Why deep neural networks?', *arXiv preprint arXiv:1610.04161* .

[43] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. [2015], 'Continuous control with deep reinforcement learning', *arXiv preprint arXiv:1509.02971* .

[44] Lin, L.-J. [1993], Reinforcement learning for robots using neural networks, Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

[45] Lin, L., Xie, H., Zhang, D. and Shen, L. [2010], 'Supervised neural q_learning based motion control for bionic underwater robots', *Journal of Bionic Engineering* **7**, S177–S184.

[46] Mayne, D. Q., Seron, M. M. and Raković, S. [2005], 'Robust model predictive control of constrained linear systems with bounded disturbances', *Automatica* **41**(2), 219–224.

[47] Miller, W. T., Glanz, F. H. and Kraft, L. G. [1990], 'Cmas: An associative neural network alternative to backpropagation', *Proceedings of the IEEE* **78**(10), 1561–1567.

[48] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K. [2016], Asynchronous methods for deep reinforcement learning, *in* 'International Conference on Machine Learning', pp. 1928–1937.

[49] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. [2013], 'Playing atari with deep reinforcement learning', *arXiv preprint arXiv:1312.5602* .

[50] Myrhaug, D. [2006], Oceanography: wind, waves, vol, Technical report, UK-2006-78. Institutt for marin teknikk, Trondheim.

[51] Nagabandi, A., Kahn, G., Fearing, R. S. and Levine, S. [2017], 'Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning', *arXiv preprint arXiv:1708.02596* .

[52] Ng, A. Y. [2003], Shaping and policy search in reinforcement learning, PhD thesis, University of California, Berkeley.

[53] Ng, A. Y., Russell, S. J. et al. [2000], Algorithms for inverse reinforcement learning., *in* 'Icml', pp. 663–670.

[54] Perkins, T. J. and Barto, A. G. [2002], 'Lyapunov design for safe reinforcement learning', *Journal of Machine Learning Research* **3**(Dec), 803–832.

[55] Peters, J. and Schaal, S. [2008], 'Reinforcement learning of motor skills with policy gradients', *Neural networks* **21**(4), 682–697.

[56] Russell, S. J. and Norvig, P. [2002], 'Artificial intelligence: a modern approach (international edition)'.

[57] Sandøy, S. S. [2016], System identification and state estimation for rov udrone, Master's thesis, NTNU.

[58] Santamaría, J. C., Sutton, R. S. and Ram, A. [1997], 'Experiments with reinforcement learning in problems with continuous state and action spaces', *Adaptive behavior* **6**(2), 163–217.

[59] Sardag, A. and Akin, H. L. [2006], 'Kalman based finite state controller for partially observable domains', *International Journal of Advanced Robotic Systems* **3**(4), 45.

[60] Schaal, S., Peters, J., Nakanishi, J. and Ijspeert, A. [2005], 'Learning movement primitives', *Robotics Research* pp. 561–572.

[61] Schaul, T., Quan, J., Antonoglou, I. and Silver, D. [2015], 'Prioritized experience replay', *arXiv preprint arXiv:1511.05952* .

[62] Schjølberg, I. and Utne, I. B. [2015], 'Towards autonomy in rov operations', *IFAC-PapersOnLine* **48**(2), 183–188.

[63] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P. [2015], Trust region policy optimization, *in* 'International Conference on Machine Learning', pp. 1889–1897.

[64] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. [2017], 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347* .

[65] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M. [2014], Deterministic policy gradient algorithms, *in* 'Proceedings of the 31st International Conference on Machine Learning (ICML-14)', pp. 387–395.

[66] Strens, M. J. and Moore, A. W. [2002], 'Policy search using paired comparisons', *Journal of Machine Learning Research* **3**(Dec), 921–950.

[67] Sutton, R. S. [1996], Generalization in reinforcement learning: Successful examples using sparse coarse coding, *in* 'Advances in neural information processing systems', pp. 1038–1044.

[68] Sutton, R. S. and Barto, A. G. [1998], *Reinforcement learning: An introduction*, Vol. 1, MIT press Cambridge.

[69] Sutton, R. S. and Barto, A. G. [2012], *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts.

[70] Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C. and Wiewiora, E. [2009], Fast gradient-descent methods for temporal-difference learning with linear function approximation, *in* 'Proceedings of the 26th Annual International Conference on Machine Learning', ACM, pp. 993–1000.

[71] Sze, V., Chen, Y.-H., Yang, T.-J. and Emer, J. [2017], 'Efficient processing of deep neural networks: A tutorial and survey', *arXiv preprint arXiv:1703.09039* .

[72] Theocharous, G., Thomas, P. S. and Ghavamzadeh, M. [2015], Personalized ad recommendation systems for life-time value optimization with guarantees., *in* 'IJCAI', pp. 1806–1812.

[73] Timmer, S. and Riedmiller, M. [2007], Fitted q iteration with cmacs, *in* 'Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on', IEEE, pp. 1–8.

[74] Todorov, E., Erez, T. and Tassa, Y. [2012], Mujoco: A physics engine for model-based control, *in* 'Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on', IEEE, pp. 5026–5033.

