Master's thesis

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

Vegard Sanden

High-level Action Planning for Marine Vessels Using Active Inference and Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2022

NTNU
Norwegian University of
Science and Technology

Vegard Sanden

High-level Action Planning for Marine Vessels Using Active Inference and Reinforcement Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2022

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

NTNU
Norwegian University of
Science and Technology

Abstract

In recent years there has been an increase in research, development and use of autonomous marine vessels. A tool that has been used for increasing autonomy in marine vessels is reinforcement learning. Reinforcement learning is very useful for automated decision-making and it has been showed that it can be useful for path planning, obstacle avoidance and autonomous docking of marine vessels. However, even if the technique can be very useful for solving decision-making problems, there are some challenges with the method. For example, there are challenges regarding sample efficiency, constructing the reward function and with using black-box models, such as neural networks, in combination with reinforcement learning. Therefore, it might be useful to look to other fields of artificial intelligence for solving decision-making problems. In this thesis we look at *active inference* which has some characteristics similar to reinforcement learning. This is a method for decision-making, perception and learning in uncertain environments. Active inference is built upon the free-energy principle which is a principle for describing how autonomous systems remains inside a specific set of states and resists a tendency to disorder. When the system minimizes its variational free energy, the behaviour can be optimized. Active inference minimizes the variational free energy and a quantity called expected free energy by exploring the environment and forming habits to find the optimal decisions.

This thesis explores the use of active inference for high-level action planning and how it works in combination with guidance and control of marine vessels. More specifically we try to solve the task with *deep active inference* which is active inference in combination with deep neural networks. In addition, we perform the same task with deep Q-networks for comparison. As a basis for making comparisons of the performance between deep active inference and deep Q-networks, we implemented a problem of guiding a marine vessel to its docking position through a port in a discrete environment. For simplicity the docking problem is solved in two parts. One part where the agent is moved from outside the port area to inside the port close to the docking position. The other part concerns guiding the agent into a specific docking position where the heading is taken into account.

In the first part of the thesis we present theory about artificial neural networks, reinforcement learning, active inference and guidance and control of marine vessels.

Next, details about the implementation of the deep active inference agent and the deep Q-learning agent are presented. In addition, we present the environment these algorithms are trained on and details about the marine vessel model and its guidance and control system is presented. Then, the results obtained with the algorithms are presented and discussed.

The results show that the implementation of deep active inference used in this thesis does only partly solve the docking problem. The algorithm manages to find a path that moves the agent into a docking position with a specific heading with performance comparable to the deep Q-network. However, it does not solve the task of moving the agent from outside the port to the docking area. Exactly why the algorithm does not solve the task is not clear, but from our evaluation it seems like the algorithm struggles with handling a combination of different characteristics in an environment. A possible reason for this is that the estimations of the expected free energy are too inaccurate to do a correct propagation of the expected free energy through the tight port area. Even though the deep active inference algorithm implemented in this thesis could not solve the docking problem, it does not mean that deep active inference and active inference cannot be used to solve this problem. In general deep active inference has a few advantages compared to reinforcement learning in the way the desired states can be represented and that it on its own can find a balance between exploration and exploitation. Also, it has been seen that deep active inference has produced promising results compared to reinforcement learning in other papers. There exists several ways of performing deep active inference and as we only tested one of them in this thesis, we might get better results from the other methods. In addition, as it is a relatively new framework it needs to be assessed further to get a good overview of its properties.

Sammendrag

I de senere år har det vært en økning i undersøkelser rundt, utvikling og bruk av autonome marine fartøy. Et felt innen kunstig intelligens som har blitt brukt for å øke autonomiteten for marine fartøy er forsterkende læring (eng. reinforcement learning). Forsterkende læring er nyttig for automatisk beslutningstaking og det har blitt vist at det kan være nyttig for stifølging (eng. path following), objektomgåelse (eng. obstacle avoidance) og automatisk dokking. Selv om dette feltet kan være nyttig for å løse beslutningstakingproblem (eng. decision-making problem) er det likevel noen utfordringer med å bruke denne metoden. For eksempel er det utfordringer med samplingeffektiviteten, å konstruere belønningsfunksjonen (eng.reward function) og å bruke black-box modeller, som nevrale nettverk, i kombinasjon med forsterkende læring. Derfor kan det være nyttig å utforske andre felt innen kunstig intelligens for å løse beslutningstakingproblem. I denne masteroppgaven skal vi utforske *active inference* som har noen like karakteristikker som forsterkende læring. Dette er en metode for beslutningstaking, oppfatning og læring i usikre miljø. Active inference er bygget på the free energy principle som er et prinsipp for å beskrive hvordan autonome systemer forblir inne i et spesifikt sett av states og motstår tendensen til desorganisering. Når systemet minimerer sin variable frie energi (eng. variational free energy) kan oppførselen til systemet optimeres. Active inference minimerer den variable frie energien og den forventede frie energien (eng. expected free energy) ved å undersøke miljøet og forme vaner for å finne optimale beslutninger.

Denne masteroppgaven undersøker bruken av active inference for høynivå handlingsplanlegging og hvordan det fungerer i kombinasjon med veiledning og kontroll (eng. guidance and control) hos marine fartøy. Vi forsøker å løse denne oppgaven ved *deep active inference* som er active inference i kombinasjon med dype nevrale nettverk. I tillegg, forsøker vi å løse samme oppgave med dyp Q-læring (eng. deep Q-learning) for å sammenligne med active inference. For å lage et grunnlag for å gjøre sammenligningene, implementerer vi en oppgave som omhandler å veilede et marint fartøy gjennom en havn og legge til kai (eng. docking position) i et diskret miljø. For enkelhetsskyld løses dokkingen i to deler. En del der agenten ledes fra utsiden av havnen til innsiden av havnen i nærheten av der den skal legges til kai. Den andre delen omhandler at agenten veiledes til en spesifikk posisjon langs kaien der retningen agenten peker i er tatt i betraktning.

I den første delen av masteroppgaven presenterer vi teori om kunstige nevrale nettverk, forsterkende læring, active inference og veiledning og kontroll av marine fartøy. Videre presenterer vi detaljer om implementasjonen av den dype active inference-agenten og den dype Q-læringsagenten. I tillegg beskriver vi miljøet disse algoritmene er trent på og detaljer om modellen av det marine fartøyet og hvordan veiledning og kontrollen av fartøyet blir gjort. Så blir resultatene vi kommer frem til presentert og diskutert.

Resultatene viser at den implementasjonen av dyp active inference som vi bruker i denne oppgaven løser kun deler av dokkingproblemet. Algoritmen klarer å finne en vei som leder agenten til kai med en spesifikk retning den peker i med ytelse sammenlignbart med dypt Q-nettverk (eng. deep Q-network). Men, den klarer ikke å løse oppgaven som omhandler å lede agenten fra utsiden av havnen til området den skal legge i kai. Nøyaktig hvorfor algoritmen ikke klarer å løse oppgaven er ikke klart for oss, men fra vår evaluering virker det som at algoritmen har problemer med å håndtere en kombinasjon av ulike karakteristikker på miljøet. En mulig grunn for dette er at estimeringen av den forventede frie energien er for unøyaktig til å propagere den forventede frie energien gjennom den trange havnen på riktig måte. Selv om den dype active inference-algoritmen implementert i denne masteroppgaven ikke klarte å løse dokkingproblemet, så betyr ikke det at dyp active inference og active inference ikke kan brukes til å løse denne oppgaven. Generelt så har active inference noen fordeler sammenlignet med forsterkende læring som måten active inference kan representere ønskede tilstander på og at det kan finne en balanse mellom utforskning og utnyttelse (eng. exploration and exploitation). I tillegg har vi sett at dyp active inference har produsert lovende resultater sammenlignet med forsterkende læring i andre rapporter. Det eksisterer flere andre måter å utføre dyp active inference på og ettersom vi testet kun en metode i denne masteroppgaven kan det hende at vi får bedre resultater med de andre metodene. Dessuten er dette et relativt nytt rammeverk som trengs å utforskes videre for å få en god oversikt over dets karakteristikker og kvaliteter.

Preface

This thesis was written in the spring of 2022 at the Norwegian University of Science and technology and represents the end of my Masters of Science in Cybernetics and robotics. It was carried out with supervision from Anastasios Lekkas. The thesis aims to explore active inference and examine if it can be used as a high-level action planner in combination with guidance and control of marine vessels. To examine the performance of active inference, deep reinforcement learning is used for comparison. This thesis is a continuation of my project thesis from last semester where I begun to explore active inference and investigated how it compares to reinforcement learning as a high-level action planner on discrete environments.

The deep Q-learning algorithm implemented in this thesis is the same as used in the project thesis last semester. In addition, the deep active inference algorithm is based on the work of Millidge [31] and van der Himst et al. [25] where the GitHub repository of van der Himst et al [24] was used as a base for the implementation.

The algorithms and the models implemented in this thesis are implemented with the Python programming language run on Google Colab. Google Colab is an online platform for running Jupyter Notebooks on the cloud servers of Google. To perform numerical operations the numerical library NumPy was used and for for making visualization of results Matplotlib was used. In addition, to construct the discrete environments used when running the algorithms, the OpenAI Gym library was utilized. Also, the machine learning library Pytorch was employed to create deep neural networks when implementing the deep Q-network and deep active inference. To run the simulations of the algorithms and models a Dell workstation provided by NTNU and a Asus ZenBook 14 were used. For writing this thesis the typesetting system LaTeX is used.

I wish to thank my supervisor Anastasios Lekkas for his guidance when performing the task and writing this thesis. His guidance and insights for development and execution of the task has been very important and valuable for the progress of the thesis.

Contents

List of Figures

List of Tables

Acronyms

ANN artificial neural network. 13–15, 23, 35

DDPG deep deterministic policy gradient. 23, 54

DP Dynamic positioning. 13, 40, 45, 46

DQN deep Q-network. ix, xi, 21, 23, 39, 40, 44–48, 50–52, 54–58, 60, 62, 69, 73, 74, 79, 80, 83, 84

LOS Line-of-sight. 9, 40, 44, 45, 59

MDP Markov decision process. 16, 17, 83

MIMO multiple-input multiple-output. 12, 13, 46

MSE mean squared error. 50–52, 75, 79

POMDP partially observable Markov decision process. 16, 21

SISO single-input single-output. 11, 12

TD temporal-difference. 18

VAE variational-autoencoder. 35, 51

Chapter 1

Introduction

This thesis examines the use of deep active inference and its potential to be used for autonomous docking of marine vessels. In this chapter the background and motivation of this paper is presented, as well as the main contributions of the thesis. Also, the outline of the thesis is given.

1.1 Background and motivation

As the technology for autonomous systems is improving there can be seen a increase in research and use of these systems. This can for example be seen in the transport industry where companies such as Google, Tesla, Uber and Mercedes Benz are researching self-driving cars [3] and with Ruter trialing self-driving vehicles as a part of public transport services [43]. Also, in air-based transport there is a development of unmanned aerial vehicles (UAV) for delivery of products [14]. There has also been done a lot of research and development in the past years on the topic autonomous marine vessels. For instance, in 2018 Rolls Royce and Finferries demonstrated that the ferry called Falco managed to perform navigation autonomously [39]. Also, autonomous ship projects such as MUNIN [11], YARA Birkeland [45] and DNV GLs ReVolt [48] are examples of projects that have been allocated for developing autonomous technology in the ship industry [34].

An example of a tool that has been explored for increasing the autonomy in marine vessels is deep reinforcement learning. After deep reinforcement learning was introduced it has been shown theoretically that it can be a useful tool for increasing autonomy for navigation and control of marine vessels. For instance, it has been shown that it can be used for path planning [23], obstacle avoidance [6, 30] and autonomous docking [40] of a marine vessel. Also, reinforcement learning has been used for in docking of underwater vehicles [2].

Deep reinforcement learning is a method in artificial intelligence where reinforcement learning and deep learning are combined. In reinforcement learning agents try to learn the actions that maximizes the cumulative reward. The agent exists in an environment which consists of a set of states and when the state space becomes too large it is useful to combine reinforcement learning with deep neural networks. Even though reinforcement learning can be very useful there are some challenges. A major challenge in reinforcement learning is sample efficiency. This means that a lot of samples is needed to reach efficient performance [9]. It can also be difficult to find a reward function that guides the agent to an optimal policy. In addition, exploration of the environment can be a challenge. There are also challenges by using deep neural networks in reinforcement learning. Deep neural networks are essentially black boxes because we cannot get any intuition of how it reaches a specific output from an input. Also, deep reinforcement learning can be unstable as unstable properties of deep neural networks is made worse with use of reinforcement learning [9]. This makes it hard to reproduce reinforcement learning models.

As there are challenges with reinforcement learning and deep learning it may be of significance to explore other methods in artificial intelligence that may have the potential to increase autonomy in robotic applications, specifically autonomous marine vessels in this thesis. In this thesis the potential of using *active inference* as a high-level action planner with low level and control of marine vessels is examined. Active inference uses the free-energy principle for solving and understanding how autonomous agents performs decision-making and learns in uncertain environments. The free-energy principle was introduced by Karl Friston and has been presented in several papers [15, 16, 19]. The principle says that to maintain the states that defines the agent, it has to minimize its variational free energy [16]. When the agent minimizes its variational free energy it also minimizes its own surprise, which is something that the agent wants to avoid. The surprise is the negative log-probability of an outcome which makes states that the agent has a high probability of being located in least surprising. This means that minimizing the variational free energy ensures that the agent maintains its non-surprising states such that the agent refuses its tendency to disorder [16].

There are several examples of research on active inference for optimizing decision-making, learning and planning in uncertain environments [18, 20, 21]. Also, there are several examples of implementing active inference in tasks with continuous or state spaces where the performance has been compared to reinforcement learning [22, 38, 44]. As active inference only can be used on small state spaces there has recently been made active inference models in combination with deep neural networks [13, 25, 31, 49]. This is called *deep active inference* and the deep neural networks are used to store densities and the expected free energy. Active inference and deep active inference have mostly been used on simulated models but there are also examples of active inference and deep active inference that have been used used for robot control [5, 10]. Another solution to the challenge of large

state spaces is sophisticated active inference which calculates expected free energy recursively [17].

In this thesis we will explore the task of high-level action planning using active inference and reinforcement learning and how it performs with guidance and control of marine vessels. This is a continuation of a project thesis written in the autumn of 2021 where we examined if active inference could be used as a high-level action planner in discrete state and action spaces. The main goal of this thesis is to further explore what we found in the project thesis and try to examine if active inference can be a useful tool for increasing autonomy in the marine industry. As will be seen in this thesis, reinforcement learning and active inference have several similarities and therefore is reinforcement learning used for comparison. Also, as it already has been seen that reinforcement learning has been used with success for autonomous marine vessels it can provide good comparisons.

1.2 Contributions

The main contributions of this thesis are:

- The construction of the problem set up by using deep active inference for high-level action planning with low-level control of marine vessels.
- A comparison between our implementation of deep active inference and deep reinforcement learning on the docking problem.
- A discussion of the deep active inference algorithm and why it performed as it did.

In this thesis we propose a new area of usage for active inference. Also, we propose an alternative to today's methods for docking of marine vessels autonomously. We construct a problem formulation where active inference is used in the form of deep active inference for high-level action planning with low-level control of marine vessel. To be able to assess how good active inference is performing we use reinforcement learning as a comparison as it has been found to be successful in solving the docking problem. By both testing the deep active inference on a discrete environment as a high-level action planner and then on a continuous environment where it can control the marine vessel directly we can get a good indication on if active inference can add value to autonomous docking.

By performing deep active inference and deep Q-learning on the discrete docking environment used in this thesis we compare performance and characteristics of the two methods. Also, we present and discuss our findings of why the deep active inference algorithm performs as it does.

1.3 Outline

The thesis is divided into seven main chapters where we start in the current chapter. In Chapter 2 we present background theory about topics used in the thesis. Here, theory about guidance and control of marine vessels, artificial neural networks and reinforcement learning is introduced. As it is not expected that the audience has prior knowledge about active inference, theory about this topic is presented and explained in its own chapter, in Chapter 3.

After presenting the relevant theory, a thorough description about the implementation of the algorithms, the environment and the marine vessel model is given in Chapter 5. Subsequently, we present the results obtained with the deep Q-network and deep active inference in Chapter 5. The results are discussed and compared in Chapter 6, in addition to an assessment of further work. Furthermore, the thesis is concluded in Chapter 7.

Chapter 2

Background and theory

In this chapter, background theory about a few theoretical aspects used in this thesis is presented. This chapter gives an introduction to the topics:

- Guidance and control of marine vessels
- Artificial neural networks
- Reinforcement learning

2.1 Guidance and control of marine vessels

2.1.1 Kinematics of the marine vessel

Before we get into the guidance and control of the marine vessel, we need to look at the kinematics of the marine vessel. The kinematics of the marine vessel used in this thesis is based on the kinematics presented in [29].

We assume that the marine vessel operates in an area close to land where its task is to move from an open area through a port to dock in a specified position. As this is a tight area with obstacles we assume that the marine vessel moves with low velocities. To simplify the model of the vessel we also assume that the vessel has 3 degrees of freedom, which means that we look away from the effects of roll and pitch motions. Thus, only the surge, sway and yaw is taken into account which means that we only look at the vessel in the planar position.

To represent the motion of the vessel we use the pose vector $\boldsymbol{\eta} = [x, y, \psi]^\top \in \mathbb{R}^2 \times \mathbb{S}$ and the velocity vector $\boldsymbol{\nu} = [u, v, r]^\top \in \mathbb{R}^3$. The pair (x, y) describes the Cartesian position of the center of the marine vessel in the North-East-Down frame. This is

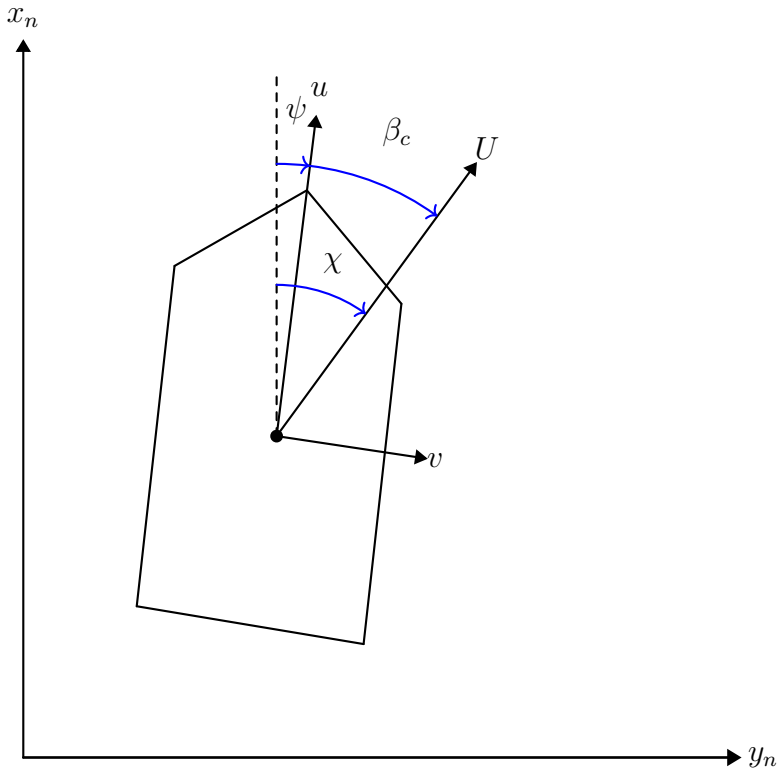


Figure 2.1: Marine vessel in a North-East-Down reference frame.

an earth-fixed reference frame and the positions x and y moves in relation to this frame. Furthermore, ψ is the yaw angle of the vessel, (u, v) is the body fixed linear velocities and r is the yaw rate. A figure of the marine vessel and the elements of the pose and velocity vector can bee seen in Figure 2.1. The vessel model can be described with the following differential equations

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\psi)\boldsymbol{\nu} \tag{2.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{2.2}$$

where $\boldsymbol{J} \in SO(3)$, $\boldsymbol{M} \in \mathbb{R}^{3\times3}$, $\boldsymbol{D}(\boldsymbol{\nu}) \in \mathbb{R}^{3\times3}$ and $\boldsymbol{\tau}$ are the rotation matrix, the intertia matrix, the dampening matrix and the force vector respectively [29]. The rotation matrix $\boldsymbol{J}$ is used for rotation from the body frame to the reference frame which is earth-fixed. This matrix is given by

$$\boldsymbol{J}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.2: Thruster configuration for the marine vessel. Illustration from [29]

2.1.2 Thrust configuration

The movement of the vessel is controlled by three thrusters. The configuration of the thrusters can be seen in Figure 2.2. Two of the thrusters are azimuth thrusters located at the back of the vessel. The azimuth thrusters can be rotated in the horizontal plane such that there is no need for a rudder [12]. Towards the front of the vessel the third thruster is located as a tunnel thruster. A tunnel thruster is a transverse tunnel which produces a force in the y-direction [12]. The force vector $\boldsymbol{\tau}$ are given by

$$\boldsymbol{\tau} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{f} \tag{2.3}$$

where $\boldsymbol{T}(\boldsymbol{\alpha}) \in \mathbb{R}^{3,\mathrm{n_{thrusters}}}$ is the thrust configuration matrix and $\boldsymbol{f}$ is the forces generated by each of the three thrusters. The thrust configuration matrix maps the force from the three thrusters into surge, sway and yaw forces and moments [12]. By considering that the vessel has two azimuth thrusters in the aft of the vessel and a tunnel thruster in the front, the thruster configuration matrix is given by The angles α_1 and α_2 are the rotation angles of the two azimuth thrusters

$$\boldsymbol{T}(\boldsymbol{\alpha}) = \begin{bmatrix} \cos(\alpha_1) & \cos(\alpha_2) & 0 \\ \sin(\alpha_1) & \sin(\alpha_2) & 1 \\ l_{x_1}\sin(\alpha_1) - l_{y_1}\cos(\alpha_1) & l_{x_2}\sin(\alpha_2) - l_{y_2}\cos(\alpha_2) & l_{x_3} \end{bmatrix}$$

in the body frame. As the angle α_3 corresponding to the tunnel thruster always will be 90 degrees in the body frame, the third column can be written out. The expressions l_{x_i} and l_{y_i} are the positions of the thrusters in relation to the centre of the vessel.

To generate the desired force $\boldsymbol{\tau}$, the rotation α of the azimuth thrusters and the forces $\boldsymbol{f}$ of the three thrusters have to be found. This is called the thrust allocation problem and can be solved in several ways [29]. The forces f_i generated by the thrusters can be represented by

$$f_i = K_i u_i \tag{2.4}$$

where K_i are gains and u_i are control inputs. By inserting this into the expression for the force vector (2.3) we get that

$$\boldsymbol{\tau} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{K}\boldsymbol{u} \tag{2.5}$$

where we define the input matrix $\boldsymbol{B}$ as the multiplication of the thrust configuration matrix $\boldsymbol{T}$ and the gain matrix $\boldsymbol{K}$ [12]

$$\boldsymbol{B} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{K} \tag{2.6}$$

which gives us the force vector

$$\boldsymbol{\tau} = \boldsymbol{B}\boldsymbol{u} \tag{2.7}$$

The control inputs u_i can be represented as $u_i = |n_i|n_i$ where n_i is the propeller revolution per second of thruster i [12].

As the azimuth thrusters can be rotated horizontally by an angle α, they can produce forces in both the x- and y-direction. Thus it can be useful to decompose the forces in two, f_{i_x} and f_{i_y}. As the force from one azimuth thruster can be decomposed in two components, we can extend the control input vector $\boldsymbol{u}$ from $\boldsymbol{u} = [u_1, u_2, u_3]^\top$ to the extended control input vector $\boldsymbol{u}_e = [u_{1_x}, u_{1_y}, u_{2_x}, u_{2_y}, u_3]$ [12]. Then, the generalized force vector τ is expressed by

$$\boldsymbol{\tau} = \boldsymbol{B}_e\boldsymbol{u}_e \tag{2.8}$$

where

$$\boldsymbol{B}_e = \boldsymbol{T}_e\boldsymbol{K}_e \tag{2.9}$$

The extended thrust configuration matrix $\boldsymbol{T}_e$ is for the vessel specified in this task given by

$$\boldsymbol{T}_e = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ -l_{y_1} & l_{x_1} & -l_{y_2} & l_{x_2} & l_{x_3} \end{bmatrix}$$

and the extended thrust coefficient matrix $\boldsymbol{K}_e$ is given by

$$\boldsymbol{K}_e = \begin{bmatrix} K_1 & 0 & 0 & 0 & 0 \\ 0 & K_2 & 0 & 0 & 0 \\ 0 & 0 & K_3 & 0 & 0 \\ 0 & 0 & 0 & K_4 & 0 \\ 0 & 0 & 0 & 0 & K_5 \end{bmatrix}$$

By using the Moore-Penrose pseudoinverse given by

$$\boldsymbol{T}^\dagger{}_e = \boldsymbol{T}_e^\top(\boldsymbol{T}_e\boldsymbol{T}_e^\top)^{-1} \tag{2.10}$$

[12], we can find the extended control input vector $\boldsymbol{u}_e$ with the equation

$$\boldsymbol{u}_e = \boldsymbol{K}_e^{-1}\boldsymbol{T}_e^\dagger\boldsymbol{\tau} \tag{2.11}$$

The propeller commands for each thruster can be found by

$$n_i = \text{sgn}(u_i)\sqrt{|u_i|} \tag{2.12}$$

As the azimuth thrusters are decomposed in two pairs u_{i_x} and u_{i_y}, the control inputs u_i is found by

$$u_1 = \sqrt{u_{1_x}^2 + u_{1_y}^2} \text{ and } u_2 = \sqrt{u_{2_x}^2 + u_{2_y}^2} \tag{2.13}$$

while the rotation of the azimuth thrusters can be found by

$$\alpha_1 = \text{atan2}(u_{1_y}, u_{1_x}) \text{ and } \alpha_2 = \text{atan2}(u_{2_y}, u_{2_x}) \tag{2.14}$$

[12].

