Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Brage Elias West Mothes

Reinforcement learning for autodocking of surface vessels

Forsterkende læring for auto-kailegging av overflatefartøy

Master's thesis in Marine Technology - Cybernetics
Supervisor: Roger Skjetne
June 2019

NTNU
Norwegian University of
Science and Technology

Brage Elias West Mothes

Reinforcement learning for autodocking of surface vessels

Forsterkende læring for auto-kailegging av overflatefartøy

Master's thesis in Marine Technology - Cybernetics
Supervisor: Roger Skjetne
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

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

MSC THESIS DESCRIPTION SHEET

Name of the candidate:	Mothes, Brage
Field of study:	Marine control engineering
Thesis title (Norwegian):	Forsterkende læring for auto-kailegging av overflatefartøy
Thesis title (English):	Reinforcement learning for autodocking of surface vessels

Background

For autonomous ships, the docking and takeoff maneuvers in a harbor area are especially critical and involve complex maneuvering by a skilled ship pilot. This involves understanding of:

- the ship dynamics (inertial delays, responses to strong currents, wind gusts, and propulsion),
- the hydrodynamic effects of using propulsion and rudders near harbor structures,
- the optimal entrance paths and speed regulation for good docking maneuvers, and
- the sensors, displays, and monitoring variables to use for necessary information feedback.

The objective of this thesis is to study how reinforcement learning can be used to make the vessel control system learn optimal docking maneuvers close to a quay by repeated (simulated) trials and errors on a selected set of action variables and according to specified rewards. The project shall as a start consider fully-actuated ships that have DP functionality as a basis for the maneuvering controller. Underactuated maneuvers is then possible to test by constraining the actuation in the thrust allocation. The thesis shall specifically consider appropriate action space definitions, reward functions, and analyze how the variations of the action spaces and reward functions affect the achieved performance.

The main goal of the thesis is not to construct a practical autodocking function, but rather to understand and gain insight on the reinforcement learning theory, how this can be applied to a marine application, and how to develop machine learning methods further as an action-planning guidance layer for autonomous ships.

Work description

1) Perform a background and literature review to provide information and relevant references on:
    - Practical theory around docking situations.
    - Relevant reinforcement learning (RL) theory and methods for path planning.
    - Autonomous system architecture and layers for marine systems.
    - The maneuvering control theory applied for DP and path following.

   Write a list with abbreviations and definitions of terms, explaining relevant concepts related to the literature study and project assignment.

2) Formulate a case study of the final docking operation:
    a) Choose and present a vessel simulation model as a case study.
    b) Design the underlying (low-level) maneuvering control algorithm based on assumed reference inputs to be provided by the RL function.
    c) Formulate the machine learning problem by definition of inputs, monitoring variables, action space definition, and relevant terms of the reward function.

3) Investigate different setups of RL by:
    a) pre-specifying the desired docking path and heading, while letting RL trim the speed along the path (including a mechanism for coming to full stop at target quay location);
    b) letting the RL stepwise command path segments to learn good docking paths, while letting speed and heading along the path be given;
    c) tentatively combine some of these setups, by expanding the action space (if time permits).

   Provide assumptions, design the action space(s), the reward function(s), implement the RL methods, perform simulations and illustrate the responses, and discuss the results. The studies should include sensitivity-like analysis of variations in relevant parameters of the reward function and action space.

Specifications

The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the various steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 70 A4-pages, 100 B5-pages, from introduction to conclusion, unless otherwise agreed upon. It shall be written in English (preferably US) and contain the following elements: Title page, abstract, acknowledgements, thesis specification, list of symbols and acronyms, table of contents, introduction with objective, background, and scope and delimitations, main body with problem formulations, derivations/developments and results, conclusions with recommendations for further work, references, and optional appendices. All figures, tables, and equations shall be numerated. The original contribution of the candidate and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. *natbib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct. Such practice is taken very seriously by the university and will have consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with an electronic copy to the main supervisor and department according to NTNU administrative procedures. The final revised version of this thesis description shall be included after the title page. Computer code, pictures, videos, dataseries, etc., shall be included electronically with the report.

Start date: January, 2019 **Due date:** As specified by the administration.

Supervisor: Roger Skjetne
Co-advisor(s): N/A

Trondheim, 13.04.2019

Roger Skjetne

Digitally signed by Roger Skjetne
Date: 2019.04.13 21:30:05 +02'00'

Roger Skjetne
Supervisor

Preface

This master thesis is composed spring 2019, and stands as a continuation of the master project written fall 2018. The work is a concluding part of a Master of Science degree in Marine Cybernetics at NTNU. The work includes relevant control theory and path planning methods, as well as an introduction to the field of machine learning. Following this a variety of techniques and approaches for reinforcement learning applied in a docking situation have been tested.

The work has been challenging, but definitely interesting and rewarding. Many hours have been spent on literature studies and experimenting in Matlab and Simulink, combining reinforcement learning with marine control systems. In general my knowledge within the field of machine learning has increased substantially, together with my programming skills in Matlab.

Readers of the thesis should preferably have knowledge of basic hydrodynamics, marine cybernetics and control theory. Knowledge within the field of reinforcement learning is also beneficial, however theory required to apprehend this is presented.

[illegible]

Brage Elias West Mothes
Trondheim 2019-06-11

Acknowledgement

Great assistance has laid the grounds for much of this thesis. I would like to thank a number of people for their contributions and assistance throughout the year.

First of all I would like to thank my supervisor, Professor Roger Skjetne, for providing guidance as well as relevant resources around both theory and building blocks for the simulations.

Following this I have received great help from PhD Candidates Jon Bjørnø and Einar Skiftestad Ueland. These have offered consistent assistance and always been available for questions if needed. This accounts specially to Jon Bjørnø in terms of helping with the vessel model applied.

Lastly I would like to thank my office mates for providing great discussions and encouragement, as well as a great final year at NTNU.

Abstract

This thesis presents a study of machine learning (ML) applied to the marine field. In specific reinforcement learning (RL) applied as an action-planning guidance layer in a docking scenario has been inspected. This was done through an analysis of the control systems ability to learn optimal docking maneuvers by trial and error simulations close to the quay. The underlying intention is to build understanding of RL for marine applications.

A model based on CyberShip Arctic Drillship operating in the Marine Cybernetics Laboratory at NTNU, has served as the platform for testing RL in the docking scheme. The buildup of this model is presented as background for the thesis, along with relevant theory of autonomous systems, path planning, maneuvering, and machine learning. For simplicity, training of the RL agent was simulated for proposing speeds and entrance paths for the vessel individually.

Training in regards of speed, we were able to show how the agent could learn improved strategies based on various measures and concurrently comply with specified restrictions. As a model-free approach, needless of knowledge connected to underlying dynamics, these strategies could be learnt in accordance with different environmental settings. Simultaneously it showed how unconsidered strategies could arise, due to a known potential of RL agents to discover unexpected ways to obtain rewards. This could have positive effects, but also highlights the importance of careful reward design, as fields unaccounted for can be violated.

When evaluating entrance paths, backpropagation of rewards proved crucial in order to converge to satisfactory results, where available states showed dependency far back. Still, this successfully guided the agent towards long term rewards, enabling improved strategies accounting for the complete process rather than just the immediate optima. However, the agent showed susceptible to conflicting rewards, seemingly shrouding global optima and reducing stability of the learning progress. N-step backpropagation has been recommended as a possible improvement to this, along with accelerated learning.

The simulations all reflected the dependency of the agents behavior on action space definition and applied reward function. Additionally the extensiveness of the action space was found to restrict possible solutions of the agent, but extensiveness comes with greater computational demands. Based on the provided results, the model-free RL shows promising capabilities for vessel guidance in docking situations, allowing optimization according to desired monitoring variables.

Sammendrag

Denne avhandlingen presenterer et studie av maskinlæring (ML) anvendt i maritim sammenheng. Mer spesifikt har forsterkende læring (FL), som et handlings-planleggende lag for veiledning i et dokningsscenario blitt inspisert. Dette ble gjort gjennom en analyse av kontrollsystemets evne til å lære optimale kaileggings manøvre ved å prøve- og feile gjennom simuleringer nær kaien. Avhandlingen tjener til å bygge opp forståelse av RL for marine applikasjoner.

En modell basert på CyberShip Arctic Drillship som opererer i Marine Cybernetics Laboratory ved NTNU, har tjent som plattform for testing av FL i kaileggingsscenarioet. Oppbyggingen av denne modellen presenteres som bakgrunn for avhandlingen, sammen med relevant teori om autonome systemer, baneplanlegging, manøvrering og maskinlæring. For enkelhets skyld ble treningen av FL-agenten gjennomført ved å forslå hastigheter og inngangsbaner for fartøyet individuelt.

Gjennom trening med hensyn på fart kunne vi vise hvordan agenten kunne lære forbedrede strategier basert på ulike tiltak og samtidig overholde spesifiserte begrensninger. Som en modellfri tilnærming, uten behov for kunnskap knyttet til underliggende dynamikk, kunne disse strategiene læres i samsvar med ulike oppsett av terreng. Samtidig viser uforutsette strategier seg å oppstå som følge av et kjent potensial hos FL-agenter til å oppdage uventede måter å anskaffe belønning. Dette kan ha positive effekter, men understreker også betydningen av presis design av belønning, ettersom oversette farer kan oppstå.

Ved evaluering av inngangsbaner viste tilbake-forplantning av belønning seg avgjørende for konvergering, ettersom tilgjengelige tilstander viste avhengighet langt tilbake. Likevel veiledet dette agenten velykket mot langsiktige belønninger, slik at forbedrede strategier omfattet hele prosessen i stedet for bare den umiddelbare optima. Agenten viste imidlertid utsatt for motstridende belønninger, som tilsynelatende maskerte globale optima og reduserte stabiliteten i læringsprosessen. N-step tilbake-forplantning har blitt anbefalt en mulig forbedring til dette, sammen med akselerert læring.

Simuleringene gjenspeiler avhengigheten av agentens oppførsel på definisjon av handlingsrom, og anvendt belønningsfunksjon. I tillegg ble omfanget av handlingsrom funnet å begrense mulige løsninger, men med utvidelsesevne kommer med større beregningsbehov. Basert på resultatene, viser modellfri FL lovende evner for fartøyets veiledning i kaileggingssituasjoner, i forhold til optimalisering i henhold til ønskede overvåkingsvariabler.

Nomenclature

Abbreviations

CLF	Control Lyapunov Function
CO	Coordinate Origin
CSAD	CyberShip Arctic Drillship
DOF	Degree Of Freedom
DP	Dynamic Programming
DP	Dynamic positioning
I/O	Input/Output
LF	Low-frequency
MC	Monte Carlo
MC-lab	Marine cybernetics laboratory
MCQ-L	Modified Connectionist Q-Learning
MDP	Markov Decision Process
ML	Machine Learning
NED	North East Down
PC	Parametric Continuity
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
TD	Temporal Difference
USV	Unmanned surfave vessel
WF	Wave-frequency

Roman letters

A	Action space
a	Action
A_k	Coefficient vector
A_{dock}	Docking area
A_{path}	Action space for path evaluation
A_{speed}	Action space for speed evaluation
b	Bias
C_A	Coriolis force matrix
C_T	Terminal reward constant
C_t	Immediate reward constant
C_{RB}	Centripetal force matrix
d	Distance to docking position

e_t	Approximation of gradient
ep_i	Current episode
ep_{tot}	Total amount of episodes
err_p	Error tolerance in position
f	Transition model
$K_{1,2}$	Maneuvering tuning constants
l	Extension length path
L_F	Length full scale vessel
L_M	Length model scale vessel
P	Power
p	Vessel position
p_0	Departure waypoint
p_d	Parametrized path
p_t	Target waypoint
p_{dock}	Docking position
$P^a_{ss'}$	Transition probability
$p_{t,RL}$	Upcoming target waypoint, returned by the RL agent
Q	Q-value
R	Radius
R	Reward
r_T	Terminal reward component
r_t	Immediate reward component
S	State space
s	Global path parameter
s	State
s_T	Terminal state
T	Terminal step
t	Time
t_{delay}	Delay time
t_{max}	Time limit
$t_{p_0 \to p_t}$	Time elapsed along a path segment
u_{const}	Constant desired speed
u_{ref}	Discrete reference speed
V	Value function
V_θ	Approximated value function
V_c	Current velocity in NED frame
v_s	Speed profile
x	Longitudinal position

y	Latitudinal position
$z_{1,2}$	Backstepping parameters
$\widetilde{p}$	Offset in position
$\{s, a\}$	State action pair
D	Damping matrix
I_{ij}	Product of inertia relative to the ij axis
M_A	Added mass matrix
M_{RB}	Rigid body mass matrix
$R(\psi)$	Rotation matrix
u	Speed
u_d	Desired speed

Greek letters

α	Learning rate
α_1	Virtual control variable
$\alpha_{\psi_{offset},\Delta_{dist}}$	Tuning constant for reward prioritization
δ_t	Temporal difference
δ_u	Step size of speed augmentation
Δ_{dist}	Change in distance
δ_{θ_T}	Step size of path angle augmentation
ϵ	Agents tendency to value exploration
η	Vessel position
η_d	Desired vessel position
γ	Discount factor
κ	Maneuvering tuning constant
λ	Design constant for path slope
λ	Eligibility trace
λ	Scaling ratio
μ	Maneuvering tuning constant
∇	Gradient operator
ν	Linear velocities in body frame
ν_c	Current velocity in body frame
ν_r	Linear velocities relative to local current in body frame
ω	Backstepping parameter
ϕ	Feature vector
π	Policy
π^*	Optimal policy
ψ	Heading

ψ_d	Desired heading
ψ_{dock}	Docking heading
ψ_{offset}	Offset angle in heading
ρ	Tuning function
τ	Force
τ_{thr}	Thruster force
θ	Local path variable
θ	Parameter vector
θ_T	Average path tangential angle
$\widetilde{p}$	Offset in position

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Today's maritime society inhabits a growing interest for the implementation of autonomous systems and the development of unmanned surface vessels (USVs). These vessels are defined by their capability to perform tasks and missions without the need of a human operator. Such characteristics makes USVs suitable for operations where human presence is undesirable, but can also be used for increased precision or decreased cost (Tanakitkorn 2018). In terms of development and installation the relevant systems can be quite expensive and the realistic value has to be individually evaluated. However it is obvious that returned value for these systems rise with frequency of the application area. Such an event of frequent occurrence is a docking process.

The culmination of almost any voyage executed by a surface vessel, includes bringing the vessel to a stationary position adjacent a berth. In harbors this process often requires the on-bringing and collaboration with a harbor pilot possessing knowledge regarding the particular waterway such as depth, currents, and further local effects and hazards (Gard AS 2014). Due to safety concerns this collaboration tends to be mandated by law. Still many ship operators are critical to this requirement on the basis of cost efficiency and need due to late advancements in technology. However, even though availability and complexity of technology has shown exponential increase the last years, it is difficult to argue there exists a substitution for the local experience a harbor pilot can provide.

In another perspective however, nothing can replace experience in terms of maneuvering a specific vessel. The requirements related to harbors usually revolves around environmental factors that are hard to map, such as rotational and cross flowing currents. However, such effects, given specific factors as time of the day to account for tide, tends to hold certain patterns. Thus rises the question if necessary experience to properly maneuver the vessel in harbours also can be extended to autonomous systems. This places the focus on artificial learning systems such as machine learning, letting the vessel gather and integrate own experience for maneuvering in docking situations.

1.2 Objective and scope

The superior objective of this project is to study how reinforcement learning can be used to make the control system of an autonomous vessel learn optimal docking maneuvers. This is to be done through repeated trial and error simulations to build experience for the vessel maneuvering close to a quay, using selected action variables and fitting rewards. The project considers fully actuated vessels with a dynamic positioning basis for the maneuvering controller.

The intention is not to construct a practical autodocking function, but rather to generate understanding and insight in the field of machine learning applied in a marine setting. Specifically the use of reinforcement learning as an action-planning guidance layer is to be evaluated. On the basis of this, final numerical results from training is of less importance, but rather the observed possibilities of improvement and behaviour according to action spaces and reward functions. The systems is created to coincide with CyberShip Arctic Drillship (CSAD) carrying out docking maneuvers in the Marine cybernetics laboratory. All simulations are based upon this vessel and environment.

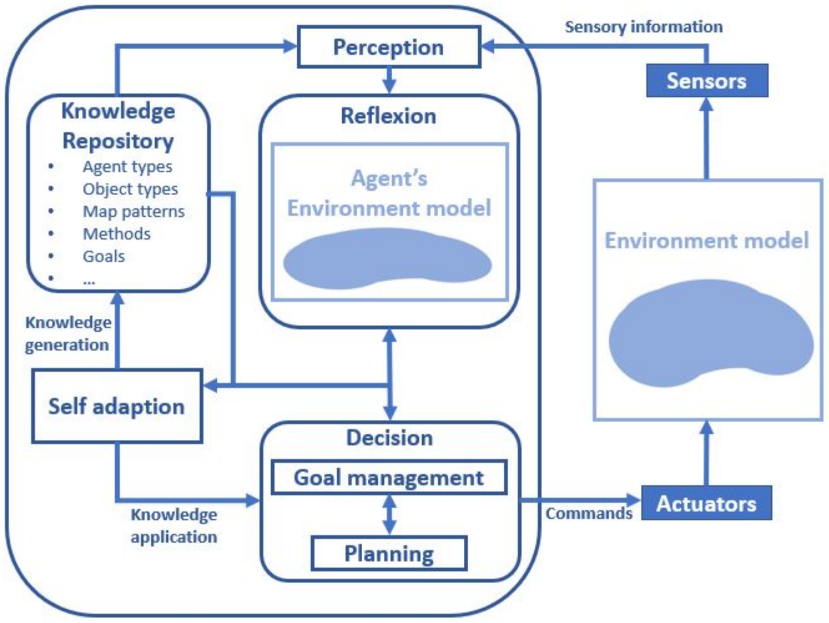
Chapter 2

Problem formulation

2.1 Autonomous systems for marine application

Autonomous systems can be defined as a class of intelligent systems with the ability to realize autonomous sensing, modeling, decision-making, and control in unfamiliar and dynamic environments (Xu et al. 2015). These consists of agents and objects sharing some common environment (Sifakis 2018). Agents inhabit the ability to observe and alter their state, while objects such as physical dynamic systems can not due to a lack of computational ability. However, objects can also undergo internal changes in state, for which agents can monitor and act on. Based on this, Sifakis (2018) proposes a computational model based off a combination of an agent architecture model and a system architecture model. The agent architecture specifies the coordination between five aspects of autonomy; Perception, Reflection, Goal Management, Planning and Self-adaption. This is depicted in Figure 2.1.1. On the other side the system architecture specifies the link between the agents and the objects involved, defining how the agents perceive the environment and elaborate the control strategy.

Based on the current state and changes in the environment, agents receive inputs and produce outputs according to their I/O (Input/Output) policy. In other words agents behave as controllers responding to the environment in order to achieve their specific goals. The actions produced in the agent environment are generally of two types: controllable actions imposed by the agent or uncontrollable actions due to internal state changes in the environment. In general, environment models are complex systems where generated plans for the specified goal can consist of infinite trees with alternating controllable and uncontrollable actions. In such situations a definite plan cannot be guaranteed. Instead it is common to rely on finite horizon plans computed on-line from the current state (Sifakis 2018). Naturally, increased dynamicity of the environment lay grounds for more unpredictable outcomes.



- *Perception*: Removing ambiguity and determining relevance of input.
- *Reflection*: Improving trustworthiness of agents environment model.
- *Goal Management*: Choosing suitable goals.
- *Planning*: Plan to achieve goals.
- *Self-adaption*: Adjusting behavior through learning.

Figure 2.1.1: General architectural model for agents. Showing the five main aspects of autonomy according to model and definitions simplified from Sifakis (2018).

The concept of system architecture can be seen as a set of rules defining the structure of a system and the interrelationships between it parts (Rødseth & Tjora 2014). With the general architectural model for agents in mind, it is clear that a specific system architecture depends on the system definition and it's level of autonomy. These levels are characterized subject to the degree of human-agent interaction, goal/mission complexity and environmental complexity. Ludvigsen & Sørensen (2016) present four system levels, ranging from remote control to highly autonomous:

1. **Automatic operation** - System operates automatically, with a human operator handling high-level mission planning functions.
2. **Management by consent** - System makes action recommendations, but awaits human consent under important parts of the mission. The system can also be delegated act on its own recommendations (human delegated).
3. **Semi-autonomous/management by exception** - System automatically executes functions on mission, under the circumstance of no human intervention due to response times. Human operator may alter parameters as well as cancel actions. Operator is prompted under critical or important exceptions previously defined for the mission.
4. **Highly autonomous** - System automatically executes mission-related functions in undefined environment. The system can on its own plan and replan missions, and work as an independent and "intelligent" unit.

With a focus on maritime operations, Ludvigsen & Sørensen (2016) propose a "bottom-up" approach to the system architecture, consisting of three layers shown in Figure 2.1.2. At the mission planner level, the objective is defined and the mission is planned accordingly. Missions may also be re-planned due to contingency along with new inputs from payload sensor data as well as new inputs from the autonomy layer. At the guidance and optimization level, waypoints and reference commands for the controller are handled. Finally the control execution level handles plant- and actuator control.

On the right hand side of the figure it can be seen how various autonomy strategies can be implemented to interact with the system architecture. One example here, relevant for this thesis, is to implement learning systems to operate in a more undefined, unstructured environment, gradually improving the systems perception and decision making.

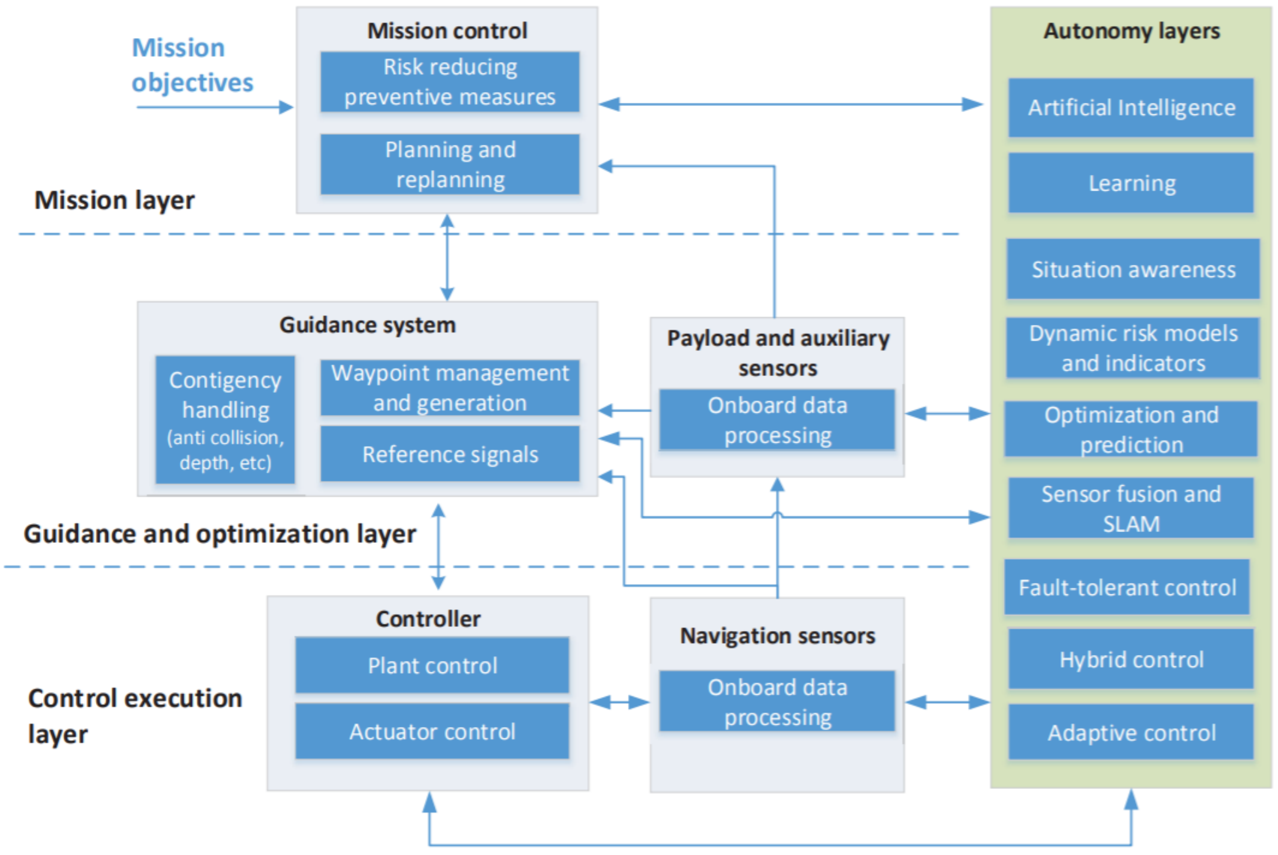


Figure 2.1.2: Control architecture of unmanned underwater vehicles as presented by Ludvigsen & Sørensen (2016).

2.2 Aspects of docking

A climax of most voyages is the finalizing steps of a controlled approach to a stationary berth, jetty or another vessel, better known as docking (Murdoch et al. 2014). In these situations of close quarter operations, understanding both the science and practice of ship maneuvering is essential. Although this is successfully handled on a daily basis, mistakes and accidents are frequent phenomenons resulting in injuries of both personnel and equipment.

In a practical context, Murdoch et al. (2014) presents a set of global rules for any such situation. These rules emphasize the importance of low speeds, controlled approach, and a well defined plan made ahead. Such a plan should include all expected maneuvers to be made by the vessel, along with possible effects from the environment. These plans must be made clear for the entire crew and other involved personnel, ensuring awareness of any risk likely to arise at different stages of the docking process. For vessel control, two elements of heightened importance is the momentary and upcoming headings and velocities of the vessel. Misunderstanding and carelessness towards these have lead to many accidents, especially due to their high connection to environmental effects in terms of both the point of influence and magnitude of the forces. As a general rule defined in Murdoch et al. (2014), speeds should be at the limited to a range necessary to maintain control. Changes in speed however can cause altered water flow over the vessel's control surfaces, especially affecting responsiveness of rudders and transverse thrusters. As a result, difficulties of maintaining control of the vessel may arise. Thus it is important to plan speed reductions in good time when bringing the vessel to a stop close to a berth.

Naturally, maneuvering characteristics varies from vessel to vessel, implying the importance of understanding the specific design. Two characteristics of high importance in the docking situation is the location of the vessel's pivot point and the vessel's underwater hull geometry, for understanding of how the vessel's course stability and rotational behaviour react due to applied forces.

Lateral docking

Perpendicular docking in a ferry-like manner is straightforward, as the vessel's heading of approach and final desired orientation is generally equal. Docking the vessel parallel to the quay however causes more difficulties, especially for underactuated vessels. In these situations Murdoch et al. (2014) proposes a sequence of steps to bring the vessel safely to rest. Approach the quay at an angle, applying astern thrust in order to turn the ship and bring it to rest parallel to the quay. Once stopped the vessel can be maneuvered into the right position using transverse thrust (if available) or by applying small kicks with an appropriate angle using a rudder. Again it's important to alter speed carefully ensuring a smooth landing at the berth side. For fully actuated vessels the problem is more simple, but for underactuated vessels using only rudder, achieving the right approach velocity and distance to the quay is a matter of precision and experience.

2.2.1 Effects of current

It is common for docking environments to include currents. These induce external forces on the vessel and are usually complex with directions and rates that can change with time and date (Murdoch et al. 2014). This increases the complexity of a docking situation, but can also be an advantage if used correctly. For example with current velocities parallel to a berth, the vessel can be directed bow to the current in order to create an advantage of higher relative speed between current and vessel while maintaining low ground speed. For vessels equipped with rudders this enhances steering by higher water flows over the control surface. In addition retardation is easier due to increased frontal resistance.

