Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Simen Theie Havenstrøm

From Beginner to Expert

Deep Reinforcement Learning Controller for 3D Path Following and Collision Avoidance by Autonomous Underwater Vehicles

Master's thesis in Cybernetics and Robotics

Supervisor: Adil Rasheed

May 2020

NTNU
Kunnskap for en bedre verden

Simen Theie Havenstrøm

From Beginner to Expert

Deep Reinforcement Learning Controller for 3D Path Following and Collision Avoidance by Autonomous Underwater Vehicles

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
May 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

This master thesis is written during the spring semester of 2020 to conclude a one-year project studying the use of deep reinforcement learning applied in motion control systems for an autonomous underwater vehicle with six degrees-of-freedom. Last semester (fall 2019), a preproject was undertaken to research and explore for feasible approaches when applying current state-of-the-art learning algorithms to solve the 3D path following problem. The project was also the entry point for my practical experience with such algorithms and way of programming. Specifically, an AUV simulation model was built and standardized to fit the **OpenAI** interface using Python as the programming language. The OpenAI library is a standard toolkit used in reinforcement learning research world-wide.

During the preproject, two distinct methods were used to solve 3D path following: End-to-end learning and a novel approach called PID-assistance. In short, the first approach lets the autonomous agent take complete control of the AUVs actuators as it learns, while the latter lets it learn to operate one actuator at a time by offering PID-assistance in the others while training. Obtaining satisfactory performance by end-to-end learning was a challenge, and the results can be seen as preliminary at best. However, the PID-assistance approach yielded controllers that tracked a path in 3D with great precision. This encouraging result motivated further research by increasing the complexity of the control objective.

The central goal of this master project is to achieve the dual-objective of 3D path following and collision avoidance by use of deep reinforcement learning controllers. Though a good foundation was laid in the preproject, the PID-assistance approach did not work as intended when the agent was introduced to the added complexity of collision avoidance. The agent did not learn to operate the steering commands in a well-behaved or optimal manner by learning to operate one at a time. A completely different approach, which is covered throughout this thesis, was instead needed. Thus, some code from the preproject could be reused, but most has been revised and fitted to the new objectives, learning method and setup.

The same principle applies to section 2 of this report, which covers the preliminary theory. Although machine learning does evolve rapidly, the fundamental principles remains the same. In addition, the models being simulated are unchanged and are based on the same first principles. Naturally, the overlapping theory is reiterated in this report, although much is revised and improved.

Acknowledgements

I would like to thank my supervisor Adil Rasheed and his PhD. candidate Haakon Robinson for guidance and support during the preproject and the master thesis. I credit Camilla Sterud for letting me use her code as inspiration and start-up help during the preproject. I would also like to thank my peers in Adil's student group for fruitful discussions and suggestions. All help was greatly appreciated.

I am also greatly thankful to the two reviewers who reviewed the article based on my preproject work which formed the basis of the continued work in the masters and resulted in another article submitted to a reputed journal.

Contents

List of Figures

List of Tables

Nomenclature

3D	Three-Dimensional
6-DOF	Six Degrees-Of-Freedom
AUV	Autonomous Underwater Vehicle
CB	Center of Buoyancy
CM	Center of Mass
CO	Center of Control
COLAV	Collision Avoidance
CPU	Central Processing Unit
DDPG	Deep Deterministic Policy Gradients
DRL	Deep Reinforcement Learning
FLS	Forward Looking Sonar
FPS	Frames Per Second
GAE	General Advantage Estimation
GNC	Guidance, Navigation and Control
GPU	Graphics Processing Unit
LOS	Line-Of-Sight
MDP	Markov Decision Process
MLP	Multilayer Perceptron
NED	North-East-Down
ODE	Ordinary Differential Equation
PID	Proportional Integral Derivative
PPO	Policy Proximal Optimization
QPMI	Qudratic Polynomial Interpolation
RAM	Rapid Access Memory
RL	Reinforcement Learning
SNAME	Society of Naval Architectures and Marine Engineers

Abstract

Traditional control theory has many to tools to offer the control engineer when faced with a wide array of dynamical systems. However, as complexity of systems grow, providing reliable mathematical representations gets more involved - possibly even infeasible. In these contexts decision-making becomes non-trivial and many of the traditional methods can not be applied. If there is no way to explicitly encode desired behaviour, then how can one hope to construct a useful control law? The framework of reinforcement learning has the potential to break this deadlock, and through experience based learning the need for explicit representations of the environment is discarded.

In this thesis, such learning controllers are developed to operate the control fins of a simulated autonomous underwater vehicle with 6 degrees-of-freedom. The control objective is for the vehicle to follow a predefined 3D path while being engaged in a hydrodynamic environment containing environmental disturbances and unforeseen obstacles intersecting the path. There is obviously many ways to operate in this environment, and for this reason the agents developed are learning by different incentives to observe the differential in behavioural outcome.

The controllers, or agents, are trained by following a learning paradigm known as curriculum learning: That is the idea of progressively exposing the agents to more complex tasks, instead of the sampled environments being completely random. Thus, there is a natural progression from beginner to expert. After training, the expert level agents are deployed in test simulations showing impressive results both in path following and in collision avoidance. Under ideal conditions (no disturbance), the best controller managed to obtain a collision rate of 0%, while still balancing the objective of path following impressively.

In a larger context, the idea of applying learning controllers to emulate human-like decision-making can be seen as a preliminary step towards reaching fully autonomous vehicles. The work presented in this report builds on a preproject and earlier earlier work with the same control objectives, albeit in 2D and with 3 degrees-of-freedom.

Sammendrag

Tradisjonelle kybernetiske metoder har mange verktøy og teknikker som kan anvendes for en rekke klasser dynamiske systemer. En forutsetning for å kunne anvende mange av de tradisjonelle metodene, er en pålitelig matematisk representasjon av systemet/miljøet man ønsker å manipulere. Med økt kompleksitet, til den grad at valg og vurderinger ikke lenger følger trivielle regler, kan det bli vanskelig å finne slike representasjoner - kanskje til og med umulig. Å konstruere lover for tilbakekoblede kontrollsystemer i slike tilfeller, kan derfor vise seg å være utfordrende. Forsterkende læring danner kontrollover basert på erfaring og belønning, og viser seg dermed som et potensielt godt verktøy der det er vanskelig å representere systemet eller ønsket oppførsel eksplisitt.

I denne oppgaven benyttes kontrollere basert på forsterkende læring til å styre et simulert autonomt undervannskjøretøy med 6 frihetsgrader. Objektivet er at kjøretøyet skal følge en forhåndsdefinert sti i 3D, samtidig som den er utsatt for hydrodynamiske forstyrrelser og obstruksjoner som kan forårsake kollisjoner hvis stien følges ukritisk. Da kjøretøy med 6 frihetsgrader og et 3D miljø tilbyr mange måter å operere kjøretøyet på i en slik kontekst, er de autonome agentene trent med forskjellig belønningsstrategi for å observere utfallet i den lærte kontrollstrategien.

Kontrollerene, eller agentene, følger et opplæringsregimet som kalles *pensumlæring* ("Curriculum learning"). Dette bygger på at agentene gradvis utsettes for vanskeligere oppgaver og følgelig økt kompleksitet, istedenfor at oppgaver introduseres helt tilfeldig. Det er dermed en naturlig progresjon fra nybegynner til ekspert når det kommer til å kunne operere kjøretøyet i det nevnte miljøet. Etter trening viste ekspert-pilotene imponerende resultater i både stifølging og kollisjonsunngåelse. Under ideelle forhold (ingen forstyrrelser) oppnådde den beste agenten en kollisjonsrate på 0%. I tillegg viste den gode prestasjoner for stifølging.

I det store bildet kan ideen om å bruke selvlærende kontrollsystemer, som etterligner menneskers evne for vurderinger og veivalg, ses som et tidlig skritt mot fullstendig autonome kjøretøy. Arbeidet som presenteres i denne rapporten bygger på et eget forprosjekt, såvel som tidligere arbeid med tilsvarende objektiver i 2D for kjøretøy med 3 frihetsgrader.

1 Introduction

"The rise of machine learning and artificial intelligence has transformed many domains of human endeavour; Business, finance, education, gaming, research and development are some examples of fields that has been impacted more or less by this change. The field of cybernetics is no exception and potentially has a lot to profit from merging with machine learning and vise versa. Particularly interesting is the close connection between reinforcement learning and continuous control, caused by the similarities with the classical feedback control loop. This thesis is dedicated to explore and further investigate this connection. It does this through studying the use of reinforcement learning controllers in practical applications, specifically in vehicle control systems for an autonomous underwater vehicle. Approaching control system design in this manner has shown exciting results in various applications so far, but we have yet only skimmed the surface of its true potential." (Havenstrøm, 2020)

1.1 Motivation and Background

Autonomous underwater vehicles (AUVs) are used in many subsea commercial applications, such as seafloor mapping, inspection of pipelines and subsea structures, ocean exploration, environmental monitoring and various research operations. The wide range of operational contexts implies that truly autonomous vehicles must be able to follow spatial trajectories (path following), avoid collisions along these trajectories (collision avoidance (COLAV)) and maintain a desired velocity profile (velocity control). In addition, AUVs are often underactuated by the fact that they operate with three generalized actuators (propeller, elevation and rudder fins) in six degrees-of-freedom (6-DOF) (Fossen, 2011, ch. 9). This is the configuration considered in the current work.

The complexity that arises when combining the control objectives, a hydrodynamic environment and disturbances, and the physical design with three generalized actuators, spurs an intriguing control challenge for which many scientific literature exist. However, the objectives of path following and collision avoidance are in most research dealt with separately. Furthermore, control systems for marine crafts are traditionally partitioned into guidance, navigation and control (GNC). In brief, guidance handles setpoints and reference/path generation; Navigation does filtering and state estimation based on modeling and sensory data; Lastly, control maps the reference from the guidance system and the feedback from the navigation system to low-level control actuation. Figure 1.1 gives an overview of this cascaded structure and its signal flow. (Fossen, 2011, ch. 1)

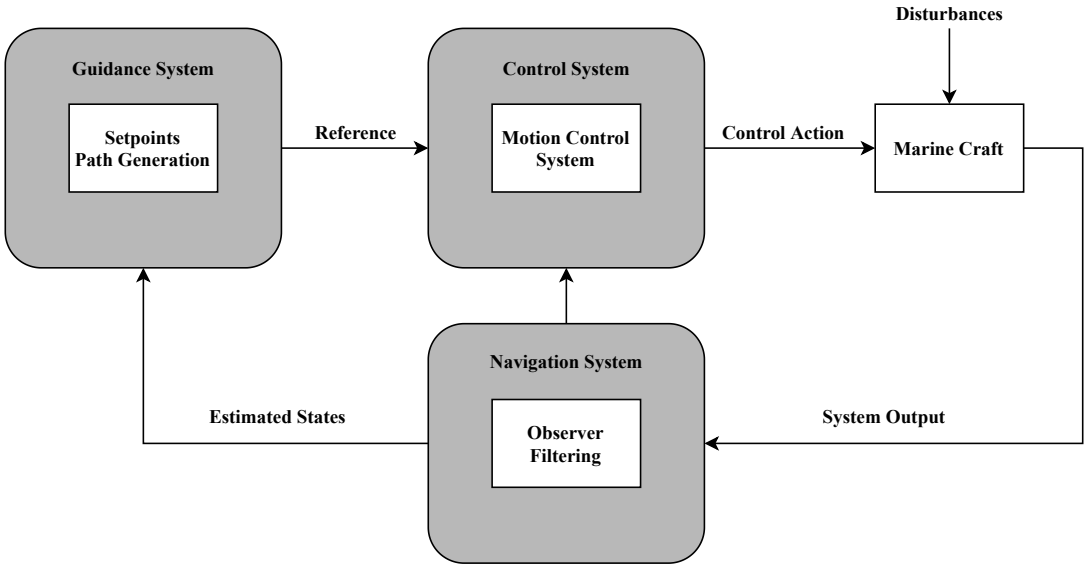


Figure 1.1: Signal flow in guidance, navigation and control systems for marine crafts.

1.1.1 Path Following

The **path following** problem is heavily researched and documented in classical control literature. The control objective is to follow a predefined path, defined relative to some inertial frame, and minimize tracking errors, i.e. the distance between the vehicle and the path. Three-dimensional (3D) path following involves tracking errors that are composed of horizontal and vertical components, and forms an accurate representation of real engineering operations for AUVs (Chu and Zhu, 2015). Typically, a variant of the Proportional Integral Derivative (PID) controller based on reduced order models is used to control elevator and rudder to eliminate tracking errors (Fossen, 2011, ch. 12).

More advanced approaches are also available; A classical nonlinear approach is found in Encarnacao and Pascoal (2000), where a kinematic controller was designed based on Lyapunov theory and integrator backstepping. To extend the nonlinear approach reliably to the presence of disturbances and parametric uncertainties, Chu and Zhu (2015) proposed using an adaptive sliding mode controller, where an adaptive control law is implemented using a radial basis function neural network. To alleviate chattering, a well-known "zig-zag" phenomenon occurring when implementing sliding mode controllers due to a finite sampling time, an adaptation rate was selected based on a so-called minimum disturbance estimate. Xiang et al. (2017) proposed fuzzy logic for adaptive tuning of a feedback linearization PID controller. The heuristic, adaptive scheme accounts for modelling errors and time-varying disturbances. They also compare the performance on 3D path following with conventional PID and non-adaptive backstepping-based controllers, both tuned with inaccurate and accurate model

parameters, to demonstrate the robust performance of the suggested controller. Liang et al. (2018) suggested using fuzzy backstepping sliding mode control to tackle the control problem. Here, the fuzzy logic was used to approximate terms for the nonlinear uncertainties and disturbances, specifically for use in the update laws for the controller design parameters. Many other methods exist, but most published work on the 3D path following problem incorporates either fuzzy logic, variants of PID control, backstepping techniques or any combination thereof.

More recently, there have been numerous attempts to achieve path following and motion control for AUVs by applying machine learning directly to low-level control. Specifically, deep reinforcement learning (DRL) seems to be a favored approach. DRL controllers are based on experience gained from self-play or exploration, using algorithms that can learn to execute tasks by reinforcing good actions based on a performance metric. Preliminary theory on DRL is presented in subsection 2.1.

Yu et al. (2017) used a DRL algorithm known as Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) to obtain a controller that outperformed PID on trajectory tracking for AUVs. A DRL Controller for underactated marine vessels was implemented in Martinsen and Lekkas (2018*b*) to achieve path following for straight-line paths, and later in Martinsen and Lekkas (2018*a*) for curved paths using transfer learning from the first study. The DRL controller demonstrated excellent performance, even compared to traditional line-of-sight (LOS) guidance. Exciting results validating the real-world applications of DRL controllers for AUVs and unmanned surface vehicles is found in Carlucho et al. (2018) and Woo et al. (2019). The first paper implemented the controller on an AUV equipped with six thrusters configured to generate actuation in pitch moment, yaw moment and surge force. They demonstrated velocity control in both linear and angular velocities. The latter paper implemented a DRL controller on an unmanned surface vehicle with path following as the control objective, and presented impressive experimental results from the full-scale test.

Common for the aforementioned work published on path following using DRL controllers is

- only horizontal motion, i.e. the 2D path following problem, has been considered, and
- all used DDPG as the learning algorithm.

The motivation lies thus in the fact that using DRL controllers to solve the 3D path following problem is unexplored territory. In addition, the state-of-the-art DRL algorithm Policy Proximal Optimization (PPO) is used to tackle

the dual-objective of path following and COLAV. Subsequently, setting up the simulation environment and training process provides a basis for further work on the combination of PPO and vehicle motion control in 3D. It can also provide insights on the 3D path following problem from a new perspective.

1.1.2 Collision Avoidance

Collision Avoidance (COLAV) systems is an important part of the control systems for all types of autonomous vehicles. AUVs are costly to produce and typically equipped with expensive instruments as well. Needless to say, maximum efforts must be made to ensure safe movement at all times. COLAV systems must be able to do *obstacle detection* using sensor data and information processing, and *obstacle avoidance* by applying steering commands based on detection and avoidance logic. Two fundamental perspectives on COLAV control architectures are described in the literature: *deliberate* and *reactive*. (Tan, 2006)

Deliberate architectures are plan driven and therefore necessitates á priori information about the environment and terrain. It could be integrated as part of the on-board guidance system (McGann et al., 2008), or at an even higher level in the control architecture, such as a waypoint planner (Ataei and Yousefi-Koma, 2015). Popular methods to solve the path planning problem includes A* algorithms (Carroll et al., 1992; Garau et al., 2005), genetic algorithms (Sugihara and Yuh, 1996) and Probabilistic roadmaps(Kavraki et al., 1996; Cashmore et al., 2014). Deliberate methods are computationally expensive, due to information processing about the global environment. However, they are more likely to make the vehicle converge to the objective (Eriksen et al., 2016).

Reactive methods are faster and processes only real-time sensor data to make decisions. In this sense, the reactive methods are considered local and are used when rapid action is required. Examples of reactive methods are the dynamic window approach (Fox et al., 1997; Eriksen et al., 2016), artificial potential fields (Williams et al., 1990) and constant avoidance angle (Wiig et al., 2018). A potential pit-fall with reactive methods, is trapping the vehicle in local minimas (dead-ends) (Eriksen et al., 2016).

To improve on both the deliberate and the reactive approach, a *hybrid* approach is used in practice by combining the strengths of both. Such architectures are comprised of a deliberate, reactive and execution layer. The deliberate layer handles high level planning, while the reactive layer tackles incidents happening in real-time. The execution layer facilitates the interaction between the deliberate and reactive architectures and decides the final commanded steering. (Tan, 2006)

There are still challenges in state-of-the-art COLAV methods for vehicles subjected to nonholonomic constraints, such as AUVs. Examples of recurring challenges seen in the literature includes

- instability issues,
- neglecting vehicle dynamics and actuator constraints leading to infeasible reference paths, and
- algorithms causing the vehicle to stop.

Additionally, extensive research discusses methods for COLAV in 2D that cannot be directly applied to 3D. In many cases where such methods are adapted to 3D, however, they do not optimally take advantage of the extra dimension (Wiig et al., 2018).

Table 1.1 summarizes state-of-the-art in path following and collision avoidance referenced in the previous sections. It also includes references to the work by Havenstrøm (2020) and Meyer et al. (2020) performed in the specialization projects preceding this master thesis.

Table 1.1: Some of the state-of-the-art research in path following and COLAV.

3D Path Following	
Method	**Reference**
PID control	(Fossen, 2011, ch. 11-12)
Adaptive sliding mode	Chu and Zhu (2015)
Fuzzy feedback linearization	Xiang et al. (2017)
Fuzzy backstepping sliding mode	Liang et al. (2018)
DRL using DDPG algorithm	Martinsen and Lekkas (2018*b*); Martinsen and Lekkas (2018*a*); Yu et al. (2017); Woo et al. (2019)
PID-assisted DRL using PPO	Havenstrøm (2020)
End-to-end DRL using PPO (2D)	Meyer et al. (2020)
Collision Avoidance	
A* path planning	Carroll et al. (1992)
Genetic algorithms	Sugihara and Yuh (1996)
Probabilistic roadmaps	Kavraki et al. (1996)
Dynamic window	Fox et al. (1997); Eriksen et al. (2016)
Artificial potential fields	Williams et al. (1990)
Constant avoidance angle	Wiig et al. (2018)
End-to-end DRL using PPO	Meyer et al. (2020)

1.2 Research Goals and Methods

Motivated by the previous sections, a trinity of interesting perspectives on the research is formed. From a computer science perspective, exploring DRL and the application thereof is a research branch that is expanding fast, and uncovering the limitations, possibilities and what problems this architecture can be used for are of high scientific value. One reason is because it is arguably the most promising form of machine learning not requiring direct supervision. From the cybernetic viewpoint, DRL used on continuous control problems are gaining momentum because of its resemblance to the traditional control loop and its adaptive nature. Lastly, from a marine engineering outlook, the suggested solution to the hybrid control objective of path following and COLAV is new and differs fundamentally from the traditional methods.

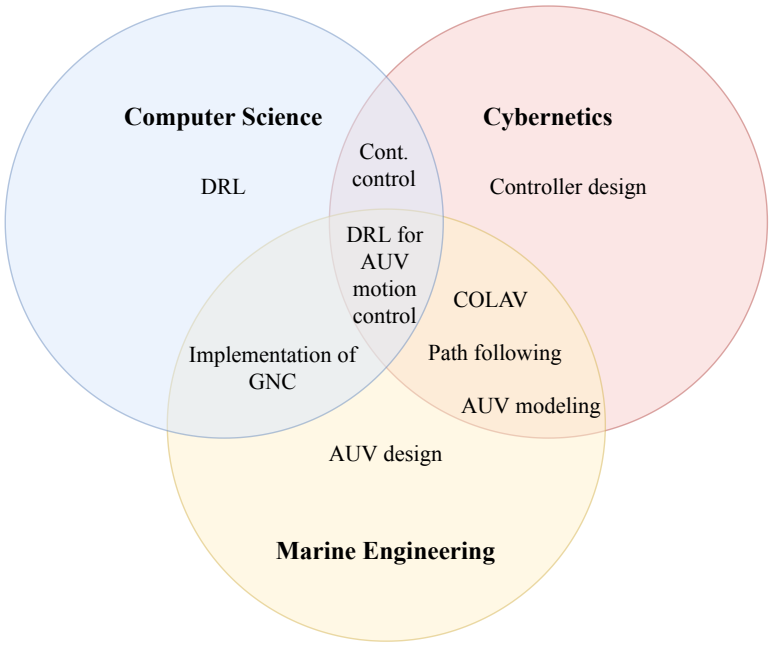


Figure 1.2: The research intersects the three engineering disciplines computer science, cybernetics and marine engineering.

