Alexandra Skau Vedeler

Learning an End-to-End Steering Model for an Unmanned Surface Vehicle

Inverse Optimal Control for End-to-End Mapping of Input Sensor Data to Action Parameters

Master's thesis in Cybernetics and Robotics

Supervisor: Kristin Y. Pettersen

June 2019

Master's thesis

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

NTNU Kunnskap for en bedre verden

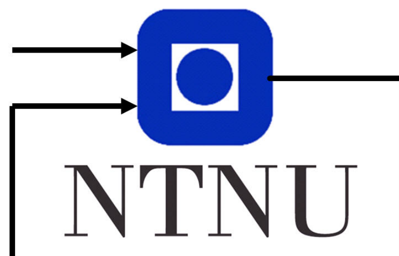
FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment

Learning an End-to-End Steering Model for an Unmanned Surface Vehicle

INVERSE OPTIMAL CONTROL FOR END-TO-END MAPPING OF INPUT SENSOR DATA TO ACTION PARAMETERS

Alexandra S. Vedeler

Master's Thesis in Cybernetics and Robotics
Supervisor: Kristin Ytterstad Pettersen, ITK
Co-supervisor: Narada Warakagoda, FFI
June 2019



Problem Formulation

This task deals with the ability of an Unmanned Surface Vehicle (USV) to move autonomously without colliding with other static or moving objects found in the environment. The vehicle is endowed with this ability through several modules such as scene understanding, navigation, route planning, and the control system. Traditionally, these modules are designed and optimized separately and combined afterward. For example, scene understanding module is designed to detect interesting objects such as other boats, ships, and buoys etc. and the control system itself needs to convert this information into appropriate action parameters, such as steering angle, engine power or degree of braking. In this proposal, we advocate a tighter coupling among individual modules, where the system directly generates the action parameters based on the sensor data. This end-to-end approach allows one to make the overall system more compact, efficient and accurate. Deep learning is an excellent candidate to implement such an end-to-end mapping of the input sensor data onto the action parameters. One well-known area of deep learning suitable for similar problems is deep reinforcement learning (DRL). However, DRL depends on a known reward function. But in the case of a USV, it is difficult to manually design a good reward function because at least in certain maneuverings the relevant reward structure is not clear. A solution to this problem is learning the reward function together with the control policy based on training data using a technique known as Inverse Reinforcement Learning (IRL).

The objective of the task is to train and test a system based on deep neural networks that map sensor inputs to action parameters of the control system of the USV using the Inverse Reinforcement Learning approach. A suitable derivative of IRL or a similar algorithm is envisaged to be adopted. Given the limited duration of the

project, the task may be simplified with respect to the type and number of inputs and outputs within the goal of end-to-end training. The system will be trained to perform a particular type of maneuverings such as collision avoidance with stationary objects. The task consists of development and testing of the system as well as evaluation of the results.

Abstract

The task of obstacle avoidance using maritime vessels, such as Unmanned Surface Vehicles (USVs), has traditionally been solved using specialized modules that are designed and optimized separately. However, this approach requires a deep insight into the environment, the vessel, and their complex dynamics. In this project, we study an alternative method that maps the USV's sensor output to steering actions in a direct end-to-end way in hopes of making the system more compact, efficient and demand less insight into the complex dynamics of the environment. Deep Reinforcement Learning is a promising alternative in this regard and has produced some impressive results over the last years. However, the requirement of a manually crafted reward function may hinder its use in cases where the wanted behavior is difficult to express.

We propose the use of Imitation Learning (IL) using Deep Reinforcement Learning (RL) and Deep Inverse Reinforcement Learning (IRL) and present a system that learns an end-to-end steering model capable of mapping radar-like images directly to steering actions in an obstacle avoidance scenario. In addition to this, we present an RL system and a handcrafted reward function for the task in order to evaluate both IL and RL performance on the task. We found that while RL performs with the greater accuracy and consistency, both systems are able to grasp the task of obstacle avoidance using only a mix of radar and GPS observations, completely model free. We thus deem both RL and IL promising options for future development in USV tasks, where IL specifically may provide an option for tasks that are even more difficult to model with a reward function than this specific case.