Another advantage is the possibility of using the current to alter the transverse distance between vessel and berth, by putting the vessel at an angle to the current, using it push the vessel sideways. This is shown in Figure 2.2.1. However, when the distance between the vessel and the berth becomes small, head currents can cause restricted inshore flow putting the vessel under interactive forces. These can lead to a pushing or pulling force respective to the quayside. Additionally, care should be taken when coming to low speeds so the increased relative velocity across the hull does not carry the vessel close to obstructions as seen in Figure 2.2.1. Lastly, counter currents can develop close to the bank, flowing in the opposite direction of the main current and causing sudden changes in forces (Murdoch et al. 2014). This as well as many of the other phenomenon can only be anticipated from local knowledge.

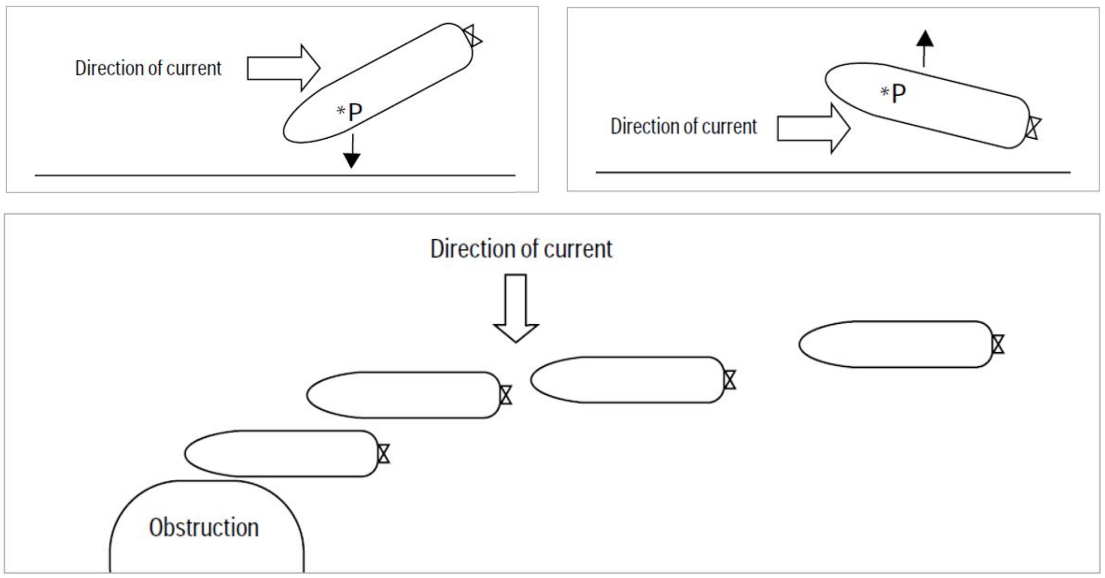


Figure 2.2.1: Representation of current effects in the docking process, as presented by Murdoch et al. (2014).

Usually, resulting magnitude and and direction of currents is dependent on vessel speeds. Predictions of these should be used with great caution, making generous allowance in distance and speeds, as mistakes are often impossible to correct (Murdoch et al. 2014).

2.3 Vessel model

In order to gauge realistic responses of a docking situation, simulations applying a high fidelity vessel model is needed. For this purpose a simulation model based on *CyberShip Arctic Drillship (CSAD)*, a 1:90 scale model of the vessel Statoil Cat I Arctic Drillship is proposed. This vessel is equipped with a total of 6 azimuth trusters, making it fully actuated, chosen due to the advantage in maneuvering this holds over underactuated vessels and it's availability of key personnel. The 6 degree of freedom (DOF) dynamic positioning (DP) model is a zero speed model with fluid memory, and has been developed using software as ShipX and MSS toolbox to create a total Simulink block as described in Bjørnø (2016). Additionally the thrust allocation for CSAD is developed for a 3 DOF control design model, as described in Frederich (2016). The mathematical model of CSADs process plant is found to account for the following effects:

- Inertial forces
- Hydrodynamic added mass and restoring forces

- Linear viscous damping
- Nonlinear cross-flow drag and surge resistance
- Fluid memory effects

The motion of marine vessels is defined by two distinct dynamics combinable through superposition, namely the low-frequency (LF) model and the wave-frequency (WF) model (Fossen 2016). Assuming an absence of both waves and wind, dynamics can be expressed as the through the LF model as:

$$M_{\mathrm{RB}}\dot{\nu} + M_A\dot{\nu}_r + C_{\mathrm{RB}}(\nu)\nu + C_{\mathrm{A}}\left(\nu_r\right)\nu_r + D\left(\nu_r\right)\nu_r = \tau_{\mathrm{thr}} \tag{2.1}$$

where $\nu = [u, v, w, p, q, r]^\top$ current effects are embedded through the relative velocity ν_r. Further terms are described in the next section. In the model the fluid memory effects is accounted for through a convolution integral including the vessel's retardation function. The cross-flow drag is calculated by representing the cross flow as a 2D problem through strip theory. Due to the connection of yaw rate and and cross-flow velocity the coupled sway-yaw motion is analyzed based on an assumption of small motions. Summarized the process plant of the Simulink model can be found to apply the following equations for calculating hydrodynamic effects:

$$\begin{aligned}
R_u &= \frac{1}{2} C_D \rho \left|u_r\right| u_r A_f, && \text{Surge resistance} \\
v_{r,tot} &= v_r - x_i \dot{r}, && \text{Relative velocity between strip and water} \\
dF_{2,CF}^{Strip} &= \frac{1}{2} C_D \rho \left|v_{r,tot}\right| v_{r,tot} D dx, && \text{Cross-flow drag on single strip} \\
F_{2,CF} &= \int_{-\frac{L_{pp}}{2}}^{\frac{L_{pp}}{2}} dF_{2,CF}^{Strip}, && \text{Force in sway} \\
F_{6,CF} &= \int_{-\frac{L_{pp}}{2}}^{\frac{L_{pp}}{2}} x dF_{2,CF}^{Strip}, && \text{Moment in yaw}
\end{aligned} \tag{2.2}$$

2.3.1 Simplified control design model

For regular docking processes in a harbour area it is natural to assume calm waters and low-speeds. A surface vessel operating in these conditions can thus be approximated as a 3 DOF process. Neglecting motions in roll, pitch and heave, that is $w = p = q = 0$, the state vectors are denoted as $\nu = [u, v, r]^T$ and $\eta = [x, y, \psi]^\top$ for linear velocity in the body frame, and position in the North East Down (NED) frame respectively. Such a 3 DOF model operating in the horizontal plane is presented in (Fossen 2016). For a surface vessel we have the general rigid-body kinetics:

$$\dot{\eta} = R(\psi)\nu \tag{2.3}$$

$$M_{RB}\dot{\nu} + M_A\dot{\nu}_r + C_{RB}(\nu)\nu + C_A(\nu_r)\nu_r + D(\nu_r)\nu_r = \tau_{thr} \tag{2.4}$$

Where $\nu_r := \nu - \nu_c$ is the relative velocity of the vessel with respect to the ocean current velocity ν_c, given in body coordinates. Here $\nu_c = R^\top(\psi)V_c := [u_c, v_c, r_c]^\top$, where $V_c = [V_x, V_y, V_r]^\top$ is the ocean current velocity in inertial coordinates. Further M_A and M_{RB} denotes rigid body mass and added mass matrices respectively, while C_{RB} and C_A denotes the Coriolis force and centripetal force matrices. Describing a single principle rotation about the z axis, the systems rotation matrix is defined as:

$$R(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Under an assumption of homogeneous mass distribution and xz-plane symmetry we have $I_{xy} = I_{yx} = 0$. Letting the body coordinate origin (CO) be set at the point in the vessel's center line, i.e located at a point $(x_g^*, 0)$ in the horizontal plane. Through these two assumptions along with an assumption that the added mass matrix is computed in CO, matrices and matrices of the defined kinetics reduce to:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & mx_g \\ 0 & mx_g & I_z \end{bmatrix} > 0 \qquad M_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} > 0 \tag{2.6}$$

$$C_{RB}(\nu) = \begin{bmatrix} 0 & 0 & -m(x_g r + v) \\ 0 & 0 & mu \\ m(x_g r + v) & -mu & 0 \end{bmatrix} \tag{2.7}$$

$$C_A(\nu_r) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v_r + Y_{\dot{r}}r_r \\ 0 & 0 & -X_{\dot{u}}u_r \\ -Y_{\dot{v}}v_r - Y_{\dot{r}}r_r & X_{\dot{u}}u_r & 0 \end{bmatrix} \tag{2.8}$$

Additionally, the damping matrix D can be defined as:

$$D(\nu_r) = -\begin{bmatrix} X_u + X_{|u|u}|u_r| + X_{uuu}u_r^2 & 0 & 0 \\ 0 & Y_v + Y_{|v|v}|v_r| + Y_{vvv}v_r^2 & Y_r + Y_{|r|r}|r_r| + Y_{rrr}r_r^2 \\ 0 & N_v + N_{|v|v}|v_r| + N_{vvv}v_r^2 & N_r + N_{|r|r}|r_r| + N_{rrr}r_r^2 \end{bmatrix} \tag{2.9}$$

Assuming slowly varying current velocities, meaning $\dot{\nu}_c \approx 0 \Rightarrow \dot{\nu}_r \approx \dot{\nu}$ we combine the two mass matrices into a general one $M := M_{RB} + M_A$. Hence,

$$M = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & mx_g - Y_{\dot{r}} \\ 0 & mx_g - Y_{\dot{r}} & I_z - N_{\dot{r}} \end{bmatrix} \tag{2.10}$$

This simplifies our equation in 2.11 to:

$$M\dot{\nu}_r = -C_{RB}(\nu)\nu - (C_A(\nu_r)\nu_r + D(\nu_r)\nu_r) + \tau_{thr} \tag{2.11}$$

Which can be restructured for the use of controlled change in vessel speed as:

$$\dot{\nu}_r = M^{-1}(-C_{RB}(\nu)\nu - (C_A(\nu_r)\nu_r + D(\nu_r)\nu_r) + \tau_{thr}) \tag{2.12}$$

Further simplifications due to an assumption of low speed maneuvering lets us ignore the influences of the Coriolis-centripetal matrices. Additionally we introduce a bias $b \in \mathbb{R}^3$ which for simplicity is assumed known (generally through use of an observer), leaving us with the model:

$$\dot{\nu}_r = M^{-1}(-D(\nu_r)\nu_r + \tau_{thr} + R(\psi)^\top b) \tag{2.13}$$

Relevant parameters are presented in Appendix A.1.

2.3.2 Fully actuated surface vessels

When dealing with control objectives it's important to be aware of the different control forces available and the number of coordinates needed to specify the state of the system. In specific it is important to distinguish between fully actuated vessels and underactuated vessels. Full actuation means that independent control forces and moments are simultaneously available in all directions, and is the case considered in this thesis. Moreover the following definition given in Fossen (2016) holds:

Definition 1 (Fully Actuated Marine Craft)
A marine craft is fully actuated if it has equal or more control inputs than generalized coordinates.

DOF is the set of displacements and rotations that together describes the change of a vessel's position and orientation (Fossen 2016). In other words, making a vessel operating in 3 DOF fully actuated involves equipment of actuators able to produce forces and moments in all directions of the state space. For such a vessel inhabiting the simplified rigid-body kinetics specified in equation

2.13 the thruster dynamics will consist of three independent forces specified as:

$$\tau_{thr} = \begin{bmatrix} \tau_u \\ \tau_v \\ \tau_r \end{bmatrix} \tag{2.14}$$

Relevant thrust parameters before and after scaling can be found in Appendix A.2. From here on τ_{thr} will be presented through the general term τ.

2.4 The path planning problem

Path planning refers to the construction of pre-defined paths to be assigned to a system in order to fulfill a specific mission (Lekkas 2014). Paths are usually designed based on a series of properties depending on the constraints of the system originating from both the vessel and the environment in which it navigates. Examplewise this involves keeping the vehicle away from hazardous situations through collision avoidance by taking constraints as the vessel's maneuverability and velocity into account. In general the path planning process consists of two main steps:

1. Determining waypoints to be reached.
2. Generating a feasible path to reach the waypoints.

Various techniques exist to fulfill these goals, ranging from simple straight line paths to more advanced paths of higher polynomial order with greater continuity. A useful criteria for evaluating paths is the level of smoothness. This property is directly linked to the vessel's dynamic constraints, and is therefore of high importance. A useful way to quantify this is by through the paths parametric continuity (PC), i.e the speed and orientation by which the path parameters propagate. The degree of PC is defined in different levels, with C^0 implying geometric continuity of the path segments, C^1 implying continuity of the velocity vector, C^2 implying acceleration is continuous and so on.

The complexity of the path planning problem increases with degrees of freedom, and is known to be non deterministic polynomial time-hard (Marin-Plaza et al. 2018). The overall aim is generally not only to find a single connection from the first to the last waypoint, but finding an optimal path yielding smooth maneuvers and fulfilling goals such as short distances. Thus again implying the importance of PC. Often the problem is divided into global and a local setting. Global planning focuses on finding a total solution, taking the entire static operation space into account. The drawback however is the need of high computational power. Moreover many algorithms in this field, such as shortest path, are not necessary compliant with vehicle constraints. Local planning is therefore usually implemented, continuously adjusting the path to more suitable segments according to vehicle constraints.

Impressive advances has been made within the field of path planning the last decades, however Lekkas (2014) specifies that there is still a large number of unsolved problems within the field. One of these is the uncertainty of the real world, including dynamic constraints of the vehicle and environmental forces such as wind and current.

2.4.1 Path generation

Following evaluation of a set of successive $k+1$ target-waypoints, a feasible trajectory has to be generated for tracking. Such waypoints are in general described by the desired vessel states in Cartesian coordinates for position (x, y), along with the vessel's heading angle ψ and speed u (Fossen 2016). To guide the vessel's between these states we take use of a time-parametrized trajectory.

Definition 2 (Parametrized path)
A parametrized path is defined as a geometric curve $p_d(s) \in \mathbb{R}^n$ with $n \geq 1$ parametrized by a continuous path variable s (Fossen 2016)

A sufficiently differentiable path $p_d(s) = [x_d(s), y_d(s)]^\top \in \mathbb{R}^2$ with C^r parametric continuity can be created through first dividing the total path from the first to the last waypoint into a series of n desired subpaths between succeeding waypoints $p_{d,i}(s)|i = 0, 1, ..., n$. For the PC requirements to be fulfilled, the connection between two succeeding subpaths must inherit the following property for $k = 0, 1, ..., r$ as presented by Skjetne (2005):

$$\lim_{s \nearrow i-1} p^{s^k}_{d,i-1} = \lim_{s \searrow i-1} p^{s^k}_{d,i} \tag{2.15}$$

To achieve this, one option proposed by Fossen (2016) is interpolating a spline between the waypoints according to polynomials of order k:

$$p_{d,i}(s) = \sum_{k=0}^{r} A_{k,i} s^k = A_{r,i} s^r + ... + A_{1,i} s + A_{0,i} \tag{2.16}$$

where the coefficient vectors $A_{k,i} = [a_{k,i}, b_{k,i}]^\top$ have to be determined. This can be accomplished through several methods. To ensure continuity at the connection points of the subpaths however, the best approach is to calculate these for each subpath individually and assign common numerical values at the connections. According to Skjetne (2019*a*) most numerically powerful way to do so is to continuously parametrize each subpath within a fixed interval. We do this by introducing a local path variable $\theta \in [0, 1)$ for each individual path segment $i \in I$. We then can employ the following equations to calculate the coefficients, ensuring continuity at the connections between the path segments and resulting in the final hybrid parametrization of the path $\hat{p}_d(i,\theta)$:

$$\begin{aligned} C^0 &: p_{d,i}(0) = p_i, & p_{d,i}(1) &= p_{i+1} \\ C^1 &: p^s_{d,i}(0) = \lambda(p_{i+1} - p_{i-1}), & p^s_{d,i}(1) &= \lambda(p_{i+2} - p_i) \\ C^j &: p^{s^j}_{d,i}(0) = 0, & p^{s^j}_{d,i}(1) &= 0 \end{aligned} \tag{2.17}$$

$$\hat{p}_d(i,\theta) = \begin{bmatrix} x_{d,i}(\theta) \\ y_{d,i}(\theta) \end{bmatrix} \tag{2.18}$$

Where the constant $\lambda > 0$ is a design constant for the slope. The slopes at the first and last waypoint are sat to the tangent between waypoints 0 and 1, and waypoints $n-1$ and n, respectively. Lastly to conform with requirements of a continuous parametrization, we let $i = \lfloor s \rfloor + 1$ and $\theta = s - \lfloor s \rfloor$, giving us the continuous C^r map:

$$s \mapsto p_d(s) := \hat{p}_d(i(s), \theta(s)) \tag{2.19}$$

Figure 2.4.1 shows the output of a parametrized path with C^2 continuity as presented by Fossen (2016):

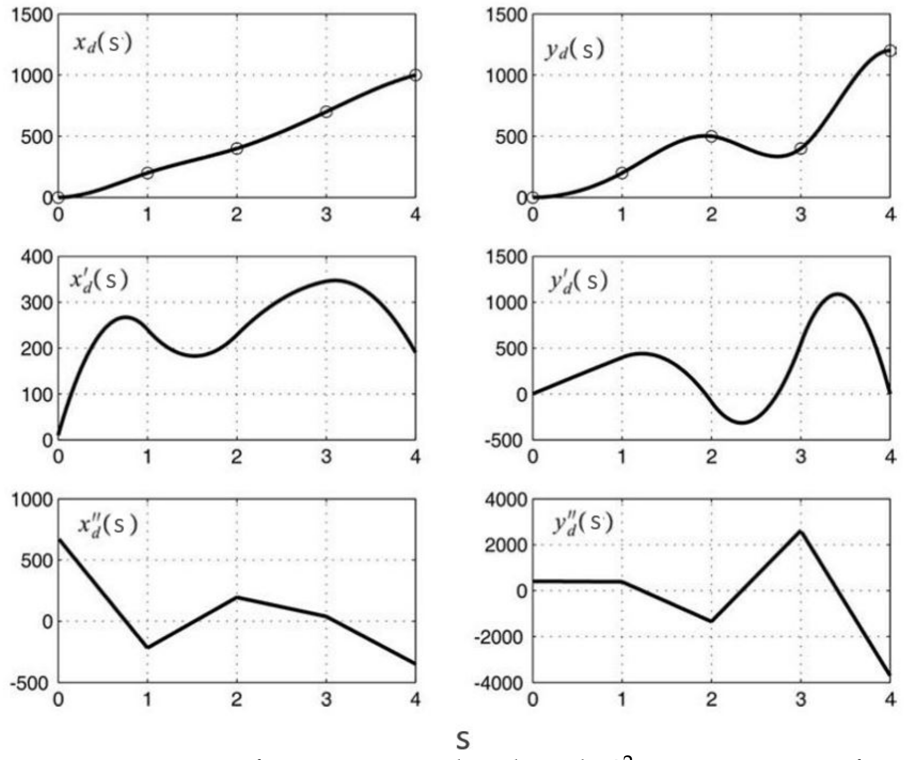


Figure 2.4.1: Representation of a parametrized path with C^2 continuity, over four waypoints. Courtesy of Fossen (2016).

2.4.2 Maneuvering

The practice of enabling vessels to converge to and follow one or more desired parametrized paths is termed the maneuvering problem (Skjetne 2005). In general this problem can be split into two subtasks of geometric and dynamic concern. The first task involves making the vessel converge geometrically to the path parametrized in terms of a continuous scalar variable. The second task is concerned with fulfilling desired dynamic behaviour along this path, defined by assignments of time, speed or acceleration. This paper is mostly concerned with the speed assignment, in order to improve speed selection for a docking situation. To solve this problem Skjetne (2005) proposes a controller based on robust recursive design techniques. As the geometric task is viewed as the most important under the maneuvering problem, this is prioritized first and solved under several recursive steps before advancing to solve the dynamic task. The two tasks are then bound together by an update law.

1. Geometric task:

$$\lim_{t\to\infty} |p^n(t) - p_d^n(s(t))| = 0 \tag{2.20}$$

$$p_d(s) = \begin{bmatrix} p_{d,x}(s) \\ p_{d,y}(s) \end{bmatrix} := \begin{bmatrix} a_1 s + b_1 \\ a_2 s + b_2 \end{bmatrix} \in \mathbb{R}^2$$

2. Dynamic task:

$$\lim_{t\to\infty} |\dot{s}(t) - v_s(s(t), t))| = 0 \tag{2.21}$$

$$v_s(t,s) = \frac{u_d(t)}{|\overline{p}_d^s(s)|}$$

For path variable $0 \leq s \leq 1$ where p^n is the first to n_{th} derivative of the vessel position p. Here $u_d(t)$ deotes the desired speed of the vessel at time t. For the desired heading ψ_d an angle tangent to the generated path is used, i.e $\psi_d(s) = \angle p_d^s(s)$. The complete objective of the maneuvering problem can then be formulated as generating a control force τ such that $\eta \to \eta_d$, where $\eta_d(s) := \begin{bmatrix} p_d(s) \; \psi_d(s) \end{bmatrix}^\top$ and $\eta_d : \mathbb{R} \mapsto \mathbb{R}^2 \times \mathcal{S}_1$. Basing the design of this control force on the methods of Skjetne (2019*b*), we define the following parameters for our backstepping design:

$$z_1 := R(\psi)^\top [\eta - \eta_d(s)], \quad z_2 = \nu - \alpha_1, \quad \omega = \dot{s} - v_s(s,t) \tag{2.22}$$

Where α_1 is a virtual control variable to be defined later on. We start by deriving the update law by differentiating z_1 with respect to time and defining the first control Lyapunov function (CLF), resulting in step 1 as:

Step 1

$$\begin{aligned}
\dot{z}_1 &= \dot{R}^\top [\eta - \eta_d] + R(\psi)^\top [\dot{\eta} - \eta_d^s \dot{s}] = -rSz_1 + z_2 + \alpha_1 - R(\psi)^\top \eta_d^s (\omega + v_s) \\
V_1 &= \frac{1}{2} z_1^\top z_1 \\
\dot{V}_1 &= -r z_1^\top S z_1 + z_1^\top z_2 + z_1^\top \left[\alpha_1 - R(\psi)^\top \eta_d^s (\omega + v_s)\right]
\end{aligned} \tag{2.23}$$

Additionally we define the virtual control variable α_1 and the first tuning function ρ_1

$$\begin{aligned}
&\alpha_1 = -K_1 z_1 + R(\psi)^\top \eta_d^s v_s - \kappa_1 z_1, \quad K_1 = K_1^\top > 0, \quad \kappa_1 > 0 \\
&\rho_1 = V_1^s(\eta, s) = -z_1^\top R(\psi)^\top \eta_d^s
\end{aligned} \tag{2.24}$$

As we want for the maneuvering update law to only act in the output space of η it's useful to define this before dealing with the second step and z_2. To ensure a gain independent of the path parametrization at all η_d, we define an update law holding a constant tangent vector length, namely a *Unit-tangent gradient update law*. The update law is defined as follows

$$\begin{aligned}
\omega &= \mu \frac{\eta_d^s(s)^\top}{|\eta_d^s(s)|} R(\psi) z_1, \quad \mu \geq 0 \\
\Rightarrow \dot{s} &= v_s(t,s) + \mu \frac{\eta_d^s(s)^\top}{|\eta_d^s(s)|} R(\psi) z_1
\end{aligned} \tag{2.25}$$

We can the move on to step 2

Step 2

$$\begin{aligned}
M\dot{z}_2 &= M\dot{\nu} - M\dot{\alpha}_1 = -D\nu + \tau + R(\psi)^\top b - M[\sigma_1 + \alpha_1^s \dot{s}] \\
V_2 &= V_1 + \frac{1}{2} z_2^\top M z_2 \\
\dot{V}_2 &= \dot{V}_1 + z_2 M \dot{z}_2 \\
&\leq -z_1^\top K_1 z_1 + \frac{1}{4\kappa_1} z_2^\top z_2 + z_2^\top \left[-D(z_2 + \alpha_1) + \tau + R(\psi)^\top b - M(\sigma_1 + \alpha_1^s \dot{s})\right]
\end{aligned} \tag{2.26}$$

Where

$$\begin{aligned}
\sigma_1(t,s,\eta,\nu) = r\tilde{K}_1 S z_1 - \tilde{K}_1 \nu - rSR^\top \eta_d^s(s) v_s(t,s) + R(\psi)^\top \eta_d^s(s) v_s^t(t,s) \\
\tilde{K}_1 := K_1 + \kappa_1 I
\end{aligned} \tag{2.27}$$

Here b is the measured bias estimated by an observer. From this we can develop the equation for thrust in order to render the system stable as

$$\tau = -K_2 z_2 + D\alpha_1 - R(\psi)^\top b + M(\sigma_1 + \alpha_1^s \dot{s}) \tag{2.28}$$

This concludes the update law for the closed loop system, with input variables $[p_d(s), u_d(t), \eta, \nu]^\top$ and tuning gains K_1, K_2, κ and μ.

2.5 Problem Statement

This thesis aims to apply the above defined techniques in a combination with machine learning, more precisely reinforcement learning. This serves to gather experience and teach the vessel to evaluate good docking procedures through a trial and error process. Through this experience the agent will serve as an action planning guidance layer with the freedom to improve inputs of path variables in order to reach a target pose by the quay subject to optimization goals and criteria. For simplicity the problem is split into two separate cases of path evaluation and speed evaluation.

Assume a vessel coming in at an initial conditions of pose and velocity close to the quay, sat to track an initial path segment. The first task is to let the agent learn a good entrance path to take the ship into a user defined target pose found at the quay side. For the task $u_d(t)$ of constant value is assumed, and the agent is instructed to generate target waypoints ensuring safe guidance of the vessel to the target pose.

The second task assumes similar initial conditions, but is given a pre-specified path to follow by the operator leading to the target pose. The assumption of a constant $u_d(t)$ is relaxed. The machine learning agent should determine good assignments of desired speeds along the path. Included in this is being able to bring the vessel to a full stop at the paths end i.e at docking pose, and uphold sufficient precision ensuring safe path tracking.

In this thesis we defined the simulation model in Section 2.3. The studies will be carried out using this model, through a series of simulations.

Chapter 3

Artificial intelligence and machine learning

3.1 Machine learning

The path planning methods, for for docking of autonomous vessel, later presented in this thesis is based on Machine learning (ML). ML can be described as the discipline within artificial intelligence that deals with a computers ability to improve over time, and learn without being explicitly programmed. ML can further be categorized into three majorly recognized sub fields of supervised learning, unsupervised learning and reinforcement learning. This thesis is mostly concerned with that of reinforcement learning as the technique used for the path planning experiment, seeing the environment as a Markov decision process.

3.2 Markov decision process

A Markov decision process (MDP) can be conceptualized a non deterministic graph representation of the environment (Bellman 1957*b*). It provides a mathematical framework for decision making by taking into account that, although there may be a distribution of possible consequences to an action, the outcome of that action will be partially random.

A state s is said to have the *Markov* property, if and only if the s gives access to all relevant information from history. In an MDP, given action a, the transition between a state at time t,s_t, containing the *Markov* property and its successor s_{t+1} at time t+1 can be defined through a probability distribution over next possible successor states:

$$P^a_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a] \tag{3.1}$$

An MDP can be represented as a directed graph, where the vertices are made up of the states space S and the set of actions available at each particular state and time. Choosing to execute an action at time t causes a transition of the system to a particular state s′. Along with this, the system receives an immediate reward R based on its transition to state s′ from state s, due to the action a, calculated by the reward function R:

$$R_s^a = \mathbb{E}[R_{t+1} = s'|s_t = s, a_t = a] \tag{3.2}$$

Due to the non-deterministic characteristics of the graph, choosing to execute a specific action a in state s at separate occasions may produce different results (e.g. a different reward is given or a different state transition occurs). This can also happen as a result of the set of actions available at that state being dynamic. An example of a simple MDP represented as a graph structure is shown in figure 3.2.1.