In this research, we attempt to achieve the control objectives by employing a DRL controller as the motion control system in the GNC paradigm, as seen in Figure 1.3. The level of complexity of the control problem suggests that using an intelligent controller, such as a DRL agent that can learn a control law by receiving feedback through observations and modify its behaviour as to optimize a reward signal, is a viable approach.

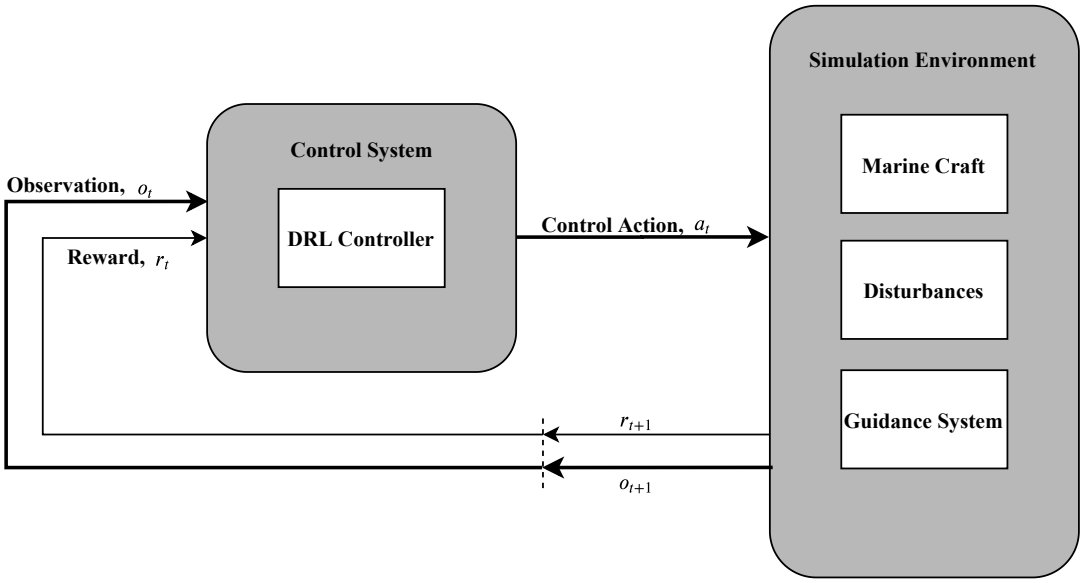


Figure 1.3: The suggested setup for AUV control using DRL in the control system.

In addition to setting up a DRL environment where learning happens through exploration and feedback through observations and a reward signal, the learning strategy known as *curriculum learning* is employed: That is the formalization of learning by being gradually and systematically exposed to more complex environments or tasks(Bengio et al., 2009). As the control objectives can be described in terms of environmental complexity, such as the density of obstacles blocking the path or the intensity of an external disturbance, it is a logical approach in this context. For instance, a scenario for testing pure path following and no disturbance can be described as a combined path following and COLAV scenario containing no obstacles and a current with zero intensity. This is just a semantic difference, but it advocates for a natural way of progressing in terms of complexity. Note that any arbitrary scenario configuring the path and obstacles can be generated, so another key component in the research is designing meaningful configurations in a practical sense. If this is achieved, any agent that has been training in simulation could in theory be uploaded to a physical unit and finish its learning in a full-scale test environment. The implementation of this framework is detailed in subsection 3.3.

In the sense of COLAV, the predefined path can be viewed as the deliberate architecture, where it is assumed that the waypoints are generated by some path planning scheme, and the random and unforeseen obstacles are placed on this presumed collision-free path. The DRL agent operate in effect as the reactive system that must handle the threat of collisions rapidly, but at the same time must chose effective trajectories to minimize tracking deviations.

To the best of our knowledge there is currently no published work on the application of DRL control on the 3D path following problem by an AUV with 6-DOF. To this end, the guiding questions governing the research can be stated as:

- Can the current state-of-the-art in DRL control be applied in end-to-end learning to achieve 3D path following by an AUV with 6-DOF?
- Can the control system build in automatic collision avoidance and achieve intelligent decision-making regarding avoidance maneuvering?
- How does the reward function affect the learned control strategy and is there a clear link to the incentives provided?

1.3 Outline of Report

The thesis comprises of the following sections and content: section 2 covers the preliminary theory forming the foundation for the methods and techniques used in the research; section 3 dissect the concrete methods and the application thereof in implementing the environment, training the DRL controllers and evaluating performance; section 4 presents the report from training, while section 5 covers the experimental results; Lastly, the results and the approach are discussed both concretely and in the wider cybernetic picture in section 6, and the thesis concluded in section 7.

2 Theory

The background theory governing the research and its areas of interest are introduced in this section. As most of the work on DRL and modelling is linked to the preproject, there is naturally much overlapping content. However, there has been significant improvements on the approach during the master project, which merits some additions to the background theory. These upgrades includes generating a curvature continuous path, as opposed to a linear piece-wise path, implementing control fin dynamics and presenting the ocean current simulation model. The fundamental building blocks of the preproject and master thesis is found in subsection 2.1, which introduces the key ideas and terminology from DRL; subsection 2.2 introduces the equations of motion for the AUV model; lastly, theory on path following is presented in subsection 2.3.

2.1 Deep Reinforcement Learning

Training machines to execute tasks via reinforcement learning (RL) is not a new field of research. In fact, RL techniques used in learning control systems was seen as early as 1965 (Waltz and Fu, 1965). Sutton and Barto (2018) traces one facet of the RL origin story back to optimal control and dynamic programming - demonstrating the deep-rooted ties between RL and cybernetics.

Dynamic programming (and other earlier solution methods to the RL problem) suffers from what is known as the "curse of dimensionality", meaning that the computational resources required to solve a problem grows exponentially with the number of state-variables. It would then seem that the classical RL methods had a natural ceiling to them. However, recent advancement in deep neural networks have yielded incredibly powerful function approximators that learns from large quantities of high-dimensional data, eviscerating the early limitations of RL. (Sutton and Barto, 2018, ch. 1)

This goes to show that the general learning principles of RL was not futile, but a key catalyst was missing. Merging together with deep neural networks to form what is now known as deep reinforcement learning was that catalyst. Combined with the computational power of today's hardware, it is now tractable to train and implement DRL controllers to solve complex control problems - such as playing Atari games or controlling robots (Schulman et al., 2017; Levine and Koltun, 2013). The algorithms used in DRL are also in constant evolution, and according the *MIT Technology Review*, RL are gaining a larger market share of published papers in the category of machine-learning each year (Zender, 2019).

A feature specific to RL, is the exploration versus exploitation trade-off: The agent has to exploit what it has learned about the environment and how to interact with it to increase rewards, but to come about better actions it must

first explore its action space. Learning by reinforcing good actions is thus synonymous with how humans (and animals) learn; RL is the the formalization of trial-and-error learning.

2.1.1 Terminology and Notation

Some important concepts and RL specific definitions to know before examining the theory includes (Sutton and Barto, 2018, ch. 1):

- **Agent:** The agent is the decision maker, analogous to the controller in control theory. The agent in this research is the pilot of the AUV, commanding the control fins for elevation and course.
- **Environment:** The environment is the world in which the agent operates, which includes a set of possible states $\mathcal{S}$, a well-defined set of possible actions $\mathcal{A}$ and a model $\rho(s_{t+1}|s_t, a_t)$ governing the transitions from one state to the next. Specific to this project, the environment incorporates a hydrodynamic model for an AUV, disturbances, obstacles and control objectives, which in sum defines this transition model.
- **Observation:** The agent makes an observation at time-step t, o_t, of the environment's state variables, s_t, drawn from $\mathcal{S}$. If the process is fully observable, which in general is not necessary, we get that $o_t = s_t$. In this work it is assumed that the environment is fully observable.
- **Action:** Based on an observation of the environment, the agent performs an action at time-step t, a_t. The actions are the control commands for the AUV actuators, and is drawn from $\mathcal{A}$.
- **Policy:** A policy, $\pi_{\boldsymbol{\theta}}$, maps the agent's observations to actions, $\pi_{\boldsymbol{\theta}} : \mathcal{S} \rightarrow \mathcal{A}$. In DRL, this mapping manifests as a forward pass through a neural network parameterised by the weights and biases $\boldsymbol{\theta}$. In practice, the policy represents the control law and our goal is to find the optimal $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ for our objectives.
- **Reward:** Every state has an intrinsic value associated with it known as a reward, representing how valuable it is to be in that state. The reward signal can also be a function of what action was taken, so in general we write the reward as $r_t = r(s_t, a_t)$.

2.1.2 The RL Goal

Reinforcement learning is the study of intelligence where an agent learns by interacting with an environment. The aim of this interaction is for the agent

to achieve an objective through taking actions. The actions chosen has consequences on the environment, and the environment gives back and observation and a reward signal associated with the current state and action taken. Figure 1.3 shows the classical RL schematic adapted to the current setting of AUV motion control.

Expressing RL as an environment containing a state space, an action space, a transition model and rewards, leads to a formalization traditionally known as Markov decision processes (MDPs) (Puterman, 2014). The study of MDPs are derived from Markov chains - that is a sequential process for which the next state only depends on the current state ("the future is independent of the past given the present") - and forms the theoretical framework for RL. MDPs formalize an optimal control problem where the goal is to maximize accumulated rewards. Special to RL, however, is that the agent does not need to know anything about the underlying transition model to solve this optimization problem. The challenge of the designer is to pose the optimization problem to capture the goals and purposes of the system in the form of a reward signal, incentivizing the agent to learn a policy achieving these goals and purposes. This is called reward function design and is about finding a function $r_t(s_t, a_t)$ befitting the problem at hand.

Some RL tasks are episodic and lasts $T < \infty$ timesteps, and the accumulated future reward in one episode is known as the **return**, R_t^γ. The expression for the return is shown in Equation 2.1. Here, γ is a discount rate weighting the importance of immediate versus long-term rewards. How to select the discount rate is not obvious and is typically part of a tuning process included in every RL project. (Sutton and Barto, 2018, ch. 3)

$$R_t^\gamma = \sum_{k=t}^{T} \gamma^{t-k} r(s_k, a_k),\ 0 < \gamma < 1 \tag{2.1}$$

From the definition of the return we obtain the **value** function which is the expected return by following the policy π starting from state s_t, and is written as $V^\pi(s_t) = \mathbb{E}\{R_t^\gamma | s_t; \pi\}$. We also obtain a similar expression known as the **Q-function**, which is the expected return by following the policy starting from state s_t, but in addition taking initial action a_t. The Q-function is also known as the state-action value-function and is written $Q^\pi(s_t, a_t) = \mathbb{E}\{R_t^\gamma | s_t, a_t; \pi\}$. Note that taking the expectation of the Q-function yields the value function (since in expectation we would follow the most likely action initially, which is identically the definition of the value function).

A potential difference between the value function and Q-function can occur based on what action is taken. This measure carries an intuitive meaning, namely the advantage of choosing that action as opposed to following the policy, and is for this reason called the **advantage** function. The advantage function is expressed in Equation 2.2.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.2}$$

The goal of the agent is to maximize accumulated future rewards by finding an optimal policy. It has to infer this from experience, and learns by continual error-correction. Based on the definition of return and policy, the RL goal can be formally stated as the optimization problem in Equation 2.3:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\rho, a\sim\pi}\left[R_t^{\gamma}\right] \tag{2.3}$$

where the actions are drawn from the policy π and the states follow ρ. In DRL, the policy is explicitly represented in terms of a deep neural network parameterized by the weights and biases $\boldsymbol{\theta}$. In this case the RL goal is finding the optimal parameters $\boldsymbol{\theta}^*$ to represent the policy.

2.1.3 Solution Methods

In the case of DRL, solving Equation 2.3 yields the optimal parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ that maximize the expected return at all times t. RL algorithms used for solving Equation 2.3 are typically divided into four categories:

- **Value-based methods**: Estimate the value function and/or the Q-function and from that derive a control policy with high probability of taking actions that maximize these functions (Tai et al., 2016). In these methods, an explicit representation of the value-function is needed, but the policy is usually implicitly represented.

- **Policy gradients method**: Policy gradient methods do not make estimates of the value functions, but instead increases the objective function directly by performing gradient ascent (Sutton et al., 1999). The policies are therefore parameterized, for instance as the weights and biases of a deep neural network, and these parameters are iteratively adjusted in the ascent direction to increase performance (Tai et al., 2016). Hence, there is always an explicit representation of a policy in this class of methods.

- **Actor-Critic methods**: A hybrid of policy gradient and value-based methods trying to capture the strengths of both. The method works in two steps: First, the value function is approximated by a *critic* neural network, then the parameters of the policy/actor is updated by taking a small step in the direction suggested by the critic (Yoon, 2019). The methods therefore uses explicit representations for both the value-function and a policy. A concept drawing of Actor-Critic methods is seen in Figure 2.1.

- **Model-based RL**: The aforementioned methods are model-free; They need not to know anything about the underlying world model in order to optimize the objective function. However, one can argue that the underlying transition model is captured implicitly through representing the optimal policy and/or the value-function. Model-based RL instead keep an explicitly representation of the transition model. This estimate can be used for planning or even simulating further time-steps, as well as in optimal control or substitute in value-based methods. Most importantly, the benefits of model-based methods are data efficiency and high adaptability. (Doll et al., 2012; Weber et al., 2017)

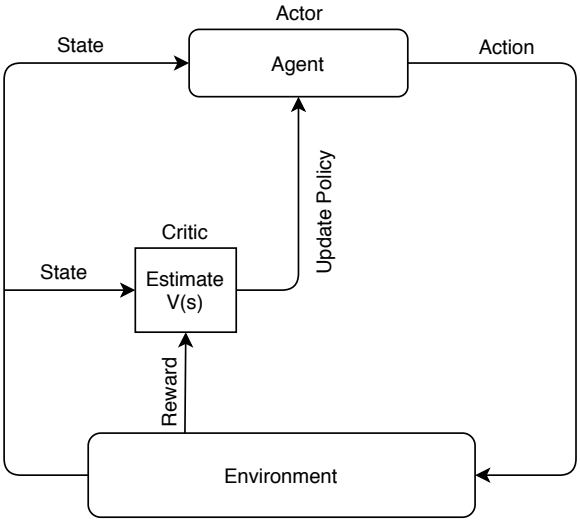


Figure 2.1: A schematic of the actor-critic method. The method is a hybrid of policy gradient and value-based methods. The policy neural network (actor) makes decisions that affect how the environment transitions from one state to the next, and the critic neural network observes the environment and the rewards received to estimate a value function. Based on this estimate, the policy is updated by some policy gradient method in the direction suggested by the critic.

2.1.4 Policy Proximal Optimization

PPO was invented by Schulman et al. (2017) and is a state-of-the-art actor-critic method. It utilizes general advantage estimation (GAE), as proposed in (Schulman, Moritz, Levine, Jordan and Abbeel, 2015), and a novel clipped surrogate objective function. Empirical results show that it outperformed other algorithms on a collection of benchmark tasks - significantly in the ones involving continuous control.

The value function and Q-function is in general unknown. Given by Equation 2.2, the advantage function is calculated from both these functions, and hence is also in general unknown. Therefore, an estimate of the advantage function, $\hat{A}_t^\pi$, is derived based on the value function estimated by the critic neural network, $\hat{V}^\pi(s)$. GAE is one method of estimating A_t, which is shown in Equation 2.4. (Schulman, Moritz, Levine, Jordan and Abbeel, 2015)

$$\begin{gathered}\hat{A}_t^\pi = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \text{where } \delta_t = r_t + \gamma\hat{V}^\pi(s_{t+1}) - \hat{V}^\pi(s_t)\end{gathered} \tag{2.4}$$

The discount factor remains symbolized by γ, and $\lambda \in [0, 1]$ represents a trade-off parameter between estimator variance and bias, referred to as the GAE parameter. The term δ_t is in RL known as the temporal-difference error (Sutton and Barto, 2018), and when $V^\pi = \hat{V}^\pi$ we have that δ_t is an unbiased estimate of A_t^π. If this were the case, we could set $\lambda = 0$ and obtain a perfect estimate. However, we can not trust this to be the case. The solution is to sum over more estimates over a horizon T to obtain a less biased estimator. Though, increasing the amount of uncertain terms (by setting λ closer to 1) increases in turn the variance of the estimate. Because the bias when using few δ_t terms are significant, the GAE parameter is usually set close to 1.

The second key component of PPO is the novel objective function. This objective is a surrogate for the true objective, meaning that increasing the surrogate in a local neighborhood - a so-called trust region - will ultimately increase the true objective function. More formally, we can define the DRL objective function as a function of the neural network weights and biases $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) \coloneqq \mathbb{E}_{s\sim\rho_{\boldsymbol{\theta}}, a\sim\pi_{\boldsymbol{\theta}}}\left[R_t^\gamma\right] \tag{2.5}$$

The objective function can be increased directly through gradient ascent, i.e. by a policy gradient method, yielding the update scheme $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$. One method of calculating $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$, which is a stochastic estimate of the policy gradient, is basing it on the advantage function estimate $\hat{A}_t^\pi$.

PPO improves the policy through so-called *conservative policy iteration*. Let $g_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t|s_t)}$ express the *probability ratio* between an old policy and an updated one. Trust region based methods are motivated by updating the policy such that g_t stays small and the approximation is valid in a local neighborhood. Trust region policy optimization (Schulman, Levine, Moritz, Jordan and Abbeel, 2015) used a constraint on the KL divergence (a measure for how one probability distribution differs from another) to limit the update, where its successor PPO uses a clipped objective function seen in Equation 2.6.

$$L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\left[\min\left(g_t(\boldsymbol{\theta})\hat{A}_t, \textit{clip}\left(g_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2.6}$$

Here, ε is a tuning parameter restricting g_t in each update. During a training iteration, N actors (parallelized agents) are enabled to execute the policy and in that way sample trajectories for T timesteps. The GAE is computed based on the sampled trajectories, then used to optimize the surrogate objective for K epochs using mini-batches of size M per update. The PPO method is given in its most general form in algorithm 1 (Schulman et al., 2017).

Algorithm 1: Proximal Policy Optimization, Actor-Critic style

for *iteration: 1,2...* **do**
  for *actor: 1,2...N* **do**
    Run policy $\pi_{\boldsymbol{\theta}_{old}}$ for T time-steps
    Compute advantage estimate $\hat{A}_1 ... \hat{A}_T$
  end
  Optimize surrogate L w.r.t. $\boldsymbol{\theta}$, with K epochs and mini-batch size $M < NT$
  $\boldsymbol{\theta}_{old} \leftarrow \boldsymbol{\theta}$
end

Choosing PPO to solve the current control problems is based on its reputable performance on a wide range of continuous control problems, indicating its potential benefit to this research as well. According to its creators, it also strikes a balance between simplicity, data efficiency and robustness.

2.2 AUV Modeling

This section introduces a dynamic model that can be used to accurately simulate an AUV in a hydrodynamic environment. This is done by using a 6-DOF maneuvering model which is represented by 12 highly coupled and nonlinear first-order ordinary differential equations (ODEs). Dynamic models for AUVs comprises a *kinematic* (subsubsection 2.2.2) and a *kinetic* (subsubsection 2.2.3) part. Kinematics represents the geometrical evolution of the vehicle and involves a coordinate transformation between two important reference frames. Kinetics considers the forces and moments causing vehicle motion. The kinetic analysis is typically important when designing motion control systems because actuation can only be achieved by applying control forces and moments. Before delving into the details of the kinematic and kinetic equations, the notation used to detail the model's states and parameters is presented in Table 2.1. This notation is used by the Society of Naval Architecths and Marine Engineers (SNAME (1950)). (Fossen, 2011, p. 16)

Table 2.1: Notation for marine vessels used by SNAME (1950).

Degree of freedom	Force/Moment	Velocities	Positions
1 motion in the x direction (surge)	X	u	x
2 motion in the y direction (sway)	Y	v	y
3 motion in the z direction (heave)	Z	w	z
4 rotation about x axis (roll)	K	q	ϕ
5 rotation about y axis (pitch)	M	p	θ
6 rotation about z axis (yaw)	N	r	ψ

2.2.1 Reference Frames

Two reference frames are especially important in modeling of vehicle dynamics: The North-East-Down (NED) frame denoted {n} and the body frame denoted {b}. The NED coordinate system is considered to be inertial, with principal axis pointing towards true north, east and downwards - normal to Earth's surface - for the x_n, y_n, z_n axes, respectively. Since the NED frame is considered inertial, Newton's laws of motion applies. However, it is based on a tangent plane of the Earth, so it is only valid for local navigation (Fossen, 2011, p. 17).

The body frame has its origin located at the vehicle's center of control (CO), which in general is a design choice. The CO is not automatically placed at the vehicle's center of mass (CM) since this point might be time-varying. A typical point for the CO for AUVs is therefore the center of buoyancy (CB). The body frame's x_b axis points along the longitudinal axis of the vehicle, the y_b axis points transversal and the z_b axis points normal to the vehicle surface. The two coordinate systems are pictured in Figure 2.2.

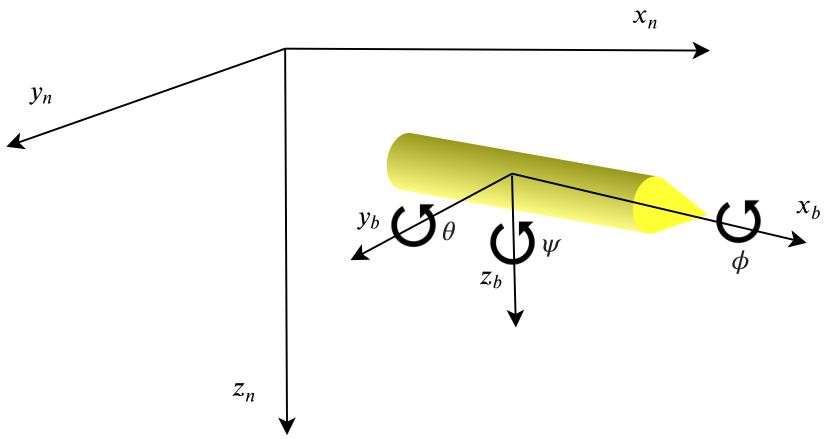


Figure 2.2: Simple illustration of BODY and NED coordinate systems. The BODY frame is obtained by rotating the NED frame about its principal axes.