2.1.3 Guidance of marine vessels

Open-loop guidance of a marine vessel is the task of making a reference trajectory or a path that a vessel can track or follow. In this paper we use guidance in form of making a path such that the vessel can do a time-invariant path following. This task can be solved in several ways, but we will in this paper use Line-of-sight (LOS) to find the desired heading and use it in combination with waypoints found by a deep Q-network and deep active inference.

Line-of-sight (LOS)

Before going into detail of the LOS guidance law there is a need to define a few terms to be able to do heading control. While the heading angle of the vessel is the direction in which the vessel is pointing with its nose, the course angle is the direction of the movement of the vessel. This course angle χ is defined by

$$\chi = \psi + \beta_c \tag{2.15}$$

where ψ is the heading or yaw, and β_c is the crab angle [12]. A visualization of the angles can be seen in Figure 2.1. The crab angle is the difference between the course angle and heading angle, and can be computed by

$$\beta_c = \sin^{-1}(v/U) \tag{2.16}$$

where U is the absolute velocity of the vessel computed by

$$U = \sqrt{u^2 + v^2} \tag{2.17}$$

When doing path following, the goal is to make the vessel follow a predefined path without having temporal constraints. With LOS three different points in the

environment, which forms a triangle, are used to make the marine vessel move to a target position. This target position is one of the three points, where the two others are the interceptor which is the marine vessel itself and the stationary reference point which is a point used to make the triangle. With these points a course angle χ_d can be found which again can be used to find a desired heading [12]. The desired course angle χ_d can be found by

$$\chi_d = \pi_d - \tan^{-1}(K_p y_e^p) \tag{2.18}$$

where y_e^p is the cross-track error and K_p is the gain. In this equation, the expression π_d is called the path-tangential angle and is the angle of the vector from the reference point to the target from north in a NED-frame [12]. This angle is computed by the expression

$$\pi_d = \text{atan2}(y_t^n - y_{\text{ref}}^n, x_t^n - x_{\text{ref}}^n) \tag{2.19}$$

In this expression, (x_t^n, y_t^n) is the position of the marine vessel and (x_{ref}^n, y_{ref}^n) is the position of the reference point. To find the desired course angle χ_d, the inverse tangent of $K_p y_e^p$ have to be calculated. The gain K_p is given by

$$K_p = 1/\Delta \tag{2.20}$$

where $\Delta > 0$ is called the lookahead distance which is the distance ahead the marine vessel is looking for a point between the stationary and target point [12]. The expression y_e^p is the cross-track error in the path-tangential reference frame. It is given as the error between the vessel and the closest point on the line between the reference point and the target point. With path following we wish to make this cross-track error go to zero.

Integral LOS

When there are no velocity measurements it is impossible to calculate the crab angle β_c as it is directly computed by the sway velocity and the absolute velocity. In these cases, an integral action can be added to compensate for the drift term β_c. The conventional way of adding integral action yields the expression

$$\psi_d = \pi_d - \tan^{-1}(K_p y_e^p + K_i \int_0^t y_e^p(\tau) d\tau) \tag{2.21}$$

where K_i is the integral gain [12]. However, there are a few issues with this integral action as there are no global stability results. Instead we can use a nonlinear guidance law given by

$$\psi_d = \pi_d - \tan^{-1}(K_p y_e^p + K_i y_{\text{int}}^p) \tag{2.22}$$

where

$$\dot{y}^p_{\text{int}} = \frac{\Delta y^p_e}{\Delta^2 + (y^p_e + \kappa y^p_{\text{int}})^2} \tag{2.23}$$

as given in [12]. In the differential equation for y^p_{int} the gain K_p is the same as in (2.20) and the integral gain is given by $K_i = \kappa K_p$ where $\kappa > 0$ is a design parameter.

2.1.4 Control of marine vessels

To control the thruster forces such that the vessel moves accordingly to the guidance system a motion control system is needed. A motion control system can be designed with several different techniques such as PID control, linear-quadratic optimal control, neural networks and nonlinear control theory.

SISO nonlinear PID control

Consider a single-input single-output (SISO) system such as

$$m\ddot{x} + d\dot{x} + kx = \tau \tag{2.24}$$

This system can be controlled by the control law

$$\tau = kx_d - (K_p\tilde{x} + K_d\dot{x} + K_i \int_0^t \tilde{x} d\tau) \tag{2.25}$$

where x_d is the reference, $\tilde{x} = x - x_d$ is the tracking error, and $k > 0$, $K_d > 0$ and $K_i > 0$ are various gains [12]. The expression inside the brackets is called a PID controller and to find the gains K_p, K_d and K_i we have to perform pole-placement. Consider a mass-damper-spring system such as

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega^2 x = 0 \tag{2.26}$$

where ζ is called the relative damping factor and ω_n is called the natural frequency. By adjusting the relative damping factor and the natural frequency it is possible to adjust the poles of the system as the eigenvalues of the mass-damper-spring system is given by

$$\lambda_{1,2} = -\zeta\omega_n + -j\omega \tag{2.27}$$

where ω is the frequency of the damped system [12].

By inserting the control law into (2.24) we get

$$m\ddot{x} + (d + K_d)\dot{x} + (k + K_p)x + K_i \int_0^t \tilde{x} d\tau = 0 \tag{2.28}$$

If we assume $K_i = 0$ we can compare this equation to the equation of the mass-damper-spring system and get that

$$\zeta = \frac{d + K_d}{2m\omega_n} \tag{2.29}$$

and

$$\omega_n^2 = \frac{k + K_p}{m} \tag{2.30}$$

which gives us the gains

$$K_p = m\omega_n^2 - k \tag{2.31}$$

and

$$K_d = 2\zeta\omega_n m - d \tag{2.32}$$

[12]. The integral action is present to counteract constant and slowly-varying disturbances and is given by

$$K_i = \frac{\omega_n}{10} K_p \tag{2.33}$$

The natural frequency ω_n is related to the bandwidth ω_b with the equation

$$\omega_n = \frac{\omega_b}{\sqrt{1 - 2\zeta^2 + \sqrt{4\zeta^4 - 4\zeta^2 + 2}}} \tag{2.34}$$

[12]. To perform the SISO PID pole-placement, the bandwidth $\omega_b > 0$ and the relative damping ratio $\zeta > 0$ have to be specified and from these the natural frequency can be calculated. Furthermore, the relative damping ratio and natural frequency can be used to find the gains of the PID controller.

MIMO nonlinear PID control

Consider a system that is multiple-input multiple-output (MIMO), for example the marine vessel model presented earlier in this chapter. The PID controller is now given by

$$\boldsymbol{\tau}_{\text{PID}} = -\boldsymbol{K}_p\tilde{\boldsymbol{\eta}} - \boldsymbol{K}_d\dot{\boldsymbol{\eta}} - \boldsymbol{K}_i \int_0^t \tilde{\boldsymbol{\eta}}(\tau)d\tau \tag{2.35}$$

where $\boldsymbol{\eta}$ is the state vector, $\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta} - \boldsymbol{\eta_d}$, and $\boldsymbol{K}_p > 0$, $\boldsymbol{K}_d > 0$ and $\boldsymbol{K}_i > 0$ are diagonal matrices with gains on the diagonal [12]. The algorithm for PID pole-placement for MIMO systems is similar to the algorithm for pole-placement for SISO systems. First, a matrix of bandwidths $\Omega_b = \text{diag}(\omega_{b_1}, ..., \omega_{b_n})$ and a matrix of relative damping ratios $\boldsymbol{Z} = \text{diag}(\zeta_1, ..., \zeta_n)$ where n is the size of the state vector η have to be specified. Then the matrix of natural frequencies Ω_n can be found by calculating each element on the diagonal with (2.34). The the gains $\boldsymbol{K}_p > 0$, $\boldsymbol{K}_d > 0$ and $\boldsymbol{K}_i > 0$ can be calculated with the equations

$$\boldsymbol{K}_p = \boldsymbol{M}^*(\boldsymbol{\eta})\boldsymbol{\Omega}_n^2 \tag{2.36}$$

$$K_d = 2\boldsymbol{M}^*(\boldsymbol{Z})\boldsymbol{\Omega}_n^2 \tag{2.37}$$

$$\boldsymbol{K}_i = \frac{1}{10}\boldsymbol{K}_p\boldsymbol{\Omega}_n \tag{2.38}$$

[12].

Dynamic positioning

When performing stationkeeping and low-speed maneuvering of a vessel the control systems are called Dynamic positioning (DP) systems. These systems perform simultaneous control of surge, sway and yaw and have traditionally been used when the goal is to keep the vessel in a fixed position or move it slowly from one position to another. However, today more high-speed operation functionality and DP are used together, such that DP can be used for all speed ranges and types of operations [12].

When it comes to the DP control system, it is important that it is robust and can compensate for environmental forces such as wind and waves, and unmodeled dynamics. The DP controller can be designed in a similar manner to a MIMO nonlinear PID controller. The control law is given as

$$\boldsymbol{\tau} = -\tilde{\boldsymbol{\tau}}_{\text{wind}} + \boldsymbol{R}^\top(t)\boldsymbol{\tau}_{\text{PID}} \tag{2.39}$$

where $\boldsymbol{\tau}$ is given as in (2.35) and $\tilde{\boldsymbol{\tau}}_{\text{wind}}$ is an estimate of the generalized forces of the wind [12]. By writing this expression out we get the control law

$$\boldsymbol{\tau} = -\tilde{\boldsymbol{\tau}}_{\text{wind}} - \boldsymbol{R}^\top(t)\boldsymbol{K}_p\tilde{\boldsymbol{\eta}} - \boldsymbol{R}^\top(t)\boldsymbol{K}_d\dot{\boldsymbol{\eta}}\boldsymbol{R}(t) - \boldsymbol{R}^\top(t)\boldsymbol{K}_i\int_0^t \tilde{\boldsymbol{\eta}}(\tau)d\tau \tag{2.40}$$

2.2 Artificial neural networks

An artificial neural network (ANN) is a technique for learning and recognizing structures in data which are based on how biological neural networks works. For example, a human neuron are cells in the neural system which are connected to other neurons in a network. These neurons communicate with each other by sending electrochemical signals between each other. Put in a simple manner, when a neuron receives a combination of input signals that exceeds a threshold the signals will travel through the neuron and then transmitted to the neighbouring neurons. One of the first implementations to imitate this behaviour of a neural network was the perceptron invented by Frank Rosenblatt [41]. A perceptron can be given by

$$y = \begin{cases} 1 & \text{if } \boldsymbol{w}\top\boldsymbol{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.41}$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the weights vector and b is the bias. A perception works as a binary classifier by taking several binary inputs and output a single binary output, where each input has a corresponding weight. The weights quantify how important an input is for the output. When the weighted sum of the input is larger than the threshold value, in this case 0, the neuron outputs 1 and when it is below the threshold the neuron outputs zero [35].

The perceptron is simple as it only can be used for binary classification. However, when the perceptrons are connected together in layers it is possible to model very complex functions. The systems of perceptrons connected together are called ANNs. An ANN consists of multiple nodes which are connected to each other through links. These links are weighted by how important the link is corresponding to the output. An example of how these nodes and links are structured can be seen as a feed-forward network in Figure 2.3. First, the input layer receives a vector of inputs with data $\boldsymbol{x}$. The data are then propagated through the links to the next layer, which is called the hidden layer. In the hidden layer will each node compute the weighted sum of its input which can be represented by the equation

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.42}$$

where $\boldsymbol{y}$ is the output of the node and $\boldsymbol{W}$ is the weight matrix. The weighted sum of its input is then sent through an activation function $f(\cdot)$. This function decides how the output should sent further. Functions such as the ReLU, tanh and sigmoid functions are common activation function which gives the output different characteristics [1]. For example, when ReLU is used as the activation function every negative weighted sum is set to zero. After the output $\boldsymbol{y}$ is calculated it is propagated through to the next layer. The next layer can be an output which outputs the calculations of the network or it can be another hidden layer if the network consists of several hidden layers. When a network has several hidden layers it is called a deep neural network where each layer represents more complex patterns features of the input. This enables the network to model complex functions [1].

For a network to be able to model the functions we want, it needs to be trained such that it learns the function. In training the weights of the network are adjusted such that the network performs better. How the network is trained is dependent on if the network has examples available or not. If the network has example input-output pairs it can use the pair to see what it should output for a given input and adjust accordingly. This is called supervised learning. If the network only has inputs but does not know the corresponding output the learning is called unsupervised. In unsupervised learning the network itself has to learn patterns and features from the input.

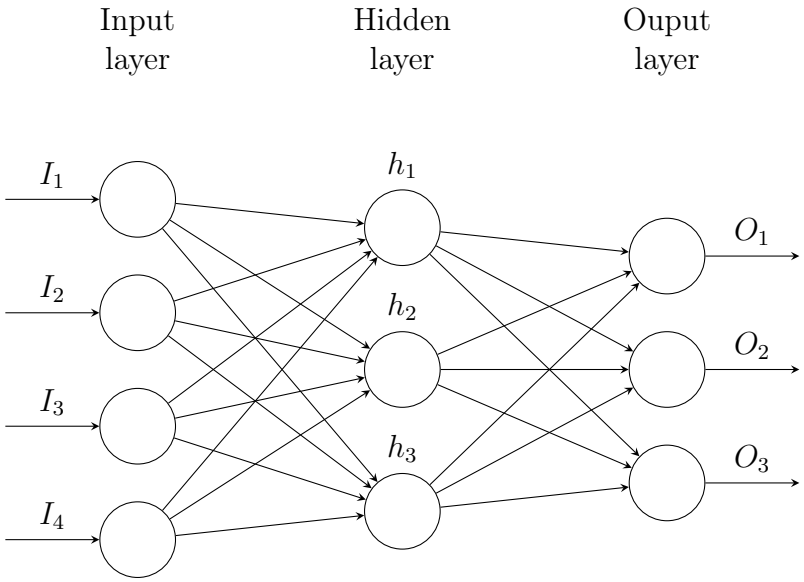


Figure 2.3: A feed-forward neural network.

Assume that the training of the network is done with supervised learning where it has available some input $\boldsymbol{x}$ with corresponding outputs $\boldsymbol{y}$ which we call target values. When training the network with supervised learning the task is to adjust the weights and biases such that the network outputs the target value for a given input. This can for example be done with gradient descent [35]. The network first calculates the output from an input $\boldsymbol{x}$ by performing a forward pass, which is propagation of the input through the hidden layers and to the output layer. Then, a loss function is used to find the error between the output calculated by the network and the target. The error is backpropagated from the output layer through all the hidden layers such that the weights of the links are adjusted to fit the network to the training data. The update of the weights in gradient descent can be exemplified through the expression

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{b}) \tag{2.43a}$$

$$\boldsymbol{b} = \boldsymbol{b} - \alpha \nabla_{\boldsymbol{b}} J(\boldsymbol{w}, \boldsymbol{b}) \tag{2.43b}$$

where $\boldsymbol{J}(\boldsymbol{w}, \boldsymbol{b})$ is the loss function and α is the learning rate [35].

Training of ANNs does not come without challenges, especially for deep ANNs. One of the most prominent challenges is overfitting. If a network is overfitted it performs well on the training data, but given unseen test data it does not manage to predict the correct outputs. Solutions to this problem include regularization of the weights, early stopping and trading off breadth for depth [1]. Another issue is that the gradient might vanish or explode when backpropagating in a network with many hidden layers. Also, it might be difficult for the network to converge as many hidden layers makes it harder for the gradients to flow smoothly through the environment. To counteract this gating networks and residual networks has been proposed [1].

2.3 Reinforcement learning

2.3.1 Markov decision process

A Markov decision process (MDP) is a way of modelling a sequential decision problem where the outcomes are under control of a decision maker. In this paper the decision maker is called an agent. AMDP is an environment defined by a finite set of states S, a finite set of actions A, the stochastic transition model $T(s'|s, a)$ and a reward function $R(s)$. The agent can be located in any of the state $s \in S$ and decides between any of the actions in A. When the agent takes an action $a \in A$, the transition model T specifies the next state of the environment $s' \in S$ from the current state and action. If the state transitions of the environment are independent of the previous states and action except the latest state and action, the model is Markov. When reaching a new state the agent receives a reward $R(s)$ corresponding to the new state [26].

A solution to the MDP is called a policy. A policy $\pi(s)$ specifies what action the agent should take for every state the agent can reach. When the policy is complete the agent will always know what to do when reaching a state, no matter the previous state and action. However, as the environment is stochastic in its transitions, the agent may move differently out in the environment even when starting position is the same. Hence, the goal of the agent is to find the policy that maximizes the cumulative rewards, in other words maximizes the expected utility. A policy that maximizes the expected utility is denoted by π^* [42].

There also exists cases where the decision process is MDP, but the states are not directly observable. This is called a partially observable Markov decision process (POMDP) and the states can only be accessed indirectly via what is called observations that are stochastically related to the state [46].

2.3.2 Value functions

To be able to maximize the expected utility, the agent must be able to estimate the expected utility of a certain state to find out how good it is for the agent to be located in this particular state. In other words, the agent has to find what rewards can be expected in the future. To find the expected utility, value functions are used and as they are dependent on what action the agent takes, the value function are defined with respect to policies [46].

The value, or expected utility, of a certain state s under a policy π is the expected return when starting in s and then following the policy π. This can be given by

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t * R_{t+k+1} | S_t = s\right] \tag{2.44}$$

where $0 < \gamma < 1$ is the discount factor and $E[\cdot]$ is the expected value. As the value functions returns the value of a particular state when following a policy π, this is called the state-value function for policy π [46]. From this equation we see that it is a connection between the current state s and its neighbouring states.

It is also possible to define a value function for a state-action s and a pair under a policy π. This value function is defined by the expected return when starting in s, taking the action a and then following the policy π

$$Q^{\pi}(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t * R_{t+k+1} | S_t = s, A_t = a\right] \tag{2.45}$$

This value function is called the action-value function for policy π [46].

These value functions can be used to find the optimal policy. A policy that has a higher or equal expected utility than another policy is said to be better than the other policy. Thus $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for every $s \in S$ [46]. The optimal value function, i.e. the function that corresponds to the optimal policy π^*, is given by

$$V^*(s) = \max_{\pi} V^{\pi}(s) \forall s \in S \tag{2.46a}$$

$$Q^*(s) = \max_{\pi} Q^{\pi}(s, a) \forall s \in S, a \in A \tag{2.46b}$$

By assuming that the agent chooses the optimal action, the utility of a state is the reward of the current state and the expected discounted value of the next state [42]. As the environment is stochastic we can write the state-value function as

$$V(s) = R(s) + \gamma * \max_{a \in A(s)} \sum_{s'} T(s'|s, a) V(s') \tag{2.47}$$

This is called the Bellman equation and for a MDP with a finite state and action set this equation yields an unique solution independent of the policy. The Bellman equation can be calculated for every N states starting with an initial guess for the value function in every state. Then, the Bellman equation is solved to update the value of each state. As the value of the states are changed, the values have to be updated again with the Bellman equation. This is performed until the value for each state have converged to a stationary value, and the optimal policy can be determined [46]. This is called dynamic programming as the problem of finding an optimal policy is divided into simpler subproblems of finding the optimal action for

each state. Two reinforcement learning algorithms that uses dynamic programming to find the optimal policy is value iteration and policy iteration.

2.3.3 Q-learning

To find the optimal policy with the value functions presented in the previous section, a model of the environment with the reward function and the transition model has to be available to the agent. However, this is not always the case, which means that the agent itself must explore the environment to learn the effects of the actions. By moving through the environment, the agent obtains experience which can be used to approximate the value function. This is called temporal-difference (TD) learning and by using experience to solve the prediction problem, there is no need for prior information about the environment [46].

The TD-algorithms updates the value estimate of a state by using the received reward and the estimated value of the next state. This is shown by the TD-method TD(0), where the update of the value of a certain state is given by

$$V(s) = V(s) + \alpha \left[r + \gamma V(s') - V(s) \right] \tag{2.48}$$

where α is the learning rate which decides how much the estimate is updated to changes in V. When the agent moves to a new state s, the value of the states is updated to be closer to what we call the target $r + \gamma V(s')$, where r is the reward from state s' and $V(s')$ is the estimated value of the next state. This update rule is guaranteed to converge to the optimal value function if the learning rate is adjusted properly and the policy is fixed [26].

The update rule in TD(0) can also be written with an action-value function which yields the update function

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{2.49}$$

This update function calculates the value function for the actions explicit and is called SARSA as it is applied at the end of the sequence s, a, r, s', a'. This is an on-policy method, which means that the value function calculates Q^π for a policy π [46]. In addition, the policy is changed in a greedy way which means that the agent will update the policy with the action with the highest value in a particular state.

There also exists an alternative TD-method which updates the value function $Q(s,a)$ in an off-policy fashion called Q-learning. Instead of using a target with a value $Q(s',a')$ from the action a' taken in s', the value function is updated with the maximum Q-value of the possible actions in s'. This gives us

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{2.50}$$

This is the equation used for updating the Q-value and the Q-learning algorithm can be seen in Algorithm 1. First, the value $Q(s,a)$ is initialized for every state $s \in S$ and $a \in A$. Then, from a starting state s an action a is chosen and performed. When reaching the next state, the agent receives a reward and the value function is updated for the action a. This is repeated until the agent reaches a terminal state. The estimates of the Q-values will converge to the optimal values if each action in every state is executed an infinite number of times and if the learning rate is decayed properly. This is also independent of how the agent explores the environment [26]. However, this is intractable and the preceding sequence is only executed to obtain a good estimate of the value function for every state-action pair. When the estimation of the Q-values is finished, the policy is found greedily by choosing the action with the highest value in each state.

Algorithm 1 Q-learning

Require: Initialize $Q(s,a) \forall s \in S$ and $a \in A, \gamma, \alpha$
  for each episode **do**
    Initialize starting state s
    repeat
      Choose an action a from state s using an exploration strategy
      Take action a, observe the next state s' and the reward r
      $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a \in A} Q(s',a) - Q(s,a))$
      $s \leftarrow s'$
    until s is a terminal state
  end for

2.3.4 Exploration and exploitation

An important topic in reinforcement learning is how the agent explores the environment in the case where it does not have a model. If the agent explores the environment in an insufficient manner it might not learn the optimal policy at all. At one hand, the agent can take an greedy action. A greedy action is the action which has the highest estimated value associated with it. This is called exploitation as the agent exploits the knowledge about the values of the actions to make a choice. At the other hand, the agent can choose a non-greedy action. This is called exploration as the agent chooses the opportunity to get a more accurate estimate of the non-greedy action. Where exploitation will maximize the reward at the current step, exploration may lead to obtaining greater reward in the future.

It is important for the agent to have the correct trade-off between exploration and exploitation to find the best possible policy. If the agent is greedy and chooses

the greedy action every time, the policy may converge to a policy that is far from optimal for the whole environment. Thus, it is also important for the agent to also explore the environment to improve the model of the environment. Even though the agent will not obtain maximal rewards in short term, exploration can lead to greater rewards in the future as the agent may find another path that will yield greater rewards. However, an agent who only explores the environment will never use its knowledge and will seldom maximize its rewards.

A method called ϵ-greedy is a way of balancing exploration and exploitation. By using this method the agent chooses the greedy action as default, but with probability ϵ the agent selects a random action. In this way, every action will be sampled an infinite number of times when the number of episodes goes towards infinity [46]. The ϵ-greedy method can be implemented in a number of ways. When using an ϵ-greedy method in this paper, we will use an epsilon which starts with value one and decays towards zero as the number of episodes increases. The agent performs action selection by drawing a random number between zero and one and compares it to the ϵ. If the random number is larger than ϵ it exploits its knowledge and chooses the action with the highest value. Otherwise, it explores the environment and chooses a random action. By using this strategy, the agent explores the environment in the beginning when it has little information and as it obtains more and more information about the envioronment it exploits its knowledge to find the optimal path.

2.3.5 Deep Q-learning

When finding an optimal policy with the Q-learning algorithm presented in Chapter 2.3.3, the Q-values of each state-action pair is stored in a table. This means that when the state space and action space grows, the Q-table where the Q-values are stored also grows. When the state and action space gets to big, it is intractable to use the required memory for storing all Q-values. Also, as a Q-learning agent has to visit a state to estimate the Q-value, the exploring of the environment would take very long time. To solve these issues, the Q-learning algorithm can be combined with a deep neural network to approximate the Q-values, which gives us deep Q-learning.

Instead of storing the Q-values for every state-action pair in a table, a deep neural network is used to map the state-action pair to the corresponding Q-value. We will here look at Deep Q-learning when receiving the input as a visual input in terms of pixels, as is done in the papers [32] and [33] by DeepMind who first presented the deep Q-learning algorithm. The input to the network is given by the preprocessing of the m most recent frames of the environment. This is further sent through the network, which in this case is a convolutional network. Instead of also sending in a corresponding action to the state, as done when calculating $Q(s, a)$, the network outputs the estimated Q-values of every action which is more

effective than computing the Q-value for each state-action pair in separate calls [33].

When training, the network it is initialized with parameters θ. First, a starting state $s_1 = x_1$ is initialized. Then, at each time t the agent selects an action a_t according to an ϵ-greedy policy from the set of actions. After performing the action, the agent receives an image x_{t+1} that represent the current state of the agent, for example a vector of pixel values. In addition, the agent receives a reward r_t. As the agent only observes the current screen x_t, the environment is partially observed [33] which means we have a POMDP. This means that the combination of the previous state s_t, the action a_t and the next frame x_{t+1} have to represent the next state, as the whole situation cannot be understood from only one frame. Thus, these parts of the environment is used as the state representation and stored in s_{t+1}. Before the state is used as an input in the network it is preprocessed by the function $\phi(.)$ to reduce the input dimensionality.