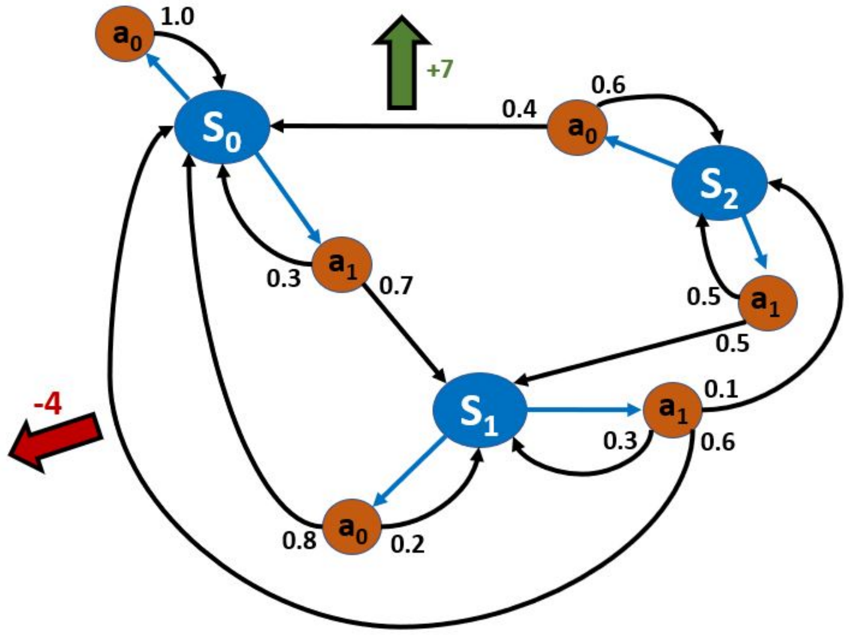


Figure 3.2.1: Example of an MDP with state space S = $\{s_0, s_1, s_2\}$ and action space A = $\{a_0, a_1\}$. Yellow arrows show examples of returned reward for that specific action.

In order to choose actions we define a policy as a distribution over actions given states. This policy fully defines the behavior of an agent,

$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s] \tag{3.3}$$

Once an MDP is combined with a policy in this way, this fixes the action for the states making the resulting combination behave like a *Markov Chain*. As the action chosen at that state will be completely determined by $\pi(s)$ 3.4 will reduce to:

$$P^a_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s] \tag{3.4}$$

The goal is to develop a policy that will maximize some cumulative function of the random rewards returned by the reward function 3.2. Typically:

$$\sum_{t=0}^{\infty} \gamma R^a_s \tag{3.5}$$

Where actions are chosen as $a_t = \pi(s_t)$ and $0 \leq \gamma \leq 1$ is a set discount factor based on the value of immediate and future rewards.

3.3 Bellman equation

The Bellman equation presented in Bellman (1957*a*), is a necessary condition for optimal solutions using mathematical techniques such as dynamic programming (DP). Shortly described, the equation returns a value of a decision problem associated with a specific time instant based on the initial choices leading up to that point, and the following value of the remaining decision problem. Bellman (1957*a*) presents the *Principle of optimality* which mathematical transliteration yields all functional equations associated with the Bellman equation, and reads:

Bellman's Principle of Optimality
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision
(Bellman 1957*a*, Chapter III)

Close to all problems solvable by optimal control theory can be solved by analyzing the appropriate bellman equation. In order to derive these equations it's necessary to understand the general build up of optimization problems. All optimization problems directly depend on the defined objective function i.e what is to be minimized or maximized. To optimize this the DP approach relies on breaking down the complex planning problem into smaller steps of time. Through the use of a *value function* measuring the quality of certain control variables at a specific state, it evaluates a *policy function* holding the ability to return the optimal control variables as a function of this state.

To derive the necessary equations we start by introducing the reward function $R(s_t, a_t)$ as the returned reward for executing an action a_t in state s_t at time instant t for a discrete deterministic process. This lets us define the value function $V(s)$ as the sum of all rewards taken throughout an infinite-horizon process

$$V(s_0) = \sum_{t=0}^{\infty} R(s_t, a_t) \tag{3.6}$$
$$\text{subject to: } a_t \in A(s_t),\ s_{t+1} = f(s_t, a_t), \forall t = 1, 2, 3...$$

Where $A(s_t)$ is the action space at s_t, $f(s_t, a_t)$ is the transition model, and s_0 is used due to the value functions dependency on the initial state.

Now we aim to find the optimal value of this value function, by maximizing the returned rewards over time. Additionally by assuming impatience we introduce the discount factor γ resulting in the complete value function $V(s_0)$ as the maximum of the sum of all discounted rewards calculated throughout the process:

$$V(s_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma R(s_t, a_t)$$
$$\text{subject to: } a_t \in A(a_t),\ a_{t+1} = f(a_t, a_t), \forall t = 1, 2, 3...$$

As mentioned works DP by breaking the larger problem into smaller individual problems causing a simpler calculation. Based on this, and what is inferred by the principle of optimality we can consider the decisions separately, starting with the first decision as following

$$V(s_0) = \max_{a_0} \left\{ R(s_0, a_0) + \gamma \left[\max_{\{a_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) : a_t \in A(s_t),\ s_{t+1} = f(s_t, a_t), \forall t \geq 1 \right] \right\} \tag{3.7}$$
$$\text{subject to: } a_0 \in A(s_0),\ s_0 = f(s_0, a_0)$$

Where the first action is represented to the left of the brackets and the remaining decisions made in the future are collected inside the brackets. Subsequently we can simplify this, by noticing that the collected future rewards inside the brackets is the value of the decision problem at time instant $t = 1$. Rewriting the equation as a recursive definition of the value function we get the Bellman equation as

$$V(s) = \max_{a_0} \{R(s_0, a_0) + \gamma V(s_1)\} \text{subject to: } a_0 \in A(s_0),\ s_1 = f(s_0, a_0)$$

Which can be generalized into an even simpler form for any decision and transition in the process

$$V(s) = \max_{a \in A(s)} \{R(s, a) + \gamma V(f(s, a))\} \tag{3.8}$$

Solving the Bellman equation gives the *value function* and also leaves us with an equation $\pi(s)$, i.e the *policy function* describing the optimal action of any state s. Solving the equation guarantees an optimal solution and is one of the most convenient approaches to optimize control problems. Although the presented equation is derived assuming a deterministic process, it can also be applied to stochastic processes such as MDPs described in section 3.2. The equation then computes the the *policy function* using the transition probability $P^a_{ss'}$ describing how to act optimally under uncertainty. Unlike the deterministic property however, an global optimal solution is not guaranteed due to the uncertainty from the Markov property

3.4 Curse of dimensionality

The curse of dimensionality (Bellman 1957*a*) is a phenomenon that arises when organizing or analyzing data in large spaces with many dimensions, and is highly relevant within ML. Simply put, the problem originates due to the fact that when the number of dimensions increase, the volume of the space increases so fast that available data becomes sparse. RL problems that involve learning in a high dimensional feature space, with each feature having a range of values usually requires an enormous amount computational power and training to ensure proper exploration of the space and/or actions. This problem is likely to quickly arise with vessls in maneuvering and seakeeping problems due to many features in position, velocity and thrust. As discussed in Section 2.3.2, the generalized position, velocity, and force vectors for a 3DOF surface vessel can be defined as:

$$\boldsymbol{\eta} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \qquad \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ r \end{bmatrix} \qquad \boldsymbol{\tau} = \begin{bmatrix} \tau_u \\ \tau_v \\ \tau_r \end{bmatrix} \tag{3.9}$$

implying that the agent potentially has to take a total of 9 states into account. Additionally external conditions as currents and wind may also want to be considered. A typical rule of thumb is that there should be at least five training examples for each dimension in the representation (Theodoridis & Koutroumbas 2008). This implication may result in the necessity of a tremendous amount of training episodes to show improved results. A possible approach to reduce this problem of dimensionality however is by using function approximation, later presented in Section 3.7.2.

3.5 Reinforcement learning

Reinforcement Learning (RL) is one of the most active and rapidly developing areas within ML (Wang 2019). Formally the technique presents a way to solve MDP problems built on the core idea of taking advantage of newly gathered information from a previously unknown environment (Kunz 2013, Huys et al. 2014). The approach deals with teaching an agent the connection in between states and actions, known as state-action pairs $\{s, a\}$, with the aim of maximizing a user defined numerical reward. Subsequently, with environments of strong interconnections of such pairs and future rewards possible RL is rendered a complex problem.

Unlike other approaches to ML such as supervised learning, where a batch of data for training is made available for the agent, RL methods depend on gathering this data in the process. This renders the trade-off between exploration and exploitation. The ultimate aim is to maximize the total reward returned through out the process, implying that the action holding the highest reward in any state should be applied. However exploring new and unknown actions could result in new findings resulting in even more reward. Another important aspect in this trade-off is the handling of delayed rewards. In a given state, an action may return a certain instantaneous reward but will also lay grounds for all succeeding rewards.

Figure 3.5.1 shows a representation of the standard model for reinforcement-learning. On each step, information about the current state s is fed to the agent as input i. The agent then, based on it's current behavior, or so called *policy* π, decides on an action a to return as output. The change that specific action makes to the state of the environment is returned to the agent as a scalar reinforcement signal, i.e reward R. The goal of the process is for the agents behaviour to choose actions that return a long-range sum of values of R. For real world systems this learning process is usually carried out through simulation of the system, to reduce risk of accidents occurring through learning.

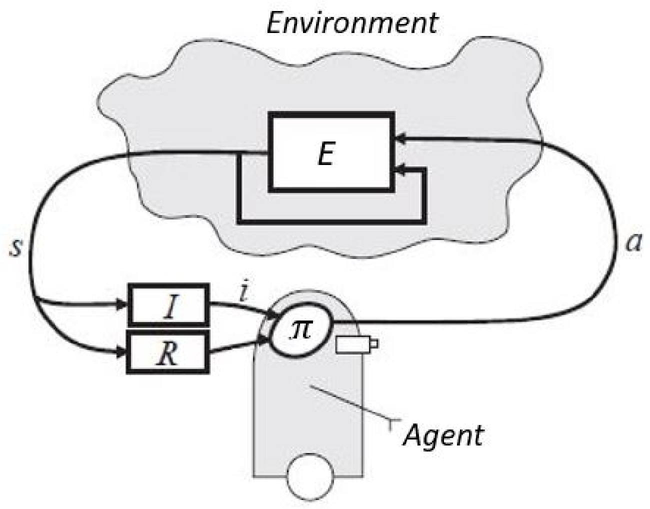


Figure 3.5.1: The standard reinforcement-learning model-based on Kaelbling et al. (1996).

For a given problem the agent must must learn its policy through trial-and-error interactions with a dynamic environment to find more optimal solutions. *How* the task is to be achieved is usually not specified, but computational obstacles create a challenge to fulfill the purpose. In order to enable a system to have these algorithms applied, it's necessary to model the system in terms of states, actions and rewards. This is most commonly done using an MDP, which therefore also is the world most RL techniques assume to be in.

3.5.1 Model-based vs. Model-free

RL techniques can be categorized into classes of either model-free or model-based (Huys et al. 2014). Model-based approaches assume knowledge of the environment in terms of how executed actions affect the current state and how this affects performance. Through this the agent attempts to learn a transition probability $P(s_{t+1}|s_t, a_t)$ based on the current state-action pair $\{s_t, a_t\}$ onto the succeeding state s_{t+1}. The agent can then take use of this learned model to make predictions about upcoming states as well as associated rewards resulting from carrying out an action, ahead of executing it.

Model-free approaches however, rely solely on trial and error to update its knowledge. These do not require any previous information of the underlying model or how performance is measured. Thus it is not necessary for the agent to learn the model as for model-based techniques, and the agent can instead estimate a policy. This is the approach we are most concerned with in this thesis. Such a method can for example be applied with *Temporal difference* though iterative updates, discussed in section 3.6.

3.5.2 Reward function

All traditional RL methods require a way of measuring performance of the agents actions, known as a reward function. This function is used to indicate how well the agent is performing overall and is used as a tool to guide the agent toward desired behaviour (Daniel et al. 2014). The agents sole goal is to maximize the output of this function, and thus this function of normative content has to be carefully defined by the user, and has a lot to say for convergence speed and functionality of the training. The reward function is most often not considered as a part of the RL agent itself, but rather as a part of the environment. This is on the basis of the reward function usually being defined as a measurable quantity in terms of the environment, and for model-free approaches in specific, the underlying mapping to state-action pairs is unknown (Marsland 2015). Agents of RL can often come upon new, unconsidered, strategies to harvest rewards from the environment. This can be a clear strength, but can also lead to undesirable and dangerous solutions, implying the importance of careful reward function design (Sutton, Richard S. and Barto 2010). The optimal formulation highly depends on the type of learning algorithms applied, as well as action space formulation and environmental properties as the determinism of the process. The intimate connections to the action and state space formulations makes the definition a complex problem and is generally viewed as the most challenging part of a RL, highly defining the hardness.

For simple problems of few steps it might be enough to return a single reward when the agent reaches a terminal state s_T i.e the end of the episode. Performance can then be improved through backpropagation of this delayed reward. However, for learning problems of many steps this might result in a very lengthy process and problems of optimal convergence. In such cases a continuously harvested reward can be applied in addition or instead, in order to facilitate steeper learning, but potentially requiring additional exploration. However, an important aspect is that the function stands to inform the learner of what the goal is, and not how it should be achieved (as for supervised learning methods). Marsland (2015) therefore specifies that the implementation of sub goals is often a bad idea, due to the introduction of local optimums where the agent finds ways to optimize these sub goals while missing the real intention. Such problems can also occur due to conflicting goals, possibly shrouding clear optimal solutions (Nowe et al. 2012).

As mentioned can rewards backpropagate and lead to convergence of the process. However there is a lot of uncertainty present in a learning process, especially in terms of future rewards. To account for this outputs of the reward function is discounted by a factor $0 \leq \gamma^t \leq 1$, where t is the number of time steps into the future. This makes the total returned reward returned at time t defined as

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{k-1} r_{t+k} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.10}$$

This parameter can also be seen used in the Bellman equation in 3.8. This facilitates of better reward predictions when choosing actions based on rewards through out the learning process in an attempt to maximize it, or alternatively carry out further exploration.

With so many factors to be considered as well as various goals of RL problems, many approaches have been experimented with. Energy and time minimization problems often includes components of energetic costs, penalizing the agent by a negative reward for each epoch or applying inverse components rewarding reductions (Zhang et al. 2018, Rioual et al. 2018). Difficulties in shaping these functions has also lead to development of newer methods such as *Inverse Reinforcement learning*. This approach does not require an explicitly defined reward function, but instead tries to learn the reward function on its own through mimicking human behavior, or in specific a master. This approach however assumes the human operator performs perfectly, which in most cases is an unrealistic assumption (Nguyen et al. 2015).

3.5.3 Exploitation vs exploration

model-free RL involve a trade off between exploration and exploitation. Treating the environment as a black box, agents apply a trial and error strategy to gradually approach more optimal solutions. This tactic involves utilizing the agents actions both to learn (explore) and to optimize (exploit) different strategies (Wang 2019). Specifically this trade off involves a balance between exploring the unfamiliar environment in search for new information to achieve long-term benefits, and applying previously acquired knowledge to yield higher rewards. Since selecting possibly sub optimal unknown actions can be a costly process, an important aspect of RL is to address the contrast between these two.

Exploration of the state space can be handled several ways by different action selection methods Marsland (2015). Most obvious is selecting the highest rated actions through a greedy policy. But this approach will potentially lead to very little exploration, and result in overfitting. Thus to handle the trade-off preventing the two opposing problems of consistent action selection, and completely random movements, a parameter $0 \leq \epsilon \leq 1$ is used (Watkins & Dayan 1992). ϵ denotes the agents tendency to favor exploitation of previous knowledge over exploration to attain more. The lower the ϵ value, the more likely the agent is to follow a "safer path" based on its past experiences. Likewise, high ϵ will on average return more penalties due to curious behaviour of the agent, and random choices. This is known as $\epsilon - \textit{greedy}$ action selection. A widely used approach for the exploration-exploitation trade-off, is to start off with higher values, and decreasing it more and more as training proceeds. This enables the agent to do loads of early exploration, followed by exploitation of the most promising sections. Further refinement of the ϵ greedy approach can be to give the agent an idea of what actions to explore first eliminating uniform selection probability and completely random movements. Another more intelligent action selection strategy frequently seen in RL is *Soft-max action selection*, but is not focused on in this thesis.

3.6 Temporal difference learning

With the Bellman equation derived in the previous section we have seen how DP can be applied to solve optimization problems, breaking it into smaller steps of discrete transitions. This approach however is model-based, i.e requires the environment to be modelled perfectly, which in most situations is not the case. Two other classes to solve RL problems are the *Monte Carlo methods* and the *Temporal Difference methods*, which are both model-free (Kunz 2013). The calculated return for Monte

Carlo (MC) with a can be defined as:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} R_{i+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T \tag{3.11}$$

Where T is the terminal time process. MC calculates the total reward of an entire run from the beginning state to the end state, before updating anything. Temporal difference (TD) however combines the two techniques applied by DP and MC. While being model-free, updates are made after each state transition by recording data along the way and learning from raw experience. These methods are also referred to as "bootstrapping" methods, due to the property of not learning by the difference of the final outcomes of the process, but by evaluating the differences between each update step. Thus instead of a single update like in MC, a total number of $T-1$ updates are made each episode of T steps. Through this TD aims to find an approximation of the value function V, denoted V_θ with an error defined as

$$\text{MSE}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (V_\theta(s_i) - V(s_i))^2 \tag{3.12}$$

Which is optimized by minimizing the equation. Now by applying the Bellman equation from Equation 3.8, for a using expected returns due to the stochastic process we can estimate the unknown value function $V(s)$

$$V(s_t) \approx \mathbb{E}\{R_t + \gamma V_\theta(s_{t+1})\} \tag{3.13}$$

Or similarly when when transitioning between state-action pairs

$$V(s) = \mathbb{E}(R(s_{t+1}, s, a) + \gamma V_\theta(s_{t+1})) \tag{3.14}$$

A key difficulty however in RL problems is the *curse of dimensionality*, coined by Richard Bellman himself in Bellman (1957*a*) to explain how state spaces quickly grow large and cause computational difficulties. This is further described in Section 3.4. Due to this calculating the minimum error by can become impossible. To overcome this a local minimum can instead be used through following the error functions gradient as:

$$\theta' = -\alpha \nabla \text{MSE}(\theta) \tag{3.15}$$

Where the learning rate α is applied to adjust the step size and prevent overshooting. By letting e_t denote the approximation of the gradient $\nabla_{\theta_t} V_{\theta_t}$, and δ_t denote the TD, the TD update function can then be formulated as:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha \delta_t e_t \\ \delta_t &= r_{t+1} + \gamma V_{\theta_t}(s_{t+1}) - V_{\theta_t}(s_t) \end{aligned} \tag{3.16}$$

For faster learning eligibility traces is introduced, using $0 \leq \lambda \leq 1$ to describe how far back over the trajectory updates should be made. High values lead to longer lasting traces, reaching more distant states with reward updates from $\lambda = 0$ implying backwards propagation of one step while $\lambda = 1$ traverses back the entire trajectory from start to end resulting in MC behaviour. This is achieved by redefining e_t as:

$$e_t = \sum_{k=t_0}^{t} \lambda^{t-k} V_{\theta_{\theta_t}}(s_k) \tag{3.17}$$

This gives rise to the general $TD(\lambda)$ function represented in Algorithm 3. Sampling back over a set number of steps is also referred to as *n-step backpropagation*, as depicted in Figure 3.6.1. Neither of the two extremes of one step TD or MC is always best, and in practice the optimal update law is found by an intermediate value of the two (Sutton, Richard S. and Barto 2010).

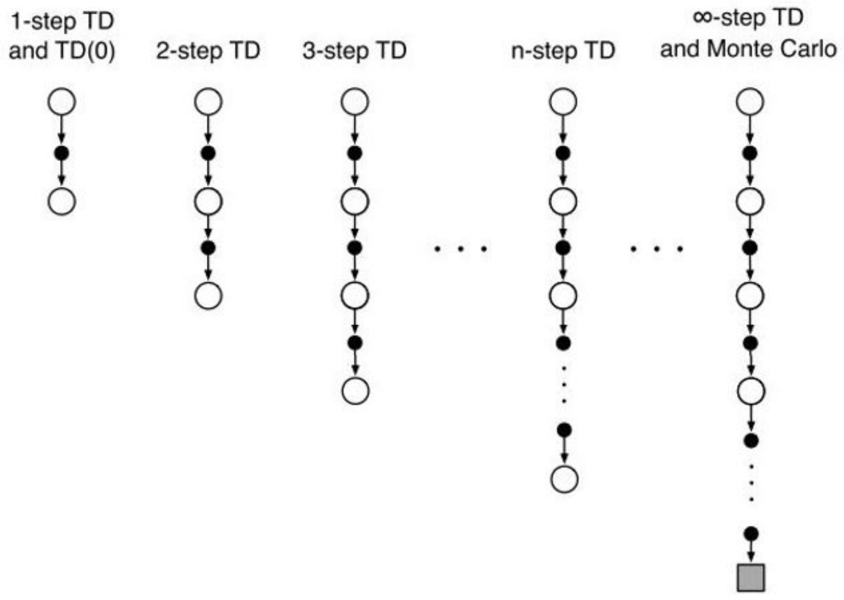


Figure 3.6.1: n-step backpropagation methods as depicted in Sutton, Richard S. and Barto (2010).

These returns can be viewed as approximated to the full return following an action, truncated after n-steps. In the event of $t + n \geq T$ i.e the n-step backpropagation extending beyond the systems goal state, all missing terms are accounted for by 0 values. Similarly this is the case during the agents first exploratory episodes. The TD method is an approach to solving RL problems based on policy evaluation and improvement. It approximates a state-value function, but can with modifications be used to approximate action-value functions giving rise to learning methods as Q-learning and SARSA. Furthermore the n-step approach can be combined with SARSA for an on-policy TD control method.

Algorithm 1 TD(λ) based on an example from Boyan (2002)

```
1:  begin
2:      set $\theta$ to an arbitrary initial estimate, t := 0
3:      for $n := 1, 2, 3, ...$ do
4:          set $\delta := 0$
5:          select a starting state $s_t$
6:          while $s_t \neq T$ do
7:              Propagate one step, producing $R_t$ and $s_{t+1}$
8:              set $\delta = \delta + e_t(r_t + (e_{t+1} - e_t)^T\theta)$
9:              set $e_{t+1} = \lambda e_t + e_t$
10:             t = t + 1
11:         end while
12:         set $\theta = \theta + \alpha\delta$
13:     end for
14: end
```

3.7 Q-learning

Q-learning (Watkins & Dayan 1992) is a model-free algorithm commonly used in RL problems. The algorithm is a bootstrapping method teaching an agent to act optimally in Markovian domains by recording the consequences of actions in accordance to certain instants of the process, similar to that of TD learning. By letting the agent select an action at a particular state the immediate consequences is evaluated through basic reward/penalty feedback based on a reward function and memorized. Using this reward, combined with an initial or previously recorded value-estimate of the state to which it is taken, the agent instantly updates the value of the executed action in the previous state as an on-line learning process. More delayed learning is also possible as an n-step process, but could cause issues depending on the amount of possible actions between s_t and s_{t+n} (Marsland 2015). Including the value of the upcoming state creates a connection of the adjacent states and allows the system to consider global consequences instead of only evaluating a local optima. This step-by-step manner in which Q-learning evaluates its optimal policy, Watkins & Dayan (1992) classes the algorithm a form of incremental DP.

Let the term *Q(s,a)* denote the discounted future reward when performing an action a in state s. This term is also known as Q-value of state-action pair $\{s, a\}$ (Matiisen 2015). Choosing actions that maximizes discounted future reward we have:

$$Q(s_t, a_t) = max_{\pi} R_{t+1} \tag{3.18}$$

Where π represents the *policy*, the definition of how an action is selected for the particular state:

$$\pi(s) = \text{argmax}_a Q(s,a) \tag{3.19}$$

This originates from the theory that if a^* is an action for which the greatest value is attained, then the optimal policy $\pi^*(s)$ can be evaluated from this. Thus if the agent can learn the Q-values, these can be utilized for optimal decision making. Considering a single transition $< s_t, a_t, r_{t+1}, s_{t+1} >$, with r_{t+1} being the immediate reward when reaching the next state s_{t+1}, the Q-value of the transition is expressed as:

$$Q(s_t, a_t) = R_t + \gamma \text{max}_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{3.20}$$

With the expected value multiplied by a discount factor $0 \leq \gamma \leq 1$ due to the uncertainty of future rewards. The Q-value can be depicted as the expected discounted reward from executing action a_t in state s_t proceeded by following the policy π. Here the update is done through an greedy selection of Q-values in the next state, as the highest potential outcome of that action i.e the outcome of $\pi^*(s)$. Using this optimistic selection allows for the agent to learn optimal action decisions independent of π, enabling the algorithm to learn the optimal policy without having to perform the optimal policy, i.e. an off policy algorithm. Multiplying this with a chosen rate of learning $0 \leq \alpha \leq 1$, the complete update rule for for Q-Learning including both immediate reward and expected future reward becomes:

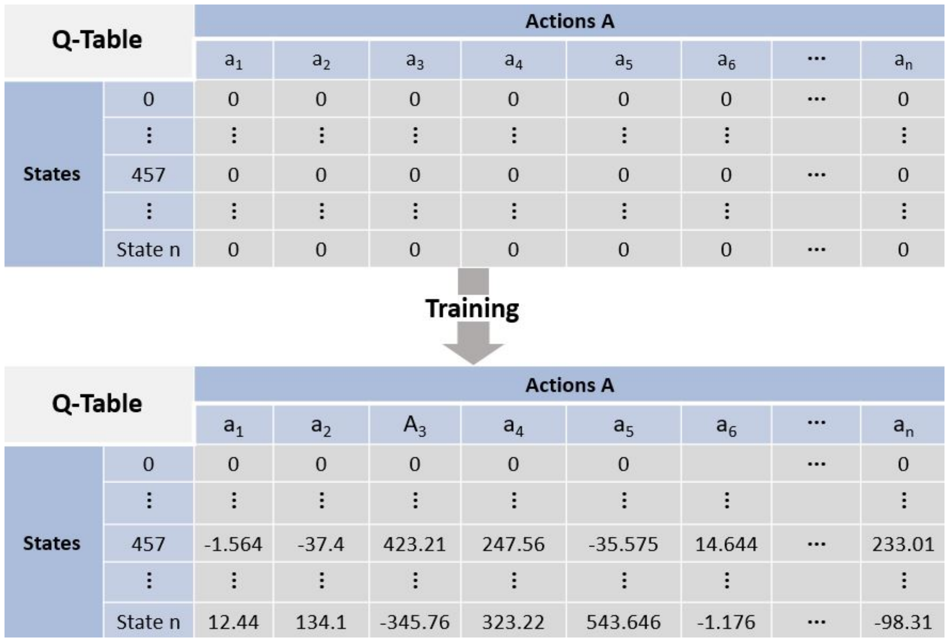
$$\underbrace{Q(s_t, a_t)}_{\text{new value}} \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{R_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\text{max}_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{estimate of optimal future value}} \Big)}^{\text{learned value}} \tag{3.21}$$

Here the term $(1-\alpha)$ is multiplied with the old value to make sure incrementations are made by the difference in the estimate. The general Q-learning has been shown to converge to the optimal action-value function by probability 1 given two conditions; all action values are represented discretely, and these actions are repeatedly sampled in all states (Watkins & Dayan 1992). Further, in order to resemble memory through a finite size Q-table, the algorithm assumes the environment to be represented as discrete state space.