To relate vectors in different coordinates, we utilize the Euler angle rotation

matrix seen in Equation 2.7.

$$\mathbf{R}_b^n(\mathbf{\Theta}_{nb}) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \text{ [1]} \tag{2.7}$$

The Euler-angles describing the vehicle's attitude is contained in $\mathbf{\Theta}_{nb} = [\phi, \theta, \psi]^T$. To obtain a vector expressed in the body frame in NED coordinates, a matrix multiplication with the rotation matrix is applied. To rotate the inverse way, i.e. from {n} to {b}, we use the transposed rotation matrix $(\mathbf{R}_b^n)^T = \mathbf{R}_n^b$.

2.2.2 Kinematic Equations

The kinematic state vector is the concatenation of the position of the vehicle in NED coordinates and the vehicle's attitude with respect to the NED frame. This vector is symbolized by $\boldsymbol{\eta} = [\mathbf{p}^n, \mathbf{\Theta}_{nb}]^T = [x, y, z, \phi, \theta, \psi]^T$. The velocity vector expressed in {b}, $\mathbf{v}^b$, is utilized to find a differential equation for $\boldsymbol{p}^n$. Rotating this vector by applying Equation 2.7, yield the differential equation for the position in {n}:

$$\dot{\mathbf{p}}^n = \mathbf{v}^n = \mathbf{R}_b^n(\mathbf{\Theta}_{nb})\mathbf{v}^b \tag{2.8}$$

where the body-fixed velocity vector is defined as $\mathbf{v}^b = [u, v, w]^T$ and the components are defined according to Table 2.1.

To write a differential equation for the whole kinematic state vector, an equation describing the time-evolution of the Euler-angles is obtained by transforming the linear velocities expressed in {b}, according to Equation 2.9. Note that this transformation is not well-defined for $\theta = \frac{\pi}{2}$. An alternative representation avoiding the singularity is quaternion parameterization (Fossen, 2011, p. 25).

$$\dot{\mathbf{\Theta}}_{nb} = \mathbf{T}_\Theta(\mathbf{\Theta}_{nb})\boldsymbol{\omega}_{b/n}^b = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \begin{bmatrix} q \\ p \\ r \end{bmatrix} \text{ [2]}, \tag{2.9}$$

Now the complete kinematic differential equation in Equation 2.10 can be written by combining Equation 2.8 and Equation 2.9.

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \dot{\mathbf{p}}^n \\ \dot{\mathbf{\Theta}}_{nb} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_b^n(\mathbf{\Theta}_{nb}) & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_\Theta(\mathbf{\Theta}_{nb}) \end{bmatrix} \begin{bmatrix} \mathbf{v}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix} = \mathbf{J}_\Theta(\boldsymbol{\eta})\boldsymbol{\nu} \tag{2.10}$$

[1] $s\phi = \sin\phi, c\phi = \cos\phi$

[2] $t\theta = \tan\theta$

2.2.3 Kinetic Equations

The Kinetic equations of motion for a marine craft can be expressed as a mass-spring-damper system. The mass terms naturally stems from vessel body, while the spring forces acting on the body arise from buoyancy. The damping is a result of the hydrodynamic forces caused by motion. The model implemented is adapted from da Silva et al. (2007) and all model parameters can be seen in Appendix A. The AUV specifications on which the model parameters is based is given by Table 2.2:

Table 2.2: Specifications for simulated AUV adapted from da Silva et al. (2007).

Symbol	Description	Value	Unit
m	Mass	18	kg
L	Length	108	cm
W	Weight	176	N
B	Buoyancy	177	N
z_G	Position of CM w.r.t. CB in z-axis	1	cm
d	Diameter	15	cm
δ_{max}	Maximum control fin deflection	30°	deg
η_{max}	Maximum propeller thrust	14	N

Furthermore, it is based on the following assumptions:

1. **Assumption 1:** The AUV operates at a depths below disturbances from wind and waves.
2. **Assumption 2:** The maximum speed is $2m/s$.
3. **Assumption 3:** The moment of inertia can be approximated by that of a spheroid.
4. **Assumption 4:** The AUV is passively stabilized in roll and pitch by placing the CM a distance z_G under the CO.
5. **Assumption 5:** The AUV shape is top-bottom and port-starboard symmetric.
6. **Assumption 6:** As a fail-safe mechanism, the AUV is slightly buoyant.

The vessel's motion is governed by the nonlinear kinetic equations expressed in {b} according to Equation 2.11:

$$\underbrace{\mathbf{M}\dot{\boldsymbol{\nu}}_r}_{Mass forces} + \underbrace{\mathbf{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{Coriolis forces} + \underbrace{\mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{Damping forces} + \underbrace{\mathbf{g}(\boldsymbol{\eta})}_{Restoring forces} = \boldsymbol{\tau}_{control} \tag{2.11}$$

where $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$ is the velocity relative to the velocity of an ocean current, represented by $\boldsymbol{\nu}_c$ in {b} . When no currents are present, we see that $\boldsymbol{\nu} = \boldsymbol{\nu}_r$. Furthermore, only irrotational currents are considered.

Mass Forces The systems inertia matrix, M, is the sum of the inertia matrix for the rigid body (RB) and the added mass (A). Added mass is the inertia added from the weight of fluid the vessel displaces when moving through it. Because of the symmetry assumptions, both matrices are diagonal. However, the rigid body matrix is defined in the center of gravity, such that it must be shifted to the center of control, yielding some coupling terms:

$$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 & 0 & mz_G & 0 \\ 0 & m - Y_{\dot{v}} & 0 & -mz_G & 0 & 0 \\ 0 & 0 & m - Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & -mz_G & 0 & I_x - K_{\dot{p}} & 0 & 0 \\ mz_G & 0 & 0 & 0 & I_y - M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z - N_{\dot{r}} \end{bmatrix} \tag{2.12}$$

Coriolis Forces Naturally, the added mass will also effect the Coriolis-centripetal matrix, $\mathbf{C}(\boldsymbol{\nu}_\mathbf{r})$, which defines the forces occurring due to {b} rotating about {n}. Moreover, the linear-velocity independent parameterization of the rigid-body Coriolis-centripetal matrix is utilized, easing the implementation of irrotational ocean currents (Fossen, 2011, p. 222). (Note that there are still linear velocity terms caused by the added mass). It is this trick that makes it possible to collect the rigid-body and added mass terms to represent the 6-DOF model by the elegant Equation 2.11. When using the linear-velocity independent parameterization, the Coriolis-centripetal matrix is written:

$$\mathbf{C}(\boldsymbol{\nu}_r) = \mathbf{C}(\boldsymbol{\nu}_r)_{RB} + \mathbf{C}(\boldsymbol{\nu}_r)_A =$$

$$\begin{bmatrix} 0 & -mr & mq & mz_G r & -Z_{\dot{w}} w_r & Y_{\dot{v}} v_r \\ mr & 0 & -mp & Z_{\dot{w}} w_r & mz_G r & -X_{\dot{u}} u_r \\ -mq & mp & 0 & -(mz_G p + Y_{\dot{v}} v_r) & -mz_G q + X_{\dot{u}} u_r & 0 \\ -mz_G r & -Z_{\dot{w}} w_r & mz_G p + Y_{\dot{v}} v_r & 0 & (I_z - mz_G^2 - N_{\dot{r}}) r & (-I_y + M_{\dot{q}}) q \\ Z_{\dot{w}} w_r & -mz_G r & mz_G q - X_{\dot{u}} u_r & (-I_z + mz_G^2 + N_{\dot{r}}) r & 0 & (I_x - K_{\dot{p}} p \\ -Y_{\dot{v}} v_r & X_{\dot{u}} u_r & 0 & (I_y - M_{\dot{q}}) q & (-I_x + K_{\dot{p}}) p & 0 \end{bmatrix}$$

(2.13)

Damping Forces The components of hydrodynamic damping modelled is linear viscous damping, nonlinear (quadratic) damping due to vortex shedding and lift forces from the body and control fins. Thus, the damping matrix, $\mathbf{D}(\boldsymbol{\nu}_r)$, can be expressed as:

$$\mathbf{D}(\boldsymbol{\nu}_r) = \mathbf{D} + \mathbf{D}_n(\boldsymbol{\nu}_r) + \mathbf{L}(\boldsymbol{\nu}_r) \tag{2.14}$$

The linear damping is given by:

$$\mathbf{D} = -\begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & Y_r \\ 0 & 0 & Z_w & 0 & Z_q & 0 \\ 0 & 0 & 0 & K_p & 0 & 0 \\ 0 & 0 & M_w & 0 & M_q & 0 \\ 0 & N_v & 0 & 0 & 0 & N_r \end{bmatrix}$$

The nonlinear damping is given by:

$$\mathbf{D}_n(\boldsymbol{\nu}) = -\begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & X_{v|v|}|v| & 0 & 0 & 0 & Y_{r|r|}|r| \\ 0 & 0 & Z_{w|w|}|w| & 0 & Z_{q|q|}|q| & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & M_{w|w|}|w| & 0 & M_{q|q|}|q| & 0 \\ 0 & N_{v|v|}|v| & 0 & 0 & 0 & N_{r|r|}|r| \end{bmatrix}$$

Finally, the lift is given by:

$$\mathbf{L}(\boldsymbol{\nu}) = -\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{uv_f} + Y_{uv_b} & 0 & 0 & 0 & Y_{ur_f} \\ 0 & 0 & Z_{uw_f} + Z_{uw_b} & 0 & Z_{uq_f} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & M_{uw_f} + M_{uw_b} & 0 & M_{uq_f} & 0 \\ 0 & N_{uv_f} + N_{uv_b} & 0 & 0 & 0 & N_{ur_f} \end{bmatrix} u$$

Restoring Forces The restoring forces working on the AUV body are functions of the orientation, weight and buoyancy of the vehicle. Because the vehicle is assumed to be slightly buoyant and the passive stabilization of roll and pitch, the restoring force vector can be written as:

$$\mathbf{G}(\boldsymbol{\eta}) = \begin{bmatrix} (W - B)\sin\theta \\ -(W - B)\cos\theta\sin\phi \\ -(W - B)\cos\theta\cos\phi \\ z_G W\cos\theta\sin\phi \\ z_G W\sin\theta \\ 0 \end{bmatrix} \tag{2.15}$$

Control Inputs There are 3 control inputs: propeller thrust, rudder and elevator fins denoted n, δ_r and δ_s, respectively. All actuators are constrained according to Table 2.2. The constraint on the thrust force guarantees that the low-speed assumption holds. The control inputs are related to the control force vector according to Equation 2.16:

$$\boldsymbol{\tau}_{control} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & Y_{uu\delta_r} u_r^2 & 0 \\ 0 & 0 & Z_{uu\delta_s} u_r^2 \\ 0 & 0 & 0 \\ 0 & 0 & M_{uu\delta_s} u_r^2 \\ 0 & N_{uu\delta_r} u_r^2 & 0 \end{bmatrix} \begin{bmatrix} n \\ \delta_r \\ \delta_s \end{bmatrix} \tag{2.16}$$

This completes the details of the model implemented. The numerical values used in the simulation can be found in Appendix A. For a complete derivation of the model and how the numerical values are obtained from the specifications and assumptions, da Silva et al. (2007) and Fossen (2011) are referred to for extensive explanations.

2.2.4 Simulation Model for Ocean Current

To simulate the environmental disturbance in the form of ocean currents, a 3D irrotational ocean current model is implemented. The model is based on generating the intensity of the current, $V_c = \|\boldsymbol{\nu}_c\|_2$, by utilizing a first-order *Gauss-Markov Process* (Fossen, 2011, Ch. 8):

$$\dot{V}_c = -\mu V_c + w \tag{2.17}$$

where w is *white noise* and $\mu \geq 0$ a constant. An integration limit is set so that the current speed is limited between 0.5 to $1\ m/s$. The current direction is static and initialized randomly for each episode. The current direction is described by the sideslip angle and angle of attack are symbolized by α_c and β_c, respectively. These angles represent from what direction the current hits the body frame. In NED coordinates, the linear ocean current velocities can be obtained by Equation 2.18 (Fossen, 2011, Ch. 8).

$$\mathbf{v}_c^n = V_c \begin{bmatrix} \cos\alpha_c \cos\beta_c \\ \sin\beta_c \\ \sin\alpha_c \cos\beta_c \end{bmatrix} \tag{2.18}$$

There are no dynamics associated with the sideslip angle and the angle of attack in the simulations; The current direction stays fixed throughout an episode. To obtain the linear velocities in the body frame, we apply the inverse Euler-angle rotation matrix, as seen in Equation 2.19:

$$\begin{bmatrix} u_c \\ v_c \\ w_c \end{bmatrix} = \mathbf{R}_b^n(\mathbf{\Theta}_{nb})^T \mathbf{v_c^n} \tag{2.19}$$

Since the ocean current is defined to be irrotational, the full current velocity vector is written $\boldsymbol{\nu}_c = [u_c, v_c, w_c, 0, 0, 0]$.

2.2.5 Control Fin Dynamics

To simulate operation of the control fins more realistically, the output of the controller passes through a first-order low-pass filter with time-constant T_f. The intention behind this implementation is to remove noisy outputs from the DRL agent, without having to add a cost to the control action derivatives $\dot{\delta}_r$ and $\dot{\delta}_s$. Ideally, the agent learns that abrupt control action is pointless since high frequency commands are filtered out.

The implementation of the discrete low-pass filter for the control fins is given by Equation 2.20:

$$\delta_{i,t} = (1-a)\delta_{i,t-1} + au_t \quad , \quad i = r \text{ or } s \tag{2.20}$$

where the filter-parameter a is related to the time-constant by $a = \frac{\Delta t}{T_f + \Delta t}$, u_t is the raw control action and Δt is the simulation step-size. (Haugen, 2008)

2.3 3D Path Following

In this section, the path following problem is formally introduced. A 3D path is defined relative to the NED frame and the control problem is independent of time. This means that the vehicle should only progress along the path while minimizing tracking errors, but is not required to be at a specific point at a specific time, as opposed to *trajectory tracking* (Fossen, 2011, ch. 9).

A set of n_w waypoints is used to generate the path, starting at the origin of the NED coordinates for simplicity. In the preproject, the path was generated by linear interpolation between the waypoints, resulting in a linear piece-wise path. The parametric equations for the linear interpolation scheme is seen in Equation 2.21 (Breivik and Fossen, 2009).

$$\begin{aligned} x_{p,m}(s) &= x_{p,m-1} + s\cos\chi_{p,m-1}\cos\upsilon_{p,m-1} \\ y_{p,m}(s) &= y_{p,m-1} + s\sin\chi_{p,m-1}\cos\upsilon_{p,m-1} \\ z_{p,m}(s) &= z_{p,m-1} - s\sin\upsilon_{p,m-1} \end{aligned} \tag{2.21}$$

Subscript p is used to indicate that the coordinate is representing the path and m denotes the waypoint index. $\chi_{p,m-1}$ and $\upsilon_{p,m-1}$ are the azimuth and elevation angle from waypoint $m-1$ to m. The parametric equations are continuously

differentiable in s, which is the along-track distance travelled on the path from waypoint $m-1$ to m.

However, an obvious flaw with the linear interpolation scheme are the abrupt steps in the reference when switching between waypoints; There is a continuity problem because the vehicle cannot move in a discontinuous manner. A proposed solution to this has been smoothing out the vertices, creating paths comprised of circle arcs interpolating between the linear segments connecting waypoints. This approach is known as the *Dubin's path*(Fossen, 2011), but contains another flaw: When switching from linear segments to cirlce arcs, the desired yaw rate/heading rate (r/q) steps. In other words, it is velocity continuous but not curvature continuous. The classes of geometric continuity is referred to as $G^0, G^1, G^2 ... G^n$ in the literature (Chang and Huh, 2015). To elaborate, G^2 continuity means that the path is second-order differentiable and therefore curvature continuous, where as G^0 refers to piece-wise linear paths.

2.3.1 Quadratic Polynomial Interpolation

Any well-defined path for a vehicle that cannot accelerate infinitely fast must be G^2 continuous. Methods such as cubic and spline interpolation establish G^2 continuity and are straightforward to implement in 2D, but cannot be applied directly in 3D interpolation. In fact, some spline-methods has been shown to produce paths that do not visit all waypoints in 3D (Chang and Huh, 2015). This is undesirable and the path should visit all waypoints in the correct sequence.

Chang and Huh (2015) proposed a 3D extension of qudratic polynomial interpolation (QPMI) to create a G^2 continuous path by using second-order polynomials and a membership function to smoothly switch between polynomials. They choose quadratic polynomials because this is the lowest order possible for obtaining G^2 continuity, and higher order polynomials are prone to be corrupted by Runge's phenomenon.

To generate a QPMI path in 3D, we start by writing the path $\mathbf{P}_p$ as a function of the along-track distance, s, such that $\mathbf{P}_p(s) : (x(s), y(s), z(s))$. Each waypoint m has a euclidian distance s_m associated with it. For the first waypoint this distance is zero, i.e. $s_1 = 0$, and the others are obtained by the generalized equation $s_m = \sum_{i=2}^{m} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2}$. The quadratic polynomials linking three waypoints togheter can be written:

$$\begin{aligned} x_m(s) &= a_{x_m} s^2 + b_{x_m} s + c_{x_m} \\ y_m(s) &= a_{y_m} s^2 + b_{y_m} s + c_{y_m} \\ z_m(s) &= a_{z_m} s^2 + b_{z_m} s + c_{z_m} \\ m &= 2, 3, ..., n_w - 1 \end{aligned} \tag{2.22}$$

And the coefficients can be found by solving the following matrix equations:

$$\begin{bmatrix} a_{x_m} \\ b_{x_m} \\ c_{x_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} x(s_{m-1}) \\ x(s_m) \\ x(s_{m+1}) \end{bmatrix} \tag{2.23}$$

$$\begin{bmatrix} a_{y_m} \\ b_{y_m} \\ c_{y_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} y(s_{m-1}) \\ y(s_m) \\ y(s_{m+1}) \end{bmatrix} \tag{2.24}$$

$$\begin{bmatrix} a_{z_m} \\ b_{z_m} \\ c_{z_m} \end{bmatrix} = \begin{bmatrix} u_{m-1}^2 & u_{m-1} & 1 \\ u_m^2 & u_m & 1 \\ u_{m+1}^2 & u_{m+1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} z(s_{m-1}) \\ z(s_m) \\ z(s_{m+1}) \end{bmatrix} \tag{2.25}$$

$$m = 2, 3, ..., n_w - 1$$

A path represented by n_w waypoints require $n_w - 2$ polynomials to generate the QPMI path, as seen in the previous equations. A group of polynomials linking three and three waypoints is therefore obtained. The group of polynomials representing the path is written $\mathbf{P}_p(s) : (X(s), Y(s), Z(s))$, where the groups X, Y, Z is expressed in general form as

$$X(s) = \begin{cases} x_2(s) & s_1 \leq s < s_2 \\ \mu_{r,m}(s)x_{m+1}(s) + \mu_{f,m}(s)x_m(s), (2 \leq m < n_w - 1) & s_2 \leq s < s_{n_w-1} \\ x_{n_w-1}(s) & s_{n_w-1} \leq s \leq s_{n_w} \end{cases} \tag{2.26}$$

$$Y(s) = \begin{cases} y_2(s) & s_1 \leq s < s_2 \\ \mu_{r,m}(s)y_{m+1}(s) + \mu_{f,m}(s)y_m(s), (2 \leq m < n_w - 1) & s_2 \leq s < s_{n_w-1} \\ y_{n_w-1}(s) & s_{n_w-1} \leq s \leq s_{n_w} \end{cases} \tag{2.27}$$

$$Z(s) = \begin{cases} z_2(s) & s_1 \leq s < s_2 \\ \mu_{r,m}(s)z_{m+1}(s) + \mu_{f,m}(s)z_m(s), (2 \leq m < n_w - 1) & s_2 \leq s < s_{n_w-1} \\ z_{n_w-1}(s) & s_{n_w-1} \leq s \leq s_{n_w} \end{cases} \tag{2.28}$$

and $\mu_{r,m}(s),\ \mu_{f,m}(s)$ are membership functions given by:

$$\begin{aligned} \mu_{r,m}(s) &= \frac{s - s_m}{s_{m+1} - s_m} \\ \mu_{f,m}(s) &= \frac{s_{m+1} - s}{s_{m+1} - s_m} \\ m &= 2, 3, ..., n_w - 1 \end{aligned} \tag{2.29}$$

Note that the first and the last polynomial is not overlapping any of the other, hence the membership functions can be thought of as equal to one and zero in these regions. In the intermediate waypoints, we "blend" the polynomials smoothly by linearly increasing and decreasing the membership of two polynomials. In the paper by Chang and Huh (2015), they go on to prove G^2 continuity and details the derivation of the method even more. A visual example of the QPMI method is seen in Figure 2.3.

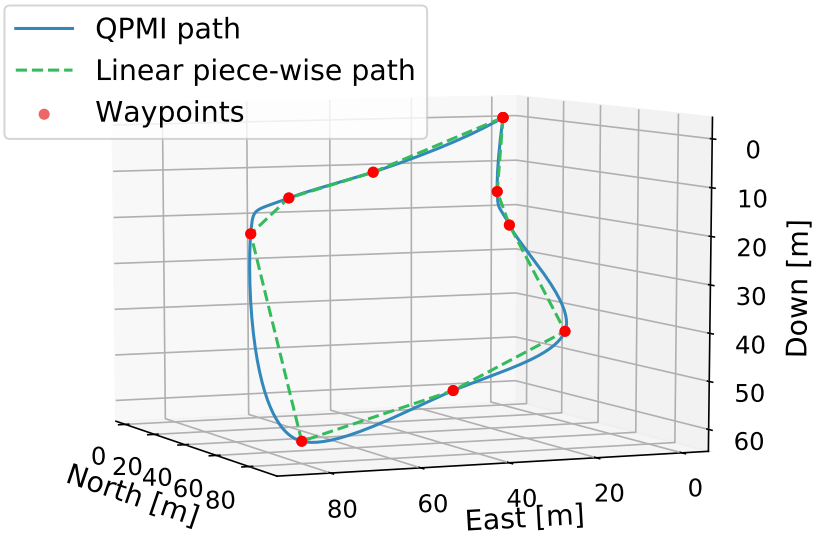


Figure 2.3: A set of ten waypoints connecting a path using linear and QPMI method.