When using a nonlinear function approximator such as a neural network to represent the action-value function, reinforcement learning methods are prone to be unstable and might diverge [33]. One of the reasons for the instability is that there is a correlation in the sequence of observations. To address this issue a technique called experience replay is used. In experience replay we define the transition $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$ as the experience at the time step t and store this in the data set D. The data set D is called the replay memory and when training the deep Q-network (DQN) a random set of transitions is sampled from the replay memory and applied to the target function

$$y_j = \begin{cases} r_j & \text{if episode terminates as step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases} \tag{2.51}$$

When using experience replay the correlations between the consecutive samples are removed which reduces the variance in the samples. Also, experience replay secures that the network is not stuck in local minimum or diverges. If the network was trained with consecutive samples, a maximizing action of moving to the left would make the training dominated with samples from the left side of the environment. With experience replay, the action distribution is averaged over the previous states which makes sure that oscillations and divergence in the parameters are avoided [32].

There are also other issues with using a neural network as a function approximator in a reinforcement learning algorithm. Small updates to the action-value function can lead to big changes in the policy which further can change the data distribution and the correlations between target values and the action-value function [33]. This can lead to oscillations or divergence in the policy. To address this issue, a target network is used when calculating the target function. This can be seen in (2.51) where a target network $\hat{Q}$ is used instead of the main network Q. The parameters of the target network θ^- is updated every C steps to be equal to the parameters of

the main network θ, and are otherwise kept constant. Without a target network when calculating the target function, the updates in $Q(s_t, a_t)$ will often increase $Q(s_{t+1}, a_t)$ for every action which in turn increases the target value as it is a function of $Q(s_{t+1}, a_t)$. By calculating the target values with a target network there is a delay between the update of the main network Q and when this affects the target y_j. This leads to less oscillations and divergence in the policy is less likely [33].

A gradient descent step is performed on the loss function

$$L(\theta_j) = (y_j - Q(\phi_j, a_j; \theta))^2 \tag{2.52}$$

with respect to the parameters θ. The loss function is given as the mean square error of the expected Q-value calculated with the target function and the estimated Q-value from the network. This yields the update rule of the weights

$$\theta = \theta + \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i) \tag{2.53}$$

This is run for M episodes until the network hopefully is trained well enough to give a good approximation of the action-value function. Then, the Q-network can be used to find the optimal policy just as in Q-learning.

Algorithm 2 Deep Q-learning

Require: Initialize a replay memory D to capacity N.
Require: Initialize a action-value function Q with weights θ.
Require: Initialize a target function $\hat{Q}$ with weights $\theta^- = \theta$
  for each episode **do**
    Initialize sequence $s_1 = x_1$ and preprocessed sequence $\phi_1(s_1)$
    for each $t = 1, ..., T$ **do**
      With probability ϵ select a random action a_t,
      otherwise select $a_t = \text{argmax}_a\, Q(\phi(s_t), a; \theta)$
      Execute action a_t and observe reward r_t and image x_{t+1}
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi(s_{t+1})$
      Store transition $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$ in D
      Sample random minibatch $(\phi(s_j), a_j, r_j, \phi(s_{j+1}))$ from D
      Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
      Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ wrt. to θ
      Every C steps reset $\hat{Q} = Q$
    end for
  end for

2.3.6 Deep Deterministic Policy Gradient

While we can solve problems with high-dimensional state spaces with DQNs, it is not possible to solve problems with DQNs when the action space becomes high-dimensional. For example, if the action space is continuous action it is impossible to use DQNs directly as it wants to find the action-value for every action. A possible solution to this problem is to discretize the action space, but this can fast become intractable as the number of actions increases exponentially with the degrees of freedom of the agent [28]. Instead it is possible to use an algorithm called DDPG which uses approaches from DQN with deterministic policy gradients to solve problems with continuous action spaces.

The DDPG-algorithm uses two ANNs which are called the actor, denoted $\mu(s|\theta^{\mu})$, and the critic, denoted $Q(s, a|\theta^{Q})$. The actor network, which also is called the policy network, takes in the state and outputs the action according to the current policy. The other network, the critic, takes in a state-action pair and outputs the Q-value of the pair. When training the networks a start state is used as a input in the actor network which outputs the action according to the current policy. Also, noise is added to the action for exploration of the environment as this is a big challenge with continuous actions [28]. Then the action is executed and after the next state and the reward are observed the transition between the states are stored in a replay buffer as experience replay are used in this algorithm as well.

Further, a minibatch of transitions are sampled and the target function is calculated by

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}, \theta^{\mu'}|\theta^{Q'})) \tag{2.54}$$

where r_i is the reward and γ is the discount factor [28]. In the calculation of the target function we use target networks Q' and μ' for the same reasons as for DQN. After calculating the target the weights of the critic is updated by minimizing a loss with a loss function as in (2.52). Then, the actor is updated by using policy gradient [28].

Chapter 3

Active inference

This chapter presents theory about active inference. To lay a foundation for the theory about active inference, the basics of the free-energy principle is explained. In addition, algorithms for using active inference on larger state spaces are presented.

3.1 The free-energy principle

As active inference utilizes the free energy principle, we need to have an understanding of what the free energy principle is.

The free-energy principle is a principle developed by Karl Friston that tries to explain how the brain works by providing a unified account for action, perception and learning in the brain. The principle states that a self-organizing system that is contained inside a set of states has to minimize its variational free energy. This means that the system must remain inside a set of states defined for the specific system even when the external environment changes [16]. For example, a human being will regulate the internal temperature inside a specific interval to stay alive.

The variational free energy quantity is defined as an information theory measure that is the upper bound on what is called surprise. Systems that wants to minimize the variational free energy can for example be biological systems. A characteristic of biological systems is that they try to remain in the states defined for that specific system. A system has a certain amount of states it can be located in, both physical and sensory states. These states define the phenotype of the system, i.e. the traits of the system. Thus, the system has a high probability of being located in these states and a low probability of being in the remaining states [16]. These states can be perceived by the system through sensory states and the sensory states defined for the system will have low entropy. The entropy is the average surprise

of an outcome. This means that the states with a high probability will not be surprising for the system, while the states assigned low probability are surprising. Surprise is hard to quantify and the value of surprise will be different for different systems in equal situations. Consequently, instead of trying to find the surprise, the variational free energy can be used. As the variational free energy is an upper bound on surprise it is possible to minimize the surprise indirectly by minimizing the variational free energy. In this way a system can be at its equilibrium in its environment and the free-energy principle can be used to account for how systems act, percept and learn [16].

The free-energy principle uses that a self-organizing system, which we can call an agent, remains at equilibrium in their environment in a mathematical formulation. Before we get to the equations that describe the free-energy principle we need to define the variables of the environment, which include the internal variables of the brain, the external environment outside the agent and the sensory states. The interactions between the states can be formalized by a Markov blanket. A Markov blanket defines a statistical boundary and divides the system into internal and external states, with a blanket of states that separate the two parts of the system. The states in the Markov blanket consists of active and sensory states and the internal and external states are conditionally independent because they can only influence each other via the active and sensory states in the blanket [27].

Every living system must have a Markov blanket. The reason for this is that to be alive, the system must have some degree of conditional independence from the environment. Without the Markov blanket it would be impossible to distinguish the living system from the environment and it could not be possible to prove its existence [8]. Hence, we describe the environment with external states that are hidden for the internal state behind the Markov blanket, which influence sensory states. These sensory states influence the internal states which again influence the active states that influence the external states [27]. An example of this is the human brain. It cannot directly perceive the states outside the skull, but gets sensory inputs in the form of sound, sight, touch etc., which are influenced by the external states. Then the sensory inputs influence the active states, which here is muscles, to act in the external environment.

The sensory input, that the internal states are influenced by, are represented by the sensory states s. These sensory signals are generated by the external states. The external states are represented by $\vartheta \supset x, \theta, \gamma$ where x is the external hidden states, the parameters θ controls the amplitude of the noise of the external states w and the precisions γ controls the amplitude of the noise of the sensory input z [16]. An example of a hidden state is the location of the agent if the agent tries to move between two locations in an environment. From [16] we see that the sensory states is described by the probabilistic mapping

$$s = g(x, \vartheta) + z \tag{3.1}$$

where $g(x, \vartheta)$ is a continuous nonlinear function that describes the evolution of the

sensory states. The evolution of the hidden states x is described by

$$\dot{x} = f(x, a, \vartheta) + w \tag{3.2}$$

where $f(x, a, \vartheta)$ also is a continuous nonlinear function that describes the change in the external states and a is the action taken. As the agent is separated from the rest of the environment with a Markov blanket, it cannot obtain the external states directly. Hence, the agent must make an internal representation of the external world with the internal states. The internal states are represented by μ. These internal states are used to encode the recognition density $q(\vartheta|\mu)$ [16]. To make a model of the external environment, the brain infers the probability of the hidden states ϑ by using the sensory inputs s. By using Bayes' theorem, the probability density of the hidden states given a sensory input can be represented by the expression

$$p(\vartheta|s) = \frac{p(s|\vartheta) * p(\vartheta)}{p(s)} \tag{3.3}$$

A problem with this equation is that the denominator $p(s)$ is difficult to calculate which makes it difficult to calculate $p(\vartheta|s)$. The reason is that to calculate the probability $p(s)$, every possible state that can cause the sensory input s have to be included. To circumvent this issue, the recognition density is used as an approximation of $p(\vartheta|s)$ [4]. In this way, the brain uses the recognition density to represent the external world such that it can infer what caused a certain sensation. With the help of the internal states and the sensory input the brain tries to recognize the causes of the sensory input it receives with the use of this density.

Because the recognition density is an approximation of the true posterior $p(\vartheta|s)$, we wish to get the approximate distribution as close as possible to the true posterior. The measure of the difference between the two probabilities can be calculated with the Kullback-Leibler divergence [4], which is given by

$$D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta|s)} d\vartheta \tag{3.4}$$

The Kullback-Leibler divergence is also called cross entropy and calculates the non-negative between probability distributions [16]. If the two densities are equal, the ratio inside the natural logarithm is equal to 1. As the logarithm of 1 is equal to zero the whole expression for the Kullback-Leibler divergence would be 0. The Kullback-Leibler divergence also has the property that the larger the difference between the distributions, the higher the value will be [4]. Also, the Kullback-Leibler divergence is always larger than 0.

As we mentioned earlier, it is difficult to calculate $p(\vartheta|s)$ which makes it hard to calculate the Kullback-Leibler divergence. The posterior density $p(\vartheta|s)$ can be represented with the following expression:

$$p(\vartheta|s) = \frac{p(\vartheta, s)}{p(s)} \tag{3.5}$$

The density $p(\vartheta, s)$ is called the generative density and with this density the brain generates sensory samples and their causes. This density can be written as a product of a likelihood and prior distribution $p(s|\vartheta) * p(\vartheta)$, and is also called the generative model [19]. These densities represent the believes about how the states cause the sensory input and prior beliefs about the causes, respectively. Hence, the brain has an internal probabilistic model of the world.

By rewriting $p(\vartheta|s)$ and using that $\ln(ab) = \ln(a) + \ln(b)$ we get the expression for the Kullback-Leibler divergence

$$D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta + \ln p(s) \int q(\vartheta|\mu) d\vartheta \tag{3.6}$$

which can also be seen in [4].

As the integral in the rightmost term is equal to 1 we get that

$$D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta + \ln p(s) \tag{3.7}$$

In [19], the variational free energy is defined as

$$F(s, \mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta = -\int q(\vartheta|\mu) \ln \frac{p(\vartheta, s)}{q(\vartheta|\mu)} d\vartheta \tag{3.8}$$

By inserting this definition for the leftmost expression on the right hand side in (3.7), we get the following expression for the variational free energy

$$F(s, \mu) = D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] - \ln p(s) \tag{3.9}$$

which is made up by the Kullback-Leibler divergence between the recognition density and the true posterior density, and the negative logarithm model evidence, also called the surprise [16]. The surprise will output low values when the probability of the sensory input is large and high values when the probability of the sensory state is small. This means that if a sensory state is expected to be sensed, the state will have a high prior probability which yields a low surprise and vice versa.

The Kullback-Leibler divergence is defined to always be larger or equal to zero [15]. By using this in (3.9) we get that

$$F(s, \mu) \geq -\ln p(s) \tag{3.10}$$

Here it is seen that the variational free energy is an upper bound on the surprise. Hence, by minimizing the variational free energy, the surprise will indirectly be minimized. In this way minimization of variational free energy gives an explanation to what way self-organizing agents can avoid surprising states [16]. But how do they minimize the variational free energy?

3.2 Minimization of variational free energy

The agent can minimize the variational free energy by either changing its internal model of the world or change its sensory input. In this way the agent can either make its perception of the external world more accurate or take an action to make sure that the sensory input agrees with the internal model. By using these two mechanisms, the agents can avoid the sensory inputs that are associated with risk [8].

3.2.1 Perception

By optimizing the recognition density such that it resembles the conditional density on the causes of sensory input, the internal model gets more accurate. In the expression for the variational free energy, given in (3.9), it is seen that it consists of the Kullback-Leibler divergence between the recognition density and the conditional density of the causes of the sensory input. The Kullback-Leibler divergence is a measure of how different one probability density is different from a second one. Minimizing the variational free energy also minimizes the Kullback-Leibler divergence, which means that the recognition density will get closer to the posterior when the Kullback-Leibler divergence approaches zero [16]. When the recognition density is an approximate conditional probability the agent infers the causes of the sensory input in a Bayesian manner, i.e. it percepts in a Bayes-optimal fashion [22]. The minimization of the variational free energy with perception is done by changing the internal states μ according to the equation

$$\mu = \underset{\mu}{\operatorname{argmin}} F(s, \mu) \tag{3.11}$$

3.2.2 Action

In addition to minimize the variational free energy by perception, the agent can act on the environment such that the sensory input samples are consistent with the predictions of the recognition density. This can be seen by rewriting the expression for the variational free energy given in (3.8)

$$F(s,\mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(s|\vartheta)p(\vartheta)} d\vartheta \tag{3.12}$$

where the definition of conditional probability is used in the denominator. By using the quotient rule for logarithms, $\ln(\frac{a}{b}) = \ln(a) - \ln(b)$, and the product rule, $\ln(a \cdot b) = \ln(a) + \ln(b)$, we get that

$$F(s,\mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta)} + \int q(\vartheta|\mu) \frac{1}{p(s|\vartheta)} \tag{3.13a}$$

$$= \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta)} - \int q(\vartheta|\mu) \ln p(s|\vartheta) \tag{3.13b}$$

$$= D_{KL}[q(\vartheta|\mu)||p(\vartheta)] - \langle \ln p(s|\vartheta) \rangle_q \tag{3.13c}$$

The reason for wanting the variational free energy on this form is that now the variational free energy is given as complexity minus accuracy [16]. The complexity is given as the Kullback-Leibler divergence between the recognition density and the prior probability of external causes, while the accuracy is the surprise of the sensory input under the recognition density. In the case of the Kullback-Leibler divergence approaching zero, the variational free energy is only affected by the negative log evidence for the generative model, which is given by $-\ln p(s|\vartheta)$. Then, minimizing the variational free energy is the same as maximizing model evidence. This is the same as minimizing the complexity of accurate explanations for the observed outcomes [18].

The sensory input of the agent can only change when taking an action, which means that actions only affect the accuracy. To increase the accuracy, the agent must take an action that reduces the surprise. Essentially, when an action reduces the surprise, the sensory input agree more with the predictions made by the recognition density [16]. So, by minimizing the variational free energy with actions, the sensory input aligns more with the the predictions made by the recognition density which minimizes the prediction error. The action that minimizes the variational free energy can be found by

$$a = \underset{a}{\operatorname{argmin}} F(s, \mu) \tag{3.14}$$

as presented in [22].

3.3 Active inference

Active inference is a principle for solving and understanding how autonomous agents percept, act, plan and learn when they have to make a decisions under uncertainty. This principle is built upon the free-energy principle and uses optimization of variational free energy and expected free energy to explain perception, action, planning and learning in agents [7]. By minimizing the variational free energy, which describes the discrepancy between the internal model and the sensory input, and the finding the minimal expected free energy, which quantifies how well different policies scores towards prior preferences, Bayes-optimal behaviour can be achieved [44].

To represent the external world, which we have seen that the agent cannot access directly, the agent has a generative model which is used to represent the external

environment. By minimizing the variational free energy, the generative model gets close to the real environment. This is done by minimizing the variational free energy with perception. As the model is capable of making predictions about sensations and beliefs about the future states, it is considered to be generative [7]. By having a generative model, the agent can generate beliefs about the future, which means that it can find the policy which is the most coherent with prior beliefs such the outcomes becomes Bayes-optimal [44].

With the use of variational free energy the agent can find a generative model that describes the external environment and make inferences about the environment. However, if the agent for example receives surprising sensory input, the agent should also be able to perform actions to reduce the surprise. This is the active part in the expression active inference. In addition to minimizing the variational free energy, the agent can also minimize the expected free energy, denoted G. The expected free energy is the sum of the variational free energy for future trajectories [44]. As the agent only has beliefs of what is observed in the future, the expected free energy is dependent on the agents beliefs in the future. By minimizing the expected free energy, the agent can find the action path that realize the prior preferences such that the expected future surprise can be minimized [7].

This gives us the active inference-cycle: The agent minimizes the variational free energy to make sure that the internal model is consistent with the external environment. In this way the agent can make correct inferences about the environment. Then, it can act by minimizing expected free energy such that the future sensory input is consistent with the internal model to avoid surprise [7].

If we assume that the states and actions are discrete, the expected free energy of a given policy π is given as

$$G(\pi) = \sum_{\tau} G(\pi, \tau) \tag{3.15}$$

where $G(\pi, \tau)$ is the expected free energy at the time τ. This means that the agent has to find the expected free energy at each time step to calculate the complete expected free energy of the whole trajectory when following the policy. To calculate the expected free energy at a single time step we first have to rewrite the expression of the variational free energy to be dependent on both time τ and policy π. By rewriting the expression of the variational free energy in (3.13c) such that it is dependent on time and policy we get that

$$F(\tau, \pi) = D_{KL}(q(\vartheta_\tau|\pi)||p(\vartheta_\tau|, \vartheta_{\tau-1}, \pi)) - \langle \ln p(s_\tau|\vartheta_\tau)\rangle_q \tag{3.16}$$

[18]. By gathering the two terms into one and writing the expression out, we get the following expression for the variational free energy

$$F(\tau, \pi) = \sum_{\vartheta_\tau^\pi} q(\vartheta_\tau|\pi) q(\vartheta_{\tau-1}|\pi) \ln \frac{q(\vartheta_\tau|\pi)}{p(\vartheta_\tau, s_\tau|\vartheta_{\tau-1}, \pi)} \tag{3.17}$$

The expected free energy at a time step τ can be found by making two changes to this expression. By adding beliefs about future outcomes and condition the joint probability in the denominator on the desired outcome C, instead of a specific policy, the expected free energy can be obtained. The desired outcomes are the preferred states the agent wishes to be in. With these two changes the expected free energy can be calculated before any observations are obtained. Also, policies that yield results consistent with the desired outcomes are encouraged. These changes gives the following expression for the expected free energy

$$G(\tau, \pi) = \sum_{\vartheta_\tau, s_\tau} p(s_\tau|\vartheta_\tau) q(\vartheta_\tau|\pi) q(\vartheta_{\tau-1}|\pi) \ln \frac{q(\vartheta_\tau|\pi)}{p(\vartheta_\tau, s_\tau|\vartheta_{\tau-1}, C)} \tag{3.18}$$

which can be found in [44]. The expected free energy can be represented in different manners as we saw with the variational free energy. A representation that is consistent with both [44] and [18] is the following expression

$$G(\tau, \pi) = D_{KL}[q(s_\tau|\pi)||p(s_\tau|C)] + E_{\tilde{q}}[H(p(s_\tau|\vartheta_\tau))] \tag{3.19}$$

where $\tilde{q} = p(s_\tau|\vartheta_\tau) q(s_\tau|\pi)$. This is the sum of the expected cost, given by the Kullback-Leibler divergence, and the expected ambiguity, given by the expectation. This can further be written as

$$G(\tau, \pi) = s_\tau^\pi \cdot (s_\tau^\pi - \mathbf{C}_\tau) + s_\tau^\pi \cdot \mathbf{H} \tag{3.20}$$

which is the risk summed with the ambiguity. Here $s_\tau^\pi = \mathbf{A}\vartheta_\tau^\pi$ where $\mathbf{A}$ is the likelihood matrix mapping from the external states to the sensory input and $\mathbf{H} = -diag(E_q[\mathbf{A}_{i,j}], E_q[\mathbf{A}])$ encodes the ambiguity over the sensory input for every hidden state [44]. The expected cost, or the risk, is the uncertainty about the outcomes compared to the preferences C_τ. Furthermore, ambiguity is the uncertainty about the sensory input given the external states [18]. In this way it is ensured that the agent is both exploitative, such that it minimizes risk, and explorative to minimize the ambiguity about the states. By having this balance between exploring and exploiting, the expected free energy gives the optimal balance between the goal-seeking behaviour and the novelty-seeking behaviour [7].

Now, we have a way of calculating the expected free energy of a given policy π. The active inference agent wants to select the policy that minimizes the expected free energy such that it reaches the prior preferences C. To do that, a prior over the policies over policies is specified as in the following expression

$$q(\pi) = \sigma(-G(\pi)) \tag{3.21}$$

where σ is the softmax-function and $q(\pi)$ is called the approximate posterior over the policy [7]. From the expression it is seen that the policies that produces a lower expected free energy are more likely. The selection of the best policy comes down to which actions gets the agent closest to its preferred states [44].

With an approximate posterior over the policy, the agent can find the most plausible action at a given time step as the action that is the most likely under all possible policies. In this way the agent can select the best action to perform at a time step τ with the expression

$$u_\tau = \underset{u \in U}{\operatorname{argmax}}(\sum_{\pi \in \Pi} \delta_{u,\pi_\tau} q(\pi)) \tag{3.22}$$

where δ_{u,π_τ} is the Kronecker delta, U is the action set and Π is every possible policy [7].

To summarize, when receiving an observation, the agent will infer the hidden state it is in at a certain time step. By minimizing the variational free energy the inferences about the hidden state will be a good approximation of the real hidden states. Then, the agent evaluates the expected free energy for each possible policy, which is based on prior beliefs and preferences about future outcomes. After calculating the expected free energy of a policy, a probability of the policy can be calculated with a softmax function such that you get $q(\pi)$. This can be used to find the best action to perform next by summing the probabilities of the policies and select the action with the highest cumulative probability. The agent will repeat this cycle until it reaches the prior preferred outcomes.

3.4 Active inference on larger spaces

In the previous section, we have seen that active inference calculates the expected free energy of every possible policy to find the best actions to perform. The length of a policy is dependent on the size of the environment, which we can define as $nS \times nS$. When the size of the environment increases, the time it takes to calculate the expected free energy for a single policy increases. In addition, the number of policies needed to be calculated such that all policies are covered can be calculated by the expression nA^{nS}, where nA is the size of the action space. when both the environment and the action space is big enough it will be intractable to find the optimal policy with active inference as explained in the previous section. To solve these issues such that active inference can be used in larger discrete environments or in continuous state-spaces active inference, we can use deep active inference or sophisticated active inference.

3.4.1 Deep active inference

In recent time, there has been several examples on how to solve tasks with higher complexity with active inference. These methods all use neural networks to approximate different densities, but there are differences in how they calculate variational free energy and expected free energy. For example, [47] uses a reduced version of the expected free energy, [13] uses a Monte-Carlo tree search to find the free-energy-optimal policy, while [31] and [25] calculates an approximation of the expected free energy with bootstrapping. In this paper, we will look closer at [31] and [25] when getting into deep active inference.

The deep active inference algorithm presented in this section utilize a lot of the same formulations as for the active inference presented in the previous section. From now on this will be called tabular active inference as it represents the state-space explicitly as a table which gets larger and larger when the environment scales up.

As we have explained, the agent has a generative model of the environment which is used to make inferences about the external environment. Earlier we have only described this with the hidden states and the observations, as we have assumed that the agent can directly access its own actions. But now we also add the actions which in addition to the hidden states cannot be observed directly. Thus, in addition to inferring the hidden states, the agent must also infer the actions. This means that the generative model can be represented by $p(\vartheta, s, a)$. In (3.8) we saw that the variational free energy is represented by the Kullback-Leibler divergence between the recognition density $q(\vartheta|\mu)$ and the generative model $p(\vartheta, s)$. By also adding the action into the recognition density and the generative model, the varitional free energy can be expressed by

$$F = D_{KL}[q(\vartheta, a)||p(\vartheta, s, a)] \tag{3.23}$$

which is the Kullback-Leibler divergence between the recognition distribution and the generative distribution. The generative distribution can be factorized in the following manner

$$p(\vartheta, a, s) = p(s|\vartheta)p(a|\vartheta)p(\vartheta|\vartheta_{t-1}, a_{t-1}) \tag{3.24}$$

The recognition distribution $q(\vartheta, a)$ can also be factorized, where we use Bayes theorem

$$q(\vartheta, a) = q(a|\vartheta)q(\vartheta) \tag{3.25}$$

With the factorization of the generative distribution and the recognition distribution, the variational free energy can be written as

$$F = D_{KL}[q(a|\vartheta)q(\vartheta)||p(s, \vartheta)p(a|\vartheta)p(\vartheta|\vartheta_{t-1}, a_{t-1})] \tag{3.26}$$

[31]. The expression can further be written out by using properties of logarithms and the definition of the Kullback-Leibler divergence

$$-F = \int q(\vartheta) \ln p(s|\vartheta) + D_{KL}[q(\vartheta)||p(\vartheta|\vartheta, a_{t-1})] + E_{q(\vartheta)}[D_{KL}[q(a|\vartheta)||p(a|\vartheta)]] \tag{3.27}$$

This expression for the variational free energy is used in the deep active inference model. The densities in this expression is approximated by deep neural networks, which is the reason for this method being called deep active inference [31].