3.7.1 Q-table

To simulate the agents memory of past experiences a Q-table can be used. The Q-table is to contain the calculated value of the state-action pairs present, based on the update rule defined in Equation (3.21). Before training starts, the Q-table is initialized to contain a space for each state-action pair available for the agent. This means, the Q-table will be of dimension n states **x** m actions. As the agent learns through repetitive episodes, the quality of a state-action pair is noted by logging the calculated value in the respective position. As the training process goes on, the agent builds more and more data to base choices of actions on, spanning over the total state space.

Before initializing the training the Q-table is usually initialized to some arbitrarily value. The choice of this value depends on the range the logged values are going to span over, and is thus connected to the range of the reward function as well as γ and α. For most problems this is sat to zero, however the initial values can also be applied as a means to encourage exploration through applying optimistic values (Sutton, Richard S. and Barto 2010). This would lead to the agent to be "disappointed" when making early selections, and select unexplored actions over the previously learned lower ones. Figure 3.7.1 represents an example of a Q-table with initial values of 0 before and after training.



Q-Table		Actions A							
		a_1	a_2	a_3	a_4	a_5	a_6	···	a_n
States	0	0	0	0	0	0	0	···	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮
	457	0	0	0	0	0	0	···	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮
	State n	0	0	0	0	0	0	···	0

Q-Table		Actions A							
		a_1	a_2	A_3	a_4	a_5	a_6	···	a_n
States	0	0	0	0	0	0		···	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮
	457	-1.564	-37.4	423.21	247.56	-35.575	14.644	···	233.01
	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮
	State n	12.44	134.1	-345.76	323.22	543.646	-1.176	···	-98.31

Figure 3.7.1: Example of Q-table before and after training.

3.7.2 Function approximation

In the previous section memory storage has been defined as a lookup table. However with growing dimensionality this can be inefficient due to necessary exploration and memory. An alternative way to store information of state-action pairs is thus to generalize across states using function approximation (van Otterlo & Wiering 2013). In general function approximation deals with evaluating a function that closely matches a target function, which in our case refers to the value function $V(s)$.

Numerous methods exists for this, but the most basic approach is through linear approximation:

$$V(s) \approx V_\theta(s) = \theta^T \phi(s) \tag{3.22}$$

Here V_θ denotes the approximated value function as seen in Section 3.6. $\phi(s)$ is a feature vector holding a number of linear basis functions K, and $\theta \in \mathbb{R}^K$ is an adaptable parameter vector for alignment. This enables reduction in dimensionality from dim(S) to dim(K) (Kunz 2013). This enables efficient storage about huge state spaces, and even continuous ones. Yet this reduction comes at a cost as $\phi(s)$ is only able to capture a finite number of functions, thus inducing limited precision and error in state representation. However this approach is not applied in this thesis and therefore just briefly introduced as a possibility.

3.7.3 Implementation

For simplicity, the use of Q-learning can be broken down to seven consecutive steps as explained by Kansal & Martin (2015):

1. Define action space: set up a system for the agent to choose actions from.
2. Initialize Q-table: define storage space for agent to memorize the value of each action (a) given the space (S) it came from. i.e the Q-table should be of dimension *number of states* X *possible actions*.
3. Start exploring actions: For each state (s_t), select an action (a_t) best noted action or a random one.
4. Travel to the next state (s_t) as a result of that action (a_t).
5. Update Q-table values using the reward equation, together with previously saved value of action (a_t) and estimated future reward based next state (s_{t+1}).
6. Set the next state (s_{t+1}) as the current state (s_{t+1}) and repeat from 3).
7. If terminal state is reached, end process, reset environment and run again.

Algorithm 2 Q-Learning

```
1:  begin
2:      Initialize Q-matrix $Q : SxA \rightarrow \mathbb{R}$ to 0(or any arbitrary value)
3:      for $n := 1, 2, 3, ...$ do
4:          select a starting state $s_t$
5:          while $s_t \neq T$ do
6:              Evaluate policy $\pi$ based on $Q(s_t, a_t)$ (e.g $\epsilon$ greedy: $\pi \leftarrow \arg\max_a Q(s_t, a_t)$ )
7:              set $a_t = \pi(s_t)$
8:              Propagate one step, producing $r_{t+1}$ and $s_{t+1}$
9:              set $\delta = R_t + \gamma \cdot \max_a Q(s_{t+1}, a_{t+1})$
10:             set $Q(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot \delta$
11:             t = t + 1
12:         end while
13:         return Q
14:     end for
15: end
```

3.8 Modified Connectionist Q-Learning

Modified Connectionist Q-Learning (MCQ-L) (Rummery & Niranjan 1994) is another RL algorithm used in MDPs. The algorithm more commonly known as State-Action-Reward-State-Action (SARSA), is as the name implies a modification of the previously explained Q-learning algorithm in section 3.7. Just like Q-learning, MCQ-L makes updates through information of s, a, s_{t+1}. But rather than using a greedy selection of Q-values as s_{t+1} ($\gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$) the algorithm updates its Q-value using an error ($\gamma \cdot Q(s_{t+1}, s_{t+1}) - Q(s_t, a_t)$) making the update rule defined as:

$$\underbrace{Q(s_t, a_t)}_{\text{new value}} \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{(\underbrace{R_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))}_{\textit{error}}}^{\text{learned value}} \tag{3.23}$$

This approach is considered to be more robust than the standard Q-learning updates. An agent utilizing MCQ-L interacts with the environment and updates its policy, making it a *On-policy*. In other words it estimates the return for state-action pairs assuming the policy continuous to be followed rather than choosing actions based on a greedy policy.

Algorithm 3 MCQ-L(λ)

```
1: begin
2:     Initialize Q-matrix $Q : SxA \to \mathbb{R}$ to 0(or any arbitrary value)
3:     Initialize eligibility trace $e : SxA \to \mathbb{R}$ to 0
4:     for $n := 1, 2, 3, ...$ do
5:         select a starting state $s_t$
6:         while $s_t \neq s_T$ do
7:             Propagate one step, producing $r_{t+1}$ and $s_{t+1}$
8:             Evaluate policy $\pi$ based on $Q(s_t, a_t)$ (e.g $\epsilon$ greedy)
9:             set $a_{t+1} = \pi(s_{t+1})$
10:            set $\delta = R_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$
11:            set $e(s_t, a_t) = e(s_t, a_t) + 1$
12:            for $(\widetilde{s_t}, \widetilde{a_t}) \in SxA$ do
13:                set $Q(\widetilde{s_t}, \widetilde{a_t}) = Q(\widetilde{s_t}, \widetilde{a_t}) + \alpha \cdot \delta \cdot e(\widetilde{s_t}, \widetilde{a_t})$
14:                set $e(\widetilde{s_t}, \widetilde{a_t}) = \gamma \cdot e(\widetilde{s_t}, \widetilde{a_t})$
15:            end for
16:            t = t + 1
17:        end while
18:        return Q
19:    end for
20: end
```

3.9 Expected SARSA

In many situations, state transitions are fully deterministic making any possible randomness originate from the chosen policy, such as epsilon greedy choices (Sutton, Richard S. and Barto 2010). In such cases SARSA can only perform well over longer periods of time at sufficiently small learning rates, making short term performances poor. Here algorithms as Expected SARSA holds consistent empirical advantage permitting any learning rate $0 \leq \alpha \leq 1$ without suffering degradation of asymptotic performance. The algorithm originates from altering the structure of Q-learning to instead of making greedy choices over the next state-action pairs $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ it uses the expected value of each action under the given policy $\sum_a \pi\left(a_t | s_{t+1}\right) Q\left(s_{t+1}, a_t\right) - Q\left(s_t, a_t\right),$. That is making the update rule

$$\begin{aligned} Q\left(s_t, s_t\right) &\leftarrow Q\left(s_t, a_t\right) + \alpha\left[R_t + \gamma \mathbb{E}\left[Q\left(s_{t+1}, a_{t+1}\right) | s_{t+1}\right] - Q\left(s_t, a_t\right)\right] \\ &\leftarrow Q\left(s_t, a_t\right) + \alpha\left[R_t + \gamma \sum_a \pi\left(a | s_{t+1}\right) Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right] \end{aligned} \tag{3.24}$$

The application of this update rule makes Expected SARSA move deterministically in the same direction as SARSA moves in expectation. The computational requirements are more complex than that of SARSA, but eliminates the variance from random selections of upcoming actions a_{t+1}. This algorithm is usually found to be on-policy, but can also in some cases be off-policy depending on π. For example if π is sat to make greedy choices, then the algorithm will be identical to Q-learning and an off-policy method (Sutton, Richard S. and Barto 2010). In such a manner Expected SARSA becomes a generalized form and follows the traits of Q-learning while reliably improving over SARSA.

Chapter 4

Autodocking path planning by machine learning

4.1 Vessel and environment

As discussed in Section 2.5, simulations are performed with a model built upon the characteristics of CSAD. This vessel holds a high block coefficient and is thus not necessarily optimal for maneuvering problems due to subsequent limitations dynamic stability (Kobylinski 2003). However the abilities will suffice for the purpose of testing some RL applications under simple maneuvering tasks. As CSAD usually is ran in the Marine cybernetics laboratory (MC-lab) operated by the Department of Marine Technology at NTNU, this is chosen as the environment to yield realistic tests and facilitate for extensions to real world testing. The laboratory is a small basin with dimensions L x B x D = 40m x 6.45m x 1.5 and is equipped with a coordinate system for precise real-time measurements of vessel position. This measuring system is mounted on a towing carriage with adjustable speed along the tank length. With the idea that a moving coordinate frame can be used to simulate the influence of irrotational currents, the boundaries of the environment is restricted to a portion of the tanks length equal to the width. These boundaries are the limited further due to measurement scope of the coordinate system and to facilitate for tests with currents working at various angles. The idea is depicted in Figure 4.1.1. Consequently the final L X B surface area available for the docking simulations is sat to 6m X 6m. All physical influences such as current are reduced using Froude scaling laws for more realistic values to the models scale. This implies multiplying all current velocities with a factor of $\frac{1}{\sqrt{\lambda}}$ where $\lambda = \frac{L_F}{L_M}$ is the scaling ratio between the CSAD model and the full scale model.

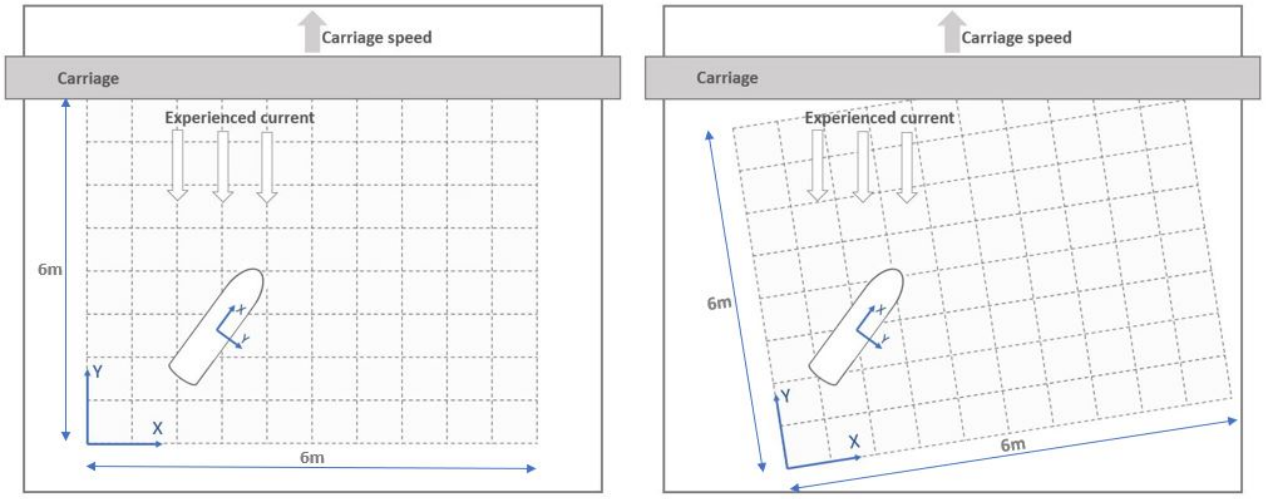


Figure 4.1.1: Idea of environment setup based on MC-lab. First image shows a coordinate system parallel to the tank while the second shows angled coordinate system for angled currents.

4.2 Simulation setup

In order to gauge the performance of RL in a specific docking situation, the vessel model is combined with appropriate guidance and control systems in accordance with Skjetne (2005). The closed loop system is constructed around the CSAD vessel model and a nonlinear passive observer to for noise filtering and bias estimation, to accurately represent responses and state. These variables are further fed to the guidance system applying an RL to generate path variables and further create detailed signals regarding the trajectory with necessary derivatives for the control law. The closed loop system is then completed by the control system generating actions and responses of the vessels actuators. The signal flow is represented in Figure 4.2.1.

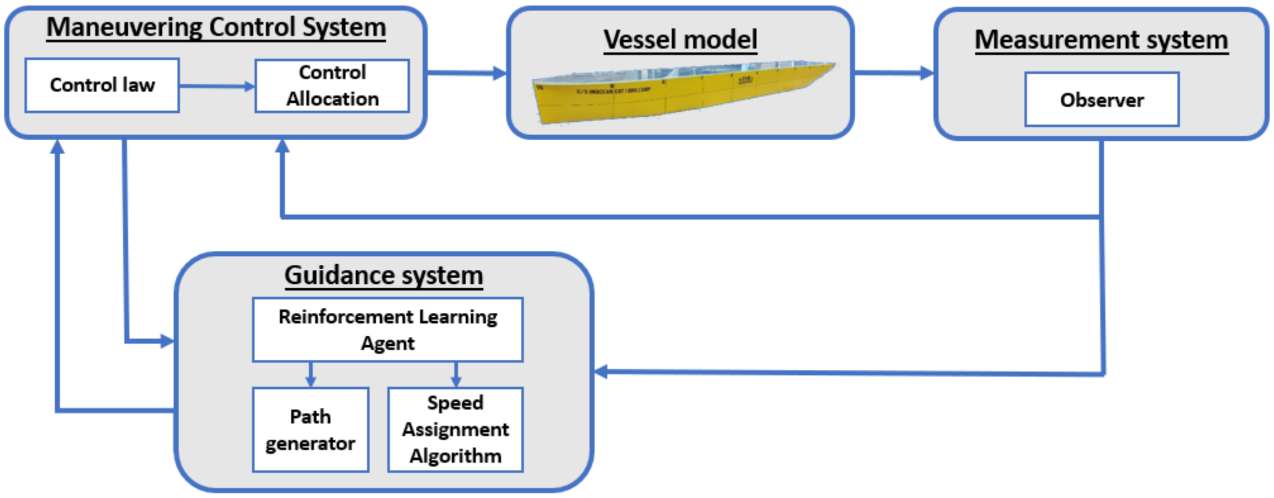


Figure 4.2.1: Signal flow in the closed-loop system. Adapted from Skjetne (2005)

4.2.1 Stepwise path generation

To generate a feasible paths for the vessel to follow we employ the techniques and algorithms discussed in Section 2.4.1. However, as we aim to extend the path generation procedure to be controlled by RL on-line the assumption of a previously known list of target waypoints and target speeds becomes infeasible. Continuously selecting and evaluating actions in different states we will only able to determine characteristics of upcoming path segments one step ahead i.e only deciding on a single waypoint ahead of time. This motivates a strategy stepwise path generation. From here on we let p_0 denote the departure waypoint previously visited, and p_t denote the current targeted waypoint.

Starting with the geometric task knowing only a general previous waypoint $p_{0,i}$ and the current destination waypoint $p_{t,i}$, we ensure continuity at the connection points by requiring $p_{0,i+1} = p_{t,i}$ and tangent vectors of adjacent path segments lining up through the following laws:

$$p^{\theta}_{d,i}(1) = \lambda \frac{p_{t,i} - p_{0,i}}{|p_{t,i} - p_{0,i}|}, \quad p^{\theta}_{d,i+1}(0) = \lambda \frac{p_{t,i} - p_{0,i}}{|p_{t,i} - p_{0,i}|} \tag{4.1}$$

Where $p^{\theta}_{d,i}$ is the first derivative of current path segment, $p^{\theta}_{d,i+1}$ is the first derivative of the upcoming path segment and $\lambda > 0$ is the tuning constant discussed in 2.4.1. We present a parametrized path segment through the following equations, letting $\theta \in [0, 1)$:

$$\overline{p}_d(\theta) = \begin{bmatrix} x_{d,i}(\theta) \\ y_{d,i}(\theta) \end{bmatrix}, \quad \overline{p}^{\theta^1}_d(\theta) = \begin{bmatrix} x^{\theta^1}_{d,i}(\theta) \\ y^{\theta^1}_{d,i}(\theta) \end{bmatrix}, \quad \overline{p}^{\theta^2}_d(\theta) = \begin{bmatrix} x^{\theta^2}_{d,i}(\theta) \\ y^{\theta^2}_{d,i}(\theta) \end{bmatrix} \tag{4.2}$$

In order to fulfill the criteria for the connection points between the path segments as presented in Equation 4.1 we employ the following equations for generating the coefficients of a path segment by continuously evaluating the coefficient vector $A_k = [a_k, b_k]^\top$ of polynomials of order k according to:

$$p_d(\theta) = A_k\theta^k + \ldots + A_3\theta^3 + A_2\theta^2 + A_1\theta + A_0 \tag{4.3}$$

$$p^{\theta}_d(\theta) = kA_k\theta^{k-1} + \ldots + 3A_3\theta^2 + 2A_2\theta + A_1 \tag{4.4}$$

$$p^{\theta^2}_d(\theta) = k(k-1)A_k\theta^{k-2} + \ldots + 6A_3\theta + 2A_2 \tag{4.5}$$

$$\begin{aligned} &p^{\theta^3}_d(\theta) = k(k-1)(k-2)A_k\theta^{k-3} + \ldots + 6A_3 \\ &\text{etc...} \end{aligned} \tag{4.6}$$

These are solved he based on our PC criteria as follows: C^0 continuity by ensuring connectivity to the next path segment as:

$$\begin{aligned} &p_{d,i}(0) = p_{0,i} && p_{d,i}(1) = p_{t,i} \\ &p_{d,i}(0) = A_{0,i} = p_{0,i} && p_{d,i}(1) = A_{k,i} + \ldots + A_{1,i} + A_{0,i} = p_{t,i} \end{aligned} \tag{4.7}$$

C^1 continuity by ensuring equal slopes at the connection points according to Equation 4.1 as:

$$\begin{array}{ll} p^{\theta}_{d,i}(0) = T_{0,i} & p^{\theta}_{d,i}(1) = \lambda \frac{p_{t,i}-p_{0,i}}{|p_{t,i}-p_{0,i}|} \\ p^{\theta}_{d,i}(0) = A_{1,i} = T_{0,i} & p^{\theta}_{d,n}(1) = kA_{k,i} + \ldots + 2A_{2,i} + A_{1,i} = \lambda \frac{p_{t,i}-p_{0,i}}{|p_{t,i}-p_{0,i}|} \end{array} \tag{4.8}$$

C^2 by further setting the following path derivatives equal to 0 at the connection points i.e $p^{\theta^j}_{d,i}(0) = 0$ and $p^{\theta^j}_{d,i}(1) = 0$, for $j \geq 2$. For a 3 DOF maneuvering design model with states defined by $\eta = [x, y, \psi]^\top$ a reference heading ψ_d is also required. Under an assumption of greatest ν_r along the tangent of the vessels velocity, this is generated as a heading tangent to the path as follows:

$$\overline{\psi}_d(i,\theta) = \text{atan}\left(\frac{\overline{y}^{\theta}_d(i,\theta)}{\overline{x}^{\theta}_d(i,\theta)}\right) \tag{4.9}$$

for a general path segment i. Further $\overline{\psi}^{\theta^j}_d(i,\theta)$ is found by evaluating derivatives in regards of θ . For this purpose of this thesis, derivatives up to $j = 2$ is found to be sufficient giving a PC of C^2. For the dynamic task we output a speed profile $v_{s,i}(t,\theta)$ for a general path segment according to an input target speed $u_d(t)$ and its derivatives:

$$v_{s,i}(t,\theta) = \frac{u_d(t)}{\left|\overline{p}^{\theta}_d(i,\theta)\right|} \tag{4.10}$$

Similarly to ψ_d information about the necessary derivatives is found by taking the derivative of the speed profile in regards of θ and t.

4.2.2 System functions

For the guidance scheme a continuous output of reference signals is made through evaluating the above equations according to θ. Using information of the vessels current state η we now let s denote a global path parameter for the total path composed of n path segments i.e $s \in [0, n)$, while θ is the local path parameter for a single segment $\theta \in [0, 1)$ evaluated as $\theta = s - \lfloor s \rfloor$. The dynamics is given according to the unit-tangent update law as discussed in Section 2.4.2 and shown in Equation 2.25, that is:

$$\begin{aligned} \omega &= \mu \frac{\eta^s_d(s)^\top}{|\eta^s_d(s)|}(\eta - \eta_d), \quad \mu \geq 0 \\ \Rightarrow \dot{s} &= \frac{u_d(t)}{|\overline{p}^s_d(s)|} + \mu \frac{\eta^s_d(s)^\top}{|\eta^s_d(s)|}(\eta - \eta_d) \end{aligned} \tag{4.11}$$

μ is sat to 0.025 through a tuning process.

With processes of unknown extent and required number of path segments, we choose to carry out generation and parametrization of the immediate path segment only. The RL agent, later defined, evaluates system feedback through rewards and returns upcoming path variables to be stored for future path generation, as the vessel enters a set radius R of the current p_t. To guarantee smooth transitions between discretely selected waypoints and continuous path parametrization, an update logic between the succeeding target waypoints is implemented as $\lfloor s \rfloor \neq \lfloor s \rfloor_{prev} \rightarrow \begin{matrix} p_0 = p_t \\ p_t = p_{t,RL} \end{matrix}$. Where $p_{t,RL}$ denotes the upcoming target waypoint returned by the agent. This is depicted in Figure 4.2.2 below leading to continuous path parametrization and evaluation of $p_d^{\theta^j}$.

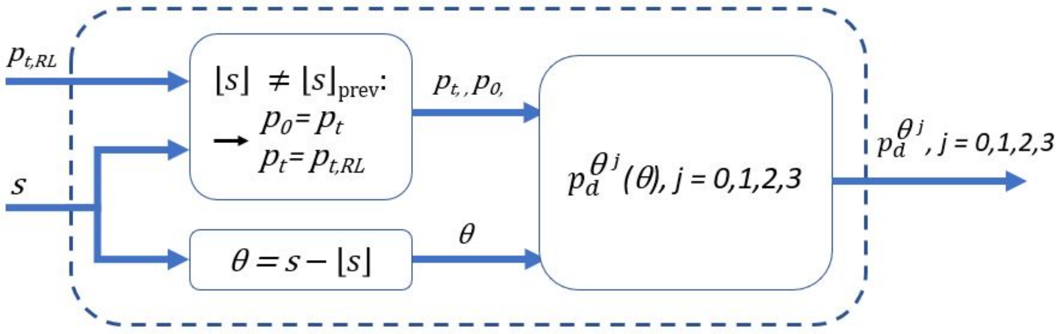

$p_{t,RL}$
$\lfloor s \rfloor \neq \lfloor s \rfloor_{\text{prev}}$:
$\rightarrow$ $p_0 = p_t$
$p_t = p_{t,RL}$
s
$\theta = s - \lfloor s \rfloor$
$p_t, p_0,$
θ
$p_d^{\theta^j}(\theta), j = 0,1,2,3$
$p_d^{\theta^j}, j = 0,1,2,3$

Figure 4.2.2: Update logic for transitioning between waypoints.

Further the control law discussed in Section 2.4.2 and defined in Equation 2.28 is applied to calculate necessary thrust outputs for the vessel to track the path, applying inputs The following gains were found to give well functioning path following for CSAD and associated thrust allocation:

$$K_1 = \begin{bmatrix} 0.36 & 0 & 0 \\ 0 & 0.36 & 0 \\ 0 & 0 & 0.135 \end{bmatrix} \quad , \quad K_2 = K_1 \cdot 10 \tag{4.12}$$

4.3 Planning by Reinforcement Learning

A challenge that can arise in ports and other docking areas is the presence of a large variety currents and other environmental effects difficult to map. This makes local experience tied up to the present magnitudes and locations of forces essential for proposing entrance paths and speed augmentations. At the same time however it's important to know be familiar with the constraints of the vessel to understand what maneuvering missions are possible and optimal. In order to test how direct experience can be applied to generate new paths and possible docking approaches we experiment with letting RL handle the planning through gathering and applying its own experiences. Through a trial and error process RL allows for optimizing measures in the situation such as applied thrust and time efficiency, accounting for more of the system dynamics than traditional planning methods would.

We propose applying RL to the problem by letting the agent control the relevant control outputs of the path planner. For a general docking situation this includes the evaluation of both a good entry path through p_t and associated speed assignments through u_d. Although for a there exists a definite connection between the selected path and the appropriate speeds, a combined evaluation of the two leads to high computational demands due to a curse of dimensionality as described in Section 3.4. We therefore choose to separate the two cases, investigating the evaluation of a good entry path to the area of docking and appropriate speed assignments separately. The proposed state- and action spaces of the two tasks are all made under the consideration of the same trade-off between computational complexity and precision.

4.3.1 State formulation and RL set-up

The main purpose behind this thesis is to gain insight in RL theory and how this can be applied to marine application. To implement such a method a vital part is the build up of the underlying learning algorithms for exploration, learning and further to act as an action-planning guidance layer for the vessel. The RL algorithms are incorporated in the path planner block seen to the very left in figure 4.2.1. For the following formulations we let p denote the position of the vessels CO in the fixed coordinate system of the environment i.e $p = [\eta_1, \eta_2]^\top = [x, y]^\top$.

The proposed algorithms, later described, operate by discrete state inputs and action outputs. In order to capture as much as possible of the vessels situation in the environment in a simple way, the state space is formulated to consist of 3 elements; p_0, p_t and u_{ref}. Where u_{ref} is a discrete reference speed further described in Section 4.4. This formulation simplifies the RL process as the agent can be built with an action space directly altering states and reduce stochastic behavior for more intuitive processes. The state space thus stands to represent path segment characteristics returned by the agent. For updates of the agent to take place, a discrete update rule has to be implemented. This is done by triggering the agent to evaluate the rewards and output upcoming path parameters as the vessel closes in on p_t, i.e $p \approx p_t$, defined as the vessels position p being within a circle of acceptance with radius R of p_t as seen in Fossen (2016):

$$[p_{t,1} - p_1]^2 + [p_{t,2} - p_2]^2 \leq R^2 \tag{4.13}$$

When this condition is confirmed, a signal for the discrete update is sent to the RL algorithm. Such a tactic updating the RL close to but ahead of the target waypoint is beneficial as it allows measurements for learning include vessel behavior during transitions between succeeding segments. Additionally this is where the vessel dynamics is most likely to be steady while ensuring characteristics of the upcoming segment are available ahead of time in the on-line update. For simulations we apply $R = 0.15$ m.

There are many different aspects that can be studied for optimization of a docking process. Relevant fields may be cost efficiency, safety, promptitude and precision. The most important of these is without doubt safety, and failures to comply with this will affect the process in its totality. We therefore put this focus above all others in simulations. Further we aim to shape the reward functions to account for the additional optimization goals of efficiency, such as energy and promptitude. Each episode of training is concluded as the vessel reaches the assigned docking position or fails to comply with safety. Due to difficulty of mapping the complex currents and environmental effects present in a docking process, we propose the use of model-free algorithms in order to learn from vessel responses. This approach also facilitates for easy transitioning to more advanced simulations and extending to real world training for more complex models.