2.3.2 Path-centered Coordinate System

To define the tracking errors, the Serret-Frenet ({SF}) reference frame associated with each point of the path is introduced. The x_{SF} axis points tangent to the path, the y_{SF} axis normal to the path and the z_{SF} axis points orthogonal to both such that $z_{SF} = x_{SF} \times y_{SF}$ (Encarnacao and Pascoal, 2000). The tracking-error vector, $\boldsymbol{\varepsilon} = [\bar{s}, e, h]^T$ is defined by the along-track, cross-track and vertical-track error pictured in Figure 2.4. The tracking-error vector points towards the closest point on the path from the vessel. Because the origin of the {SF} frame can be arbitrarily placed, the point on the path closest to vessel is chosen as the origin in the simulation. This yields $\bar{s} = 0$, which intuitively makes sense in a path following scenario since the path is not dependent on time. There is therefore no error in the along-track distance component.

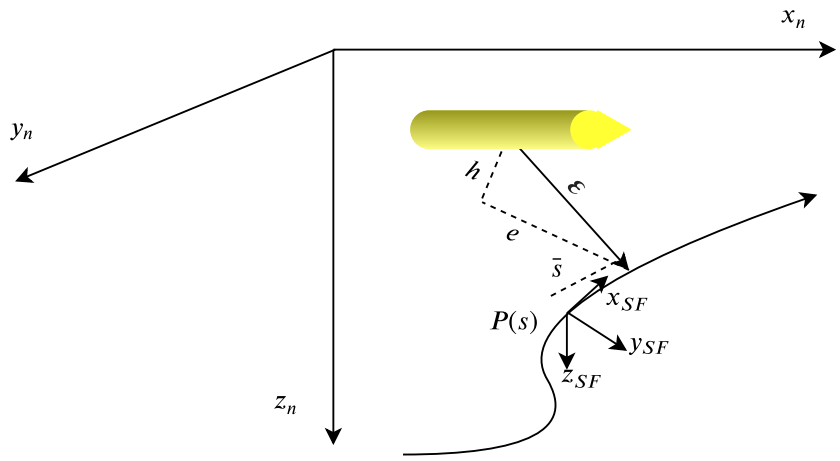


Figure 2.4: The Serret-Frenet reference frame defines the components for the tracking error vector. For illustration, we have shown that the origin of the SF frame can be placed arbitrarily along $P(s)$, but in the simulation it coincides with the closest point on the path from the vessel. This because the control objective in path following is to drive e and h to zero.

2.3.3 Guidance Laws for Path Following

The goal of path following is aligning the vehicle's velocity vector in n, $\mathbf{V}^n$, with the tangent direction of the path. In case of a tracking error, as demonstrated in Figure 2.4, a user-specified look-ahead distance Δ is set to guide the vessel back to the path. Hence, the goal is not to guide the vehicle perpendicularly towards the path, but instead smoothly converge back to the path further downstream. The guidance laws used for generating desired pitch and heading is therefore a look-ahead based steering-law generating a LOS-vector. These signals are calculated from ε and Δ. First, we obtain the tracking errors by Equation 2.30 (Breivik and Fossen, 2009):

$$\varepsilon = \mathbf{R}_n^{SF}(\upsilon_p, \chi_p)^T(\mathbf{P}^n - \mathbf{P}_p^n) \tag{2.30}$$

where $\mathbf{P}^n$ is the position of the vessel and $\mathbf{P}_p^n$ is the closest point on the path. Now the desired azimuth and elevation angle can be calculated according to:

$$\chi_d(e) = \chi_p + \chi_r(e) \quad , \quad \upsilon_d(h) = \upsilon_p + \upsilon_r(h) \tag{2.31}$$

where

$$\chi_r(e) = \arctan(-\frac{e}{\Delta}) \quad , \quad \upsilon_r(h) = \arctan(\frac{h}{\sqrt{e^2 + \Delta^2}}) \tag{2.32}$$

It is seen that driving e and h to zero will in turn drive the correction angles $\chi_r(e)$ and $\upsilon_r(h)$ to zero, achieving the goal of path following by aligning the velocity vector with the path-tangent when $\chi_d(e) = \chi_p$ and $\upsilon_d(h) = \upsilon_p$.

3 Method and Implementation of the Environment

This section aims to give an overview of the tools and methods used in implementing the simulation model and setting up the RL environment which the DRL controllers are developed in. As previously stated, the two practical challenges by applying RL methods are 1) how to build meaningful and challenging tasks for the agents, and 2) how should the training process be setup in detail; E.g. what information should be available for the agents, what range of obstacle detection, how to incentivize behaviour through a reward system etc. What follows is a suggested solution, but by no means the only solution.

First, a standard interface used in RL research is presented in subsection 3.1 before subsection 3.2 details the engineering of the specific simulation environment tailored to the current work. The application of the learning method known as *curriculum learning* is presented in subsection 3.3 with the related training scenarios developed to fit into this framework. The environment serves as the foundation for all scenarios, and a scenario refers to how the environment's building blocks are configured. Next, a proposed solution to simulate a forward looking sonar in 3D is offered in subsection 3.4. This provides perception of obstacles relative to the AUV's position. The reward function design is presented in subsection 3.5, before detailing the observation vector (i.e. the available information for the agent to base its decisions on) concludes the section in subsection 3.6.

3.1 DRL Framework and the OpenAI Interface

In order to build the environments, the **OpenAI Gym** interface and framework for RL is used to ease implementation and, to some extent, standardize the code. OpenAI Gym (Brockman et al., 2016) was created with this exact purpose in mind: To expose a common interface, shown in Listing 1, and provide a standard toolkit for reinforcement learning research. In addition to the common interface, the framework contains a set of test environments where anyone can implement and benchmark their RL algorithm against the current state-of-the-art. This lies beside the point in this research, however, as we here instead want to use the current state-of-the-art on a custom environment created to simulate AUV motion control. In other words, no improvements on current RL algorithms are proposed.

Another RL framework built on top of Open AI is **Stable Baselines** developed by Hill et al. (2018), which implements improved parallelizable versions of the RL algorithms found in OpenAI Baselines (Dhariwal et al., 2017). These libraries are written in Python, and this warranted the further use of Python for the current project as well. The complete code can be found on Github[3].

[3]Link: `https://github.com/simentha/gym-auv`

```python
import gym
from gym import spaces

class CustomEnv(gym.Env):
  """Custom Environment that follows gym interface"""
  metadata = {'render.modes': ['human']}

  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()
    # Define action and observation space as gym.spaces objects
    ...

  def step(self, action):
    # Execute one time step within the environment
    ...
  def reset(self):
    # Reset the state of the environment to an initial state
    ...
  def render(self, mode='human', close=False):
    # Render the environment to the screen
    ...
```

Listing 1: The OpenAI Gym interface

3.2 Building and Simulating the Environment

The simulation environment contains an implementation of the AUV dynamics, a path generated from a set of waypoints $\mathcal{W}_p$, n_{obs} obstacles hindering the AUV from advancing along (or outside of) the path and an ocean current $\mathcal{C}$ perturbing the AUV's motion. How these components are configured depends on which scenario S is chosen. Further requirements to generate an environment is specifying the integration time-step Δt. The environment parameters and the various scenarios are easily modified through a configuration file.

From the common interface, we have that the environment object must implement a step function. Following the schematic drawn in Figure 1.3, the step comprises of the following sequence: 1) The agent receives an observation vector containing information about the environment. This vector is passed through the policy neural network to obtain a control action. 2) The control action has a consequence on the environment and the resulting AUV motion is calculated by simulating the equations of motion one step forward using a numerical solver. Here, the environmental disturbances are also a driving force impacting motion, such that the ocean current model is also simulated one step forward. 3) In turn, this affects the AUVs position and orientation relative to the path. Consequently, the guidance system calculates a new desired direction. This new information and a reward is so passed back to the agent and we start over at 1). Note that a reward signal is only necessary during training.

Algorithm 2 is used to generate an instance of the **PathColav3D** object, which inherits the OpenAI Gym base environment class and with that the standard interface seen in Listing 1.

Algorithm 2: Algorithm for creating simulation environment

Require:
Number of observations n_o and actions n_a
Scenario S
Simulation step-size Δt

Procedure: GenerateEnviroment(n_o, n_a, S, Δ_t)
Define action and observation spaces with dimensions given by n_o, n_a
Obtain initial AUV kinematic state η_0, $\mathcal{W}_p$, n_o and $\mathcal{C}$ from S
Construct a G^2 continuous path P by applying QPMI($\mathcal{W}_p$)
Distribute n_o obstacles along P
Initialize AUV at η_0
Set Δt as the integration step-size
Reset simulation history and episode reward

3.2.1 Numerical Solver

An ODE solver using the Dormand-Prince method of order five is implemented to simulate the AUV equations of motion. This method can be used with automatic adjustment of the step-size, but for simplicity this application uses a fixed step-size of $\Delta t = 0.10s$. This is the standard ODE solver used in MATLAB (the well-known "ode45" method) which is optimized for accuracy (Egeland and Gravdahl, 2002). According to MathWorks [4], this should generally be the first choice for numerical solvers. The butcher array for the Dormand-Prince method can be seen in Table 3.1. This implementation is a massive improvement from the preproject, where a single-step Euler integration was used with step-size $\Delta t = 0.01s$. In other words, the RL agent can now explore and sample 10 times faster.

Table 3.1: Dormand-Prince butcher array.

0						
$\frac{1}{5}$	$\frac{1}{5}$					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$			
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$		
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{49}{176}$	$\frac{-5103}{18656}$
y	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$

[4] The makers of MATLAB and Simulink. Link: mathworks.com

3.3 Environment Scenarios and Curriculum Learning

Curriculum learning is the idea of learning progressively harder tasks, to incrementally obtain better and more complex skills. Additionally, as the agent is introduced to more complexity it can exploit what it has already learned. The agent can "start small", learning easier aspects of a task and the level of difficulty is increased as the current level is mastered. Hence, in this context, five scenarios labeled "beginner", "intermediate", "proficient", "advanced" and "expert" level training scenarios are developed, where an expert level pilot should be able to solve all previous tasks impressively. The hybrid control problem of path following and collision avoidance can be effectively described as a multi-task RL problem: The desired outcome is for the agent to solve both tasks of following the 3D trajectory in space and deviating if an obstacle happens to hinder further on-path progress. However, it seems natural to define it as a progressive problem, as environmental complexity can be defined in terms of obstacle density and even the level of perturbation from external disturbances, such as the intensity of an ocean current. (Arguably, the level of path curvature is also a variable influencing environmental complexity. However, path curvature does not differ between scenarios.)

Curriculum learning was formalized by Bengio et al. (2009), where they showed that indeed machine learning algorithms do benefit from the concept of "starting small". What is truly interesting about this approach, is that it mirrors training techniques shown to be effective in human and animal learning. It also echoes the concept of self-play, seen when two game-playing agents play against each other over and over while complexity evolves as skill level increases.

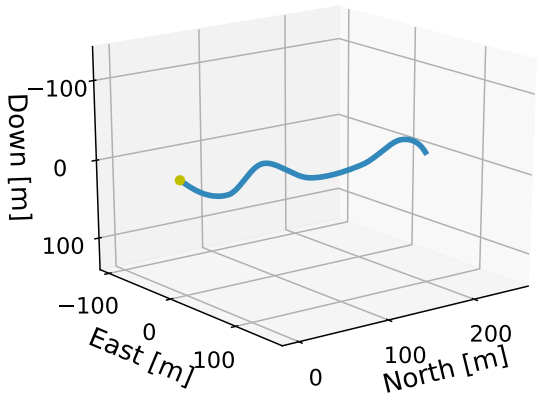
3.3.1 Training Scenarios

Training scenarios are constructed by generating a path from a random set of waypoints, drawn such that the elevation and azimuth angle between two adjacent waypoints are limited, distributing obstacles along this path and instantiate an ocean current with either zero or time-varying intensity. The scenarios defining the training levels used in curriculum learning are:

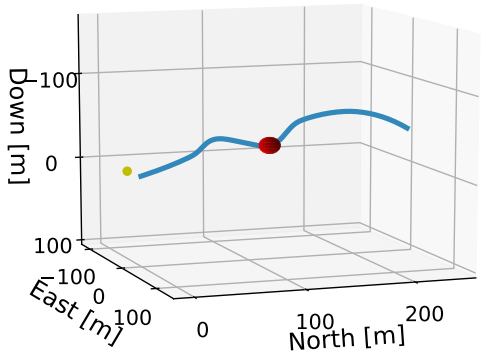
- **Beginner level**: Only a path and no obstacles or ocean current are present.
- **Intermediate level**: A single obstacle is placed on the half-way mark.
- **Proficient level**: Two more obstacles are placed equally distanced from the half-way mark.
- **Advanced level**: Additional obstacles are placed randomly outside the path such that an avoidance maneuver could induce a new collision course.

- **Expert level**: Similar to the advanced level, but here an ocean current is present.

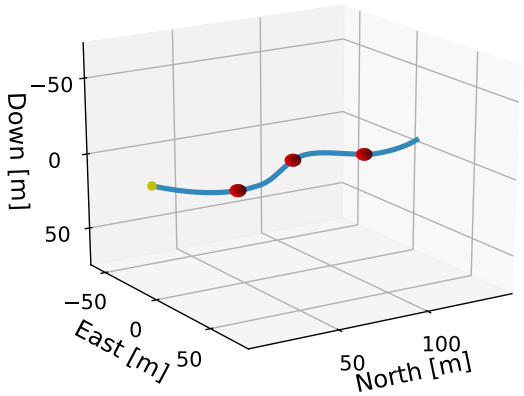
Figure 3.1 visualizes all training scenarios.



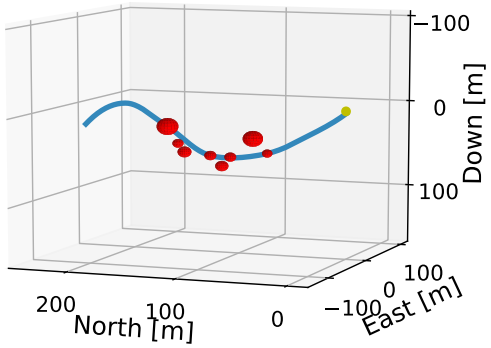
(1a) Beginner level



(1b) Intermediate level



(1c) Proficient level



(1d) Advanced/Expert level

Figure 3.1: Training scenarios used in curriculum learning and quantitative analysis.

As can be seen, in all scenarios the first and the last third of the path is collision-free, in order to keep part of curriculum from the beginner scenario (pure path following) present throughout the various training stages. This is intended to fight *catastrophic forgetting*, a phenomena seen occurring in neural

networks when learned knowledge are abruptly forgotten upon acquiring new knowledge (Goodfellow et al., 2013). In practice it has been seen that neural networks seemingly forgets how to do task 1 after having learned task 2, and by having the presence of pure path following throughout all scenarios, this knowledge should be kept intact from beginner to expert.

To ensure that the argument of increased complexity holds, all scenarios are generated such that they generate the previous difficulty level *plus* something more. For instance, intermediate level is generated by first instantiating the beginner level, then adding an obstacle. Similarly, the proficient level is generated by instantiating the intermediate level, then adding two more obstacles, etc. This way complexity always increase on average as we progress in the learning process.

In addition to train the agent progressively in these scenarios, quantitative evaluation is performed by sampling a number of episodes such that the agents' average performance across the various difficulty levels can be established. We then extract key metrics, specifically the average tracking errors, the collision rate and the success rate. This will be elaborated on in subsequent sections.

3.3.2 Test Scenarios

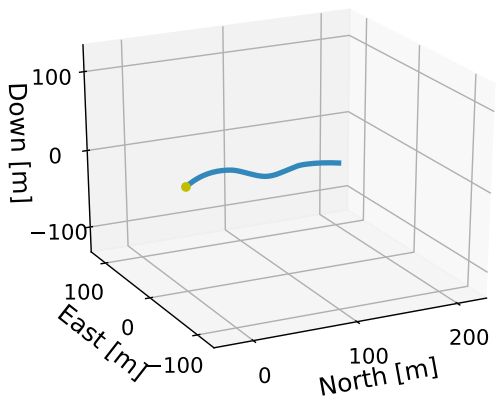
As neural networks are generally seen as "black boxes", it is desirable to evaluate performance in special-purposed test scenarios designed to test specific aspects of the agents' behaviour. Thus, four specialized scenarios targeting to expose certain behavioural features are created.

The first scenario tests the agent in pure path following. In order for results to be reproducible, a non-random path is generated from the waypoints given by Table 3.2. Furthermore, the test is performed both with and without the presence of an ocean current. Evaluation is based on the average tracking error obtained by calculating the integral absolute error and divide by the number of time-steps simulated.

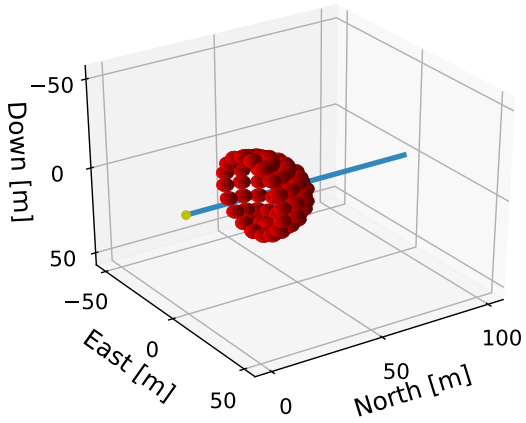
Next, special (extreme) cases where it would be preferable to use only one actuator for COLAV, i.e. horizontally and vertically stacked obstacles, are generated. The agents are also tested in a typical pitfall scenario for reactive COLAV algorithms: A dead-end. See Figure 3.2 for illustrations of the test scenarios.

Table 3.2: Waypoints for test path.

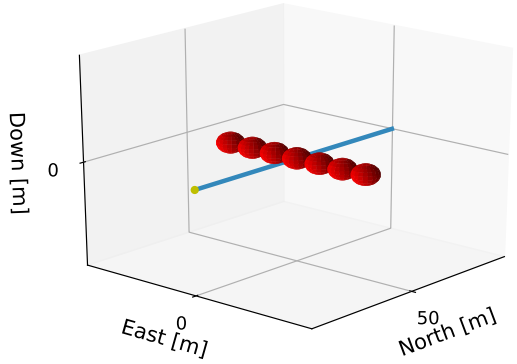
	WP_1	WP_2	WP_3	WP_4	WP_5
x	0	50	80	120	150
y	0	5	5	10	0
z	0	15	-5	0	0



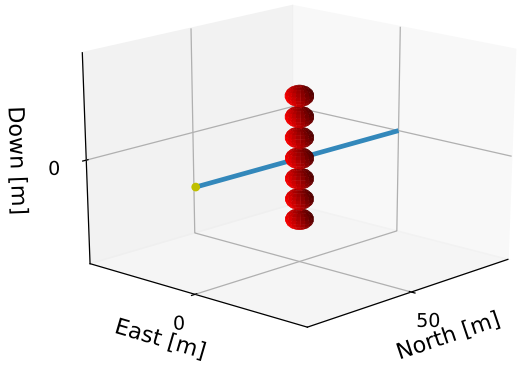
(2a) Path Following



(2b) Dead-end



(2c) Horizontal obstacles



(2d) Vertical obstacles

Figure 3.2: Test scenarios for qualitative analysis.

3.4 Forward Looking Sonar

Being able to react to the unforeseen obstacles require the AUV to perceive the environment through sensory inputs. This perception, or obstacle detection, is simulated by providing the agent a 2D sonar image, representing distance measurements to a potential intersecting object in front of the AUV. This setup emulates a forward looking sonar (FLS) mounted on the front of the AUV, as illustrated in Figure 3.3. A 3D rendering of the FLS simulation is seen in Figure 3.4. The specific sensor suite, the sonar range and the sonar apex angle is configurable, and can therefore be thought of as hyperparameters.