[75] Tsitsiklis, J. and Van Roy, B. [1996], An analysis of temporal-difference learning with function approximation, Technical report, Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

[76] Van Hasselt, H. and Wiering, M. A. [2007], Reinforcement learning in continuous action spaces, *in* 'Approximate Dynamic Programming and Reinforce-

ment Learning, 2007. ADPRL 2007. IEEE International Symposium on', IEEE, pp. 272–279.

[77] Watkins, C. J. C. H. [1989], Learning from delayed rewards, PhD thesis, King's College, Cambridge.

[78] Wu, H., Song, S., You, K. and Wu, C. [2017], 'Depth control of model-free auvs via reinforcement learning', *arXiv preprint arXiv:1711.08224* .

[79] Xia, C. and El Kamel, A. [2016], 'Neural inverse reinforcement learning in autonomous navigation', *Robotics and Autonomous Systems* **84**, 1–14.

[80] Yamaguchi, J. and Takanishi, A. [1997], Development of a biped walking robot having antagonistic driven joints using nonlinear spring mechanism, *in* 'Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on', Vol. 1, IEEE, pp. 185–192.

[81] Young, T., Hazarika, D., Poria, S. and Cambria, E. [2017], 'Recent trends in deep learning based natural language processing', *arXiv preprint arXiv:1708.02709* .

[82] Zhang, S. and Sutton, R. S. [2017], 'A deeper look at experience replay', *arXiv preprint arXiv:1712.01275* .

[83] Zhang, T., Kahn, G., Levine, S. and Abbeel, P. [2016], Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search, *in* 'Robotics and Automation (ICRA), 2016 IEEE International Conference on', IEEE, pp. 528–535.

[84] Ziebart, B. D., Maas, A. L., Bagnell, J. A. and Dey, A. K. [2008], Maximum entropy inverse reinforcement learning., *in* 'AAAI', Vol. 8, Chicago, IL, USA, pp. 1433–1438.

[85] Zimmer, M., Boniface, Y. and Dutech, A. [2016], Neural fitted actor-critic, *in* 'European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2016)'.

A Nomenclature

A3C Asynchronous Off-Policy Updates

AC3 Asynchronous Advantage Actor-critic

ACLA Actor-Critic Learning Automation

BODY Body-fixed frame

CACLA Continuous Actor-Critic Learning Automation

CG Conjugate Gradient

CMA-ES Covariance Matrix Adaption - Evolutionary Strategy

CO Center of Origin

DNN Deep Neural Network

DOF Degrees of Freedom

DPG Deterministic Policy Gradient

IMR Inspection Repair Maintenance

IMU Inertial Measurement Unit

MDP Markov Decision Process

ML Machine Learning

MOOS Mission Oriented Operating Suite

MPC Model Predictive Control

NED North-East-Down frame

NFAC Neural Fitted Actor-Critic

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

PPO Proximal Policy Optimization

PWM Pulse Width Modulation

QMT Qualisys Motion Tracking

ReLu Rectified Linear Unit

RL Reinforcement Learning

RPI3 Raspberry Pi 3

SGD Stochastic Gradient Method

SL Supervised Learning

TRPO Trust Region Policy Optimization

UAV Unmanned Aerial Vehicle

B BlueROV2 hydrodynamic parameters

B.1 Mass matrices

B.1.1 Rigid body mass matrix

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} = \begin{bmatrix} 7.31 & 0 & 0 \\ 0 & 7.31 & 0 \\ 0 & 0 & 0.16 \end{bmatrix} \tag{B.1}$$

B.1.2 Added mass matrix

$$\boldsymbol{M}_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 \\ 0 & 0 & N_{\dot{r}} \end{bmatrix} = \begin{bmatrix} 2.6 & 0 & 0 \\ 0 & 18.5 & 0 \\ 0 & 0 & 0.28 \end{bmatrix} \tag{B.2}$$

B.2 Coriolis matrices

B.2.1 Coriolis rigid body matrix

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -7.31v \\ 0 & 0 & 7.31u \\ 7.31v & -7.31u & 0 \end{bmatrix} \tag{B.3}$$

B.2.2 Coriolis added mass matrix

$$\boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 18.5v \\ 0 & 0 & -2.6u \\ -18.5v & 2.6u & 0 \end{bmatrix} \tag{B.4}$$

B.3 Damping matrices

B.3.1 Linear damping matrix

$$\boldsymbol{D}_L = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & 0 \\ 0 & 0 & N_r \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.26 & 0 \\ 0 & 0 & 4.64 \end{bmatrix} \tag{B.5}$$

B.3.2 Quadratic damping matrix

$$\boldsymbol{D}_Q = \begin{bmatrix} X_{u|u|} & 0 & 0 \\ 0 & Y_{v|v|} & 0 \\ 0 & 0 & N_{r|r|} \end{bmatrix} |\boldsymbol{\nu}| = \begin{bmatrix} 34.96 & 0 & 0 \\ 0 & 103.25 & 0 \\ 0 & 0 & 0.43 \end{bmatrix} |\boldsymbol{\nu}| \tag{B.6}$$

Where $\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}$.