First, we look at the approximation of the probabilities $q(\vartheta)$ and $p(s|\vartheta)$. In essence, these two probabilities does the opposite of each other. Where $q(\vartheta)$ maps the hidden states when receiving sensory input, also called observations, the density $p(s|\vartheta)$ maps back to the observations when receiving a hidden state. This is reminiscent of a variational-autoencoder (VAE) and the approximation of the two densities can be modelled as one [25]. A VAE consists of an encoder and a decoder which are modelled by ANNs. The encoder network, given by $q_\theta(\vartheta_t|s_{t-3:t})$, models the density $q(\vartheta)$ and the decoder network $p(s_{t-3:t}|\vartheta)$ models the density $p(s|\vartheta)$. The encoder network takes in the three latest observations and encodes it as a distribution over the latent states. Then, the encoder outputs the parameters of the state sufficient for a multivariate Gaussian, i.e. the mean ϑ_μ and the variance ϑ_Σ. The decoder receives this representation of the state and reconstructs the observations the encoder received [25].

Furthermore, the transition density $p(\vartheta|\vartheta_{t-1}, a_{t-1})$ can be modelled by a neural network as a Gaussian. The neural network outputs the mean and the variance of the Gaussian, while the inputs of the network is the previous state and action. In the calculation of the variational free energy, this density is used in the Kullback-Leibler divergence with the approximate density $q(\vartheta)$. As both densities are assumed to be Gaussian it is possible to compute the divergence between the two densities analytically [31].

This leaves us with the third term of the variational free energy expression. Inside the expected value under the approximate density, the Kullback-Leibler divergence between $q(a|\vartheta)$ and $p(a|\vartheta)$ is the most important term for active inference. The reason is that this is used to approximate the internal action posterior $q(a|\vartheta)$, which is used for action selection. As this density is an internal density, the agent can change it as it wants and has fully control over it [31]. This is not the case for the true action posterior $p(a|\vartheta)$, which cannot be observed directly by the agent. We assume that the agent expect to minimize the expected free energy in the future [25]. This enables us the calculate the true action distribution as a precision-weighted Boltzmann distribution over the expected free energy. This yields the expression

$$p(a|\vartheta) = \sigma(-\gamma G(\vartheta, a)) \tag{3.28}$$

From this equation it is seen that the agent first computes the expected free energy of every action on all possible paths in the future and then chooses an action by sampling an action from the distribution [31]. This expression leads us to the expected free energy. Given the hidden state and the action, expected free energy can be represented as the sum of the expected free energy of the expected paths into the future

$$G(\vartheta, a) = \sum_{t}^{T} G(\vartheta, a) \tag{3.29}$$

By taking the first time step out of the sum, we get an expression for the expected free energy as a sum of the expected free energy of the first time step and the expectation of the expected free energy of future time steps under the recognition density

$$G(\vartheta, a) = G(\vartheta_t, a_t) + E_{q(\vartheta_{t+1}, a_{t+1})}[\sum_{t}^{T} G(\vartheta_{t+1}, a_{t+1})] \tag{3.30}$$

In [31] it is seen that the expected free energy at a single time step can be given by the definition of variational free energy at a single time step as we previously has seen in (3.8)

$$G(\vartheta_t, a_t) = \int q(\vartheta) \ln \frac{q(\vartheta)}{p(\vartheta, s)} d\vartheta = \int q(\vartheta) \ln \frac{q(\vartheta)}{p(\vartheta|s)p(s)} d\vartheta \tag{3.31}$$

As the agent cannot access the true posterior $p(\vartheta|s)$ we use the recognition density instead. By using logarithmic rules we get that

$$G(\vartheta_t, a_t) = -\int q(\vartheta) \ln p(s) d\vartheta + \int q(\vartheta)(\ln q(\vartheta) - \ln q(\vartheta|s)) d\vartheta \tag{3.32a}$$

$$= -\ln p(s) + \int q(\vartheta)(\ln q(\vartheta) - \ln q(\vartheta|s)) d\vartheta \tag{3.32b}$$

$$= -\ln p(s) + D_{KL}(q(\vartheta)||q(\vartheta|s)) \tag{3.32c}$$

The term $-\ln p(s)$ encodes the prior preferences of the outcomes, i.e. this term contains which states the agent prefers to be located in. By using the complete class theorem this term can be replaced by a reward function $-r(s)$ [31]. The complete class theorem states that any behaviour is Bayes-optimal for at least one pair of prior beliefs, which are used in active inference, and cost function, which for example is used in reinforcement learning [21]. Hence, a reward signal can be encoded as a prior because the prior and the reward signal are proportional to each other, $p(s) \propto exp(r(s))$. By using a reward function directly as a prior means that the preferences of the agent is to maximize rewards [31].

The second term of the expected free energy of a single step is often called the epistemic value. The epistemic value is the reduction in uncertainty about the hidden states when a sensory input is received. This quantity makes sure that the agent resolves uncertainty about the hidden states and explores uncertain states. When the Kullback-Leibler divergence is minimized, the prior recognition density is equal to the posterior recognition density after receiving a sensory input. This means that new observations will not change the posterior recognition distribution over future states. Then, the agent is confident about the states and the epistemic values will be the same for all policies [20].

Now, we can write the expected free energy as

$$G(\vartheta, a) = -r(s) + \int q(\vartheta)(\ln q(\vartheta) - \ln q(\vartheta|s))d\vartheta + E_{q(\vartheta_{t+1}, a_{t+1})}[\sum_{t}^{T} G(\vartheta_{t+1}, a_{t+1})] \tag{3.33}$$

To compute this quantity exactly is hard as the expected free energy associated with every one of the possible paths has to be calculated. A solution to this issue is to learn a bootstrapped estimate of this function by using a neural network to sample from such that an amortized inference distribution can be learned [31]. The expected free energy neural network is defined with parameters ψ and returns an estimate G_ψ, which is an estimated expected free energy quantity for a state and action pair [25]. The network is trained with the bootstrapped estimate $\hat{G}$ which is the sum of the variational free energy at the current time step and the approximate the expected free energy of the rest of the path. The expected free energy of the rest of the path is approximated by using the estimated expected free energy of the next time step [31] and we get the expression

$$\hat{G}(\vartheta, a) = -r(s_t) + \int q(\vartheta_t)(\ln q(\vartheta_t) - \ln q(\vartheta_t|s_t))d\vartheta + G_\psi(\vartheta_{t+1}, a_{t+1}) \tag{3.34}$$

The expected free energy network can be optimized through a gradient decent on the loss function

$$L = ||G_\psi(\vartheta, a) - G(\hat{\vartheta}, a)||^2 \tag{3.35}$$

This estimate of the expected free energy is used to calculate the true action distribution $p(a|\vartheta)$ which is used to find $q(a|\vartheta)$. This approximated action model is then used for action selection to find the best policy.

To summarize, the deep active inference agent uses three neural networks to approximate the densities $q(\vartheta|s)$, $p(s|\vartheta)$, $p(\vartheta_t|\vartheta_{t-1}, a_{t-1})$ and $q(a|\vartheta)$. These networks are optimized with gradient descent where the variational free energy, given in (3.27), is the loss function. Also, the agent uses a neural network to approximate the expected free energy which is trained by using a bootstrapped estimate of the expected free energy. Furthermore, the expected free energy is used to calculate $p(a|\vartheta)$ which again is used to find $q(a|\vartheta)$ that is used for action selection.

3.4.2 Sophisticated active inference

Sophisticated active inference is another method for handling large state spaces. In this method planning is done by calculating the expected free energy G recursively. The term "sophisticated" comes from economics and if an agent is sophisticated it has beliefs about its own or another agents belief [17]. This suggests that an sophisticated active inference agent considers beliefs about what it would believe if it performed a certain action. In contrast, most active inference agents can be considered unsophisticated as they consider beliefs about what would actually happen after performing an action [17].

In sophisticated active inference, the expected free energy is calculated over a time horizon $\tau = 1, ...T-1$ where the expression for $\tau = T-1$ is given as the immediate free energy

$$G(a_\tau|\vartheta_\tau) = G(a_{T-1}|\vartheta_{T-1}) = D_{KL}(q(\vartheta_\tau|a_{T-1}, \vartheta_{T-1}||C(\vartheta_T)) \tag{3.36}$$

The expression for $\tau = 1, ..., T-2$ is given as the immediate free energy plus the expected free energy for possible future actions given as

$$G(a_\tau|\vartheta_t) = D_{KL}(q(\vartheta_{\tau+1}|a_\tau, \vartheta_\tau)||C(\vartheta_{\tau+1})) + E_q(G(\text{nextstep})) \tag{3.37}$$

as seen in [38]. The second term in (3.37) is the expected value of the next step and is given by

$$E_q(G(\text{nextstep})) = E_{q(a_{\tau+1}, \vartheta_{\tau+1}|s_\tau, a_\tau)}(G(a_{\tau+1}|s_{\tau+1})) \tag{3.38}$$

In this way, the calculation of the expected free energy of an action a_τ in a state ϑ_τ is a combination of the expected free energy of the action at the current time step plus the average expected free energy over all future actions [38]. After calculating the expected free energy a action distribution $q(a_\tau|\vartheta_t)$ is updated with the formula

$$q(a_\tau|\vartheta) = \sigma(-G(U|\vartheta)) \tag{3.39}$$

where U is the set of possible actions it can take and the softmax function secures that the distribution sums up to one [38]. After training, the action distribution has assigned a probability to each action in every state and it can be used for action selection. By taking the action assigned the maximum probability in each state a policy can be found moving the agent from the starting state to the goal. To summarize, the expected free energy can be calculated by using the fact that it has already calculated the expected free energy of future actions such that it does not need to calculate the expected free energy for every policy. The expected free energy is further used to find an action distribution which is used for action selection and finding a policy.

Chapter 4

Implementation and method

In this chapter, a presentation of the implementation of the marine vessel and the environments used for simulations and training are made. The task of this thesis is to use a deep Q-network and deep active inference to perform high-level action planning for the marine vessel and we also give a thorough description of details about the implementations of these algorithms.

4.1 Problem description

In this thesis we will look at the problem of performing a docking operation of a marine vessel with active inference and deep Q-learning. When performing the docking operation, the marine vessel has to be able to navigate from an open area to the harbour, through obstacles in the port area and to a docking position. This requires being able to adapt to dynamic environments, avoid obstacles and learning efficient policies. In recent years, it has been seen that reinforcement learning can be utilized for solving the docking problem and adding autonomy for marine vessels and we wish to see if other parts of artificial intelligence can be used for solving this problem.

This thesis aims to explore if active inference can be used for solving the docking problem. To perform active inference we use a deep active inference algorithm where active inference is combined with deep neural networks. First, to examine its potential as an high-level action planner the deep active inference algorithm will be implemented on a discrete environment. The discrete environment used in this thesis is an attempt to model a part of the Trondheim harbour. To assess the performance of deep active inference we also implement a DQN algorithm for comparison. After training the algorithms we connect the obtained results with guidance and control of a marine vessel in order to see how the complete system works. To perform the complete system, we simulate the dynamics of the marine

vessel with the use of Dynamic positioning and integral LOS after training of the algorithms.

Secondly, we aim to train the active inference and reinforcement learning on a marine vessel with continuous states and actions. This is in order to use active inference and reinforcement learning as high-level action planners in combination with being used for low-level guidance and control of marine vessels. In this case, the agents are implemented with the marine vessel dynamics and the action space is the direct inputs to the vessel. By performing this, we get to see the real potential of using active inference for solving the docking problem as this is a more realistic case. We also can assess how active inference can be used for guidance and control and if it can add autonomy to marine vessels.

4.2 Tools

In this thesis, the Python programming language was used for implementation of the algorithms and models. In addition, the numerical library NumPy was used for numerical operations, while Matplotlib was used for making visualizations of results. Also, the OpenAI Gym library was used for making the custom discrete environment used for training of both the DQN-agent and deep active inference-agent [36].

The open source machine learning framework PyTorch was used for implementations of deep neural networks. PyTorch provides tensor computations and deep neural networks which makes implementation of neural network architecture easy for the user. Building blocks such as network models, automatic gradient differentiation, backpropagation and optimizers are already implemented in order to build and train deep learning models [37].

4.3 Marine vessel

The model of the marine vessel used in this thesis is based on the vessel model from [29]. To summarize, the vessel has three thrusters. Two azimuth thrusters at the back of the vessel and one tunnel thruster in the front as seen in Figure 2.2. The positioning of the thrusters and their angle in the body frame can be seen in Table 4.1.

The vessel model is given by

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\psi)\boldsymbol{\nu} \tag{4.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{4.2}$$

Thruster	x-position	y-position	Angle
Azimuth left	$l_{x_1} = -35m$	$l_{y_1} = -7m$	α_1
Azimuth right	$l_{x_2} = -35m$	$l_{y_2} = 7m$	α_2
Tunnel	$l_{x_3} = 35m$	$l_{y_3} = 0m$	$\pi/2$

Table 4.1: Position and angle of the thrusters in relation to the centre of the vessel.

Further explanations about the kinematics used for simulation of the movement of the vessel are defined in Section 2.1.1. When it comes to the thrust configuration, the force vector is found by the equation

$$\tau = \boldsymbol{Bu} \tag{4.3}$$

where the thrust configurations can be seen in Section 4.4. To make the behaviour of the thrusters more realistic, constraints on the maximum force are applied. The maximum thrust from the azimuth thrusters are set to 1/30 of the dry ship weight, while the maximum thrust from the tunnel thruster is set to $\pm 1/60$ of the dry ships weight. In addition, constraints on the angle of the azimuth thrusters are applied. To make sure that the two azimuth thrusters cannot produce forces that work against each other a 20 degree forbidden sector is added for both thrusters in the direction of the other. [29].

In the vessel model there are two matrices that needs to be defined, the inertia matrix and the dampening matrix. The inertia matrix is given by

$$\boldsymbol{M} = m\boldsymbol{N}\boldsymbol{M}_{bis}\boldsymbol{N} \tag{4.4}$$

where m is the mass of the vessel which is given as $m = 6000e3(kg)$ for this vessel. The normalization matrix $\boldsymbol{N}$ is given by

$$\boldsymbol{N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & L \end{bmatrix}$$

where $L = 76.2(m)$ is the original length of the vessel but for simplicity we set the length to $L = 75(m)$. In the middle of the two normalization matrices there is a matrix M_{bis}. This is called a non-dimensional matrix, which means that the units involving physical quantities are removed and substituted by suitable variables [12]. Here, the Bis system is used which yields the matrix

$$\boldsymbol{M}_{bis} = \begin{bmatrix} 1.1274 & 0 & 0 \\ 0 & 1.8902 & -0.0744 \\ 0 & -0.0744 & 0.1278 \end{bmatrix}$$

The dampening matrix $\boldsymbol{D}$ is given by

$$\boldsymbol{D} = m\sqrt{\frac{g}{L}}\boldsymbol{N}\boldsymbol{D}_{bis}\boldsymbol{N} \tag{4.5}$$

where $g = 9.8(m/s^2)$ and the non-dimensional matrix $\boldsymbol{D}_{bis}$ is given by

$$\boldsymbol{D}_{bis} = \begin{bmatrix} 0.0358 & 0 & 0 \\ 0 & 0.1183 & -0.0124 \\ 0 & -0.0041 & 0.0308 \end{bmatrix}$$

```
custom_map = np.array([
        [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 0
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 1
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 2
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 3
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 4
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 5
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 6
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 7
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], # 8
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1], # 9
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1], # 10
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1], # 11
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1], # 12
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1], # 13
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1], # 14
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1], # 15
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1], # 16
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1], # 17
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1], # 18
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,0,0,0], # 19
        [0,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1,1,0,0,0,1,1,1], # 20
        [0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,0,0,0,0,0,1,1], # 21
        [0,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0,1], # 22
        [0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,0,0,1,1,3,0,0,1], # 23
        [0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0,1], # 24
        [0,0,0,0,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1], # 25
        [0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,0,0,1,0,0,0,1,1], # 26
        [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,0,1], # 27
        [0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1], # 28
        [0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]  # 29
      # 0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9
```

Figure 4.1: Visualization of the environment in matrix form. The starting state is represented by 2 in the upper left corner and the end state is represented by 3 towards the bottom right corner.

4.3.1 Discrete docking environment

The simulation environment is an attempt to model a part of the Trondheim harbour. We create a discretized model of the harbour by making a grid of 30x30. The matrix that represents the discretized environment can be seen in Figure 4.1, where we define each element in the grid with both a height and width of 50

meters. In this environment the starting point of the marine vessel is set to be in the upper left corner represented by the number 2 and the goal state is set to the bottom right corner represented by the number 3.

Another representation of the environment can be seen in 4.2 where the blue area is the area of obstacles. This is the representation we use when simulating the marine vessel. The red marker in the upper left corner is the start position, while the red marker in the lower right corner is the goal position. We assume that the coordinate system is a North-East-Down frame. This means that the x-coordinates are given on the vertical axis and the y-coordinates on the horizontal axis. We define the area of open states inside the obstacles as the port area. Also, we define the area where the vessel can enter through the obstacles as the port entrance, while the goal position is called the docking position.

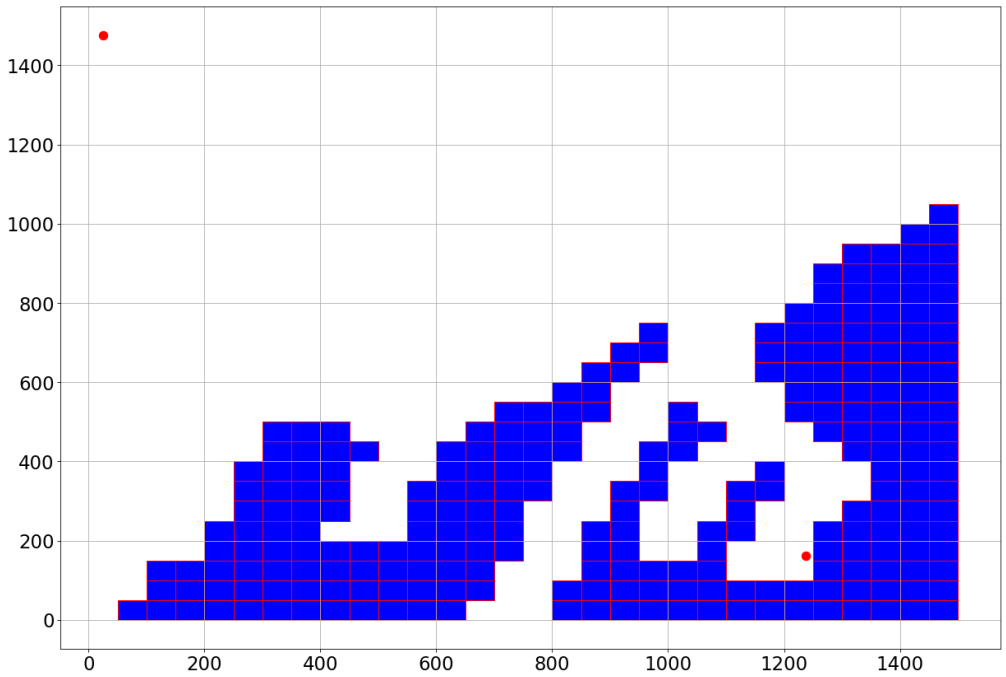


Figure 4.2: Visualization of the environment. The starting state is represented by the red dot in the upper left corner and the docking position is represented by the red dot in the bottom right corner.

We want the vessel to dock in a certain position which means that in addition to having a Cartesian goal state, a goal state for the heading also have to be defined. We define this as the desired heading and set it upwards to the north. This means that the vessel will have to turn as it enters the docking area pointing towards the south.

When the marine vessel is moving through the environment to the docking position it is not necessary to define the heading of the vessel as we can control it to what suits best. However, as we wish the marine vessel to reach the docking position in

a specific heading we also want to include the heading when finding a path with DQN and deep active inference. Therefore, we split the planning task in two. One part which moves the vessel from the starting point to the docking area, which we call Part 1, and another which performs the docking operation, called Part 2. In Part 1 we do not care about the desired heading and control the heading with integral LOS. Here, we only care about taking the vessel safely to the area we call the docking area in an optimal manner. In Figure 4.3, the red marker visualizes the goal position of Part 1, while the open area below this marker is considered the docking area.

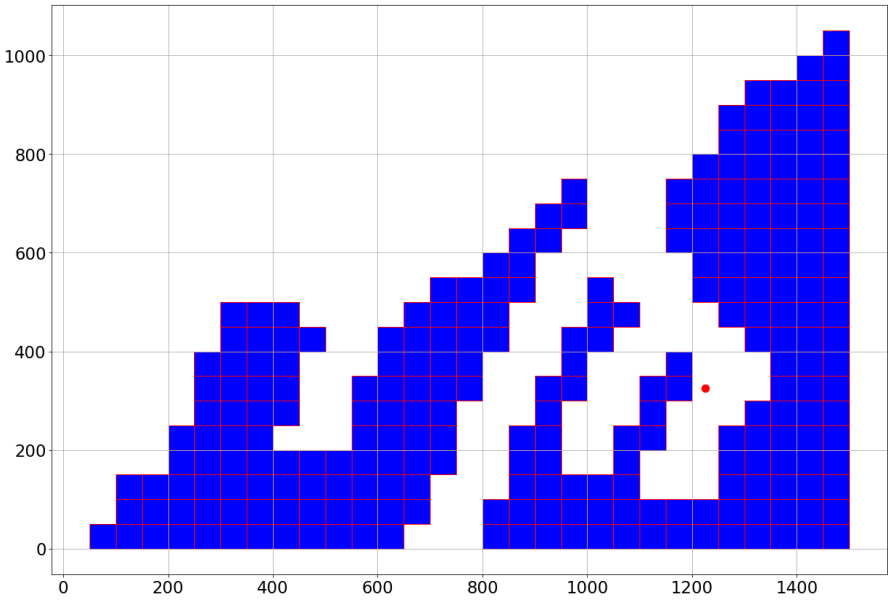


Figure 4.3: Red marker visualizes the goal position of Part 1.

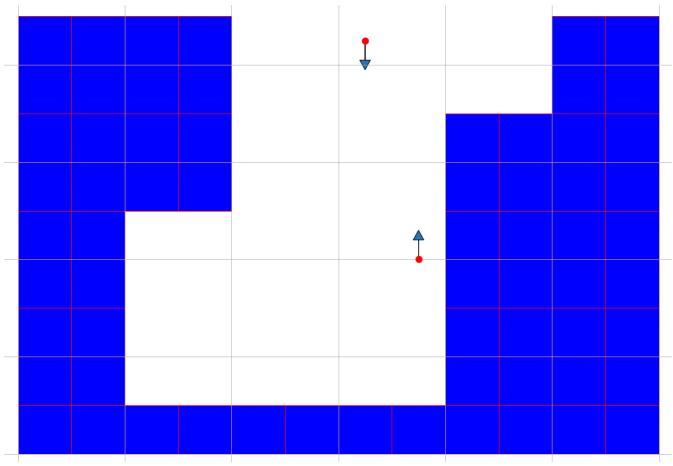


Figure 4.4: Top red marker represents the start position of the vessel and the bottom red marker represents the goal position in Part 2. The arrows visualizes the heading of the vessel.

In Figure 4.4 we see the docking area, which we label Part 2. Here, the start and goal position is represented by both a Cartesian position and a heading where the arrow is pointing in the direction of the heading.

The docking area is also represented as a matrix when calculating the path with DQN. This matrix is set up in the same way as the matrix in Figure 4.1 except that each matrix entry is assumed to be 25×25 meters. This gives a more detailed environment that is required for finding the policy that moves the agent safely to the goal.

4.4 Guidance and Control

The guidance of the vessel is based on waypoints from a path calculated by utilizing the policy found by the DQN-agent and the deep active inference-agent. By using consecutive waypoints a straight line can be formed for the vessel to follow. When simulating the dynamics of the marine vessel we used discrete time steps with a step size of 0.02. At each iteration the desired point is moved along this line such that the vessel gets closer to the waypoint we want the vessel to move to. When the marine vessel is inside a radius of 3 meters of the waypoint and when the error between the heading and the desired heading is less than 0.01, the next waypoint in the path is used to make the next line. This process is repeated until the vessel hits the goal position or if the number of steps of 12000 is reached. In this way the marine vessel follows the Cartesian coordinates. In addition to guiding of the position of the vessel we need to guide the heading of the marine vessel such that the vessel moves with the correct orientation. The desired heading is calculated with the help of integral LOS if the desired heading it not provided in combination with the path. First LOS was used, but when the vessel moved inside the port area it struggled to compute the correct desired heading. We saw that when the velocity approached 0, the crab angle increased. This is due to the velocity U used in the denominator of the expression for the crab angle. When the crab angle gets large we get an undesired heading which does not represent the heading we want. As integral LOS is an alternative to compensate for the drift term β_c, we tried this solution instead. By using integral LOS this problem was omitted. When using integral LOS we need to set the lookahead distance and the constant κ, which is used in the calculation of the integral gain. In this case the lookahead distance is set to three times the length of the vessel and κ is set to 1.