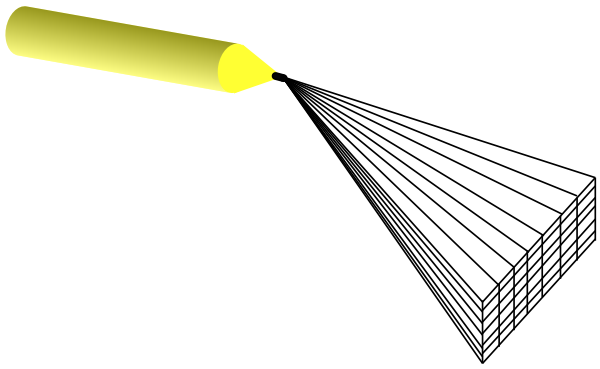
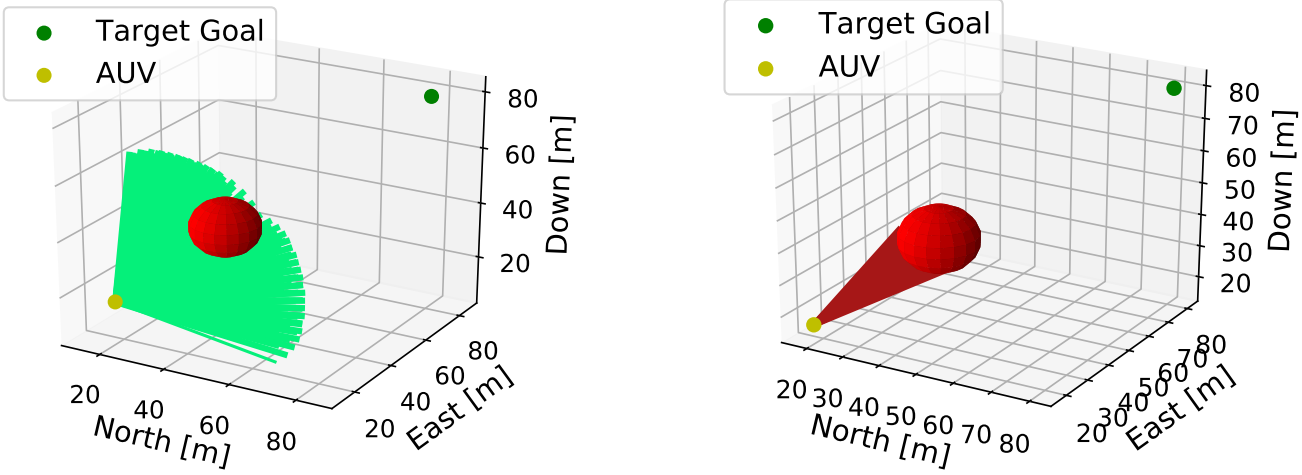


Figure 3.3: The FLS mounted on the front of the AUV scans both vertically and horizontally.

Depending on the chosen sensor suite, the number of sensor rays can get quite large. It is also notable that this issue is exponentially larger in 3D compared to 2D, slowing the simulation speed significantly as searching through the sonar rays (line search) is computationally expensive. For this reason, the sensor suite used in this research is 15 by 15, providing a grid with $10°$ spacing between each sonar ray when scanning with a $140°$ apex angle. This amounts to a total of 225 line searches per sensor update and in order to limit this stress on computational resources, the update frequency is set to $1Hz$ (every $10\Delta t$). Moreover, the sonar range s_r is limited to $25m$.



(4a) The green sonars rays are not detecting any obstacles. The red beams are not seen here due to overlay from the collision-free zones.

(4b) Here, in the exact same time-stamp, the collision-free sectors are removed and only the rays detecting an obstacle is showed.

Figure 3.4: 3D rendering of sonar simulation.

3.5 Reward Function

Reward function design is a crucial part of any RL process. The goal is to establish an incentive so the agent learn certain behavioural aspects. This is done by trying to capture human-like intuition in the form of a reward function. For instance, following the path is objectively desirable, but this goal must be suspended in the case off a potential collision. When to react and by what safety margin is inherently a subjective choice. Regulating this trade-off is a balancing act, where following the path notoriously would result in many collisions and being too cautious would be ineffective in reaching the end-goal. Additionally, a configuration involving excessive roll, i.e. the angular displacement of the AUV arounds its own longitudinal axis, is undesirable because that implies inverting or even swapping the two actuators' effect (the rudder would operate as the elevator and vise versa) in terms of combating course and elevation errors. Not using the actuators to aggressively is therefore key in achieving smooth and safe operation. Thus, a reward function incorporating these important aspects of AUV motion control are developed.

The first part focuses on path following and simply penalizes errors between desired and actual course and elevation angle, as given by Equation 3.1:

$$r_t^{pf}(\tilde{\chi}, \tilde{\upsilon}) = c_\chi \tilde{\chi}^2 + c_\upsilon \tilde{\upsilon}^2 \tag{3.1}$$

Where c_χ and c_υ are negative weights deciding the severity of being off the course and elevation angles calculated by the guidance laws. The next incentive is avoiding obstacles blocking the path seen through the 2D sonar image. First, the range measurements are converted to a proportionally reverse quantity called *obstacle closeness*. This quantity is written $c(d_{i,j}) = \text{clip}\left(1 - \frac{d_{i,j}}{d_{max}}, 0, 1\right)$, where $d_{i,j}$ is the i'th and j'th pixel distance measurement and d_{max} is the sonar range. This transformation sets all sensor inputs zero as long as there are no obstacles nearby, effectively deactivating learning in this part of the neural net during the beginner scenario. The term incentivizing obstacle avoidance is written in Equation 3.2.

$$r_t^{oa}(\mathbf{d}) = -\frac{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\left(\beta_{oa}(\theta_j,\psi_i)\left(\max\left(\gamma_c(1-c(d_{i,j}))^2,\epsilon_c\right)\right)^{-1}\right)}{\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\beta_{oa}(\theta_j,\psi_i)} \tag{3.2}$$

A small constant ϵ_c is used to remove singularities occurring when obstacle closeness in a sector is exactly 1 and γ_c is a scaling parameter. Since the vessel-relative orientation of an obstacle determines whether a collision is likely, the penalty related to a specific closeness measurement is scaled by an orientation factor dependent on the relative orientation. The vessel-relative scaling factor is written $\beta_{oa}(\theta_j, \psi_i) = (1 - \frac{2|\theta_i|}{\gamma_a})(1 - \frac{2|\psi_j|}{\gamma_a}) + \epsilon_{oa}$. Here, ϵ_{oa} is a small design

constant used to penalize obstacles at the edge of the configuration, and θ_j and ψ_j defines the vessel-relative sonar direction. The reward term is averaged by the scaling factor in order to remove the dependency on a specific sensor suite configuration. Figure 3.5 illustrates how the 2D sonar image is weighted in terms of the sector importance given by β_{oa}. As is clear, obstacles that appear centermost in the sonar image will yield the largest penalty.

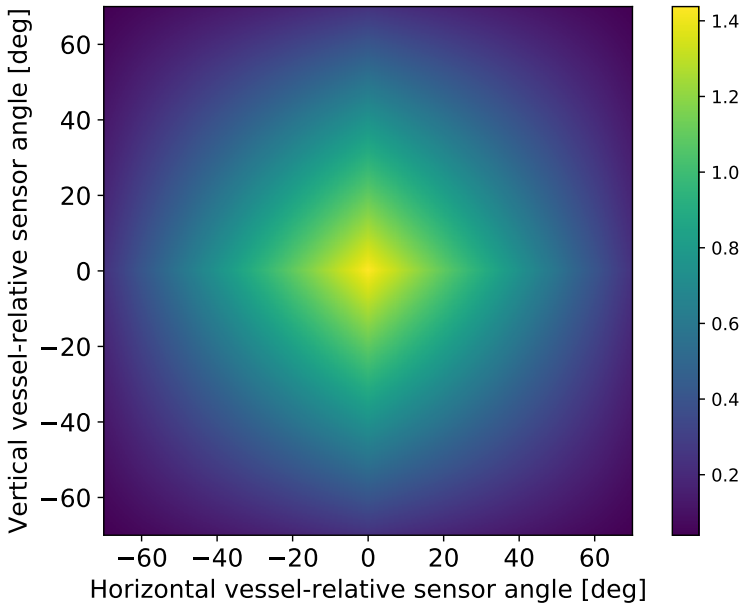


Figure 3.5: How the reward is scaled according to the sonar-data's vessel-relative direction. Note that the grid illustrated is much finer than the 15 by 15 sensor suite used during simulation.

To find the right balance between penalizing being off-track and avoiding obstacles - which are competing objectives - the weight parameter $\lambda_r \in [0, 1]$ is used to regulate the trade-off. This structure is adapted from the work by Meyer et al. (2020), which performed the analogous experiments in 2D. In addition, penalties are added to roll, roll rate and the use of control actuation to form the complete reward function:

$$r_t(\tilde{\chi}, \tilde{v}, \mathbf{d}, \phi, r, \delta_r, \delta_s) = \lambda_r r_t^{pf}(\tilde{\chi}, \tilde{v}) + (1 - \lambda_r) r_t^{oa}(\mathbf{d}) + c_\phi \phi^2 + c_r r^2 + c_{\delta_r} \delta_r^2 + c_{\delta_s} \delta_s^2 \tag{3.3}$$

In the subsequent sections, three agents with different values for the trade-off parameter λ_r are trained and simulation results are presented.

3.6 Feedback/Observations

The list of state observations, referring to the states of the dynamical model, the agents inputs during training and in operation is seen in Table 3.3. The inputs are normalized by the true or the empirical maximum, so that values passed into the neural network is in the range $[-1, 1]$. Input normalization is used to improve the speed of convergence and the symbols are denoted by subscript o to indicate that these are the actual values passed as observations. The nonlinear activation functions of neural networks tend to saturate if the inputs gets too large, hence normalization is a means used to counteract this effect. Furthermore, large input values might lead to huge error gradients, which in turn causes unstable training. Normalization is therefore a simple form of preprocessing contributing to faster and more stable training. (Yann LeCun and Müller, 1998)

Table 3.3: Observation table for end-to-end training of δ_r and δ_s. All states and errors are normalized by the empirical or true maximum value.

Observation	**Symbol**	**Max**
Relative surge speed	$u_{r,o} = \frac{u_r}{u_{max}} \in [-1, 1]$	2
Relative sway speed	$v_{r,o} = \frac{v_r}{v_{max}} \in [-1, 1]$	0.3
Relative heave speed	$w_{r,o} = \frac{w_r}{w_{max}} \in [-1, 1]$	0.3
Roll	$\phi_o = \frac{\phi}{\phi_{max}} \in [-1, 1]$	π
Pitch	$\theta_o = \frac{\theta}{\theta_{max}} \in [-1, 1]$	π
Yaw	$\psi_o = \frac{\psi}{\psi_{max}} \in [-1, 1]$	π
Roll rate	$p_o = \frac{p}{p_{max}} \in [-1, 1]$	1.2
Pitch rate	$q_o = \frac{q}{q_{max}} \in [-1, 1]$	0.4
Yaw rate	$r_o = \frac{r}{r_{max}} \in [-1, 1]$	0.4
Course error	$\tilde{\chi}_o = \frac{\chi_d - \chi}{\chi_{max}} \in [-1, 1]$	π
Elevation error	$\tilde{\upsilon}_o = \frac{\upsilon_d - \upsilon}{\upsilon_{max}} \in [-1, 1]$	π
Ocean current velocity, surge	$u_{c,o} = \frac{u_c}{V_{c,max}} \in [-1, 1]$	1
Ocean current velocity, sway	$v_{c,o} = \frac{v_c}{V_{c,max}} \in [-1, 1]$	1
Ocean current velocity, heave	$w_{c,o} = \frac{w_c}{V_{c,max}} \in [-1, 1]$	1

In addition to the state observations, the neural network inputs a flattened 2D sonar image measuring closeness. It is possible to pass the sonar image di-

rectly through the neural network, essentially learning to map raw sensor data to control action. By the fact that neural networks are capable of representing any continuous nonlinear function, this should be feasible in theory (Nielsen, 2015). However, as this requires a high-dimensional observation space, a larger neural network is needed to learn a control law. In turn, a larger neural net requires more data and more updates to converge, prolonging an already time-consuming process. To address this issue, dimensionality reduction is performed by max pooling the raw closeness image from $(15, 15)$ to $(8, 8)$. While max pooling tends to be more restrictive (a high closeness measurement indicates a small distance between the vehicle and an object in a vessel-relative channel), the extra dimension that 3D offers provides a viable path to pass the obstacles in most cases. Moreover, being restrictive favors safety and obstacle avoidance.

For the neural networks we utilize the *MLP-Policy* (multilayer perceptron) provided by Stable Baselines which incorporates a fully-connected, 2 hidden-layer neural network with 64 neurons in each layer using hyperbolic tangents ($tanh$) as the activation functions. The input size and the output size is decided by the observation space and the action space, respectively. As we pass 14 state observations plus the 64 pixel output from max pooling the raw sonar image, the total input vector is of size 78×1. The action space is naturally the rudder and elevator fin commands, meaning a 2×1 output vector. Figure 3.6 illustrates the controller neural network.

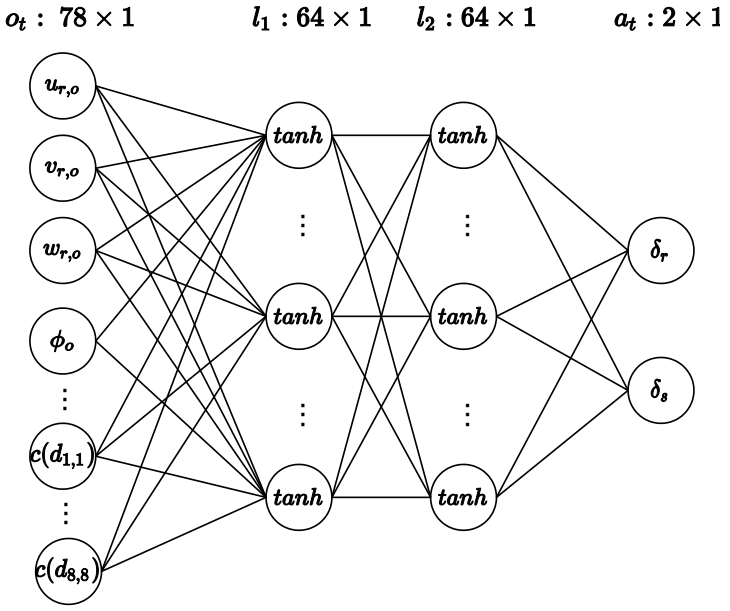


Figure 3.6: The neural network controller structure with inputs given by Table 3.3 and a max pooled closeness image from (15,15) to (8,8), yielding a total of 78 inputs. The hidden layers contains 64 neurons each that uses $tanh$ as the activation functions.

4 Training

This section documents the learning process of the controllers. Reports on key metrics is presented and analyzed in subsection 4.1 and a visual representation of the evolution of an agent is seen in subsection 4.2. The parameters and tuning used in the development of the agents are summarized in subsection 4.3.

4.1 Training History

Three important measures from training is used to document the learning process: episode reward, policy entropy and value-function loss. This information is logged every time the policy is updated, i.e. every TN timesteps where T is the number of time-steps per training iteration and N is the number of actors running in parallel, in accordance with algorithm 1. The agents trained for a total of $3000K$ timesteps, corresponding to $300K$ seconds worth of data when choosing $\Delta t = 0.1s$. The $3000K$ training timesteps is distributed over the various scenarios. How long (in terms of simulated timesteps) the agents spend at a certain stage increases with difficulty.

The training was performed on a Dell Optiplex 7060 stationary computer, with an Intel i7-8700 CPU, 32GB RAM and, unfortunately, no way to activate GPU acceleration. With that level of hardware, the early stages of training involving pure path following and low-density obstacle scenarios can run at up to 150 frames per second (FPS). Because the FLS is only updated if there are any nearby obstacles - a liberty that can be afforded in a simulated environment since all objects' positions are known - the computational stress is much lower. However, as the number of obstacles increases, the likelihood of an obstacle being within the sonar window is higher and the sonar readings is updated at the nominal frequency of $1Hz$. (If the update frequency is increased more than this the simulations just simply takes too long). In high-density situations, simulation speed can fall as low as 10 fps. The timestep limit at $3000K$ is therefore set for all practical purposes, as tuning the training parameters (finding workable parameters in Table 4.1) requires some trial-and-error before performing as intended.

4.1.1 Episode Reward

The first metric reported from the training process is the episode reward: Episode rewards are the mean return for the episodes simulated between two updates. The episode rewards should consistently increase throughout the learning process, but some volatility is to be expected because of the random nature of environment instantiations.

The reward function is shaped to be exclusively negative such that the agent hastens to reach a terminal state. There are four triggers to episode termination:

1. The AUV reaches the last waypoint/end-goal within an acceptance radius d_a such that $||p_t^n - p_{target}^n||_2 \leq d_a$.
2. The total along-track distance travelled is within $d_a/2$ distance to reach full path-length. Mathematically this is written $|s^* - s_f| \leq d_a/2$, where s_f is the full length of the path and $s^* = \min_s ||P(s) - p_t^n||_2$ is the closest along-track distance on the path from the AUV's current position. An example of this trigger is seen in Figure 5a.
3. The return is less than -1000.
4. More than 4000 timesteps has been simulated.

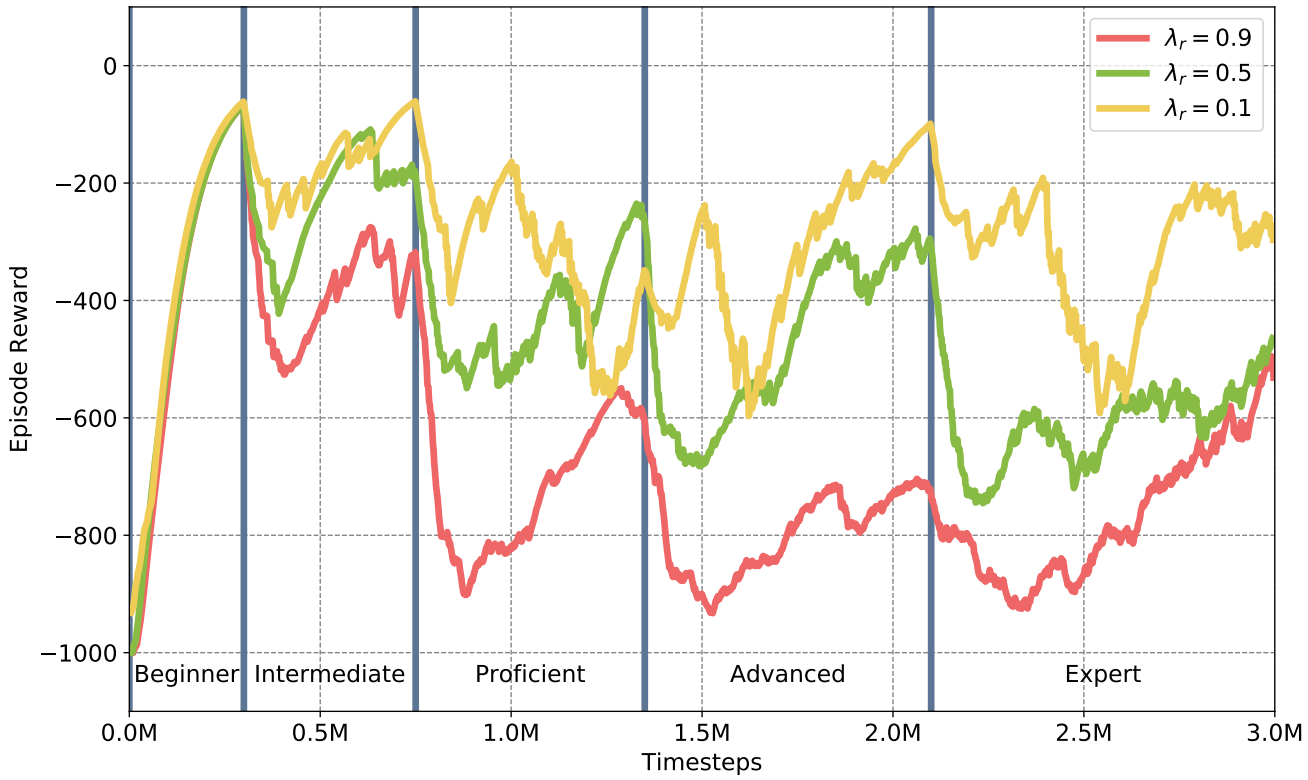


Figure 4.1: Reward plot for different reward functions sorted by the trade-off parameter λ_r.

Figure 4.1 pictures the episode reward history throughout training. The beginner scenario is text-book, with smooth and steep increase in performance as the agents learns to master path following. Then a decline in performance is

seen when an obstacle is placed on-path in the intermediate level. As the agent now follows the path naively, this is to be expected. Since all closeness readings are zero during beginner level, the weights associated with these input neurons in the MLP networks are essentially dropped out through this stage of learning. Thus, the agent following the path naively at this point is by design. After the initial decline seen when increasing difficulty level, more progress is made and this pattern repeats through to the final stage.

Another discovery from the episode reward history is made: As complexity increases, the data gets more noisy. Intuitively, this should be the case, since the obstacle density increase and even an ocean current is added to the environment in the expert scenario. This in turn increase the possibilities of the environments configuration space, exposing the agents to many new situations and data. Novel situations can uncover potential weaknesses in the agents' policies, leading to bad decisions and less return.

4.1.2 Policy Entropy

The policy entropy is a measure of how flat the distribution over the action probabilities are - a metric on to what degree the action taken is random. Since a policy is nothing but a probability distribution whose support is the action space $\mathcal{A}$, it can be written $\pi_{\boldsymbol{\theta}}(a|s) = \Pr(a_t = a|s_t = s)$. The mathematical definition of policy entropy, written $H(\pi_{\boldsymbol{\theta}}(a|s))$, stems from information theory and is expressed in Equation 4.1. (Juliani, 2018)

$$H(\pi_{\boldsymbol{\theta}}(a|s)) = -\int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \log \pi_{\boldsymbol{\theta}}(a|s) da \tag{4.1}$$

A way to think about this the agent becoming more committed, or more certain, to which action it should take given a set of observations as the entropy decreases. It is therefore a great illustration of how the agent trades off exploration for exploitation as its experience grows, just as humans and animals do. Examples of action-probability distributions with high and low entropy is seen Figure 4.2. The more entropy, the more randomness and ultimately exploration.

It is therefore quite typical for learning algorithms to include an *entropy bonus* to the overall objective function, to encourage more exploration. The PPO algorithm implemented in the Stable Baselines library is no exception, and the entropy bonus is scaled by setting the hyperparameter *ent_coeff*, symbolized by τ. In effect, this is a regularization term added to the RL goal discussed in subsubsection 2.1.2. The total objective funtion is then combined to form the extended RL goal written $\pi^*_{\boldsymbol{\theta}} = \text{argmax}_\pi \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^t (r_t + \tau H_t(\pi_{\boldsymbol{\theta}})) \right]$. Experimental results have shown that encouraging exploration yields faster convergence and in some cases better final policies. However, the latter improvement was highly environment specific (Ahmed et al., 2018). In this research, different values for

τ is not considered.