4.4 Speed evaluation along given entrance path

The first sub problem deals with evaluating optimal speed assignments along a given path. This involves applying speed assignments over the various path segments that are low enough to ensure path tracking with acceptable precision under vessel constraints, as well as ensuring safe retardation to zero velocity at the quayside. To train for these criteria the RL agent is given a pre-specified path defined by a series of target waypoints leading to a docking position p_{dock}, with the mission of evaluating speed assignments fulfilling specific goals. The path is made to resemble a reasonable entrance path for a lateral docking process.

Under an assumption of the path being constructed with reasonable safety compliance, the defined terminal states of the process are defined as reaching p_{dock} or loosing track of this path. As a means of detecting if the vessel fails the tracking objective, a time limit t_{max} is imposed for reaching p_t from the associated p_0. This is sat to a generous number in order to only be triggered under actual failure of path tracking. Accordingly the terminal conditions are defined as;

$$s_{\mathrm{T}} = \begin{cases} p \approx p_{dock} \\ t_{p_0 \to p_t} > t_{max} \end{cases} \tag{4.14}$$

Where $t_{p_0 \to p_t}$ is the time spent tracking a path segment between two waypoints, and docking is approved if p is within the circle of acceptance of p_{dock}. Each individual episode is ran until either of these conditions is fulfilled before restarting.

4.4.1 Algorithm

Docking situations involve many hazards, hence our big focus on safety. A training process involving high amounts of exploratory moves onto unknown states can potentially violate these safety concerns, and may be an unfeasible process for physical marine vessels. Because of this we propose an off-policy training algorithm to evaluate speeds through simulations. This facilitates agents to be used in potential real world problems to learn the task by following off-policy samples without having to follow the policy itself. It's worth mentioning however that carrying out training int this manner is less efficient than applying individual training due to model and environmental differences.

As we assume speed assignments are outputted with absolute certainty i.e selected values are exact and reaches the maneuvering layer with probability 1, and low correlation to values of the past we propose the use of Algorithm 2: Q-learning, with epsilon greedy action selection for exploration. To handle this exploration we propose focus on exploration to be done early in the training, followed by optimizing more and more over the current highest valued areas towards the end. We handle this trade-off through a decaying epsilon.

As the speed evaluating process is more of a finer optimization process we propose epsilon decay according to a logistic decay function, starting with an initial epsilon value of $\epsilon_0 \approx 0.3$ of the training before decaying to an optimal policy with terminal value of $\epsilon_T \approx 0$, i.e greedy selections.

$$\epsilon(ep_i, ep_{tot}) = 0.3 - \frac{0.3}{1 + e^{\left(-\frac{8}{ep_{tot}}(ep_i - ep_{tot} \times 0.4)\right)}} \tag{4.15}$$

Here ep_{tot} and ep_i denotes the total and ongoing episode respectively. The behaviour is depicted in Figure 4.4.1. The discount factor is sat to $\gamma = 0.8$, making the agent strive to achieve greater rewards over the long term. Learning rate is sat to $\alpha = 0.45$ as a trade-off between quick learning and finer optimization of logged values.

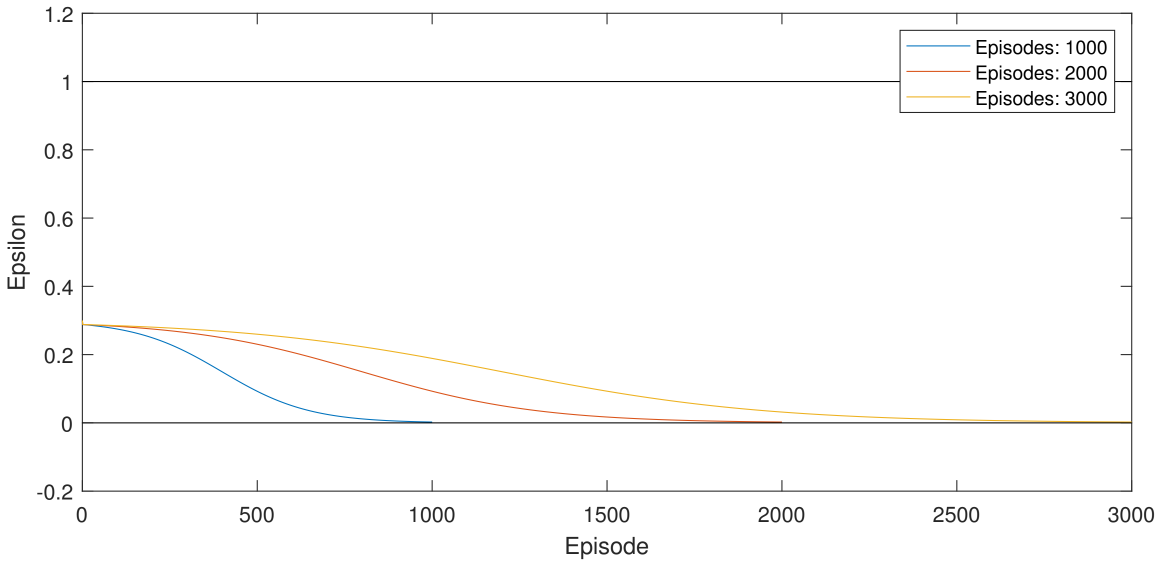


Figure 4.4.1: Epsilon decay for speed evaluation, defined by Equation 4.15.

4.4.2 Action space

due to Q-learning's assumption of a discrete action space we propose the use of a reference speed u_{ref} to make changes to the desired speed $u_d(t)$ along the path. For each individual path segment in the predefined path, the agent is instructed to output a value $u_{ref,i}$ by increasing or decreasing the parameter from the previous path segment $u_{ref,i+1}$. This value is further passed through a reference filter before being fed to the maneuvering law ensuring smooth transitions as $u_d = \frac{u_{ref}}{3 \cdot s+1}$.We propose an action space consisting of a set of augmentation variables $i \cdot \delta_u$, with $i = \pm(1, 2, ..., n)$ being an integer and δ_u being a constant step size generating a space consisting of $2n+1$ possibilities.

$$A_{speed} = \begin{cases} a_0 : & 0 \\ a_1 : & +1 \cdot \delta_u \\ a_2 : & -1 \cdot \delta_u \\ \vdots & \vdots \\ a_n : & +n \cdot \delta_u \\ a_{n+1} : & -n \cdot \delta_u \end{cases} \tag{4.16}$$

$$u_{ref,i+i}(a) = u_{ref,i} + i(a) * \delta_u \tag{4.17}$$

For simulations the values $\delta_u = 0.01$ and $n = 2$ have been applied. To further avoid unnecessary training examples involving speed assignments of zero and negative values, the output is given a lower boundary $u_{ref} \geq 0.01$ m/s through dynamicity in the action space excluding actions bringing

u_{ref} under this. When the vessel reaches the final waypoint i.e $p \approx p_{dock}$ as defined by the circle of acceptance, a signal is fed to the agent initiating a deceleration procedure bringing the vessel to rest. This is done by setting $\mu = 0$ and linearly decreasing u_d to 0 along the remainder of the path ensuring steady retardation to 0. Subsequently the training episode is counted as completed.

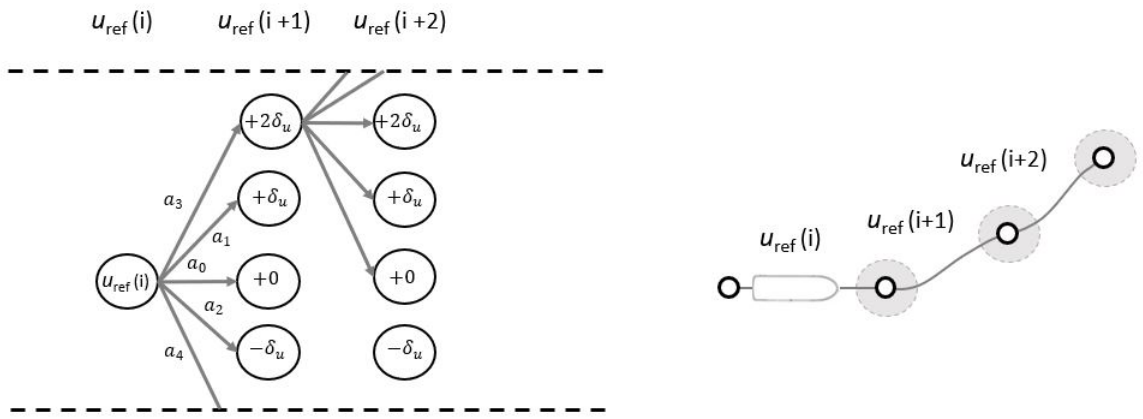


Figure 4.4.2: Visualization of action space for speed evaluation, defined with $n = 2$, exemplified by augmentation of $u_{ref}(i)$ and attainable values after a selection of a_3.

4.4.3 Reward function

The ultimate goal of RL is to maximize returned reward, as this serves as a measure of performance through training. Consequently the reward shapes the behaviour of the system and sets terms for convergence. For the task of speed evaluation there are many differences in how this function should be defined according to where emphasis is placed. We choose to inspect various aspects through multiple reward functions with components addressing associated factors. However some terminal reward components are global for all aspects and are addressed first. In the following definitions r_T and r_t denote terminal and immediate reward components, respectively.

With the path specified assumed to be safe and clear of collisions, the only immediate concerns involve failing to track the path and the final retardation at the quayside. To eliminate unfeasible path tracking, large penalties should be associated with state action pairs provoking this. As an undesired terminal condition measured as true or false, it's natural to impose this as a negative reward of constant value $r_{T,t_{max}} = -C_{T,t_{max}}$ if $t_{p_0 \to p_t} > t_{max}$. Where $C_{T,t_{max}} > 0$ is a tunable constant. All applied constants with their respective values can be found at the end of the section where introduced. Similarly we need a terminal reward associated with the desired terminal condition of reaching the quayside. We want this to be done in a manner ensuring calm and precise retardation to p_{dock} and could also be done through a constant feedback returned under certain criteria. However in the aim of a more intelligent system, we apply a reward based on the precision of the docking through a measure of resulting overshoot of p_{dock}. Assuming sufficient path tracking with ν approximately tangent to the path, the reward is calculated as p exceeding p_{dock} along the paths tangential angle

$$r_T, p_{dock}(p) = max\left(\left| \begin{bmatrix} cos(\psi_d) \; sin(\psi_d) \end{bmatrix} (p - p_{dock}) \right|, 0\right) \cdot C_{T,p_{dock}} \text{ if } p \approx p_{dock} \tag{4.18}$$

Where $C_{T,p_{dock}} > 0$ is a tuning constant. For this to be measured the final update of the agents memory is delayed a time $t_{delay} = 30$ s. Other reward parameters for the final update are recorded as the vessel reaches p_{dock}. With these components global for all cases, three different cases of speed evaluation have been studied and tested.

Table 4.1: Applied values of tuning constants for global components.

Description	Symbol	Value
Constant for unfeasible maneuvering	$C_{T,t_{max}}$	2000
Constant for overshoot of p_{dock}	$C_{T,p_{dock}}$	500

a) Promptitude with precision focus

For ferry operations and harbours running busy schedules, the promptitude of docking is of big concern. For this reason we aim to train the agent at taking the vessel from the initial conditions to the quayside as quick as possible. This translates to reducing the time spent maneuvering along the path. However we choose to keep negative rewards to denote unfeasible options, such that if the agent discovers a feasible although possibly sub optimal strategy it can improve over this. This is based on the idea of memory being initialized at 0. We therefore propose shaping the optimization goal of time to revolve in the positive range, through the use of inverse reward design.

A possible approach is to rate the agents actions though direct evaluation of total elapsed time, by a component returning a time dependent value when $p \approx p_{dock}$ and zero otherwise.This form of conditional delayed reward could through back propagation improve speed assignments in regards of thrust but would be a timely process, with possible flickering between actions. We therefore propose directing the agent towards optimal decisions earlier, through additional smaller rewards across individual path segments. The delayed time reward $r_{T,t}(t)$ and immediate time reward $r_{t,t}(t)$ are tuned through constants $C_{T,t} > 0$ and $C_{t,t} > 0$.

$$r_{T,t}(t) = \frac{C_{T,t}}{1 + t(p_{dock})} \text{ if } p \approx p_{dock} \tag{4.19}$$

$$r_{t,t}(t) = \frac{C_{t,t}}{1 + t(p_0) - t(p_t)} \tag{4.20}$$

Here $t(p_0)$ and $t(p_t)$ denotes the time for which the vessel is at waypoint p_0 and p_t. This should be sufficient for the agent to favour actions leading to reduced time. Further to ensure path following is done in an accurate manner we propose component based on precision, formulated as a boundary function dependent on the vessels offset in position $\tilde{p} = p - p_d$ and an error tolerance $err_p > 0$

$$r_{t,p}(\tilde{p}) = \begin{cases} -C_{t,\tilde{p}} \cdot ||\tilde{p}||^2_{max} & \text{if } ||\tilde{p}|| < err_p \\ -C_{t,\tilde{p}}(err_p^2 + \frac{1}{2} \cdot 300^{(||\tilde{p}||_{max} - err_p)} - \frac{1}{2}) & \text{if } ||\tilde{p}|| \geq err_p \end{cases} \tag{4.21}$$

Here $||\tilde{p}||_{max} = max(||\tilde{p}_{p_0 \to p_t}||)$ is the maximum logged value of the first norm of $\tilde{p}$ along a path segment, and $C_{t,\tilde{p}} > 0$ is a tuning constant. The component makes small adjustments trough minor penalties for small $\tilde{p}$ and heavily penalizes actions violating the precision requirements sat by err_p. The behaviour of the reward component can be seen visualized in Figure 4.4.3 for $C_{t,\tilde{p}}$ = 500 and $err_p = 0.1$ m.

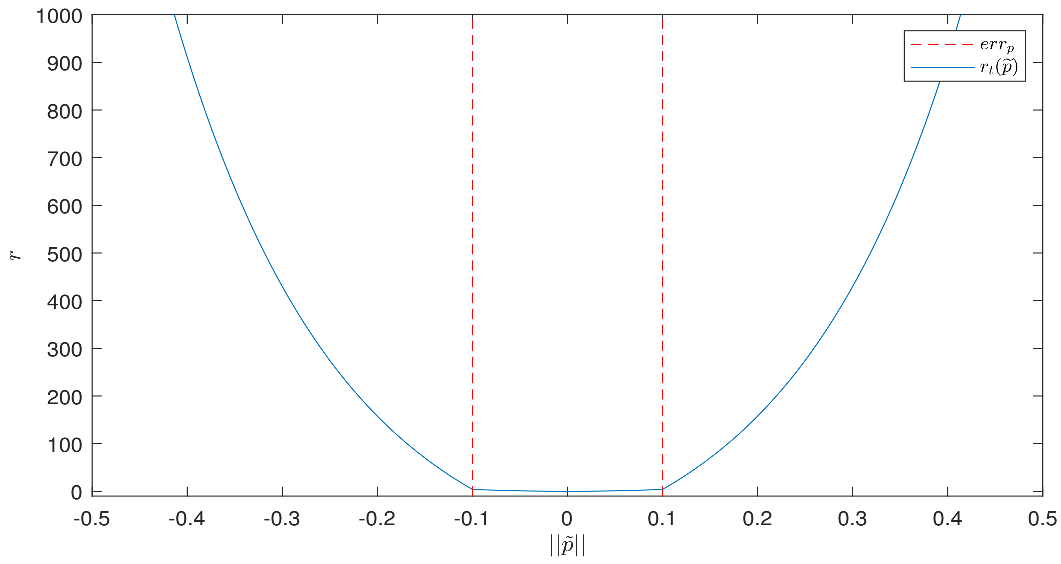


Figure 4.4.3: Offset dependent reward component $r_t(\tilde{p})$ with $C_{t,\tilde{p}} = 500$ and $err_p = 0.1$ m.

This makes the total reward function defined as

$$R = \begin{cases} r_{T,t}(t) + r_{T,p_{dock}}(p) & if p \approx p_{dock} \\ r_{T,t_{max}} & \text{if } t_{p_0 \to p_t} > t_{max} \\ r_{t,t}(t) + r_{t,p}(\tilde{p}) & \text{otherwise} \end{cases} \tag{4.22}$$

In the simulations the following parameters have been applied for tuning of the reward function

Table 4.2: Applied values of tuning constants for time optimization.

Description	**Symbol**	**Value**
Constant for terminal time reward	$C_{T,t}$	8000
Constant for continuous harvested time reward	$C_{t,t}$	4000
Constant for offset dependent reward	$C_{t,\widetilde{p}}$	400

b) Energy efficiency

Another interesting aspect is possibilities of optimizing energy efficiency. For this mission there are many options, such as inspecting resistance through damping. Yet, a strength of model-free RL is the ability to directly optimize complex parameters independent of insight in underlying features. A more beneficial approach would therefore be to minimize power output $P = \left\|\nu_r^\top \tau\right\|$ along the path directly. However, measuring ν_r can be quite difficult, especially in docking environments with advanced current fields. We therefore choose to focus on minimization of τ, as it is simple to measure and holds direct relation to P. The components are constructed in a similar manner to the time based rewards used in the previous case as defined in Equations 4.19 and 4.20, by integrating the first norm of applied thrust across a path segment:

$$r_{T,\tau}(\tau) = \frac{C_{T,\tau}}{1 + \int_0^{t(p_{dock})} ||\tau||} \text{ if } p \approx p_{dock} \tag{4.23}$$

$$r_{t,\tau}(\tau) = \frac{C_{t,\tau}}{1 + \int_{t(p_0)}^{t(p_t)} ||\tau||} \tag{4.24}$$

Where $C_{T,\tau} > 0$ and $C_{t,\tau} > 0$ are tuning constants of the terminal and immediate thrust rewards respectively.

$$R = \begin{cases} r_{T,\tau}(\tau) + r_{T,p_{dock}}(p) & if p \approx p_{dock} \\ r_{T,t_{max}} & \text{if } t_{p_0 \to p_t} > t_{max} \\ r_{t,\tau}(\tau) & \text{otherwise} \end{cases} \tag{4.25}$$

Table 4.3: Applied values of tuning constants for thrust optimization.

Description	**Symbol**	**Value**
Constant for terminal thrust reward	$C_{T,\tau}$	2000
Constant for immediate thrust reward	$C_{t,\tau}$	1000

c) Combined energy and time

As we in reality want docking processes fulfilling both of the above defined goals, we study a combination of the time and energy optimization. This is done through merging components of the above defined reward functions into a single function

$$R = \begin{cases} r_{T,t}(t) + r_{T,\tau}(\tau) + r_{T,p_{dock}}(p) & \text{if} p \approx p_{dock} \\ r_{T,t_{max}} & \text{if } t_{p_0 \to p_t} > t_{max} \\ r_{t,t}(t) + r_{t,\tau}(\tau) & \text{otherwise} \end{cases} \tag{4.26}$$

4.5 Path evaluation for docking

The goal of the second sub problem is to evaluate a good entrance path, guiding the vessel to its desired pose at the quayside. As the desired heading is kept at a tangent to the path this includes focus on correct angles of approach. For this we relax the assumption of a pre-specified path, and instead introduce a constant reference speed $u_d = u_{const}$ based on results of speed evaluation. Additionally we assume generated paths to be in compliance with the vessels maneuvering constraints.

As a foundation for the learning process we need to define the specific target to be reached, along with any additional terminal conditions. For speed evaluation this was simply as a waypoint p_{dock}, or termination due to loosing track of the path, knowing one of the two would occur. Adopting these definitions, along with a desired angle ψ_{dock} for path evaluation is possible, but could be rather difficult as the probability of hitting this state exactly and hence building experience is low. To ensure higher probabilities of reaching the target we therefore extend the target conditions to a larger area A_{dock} covering a radius R_{dock} from the chosen docking position p_{dock}:

$$A_{dock} = \{(x, y) \in \mathbb{R}^2 : (x - p_{dock,1})^2 + (y - p_{dock,2})^2 \leq R_{dock}^2\} \tag{4.27}$$

Where $s \in A_{dock}$ is defined as terminal states for an episode. Further we choose to not impose ψ_{dock} as a terminal condition, but rather a desired state handled by rewards. As we aim to let the vessel navigate freely under a assumption of a consistently feasible path to track, there arises a exploratory possibility of the vessel sailing off forever. To prevent this along with limiting computational complexity we set some limits for the state space to be explored. We do this according to defined environment presented in Section 4.1, and set state space of the training process based on the boundaries to $S = \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2 : \begin{matrix} 0 \leq x \leq 6 \\ 0 \leq y \leq 6 \end{matrix} \right\}$. Consequently each episode of training is terminated if the vessel state comes outside this space. This makes the total terminal conditions of a training episode be defined as:

$$s_T = \begin{cases} p \in A_{dock} \\ p \notin S \end{cases} \tag{4.28}$$

4.5.1 Algorithm

For the process of training to evaluate entrance paths it's natural to assume there will be a high amount of collisions early on from exploratory actions. Therefore real-world training from scratch might not be a good idea and we again propose the use of an off policy algorithm. However, we want to ensure sufficient risk avoidance by staying clear of states potentially leading to this. This risk avoiding ability can be found in SARSA, which is an on-policy algorithm. This leads us to Expected SARSA as discussed in Section 3.9.

The specific setup of the agent is thus done though the update algorithm of Expected SARSA

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha \left[r_{t+1} + \gamma \sum_a \pi\left(a | s_{t+1}\right) Q\left(r_{t+1}, a\right) - Q\left(r_t, a_t\right) \right] \tag{4.29}$$

To make for off-policy learning we let updates be done through a product of each Q-value and their associated probability calculating an expected return. Selecting a greedy policy as seen in Q-learning and SARSA, the algorithm can make off-policy updates accounting for all possibilities and will converge to stay clear of states potentially leading to large negative rewards. Just as for speed evaluation we handle exploration through ϵ-greedy action selection. Yet, where we slowly trim the 1D speed assigned for each segment as an optimization focused problem, a 2D path evaluation includes a much broader state space and thus requires more exploration. We therefore propose an ϵ function of exponential decay. This results in maximal early exploration building general knowledge of various docking entries followed by optimization of the best. The tactic also highly improves results shown from a complicated reward function later defined, where backpropagation of delayed rewards is crucial. The decay function is depicted in Figure 4.5.1.

$$\epsilon(ep_i, ep_{tot}) = e^{\left(-4 \cdot \frac{ep_i}{ep_{tot}}\right)^{1.1}} \tag{4.30}$$

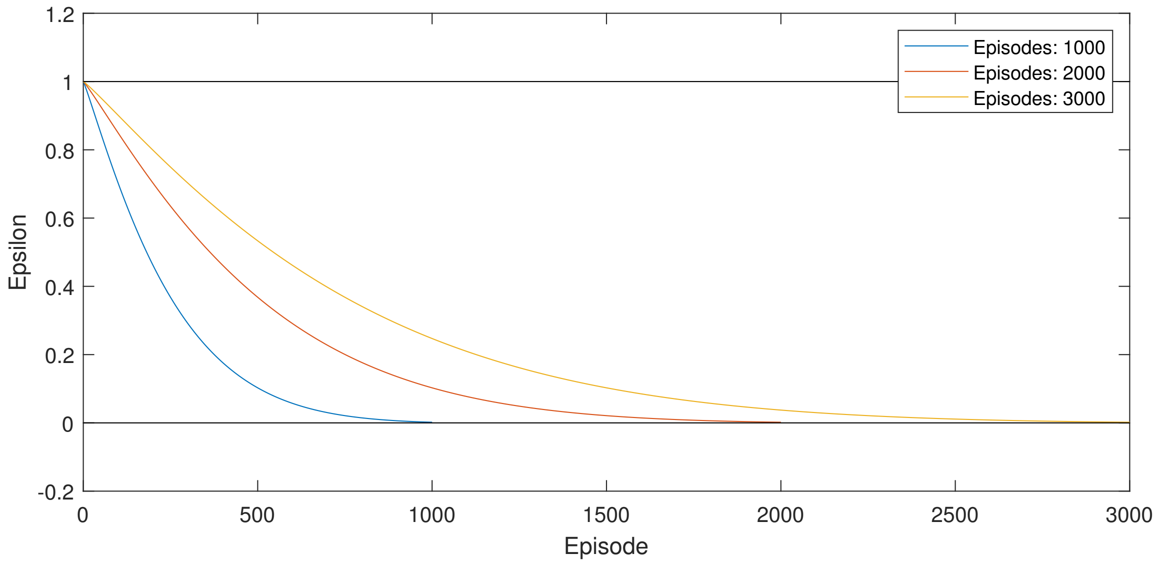


Figure 4.5.1: Exponential function for ϵ decay for path evaluation, as defined in Equation 4.30.

Other important factors applied for the algorithm is $\gamma = 0.9$ and $\alpha = 0.3$.

4.5.2 Action space

Before the update law is explicitly defined, a creation of an appropriate mesh over the 6m X 6m environment is required due to the algorithms assumption of discrete states. Selecting sparseness of this mesh can be seen as a trade-off between providing the vessel with an abundance of states to maneuver over and limiting the state space to a number ensuring convergence under a reasonable amount of episodes. For this we suggest a grid of 0.25 m X 0.25 m squares to be sufficient. Now in order to facilitate for various guidance strategies through the environment in, we let the RL agent to take control of p_t. As the agent is triggered for a discrete update the agent is to evaluate upcoming target waypoint $p_{t,RL} = p_{t,i+1}(p_{t,i})$. Setting course towards a new target waypoint, p_0 is sat to p_t and the new p_t is sat to $p_{t,RL}$ according to the update law discussed in Section 4.2.2, extending the complete path by a new path segment. With ψ_d defined tangent to the path, this renders action space formulation highly dependent on vessel maneuvering abilities.

As we aim to let the agent provide guidance in a free manner we need options of keeping a steady path as well as curving left and right at various angles. We propose the formulation of an action space sat up to alter the average path tangential angle $\theta_T = tan^{-1}(p_{t,2} - p_{0,2}, p_{t,1} - p_{0,1})$ by a variable $\delta_{\theta_T,i}$ and extend the path segment by a positive length l_i from p_t for $i = 2, 4, ..., n$. This results in an action space consisting of $n+1$ actions, where an action a_0 refers to keeping a steady course by applying the characteristics of the current θ_T. Through this process the upcoming waypoint $p_{t,RL}$ is generated based on the current p_t.

$$A_{path} = \begin{cases} a_0 : & \delta_{\theta_T,0} = 0 & l_i = l_0 \\ a_{i-1} : & \delta_{\theta_T,i-1} = +\delta_{\theta_T,i} & l_{i-1} = l_i \\ a_i : & \delta_{\theta_T,i} = -\delta_{\theta_T,i} & l_i = l_i \\ \vdots & \vdots & \vdots \\ a_{n-1} : & \delta_{\theta_T,n-1} = +\delta_{\theta_T,n} & l_{n-1} = l_n \\ a_n : & \delta_{\theta_T,n} = -\delta_{\theta_T,n} & l_n = l_n \end{cases} \tag{4.31}$$

$$p_{t,RL}(a, \theta_T, p_t) = p_t + \begin{bmatrix} l(a,\theta_T) \cdot cos(\theta_T + \delta_{\theta_T}(a,\theta_T)) \\ l(a,\theta_T) \cdot sin(\theta_T + \delta_{\theta_T}(a,\theta_T)) \end{bmatrix} \tag{4.32}$$

The proposed action space formulation however works best under continuous state spaces. As we take use of discrete states the variables in the action space have to be defined according to the meshed environment to keep discretized characteristics for upcoming states. This can impose a lengthy and finicky tuning process. For simplicity we thus apply the above formulated idea of extending the next path segment by applying an angle difference and a path length to coincide according to the meshed grid. As depicted in Figure 4.5.2 the possible path extensions of A_{path} is defined to span over a defined area where a series of discrete states are available for selection. For the simulations made in this thesis we use $n = 2$ as the applied mesh of the environment is relatively small in comparison to CSAD's length over all of 2.578 m. This facilitates small enough step sizes in waypoint evaluation for the vessel to perform maneuvering tasks. Additionally we let each segment extend a minimum of two blocks, in order to simplify maneuvering. This leaves the vessel with three options, one for steady course and two making turns in their respective direction. More options of steeper and more extensive turning of the vessel was also tested with, but proved quickly became complicated for vessels maneuvering abilities and for the intended research was concluded an unnecessary extension of A_{path}.