For the control of the vessel a DP controller is used. The controller is given as

$$\boldsymbol{\tau} = \boldsymbol{R}^{\top}(t)\boldsymbol{\tau}_{\text{PID}} \tag{4.6}$$

where $\boldsymbol{\tau}_{\text{PID}}$ is given by the same expression presented in (2.1.4). As we do not consider the wind at this stage, the part of the controller compensating for the wind is not used here. The gains K_p, K_d and K_i are dependent on the natural

frequencies and the relative damping ratios which we set to $\boldsymbol{\Omega}_n = \text{diag}(0.2, 0.2, 0.2)$ and $\boldsymbol{Z} = \text{diag}(0.8, 0.8, 0.8)$. It was also tested to use a MIMO PID controller, but it was a lot harder to tune. We did not manage to get a well tuned MIMO PID controller and ended up using the DP controller.

As we have seen, the thrust vector $\boldsymbol{\tau}$ can also be given by the the formula

$$\boldsymbol{\tau} = \boldsymbol{B}\boldsymbol{u} \tag{4.7}$$

where $\boldsymbol{B} = \boldsymbol{T}(\boldsymbol{\alpha})\boldsymbol{K}$ and we find the control input vector $\boldsymbol{u}$ though decomposing the forces produced by the azimuth thrusters. After calculating the thrust vector $\boldsymbol{\tau}$ with the DP-controller, it is used for simulating the dynamics of the marine vessels. As we decompose the control inputs with the equation

$$\boldsymbol{u}_e = \boldsymbol{K}_e^{-1}\boldsymbol{T}_e^{\dagger}\boldsymbol{\tau} \tag{4.8}$$

we can find the forces and rotations angles of the thrusters with the use of the equations presented in Section 2.1.2. This way of configuring the thrusters gives us two gain matrices to tune. After tuning the matrices we end up with the thrust coefficient matrix $\boldsymbol{K} = \text{diag}(40, 40, 40)$ and the extended thrust coefficient matrix $\boldsymbol{K}_e = \text{diag}(5, 5, 5, 5, 5)$. The tuning of the thrust configuration matrices could probably be tuned better, but gives a performance sufficient to our task.

4.5 Training environment

The DQN and deep active inference algorithms are used for finding a policy that can be used to finding a path of waypoints such that the marine vessel can move safely to the docking position without hitting any obstacles. When training these algorithms we use an environment based on the matrices presented in Section 4.3.1.

When finding a path in the discrete environment we do not take into account of the real marine vessel dynamics to make the training as simple as possible. However, we take the size of the vessel into account as we have to find a path that ensures that the vessel do not hit the obstacles. It is assumed that the vessel is located with its centre in the middle of the cell. As the marine vessel model has a length of 75 meters it is also assumed that it takes up space in the cell in front and behind the vessel. The action set of the agent can be seen in Figure 4.5. By taking an action the agent moves from one cell to an adjacent cell in the direction of the action. After taking an action the agent will point in the direction of the action. However, we wish for the marine vessel to dock with a specific heading. Thus, for Part 2 of the docking problem we increase the action set such that the agent also can turn in place in the same directions as seen in Figure 4.5. This yields an action set of size 16 where it now can both move and turn in the directions as seen in the figure.

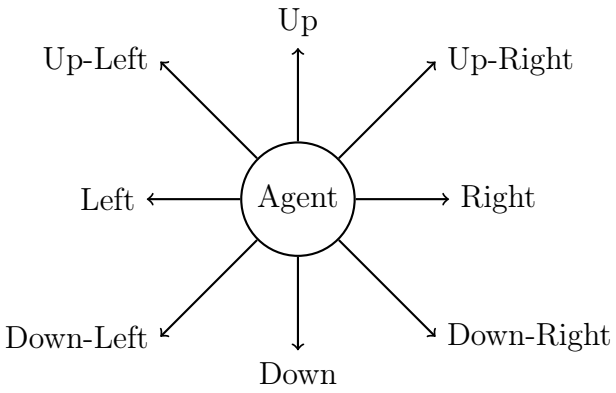


Figure 4.5: Action set of the agent.

When training the algorithms on Part 1 we use the matrix representation of the environment which can be seen in Figure 4.1. This means that we represent the position of the vessel in the same manner as the matrix does. Thus, the position in (x, y)-coordinates corresponds to a row and column position in the matrix. In this way we also represent the obstacles and we check if the agent has hit an obstacle by testing if the (x, y) position of the agent corresponds to an obstacle. The x-coordinates in this environment is represented by the numbers along the vertical axis and the y-coordinates are the numbers along the horizontal axis. The numbers along the y-axis are only the last digit to make it align with the column above. This means that the agent starts in state $(0, 0)$ and ends in $(23, 24)$. We see that the numbers on the x-axis increases downwards in this representation and that the numbers on the x-axis increases upwards in Figure 4.3. The reason for this is that the matrix represents the first row as 0, while in a Cartesian coordinate system uses $(0, 0)$ as the reference point. When applying the path calculated with the help of the matrix in Figure 4.1 on the marine vessel, we adjust the coordinates in the path by flipping the x-axis. Also, as an entry in the matrix represents 50 meters in Figure 4.3 every coordinate in the resulting path is multiplied with 50.

For training the algorithms on Part 2, we use the matrix of the docking environment. Here, the start state is set to $(0, 6)$ and the goal state is set to $(5, 7)$. Also, the heading is set to Down in the start state and Up in the goal state. After calculating the path with the algorithms, the paths are adjusted to fit in the marine vessel environment just as for Part 1. In addition, the resulting path from Part 2 should be added to the path of Part 1. However, this was only done for the DQN-agent as we reached unsatisfying results with the deep active inference-agent.

4.6 Deep Q-network

For training of the DQN-agent we use two different neural networks for the two different tasks. Optimally they should be merged together such that the two parts can be solved together and we can solve the docking task as a whole. This should be possible to do, but for comparison with deep active inference and time constraints this was not done.

The networks are implemented with linear layers and Tanh as the activation function. For input in Part 1 we use the x- and y-coordinate of the vessel to the network, while in the second part we also include the heading of the vessel. The outputs are the Q-values of the possible actions that can be performed in that certain state. We tested different structures in the networks with the number of hidden layers ranging between one and three and with different number of nodes. In Part 1 we ended up with a network of two hidden layers where each layer consisted of 100 nodes. In Part 2 the environment is much smaller, so we used a network with one hidden layer with 64 nodes.

The DQNs are also implemented with target networks and experience replay. The size of the batch used for training is set to 200 and the size of the replay memory is set to 10000. The target networks are updated every 50 iterations and are initialized with the same weights as the main network.

The training of the agents are done with an ϵ-greedy strategy to make sure that the agents explore the environment in the beginning of the training run and gradually exploiting its knowledge. First, we tested the DQN on Part 1 of the docking task. When training the network for this part, the training is split in a few different steps. First, the initial position is set to $(20, 23)$, which is a few steps above the goal position. The reason for setting the initial position this close to the goal is that the inner area is tight and when taking the length of the vessel into account, the agent struggles finding the goal consistently. It was tested if the agent managed to find the goal from outside the port area, but due to the non-convexity of the area inside the port it struggles to find the goal from these positions. With optimal exploration and with unlimited time it would probably have found it, but to speed up the training process we help the agent locating the goal. Thus, by setting the start position closer to the goal, it is easier for the agent to find the goal position. After hitting the goal 50 times, the initial position was set to the port entrance. From this position the agent had to hit the goal 50 times more before the initial position was moved to the original starting position in $(0, 0)$. As the neural network already is trained to move to the goal from the port entrance, the agent knows how which actions to perform when reaching the port entrance from the upper left starting position. The reward function used can be seen in (4.9). When the agent hits the goal it receives a reward of 5, while if it hits an obstacle it will receive a reward of -1. To make sure that the agent moves towards the goal area, a reward of 0.1 is added if it moves closer to the goal in both the x- and y-direction and 0.05

if it moves closer in one of the two directions. Also, if it moves in a direction away from the goal it receives a reward of -0.125. This is specifically helpful when the agent starts at $(0, 0)$ as it only does know how to move inside the port area. Thus, by receiving rewards for moving towards the goal it quickly learns that it should move towards the port entrance.

$$R_{\text{DQN,Part 1}} = \begin{cases} 5 & \text{if goal hit} \\ -1 & \text{if obstacle hit} \\ 0.1 & \text{if closer to goal in both } x\text{- and } y\text{-direction} \\ 0.05 & \text{if closer to goal in either } x\text{- or } y\text{-direction} \\ -0.125 & \text{otherwise} \end{cases} \tag{4.9}$$

When training Part 2 of the docking task we start the training from the start position $(0, 6)$ and wish to move the agent to $(5, 7)$. In addition, the heading is downwards in the start position and upwards in the goal position. The reward function of Part 2 is similar to the reward function of Part 1 and it is seen in (4.10). To make it easier for the agent to learn which heading it should have in the goal position we add an reward of 0.75 which the agent receives when it turns and obtains the desired heading. Also, if it hits the goal position without the correct heading it receives a reward of 0.5 such that it can learn the route to the goal. Now we also check if it moves closer to the the goal in either x- or y-direction as the agent more rarely than in Part 1 will have the opportunity to move closer in both directions as the environment is tighter.

$$R_{\text{DQN, Part 2}} = \begin{cases} 5 & \text{if goal hit with correct heading} \\ 0.5 & \text{if goal hit without correct heading} \\ -1 & \text{if obstacle hit} \\ 0.75 & \text{if desired heading obtained} \\ 0.2 & \text{if closer to goal in either } x\text{- or } y\text{-direction} \\ -0.075 & \text{otherwise} \end{cases} \tag{4.10}$$

After training the networks, we make a path of waypoints to guide the vessel. The trained neural networks are used to find which actions to take when moving in the environment. For example, the agent will take the action with the highest Q-value in the starting point. After performing the action it reaches a new state which is added to the path of waypoints. This is repeated until the agent reaches the goal state or another end state. When reaching the docking area, the desired heading of the vessel is also included in this path.

4.7 Deep active inference

Like the DQN, the deep active inference algorithm is used to calculate waypoints for the marine vessel to follow safely to a docking position. Also, the task is split in a Part 1 and Part 2 such that the docking operation is separated from the operation of moving the marine vessel to the docking area.

The deep active inference algorithm implemented in this thesis is inspired by the implementation of the algorithm in [25] and [31]. The algorithm in [25] is an extension of the algorithm in [31] which gives some differences between the two papers. Thus, we use a combination of both algorithms for the implementation in this thesis. The algorithm from [25] can be found on GitHub at [24] and the calculation of the expected free energy and the variational free energy in this implementation is also used in the implementation in this thesis, but the rest is changed to fit our environment.

The algorithm presented in Section 3.4.1 is mostly based on [31]. The two most important equations used for deep active inference is the equations for the variational free energy and the estimation of the expected free energy. These equations are repeated here

$$-F = \int q(\vartheta) \ln p(s|\vartheta) + D_{KL}[q(\vartheta)||p(\vartheta|\vartheta, a_{t-1})] + E_{q(\vartheta)}[D_{KL}[q(a|\vartheta)||p(a|\vartheta)]] \tag{4.11}$$

$$\hat{G}(\vartheta, a) = -r(s_t) + \int q(\vartheta_t)(\ln q(\vartheta_t) - \ln q(\vartheta_t|s_t))d\vartheta + G_\psi(\vartheta_{t+1}, a_{t+1}) \tag{4.12}$$

A difference between the algorithm implemented and the algorithm presented in this section is that we do not need a approximate distribution over the states $q(\vartheta)$. As we assume that the states are directly observable there is no uncertainty about which state we are in. This density is originally used in the equation for variational free energy where the Kullback-Leibler divergence is calculated between this density and the transition density. In this way, we are training the transition network to output the correct state given the previous state and action. As there are no uncertainty about the states, we adjust the transition network to output the next state directly and not a probability distribution over the states. This means that the probability distributions inside the Kullback-Leibler divergence are transformed to states and estimates of states. Therefore, we replace the Kullback-Leibler divergence with the mean squared error (MSE) between the state and its estimate from the transition network, as proposed in [25]. We call this the predicted state error in this thesis. The approximate distributions over the states $q(\vartheta)$is also used in the equation for the expected free energy in the second term which is the calculation of the epistemic value. The epistemic value makes sure that the agent resolve uncertainty about uncertain states and is useful for exploration of the environment. As there is no uncertainty about the states, this term is also

replaced by the MSE between the state and the estimated state obtained by the transition network.

As we only work with fully observable environments there is no need for the VAE-network used for mapping the hidden states given the observations and opposite. Thus, the networks used in our deep active inference algorithm are the transition network $p_\phi(\vartheta|\vartheta_{t-1}, a_{t-1})$, the action network $q_\xi(a|\vartheta)$ and the expected free energy network $G_\psi(\vartheta, a)$. Different sizes of the networks were tested, ranging from one to four hidden layers and 32 to 500 nodes in each layer. In the end, the networks were all implemented with two hidden linear layers with 100 nodes each. As activation functions we tested both Tanh and ReLU, but ReLU was used in the final implementation. We see from the expressions of the networks that all of them takes the states ϑ as input. In Part 1 of the docking task the states are given as the x- and y-coordinate of the centre of the vessel, while in Part 2 we also include the heading in the state vector. The actions the agent can choose between is the same as for the DQN and can be seen in Figure 4.5.

The networks $p_\phi(\vartheta|\vartheta_{t-1}, a_{t-1})$ and $q_\xi(a|\vartheta)$ are trained by minimizing the variational free energy, while the network $G_\psi(\vartheta, a)$ is trained by minimizing the loss between the bootstrapped expected free energy estimate and the value from the expected free energy network. All the networks were trained with gradient descent using the ADAM optimizer. A a learning rate of $1e-3$ was used for the transition and action network, while a learning rate of $1e-4$ was used for the expected free energy-network. As the transition network should output only positive values a ReLU function was used on the output. Also, a Softmax function was used on the output of the policy network as the policy network outputs a probability distribution over the actions for a given state.

When training the networks we use two techniques introduced for the DQNs. Firstly, we use experience replay which stores the previous history of transition. In this case a single transition consists of three consecutive states, the two actions taken between the states, the reward received in the second of the three states and if the agent has reached the goal or an obstacle in the third state. The reason for using three consecutive states is that when calculating the bootstrapped expected free energy estimate for a state we use the estimate expected free energy value of the next state. Also, when calculating the MSE between the state and the state estimate from the transition network, we need the previous state and action as the input for the transition model. Secondly, we use a target network for the expected free energy. As the expected free energy estimate is used when calculating the bootstrapped expected free energy estimate we need a target network to stabilize the calculation.

Several different approaches to training of the deep active inference agent on Part 1 was tested. In the first approaches, the deep active inference agent was trained in the same manner as the DQN-agent. For action selection an ϵ-greedy strategy was used and the initial position was first set to the port entrance such that the

agent could find the goal position easier. Even with an initial position at the port entrance the agent struggled to hit the goal consistently. Thus, the initial position was moved closer to the goal into the port area until the agent managed to hit the goal position consistently. The starting position that first made the agent hit the goal consistently was $(20, 23)$. Thus, Part 1 of the training was split up in several parts where the initial position first was set close to the goal and then some states further out when the agent had hit the goal a given number of times from the previous position. As the deep active inference-agent uses longer time than the DQN-agent to hit the goal consistently from the various positions it was difficult to use an ϵ-greedy strategy for action selection. When the agent is moved to a new initial position further from the goal the ϵ should be closer to 1 as the agent has to explore a new part of the area. Also, to hit the goal more often from the positions closer to the goal, the ϵ should be closer to 0. This made it hard to find a good balance between exploration and exploitation. From now on, instead of training the agent with the same approach as the DQN-agent, agent was trained in the same manner as proposed in [31].

In theory the second term in the expected free energy formulation seen in (3.33),the epistemic value, can be utilized for exploration. Even though we have changed the expression for the epistemic value from a Kullback-Leibler divergence between the state distribution and the transition network to a MSE, the expression should still incentivize visiting states where the transition model is weak and resolve ambiguity. From now on we perform the action selection with the policy network. By inputting the state the agent is located in, the policy network outputs a probability distribution over the actions. From this probability distribution an action is sampled and performed by the agent. This gives the agent a balance between exploration and exploitation. When the agent comes to an state where it has no experience the probability distribution outputted from the policy network will be approximately uniform. The more the agent learns about the actions in the state, the higher will the probability is assigned to the best action or actions. Thus, the probability of selecting the best action gets higher.

However, we saw early that the deep active inference struggled with exploring the environment and finding the goal. When we tried to let it explore the environment itself, it often explored the environment for $200 - 300$ episodes before it converged to a path that lead it to the same state again and again. Often this state was the closest obstacle, even if we assigned it a large negative reward. As this did not work we again tested different approaches of starting with different initial positions in Part 1. First, initial positions were randomly chosen from the whole environment. The reason for this is that the predicted state error often is large in the beginning of the run as the transition network does not have enough experience to make good predictions. If we set the initial position first inside the port area, such that the agent could find the goal more easily, the MSE between the state and the transition network would be large as the transition network is not trained sufficiently yet. Thus, the bootstrapped expected free energy estimate and the variational free

energy value would be very large and not give a correct representation of the state and action values. We occasionally found this to be problematic as it seemed as this could be the cause of an explosion in the expected free energy.

The training method that yielded best performance was to separate the inside of the port and the outside of the port, after exploring the environment to decrease the predicted state error. First, the agent was trained on the environment inside the port where we "closed" the port entrance such that it had no opportunity to move out in the open area. When training inside the port area we used initial states along the path from the port entrance to the goal and the first initial state was set to $(21, 24)$. This is a state which has obstacles down to the left and up to the right of the state. Even if this is a state close to the goal it was seen that the agent struggled to get to this state when starting further out. After hitting the goal a 100 times from this position the agent was moved out to $(19, 23)$. From here the agent hit the goal another 100 times before it was moved out to the port entrance. The same procedure was repeated from this position before the port entrance was opened and the agent was moved out in the open area. Here the agent was put in different start positions along the path from the start position $(0, 0)$ to the port entrance. Even though this training method gave the best performance, the agent still did not manage to hit the goal from $(0, 0)$ which is the real start position.

The prior preferences of the deep active inference agent is to maximize rewards as we directly encode the prior as a reward function. A lot of different reward functions was tested, with rewards ranging from -100 to 100. Also, both a sparse reward function, which only gave rewards if the agent hit the goal or an obstacle, and a denser reward function, which gave rewards in every step dependent on how far the agent was from the goal, were tested. In the end, the denser reward function gave the best performance and it can be seen in (4.13). When the agent hits the goal it receives a reward of 10 and when it hits an obstacle it receives a reward of -3. In addition, the agent receives a reward of 1 if it moves closer to the goal in both x- and y-direction and 0.75 if it moves closer in one of the directions. A negative reward of -1.5 is given if the agent moves away from the goal.

$$R_{\text{DAI,Part 1}} = \begin{cases} 10 & \text{if goal hit} \\ -3 & \text{if obstacle hit} \\ 1 & \text{if closer to goal in both } x\text{- and } y\text{-direction} \\ 0.75 & \text{if closer to goal in either } x\text{- or } y\text{-direction} \\ -1.5 & \text{otherwise} \end{cases} \tag{4.13}$$

When training the agent on the environment we use the matrix in Figure 4.1 to check if it has moved to a valid state or not. To make sure that the marine vessel will not crash when using the path made by the deep active inference agent we consider the size of the vessel. This means that we assign the agent three states in a row when it moves in the environment. As this makes the training

unnecessary complicated and as we struggled with hitting the goal in the original environment, we tried to implement an environment of 20×20 with states of size 75×75 meters to fit the marine vessel-agent inside one state. However, this made no significant changes in the performance and the agent struggled with finding the optimal actions from states located as close as $4 - 5$ states away from the goal. In the end we had to disregard the size of the marine vessel and use the 30×30 environment. This meant that we stopped checking for crashes in the cell in front and behind the centre cell. Instead we tried to give the agent a reward of -1 every time it was in a state next to an obstacle to make it find a path away from the obstacles. However, this did not have a significant impact on the performance.

When performing the docking operation in Part 2 we made small changes to the environment. Just as for the docking operation for the DQN we now have a state vector with the heading in addition to the Cartesian coordinates of the agent. Also, the action set is now increased to 16 actions such that the agent also can change its heading. This means that in addition to have a goal state consisting of coordinates we also specify a desired heading. The reward function used in Part 2 is seen in (4.14). While the agent still receives a reward of 10 for hitting the goal, it now receives a reward of -5 for hitting an obstacle. When the agent moves away from the goal a reward of -1 is received. In addition, the agent receives a reward of 0.99 both if it moves closer to the goal in both x- and y-direction, and if it changes the heading to the desired heading. This is set to a value lower than the value for moving away from the goal to make sure that it cannot move back and forth to accumulate a positive sum of reward.

$$R_{\text{DAI,Part 2}} = \begin{cases} 10 & \text{if goal hit} \\ -5 & \text{if obstacle hit} \\ 0.99 & \text{if closer to goal in both } x\text{- and } y\text{-direction} \\ 0.99 & \text{if desired heading obtained} \\ -1 & \text{otherwise} \end{cases} \tag{4.14}$$

The initial Cartesian position of the agent in training is first set in the middle of the environment quite close to the goal. As the agent now has 16 actions to choose from in each state, it was thought that it was important that it started a bit closer to the goal that the actual initial position. However, the initial heading is always set downwards. When the agent managed to hit the goal consistently it was moved to the initial position $(0, 6)$ in Figure 4.4. Now, the agent did not have any problems hitting the goal consistently as it knew what actions it should perform around the goal.

It was also the intention to perform both Part 1 and Part 2 with the same network to test it on the whole task. In addition, we planned to test both deep active inference and DDPG on the continuous case with continuous states and actions. As we did not manage to get the results we wanted in the discrete case we unfortunately never got to look at these parts of the problem.

Chapter 5

Results

This chapter presents the results obtained with the DQN algorithm and the deep active inference algorithm when performing the docking task. First, the results from the DQN-agent in the discrete environment are presented and discussed, before we present the results from the deep active inference-agent in the same environment.

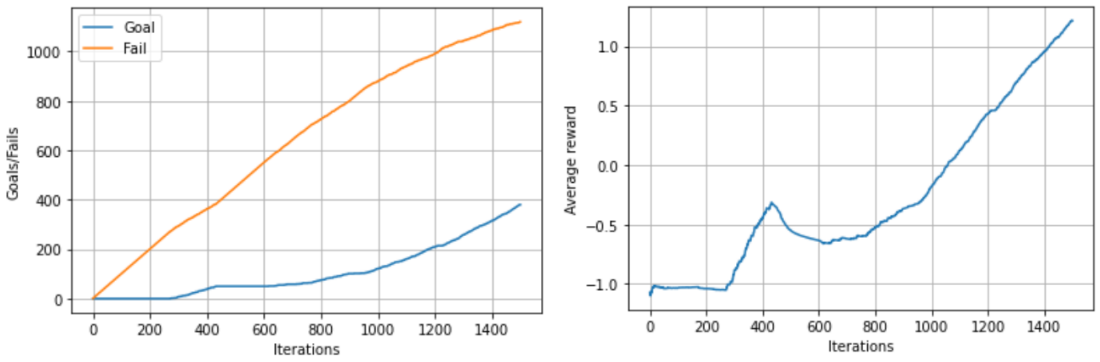
5.1 Docking in discrete environment

5.1.1 Deep Q-Network

Part 1

In Part 1 of the docking task the DQN-agent struggles with finding the goal consistently. From the plots in Figure 5.1 we see that the agent does not hit the goal until around 300 iterations from the closest start position. Even though the agent is in close proximity to the goal it has problems with locating the goal as it finds itself in a tight area. However, when the agent first has found the goal it hits it consistent from the closest start position. This can be seen as both the goal count and the average reward start to increase significantly. As the agent still explores the environment it does not hit the goal in every run. After just over 400 episodes the agent has hit the goal 50 times and the initial position is moved to the port entrance. From this initial position the agent uses around 200 episodes before it hits the goal again. From this point in the run, the average reward and the goal count increase slowly, which is due to that the agent only hits the goal occasionally as it still explores the environment. After around 900 iterations it is seen that there is a short stop in the goal increase. Here, the initial position is moved to the original start position. From the two figures in Figure 5.1 we see that both the goal count

and average reward increase steadily from this point on as the agent finds the goal almost immediately from the new start position. In total the agent hits the goal 380 times over 1500 episodes. Even if the agent fails more that it hits the goal in total, it does hit the goal more that it fails at the end of the run. This means that it has found a policy that leads the agent from the start position to the goal.



(a) Visualizes the goal hits versus fails of the DQN-agent in Part 1.

(b) The average reward the DQN-agent received in Part 1.

Figure 5.1

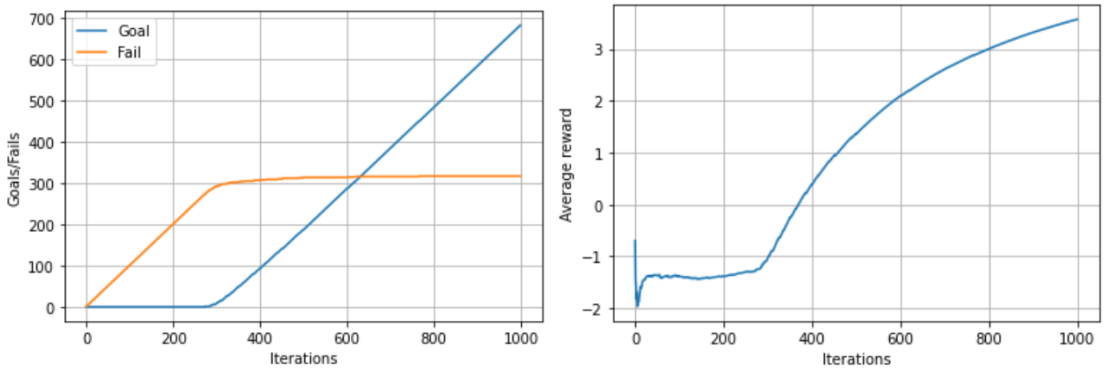
After training the network, we can use the DQN to find a path for the marine vessel to follow. This is done by using the DQN to find the policy that guides the agent from the start position to the goal position and store which states the agent visits on its way. The states the agent visits and the policy used to guide the agent is seen in Table 5.1. This table shows which state the agent is located in at each time step and the action with the highest Q-value, which it will perform. For example, we see in time step 29 that the agent moves down from state $(22, 24)$ and ends in $(23, 24)$ which matches with the goal state seen in Figure 4.1. The agent takes 29 actions to move to the end state which is the least amount of steps it can take between the start state and the goal state when accounting for the marine vessel length. This shows that the agent takes one of the optimal routes to the goal.