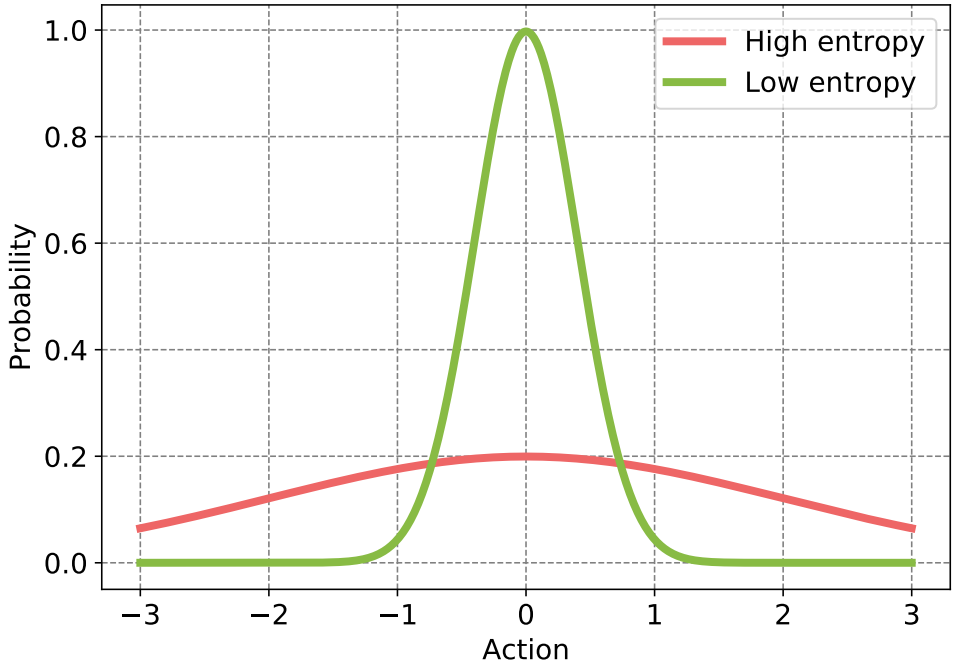


Figure 4.2: Examples of high and low entropy normal distributions.

An increasing policy entropy plot implies that the agent is exploring and is typically seen in the start of the learning process. Increasing entropy can also be observed when the return has converged and the only way increase the total objective function even more is to increase entropy bonus.

Figure 4.3 plots the progression of policy entropy as the learning process advance. Again, repeating patterns is seen as the agent is introduced to new difficulty levels. There is a striking difference in patterns between the high λ_r parameter, compared to the lower ones: From proficient level and on, this agent is mainly exploring until it suddenly finds a viable strategy in the last stage. This is in accordance with what is observed in the episode reward plots, where the agent biased toward path following is seen to collect less reward than the other controllers. But more importantly, it collects more rewards towards the end of the expert scenario, which is the most complex environment configuration, compared to the proficient and advanced level. This is what is mirrored in the entropy plot, the steep and steady increase in performance is related to the entropy decreasing after exploring.

The two other agents are behaving in a similar pattern and since they have a larger incentive to go off-track, their policies appear to be more adaptive and better prepared for increased complexity. Each scenario figures an early

exploration stage, with rapid trade-off towards exploitation. This indicates that these policies has learned something general about the path following/COLAV trade-off that the high λ_r setup has not before the last stage of training.

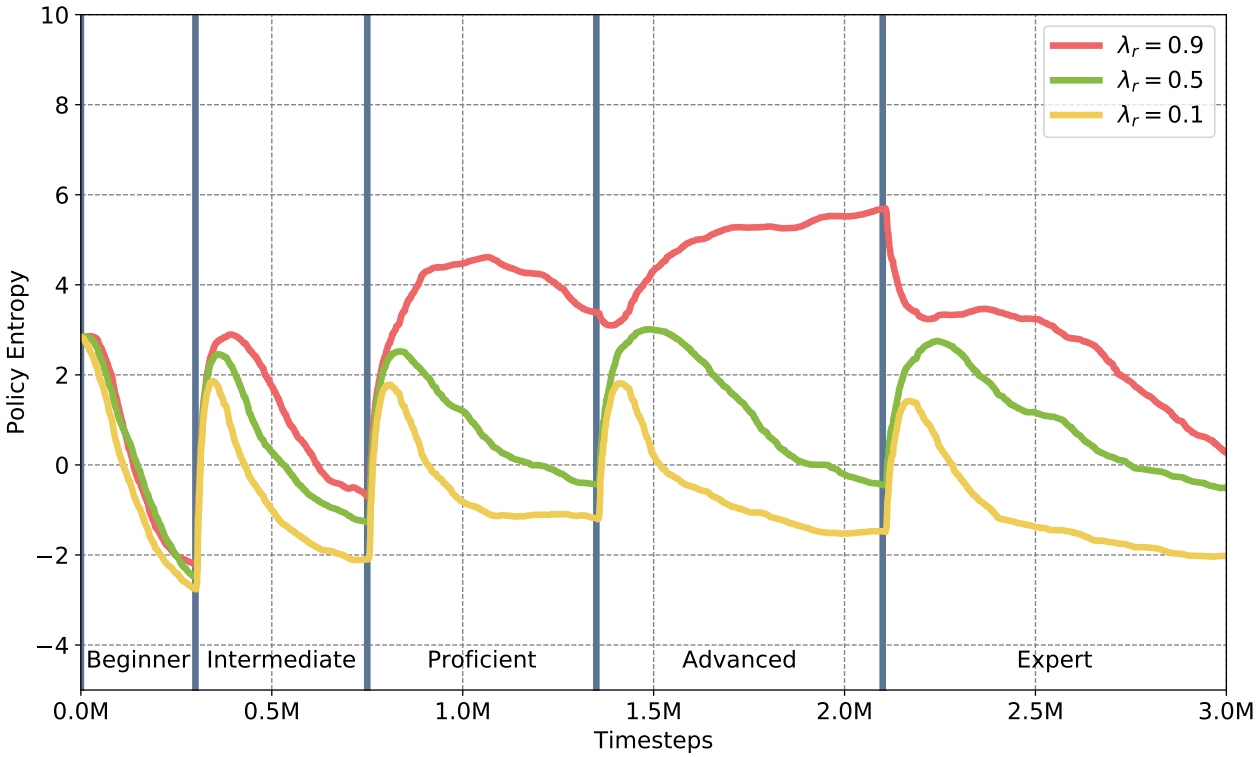


Figure 4.3: Policy entropy plot for training the different reward functions sorted by the trade-off tuning λ_r.

4.1.3 Value-function Loss

The last metric considered in the learning process is the value-function loss. As shown in subsection 2.1, the policy in PPO is updated as suggested by an approximation of the advantage function $\hat{A}_t^\pi$. Provided by Equation 2.4, the advantage-function is estimated by using the return and the value-function $\hat{V}_t^\pi$ which is approximated by the critic neural network. The value-function loss is measuring how far off the critic neural network are from the actual sampled trajectories. This metric should increase as the agent increasingly explores and decrease as the reward stably improves. As the value-function loss decrease, the critic neural network gets increasingly better at estimating the value-function. Ergo, the probability of an update improving the policy is increased.

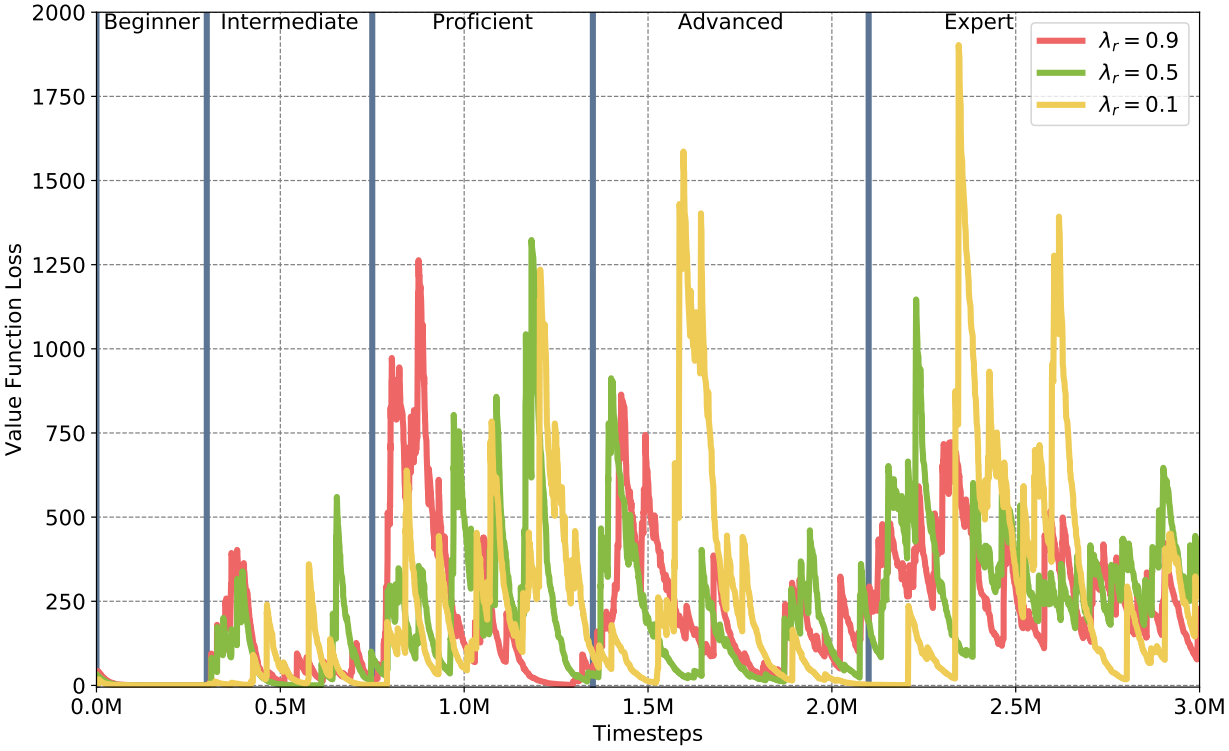
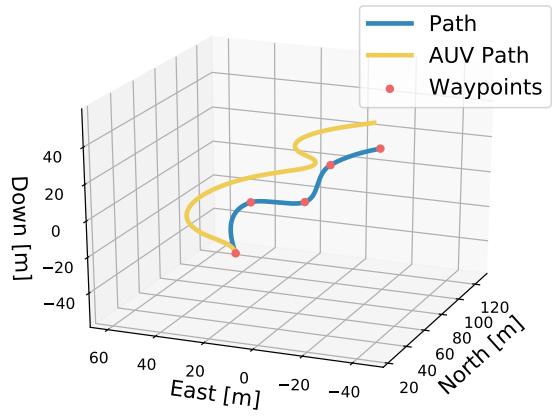


Figure 4.4: Value-function loss for training the different reward functions sorted by the trade-off tuning λ_r.

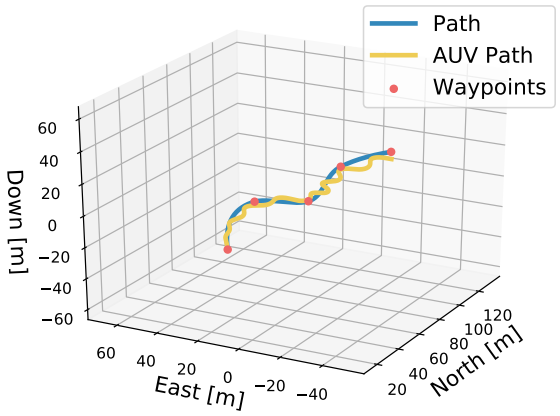
Except for the proficient level, most of the volatility in value-function loss, seen in Figure 4.4, is shown in the exploration stage and is consistently decreasing as the exploitation happens, as expected. It appears that the lower values of λ_r again are most correlated, which conforms with the evolution of the policy entropy. All of the metrics seen has reflected the same training history, although from different perspectives. In that sense they are inherently linked, and the heuristic guidelines has all been confirmed by the plots - which is a good sign in an RL training process. The plots has also highlighted the difference the trade-off parameter λ_r has on exploration vs exploitation and learning in general.

4.2 Evolution of an Agent

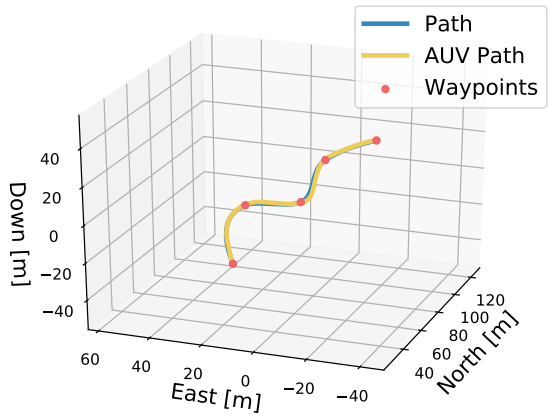
Reported key metrics from subsection 4.1 explains much of the evolution of the agents. But it is also helpful to see the performance translated into a practical setting. Figure 4.5 plots in 3D how the agent evolves from beginner to expert.



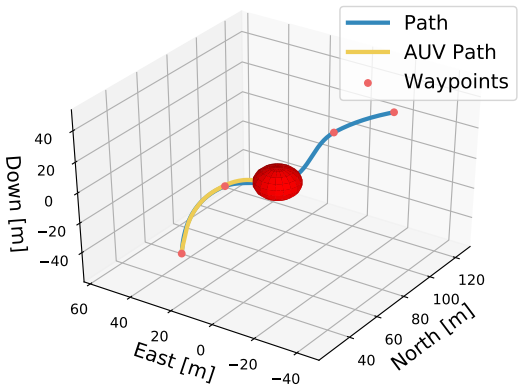
(5a) Beginner: $10K$ timesteps



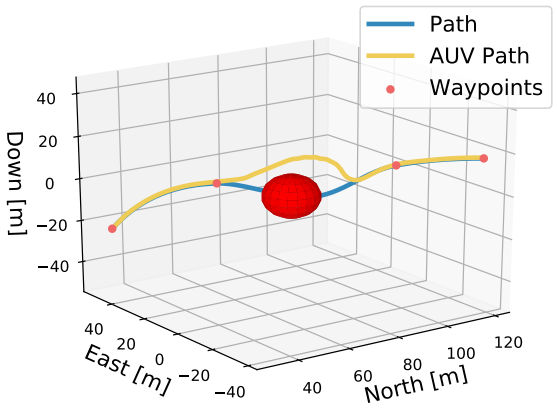
(5b) Beginner: $20K$ timesteps



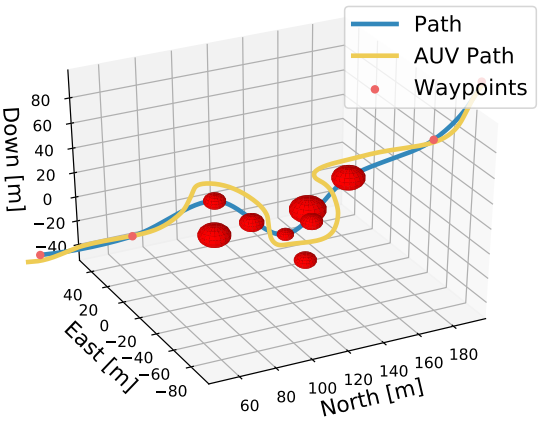
(5c) Beginner: $300K$ timesteps



(5d) Intermediate: $310K$ timesteps



(5e) Intermediate: $750K$ timesteps



(5f) Expert: $3000K$ Timesteps

Figure 4.5: Evolution of controller ($\lambda_r = 0.9$) performance throughout training.

The obtained plots is taken from the controller history with $\lambda_r = 0.9$, i.e. the controller that adheres mostly to staying on-path. Initially, the agent learns to minimize only one of the tracking errors such that it follows a projection of the path in another plane. Then, it closes this distance but behaves oscillatory. At the final stages of beginner level training, it follows the path with great precision. However, since it has not yet seen any obstacles it collides when switching to the intermediate level. After training for $450k$ timesteps in the intermediate level, it steers clear of the obstacle and follows the path with great precision in obstacle-free regions. Lastly, the agent must follow the path through a cluster of obstacles, and due to its on-path bias it navigates with precision through the obstacles instead of going around. In this scenario an ocean current is perturbing the motion as well, which is seen in the slight tracking error in the obstacle-free regions.

4.3 Summary of Training Setup

Many parameters and specifications related to the environment and training has been introduced from section 2 through section 4. Table 4.1 summarizes parameters and values used, except model parameters in the AUV dynamic equations and those that are not fixed through all scenarios. PPO parameters are associated with the learning algorithm seen in algorithm 1.

Environment parameters are parameters that define all components contained in the simulation environment, seen in Figure 1.3. This includes the AUV model, the Guidance system, disturbances and specifications that affect the system output and the observations. Note that the ocean current min/max values are only applied if the current is switched on. As there are many AUV model parameters, it was befitting to instead add this list as an attachment to the report.

The reward parameters corresponds to the penalty coefficients in the reward function expressed in Equation 3.3.

Table 4.1: Parameter table for training and simulation setup.

PPO	Description	Value
α	Learning rate	2.5e-4
γ	Discount rate	0.99
λ	GAE parameter	0.95
τ	Entropy bonus coefficient	0.001
T	# Steps per policy updates	1024
K	# Epochs	4

M	Batch size	64
N	# of parallel actors	4
Environment		
Δ	Look-ahead distance	3
n_w	# Training path waypoints	7
γ_a	Sonar span apex angle	140
s_r	Sonar range	25
$-$	Sensor suite	(15, 15)
$-$	Sensor min. pool output	(8, 8)
$-$	Sensor update frequency	1
$[V_{min}, V_{max}]$	Ocean current intensity limits	$[0.5, 1]$
d_a	End-goal acceptance radius	1
T_f	Control fins time-constant	0.2
Reward Function		
c_χ	Course error penalty coefficient	-1
c_υ	Elevation error penalty coefficient	-1
γ_c	Obst. closen. penalty scaling	-12.5
ϵ_c	Minimum obstacle penalty closeness	-5e-3
ϵ_{oa}	Minimum vessel-relative scaling	-0.05
c_ϕ	Roll penalty coefficient	-1
c_r	Roll rate penalty coefficient	-1
c_{δ_r}	Rudder action penalty coefficient	-0.1
c_{δ_s}	Elevator action penalty coefficient	-0.1
λ_r	path following/COLAV trade-off	$[0.9, 0.5, 0.1]$

5 Simulation Results

This section covers the results obtained from applying the finalized DRL controllers in the various scenarios introduced in subsection 3.3. Firstly, test reports from quantitative tests, which are obtained by running the simulation for a large sample of episodes and calculating statistical averages, are given in subsection 5.1. In light of these results, the behavioural aspects can be extrapolated to visualize and pinpoint the clearer trends. Secondly, the reports from testing the controllers in special-purposed scenarios are shown and analyzed to qualify if the agents have indeed learned to operate the AUV intelligently. As was given in Table 4.1, three values for the trade-off parameter λ_r was used during training to obtain three expert level controllers. This gives rise to a rational hypothesis on test outcome: The agent trained with $\lambda_r = 0.9$ should on average yield a lower tracking error, whilst maintaining a higher collision rate. The reversed results should be seen in the case where $\lambda_r = 0.1$.

5.1 Quantitative Results

The quantitative results are obtained by running each training scenario, configured randomly in each episode, for $N = 100$ episodes. As metrics we use success rate, collision rate and average tracking error over all episodes. Success is defined as the agent reaching the last waypoint within the acceptance radius without colliding. Equivalently, a collision has happened if the distance between the AUV and any obstacle, at any time during an episode, is less than a specified safety radius $d_{safety} = 1m$. Table 5.1 lists the full report from the quantitative tests.

Table 5.1: Test results from sampling $N = 100$ random training scenarios.

Trade-off	Metric	Interm.	Prof.	Adv.	Expert	Avg.
$\lambda_r = 0.9$	Success rate [%]	68	66	62	52	62
	Collision rate [%]	16	28	34	38	29
	Avg. tracking error [m]	1.67	2.91	3.14	3.09	2.70
$\lambda_r = 0.5$	Success rate [%]	100	100	86	59	86
	Collision rate [%]	0	0	8	36	11
	Avg. tracking error [m]	1.97	3.76	4.44	4.33	3.63
$\lambda_r = 0.1$	Success rate [%]	65	68	45	54	54
	Collision rate [%]	0	0	0	3	0.75
	Avg. tracking error [m]	3.98	6.15	7.91	7.33	6.34

The results show a clear connection to the hypothesis that higher λ_r should result in lower tracking errors but higher collision rate on an average. Con-

versely, low λ_r should result in fewer collisions but higher average tracking error. The results seen matches exactly with this expectation.

The quantitative results can be extrapolated to find general expressions for the success rate, collision rate and average tracking error as functions of λ_r. The collision rate and the average tracking error are well-described by exponential functions $y = ae^{bx} + c$. It is also seen that a quadratic function $y = ax^2 + bx + c$ describes the success rate as a function of the trade-off parameter quite well. This matches the expectations as higher λ_r induce more collisions and therefore lowers the success rate. On the other hand, during the episodes where it manages to avoid collisions it always succeeds because the tracking error is very low. Lower λ_r configurations naturally has the opposite problem: The low collision rate is due to it being more willing to go off-track, but makes it less likely to reach the end-goal within the acceptance radius. Figure 5.1 plots the data points from Table 5.1 together with the curve-fitted functions of λ_r.