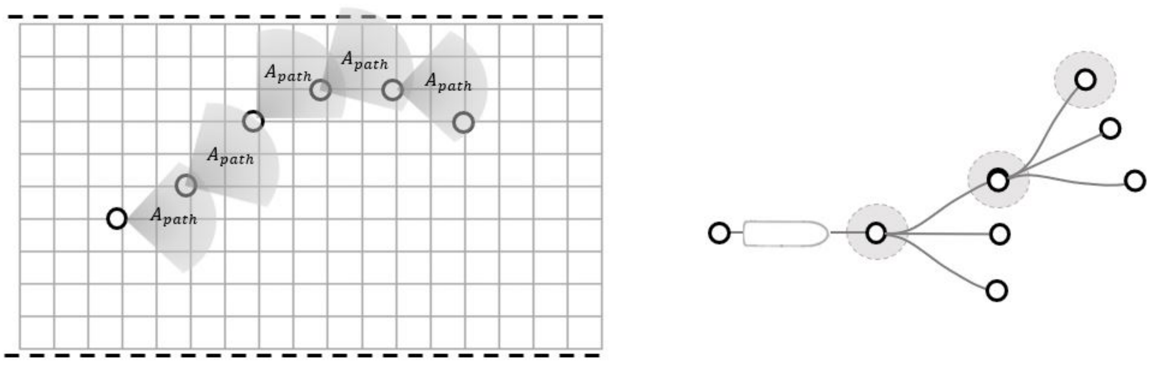


Figure 4.5.2: Visualization action space for path evaluation, defined with $n = 2$.

4.5.3 Reward function

In order to achieve intended goals of path evaluation, we need to design the fitting reward components. As this objective is more complicated than the speed evaluation it's natural that the resulting design will be more complex as well. With ψ_d defined tangent to the path and an action space formulation providing limited turning possibilities, a strong interconnection exists between chosen actions and future available configurations. Hence, early mistakes can prove very difficult to recover from and pose big impact on the future of the path. The agent is therefore dependent on a highly descriptive reward function, indicating desirable behavior with variations from early in the docking process to the final stages of reaching the quay.

An obvious indicator needed for the agent, is for what direction it should point the vessel. This is perhaps the most important part of the reward with underlying goal for the vessel to at all reach the quayside. The most straightforward and intuitive way to do so is to let the agent know if the chosen action brought it closer to the goal or not. We propose this feedback by applying a two dimensional euclidean distance as a measure of the distance between p and p_{dock} , denoted as d. More specifically the reward associated for a chosen path segment is defined as the reduction in d from p_0 to p_t multiplied by a tuning constant $C_{t,dist} > 0$

$$d = \|p_{dock} - p\|_2 = \sqrt{(p_{dock,1} - p_1)^2 + (p_{dock,2} - p_2)^2} \tag{4.33}$$

$$\Delta_{dist} = d_{p_0} - d_{p_t} \tag{4.34}$$

$$r_t(\Delta_{dist}) = C_{t,dist} * \Delta_{dist} \tag{4.35}$$

All tuning constants used in the path evaluation problem is shown in Table 4.4 with there respective value applied for simulations. This component should help training converge to guiding the vessel towards the quayside and p_{dock}.

Under the circumstances where the vessel is able to reach the designated docking position by the quay, we need a measure of the quality of arrival. Putting safety ahead, the most important measure of this approach will be whether or not the vessel is able to arrive with the correct heading. For our proposed vessel the natural heading would be the coming alongside at a parallel angle (perpendicular if a ferry situation is inspected). Hazardous situations and problems could arise if the vessel arrive at other headings, and thus a good tactic would be applying a terminal reward component based on the offset angle from the desired heading upon reaching the quay $\psi_{offset} = |\psi_{dock} - \theta_T|$. As the most dangerous angle to come in at would be at 90° we develop a reward function based on this angle being least desired. With $\psi_{offset} \in [0, 90]$ we define the reward function as a linear increase from $r(\psi_{offset}) = 0 : \psi_{offset} = 90$ to a max output of reward given $\psi_{offset} = 0$:

$$r(\psi_{offset}) = \frac{(90 - \psi_{offset})}{C_{\psi_{offset}}}, \text{if } p \in A_{dock} \tag{4.36}$$

Where $C_{T,\psi_{offset}} > 0$ is a tuning constant of the reward function. In addition to the main goal of reaching the quay at the correct heading, we also introduce a sub goal of reducing total thrust. However we want to impose a much greater value on the safety of entering at correct angles compared to thrust usage. To shape this trade-off between the two we propose a terminal reward triggered by the vessel entering the targeted docking area formulate as the reward of the total thrust raised to the reward of the entrance heading. We apply the delayed thrust reward as defined Equation 4.23 to get the complete terminal reward for reaching the docking state:

$$\begin{aligned} r_{T,dock} &= r_T(\tau, t)^{r_T(\psi_{offset})} \text{ if } p \in A_{dock} \\ \text{where } r_T(\tau, t) &= \frac{C_{T,\tau}}{1 + \int_0^{t(p_{dock})} ||\tau||} \end{aligned} \tag{4.37}$$

Additionally we supplement an immediate thrust reward as defined in Equation 4.24, to elevate the agents focus on thrust reductions throughout the process:

$$r_t(\tau, t) = \frac{C_{t,\tau}}{1 + \int_{t(p_0)}^{t(p_t)} ||\tau||} \tag{4.38}$$

It's worth mentioning that as a sub goal independent of distance and heading, this has to be tuned significantly lower to decrease chances of alone attracting the focus and introducing local optima.

The reward function so far should help the agent navigate the vessel in order to reach the quay with desired angles and reduce thrust in the process. Through initial testing of these components this was confirmed true. However, due to previously discussed limits on turning rendering available configurations having high correlation far into the past, the vessel was only able converge to smaller reductions in offset angle due to back propagation of $r_{T,dock}$, upon reaching the quay at early trials. Thus it's clear that an earlier indicator of the desired heading is required. We could impose a constant weighted reward similar to that of $r_t(\Delta_{dist})$, but this would most likely confuse the agent as of what to value between reducing d and reducing ψ_{offset}, and cause problems in the early stages of the docking process. As a better solution to this we propose a new distance dependent reward through the use of potential field characteristics, returning desired heading indications throughout the entire process

$$r_t(\psi_{offset}, d) = C_{t,dist} \cdot e^{-d} \cdot r(\psi_{offset}) \tag{4.39}$$

The reward denotes the offset angle to be of little importance at distances far from the quay, and and exponentially increases as d decreases. As a second reward component dependent on the distance to the quay, the agent can thus learn to early in the process prioritize actions for closing in on the target while prioritizing optimal headings more and more towards the end. This prioritization is further tuned by two constants $\alpha_{\psi_{offset}}$ and $\alpha_{\Delta_{dist}}$. The increase in importance of vessel heading can be seen depicted in Figure 4.5.3.

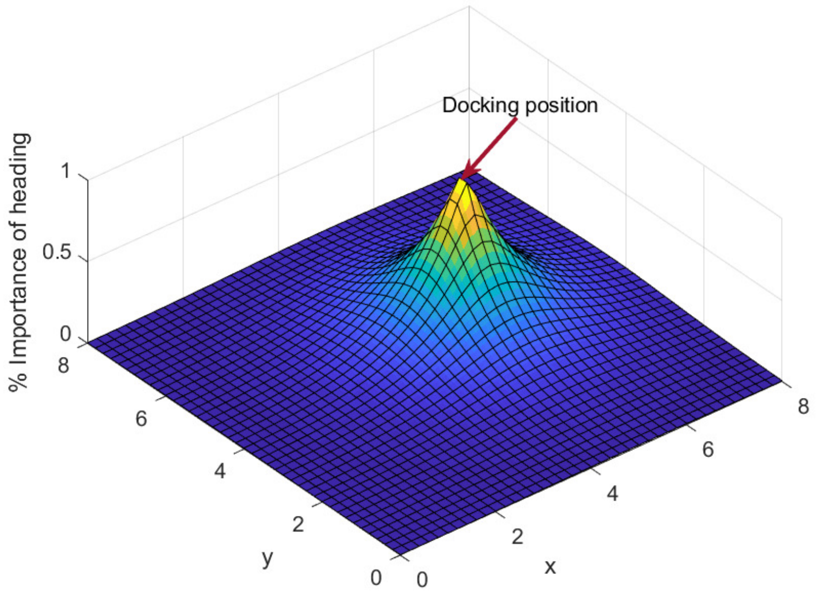


Figure 4.5.3: Importance of vessel heading approaching the docking position.

Lastly as we aim to develop paths of sufficient safety, staying clear of collisions is an important aspect of the process. We have already defined a fitting reward component if the agent is able to guide the vessel to the terminal state of A_{dock}, but also need to account for potential terminal state due to collisions i.e moving outside of the defined boundaries. As previously mentioned we chose Expected SARSA in order to reduce risk of these events, and therefore need to apply big penalties. In our environment this is equal to the vessel moving outside the defined boundaries. We propose a penalty of constant value under the event of a collision, defined as negative reward $r_{Tcol} = -C_{T,col}$ if $p \notin S$. Where $C_{T,col} > 0$ is a tuning constant. The back propagation of penalties resulting from such events will not only help Expected SARSA converge to paths keeping distance away from collisions, but also help direct it towards the only terminal part of the process not including penalties, i.e the quayside. The complete reward function for the path evaluation thus becomes

$$R = \begin{cases} r_{Tdock} & \text{if } p_t \in A_{dock} \\ r_{Tcol} & \text{if } \eta \notin S \\ (\alpha_{\Delta_{dist}} \cdot r_t(\Delta_{dist}) + \alpha_{\psi_{offset}} \cdot r_t(\psi_{offset}, d)) + r_t(\tau, t) & \text{otherwise} \end{cases} \tag{4.40}$$

Table 4.4: Applied values of tuning constants for path evaluation.

Description	**Symbol**	**Value**
Constant for change in distance	$C_{t,dist}$	10
Constant for offset in docking angle	$C_{\psi_{offset}}$	5
Constant for total thrust applied	$C_{T,\tau}$	5000
Constant for thrust applied	$C_{t,\tau}$	15
Constant for collision penalty	C_{Tcol}	1000
Constant for prioritizing $r_t(\Delta_{dist})$	$\alpha_{\Delta_{dist}}$	0.9
Constant for prioritizing $r_t(\psi_{offset}, d)$	$\alpha_{\psi_{offset}}$	0.2

Chapter 5

Simulations and results

The following sections present the results of training in a docking scenario. Here the most significant measures been presented and discussed, while additional supporting results can be found in Appendix B. Training is carried out from an initial memory of zero values, to improved values representing optimized path parameters according to the previously discussed foci and reward functions. Additionally, an approach for improved results for path evaluation is tested by suggesting an initial path for the vessel, letting the agent optimize with this as a starting point. The proposed method is a discrete approach to RL. As the vessel operate with continuous states, an update logic for discrete changes is implemented along with a discrete state space representation. The level of these discretizations along with frequency of updates influences precision of optimization due to narrowing down the options of the agent and may be a limiting factor. Yet, increasing to finer discretizations would result in higher computational complexity.

The applied vessel model (CSAD) proved a sufficient high fidelity model for the intended tests. However, during early trials it became clear that the vessels high block coefficient and strict thrust allocation caused high constraints in maneuvering. This restricted the vessel to gentle turns and low speeds. Hence the definition of the corresponding action spaces are limited to work over a more narrow range. Additionally this posed the vessel to be susceptible to environmental influences. In general current velocities above $0.3[m]$ seemed to give rise to maneuvering problems. Still the presented results gives an idea of possibilities around maneuvering of drill ships and vessels with similar maneuvering ability as CSAD. Vessels with with better dynamic stability however are likely to be open to improvements in term of extended action space and thus possibly better solutions.

The results also reflect the slowness of convergence bound to RL, being one of the main limitations of the technique. Hence the amount of episodes needed for training is substantially higher than what would be expected of a human operator. It's worth noting that progressive behavior is somewhat related to the degree of randomness present, defined by ϵ. It's also worth noting that due this randomness in exploration, results may differ between runs. Magnitudes of results presented reflect the application of a vessel in model scale and are thus given in cm.

5.1 Results for speed evaluation

The training progress for speed evaluation consisted of episodes initialized with path variables $p_{0,0} = [0,0]^\top$, $p_{t,0} = [0.5, 1]$, $u_{ref,0} = 5$ cm/s, and the vessel positioned at $p_{0,0}$ with heading tangent to the resulting segment and zero velocity i.e $\eta_0 = [0, 0, \frac{\pi}{3}]^\top$, $\nu_0 = [0, 0, 0]^\top$. As the longest segments of the path hold lengths of $\approx$ 50 cm, the lower speed boundary $u_{ref} \geq 1$ cm/s indicates target waypoints should be reached within 50 seconds. Reasonable time was therefore provided for the agent to reach the designated p_t through a contingency factor of $2 \Rightarrow t_{max} = 100$ s. The vessel then trained for 1000-1500 episodes depending on the case, evaluating speed assignments along the pre-specified path leading up to the quayside as defined by the waypoints in Table 5.1 and depicted in Figure 5.1.1. The training was executed with recordings of performance in terms of reward and associated parameters targeted for optimization. The earlier formulated cases are inspected for both calm waters and with an applied current of constant velocity in model scale $V_c = [1.58, 1.58, 0]^\top$ cm/s. Results of progression have been smoothed over a window of 10 episodes.

Table 5.1: Associated list of waypoints for speed evaluation.

Waypoint	0	1	2	3	4	5	6	7	8	9	10	11	12
Position x [cm]	50	100	150	200	250	300	350	400	450	475	500	500	500
Position y [cm]	100	150	175	175	175	175	200	225	275	325	375	450	550

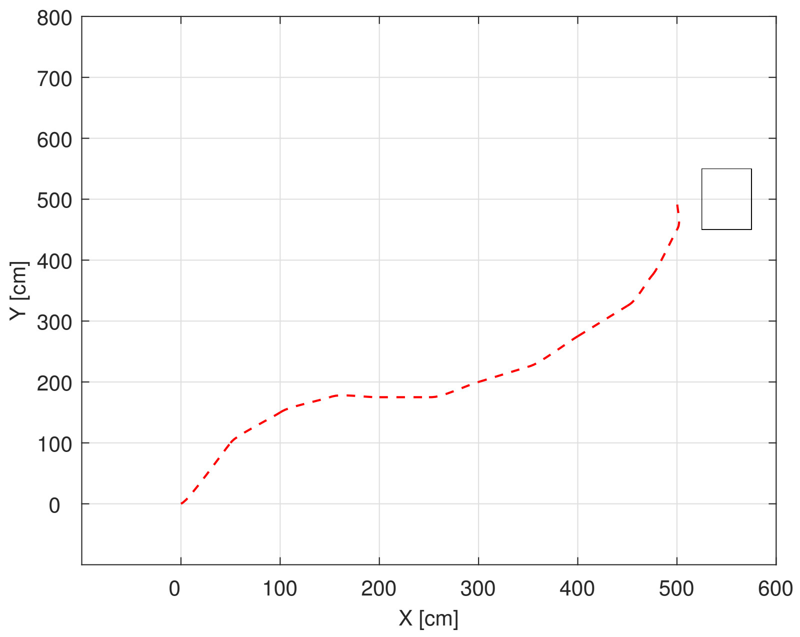


Figure 5.1.1: Pre-specified path for speed evaluation.

5.1.1 Promptitude with precision focus

Training for quick docking procedures under given precision requirements gave the progress seen in Figure 5.1.2. For the first $\frac{2}{3}$'s of training, the episode rewards inhabits tremendous dips due to inadequate maneuvering. This can be concluded as only violating the offset boundary condition as $||\tilde{p}|| > err$ and potentially loosing track of the path as $t_{p_0 \rightarrow p_t} > t_{max}$ can return penalties of this magnitude. After 1000 episodes however, the agent seems to have evaluated upper bounds of speeds along the path and is able to comply with precision requirements, showing increasing positive rewards. Inspecting the progress of elapsed time in Figure 5.1.3 it is clear that the intended purpose of decreasing time is fulfilled. Here training under the influence of a current concludes with greater time requirements than for without currents, which also is reflected by lower rewards. This is to be expected as influence from external forces complicates maneuvering, leading to requirements of lower speeds to uphold precision. Further it's worth noticing that the agent has visited strategies yielding time lower than for the final results. However, by comparison to rewards it can be seen that this is at the expense at inadequate maneuvering or overshoot of p_{dock}, which progress is depicted in Appendix B.1.1, Figure B.1.1. Lowering restrictions on precision and/or overshoot is therefore likely to yield quicker, although more inaccurate, docking processes.

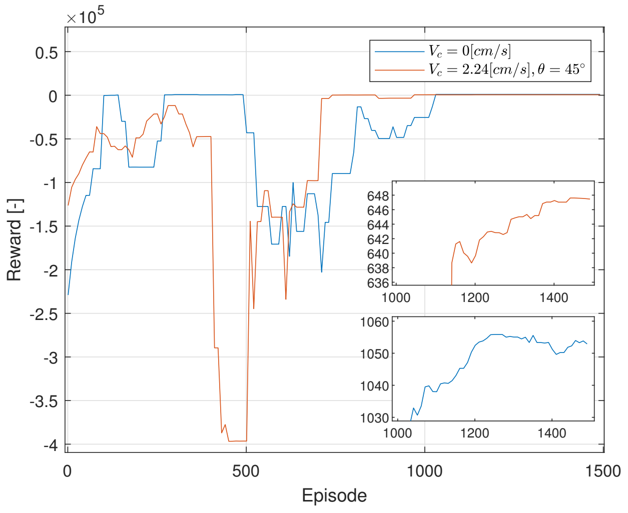


Figure 5.1.2: Progression of returned rewards. Focusing on time and precision.

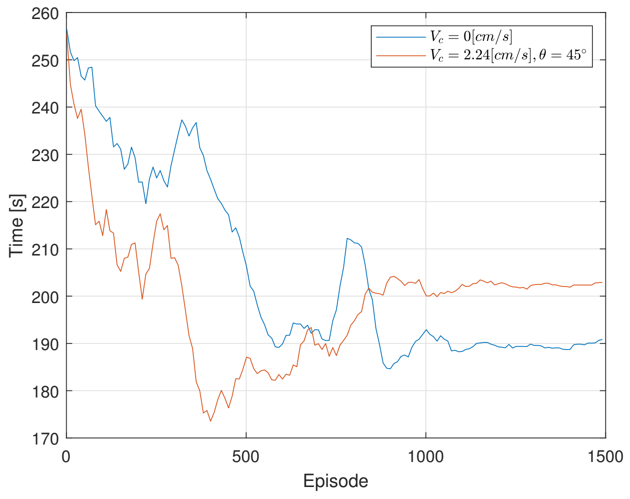


Figure 5.1.3: Progression of elapsed time. Focusing on time and precision.

Convergence to comply with precision criteria sat by the boundary function of $err_p = 10$ cm is confirmed by inspecting speed assignments and offsets post training, shown in Figure 5.1.4. The figure shows how u_d is increased to due to inverse rewards of time and reduced in events of peaking $||\tilde{p}||$, pushing speeds as much as possible without violating the boundaries for precision. An initial overshoot can be seen for in the case of no current, due to the vessel being initialized at rest while initial reference speed is sat to $u_{ref,0} = 5$ cm/s, creating an offset $||\widetilde{p}|| \approx 15$ cm as a result of heavy acceleration. For the case of applied current this problem seems to cease as acceleration is aided by the current, however similar increases can be seen at the final stages due to retardation against the current. Aside from this the vessel seems most susceptible to offsets at a length of the path ranging from 200 cm to 400 cm where there exists higher curvatures in the path. It can be noted as previously mentioned that the case of applied current result in lower speeds than for without. Aside from this the resulting speeds seem to follow the same trends, indicating speed augmentations are mostly bound to vessel constraints in regards to the path structure.

The effect of penalizing overshoot at p_{dock} also seems to work well, causing the agent to favour low speeds at the paths end for precise docking. This causes the vessel to reach the quayside at the lowest speed possible, carrying out precise placements at the quay with approximately zero overshoot confirmed by Figure B.1.1 in Appendix B.1.1. The results highly reflects the dependency on the action space, as the maximum speed alterations available (± 2 cm/s) causes the agent to begin retardation early ahead of the crucial parts. This dependency also results in lower assigned speeds than necessary by inspection of offsets, implying that an extension of A_{speed} to include greater alterations of u_{ref} might allow greater time reductions while still complying with precision. However, this comes as a trade of for computational demands.

Further visualisation of the results are given by mapping u_{ref} along the entry path in Figures 5.1.5 and 5.1.6, showing how speeds are reduced to lower values around the first bend. Following this increases are made over the longer and slighter bend to a maximum value of $u_{ref} = 8$ cm/s before retardation is initiated to reach $u_{ref} = 1$ cm/s by the quay.

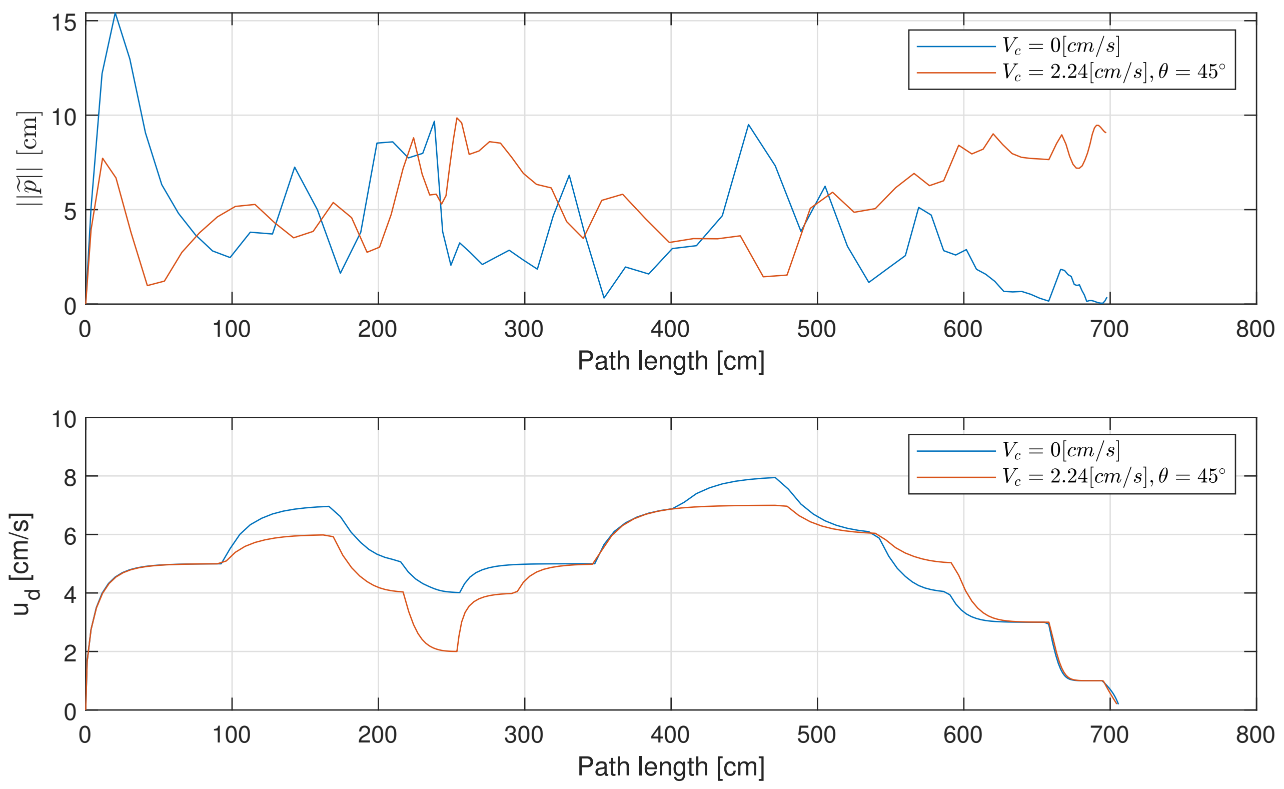


Figure 5.1.4: First norm of position offset and desired speeds along path length post training. Focusing on time and precision.

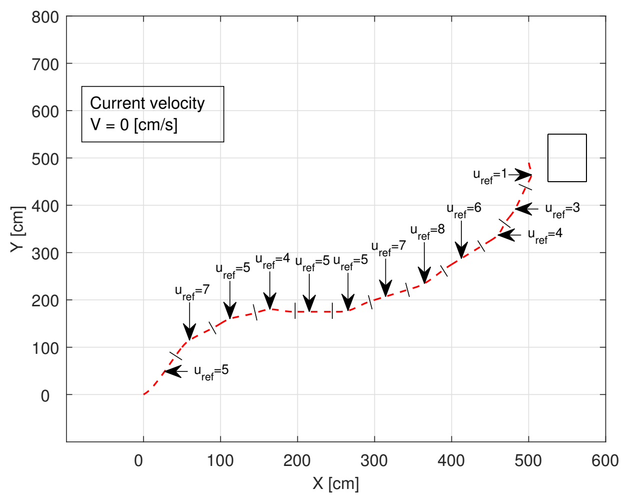


Figure 5.1.5: Assigned speeds along the entrance path post training [cm/s]. Without current, focusing on time and precision.

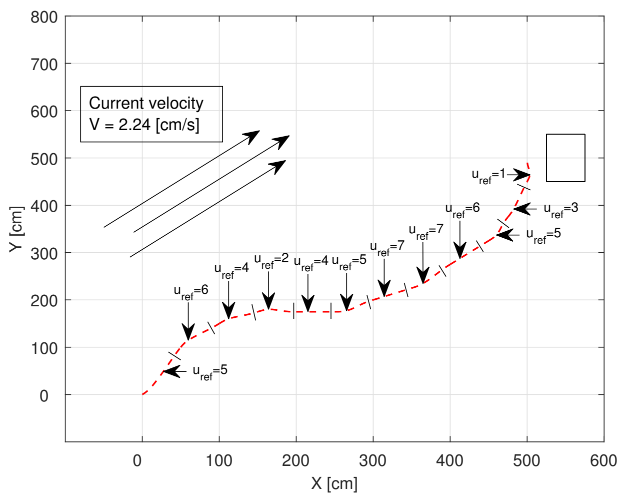


Figure 5.1.6: Assigned speeds along the entrance path post training [cm/s]. With current, focusing on time and precision.

5.1.2 Thrust reductions in current

Based on the previous case of optimizing time with focus on precision, an upper speed boundary of 5 cm/s is implemented as the median of the resulting speeds. Hence, reference speeds are bound in the interval $1 \leq u_{ref} \leq 5$. This is done to guarantee some precision for the current case and causes a reduction in dimensionality of the space to be explored. Thus the required amount of episodes was reduced to 1000. In addition the initial speed for the vessel $u_{ref,0}$ is changed to 3 cm/s as the median of the range of speeds.

As can be seen by the training progress in Figure 5.1.7, a much smoother curve is obtained due to absence of large penalties from unfeasible maneuvering. Also taking applied thrust into account, shown in Figure 5.1.8, this represents how the agent is able to increase rewards and correspondingly reduce outputs of thrust. As for the previous case, the influence of currents impose greater requirements than without. The goal of reduced overshoots at p_{dock} can also be seen fulfilled as represented by Figure B.1.2 in Appendix B.1.2. It is clear to see how the imposed upper boundary of u_{ref} also affects magnitude.