Time step, state	Action	Time step, state	Action
1, (0,0)	Down-Right	16, (14,15)	Right
2, (1,1)	Down-Right	17, (14,16)	Right
3, (2,2)	Down-Right	18, (14,17)	Right
4, (3,3)	Down-Right	19, (14,18)	Right
5, (4,4)	Down-Right	20, (14,19)	Right
6, (5,5)	Down-Right	21, (14,20)	Down
7, (6,6)	Down-Right	22, (15,20)	Down-Right
8, (7,7)	Down-Right	23, (16,21)	Down
9, (8,8)	Down-Right	24, (17,21)	Down-Right
10, (9,9)	Down-Right	25, (18,22)	Down
11, (10,10)	Down-Right	26, (19,22)	Down-Right
12, (11,11)	Down-Right	27, (20,23)	Down-Right
13, (12,12)	Down-Right	28, (21,24)	Down
14, (13,13)	Down-Right	29, (22,24)	Down
15, (14,14)	Right	30, (23,24)	-

Table 5.1: Policy that shows how the DQN-agent moves from the start state to the goal state in Part 1 of the docking task.

Part 2

Part 2 of the docking task is solved easily with DQN. As seen from the plots in Figure 5.2, the agent finds the correct position and heading at around 300 iterations. In the rest of the run the agent hits the goal in almost every episode with some exceptions where it explores the environment. The average reward seen in Figure 5.2b is on its way to stabilize around an average of just over 5 as the agent receives a reward of 5 of hitting the goal and some small rewards for moving towards the goal. This shows that the agent in almost every run hits the goal. In total, the agent hits the goal 683 times out of 1000 episodes.



(a) Visualizes the goal hits versus fails of the DQN-agent on Part 2.

(b) The average reward the DQN-agent receives in Part 2.

Figure 5.2

The policy obtained after training the DQN on the docking area can be seen in Table 5.2. As the cells in this environment is half of the size of the of the 30×30 environment, the cells in the complete environment is split in two such that the position of the cells are defined by either .0 or 0.5 instead of only integers. From this policy we see that the agent moves downwards until it gets close to the goal position. Then, it turns around to get the correct heading before it moves into the goal position.

Time step, state	**Action**
1, (24.0,24.0, Down)	Down
2, (24.5,24.0, Down)	Down
3, (25.0,24.0, Down)	Down
4, (25.5,24.0, Down)	Down
5, (26.0,24.0, Up)	Turn 180°
6, (26.0,24.0, Up)	Down-Right
7, (26.5,24.5, Up)	-

Table 5.2: Policy that shows how the DQN-agent moves from the start state to the goal state in Part 2.

Simulation of the marine vessel

By first adjusting the paths found in Part 1 and Part 2 such that they can be used in the marine vessel environment and then adding them together we get the path seen in Figure 5.3. Each red dot corresponds to a point in the paths seen in Table 5.1 and Table 5.2. By starting in the upper left corner the vessel should move to the goal position without hitting the obstacles. In the docking area the red dots are closer as the size of the discrete states were smaller here than in the rest of the environment.

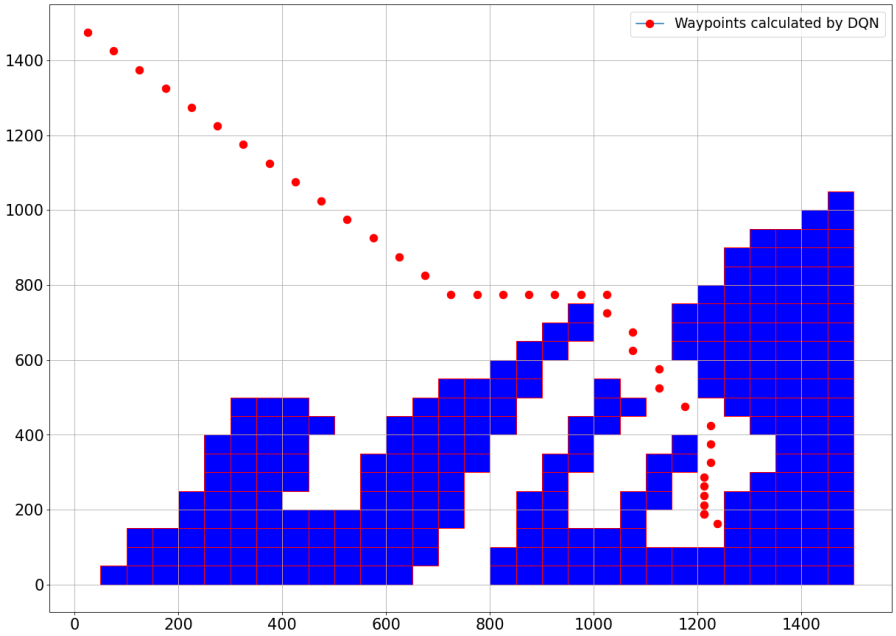


Figure 5.3: Visualization of the path found with the DQN-agent.

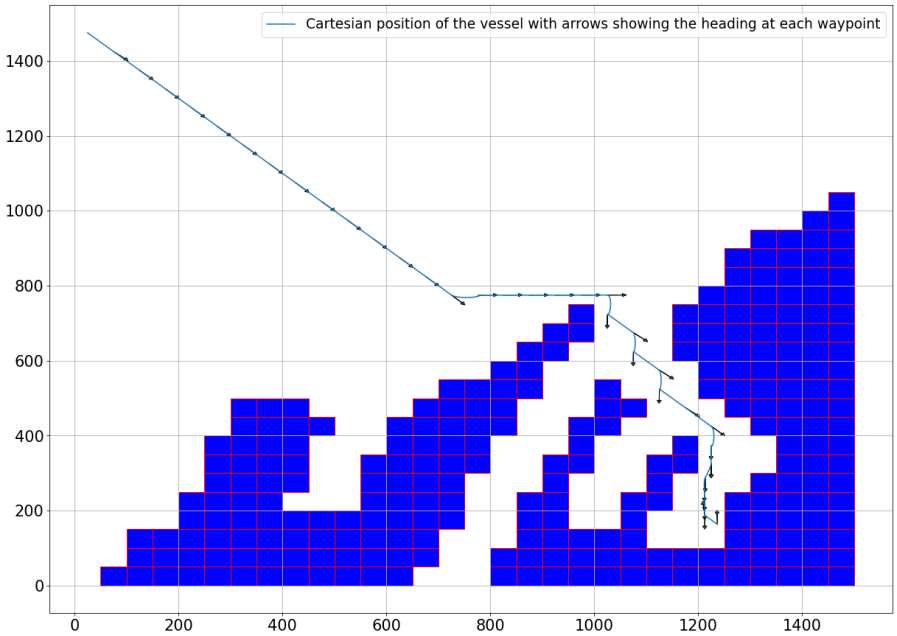


Figure 5.4: Cartesian position of the vessel moving through the environment and its heading at waypoints from the DQN.

When guiding the vessel according to this path we get the behavior of the vessel as seen in Figure 5.4. The blue line represents the position of the vessel while the arrows represent the heading at every waypoint. The desired heading is calculated by integral LOS, expect for when the agent is in the docking area where the desired

heading is calculated by the DQN-agent. It is seen that the marine vessel follows the waypoints well. However, it somewhat struggles when it have to change the heading which causes it not to hit the waypoints exactly. We can see from the figure that the marine vessel in these cases have to move a bit backwards to reach the waypoints. As it is a large vessel it is expected to a degree that it is hard for the vessel to hit the waypoints after a hard turn. However, the parameter matrices of the controller and thrust configuration are probably not optimally tuned, so the performance probably has the potential to be better.

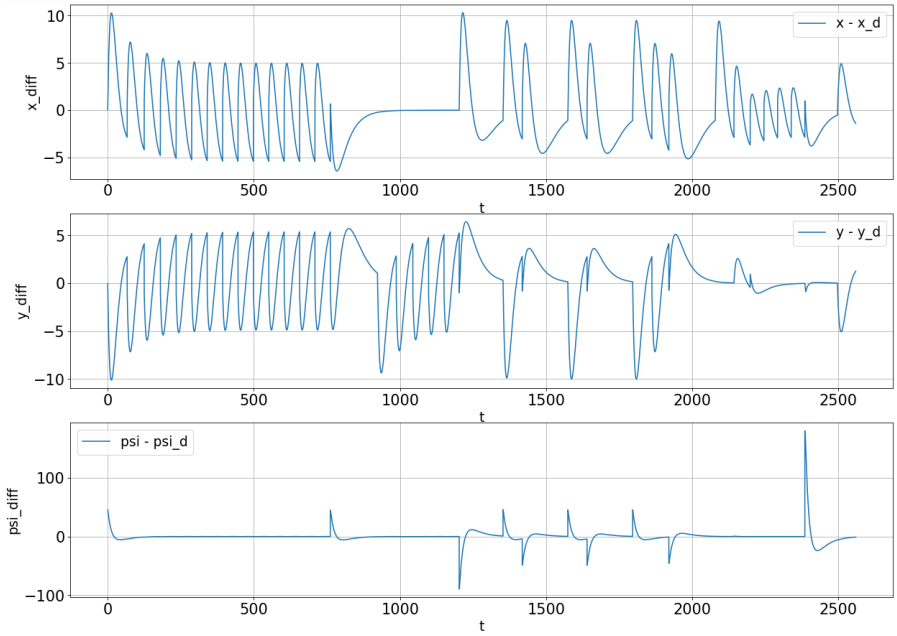


Figure 5.5: The errors between marine vessel position and heading and desired position and heading, with DQN.

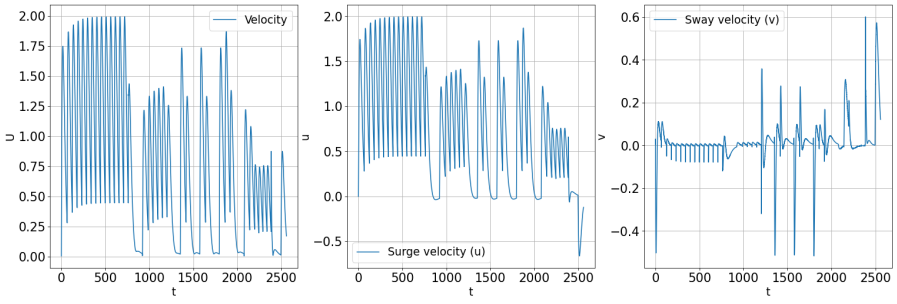


Figure 5.6: The absolute velocity and the velocity decomposed in surge and sway direction, with DQN.

From Figure 5.5 we can see that the vessel manages to get the errors in x, y and ψ close to zero. These errors does not always go to zero as we have defined that the vessel has reached the point if it it inside a radius of 3 meters of the point. It can also be seen that there are large oscillations in x and y. This is as expected because

of how we perform the guidance and control of the marine vessel. When moving through the environment, the desired position is set along a line of consecutive waypoints in front of the marine vessel. This means that the vessel have to catch up to the desired position. When the marine vessel catches up to the desired position, the error moves towards zero. After reaching the current waypoint, the next waypoint is used to calculate desired points along the line between the current and the next waypoint. This will cause a new error in the Cartesian coordinates which leads to some oscillations. In addition, another part of the oscillations comes from that, at some points between two consecutive waypoints, the vessel moves faster than the desired point is updated. In the beginning, the vessel stands still and uses a lot of time to accelerate. At the same time the desired point is moved in front of the vessel at a steady pace which gives a large error in x- and y-direction. In the x-direction this will cause positive errors as seen in the figure. To counteract these large errors, the controller of the vessel will output large forces to the vessel. This causes the vessel to increase its velocity such that it catches up to the desired points which moves along the line with a constant velocity. However, now the vessel are moving faster than the desired points are updated and it will move past the desired points, which in the case of x-direction will cause negative error. Then, the vessel must decrease its speed which in turn explains the oscillations in the velocity in Figure 5.6. In this figure it is seen that the velocity when the vessel moves straight ahead moves towards an absolute velocity of 2.0 before it is decreased to 0.25. When the velocity is decreased, the desired position catch up to the vessel again and in this environment, the vessel will often have reached the waypoint in this case. Now, the velocity is low and the desired points is again moving away from the vessel which causes the vessel to accelerate and which causes it to again move past the desired points. In this way, we get oscillations both in the errors in Cartesian position and the velocities.

When the vessel moves in the docking area, we see that the oscillations are fewer and more in one of the positive or negative direction. The reason for this is that the vessel has to turn in different directions in addition to moving forward. As the vessel turns, it moves slower in the surge direction while the velocity is increased in the sway direction. Also, the agent often cannot turn fast enough to hit the next point correctly which makes it move very slowly around the next waypoint to hit it. Therefore, the agent reaches a velocity of around zero here and does not start with a velocity when it have to accelerate again. This makes the maximum velocity between two waypoints lower and it can slow down more easily. This is seen in the errors in the x-direction after $t = 1000$. Here, the agent moves more slowly and uses more time to get to the points. Thus, the overshoots often lasts longer and the errors are larger.

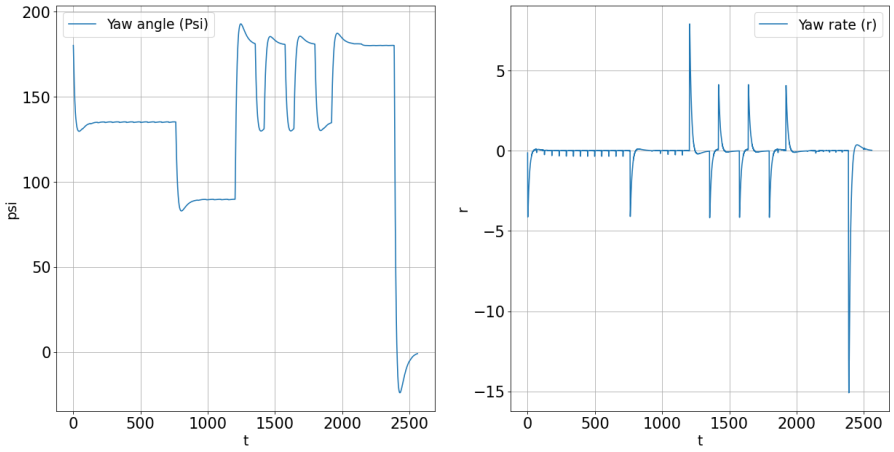


Figure 5.7: The yaw angle and the yaw rate of the vessel, with DQN.

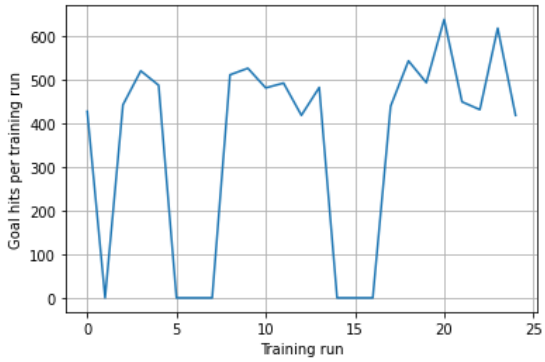
When looking at the errors in heading, we see that the vessel changes its heading relatively well when the desired heading is adjusted. This is also exemplified with Figure 5.7, where the heading, or yaw angle as it also is called, quickly stabilizes. The yaw rate, which describes how fast the yaw angle changes, has small and high spikes which tells us that the heading changes quickly. However, the vessel often overshoots slightly when the vessel approaches the desired heading. Also, we see in the yaw rate that there is small changes in the heading angle when the vessel moves straight forward. The way we perform the guidance and control leads to small changes in heading as it struggles moving straight forward. As this causes small changes in the heading, it also causes small changes in the yaw rate.

From Figure 5.6 it is seen that almost all of the absolute velocity is composed of the surge velocity u and a small amount of sway velocity v until the vessel reaches the areas with more turns. This tells us that the vessel moves almost straight ahead when moving in a straight line with only small corrections, which is what we want the vessel to do. When moving in the port area it is more natural that the sway velocity affects the absolute velocity more. As the vessel does not always hit the waypoint straight away it has to move backwards and sideways to reach it which means that the vessel will have some sway velocity. Also, when performing the docking operation the vessel has to move sideways and the absolute velocity will mainly consist of sway velocity.

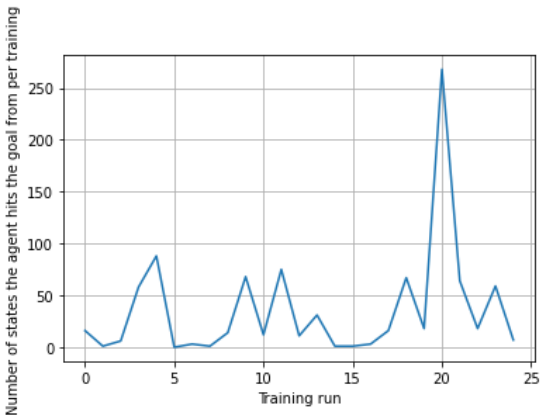
5.1.2 Deep active inference

Part 1

Part 1 of the docking problem is not solved according to the objective of finding a safe path from $(0, 0)$ to $(23, 24)$. It does not manage to find a path that avoids the obstacles at all. The state furthest from the goal from where the agent finds a path that guides the agent to the goal position, is a path going from the state $(0, 5)$. However, this path does not avoid that the marine vessel hits several obstacles on its way to the goal. As we did not receive satisfactory results, we have run the deep active inference algorithm 25 times to see how it performs over several runs. In Figure 5.8a we have plotted how many times the agent hits the goal position per training run. In most of the runs the agent hits the goal around 500 times, but there are also cases where it does not find the goal at all even if it starts close to the goal. In addition to training the algorithms, we tested from which states in the environment we could find a policy that leads the agent to the goal. Figure 5.8b shows that, at best, the agent manages to hit 268 times, but mostly the agent hits the goal from between 0 to 90 states. The most goal hits comes in run number 20, which is also where we manage to find the best policy that guides the agent from $(0, 5)$ to the goal. The agent also hit the goal from $(0, 5)$ in run 3, 4 and 11 where the agent hits the goal around 500 times in training and find a policy that hits the goal from around 75 states.



(a) Results from Part 1 with deep active inference. Visualizes the goal hits when training the agent agent per training run.



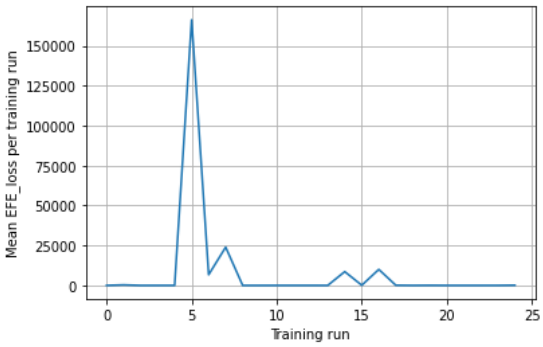
(b) Results from Part 1 with deep active inference. The number of states from which the agent finds a policy that guides the agent to the goal per training run.

Figure 5.8

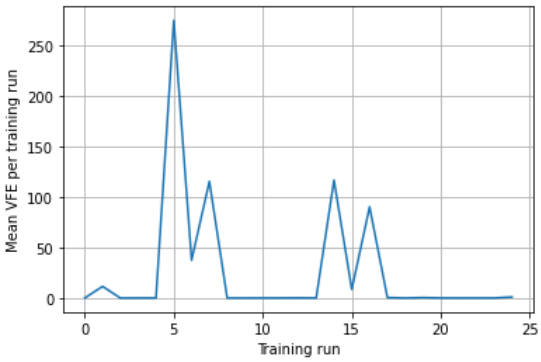
In Figure 5.9 we see different quantities used for training of the networks and for calculation of the expected free energy and variational free energy. Figure 5.9a shows the average expected free energy loss at each training step. As we have a

few high numbers which makes it difficult to get a complete picture of the lower values, these values are also written in Table 5.3. The expected free energy loss is used for training of the expected free energy network. When the average expected free energy loss is high, it tells us that the bootstrapped expected free energy value does not converge and the values of the expected free energy explode. When the numbers are lower, the estimated expected free energy values are more stable and it is more likely that we get a good estimation of the expected free energy.

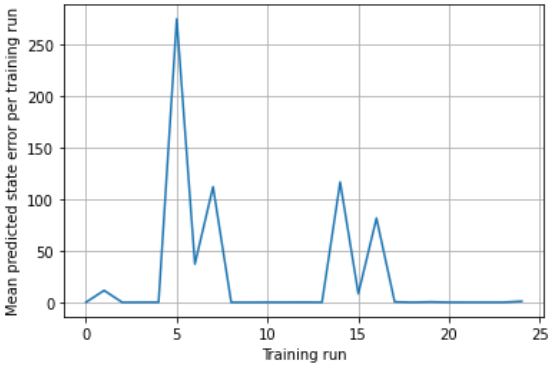
In Figure 5.9b we see the mean variational free energy values, which are used for training of the policy and transition network, and Figure 5.9c shows the predicted state error made by the transition network. We see that these values often corresponds to the expected free energy loss values, albeit much lower. Also, as the predicted state error is used for calculation of the variational free energy, we see that the it almost makes up all of the variational free energy values.



(a) Results from Part 1 with deep active inference. The mean expected free energy loss per training run.



(b) Results from Part 1 with deep active inference. The mean variational free energy per training run.



(c) Results from Part 1 with deep active inference. The mean predicted state error per training run.

Figure 5.9

Training run	Mean EFE loss	Training run	Mean EFE loss
0	13.01	13	13.56
1	342.20	14	8617.10
2	4.04	15	104.23
3	17.36	16	10004.03
4	18.25	17	107.71
5	166243.11	18	8.96
6	6703.82	19	64.93
7	23935.88	20	16.12
8	5.12	21	14.34
9	5.53	22	15.81
10	20.80	23	19.84
11	21.72	24	124.52
12	22.51	-	-

Table 5.3: The mean expected free energy loss at each training run from deep active inference in Part 1.

After training the deep active inference agent 25 times we use the action network $q_\xi(a|\vartheta)$ from run number 20 to find the policy. By performing the action assigned the highest probability in every state we get the policy seen in Table 5.4. From this policy we see that the agent does behave in an optimal manner from the starting point to the goal in terms of number of steps. However, we did not manage to adjust for the length of the marine vessel which means that for it will not manage to guide the marine vessel safely to the goal position.

Time step, state	Action	Time step, state	Action
1, (0,5)	Down-Right	13, (12,17)	Down-Right
2, (1,6)	Down-Right	14, (13,18)	Down-Right
3, (2,7)	Down-Right	15, (14,19)	Down-Right
4, (3,8)	Down-Right	16, (15,20)	Down-Right
5, (4,9)	Down-Right	17, (16,21)	Down-Right
6, (5,10)	Down-Right	18, (17,22)	Down
7, (6,11)	Down-Right	19, (18,22)	Down-Right
8, (7,12)	Down-Right	20, (19,23)	Down
9, (8,13)	Down-Right	21, (20,23)	Down
10, (9,14)	Down-Right	22, (21,23)	Down-Right
11, (10,15)	Down-Right	23, (22,24)	Down
12, (11,16)	Down-Right	24, (23,24)	-

Table 5.4: Policy that shows how the deep active inference-agent moves from the start state to the goal state in Part 1 of the docking task.

The corresponding path can be seen in Figure 5.10. Here, we clearly see that the marine vessel will move close to the obstacles inside the port area. As the centre of the vessel is the point that is used for following the waypoints it will crash into the obstacles.

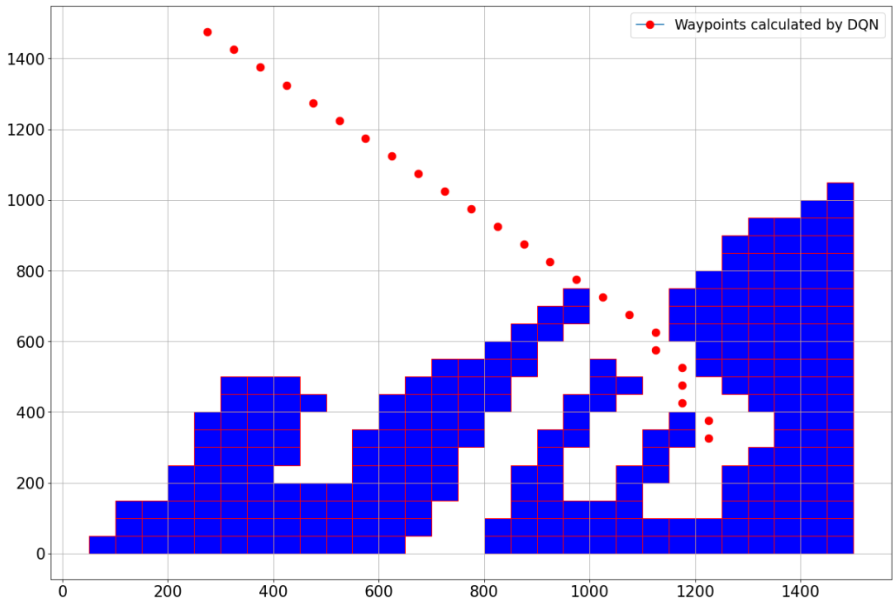


Figure 5.10: Visualization of the path found with the deep active inference-agent on Part 1.