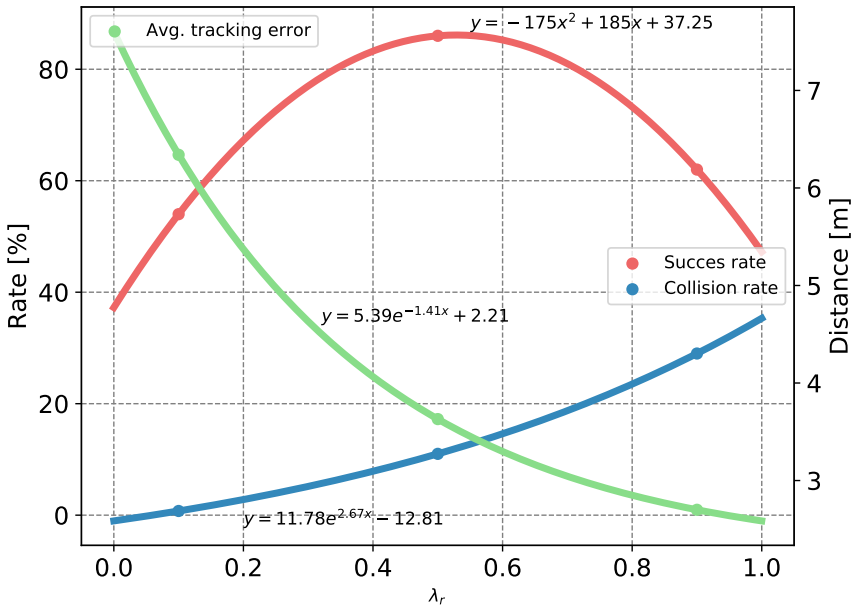


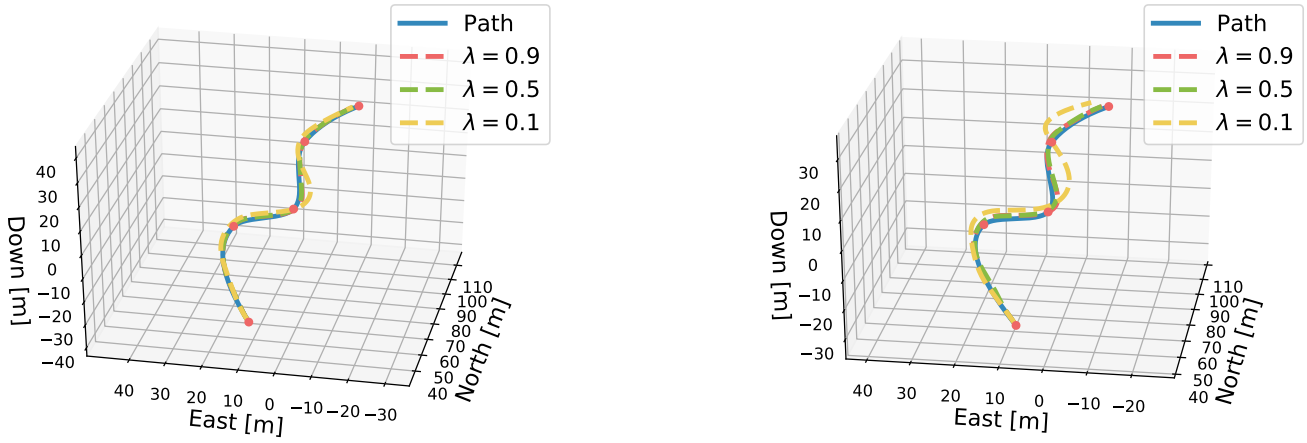
Figure 5.1: Data from Table 5.1 and their general functions obtained by curve-fitting.

5.2 Qualitative Results

In the qualitative tests, we run the three controllers in the test scenarios introduced in subsection 3.3 to observe behavioural outcome of the controllers. Controllers are run in deterministic mode to ensure that all results are reproducible (the action drawn from π_θ is always the center of the probability distribution). Equivalently, if a current is present, then the intensity and direction is fixed.

5.2.1 Path Following

The first test see the controllers tackle a pure path following test, both with and without the presence of an ocean current. Figure 5.2 plots the results from simulation. All cases are successful, except $\lambda_r = 0.1$ with current disturbance, which is visibly off-track as it passes the last waypoint.



(2a) Path following test, no disturbance.

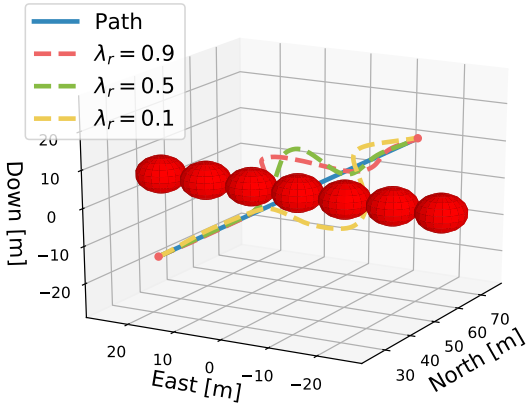
(2b) Path following test, with disturbance.

Figure 5.2: The pure path following test. (It might be hard to see $\lambda_r = 0.9$ due to overlay from the others)

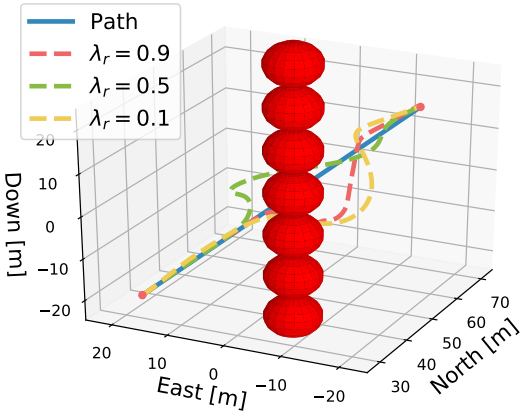
For $\lambda_r = 0.9$ an average tracking error of $0.45m$ and $0.52m$ in the ideal and disturbed environment is obtained, respectively; For $\lambda_r = 0.5$ we obtained $0.54m$ and $0.98m$; Finally, $\lambda_r = 0.1$ achieved $1.64m$ and $3.95m$. This amounts to a $15\%, 81\%$ and 141% increase in tracking error due to the disturbance, respectively.

5.2.2 Optimality Check - Extreme Obstacle Pose

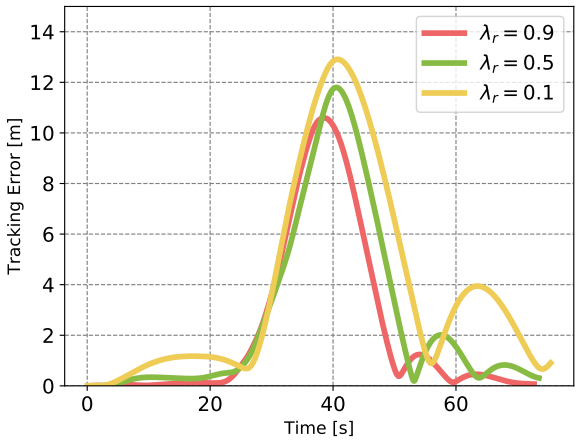
In the next test we dissect if the agents has learned to operate the actuators effectively according to how obstacles are posed. In the extreme cases, obstacles would be stacked horizontally and vertically, and optimally no control energy should be spent on the taking the AUV towards "the long way around". Instead it should use the actuator to avoid on the lateral side of the stacking direction; Surely, an intelligent pilot would pass the obstacles in this manner.



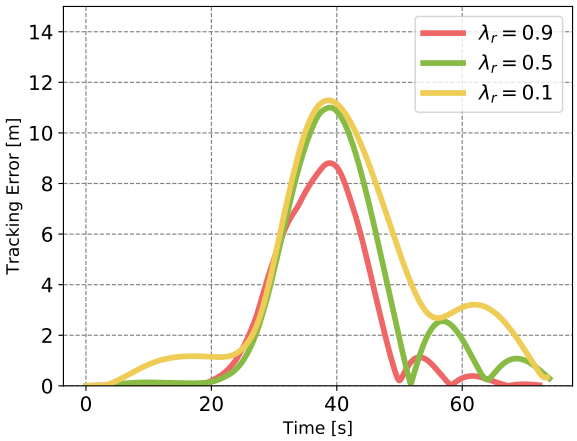
(3a) Horizontal obstacles, 3D plot



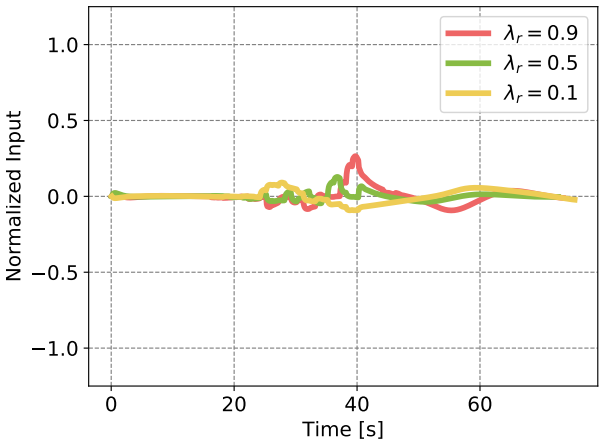
(3b) Vertical obstacles, 3D plot



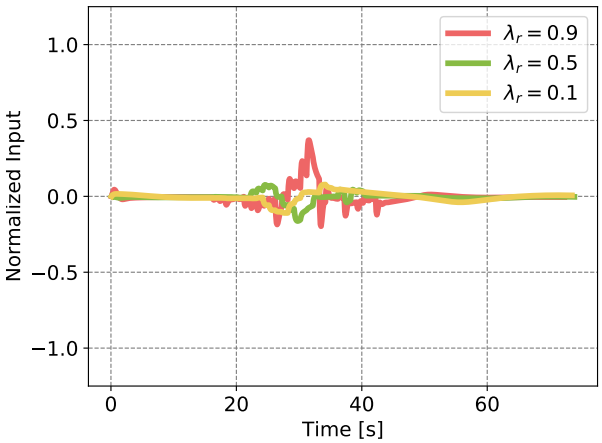
(3c) Horizontal obstacles, Tracking error



(3d) Vertical obstacles, Tracking error



(3e) Horizontal obstacles, Rudder input



(3f) Vertical obstacles, Elevator input

Figure 5.3: The horizontal and vertical obstacle test. Here, we are interested in seeing if the agent has learned which actuator to use to avoid the obstacles.

From the plots, it is observed that all agents waste little control energy using the "opposite" control fin. The agent with $\lambda_r = 0.9$ uses more than the others due to it being slower to react. It therefore has to spend more energy as it approaches the obstacle and might have to pull all levers to avoid collisions. The agent with $\lambda_r = 0.1$ is seen to plan further ahead, as it takes action earlier than the other two, but it also travels far off the path. The controller with $\lambda_r = 0.5$ can be seen to operate with human-like decision making. It steers clear off the obstacles in a nice and smooth curve, and it does not deviate in the plane that is not obstructed by obstacles.

5.2.3 Dead-end

A recurring problem seen when applying purely reactive algorithms is getting trapped in local minimas, which in a practical sense materialize as dead-ends. Therefore, the last test investigates if the agents have acquired the intelligence to solve a local minima trap - in the form of a dead-end challenge. In addition, this can affirm the robustness and generality learned by the agents, as this is a completly novel situation.

The obstacles are configured as a half-sphere with radius 20m. This means that the agent will sense the dead-end $5m$ prior to the center (due to $25m$ sonar range), and must take the appropriate actions to escape it.

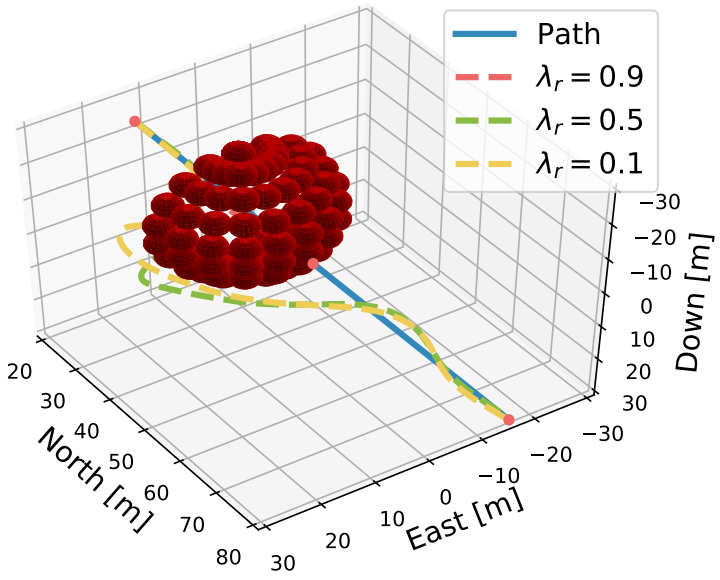


Figure 5.4: A dead-end test, where the the obstacles are configured as a half-sphere with a radius of $20m$.

The simulation, figured in Figure 5.4, shows that $\lambda_r = 0.9$ fails this test and can not escape the dead-end on account of it being too biased to staying on path. On the other hand, $\lambda_r = 0.5, 0.1$ behaves somewhat similarly and manages to escape and reach the goal position. This is impressive performance as this scenario is novel for the agents, and due to it being a classical pitfall scenario for reactive algorithms.

6 Discussion

6.1 Model Assumptions and Implementation

The 6-DOF simulation model implemented to emulate AUV motion is based on standard, justifiable assumptions seen in the relevant literature (Fossen, 2011; da Silva et al., 2007). However, the classical GNC structure figured in Figure 1.1 involves a form of state estimation and filtering in the *navigation* module. This module was omitted in the setup used in this research, and it was assumed that all states, including information about the ocean current, were available for feedback. In a full-scale test, state estimation would naturally be part of the feedback-loop, necessitating the need for a navigation module.

By posing the complete control system as a cascaded structure, tools from nonlinear theory can be exploited in analyzing stability and robustness. For instance, a nonlinear *separation principle* can be applied to prove stability by showing that each cascaded subsystem is passive. If this requirement is satisfied, and the states are bounded globally by the state feedback controller, then the feedback connection will be stable after replacing the states (x) with estimated states ($\hat{x}$) (Khalil, 2002). This fact is taken advantage of in Fossen (2000), where an output feedback controller was used in conjecture with a nonlinear passive observer to render closed-loop control for dynamic positioning uniformly, globally, asymptotically stable. However, this strong stability result hinges on the fact that the controller is also passive, and this property can not be shown for the DRL controllers developed in the current research.

Another source of discrepancy to a real-world application was the exclusion of sensor noise. As neural networks are capable of learning function maps (the policy in DRL terminology) whose output changes substantially with small changes in input (Zhang et al., 2019), training the neural networks in the presence of sensor noise and possibly noisy model parameters could prove important if the agents were to be used in a full-scale test. A DRL specific approach intended to encourage exploration, is adding parameter noise to the weights of the neural network, $\boldsymbol{\theta}$ (Fortunato et al., 2017). However, adding sample noise instead, which in the current context translates to AUV model parameter noise and sensor noise, has been shown empircally to increase performance and act as a regularizer (Xu et al., 2009). There is thus reason to believe that training the model with such noise in the environment could automatically build robustness wrt. uncertain model parameters and protection against sensor noise. More importantly, this setup could also integrate a Kalman filter, which then would serve as the navigation module and close a GNC loop well-suited for real-world application.

6.2 On the Method

The promise of DRL is that truly autonomous agents can be developed using general learning principles and succeed in achieving any goal, no matter which domain it is applied in. Nonetheless, there are challenges related to the application in safety-critical systems, such as autonomous vehicles. An important one is the lack of strong guarantees on behaviour and stability. Some approaches have tried to use the passivity framework to analyze stability of artificial neural networks, though in smaller MLP networks (Wen Yu and Xiaoou Li, 2000; Menhaj and Rouhani, 2002). However, the networks were used in an architecture mirroring direct-adaptive control and had special update rules for the weight parameters to obtain said stability properties. It is not certain that such constraints on the update rules can be used in the context of deep reinforcement learning.

The use of DRL controllers are motivated by complex environments where making decisions are based on intuition rather than concrete behavioural laws. In such environments, the only way an agent can learn is by making errors and correct for them. Any simulation model emulating a physical process contains modelling errors, implying that agents deployed in full-scale test must adapt through a new learning process. In the context of AUVs, tracking errors could be accepted in this adjustment process; A collision, however, would be fatal. Safe exploration while training agents directly in the real world is an on-going issue in the scientific community, and an attempt to address it has been made by Joshua Achiam (2019) by making a safety AI benchmark and introducing the concept of constrained RL. Constrained meaning that the RL objective, seen in Equation 2.3, should additionally satisfy constraints limiting unsafe interactions. In their work, PPO was adapted to fit the constrained RL formalizm, and this would be an approach worth considering if the DRL controllers for AUVs ever should be implemented in full-scale.

A third challenge with the method is setting up the environment and the learning. As there are some guidelines and heuristic principles one can use to tune the hyperparameters and the rewards, there are no guaranteed success formulas. Some machine learning libraries, such as *Tune* or *SMACV3* (Mikko, 2018), can offer assistance in this search, but considerable computational resources are required to use these. The hyperparameter tuning done in this project was therefore a manual effort.

In spite of the challenges related to DRL control, however, it is a powerful tool with great versatility:

- Firstly, it requires no simplified representations of the system: By learning through continual interaction and feedback from the full-state simulation model (or the real world in a full-scale test), the representations learned

by the neural nets might capture truths about the underlying environment that are out of reach for control engineers.

- Secondly, this unlocks the potential for learning in environments even too complex to model.

- Thirdly, as science progresses on understanding the representations learned by neural nets, it might help in building our understanding of physical phenomena as well. A concrete example could be using model-based RL in AUVs - that is algorithms that build an explicit representation of the transition model - for building a hydrodynamic model from experience. If this representation could be dissected and understood, knowledge on e.g. vessel models, higher-order hydrodynamic damping and turbulence might come out of it. There are many exciting discoveries yet to be made.

A key factor in achieving the research goals, was applying the curriculum learning paradigm. Pivotal to obtaining controllers with automatic collision avoidance in 3D, was learning to operate both control fins together. By deploying the agent in the most complex scenario from start to finish (no curriculum), it was not possible to obtain any working controllers on the described level of hardware. However, using the curriculum learning method made it possible to achieve the dual objective of 3D path following and collision avoidance by accelerating convergence significantly, providing further evidence for the utility of sorting the training data by complexity - as was found by Bengio et al. (2009).

6.3 On the Results

The reward function was set up to produce a clear hypothesis: Agents tuned with higher values for the path following/COLAV bias λ_r should tend to follow the path, while agents tuned with smaller values for λ_r should be more willing to go off-track and prefer to pass obstacles at a larger distance. The simulation results showed a clear connection between the incentive provided and the resulting control strategy.

The agent biased towards path following showed great tracking precision, even in the presence of a perturbing ocean current. The COLAV biased agents showed promising path following performance, but were more susceptible to perturbation from the ocean current. The converse relationship was seen in the collision rate, as expected. The best agent in this sense managed to run without collision for 100% of the first 300 samples. However, a 3% collision rate was seen in the expert scenario where an ocean current is present.

In light of this, it then seems that being biased towards path-adherence resulted in the agent learning more about the environmental disturbance and its impact on tracking errors. This is also confirmed by the drastic reduction in

performance for $\lambda_r = 0.1$, moving from ideal to perturbed conditions. This suggests that agents with higher bias towards COLAV would need more training in perturbed conditions, in order to build the robustness that was seen in the $\lambda_r = 0.9$ case. This could potentially eliminate the collisions happening in the expert scenario for $\lambda_r = 0.1$, increasing performance even more.

It was also observed that the trade-off regulation had a direct impact on how far ahead the agent took action. High valued λ_r resulted in aggressive action close to obstacles, where as the low valued λ_r took an earlier and measured approach in commanding the steering fins. Combined with the observation from the policy entropy history, the data suggests that high λ_r should train longer in environments that contains obstacles, due to its tendency to explore in these scenarios and its high collision rate.

Overall the findings were encouraging, and there is a reason to believe that the method is viable in a full-scale test. It does seem, however, that the training should be modified to address specific weaknesses shown when selecting the trade-off, as outlined.

6.4 Suggestions for Future Work

The achievements of this thesis can be used as a pointer for further work. Some ways to take it further has already been mentioned in terms of assuring stability, implementing noise and safe exploration. There are also many other directions to make progression. Some of the ideas that was not implemented are suggested:

6.4.1 Moving Obstacles and Velocity Control

The velocity control used for training and simulation was done by a PI-controller maintaining a desired (constant) cruise speed. The DRL controllers were used to find optimal spatial trajectories as obstacles were stationary. A logical next step, which could mirror a real-world application better, is to implement moving obstacles. This could emulate movement induced by the ocean current, or other autonomous agents trying to follow intersecting spatial trajectories. In this setup it could be interesting to let the agent take control of the thrust control as well as the control fins, making it able to control the AUV surge speed. Hence, it would have a the ability to control its temporal trajectory, and could exploit this new dimension to progress on-path.

6.4.2 Sonar Pooling by Convolutional Neural Network

In the architecture used in this research, the sonar image was max pooled to extract a smaller image to reduce the observation space of the agents. There are many ways this pooling can be done, but essentially there is only one piece of

important information to extract from the sonar image: A feasible direction. In 3D this corresponds to an azimuth and elevation angle which is collision free.

A suggestion for further work is therefore to implement a convolutional neural network, which can learn this feature extraction from the sonar image by being trained simultaneously with the agent end-to-end. In this way, it does not matter what source the perception comes from, be it a RGB camera, a sonar or a LIDAR. Thus, seperating the perception from the decision-making could yield a more general-purpose controller.

6.4.3 Real-world Implementation

To verify the method, a practical application in a full-scale environment would be a natural next-step. However, AUVs are very costly. A lower threshold starting point is a flying drone. As the methods and algorithms used are general, the only changes needed should in theory be implementing a suiting simulation model with the appropriate observation and action space and reward function.