Figure 5.1.9 represents the final assigned speeds along the path length. Training without currents the agent seems to favour speeds as low as possible. This is as expected taking the control design model presented in Equation 2.13 into account, pointing to that lower τ should be required for low ν_r. For training under current however, the vessels seems to obtain a more unexpected result. For the initial and final stages of the path, the resulting speeds hold values in the range close to current velocity at 2.24 cm/s. But for the middle section of the path the agent selects maximum speeds of $u_{ref} = 5$ cm/s. At first this was presumed to be the result of some local optimum. However, running the training process multiple times with various training parameters this behavior consisted. Additionally, manually selecting the speeds closer to the expected results at $u_{ref} = 3$ cm/s along the entire path, showed 5.9% higher outputs in thrust and 24.5% higher outputs in power, confirming the agents results. This might be due to stronger transverse currents from reduced inclination in the path as seen in Figure 5.1.10, or the susceptibility to offset shown in the previous case, causing a desire to rush past this section. Another and more likely result however is that the superiority of high speeds over these segments have roots in the underlying thrust allocation and/or path parametrization. Nevertheless, the model-free approach fulfills the intended purpose of optimizing thrust for the underlying models through a new solution, which is an ability of RL discussed in Section 3.2. The comparison of the expected result and the agents result can be seen visualized in Figure B.1.3 in Appendix B.1.2.

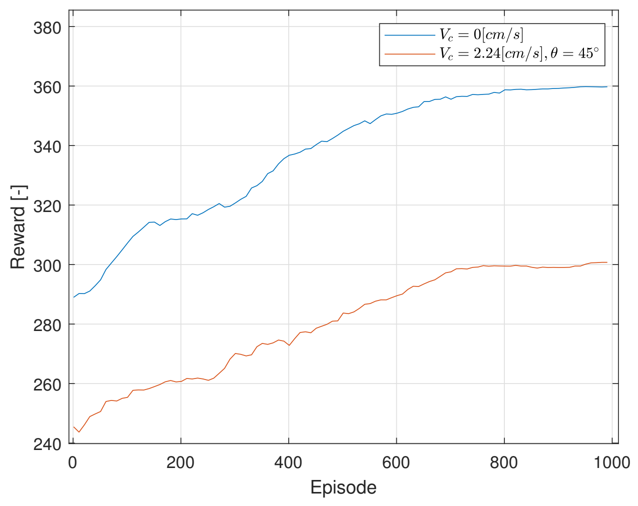


Figure 5.1.7: Progression of returned rewards. Focusing on thrust.

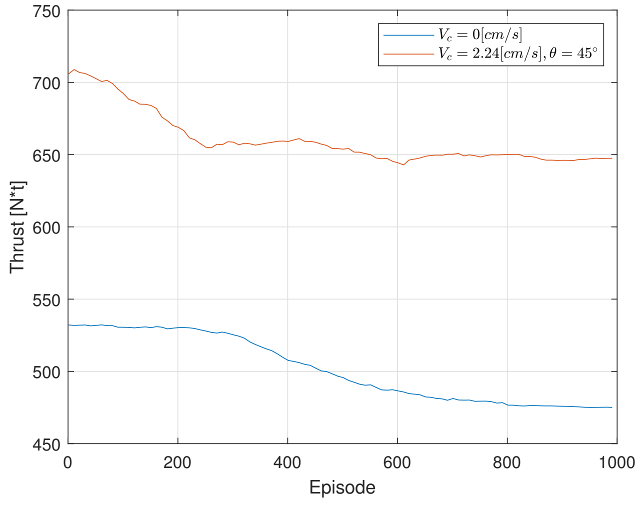


Figure 5.1.8: Progression of applied thrust. Focusing on thrust.

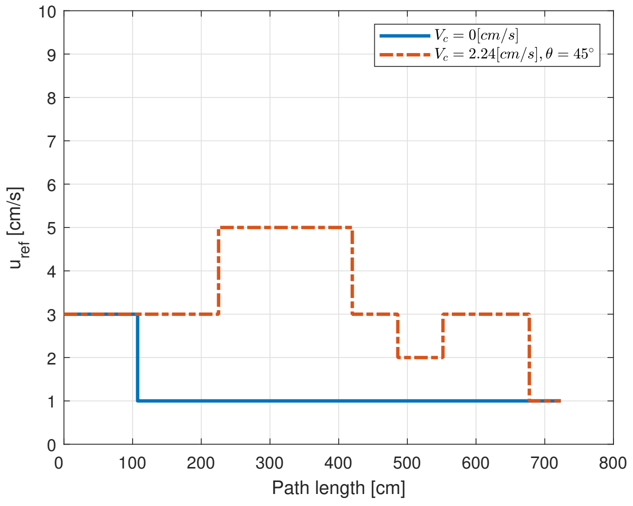


Figure 5.1.9: Assigned speeds along path length post training. Focusing on thrust.

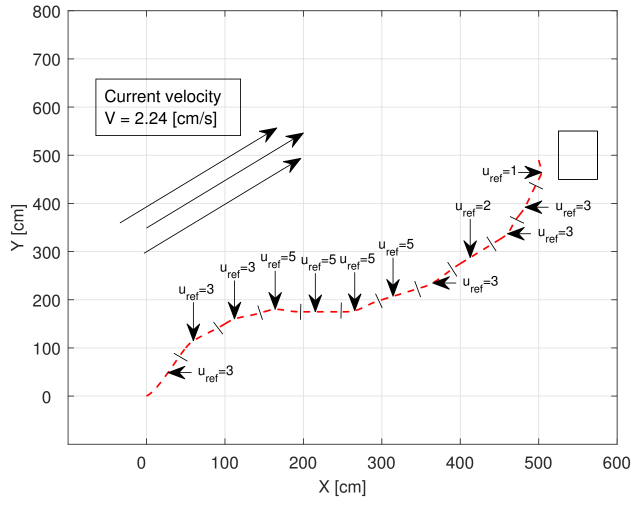


Figure 5.1.10: Assigned speeds along path post training [cm/s]. Focusing on thrust.

5.1.3 Reduction of time and thrust

Training the vessel in regards of both promptitude and energy efficiency we got the progress shown in Figure 5.1.11. Here the agent has managed improvement, but shows a tendency of oscillatory behavior in returned rewards. By further inspection this seems to result from inverse relations of thrust and time as reflected by Figure 5.1.12. In the reward design, heuristic understanding of codependency of the objectives has been ignored. Thus any action taken to increase rewards in terms of thrust is penalized by a decrease in time based rewards and vice versa, lowering stability. Training does conclude with reduced values of both components, but by running it multiple times the outcome seems little consistent in terms of a clear optima. Still, it's clear from the behavior how thrust is the dominating component. Issues of stability seemed mostly due to introduction of local optima where the agent got stuck at solely focusing on one of the objectives while ignoring the other. Consequently the approach also showed great sensitivity in weighting of reward components. However, this mostly occurred for minimizing thrust, likely due to co action from overshoot rewards pulling to lower speeds at the end. Such an outcome is presented in Appendix B.1.3 Figure B.1.7.

With tuning giving somewhat equal focus on the two objectives, the agent results in selecting outputs in a fluttering manner attempting to please both. This is shown in in Figure B.1.4 in Appendix B.1.3. The strategy shows similar traits to that of thrust reduction in current of the previous case, strengthening the possibility of this being a strategy utilizing the underlying thrust dynamics and path parametrization. Further refinement of path and thrust variables might weaken this high speed strategy and give the agent more trouble in terms of pleasing both objectives. Still, even if the agent got stuck at a local optima, the goal of minimal overshoot consistently succeeded, which is interesting as it suggest how parts of the reward function can have stable convergence independent of others. Figure B.1.6 in Appendix B.1.3 shows the current examples progress in overshoot. Many studies have been conducted within the field of RL on handling conflicting goals.

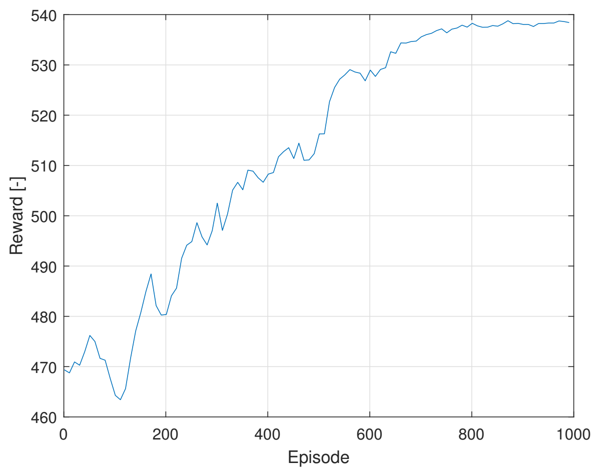


Figure 5.1.11: Progression of returned rewards. Focusing on time and thrust.

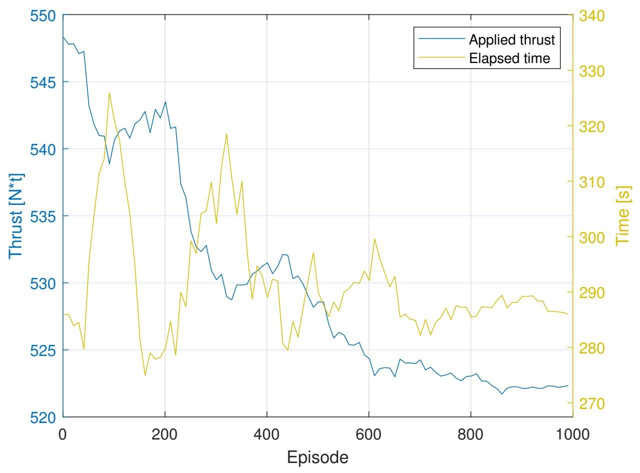


Figure 5.1.12: Progression of time and thrust. Focusing on time and thrust.

5.2 Results for path evaluation

Episodes of path evaluation are initialized under mostly the same conditions as speed evaluation, i.e $p_{0,0} = [0, 0]^\top$, $p_{t,0} = [0.5, 1]$, $\eta_0 = [0, 0, \frac{\pi}{3}]^\top$, $\nu_0 = [0, 0, 0]^\top$. The only difference is discarding the pre-specified entrance path and introducing a constant reference speed $u_{const} = 3$ cm/s. This is selected on the basis of accuracy for all possible path segments in accordance with results from precision based speed evaluation, and at the same time lower episode time requirements. Current is turned off. As a problem of greater dimensionality, training required a higher amount of episodes and exploration than for speed evaluations. The number of episodes found sufficient was $ep_{tot} = 2000$.

Training for optimal entrance paths resulted in the progress shown in Figure 5.2.1. The behavior of the rewards is somewhat expected as it carries the same traits as the exponential ϵ-decay function, causing higher exploitation of rewards as the process goes on. Yet, the agent is able to conclude at peaking rewards. It can be noted due to magnitude that the behavior is mostly represented by the terminal docking component $r_{T_{dock}}$. Still this is the most significant term representing the entrance path as a whole in terms of final heading, total applied τ, and that the vessel has reached A_{dock}. We can interpret due to the introduction of large values after 700 episodes that the agent is able to follow paths leading to A_{dock} with low ψ_{offset} from this stage. The early parts of training is thus sacrificed in terms of rewards to map general strategies and rewards scattered across the environment for later backpropagation and connections. Additionally the sub goal of thrust reduction can be seen depicted in Figure 5.2.2. Although being a secondly prioritized goal defined by lower reward magnitude, applied thrust shows clear reductions from refinement of discovered strategies.

Evolution of optimal strategies throughout training can be seen mapped in Figure 5.2.3. Here the path resulting from the final optimal policy is shown as a solid line, while dotted lines represents results of current optimal policies found throughout training, logged every 150th run as indicated by color. This shows how the agent early discovers a sub optimal path with less favourable ψ_{offset}. At around 700 runs however it discovers a strategy leading to $\psi_{offset} = 0$, succeeded by further refinements up until around episode 1300 concluding the optimal entrance path. This structure of this final path can easily be understood through the build up of the reward function. Early on it's clear how the distance reward $r_t(\Delta_{dist})$ is most profitable causing the vessel to head directly towards the quay. However, 1/3 of the way there, in a strive for long term rewards, the vessel shifts right sacrificing immediate $r_t(\Delta_{dist})$ to facilitate for later harvesting of heading and docking rewards, $r_t(\psi_{offset}, d))$ and r_{Tdock}, having greater magnitude closer to the quay. This shows that the exponential docking reward and weighting of $r_t(\psi_{offset}, d))$ in terms of a potential field serves the intended purpose of ruling close to the quay. The greater magnitude of these reaches the agent early in the process through backpropagation. Additionally rewards for low thrust values influence the results by smoothing the path and further promoting shorter and smoother paths, while collision penalties helped guiding the vessel away from incidents and thus also towards the quay.

One of the main complications of the training for path evaluation seemed to result from the interdependence of heading and augmentation in position. With ψ_d defined tangent to the path, this caused the guidance of the vessel to behave in an underactuated manner, making future configurations dependent far back over previous states. Thus the influence of delayed rewards and backpropagation became of high importance. In other words future available states were dependent many steps back into the past. This meant that the agent could select an action deviating from the current optimal policy, leading to higher long term rewards, without immediately understanding it as a connection of the newly chosen action. Noting this as an optimal action would require backpropagation only attained through selecting that same action multiple times, hence another reason for the large initial exploration. However, implementation of n-step backpropagation as discussed in Section 3.6 could yield high improvements and faster learning. On the positive side the applied ψ_d strategy makes proposed paths more applicable for potential underactuated vessels.

As many correlated tasks had to be fulfilled by the agent, e.g alter position, heading, reduce thrust etc., a complex reward function design was needed. As discussed in Section 3.5.2 great carefulness should be taken in regards of multiple goals due to possible introductions of local optima and shrouding a clear global solution. This also seemed the case for conflicting goals, as previously seen, and arises for immediate rewards like $r_t(\Delta_{dist})$ and $r_t(\psi_{offset}, d)$ due to correlation between θ_{dist} and Δ_{dist}. Although components of greater magnitude are more likely catch the agents focus, it required careful weighing of components and lowered stability of training. Such behavior of catching onto local optima might be what is reflected in the dips in reward throughout training. This could even hold up until completion in some cases such as can be seen by the figures presented in Appendix B.2. Here the agent is able to discover strategies resulting in better approach angles at about 600 runs, but gets stuck at a local optimum of distance and thrust reductions. This can also be seen by the associated rewards from the progress, holding occasional higher values than for the end result. Another downside of the imposed strategy is the high amount of collisions due to the extensive exploration. In total 29.75% of the docking attempts resulted in a collision (595 collisions), making potential real world training from scratch a dangerous and likely to be costly affair.

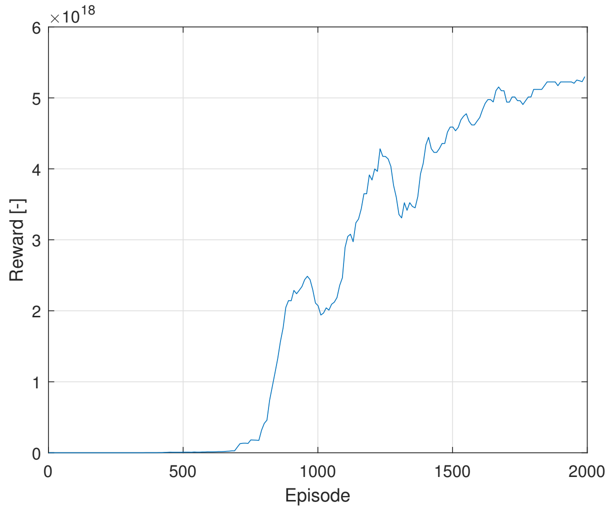


Figure 5.2.1: Progression of returned rewards for path evaluation.

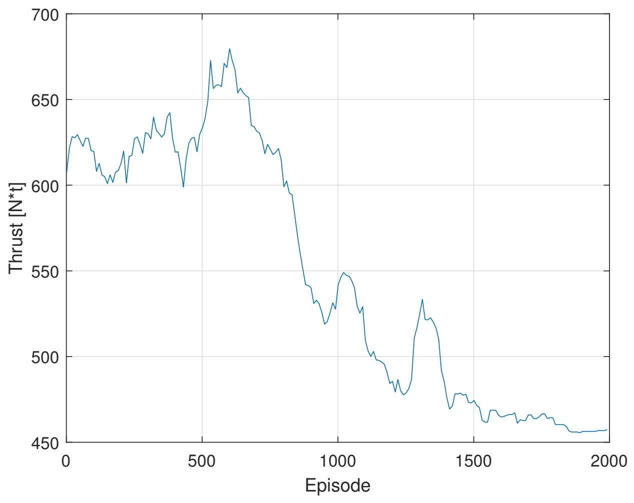


Figure 5.2.2: Progression of applied thrust for path evaluation.

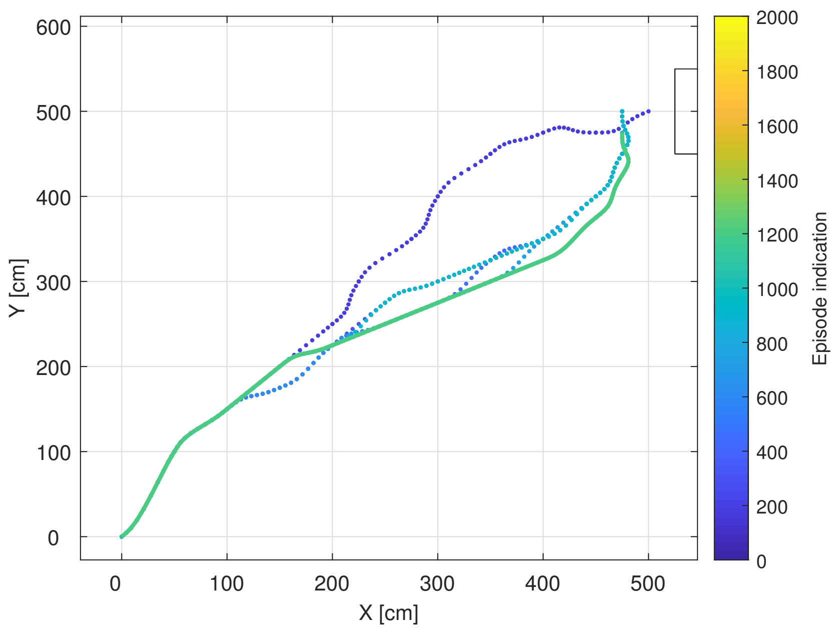


Figure 5.2.3: Evolution of optimal strategy for path evaluation with color indication. Solid line showing final result.

5.2.1 Path improvement

In an attempt to reduce required exploration and the amount of collisions present for path evaluation, the path defined by the waypoints in Table 5.1 was proposed for the vessel as a starting point. The agent was guided along this path recording the feedbacks prior to training. Training was then carried out to optimize over the proposed path applying ϵ-decay consistent with that of the speed evaluation, seen in Equation 4.15. One disadvantage of the intended approach quickly became clear, due to independence of Q-values. Although some initial existed, the agent was oblivious to the initially proposed path as soon as new actions guided the vessel into unfamiliar states. To handle this the agent was instructed to proceed in a manner equivalent to the initial path upon exploration of unknown states. This is a further refinement of the ϵ strategy as discussed in Section 3.5.3. Revisiting previously seen states exploration carried on as regularly with potentially random choices based on ϵ. This approach of reduced exploration required only half the episodes as the previous tactic i.e 1000 episodes.

As can be interpreted from the progress shown in Figure 5.2.4 the agent was able to start converging to better paths earlier than training from scratch, after about 300 runs. However, collision percentage turned out as 27.4%, which is considered an insignificant improvement. This is most likely the case as even though the agent received some initial data and input on how to proceed, actions branching out into new paths can be counted as more valuable in terms of immediate rewards even if a collision is imminent. These actions will not be noted as bad until penalty backpropagates far enough as discussed in the previous case. Still the total number of collisions (274) is substantially lower, due to less episodes than for training from scratch. simultaneously, less episodes implies less computational demands. Just as for the previous case, n-step backpropagation of rewards could give great improvements.

Observing Figure 5.2.5 the process the agent was yet again able to reduce required thrust as seen in while fulfilling the docking goals. Concluding with the final path represented by the solid line in Figure 5.2.6, equal to that of the previous case. Here current optimal paths have been logged every 10th episode. Thus the agent is able to obtain the dame results for half the amount of episodes which is a substantial improvement. However, inspecting the figure it's clear how the traits of the initial paths are reflected in the agents proposed paths throughout training. Advantages of removing early randomness by guiding initial exploration presupposes that the global optimal path is of similar structure to the proposed one. Other scenarios or different initial paths might confuse the agent. Thus this altered approach might be too restrictive and be less robust. Still for the present case the reduced amount of episodes required is a valuable asset for computational purposes. Another proposition to overcome the independence of Q-values is to generalize over states by function approximation as discussed in Section 3.7.2. This would not only tie acquired knowledge of various states together, but reduce dimensionality of the state space to a set number of basis functions. Also by finer discretizations this could be increasingly valuable. In this event an initial proposed path could give the agent an idea of how to move in unseen states rather having to follow a forced exploration strategy. This could not only prove valuable for path evaluation, but the earlier cases as well. However, the limited precision and reduced state representation resulting from error in approximations would have to be gauged against the current approach.

Another important aspect to note is that no changes are made to the optimal strategy in the later stages, which also is the case for the previous approach. This indicates that the interdependence of available states and following importance of backpropagation might limit the effect of exploration for $\epsilon < 0.1$. This is on the basis as previously discussed, new actions would have to be explored multiple times before propagation would indicate them optimal. Consequently steeper decays or earlier termination can be imposed to limit computational demands.

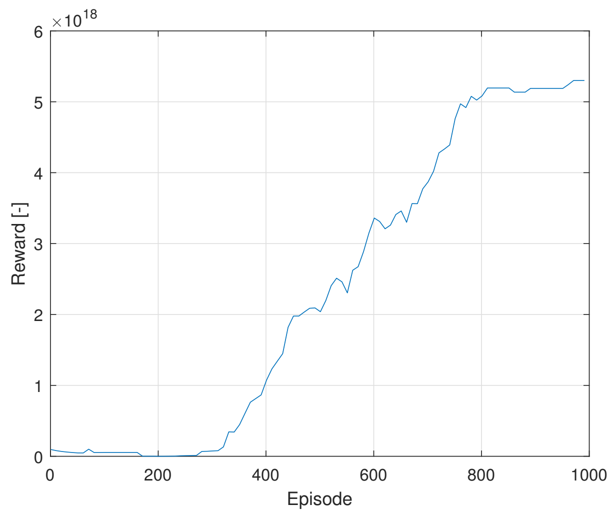


Figure 5.2.4: Progression of returned rewards for path improvement.

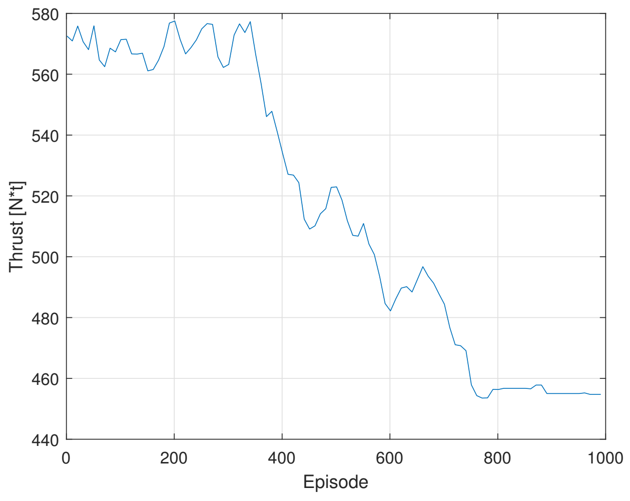


Figure 5.2.5: Progression of applied thrust for path improvement.

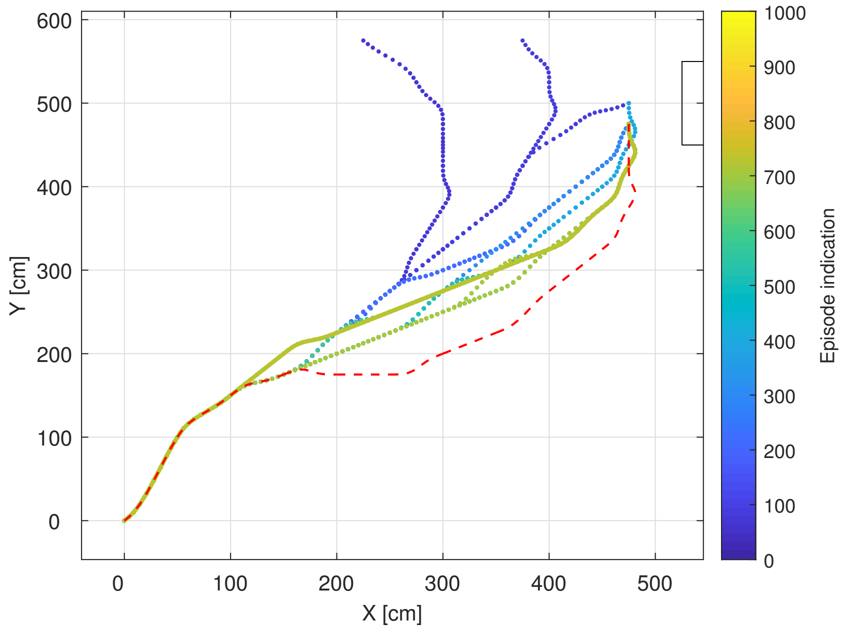


Figure 5.2.6: Evolution of optimal strategy for path improvement with color indication. Solid line showing the final result and red dashed line showing proposed path.

Chapter 6

Concluding remarks

6.1 Conclusion

This thesis has studied applications of machine learning in a marine setting, specifically reinforcement learning applied as an action-planning guidance layer for the vessel. A series of tasks have been tested and presented subject to performance and behavior. From the observed results, it is clear how the methods are able to propose improved strategies in various environmental settings, due to a model free design. These improvements are made through collected experience based on rewards of desired monitoring variables. The results reflect how the agents behavior, during and after training, is highly dependent on reward function design and action space definition. Further, it's clear how there exists a trade off regarding the extensiveness of the action space, between limiting computational demands and available solutions for the agent.

Training to assign speeds, the proposed methods were able to evaluate the vessels limits in regard to precision requirements, while maneuvering quickly to the quay and accurately come to rest. This showed how the reward function can be shaped to incorporate multiple areas of focus. Large penalty based rewards for precision and overshoot provided easy indication to avoid associated actions, while positive rewards of promptitude led to further optimization in the feasible range of state action pairs. Direct use of monitoring variables in rewards yielded strategies that were not always as anticipated, showing a known ability of reinforcement learning to discover new creative solutions. In specific, while training in regards to energy, the agent resolved a new strategy employing an unforeseen behavior of the thruster dynamics. However this also points out the importance of careful reward design as unforeseen behavior can lead to violations outside of considerations. Conflicting rewards proved to complicate learning by seemingly lowering stability due to the introduction of local optima. Still other reward components gave consistent improvement, suggesting how parts of the reward function can converge to optimal results despite others varying results.

Local optima also showed in training for good entrance paths as the many subtasks gave needs for a highly descriptive reward function. Here, as in the speed evaluation case, careful weighting of the various components had to be made to balance conflicting rewards. Still, good improvements were achieved, showing desired behavior of reaching the quay with correct headings under reduced requirements of thrust. Inspecting the final results it was clear how the path got shaped from the structure and magnitude of the reward components. Starting off the vessel headed directly towards the quay focusing on harvesting rewards of distance. Soon after however a rightward change in heading was made to facilitate for heading and docking rewards. This showed the effect of backpropagation in the agents strive for long term rewards, as the exponential structure of the docking reward and potential field weighting of the heading rewards gave the components dominance closer to the quay.

As path tracking was carried out in an underactuated manner with desired heading tangent to the path, future vessel configurations held dependency far back. This rendered backpropagation of rewards not only important but crucial for convergence to good strategies. This caused a need for increased exploration and thus increased risk. Here, implementing n-step updates is deemed likely to resolve much of this issue as well as give faster learning, and is thus proposed for future studies. Extending the approach to improve over a proposed path was found to reduce computational and exploratory demands, while converging to identical results. However, it's questionable if the new exploratory approach bound to the proposed path is less robust, and might weaken training progress in other circumstances.