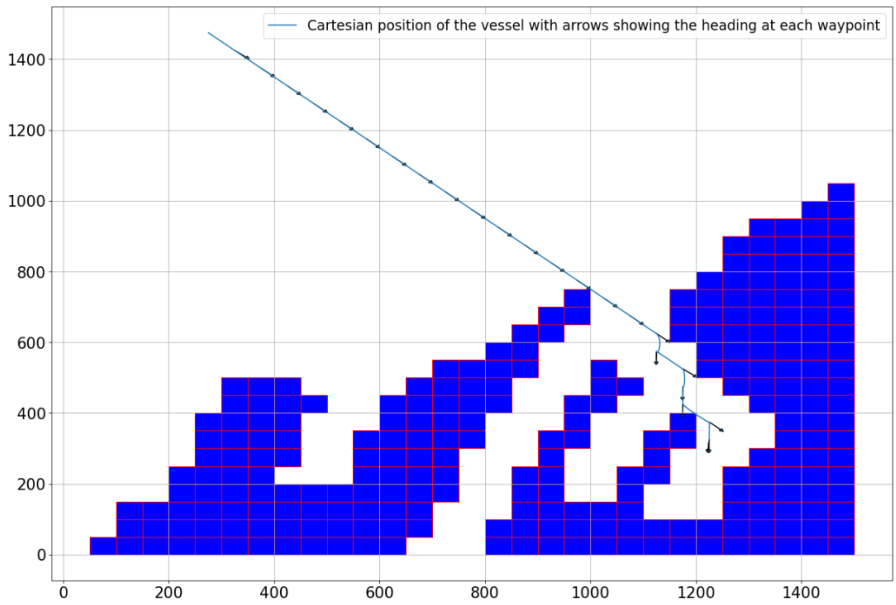


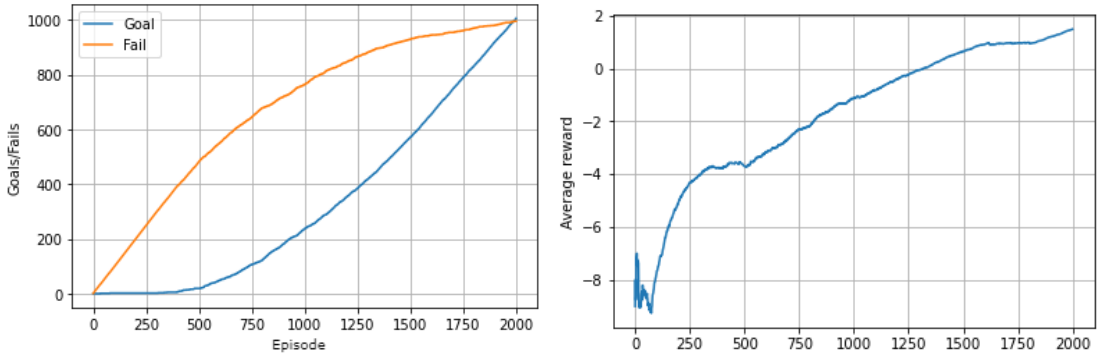
Figure 5.11: Cartesian position of the vessel moving through the environment and its heading at waypoints from the deep active inference-agent on Part 1.

This can also be visualized with the position and the heading of the vessel when

simulating the marine vessel with a controller that follows the waypoints. In Figure 5.11 we see the vessel is moving through the waypoints and crashing at different parts of the environment.

Part 2

Part 2 of the docking task is solved relatively easily. The agent is first placed in close proximity of the goal position, but with the opposite heading, such that the agent finds the goal. We see from Figure 5.12a that the agent uses between 300 and 400 episodes before it finds the goal position. However, we see that the average reward starts to increase before it even hits the goal for the first time. As the agent receives positive rewards of 0.99 for getting closer to the goal and for changing its heading to the desired heading, it can gather positive rewards even if it has not hit the goal. Even though the agent gathers some positive rewards, it still hits an obstacle in the end which gives a reward of -5. In total this gives a reward of around -4 and we see that the average reward stabilizes around this value. Then, when the agent finds the correct goal position it starts hitting the desired position consistently for the rest of the run. Even when the initial position is changed to the actual goal position it continues to hit the goal. In the end of the training run, the agent has hit the goal in 1005 out of the 2000 episodes. This indicates that the agent has found a consistent policy that takes it to the desired position.



(a) Results from Part 2 with deep active inference. Visualizes the goal hits versus fails of the agent.

(b) Results from Part 2 with deep active inference. The average reward the agent receives.

Figure 5.12

In Figure 5.13 we see how the expected free energy loss and the variational free energy develops through the training run. The expected free energy loss starts at a high value as the prediction state error is high, but this quantity decreases when the transition network gets a more correct representation of the environment. The same is the case for the variational free energy quantity seen at the bottom

in the figure. That both the expected free energy loss and the variational free energy moves towards zero as the agent explores the environment shows that the agent finds a good approximation of the different quantities it estimates with the three neural networks. Also, this indicates that the expected free energy network stabilizes at the bootstrapped estimate of the expected free energy and that it can be useful for finding the action distribution $q(a|\vartheta)$.

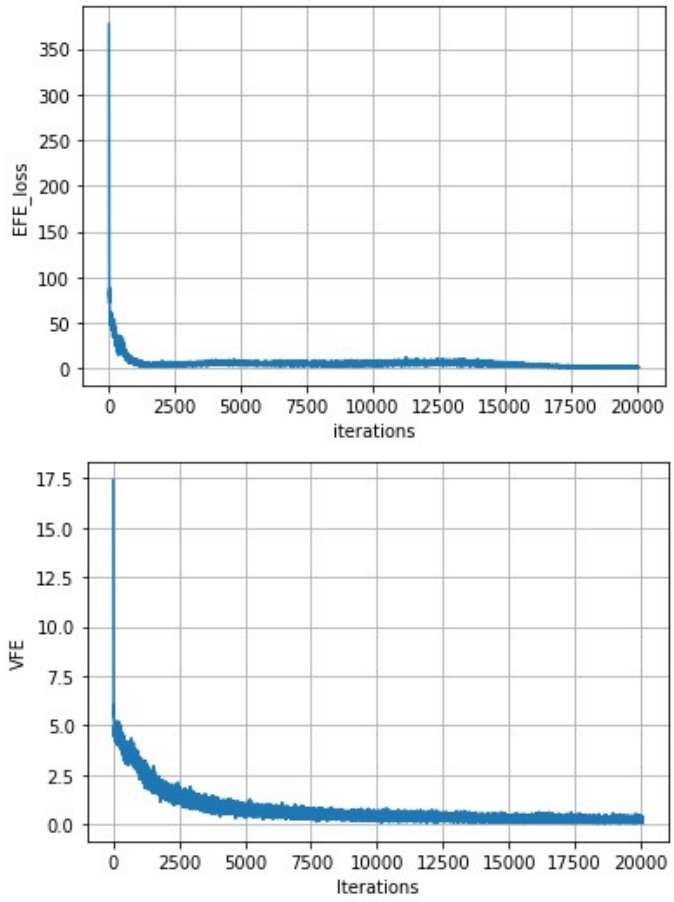


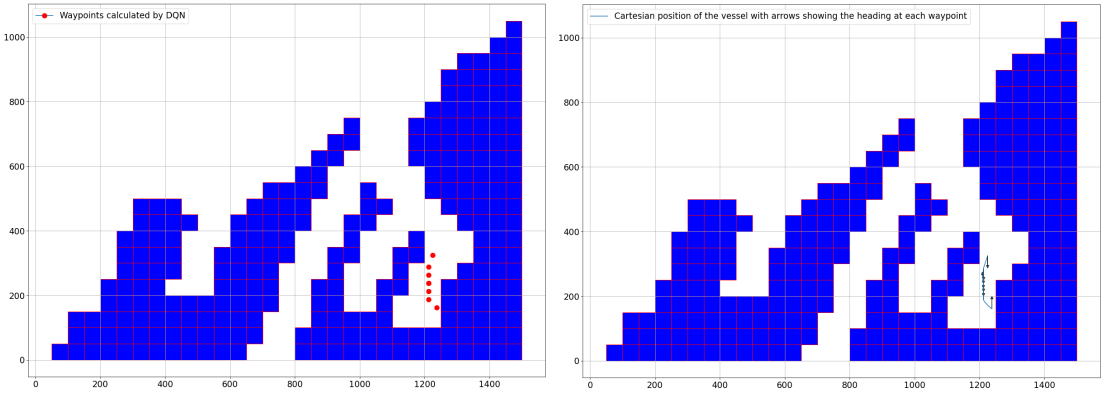
Figure 5.13: The expected free energy loss and the variational free energy in Part 2.

The policy produced by the deep active inference agent is seen in Table 5.5. Firstly, the agent starts with moving downwards. Then, when the agent has space around it to turn, it turns around to obtain the desired heading. Furthermore, it takes one more step downwards before it moves downwards to the right and into the goal position. As the agent uses the least amount of steps it can take, it is optimal in terms of number of steps.

Time step, state	Action
1, (24.0,24.0, Down)	Down
2, (24.5,24.0, Down)	Down
3, (25.0,24.0, Down)	Down
4, (25.5,24.0, Down)	Turn 180°
5, (25.5,24.0, Up)	Down
6, (26.0,24.0, Up)	Down-Right
8, (26.5,24.5, Up)	-

Table 5.5: Policy that shows how the deep active inference agent moves from the start state to the goal state in Part 2.

The waypoints and the movement of the vessel with the path calculated with deep active inference can be seen in Figure 5.14. The vessel follows the path without large problems, avoids every obstacle and moves into the docking position. However, when it turns we can see that it is not pointing straight up and has an offset of a few degrees.



(a) Waypoints calculated with deep active inference in Part 2.

(b) Cartesian position and heading of marine vessel in Part 2.

Figure 5.14

From the figures Figure 5.15 and Figure 5.17 we see that the vessel overshoots, just as we saw when using the DQN-path for guidance, but the vessel manages to hit the waypoints. As the waypoint the marine vessel is following changes when the vessel is inside a radius of 3 meter of the current waypoint, we see that the error in the x- and y-direction does not converge to exactly 0. Here, the overshoots and errors are largest in the x-direction. The reason for this is that the vessel mostly moves straight downwards and does only move in the y-direction when moving out from the starting state or into the goal position. However, it is seen a small change error from around $t = 75$ to $t = 175$ where the vessel should not move in the y-direction. This shows that the vessel does not move in a completely straight line downwards which also can be seen in Figure 5.16 as the sway velocity has a

negative value in this period of time. Between $t = 150$ and $t = 175$ we see small spikes in x-direction and y-direction and a larger spike in the velocity and sway velocity. At this point the vessel turns around and the centre of the vessel changes slightly. As there are only small changes it does not have an effect on if the vessel hits an obstacle or not. The development of the heading of the vessel can be seen in Figure 5.17. When the vessel turns around it is seen that the vessel turns too far such that it overshoots with 25 degrees, which is a relatively large error. In Figure 5.14b it is seen that the heading of the vessel is not completely straight up when it just has turned which probably is due to the overshoot in the heading. Also, we see that it takes a long time before the heading is properly corrected.

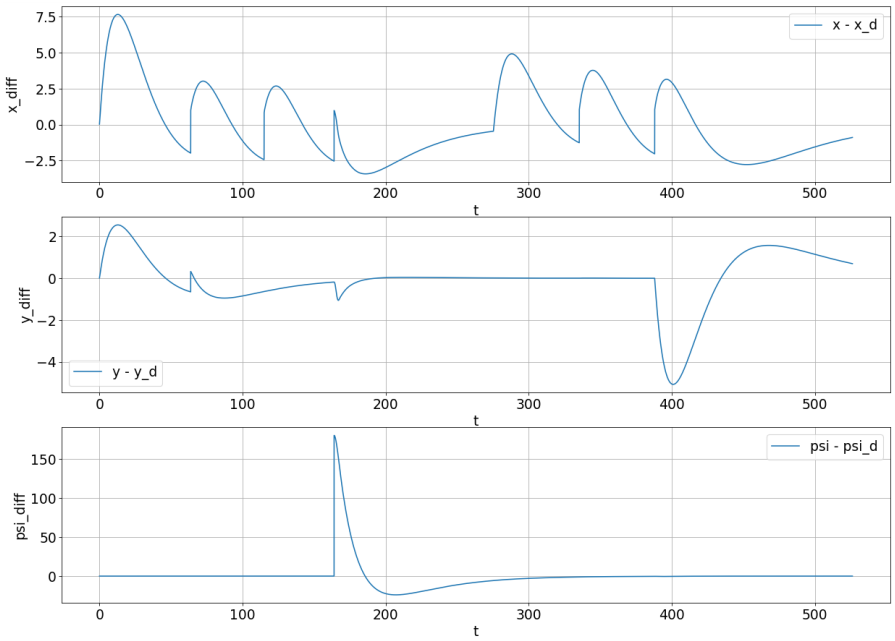


Figure 5.15: The errors between marine vessel position and heading and desired position and heading in Part 2 with deep active inference.

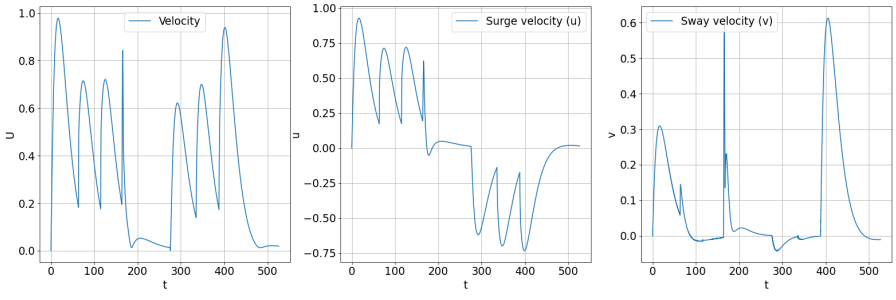


Figure 5.16: The absolute velocity and the velocity decomposed in surge and sway direction in Part 2 with deep active inference.

From Figure 5.16 it is seen that the velocity of the marine vessel mainly is in the surge direction when it moves forward. When the marine vessel moves sideways

or around we see that the marine vessel also has a velocity in the sway direction, which is as expected.

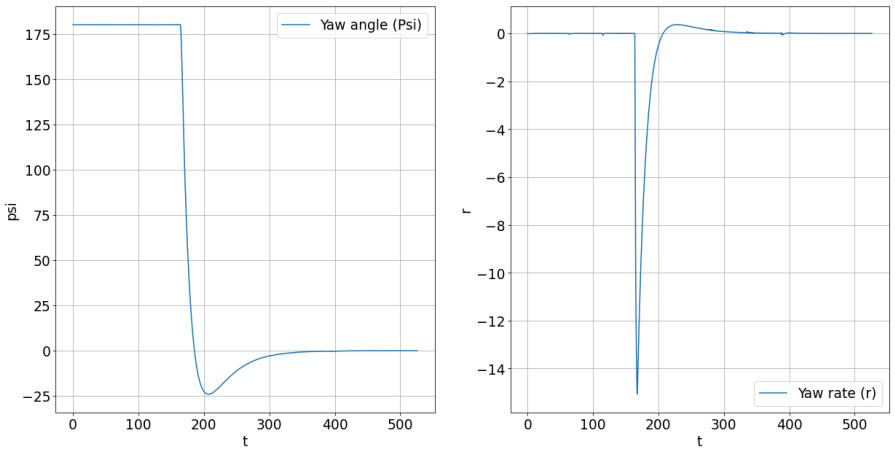


Figure 5.17: The yaw angle and the yaw rate of the vessel in Part 2 with deep active inference.

Chapter 6

Discussion

In this chapter, the results from the simulations are discussed. Also, properties of deep active inference are discussed and compared to reinforcement learning and deep Q-networks. In addition further work is proposed.

6.1 Discussion of results and algorithms

From Figure 5.3 and Figure 5.4 we see that the DQN-agent safely manages to find a path for the marine vessel to follow. The resulting policy, which is seen in Table 5.1, makes an optimal path in the way we defined the environment, but inside the port area we see that the vessel have to zigzag its way through. For a large vessel used in this thesis, it would be favourable for the vessel to move in a more straightforward manner without the changes of direction when it does not have to. Instead of moving back and forth right after it came inside the port area it should instead move a step further down and then move to its own left. This would have have saved the vessel of making two direction changes. A more efficient path could probably have been found by optimizing the reward function to make sure that the agent would not take turns unless it have to. When it comes to Part 2 of the problem, the DQN-agent solves it without any problems. It quite easily finds a good policy for the marine vessel to follow and from the simulation of the marine vessel we see that the marine vessel efficiently moves into the docking position with the actions available. It would have be even better if the action set also included actions where the agent turned and moved in a direction at the same time. This would have made the docking even more efficient.

In Part 1, the DQN-agent somewhat struggles with finding the goal from the first initial position. As the marine vessel do not fit in the cells of the discrete environment we assume that it takes up three cells to make sure that i does not crash. This makes it hard for the agent to find a way through the network as it is

tight and there is few paths through the tightest areas. As the marine vessel is 75 meters and one cell is of length and height 50, the vessel in reality only takes up a quarter of the cell behind and in front of the centre cell. Thus, we have made it unnecessary difficult for our selves by defining the environment in this way as we make the vessel 75 meters longer than it should be. Therefore, the environment should be made simpler where the marine vessel fits in one cell.

From the simulations of the marine vessel we see that the way of guidance and control of the vessel used in this thesis is not very good. We get oscillations in the velocities where the marine vessel have to start and stop frequently. Also, the marine vessel does not manage to follow the path perfectly either, as it sometimes have to stop and move backwards or sideways to get to the waypoint. The vessel also somewhat struggles with moving straight ahead which it should be able to when moving without any disturbances. In addition, there is overshoot in the control of the heading and the position of the vessel. Some of the issues could maybe be removed by optimizing the tuning parameters, but in general the way of following the waypoints should be changed. Especially when the vessel moves straight ahead it should be able to move forward with an even velocity. However, it was not the goal of the task to tune and guide and control the marine vessel model perfectly. Due to the issues with deep active inference and trying to solve these, it was not taken time to optimize the marine vessel model. Even if it the guidance and control is done in a suboptimal manner, it can still give an indication on how good the solutions are.

When looking at the results obtained with the deep active inference-agent we see that it does not manage to solve Part 1 of the docking task completely. It does manage to find the goal from a position from outside the port area, but this does not avoids crashing of the marine vessel. Also, it did not find a path to the goal from the position we consider as the start position which ultimately was the goal of that part of the problem. However, in the open area of Part 2 of the docking task the deep active inference works well. Compared to the policy and resulting path calculated with the DQN-agent, we almost get the same path with from the deep active inference-agent in Part 2. The only difference is when the agent turns around. Where the DQN-agent does the turn just before it moves into the goal position, the deep active inference-agent performs the turning action in the step before. In the way we have defined the environment we only care that the agent does not crash when turning and not when it turns. Hence, there is no significant difference in performance of the algorithms on this part.

By comparing the goal hits versus the fails in training in of the different agents in Part 2, we see that the deep active inference-agent uses more time before it hits the goal consistently than the DQN-agent. This is because the deep active inference-agent learns the environment in a different way where it draws actions from the action network when exploring the environment. As minimizing expected free energy makes sure that the epistemic value is maximized the agent first uses its

time to explore the environment to obtain information. Even though we altered the epistemic value expression from a Kullback-Leibler divergence between two probability distributions to a MSE between two states, the new MSE expression still seems to take care of the exploration and exploitation. When the agent has explored the environment sufficiently, such that it has a good transition model, it exploits what it knows. In Figure 5.13 we see that the variational free energy starts at a high value and then it approaches zero as it receives more information about the environment. The variational free energy value is mostly affected of the predicted state error value as the Kullback-Leibler divergence value between the approximate internal action distribution and the actual action distribution often will be quite low. Thus, in this figure we see that when the agent improves the transition model, the predicted state error gets smaller and the variational free energy also gets smaller. This tells us that the agent explores the environment to make better transition estimates and in this way makes sure to explore the different parts of the environment. In this way the deep active inference agent adjusts the exploration and exploitation dilemma on its own, in contrast to reinforcement learning algorithms which needs an algorithm that ensures a good balance between exploration and exploitation.

Looking at Part 1 of the docking task, it seems like the deep active inference algorithm struggles when it have to work with "two" different environments. At best the agent finds a policy to the goal from $(0, 5)$ where the marine vessel length is not accounted for. When training on Part 1, the agent both have to navigate through the port area with obstacles on both sides and in an open area. When trying to understand why the deep active inference algorithm behaves as it does, it was tested to run the deep active inference agent on a 30×30 area with no obstacles between the start position $(0, 0)$ and the goal. In this case the agent managed to find the goal and a good path in a reasonable amount of time. It was also tested to remove some the obstacles closest around the goal. After around 400 episodes, the agent managed to hit the goal consistently from the start position. Even though the agent found the goal consistently in training, it did not manage to find an optimal policy. However, it did at least produce a policy that would lead the agent to the goal, in contrast to when the agent was run on complete environment. From training on the complete environment we saw that if the deep active inference-agent first was trained with start positions inside the port area it managed to find the goal, but when it was moved outside the port area it did not manage to find the goal again. The same happened when it first was trained to move in the open area and then trained in the port area. As the algorithm works well on Part 2 and when we split up Part 1, a possible explanation for the behaviour of the deep active inference-agent on the complete environment is that the environment is too complex. The agent seems to struggle to get the correct expected free estimations when the environment has different characteristics which leads to suboptimal policies.

To further try to understand why the deep active inference-agent behaves as it

does we ran the algorithm several times. In Figure 5.9 we see the mean expected free energy loss, the mean variational free energy and the predicted state error which are quantities that can help explain why the behaviour of the algorithm. By looking at these figures we see that in some of the training runs, the expected free energy struggles to stabilize. This can be seen by the mean expected free energy loss having high values. Often in these cases, the expected free energy loss starts with relatively low values and then suddenly exploding to values of several thousand. This can be seen closer in Table 5.3. Here, we have a several runs with what we consider low mean expected free energy loss and some runs with a mean expected free energy loss of several thousand, where training run 5 is an extreme of 166243.11. The runs with lower mean expected free energy loss indicates that the expected free energy network manages to get good estimates and converges to stable values. As the loss is small, it does not need to change its estimations too much to output the correct estimations. We also see that we in general obtain the best results when the mean expected free energy loss is low. In Figure 5.8b we see that when the expected free energy loss reaches a mean value of over 100, the agent often does not manage to find a policy that guides the agent to the goal from any of the states, even not the states adjacent to the goal. We also see that these are the training runs where the agent never finds the goal at all when training. This tells us that if the agent does not find the goal in a reasonable amount of time the expected free energy values and the predicted state error will diverge. This makes the expected free energy and the action probability, which is affected by the expected free energy, not representative of the behaviour we want to promote and the agent may converge to a path that leads to an obstacle.

In the cases where both the mean expected free energy loss and the mean variational free energy are low, the network for approximating the expected free energy should be stabilized. As the expected free energy is used for calculating the action distribution, the action network should also stabilize and the agent should be able to use this action network to find optimal policies. Yet, the agent does not manage to find valid policies from the start position to the goal position in Part 1 in these cases. A possible problem might be that the approximation of the expected free energy is to inaccurate to be used for bootstrapping. In the tight areas around the goal position it is important that the expected free energy is well approximated. As we deal with neural networks and a bootstrapped estimate of the expected free energy, the estimate might not be precise enough for the agent to choose the correct actions when it minimizes the expected free energy. This might also make it hard for the agent to propagate the expected free energy values in the port area. The expected free energy of the goal position is just the reward received in this position. If we want the agent to know that it wants to move here from the port entrance, in addition to giving the agent rewards when it moves closer to the goal, the goal reward has to be propagated to the states adjacent to the goal position and further outwards. By looking at the policy in the open area, we see that the agent only moves downwards to the right as this moves the agent towards the goal in which it receives a reward. The state $(0, 5)$ is the state furthest to the left from

where the agent can start from and hit the port entrance while taking the action "Down-Right". This shows that the agent does not know how to move around the obstacles to move into the port entrance and we can see that the reward received at the goal position is not propagated properly out. If the expected free energy estimate is too inaccurate, the propagation might not be performed in a correct manner. Therefore, a possible problem of this algorithm is that the approximations are too inaccurate to estimate the correct expected free energy such that it does not get propagated properly out.

When the mean expected free energy loss has a value under 100 the agent generally finds a policy from more states. However, there is not a complete correlation between a low mean expected free energy loss and finding policies that hit the goal from more states. This can be exemplified with run 0 and run 19. Even though run 0 and run 19 have a mean expected free energy loss of 13.01 and 64.98 respectively, they both find the goal from about the same amount of states. This can be explained by that it takes a lot of runs before the agent finds the goal. When it eventually does, the expected free energy of a certain state and action will have a value that is not affected by the high positive rewards received for reaching the goal. Then, the expected free energy have to converge to new values. This causes an increase in the mean expected free energy loss, which can explain why there is a difference in the mean expected free energy loss, but not in the number of states the agent can hit the goal from when testing the agent. However, when the expected free energy loss is large enough it indicates that the estimate expected free energy values from the network is not good. In general, it is seen that a mean expected free energy loss closer to 0 than 100 gives better results than when it is over 100.