6.4.4 Control System Architecture

This implementation of DRL control was aimed at the *control* level in the classical GNC-loop. In other words, the agents were directly responsible for controlling the physical actuators of the AUV. Though, if the aspiration is to mirror human-like intelligence, it is not clear that the decision-making is best served at low-level in the control-hierarchy. For instance, consider a human pilot operating a commercial aircraft: The pilot does not directly interact with the physical actuators of the aircraft, but rather controls the reference signals generating desired pitch and course signals forwarded to low-level control. An interesting architecture could therefore place the DRL control in the *guidance* system. Here, it could generate desired pitch and course angles based on perception and a nominal path received from a path-planner.

7 Conclusion

In this research, DRL agents was trained and deployed to tackle the hybrid objective of 3D path following and COLAV by an AUV. Specifically, the state-of-the-art learning algorithm PPO was used to train the neural networks. In addition, the learning framework *curriculum learning* was used: The agents started small by learning path following, and then was introduced to progressively more complex maneuvering tasks as obstacle density was increased. The final stage of training also implemented an ocean current perturbing the AUV's motion, which the agents ideally would compensate for.

The AUV was operated by commanding three actuator signals in the form of propeller thrust and rudder and elevator fin angles. A PI-controller maintained a desired cruise speed, while the DRL agent operated the control fins. The agent made decisions from observing the state variables of the dynamical model, control errors, the disturbances and through sensory inputs from an FLS.

To reiterate, the guiding questions governing the research were stated in section 1 as:

- Can the current state-of-the-art in DRL control be applied in end-to-end learning to achieve 3D path following by an AUV with 6-DOF?

The current state-of-the-art in DRL control has previously been seen applied to achieve 2D path following for AUVs. This research has extended use of the DRL framework to incorporate 3D curvature continuous path following by AUVs with the ability to affect elevation and course. It was observed that agents biased towards path following achieved the objective with an average error of $0.5m$ even in the presence of a perturbing ocean current, clearly indicating its utility in the 3D case for vehicles with 6-DOF and multiple control fins.

- Can the control system build in automatic collision avoidance and achieve intelligent decision-making regarding avoidance maneuvering?

Quantitative evaluation was performed using statistical averages by sampling $N = 100$ episodes per difficulty level and measuring the success rate (reaching the last waypoint within an acceptance radius without collision), collision rate and average tracking error. By giving the agents the ability to perceive the environment through an FLS and providing the right incentives, it was observed that the agents biased towards COLAV demonstrated great obstacle avoidance under ideal conditions. The best agent accomplished zero collisions out of 300 samples without an ocean current and 3 out of 100 with.

The DRL controllers were also tested in special-purposed scenarios to investigate the quality of path following in the special cases where no objects are

restricting the path, optimal use of actuators in extreme obstacle configurations and in a dead-end test. Testing showed that the agents indeed had learned to maneuver the AUV effectively applying most control action in the unobstructed direction when encountering extreme obstacle configurations. Moreover, the agents with less incentive to stay on-path, managed to escape the local minima trap involved in the dead-end challenge. Hence, the results indicates that the agents had acquired enough general knowledge about the system, to make intelligent decisions when faced with novel situations.

- How does the reward function affect the learned control strategy and is there a clear link to the incentives provided?

A reward system based on quadratic penalizations was designed to incentivize the agent to follow the path, but also be willing to deviate if further on-path progress was unsafe. In addition, avoiding excessive roll and use of control actuation was avoided by penalizing such behaviour. As path following and avoiding collisions are competing objectives, the agent must trade-off one for the other in order to achieve a successful outcome in an episode. Since this trade-off is non-trivial, a regulating parameter λ_r was introduced and tuned with three different values to observe behavioural outcome.

Both the quantitative and qualitative evaluation confirmed the intended relationship between behavioural outcome and the trade-off regulation parameter. In addition, the training history revealed differences in adaptability and exploration/exploitation as the learning process advanced. The implications of this finding is that specific incentives makes the agents more prone to certain weaknesses, which then should be addressed when setting up the learning process.

Finally, the thesis has discussed the strengths and weaknesses of the approach, and improvements and further paths to explore were suggested. The results obtained realized the research goals set and indicates that RL could play a part in achieving truly autonomous vehicles capable of human-level decision-making.

8 Bibliography

Ahmed, Z., Roux, N. L., Norouzi, M. and Schuurmans, D. (2018), 'Understanding the impact of entropy on policy optimization'.

Ataei, M. and Yousefi-Koma, A. (2015), 'Three-dimensional optimal path planning for waypoint guidance of an autonomous underwater vehicle', *Robotics and Autonomous Systems* **67**, 23 – 32. Advances in Autonomous Underwater Robotics.
URL: *http://www.sciencedirect.com/science/article/pii/S0921889014002279*

Bengio, Y., Louradour, J., Collobert, R. and Weston, J. (2009), Curriculum learning, *in* 'Proceedings of the 26th Annual International Conference on Machine Learning', ICML '09, Association for Computing Machinery, New York, NY, USA, p. 41–48.
URL: *https://doi.org/10.1145/1553374.1553380*

Breivik, M. and Fossen, T. I. (2009), Guidance laws for autonomous underwater vehicles, *in* A. V. Inzartsev, ed., 'Underwater Vehicles', IntechOpen, Rijeka, chapter 4.
URL: *https://doi.org/10.5772/6696*

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016), 'Openai gym'.

Carlucho, I., Paula, M. D., Wang, S., Petillot, Y. and Acosta, G. G. (2018), 'Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning', *Robotics and Autonomous Systems* **107**, 71 – 86.
URL: *http://www.sciencedirect.com/science/article/pii/S0921889018301519*

Carroll, K. P., McClaran, S. R., Nelson, E. L., Barnett, D. M., Friesen, D. K. and William, G. N. (1992), Auv path planning: an a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones, *in* 'Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology', pp. 79–84.

Cashmore, M., Fox, M., Larkworthy, T., Long, D. and Magazzeni, D. (2014), Auv mission control via temporal planning, *in* '2014 IEEE International Conference on Robotics and Automation (ICRA)', pp. 6535–6541.

Chang, S.-R. and Huh, U.-Y. (2015), 'Curvature-continuous 3d path-planning using qpmi method', *International Journal of Advanced Robotic Systems*

12(6), 76.
URL: *https://doi.org/10.5772/60718*

Chu, Z. and Zhu, D. (2015), 3d path-following control for autonomous underwater vehicle based on adaptive backstepping sliding mode, *in* '2015 IEEE International Conference on Information and Automation', pp. 1143–1147.

da Silva et al., J. E. (2007), 'Modeling and simulation of the lauv autonomous underwater vehicle', *Conference Paper: 13th IEEE IFAC International Conference on Methods and Models in Automation and Robotics* .

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y. and Zhokhov, P. (2017), 'Openai baselines', `https://github.com/openai/baselines`.

Doll, B. B., Simon, D. A. and Daw, N. D. (2012), 'The ubiquity of model-based reinforcement learning', *Current Opinion in Neurobiology* **22**(6), 1075 – 1081. Decision making.
URL: *http://www.sciencedirect.com/science/article/pii/S0959438812001316*

Egeland, O. and Gravdahl, J. T. (2002), *Modeling and Simulation for Automatic Control*, Marine Cybernetics.

Encarnacao, P. and Pascoal, A. (2000), 3d path following for autonomous underwater vehicle, *in* 'Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)', Vol. 3, pp. 2977–2982 vol.3.

Eriksen, B. H., Breivik, M., Pettersen, K. Y. and Wiig, M. S. (2016), A modified dynamic window algorithm for horizontal collision avoidance for auvs, *in* '2016 IEEE Conference on Control Applications (CCA)', pp. 499–506.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C. and Legg, S. (2017), 'Noisy networks for exploration'.

Fossen, T. (2011), *Handbook of Marine Craft Hydrodynamics and Motion Control*, John Wiley & Sons.

Fossen, T. I. (2000), 'Nonlinear Passive Control and Observer Design for Ships', *Modeling, Identification and Control* **21**(3), 129–184.

Fox, D., Burgard, W. and Thrun, S. (1997), 'The dynamic window approach to collision avoidance', *IEEE Robotics Automation Magazine* **4**(1), 23–33.

Garau, B., Alvarez, A. and Oliver, G. (2005), Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an a* approach, *in* 'Proceedings of the 2005 IEEE International Conference on Robotics and Automation', pp. 194–198.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A. and Bengio, Y. (2013), 'An empirical investigation of catastrophic forgetting in gradient-based neural networks'.

Haugen, F. (2008), 'Derivation of a discrete-time lowpass filter', `http://techteach.no/simview/lowpass_filter/doc/filter_algorithm.pdf`.

Havenstrøm, S. T. (2020), 3d path following and motion control for autonomous underwater vehicles using deep reinforcement learning, Project report in TTK4551, Faculty of Information Technology and Electrical Engineering, NTNU – Norwegian University of Science and Technology.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S. and Wu, Y. (2018), 'Stable baselines', `https://github.com/hill-a/stable-baselines`.

Joshua Achiam, Alex Ray, D. A. (2019), Benchmarking safe exploration in deep reinforcement learning.

Juliani, A. (2018), 'Maximum entropy policies in reinforcement learning & everyday life'.
URL: *https://medium.com/@awjuliani/maximum-entropy-policies-in-reinforcement-learning-everyday-life-f5a1cc18d32d*

Kavraki, L. E., Svestka, P., Latombe, J. . and Overmars, M. H. (1996), 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces', *IEEE Transactions on Robotics and Automation* **12**(4), 566–580.

Khalil, H. K. (2002), *Nonlinear Systems*, Pearson.

Levine, S. and Koltun, V. (2013), Guided policy search, *in* '30th International Conference on Machine Learning, ICML 2013'.

Liang, X., Qu, X., Wan, L. and Ma, Q. (2018), 'Three-dimensional path following of an underactuated auv based on fuzzy backstepping sliding mode control', *International Journal of Fuzzy Systems* **20**(2), 640–649.
URL: *https://doi.org/10.1007/s40815-017-0386-y*

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2015), 'Continuous control with deep reinforcement learning', *CoRR* **abs/1509.02971**.

Martinsen, A. B. and Lekkas, A. M. (2018*a*), Curved path following with deep reinforcement learning: Results from three vessel models, *in* 'OCEANS 2018 MTS/IEEE Charleston', pp. 1–8.

Martinsen, A. B. and Lekkas, A. M. (2018*b*), 'Straight-path following for underactuated marine vessels using deep reinforcement learning', *IFAC-PapersOnLine* **51**(29), 329 – 334. 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.
URL: *http://www.sciencedirect.com/science/article/pii/S2405896318321918*

McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R. and McEwen, R. (2008), A deliberative architecture for auv control, *in* '2008 IEEE International Conference on Robotics and Automation', pp. 1049–1054.

Menhaj, M. and Rouhani, M. (2002), A neuro-controller with guaranteed stability, *in* 'The 2002 45th Midwest Symposium on Circuits and Systems', Vol. 3, pp. III–33.

Meyer, E., Robinson, H., Rasheed, A. and San, O. (2020), 'Taming an autonomous surface vehicle for path following and collision avoidance using deep reinforcement learning', *IEEE Access* **8**, 41466–41481.

Mikko (2018), 'A comprehensive list of hyperparameter optimization & tuning solutions'.
URL: *https://medium.com/@mikkokotila/a-comprehensive-list-of-hyperparameter-optimization-tuning-solutions-88e067f19d9*

Nielsen, M. A. (2015), *Neural Networks and Deep Learning*, Determination Press.

Puterman, M. L. (2014), *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I. and Abbeel, P. (2015), 'Trust region policy optimization'.

Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P. (2015), 'High-dimensional continuous control using generalized advantage estimation'.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017), 'Proximal policy optimization algorithms', *CoRR* **abs/1707.06347**.
URL: *http://arxiv.org/abs/1707.06347*

Sugihara, K. and Yuh, J. (1996), Ga-based motion planning for underwater robotic vehicles, *in* 'Proc. 10th International Symp. on Unmanned Untethered Submersible Technology. Autonomous Undersea Systems Institute', pp. 406–415.

Sutton, R. and Barto, A. (2018), *Reinforcement Learning*, MIT Press.

Sutton, R. S., McAllester, D., Singh, S. and Mansour, Y. (1999), Policy gradient methods for reinforcement learning with function approximation, *in* 'Proceedings of the 12th International Conference on Neural Information Processing Systems', NIPS'99, MIT Press, Cambridge, MA, USA, pp. 1057–1063.
URL: *http://dl.acm.org/citation.cfm?id=3009657.3009806*

Tai, L., Zhang, J., Liu, M., Boedecker, J. and Burgard, W. (2016), 'A survey of deep network solutions for learning control in robotics: From reinforcement to imitation'.

Tan, C. S. (2006), A Collision Avoidance System for Autonomous Underwater Vehicles, PhD dissertation, University of Plymouth.

Waltz, M. and Fu, K. (1965), 'A heuristic approach to reinforcement learning control systems', *IEEE Transactions on Automatic Control* **10**, 390–398.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D. and Wierstra, D. (2017), 'Imagination-augmented agents for deep reinforcement learning'.

Wen Yu and Xiaoou Li (2000), Passive properties of dynamic neural networks, *in* 'Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)', Vol. 2, pp. 1445–1449 vol.2.

Wiig, M. S., Pettersen, K. Y. and Krogstad, T. R. (2018), A 3d reactive collision avoidance algorithm for nonholonomic vehicles, *in* '2018 IEEE Conference on Control Technology and Applications (CCTA)', pp. 67–74.

Williams, G. N., Lagace, G. E. and Woodfin, A. (1990), A collision avoidance controller for autonomous underwater vehicles, *in* 'Symposium on Autonomous Underwater Vehicle Technology', pp. 206–212.

Woo, J., Yu, C. and Kim, N. (2019), 'Deep reinforcement learning-based controller for path following of an unmanned surface vehicle', *Ocean Engineering* **183**, 155 – 166.
URL: *http://www.sciencedirect.com/science/article/pii/S0029801819302203*

Xiang, X., Yu, C. and Zhang, Q. (2017), 'Robust fuzzy 3d path following for autonomous underwater vehicle subject to uncertainties', *Computers & Operations Research* **84**, 165 – 177.
URL: *http://www.sciencedirect.com/science/article/pii/S0305054816302374*

Xu, H., Caramanis, C. and Mannor, S. (2009), 'Robustness and regularization of support vector machines', *Journal of Machine Learning Research* **10**(51), 1485–1510.
URL: *http://jmlr.org/papers/v10/xu09b.html*

Yann LeCun, Leon Bottou, G. B. O. and Müller, K.-R. (1998), *Efficient BackProp*, Springer, Berlin, Heidelberg.

Yoon, C. (2019), 'Understanding actor critic methods and a2c'.
URL: *https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f*

Yu, R., Shi, Z., Huang, C., Li, T. and Ma, Q. (2017), Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle, *in* '2017 36th Chinese Control Conference (CCC)', pp. 4958–4965.

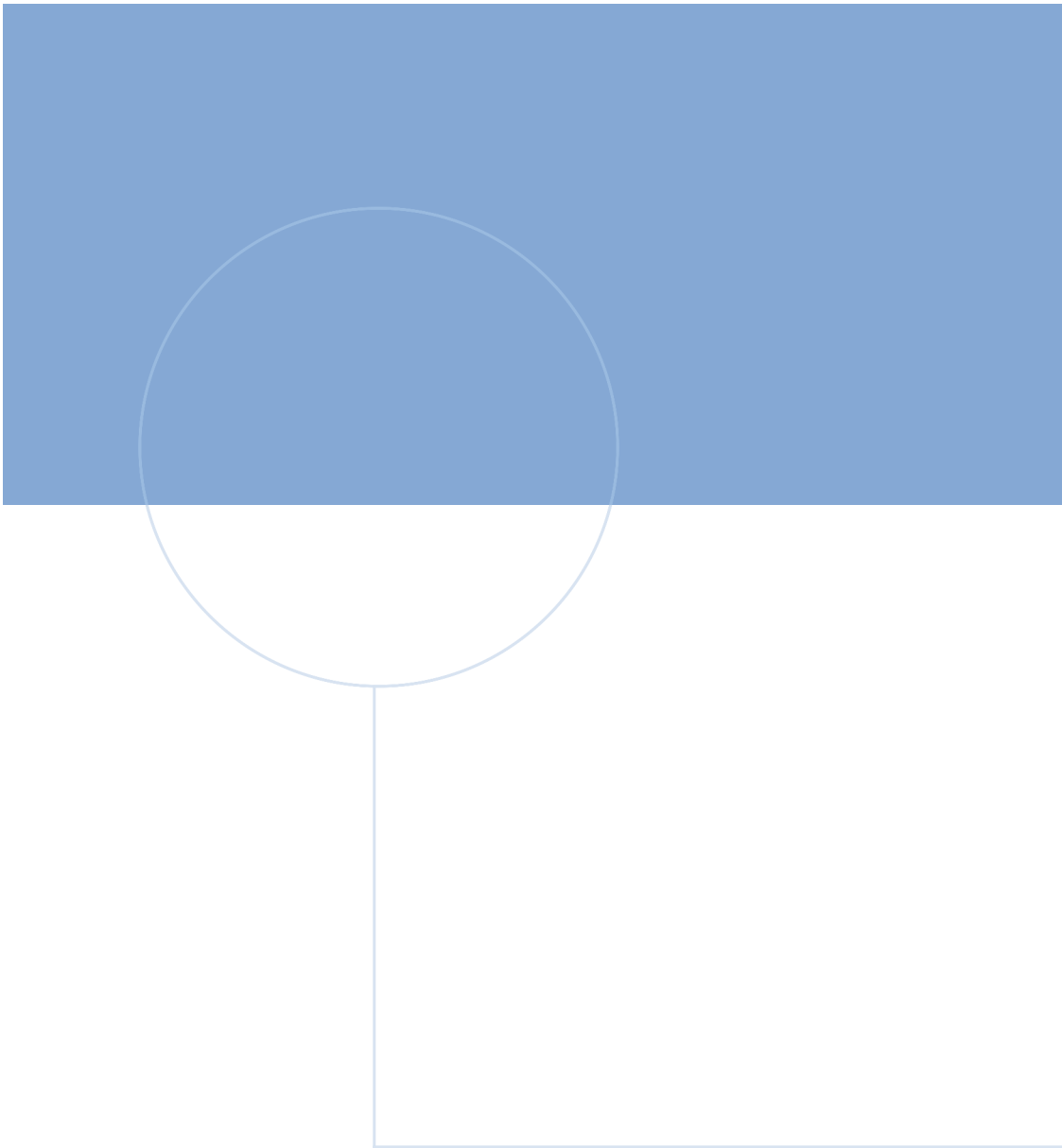
Zender, D. (2019), 'We analyzed 16,625 papers to figure out where ai is headed next'. [Online; posted 25-January-2019].
URL: *https://www.technologyreview.com/2019/01/25/1436/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/*

Zhang, L., Sun, X., Li, Y. and Zhang, Z. (2019), 'A noise-sensitivity-analysis-based test prioritization technique for deep neural networks'.

Appendix A AUV Model Parameters

Parameter	Description	Value
Mass & Coriolis Matrix		
m	Mass	18
Z_g	COG relative to CO	0.01
I_x	Moment of inertia - roll	0.0405
I_y	Moment of inertia - pitch	1.070
I_z	Moment of inertia - yaw	1.070
$X_{\dot{u}}$	Added mass - surge	-1.029
$Y_{\dot{v}}$	Added mass - sway	-16.153
$Z_{\dot{w}}$	Added mass - heave	-16.153
$K_{\dot{p}}$	Added mass - roll	0
$M_{\dot{q}}$	Added mass - pitch	-0.758
$N_{\dot{r}}$	Added mass - yaw	-0.758
Damping Matrix		
X_u	Linear damping - surge	-2.4
Y_v	Linear damping - sway	-23
Z_w	Linear damping - heave	-23
K_p	Linear damping - roll	-0.3
M_q	Linear damping - pitch	-9.7
N_r	Linear damping - yaw	-9.7
Y_r	Linear damping - yaw on sway	11.5
Z_q	Linear damping - pitch on heave	-11.5
M_w	Linear damping - heave on pitch	3.1
N_v	Linear damping - sway on yaw	-3.1
$X_{u\|u\|}$	Nonlinear damping - surge	-2.4
$Y_{v\|v\|}$	Nonlinear damping - sway	-80
$Z_{w\|w\|}$	Nonlinear damping - heave	-80
$K_{p\|p\|}$	Nonlinear damping - roll	-6.4e-4

$M_{q\|q\|}$	Nonlinear damping - pitch	-9.1
$N_{r\|r\|}$	Nonlinear damping - yaw	-9.1
$Y_{r\|r\|}$	Nonlinear damping - yaw on sway	0.3
$Z_{q\|q\|}$	Nonlinear damping - pitch on heave	-0.3
$M_{w\|w\|}$	Nonlinear damping - heave on pitch	1.5
$N_{v\|v\|}$	Nonlinear damping - sway on yaw	-1.5
Y_{uv_f}	Fin lift - sway	-19.2
Z_{uw_f}	Fin lift - heave	-19.2
M_{uq_f}	Fin lift - pitch	-3.072
N_{uq_f}	Fin lift - yaw	-3.072
Y_{ur_f}	Fin lift - yaw on sway	7.68
Z_{uq_f}	Fin lift - pitch on heave	-7.68
M_{uw_f}	Fin lift - heave on pitch	-7.68
N_{uv_f}	Fin lift - sway on yaw	7.68
Y_{uv_b}	Body lift - sway	-10.956
Z_{uw_b}	Body lift - heave	-10.956
M_{uw_b}	Body lift - heave on pitch	-3.309
N_{uv_b}	Body lift - sway on yaw	3.309
Restoring Force Matrix		
W	Weight	176.58
B	Buoyancy	177.58
Control Force Matrix		
$Y_{uu\delta_r}$	Rudder fin on sway	19.2
$N_{uu\delta_r}$	Rudder fin on yaw	7.68
$Z_{uu\delta_s}$	Elevator fin on heave	-19.2
$M_{uu\delta_s}$	Elevator fin on pitch	-7.68



NTNU

Kunnskap for en bedre verden