While the methods returned promising results for all cases, some drawbacks should be taken into consideration. One is the lack of generalization across states, meaning all state-action pairs have to be tested individually leading to slow learning and need for extensive exploration. This could be improved through function approximation giving a reduction in dimensionality from N states to a fixed number of basis functions. However this would also introduce an approximation error resulting in limited precision. Another concern is bound to the inhabited randomness in exploration, creating uncertainty in convergence to global optimality. Improved results however are guaranteed. Despite these drawbacks, reinforcement learning has proved extensive problem-solving capabilities for marine applications, easily shaped to fit desired goals. All cases have returned improvements to more optimal strategies throughout training, leaving no doubt of great potential.

6.2 Further work

Based on the conclusion there are still some potential improvements present. Initially an n-step update rule should be implemented to gauge acceleration of training and improved stability of convergence. Additionally applying function approximation is likely to speed up training, and will be increasingly valuable if finer mesh was desired due to increased dimensionality. However as mentioned this update should be evaluated in terms of the introduced approximation error, and must therefore be compared against the current approach. Further, improvements of the proposed methods may be found through finer discretizations of state and action space. New formulations of these as well as new reward functions may also prove better suited. Lastly, different weighting of reward components, training parameters, and more intelligent exploration strategies might yield better convergence and results.

Extending to real world situations, running CSAD in a laboratory setting would be interesting. As the applied model is a high fidelity model quite similar to that of the actual vessel, the methods are expected to work here as well. Still, the introduction of noise as well as more complicated environmental effects could affect performance and give rise to new behavior. For actual training, starting from scratch in a real world situation may be unfeasible due to safety concerns, especially in regards of the time based speed training and path evaluation holding high risk of loosing path and collision respectively. A possible approach to overcome this is to provide optimal policies evaluated through simulations as a starting point, and let further training personalize these in a real world setting under low ϵ.

The area of focus for implementation of RL has been the guidance layer of the vessel. Deeper implementation into the vessels control systems would be an interesting task. Making the RL independent of the path generator through control of the vessels desired positions and speeds directly could give more freedom to the vessel. However this would likely increase the complexity and require more extensive training along with more well formulated states and actions.

Bibliography

Bellman, R. (1957*a*), *Dynamic programming*, sixth edn, Princeton University Press, Princeton, New Jersey.

Bellman, R. (1957*b*), 'A markovian decision process', *Indiana University Mathematics Journal* **6**, 15.

Bjørnø, J. (2016), Thruster-Assisted Position Mooring of C/S Inocean Cat 1 Drillship, Master thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Boyan, J. A. (2002), 'Technical Update : Least-Squares Temporal Difference Learning', (1988), 233–246.

Daniel, C., Viering, M., Metz, J., Kroemer, O. & Peters, J. (2014), Active Reward Learning, Technical report, Technische Universitat Darmstadt, Berkeley.

Fossen, T. I. (2016), *Handbook of Marine Craft Hydrodynamics and Motion Control [Bookshelf]*, Vol. 36, John Wiley & Sons, Ltd Registered, Trondheim, Norway.
URL: *http://ieeexplore.ieee.org/document/7393951/*

Frederich, P. (2016), Constrained Optimal Thrust Allocation for C / S Inocean Cat I Drillship, Master thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Gard AS (2014), 'Pilotage', *Selection of articles published by Gard AS* (July).

Huys, Q. J. M., Cruickshankc, A. & Serièsc, P. (2014), Reward-Based Learning, Model-Based and Model-Free Quentin, Technical Report 12.
URL: *http://www.scholarpedia.org/article/Encyclopedia_of_computational_neuroscience*

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), 'Reinforcement Learning: A Survey', *Journal of Artifocial Intelligence Research 4* pp. 238–285.

Kansal, S. & Martin, B. (2015), 'Reinforcement q-learning from scratch in python with openai gym'.
URL: *https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/*

Kobylinski, L. K. (2003), Directional stability of ships and safe handling G ,, Technical Report 18, Polish Academy of Sciences and Foundation for Safety of Navigation and Environment Protection, Poland.

Kunz, F. (2013), An Introduction to Temporal Difference Learning, *in* 'Seminar on Autonomous Learning Systems', pp. 1–8.

Lekkas, A. M. (2014), Guidance and Path-Planning Systems for Autonomous Vehicles, PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Ludvigsen, M. & Sørensen, A. J. (2016), 'Towards integrated autonomous underwater operations for ocean mapping and monitoring'.
URL: *https://www.sciencedirect.com/science/article/pii/S1367578816300256*

Marin-Plaza, P., Hussein, A., Martin, D. & De La Escalera, A. (2018), 'Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles', *Journal of Advanced Transportation* **2018**.

Marsland, S. (2015), *Machine learning: An Algorithmic Perspective*, second edi edn, CRC Press.

Matiisen, T. (2015), 'Demystifying deep reinforcement learning'.
URL: *https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/*

Murdoch, E., Clarke, C., Dand, I. W., Glover, B., Busoniu, L. & Yang, S. X. (2014), 'Master's guide to berthing - the standard club'.

Nguyen, Q. P., Low, K. H. & Jaillet, P. (2015), 'Inverse Reinforcement Learning with Locally Consistent Reward Functions'.

Nowe, A., Vrancx, P. & De Hauwere, Y.-M. (2012), *Game Theory and Multi-agent Reinforcement Learning*, p. 30.

Rioual, Y., Moullec, Y. L., Laurent, J., Khan, M. I. & Diguet, J.-p. (2018), 'Reward Function Evaluation in a Reinforcement Learning Approach for Energy Management', (August).

Rummery, G. A. & Niranjan, M. (1994), On-line Q-Learning usinf connectionist systems, Technical report, Cambridge University Engineering Department, Cambridge.

Rødseth, J. & Tjora, (2014), 'A system architecture for an unmanned ship'.

Sifakis, J. (2018), 'Autonomous systems - an architectural characterization', *CoRR* **1**.
URL: *http://arxiv.org/abs/1811.10277*

Skjetne, R. (2005), Roger Skjetne The Maneuvering Problem, PhD thesis, Norwegian university of Science and Technology.

Skjetne, R. (2019*a*), Marine control systems ii lecture 7: Maneuvering control design.

Skjetne, R. (2019*b*), 'Notes on : Maneuvering control design of a low-speed fully-actuated vessel with stepwise path generation'.

Sutton, Richard S. and Barto, A. G. (2010), *Reinforcement Learning: An Introduction*, number 10884.

Tanakitkorn, K. (2018), 'A Review on Unmanned Surface Vehicles Development', *Maritime Technology and Research* **1**(1), Proof.

Theodoridis, S. & Koutroumbas, K. (2008), *Pattern Recognition, Fourth Edition*, 4th edn, Academic Press, Inc., Orlando, FL, USA.

van Otterlo, M. & Wiering, M. (2013), *Reinforcement Learning State-of-the-Art*, twelth edi edn, Spinger.

Wang, H. (2019), 'Exploration versus exploitation in reinforcement learning : a stochastic control approach ', (March 2018), 1–38.

Watkins, C. J. & Dayan, P. (1992), 'Technical note: Q-learning', *Machine Learning* **8**(3), 279–292.
URL: *https://doi.org/10.1023/A:1022676722315*

Xu, X., He, H., Zhao, D., Sun, S., Busoniu, L. & Yang, S. X. (2015), 'Machine learning with applications to autonomous systems'.

Zhang, C., Liu, Z., Gu, B., Yamori, K. & Tanaka, Y. (2018), 'for Cost- and Energy-Aware Multi-Flow Mobile Data Offloading', pp. 1–8.

Appendices

Appendix A

CSAD Model specifications

A.1 3 DOF Maneuvering Model

Table A.1: CSAD: Rigid body and added mass parameters.

Rigid body

Parameter	Value
m	127.92
I_z	61.967
x_g	0

Added mass

Parameter	Value
$X_{\dot{u}}$	3.262
$Y_{\dot{v}}$	28.89
$Y_{\dot{r}}$	0.525
$N_{\dot{v}}$	0.157
$N_{\dot{r}}$	13.98

Table A.2: CSAD: Drag coefficients in surge, sway and yaw.

Surge

Parameter	Value
X_u	-2.332
$X_{\|u\|u}$	0
X_{uuu}	-8.557

Sway

Parameter	Value
Y_v	-4.673
$Y_{\|v\|v}$	0.3976
Y_{vvv}	313.3

Yaw

Parameter	Value
N_r	-0.01675
$N_{\|r\|r}$	-0.01148
N_{rrr}	0.000357

A.2 Thruster parameters

The following presents the thruster parameters of CSAD as described in Frederich (2016). This should be considered with care as some parameters might be flawed. This is especially directed at the full model parameters seemingly being of wrong dimension.

Table A.3: CSAD: Thruster parameters after scaling.

	Scaled Model	**Full Model**	**Unit**
Max shaft speed	94.87	152	[RPS]
Power	0.8676	-	[W]
Propeller Diameter	0.0467	0.030	[m]
Steering speed	113.84	270	[Deg/s]
Max Torque	0.0015	-	[Nm]
Max Thrust	1.5034	2.6002	[N]

Appendix B

Additional results

B.1 Speed evaluation results

This appendix presents additional and supporting results. It should be noted that negative values of the overshoot reward is not due to insufficient precision, but rather due to simulations being terminated before the agent is able to reach the quayside. Yet the values result due to the linear decrease in speed at the end, and are small enough to be counted as sufficient precision.

B.1.1 Time reduction with precision focus

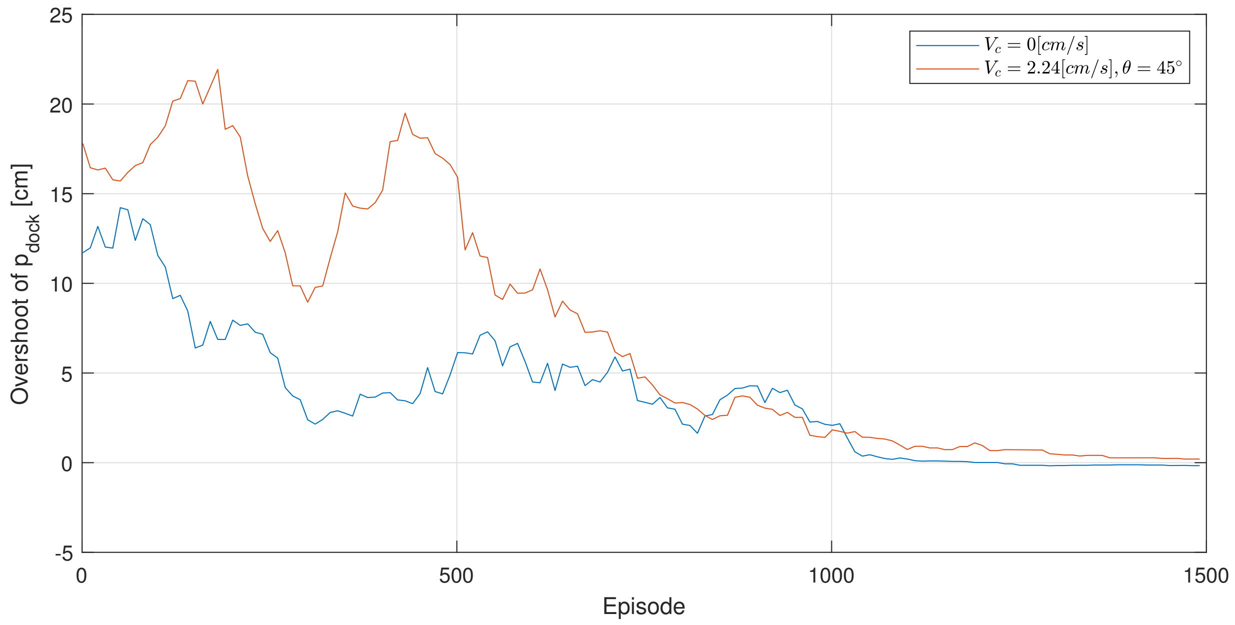


Figure B.1.1: Progression of overshoot of p_{dock}. Focusing on time and precision.

B.1.2 Energy efficiency

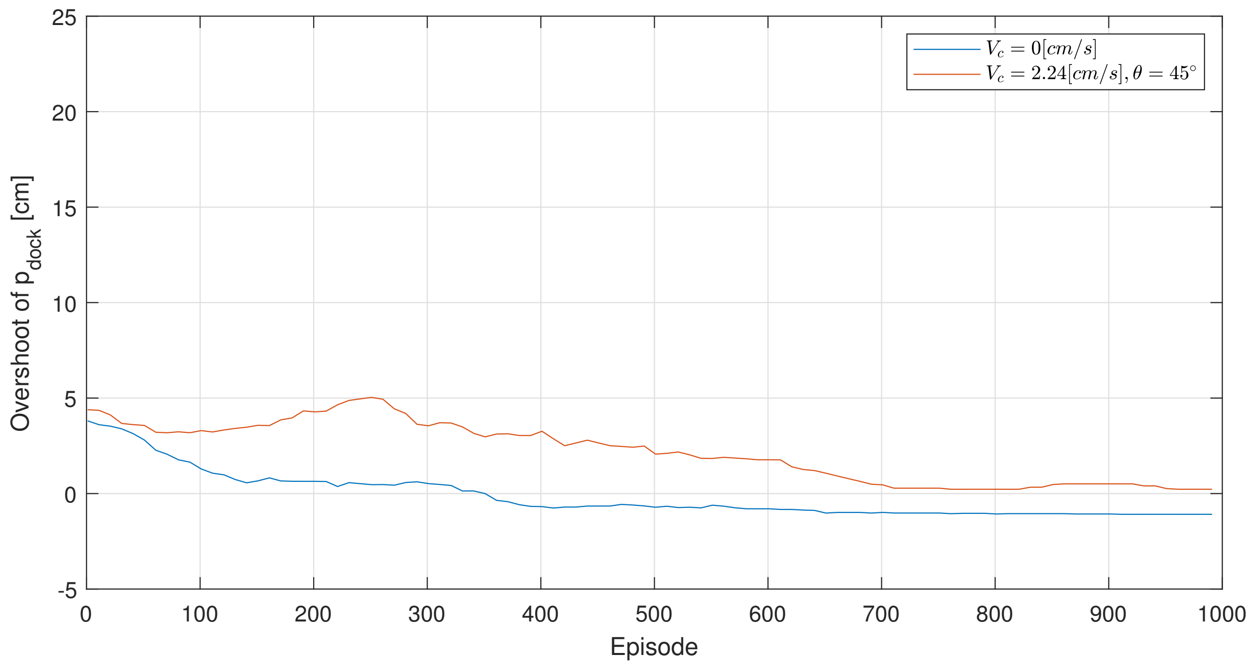


Figure B.1.2: Progression of overshoot of p_{dock}. Focusing on thrust.

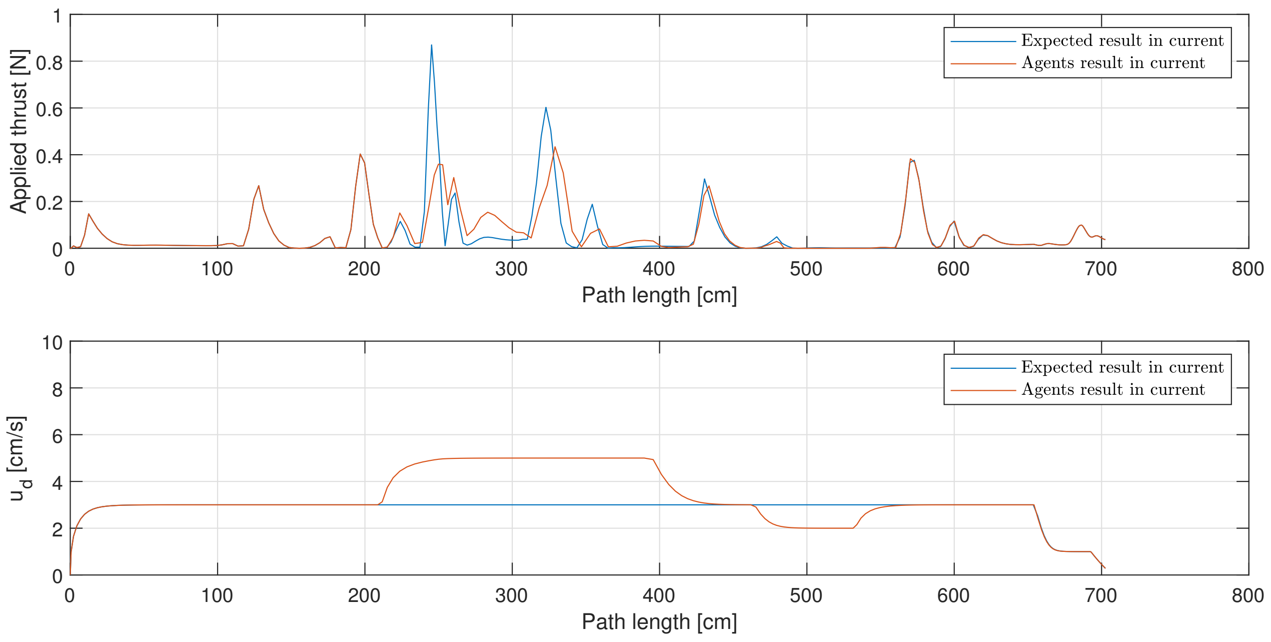


Figure B.1.3: Applied thrust along path length for the expected and final result in terms of desired speed, post training. Focusing on thrust.

B.1.3 Reduction of time and thrust

Assigned speeds

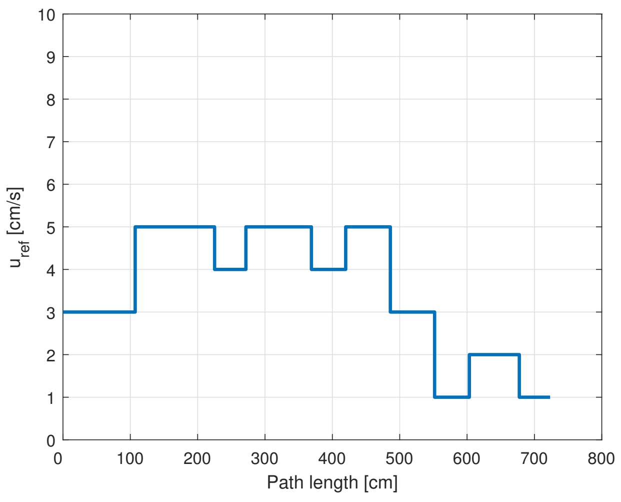


Figure B.1.4: Reference speeds along path length post training. Focusing on time and thrust.

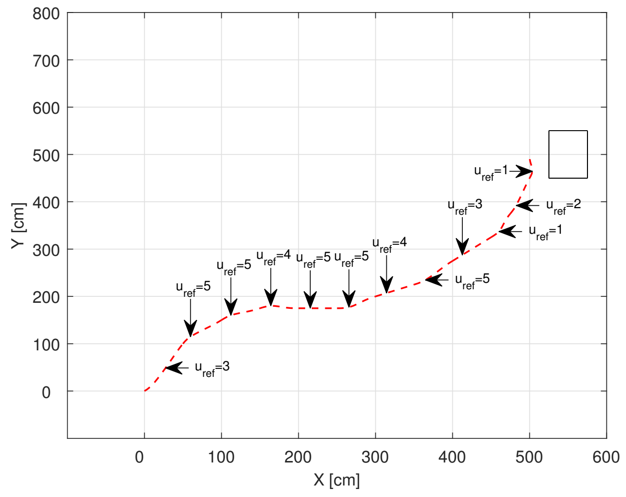


Figure B.1.5: Reference speeds along path post training. Focusing on time and thrust.

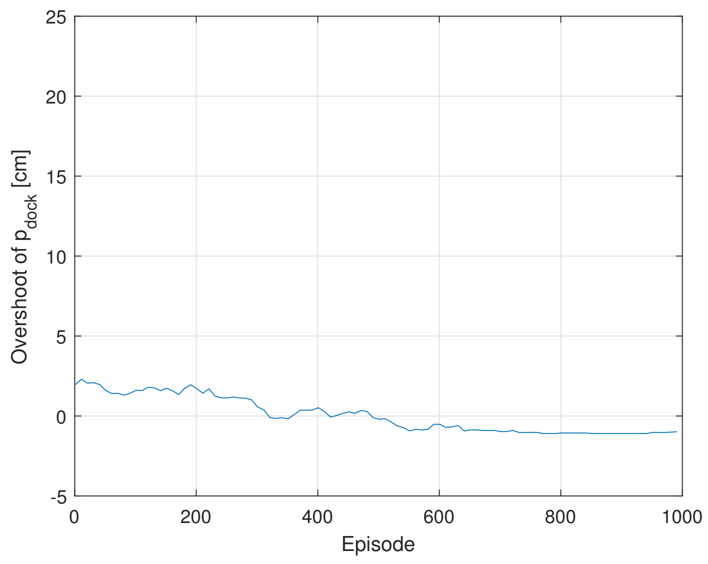


Figure B.1.6: Progression of overshoot of p_{dock}. Focusing on time and thrust.

Agent converging to focusing solely on thrust

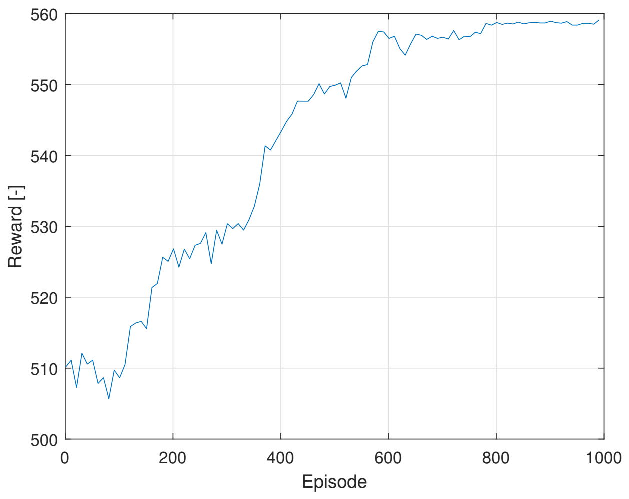


Figure B.1.7: Progression of returned rewards, converging to thrust only. Focusing on time and thrust.

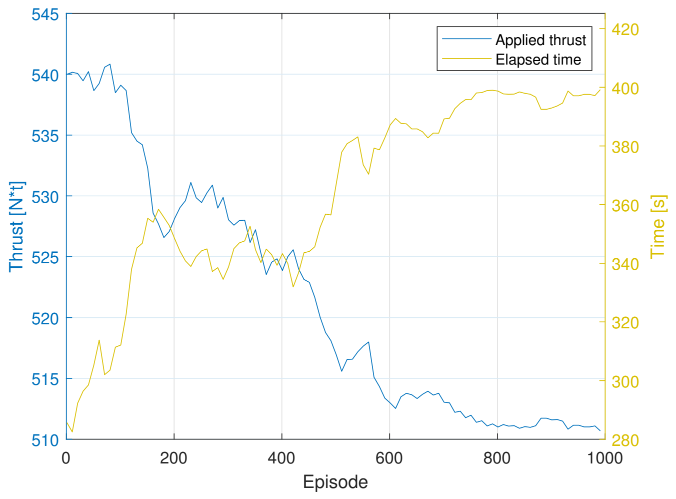


Figure B.1.8: Progression of time and thrust, converging to thrust only. Focusing on time and thrust.

B.2 Path evaluation results

Agent stuck at local optima of thrust

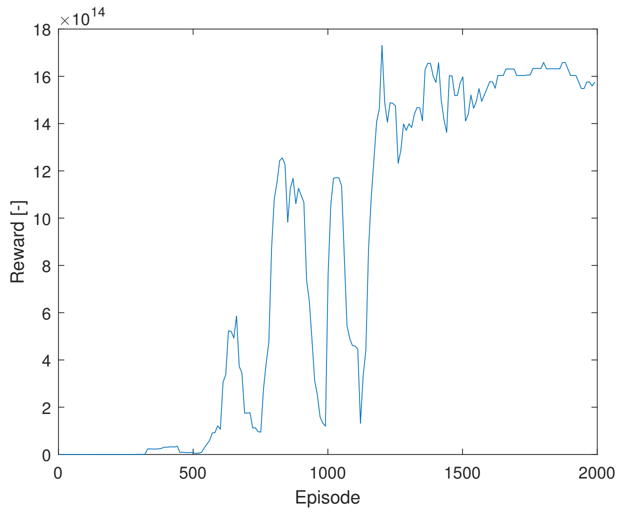


Figure B.2.1: Progression of reward for path evaluation. Agent stuck at local optimum.

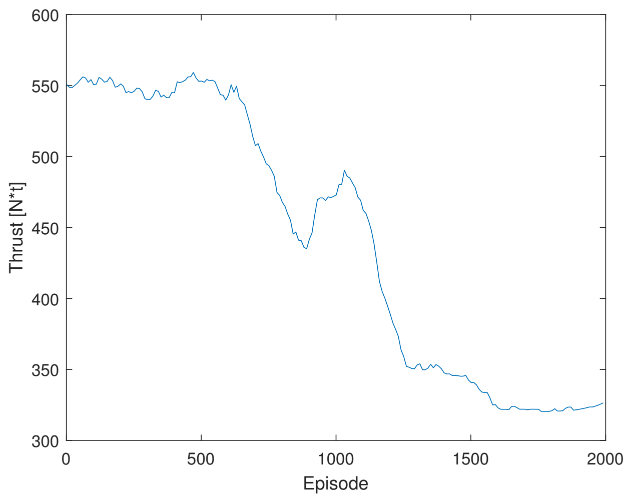


Figure B.2.2: Progression of thrust for path evaluation. Agent stuck at local optimum.

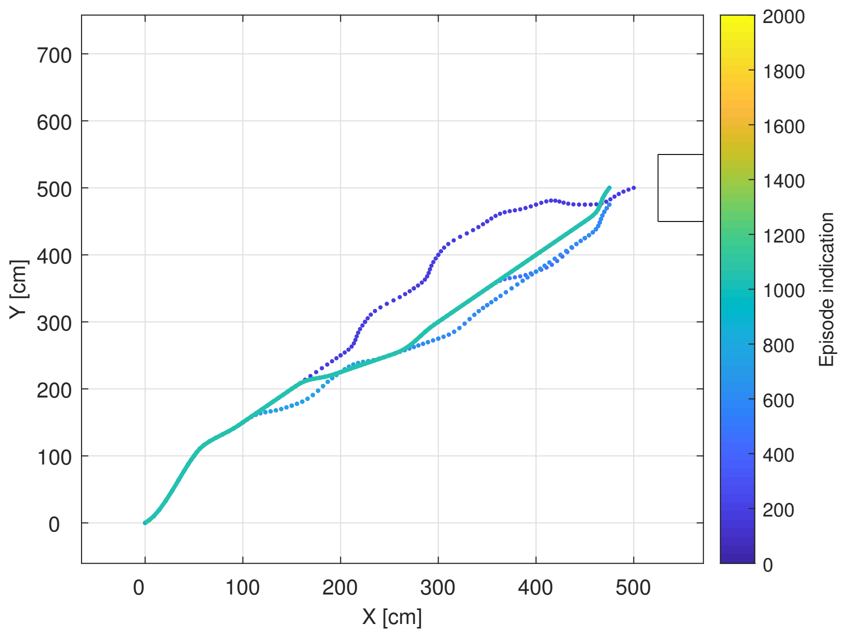


Figure B.2.3: Evolution of optimal strategy for path evaluation with color indication. Solid line showing the final result. Agent stuck at local optimum.

Brage Elias West Mothes

Reinforcement learning for autodocking of surface vessels

NTNU
Norwegian University of
Science and Technology