In addition, from Figure 5.8 we see that here is not too much correlation between the number of goal hits in training and the number of states from which the agent finds a policy that leads to the goal. In training runs where the agent hits the goal a similar amount of times, we see a variation in from how many states the agent finds a policy that results in the goal position. By comparing run 20 and 23 which are the runs where the agent hits the goal the most times, we see that the agent hits the goal from over 260 states for run 20 and from just over 50 for run 23. The results from run 23 is actually worse than run 3 and 4 which hits the goal around 500 times in training, as in these two runs, the agent finds the goal from more states. In addition, the agent finds the goal from states further away in run 3 and 4 compared to run 23. From this, we also see that the training is not consistent and relatively unstable. There is a lot of variation between each training run and it seems like it is dependent on what states it visits and when it visits them. In consecutive runs, when the start position is moved to $(19, 22)$, the agent in one run hit the goal immediately as it already had done from $(21, 24)$ and in another run it converged to the obstacle in $(22, 23)$ and it would not hit the goal again.

The reasons for the issues mentioned here are not completely clear. Thus, we used a

lot of time to see if the networks either was too large or too small we tested different sizes of the networks. It was both tested with the same configuration on all three networks and different configuration. By testing different configurations of the networks we did not make any significant findings. There was some configurations that worked better than others, but this is to expect as the size and configurations will change the behaviour of the networks. However, as far as we could see, all the networks configurations struggled with the expected free energy exploding to some degree and large variations in performance between the training runs.

We see in Figure 5.9 that when the expected free energy explodes, the variational free energy also increases a lot. The mean variational free energy does not reach the values of the mean expected free energy loss, but at runs where the mean expected free energy losses are high, the mean variational free energy is also high. This is also the case for the mean predicted state error. As we have seen, the variational free energy is mostly affected by the predicted state error when the predicted state error is high. This is also seen by Figure 5.9 showing that the mean variational free energy and the mean predicted state error are almost outputting the same curve. In addition, we see that the mean predicted state error does coincide with the variations of the mean expected free energy loss. The reason for this is that the bootstrapped expected free energy estimate is calculated with the predicted state error. Therefore, an error in the predicted state error will increase the difference between the bootstrapped expected free energy estimate and the expected free energy network. This will in turn increase the loss function and lead to a large change in the expected free energy network. The variational free energy is used for training of the transition network and when errors occur in the predicted state error, the transition network should be corrected to make better estimations. When the transition network makes better estimations, such that the predicted state error is smaller, the expected free energy network is corrected with a new bootstrapped estimate. However, if the changes in the predicted state error is too large, it could lead to large changes in the weights in the expected free energy network which can cause the network to diverge. This can lead to that the expected free energy increases out of control. As the expected free energy is used for calculating the action distribution, this may lead to action distributions that does not reflect the desired behaviour. Thus, the agent never finds the goal and this training run will provide poor results.

It was also tested to calculate the bootstrapped estimate of the expected free energy and the variational free energy without the predicted state estimate. Now, the cases with exploding mean expected free energy loss and mean variational free energy were removed and these quantities were much more consistent. Over 25 training runs, the mean expected free energy loss varied between 3.0 and 4.75 and the mean variational free energy only had values lower than 0.0025. However, even if the extremes was removed by removing the predicted state error, the goal hits per training run was about the same but without the cases with zero hits. Even though the training seems to be the about the same, however much more stable,

the testing performance is not as good as when we include the predicted state error. In the best run, the agent found a policy that leads to the goal from 72 states with $(1, 11)$ as the state furthest away from the goal. Mostly, the agent hits the goal from under 20 of the states. This is worse than when we included the predicted state error where the agent at most hits the goal from 268 states and from $(0, 5)$ at best. Even though this best case when using the predicted state error can be looked as a deviation from what we usually will get, the performance by using the predicted state error is on average better when not using it. This shows that this MSE between the true state and the transition network works as an epistemic value even if it is not the original expression. This causes the agent to visit states minimize errors in the transition model and minimizes ambiguity. Therefore, we still included the predicted state error even though it is the cause for occasionally poor results. However, the predicted state error might be the issue for not finding a correct the correct policy in Part 1. Even in the cases with a lower mean predicted state error and mean expected free energy loss, this quantity might make the bootstrapped expected free energy estimation too inaccurate to be able to use it for action selection. In an open and small environment as in Part 2 we see that the predicted state error moves towards zero. But, in the larger Part 1, the environment might be too complex to get the predicted state error small enough for the estimated expected free energy values to stabilize completely.

By further comparing deep active inference to reinforcement learning and deep q-learning we see that both method uses bootstrapping to calculate what we can call the value function. As the expected free energy assigns a value to each state-action pair, just as the Q-value function in deep Q-learning, we can look at this quantity as a value function even though it is not calculated in the same manner. As we use rewards as a prior desire in the deep active inference algorithm, both value functions uses rewards to calculate the bootstrapped value. However, in addition to reward, deep Q-learning uses the maximum Q-value from next state when estimating the Q-value, while deep active inference uses the epistemic value and the expected free energy of the next state times the action distribution. This leads to a different performance where the deep active inference does the exploration itself, but it seems like it does not estimate the expected free energy value good enough to get as good performance as the DQN-agent.

Where the reinforcement learning-agent learns a policy to maximize the sum of expected reward, the deep active inference-agent selects action to maximize its model evidence and to reach its desires. These desires can be represented in several different ways, which makes active inference more flexible than reinforcement learning when it comes to representing the rewards. When training the deep active inference-agent, we tested several different reward functions where many gave decent performance. In Section 3.4.1 we saw that due to the complete class theorem, any scalar reward signal could be encodes as a prior. Thus, the reward function can be though of like a probability distribution over states. Receiving high, positive rewards for being in a state is more or less the same as having a high prior

probability probability of being in these states. This can make it more logical to assign rewards, as you assign high rewards to the states that you want the agent to be located in and low rewards to undesired states. Also, a probability distribution can be used directly as the prior goal, which makes it easier for active inference to represent its prior desires than reinforcement learning where the reward function have to be carefully constructed.

When it comes to training of the two different methods, the training of the deep active inference model took longer time than the DQN model. A reason for this may be that the deep active inference has to train three networks against the one of the DQN algorithm. In addition, we found that the deep active inference needed a lot of samples when training. As we have discussed, there seems to be issues with the approximation of the expected free energy and to make this as good as possible we need a lot of samples for everywhere in the environment. In total we ran the algorithm 12000 times each run in Part 1 for the deep active inference, while we only ran the DQN 1500 times each run. This tells us that the deep active inference algorithm is very sample inefficient.

In the papers [31] and [25] the deep active inference algorithms performs better or as good as the reinforcement learning algorithms used. Exactly why the deep active inference model performs worse than the model in this thesis is not clear. As we earlier have discussed, a possibility is that the environment used in Part 1 is too complex for the deep active inference agent. In the papers mentioned, they have tested the algorithms on discrete OpenAI Gym environments such as the Cartpole-v1, Acrobot-v1 and LunarLander-v2. These tasks are quite different from the task proposed in this thesis, as these have fewer possible actions to perform and as there are no obstacles present. Also, there is always the possibility of errors in the implementation of the algorithms and environments. However, as we receive good results on Part 2 of the task we think that it is more likely that this method of deep active inference is not suitable for the type of task we have performed in this thesis.

There also exists other methods to perform deep active inference and estimate expected free energy that might give better results. As far as we could see when researching active inference, there is no method of performing deep active inference that the consensus think is better than others. There exists different methods that works in different ways with different strengths and weaknesses. The reason for choosing the algorithm as we did is that we saw from the papers that the deep active inference algorithm gave good performance compared with reinforcement learning algorithms on problems familiar to us beforehand, as they used environments from OpenAI Gym. Also, in our opinion, the algorithm in [31] was the algorithm that was the easiest to understand and the algorithm that was presented in the best manner. Another paper and way of implementing deep active inference that we looked at, and also begun to implement, was the implementation of deep active inference with Monte-Carlo tree search in [13]. This paper criticize

[31], which was one of the papers we ended up using, for deviating from vanilla active inference when it comes to the ability to plan, as they approximate the expected free energy based on bootstrapped samples. They also question if the algorithm in [31] can scale up to more complex problems as it only has been tested on low-dimensional problems. These remarks were taken into account when we chose between the different implementations. However, it was though that as the deep active inference model in [31] performed in continuous state environments such as the OpenAI Gym environments Cartpole-v1, Acrobat-v1 and Lunar-v1 Lander, the complexity of the discrete environment in this task should not be an issue. Retrospectively, we see that this is probably not completely true. Even though the size of the environment is not too big for the method, we have through the results obtained found that the way the environment is constructed may be too complex for the method. Another factor for choosing the approach in [31] is that this method has a few similarities to the way of training reinforcement learning algorithms. Also, [13] use Monte-Carlo search trees it was thought that this quickly takes long time to run when the environments get large. Furthermore, we reached promising results first with the approach from [31] and thus we stopped looking at the approach of deep active inference with Monte-Carlo search trees. If we had chosen the other model we might have received better results, but it is difficult to know exactly.

Another alternative to the deep active inference algorithm and deep active inference in general is sophisticated active inference. This thesis is a continuation of our project thesis, where we used sophisticated active inference to find a policy in a discrete environment similar to the one used here. By using sophisticated active inference we managed to find the goal from every open position in a 30×30 environment. Therefore it was considered to also use this method on the environment in this thesis. However, in the sophisticated active inference case we could not take the vessel length into account. Also, in sophisticated active inference the expected free energy is calculated in a recursive way by iterating through every single state and action combination. In this case it is not performed any learning or exploration of the environment. Basically, this method uses a mathematical function that recursively calculates the expected free energy of every state-action pair instead of using an agent that explores the environment. If it gets enough run time it will in this way manage to find the optimal path. In addition, for this to work the environment has to be known or learnt in advance. As the transition model is used in the calculation of the expected free energy we have to be able to use it directly. It can be learned, but this will probably be difficult get this completely correct unless the environment is small. Because of these drawbacks and that we wanted to see how the exploration and exploitation can be done with active inference we decided to use deep active inference instead in this thesis.

6.2 Future work

From the results and the discussion it is seen that the deep active inference algorithm presented in this thesis can only solve parts of the problem considered here. Therefore, for further work, this method of solving deep active inference should be examined further to investigate other advantages and disadvantages of this method and further examine why it did not managed to solve all parts of the problem. We have proposed some reasons for why it did not work as we hoped, but more aspects can probably be found by digging further into this algorithm and deep active inference. Also, this method of deep active inference should be tested on similar planning problems as proposed in this thesis. The way of performing deep active inference in this thesis has been questioned in Fountas et al. [13] if this method scales for more complex problems. Thus, deep active inference should be performed on other problems with similar complexity as the one in this thesis to see if it does not scale well. Nonetheless, we reached good performance from the deep active inference agent on part 2 of the problem and it has been seen in other papers that this method can be useful for planning and decision making. Thus, there might still be a potential of using this method for solving the problem proposed in this thesis and being an alternative to reinforcement learning. In our opinion it should be continued to be explored.

In addition, it has also been proposed several other ways of performing deep active inference. Therefore, it should be tested if some of these methods can solve the problem proposed and compared to the method of deep active inference used in this thesis. Even though we did not reach optimal results in this thesis, we still think deep active inference has the potential to solve this problem. Also, it should be investigated how the methods perform compared to each other to look at advantages and disadvantages with the different methods.

Furthermore, deep active inference and an reinforcement learning algorithm should be tested and compared as high-level action planners in a continuous version of the marine vessel. This part of the problem proposal was intended to be looked at, but we did not managed to explore it in this thesis. By making the state and action space continuous we get a more realistic model of the marine vessel. Thus, the deep active inference agent and reinforcement learning agent can be used directly for guidance and control of the marine vessel as well as planning. In addition, it should be explored how the deep active inference handles the problem when the environment is partially observable. The factors in deep active inference that handle the uncertainties regarding the partially observability was omitted in this thesis and it should be explored how these aspects affect the performance. These factors may contribute in showing other sides and properties of deep active inference.

Chapter 7

Conclusion

In this thesis we have formulated a problem for using active inference and reinforcement learning as high-level action planners in combination with low-level guidance and control of marine vessels. In addition, we explored how deep active inference performs as a high-level action planner for docking of a marine vessel in a discrete environment. For comparison, the reinforcement learning algorithm deep Q-learning was used on the same problem as both algorithms can be used for decision making and planning on Markov decision process (MDP) environments. To see how well the algorithms performs as high-level action planners for a marine vessel, a marine vessel model was implemented and simulated. This was done by producing paths with the deep active inference and DQN algorithm and then guidance and control of the vessel was performed by following the paths. Also, characteristics of the deep active inference algorithm was discussed and compared to reinforcement learning.

Deep active inference and deep Q-learning were tested on a 30×30 discrete harbour environment. The objective of the task was to move a discretized version of the marine vessel with discrete actions from outside a port area to a goal position surrounded by obstacles. For simplicity, the task was split in two. Part 1 of the problem dealt with guiding the agent from outside the port area to the docking area. In Part 1, the deep active inference-agent did not manage to solve the problem of moving through the environment without avoiding obstacles. The agent did not manage to find a path that leads the vessel from the specified starting position, and at best it managed to hit the goal from position $(0, 5)$. However, the path from this position would cause the marine vessel to crash. On the other side, the DQN-agent solved Part 1 of the task without any problems.

Part 2 of the task considers the docking operation of moving the marine vessel into a prespecified docking position which consists of both a desired Cartesian position and desired heading. In this part, the deep active inference-agent performs well. In this case it manages to find an optimal path from the start position and obtains

a path almost equal to the DQN-agent. The only difference between the results of the two algorithms is when the marine vessel is turned around.

Overall, the results in this thesis show that the deep active inference model does not work well for action planning on larger discrete environments. We see from the results from Part 1 of the task that the deep active inference struggles with consistency and with the expected free energy diverging in training, due to bad estimations made by the transition network. Also, in cases with better training performance it does not find a policy that guides the agent to the goal both in the open area and the port area. It seems like the agent does not manage to handle a combination of the different characteristics of the open area and the port, and that the environment in Part 1 is too complex. A possible explanation may be that it struggles with propagation of the expected free energy due to inaccuracy of the estimates due to the use of the predicted state error. However, we cannot give a definite answer to why it does not perform on Part 1 and the algorithm should be further examined.

Despite the poor results on Part 1, the deep active inference agent obtained good results on Part 2 where it performed as well as the DQN-agent. In addition, deep active inference has some advantages compared to reinforcement learning. Its way of being able to representing prior desires, both as rewards and probability distributions instead of only rewards, is more flexible and easier to use instead of finding a good reward function. In addition, the value function in deep active inference has a epistemic value that gives a balance between exploration and exploitation. Also, it has been seen in other papers that deep active inference often performs as well as or better than deep reinforcement learning and there are several ways of performing deep active inference. Therefore, despite our poor results, we think that deep active inference and active inference is still worth exploring further. In addition, it is worth to note that this is a relatively new framework that needs to be assessed further to get a good view of its properties and performance.

Bibliography

[1] C. C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] E. Anderlini, G. G. Parker, and G. Thomas. "Docking control of an autonomous underwater vehicle using reinforcement learning". In: *Applied Sciences* 9.17 (2019), p. 3456.

[3] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz, et al. "Self-driving cars: A survey". In: *Expert Systems with Applications* 165 (2021), p. 113816.

[4] R. Bogacz. "A tutorial on the free-energy framework for modelling perception and learning". In: *Journal of mathematical psychology* 76 (2017), pp. 198–211.

[5] O. Çatal, S. Wauthier, T. Verbelen, C. De Boom, and B. Dhoedt. "Deep active inference for autonomous robot navigation". In: *arXiv preprint arXiv :2003.03220* (2020).

[6] Y. Cheng and W. Zhang. "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels". In: *Neurocomputing* 272 (2018), pp. 63–73.

[7] L. Da Costa, T. Parr, N. Sajid, S. Veselic, V. Neacsu, and K. Friston. "Active inference on discrete state-spaces: a synthesis". In: *Journal of Mathematical Psychology* 99 (2020), p. 102447.

[8] D. Demekas, T. Parr, and K. J. Friston. "An investigation of the free energy principle for emotion recognition". In: *Frontiers in Computational Neuroscience* 14 (2020), p. 30.

[9] Z. Ding and H. Dong. "Challenges of reinforcement learning". In: *Deep Reinforcement Learning*. Springer, 2020, pp. 249–272.

[10] B. Ergul, T. van de Laar, M. Koudahl, M. Roa-Villescas, and B. d. Vries. "Learning Where to Park". In: *International Workshop on Active Inference*. Springer. 2020, pp. 125–132.

[11] EU Publications Office. *Maritime Unmanned Navigation through Intelligence in Networks*. Last accessed 04 June 2022. 2022. URL: `https://cordis.europa.eu/project/id/314286`.

[12] T. I. Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2021.

[13] Z. Fountas, N. Sajid, P. Mediano, and K. Friston. "Deep active inference agents using Monte-Carlo methods". In: *Advances in neural information processing systems* 33 (2020), pp. 11662–11675.

[14] J. C. de Freitas and P. H. V. Penna. "A variable neighborhood search for flying sidekick traveling salesman problem". In: *International Transactions in Operational Research* 27.1 (2020), pp. 267–290.

[15] K. Friston. "The free-energy principle: a rough guide to the brain?" In: *Trends in cognitive sciences* 13.7 (2009), pp. 293–301.

[16] K. Friston. "The free-energy principle: a unified brain theory?" In: *Nature reviews neuroscience* 11.2 (2010), pp. 127–138.

[17] K. Friston, L. Da Costa, D. Hafner, C. Hesp, and T. Parr. "Sophisticated inference". In: *Neural Computation* 33.3 (2021), pp. 713–763.

[18] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, G. Pezzulo, et al. "Active inference and learning". In: *Neuroscience & Biobehavioral Reviews* 68 (2016), pp. 862–879.

[19] K. Friston, J. Kilner, and L. Harrison. "A free energy principle for the brain". In: *Journal of physiology-Paris* 100.1-3 (2006), pp. 70–87.

[20] K. Friston, F. Rigoli, D. Ognibene, C. Mathys, T. Fitzgerald, and G. Pezzulo. "Active inference and epistemic value". In: *Cognitive neuroscience* 6.4 (2015), pp. 187–214.

[21] K. Friston, S. Samothrakis, and R. Montague. "Active inference and agency: optimal control without cost functions". In: *Biological cybernetics* 106.8 (2012), pp. 523–541.

[22] K. J. Friston, J. Daunizeau, and S. J. Kiebel. "Reinforcement learning or active inference?" In: *PloS one* 4.7 (2009), e6421.

[23] S. Guo, X. Zhang, Y. Zheng, and Y. Du. "An autonomous path planning model for unmanned ships based on deep reinforcement learning". In: *Sensors* 20.2 (2020), p. 426.

[24] O. van der Himst and P. Lanillos. *Grottoh/Deep-Active-Inference-for-Partially-Observable-MDPs*. Last accessed 22 May 2022. 2021. URL: `https://github.com/Grottoh/Deep-Active-Inference-for-Partially-Observable-MDPs`.

[25] O. v. d. Himst and P. Lanillos. "Deep active inference for partially observable mdps". In: *International Workshop on Active Inference*. Springer. 2020, pp. 61–71.

[26] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[27] M. Kirchhoff, T. Parr, E. Palacios, K. Friston, and J. Kiverstein. "The Markov blankets of life: autonomy, active inference and the free energy principle". In: *Journal of The royal society interface* 15.138 (2018), p. 20170792.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[29] A. B. Martinsen, A. M. Lekkas, and S. Gros. "Autonomous docking using direct optimal control". In: *IFAC-PapersOnLine* 52.21 (2019), pp. 97–102.

[30] E. Meyer, A. Heiberg, A. Rasheed, and O. San. "COLREG-Compliant Collision Avoidance for Unmanned Surface Vehicle Using Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 165344–165364. DOI: 10.1109/ACCESS.2020.3022600.

[31] B. Millidge. "Deep active inference as variational policy gradients". In: *Journal of Mathematical Psychology* 96 (2020), p. 102348.

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

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[34] Z. H. Munim. "Autonomous ships: a review, innovative applications and future maritime business models". In: *Supply Chain Forum: An International Journal*. Vol. 20. 4. Taylor & Francis. 2019, pp. 266–279.

[35] M. A. Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[36] OpenAi. *Environment creation*. Last accessed 22 May 2022. URL: https://www.gymlibrary.ml/content/environment_creation/#.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[38] A. Paul, N. Sajid, M. Gopalkrishnan, and A. Razi. "Active Inference for Stochastic Control". In: *arXiv preprint arXiv:2108.12245* (2021).

[39] Rolls-Royce. *Rolls-Royce and Finferries demonstrate world's first fully autonomous ferry*. Last accessed 22 May 2022. URL: https://www.rolls-royce.com/media/press-releases/2018/03-12-2018-rr-and-finferries-demonstrate-worlds-first-fully-autonomous-ferry.aspx.

[40] E. Rørvik. "Automatic docking of an autonomous surface vessel: Developed using deep reinforcement learning and analysed with Explainable AI". In: *MA thesis* (2020).

[41] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[42] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[43] Ruter. *Self-driving vehicles*. Last accessed 22 May 2022. URL: https://ruter.no/en/about-ruter/reports-projects-plans/autonomous-vehicles/.

[44] N. Sajid, P. J. Ball, T. Parr, and K. J. Friston. "Active inference: demystified and compared". In: *Neural computation* 33.3 (2021), pp. 674–712.

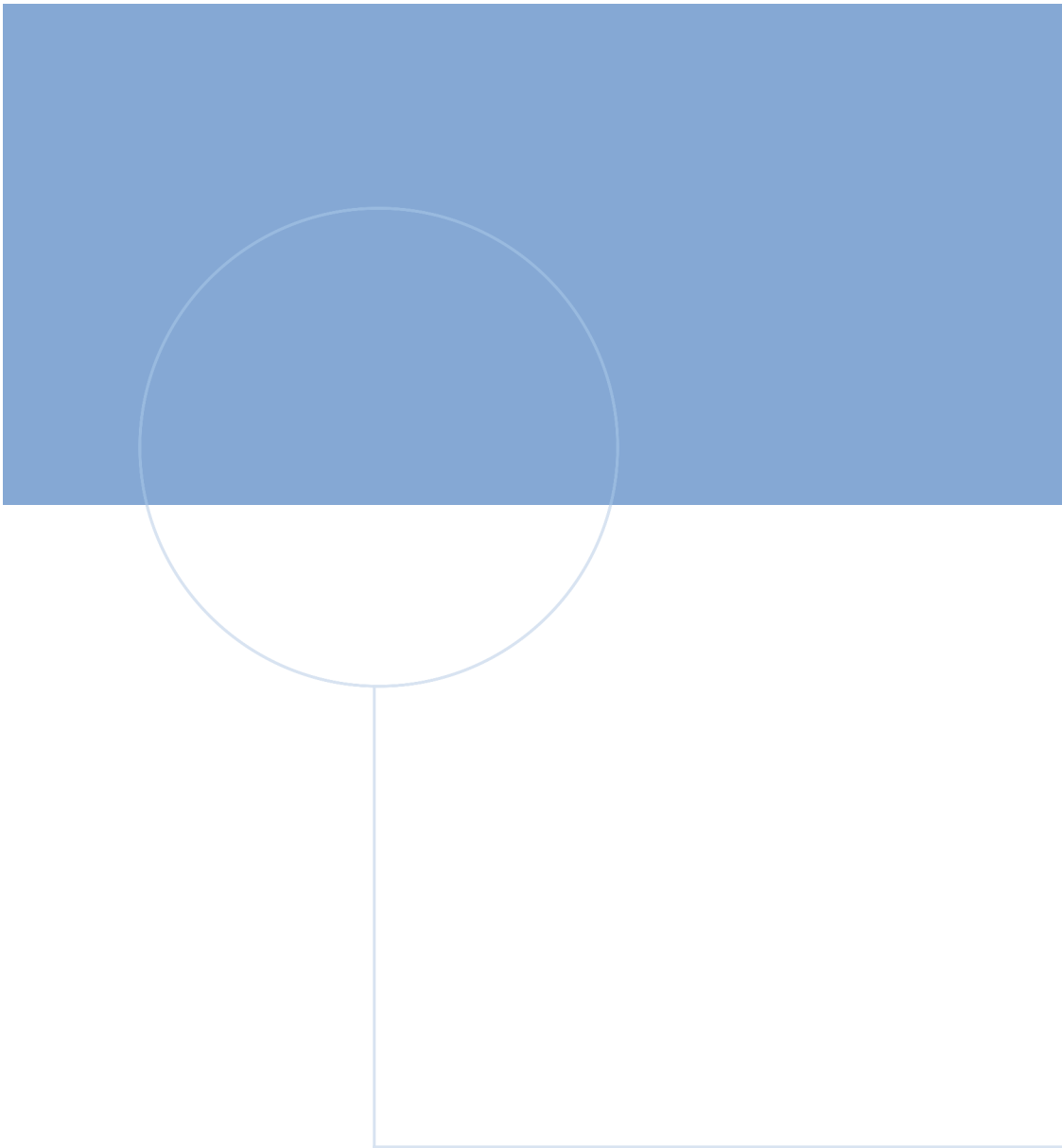
[45] A. Skredderberget. *The first ever zero emission, autonomous ship*. Last accessed 04 June 2022. 2018. URL: https://www.yara.com/knowledge-grows/game-changer-for-the-environment/.

[46] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[47] A. Tschantz, M. Baltieri, A. K. Seth, and C. L. Buckley. "Scaling active inference". In: *2020 international joint conference on neural networks (ijcnn)*. IEEE. 2020, pp. 1–8.

[48] H. Tvete. *The ReVolt. A new inspirational ship concept*. Last accessed 04 June 2022. URL: https://www.dnv.com/technology-innovation/revolt/.

[49] K. Ueltzhöffer. "Deep active inference". In: *Biological cybernetics* 112.6 (2018), pp. 547–573.



NTNU

Norwegian University of
Science and Technology