Sammendrag

Hindringsunngåelse med maritime fartøy, slik som Ubemannede Overflatefartøy (UOF), har tradisjonelt sett blitt løst ved hjelp av spesialiserte moduler som er designet og optimalisert separat fra hverandre. Denne tilnærmelsen krever imidletid et høyt kunnskapsnivå om miljøet, fartøyet, og deres komplekse dynamikk. I dette prosjektet studerer vi en alternativ metode som kan transformere fartøyets sensor data til styringsaksjoner i en mer direkte, ende-til-ende-metode i håp om å gjøre systemet mer kompakt, effektivt og at det krever mindre innsikt inn i den komplekse dynamikken til det maritime miljøet. Dyp Forsterkende Læring (FL) er slikt sett et lovende alternativ som har hatt flere imponerende resultater i de siste årene. Imidlertid gjør denne metodens bruk av en manuelt utformet belønningsfunksjon, noe som kan være vanskelig å anskaffe i tilfeller der den ønskede atferden er vanskelig å formulere matematisk.

Vi foreslår her bruk av Imitasjonelæring (IL) med bruk av Dyp Forsterkende Læring (FL) of Dyp Invertert Forsterkningslæring (IFL), og vi presanterer et system som lærer en ende-til-ende-styringsmodell som kan transformere radar-lignende bilder direkte til styringsaksjoner i en hindringsunngåelsessituasjon. I tillegg til dette presenterer vi et FL-system og en håndlaget belønnelsesfunskjon for oppgaven slik at vi kan evaluere prestasjonen til både IL og FL. Vi fant at FL presterer best, men at begge systemene klarer å fatte oppgaven ved bruk av observasjoner kun bestående av radar og GPS data. Vi anser dermed både RL og IL som lovende alternativer for videre utvikling i UOF-oppgaver, hvor IL spesielt kan funcgere som et alternativ i oppgaver som er vanskeligere å beskrive i en belønningsfunksjon enn dette spesifikke tilfellet.

Preface

This thesis concludes my five-year journey at the Master of Cybernetics and Robotics program of the Norwegian University of Science and Technology and is the culmination of my work under the supervision of Kristin Y. Pettersen and Narada Warakagoda, spring 2019. The thesis summarizes my work and my findings as well as the theory and methods which my work builds upon.

The project was performed in collaboration with the Norwegian Defence and Research Establishment (FFI). The results rely on a USV simulator received from FFI and a dataset collected in collaboration with FFI during a previous preparatory project (Vedeler; 2018). The reader is not required to know the specifics of this preparatory project.

Acknowledgment

I would like to thank my co-supervisor, Narada Warakagoda, for insightful discussions and support throughout this project. Your guidance and support have been invaluable. I would also like to thank my supervisor, Professor Kristin Y. Pettersen, for providing guidance on the writing process and other formalities regarding this thesis. In addition, I would like to extend my gratitude towards the Norwegian Defence and Research Establishment (FFI) for providing me with the necessary resources to complete the task. Specifically, I would like to thank Else-Line Malene Ruud for her work on the USV simulator which this project relies upon and Jarle Sandrib for his help in collecting the demonstration dataset used in the project. Lastly, I want to thank my fellow graduate students for inspiring discussions as well as my family and friends for their love and support.

07.06.2019

Alexandra Skau Vedeler

Table of Contents

Abstract **iv**

Sammendrag **v**

1 Introduction **1**

1.1 Previous works 2
1.2 Contribution and Background 3
1.3 Abbreviations and Notation 5
1.4 Outline 6

2 Background and Theory **7**

2.1 Deep Learning 7
2.1.1 Artificial Neural Networks 8
2.1.2 Deep Neural Networks 11
2.1.3 Convolutional Neural Networks 13
2.1.4 Training 15
2.2 GAN 16
2.2.1 Training 17
2.2.2 Advantages and Drawbacks 19
2.3 Reinforcement Learning 20
2.3.1 The core idea of Reinforcement Learning 21
2.3.2 Marcov Decision Process 21
2.3.3 Optimal Policy 22
2.3.4 Deep Reinforcement Learning 24
2.3.5 Trust Region Policy Optimization 26

2.4 Inverse Reinforcement Learning 31
2.4.1 The concept of IRL 31
2.4.2 IRL algorithms 32
2.5 Imitation Learning 33
2.5.1 GAN and Imitation Learning 33
2.5.2 GAIL 34
2.6 Unmanned Surface Vehicle 37
2.6.1 Radar 38

3 Method 41
3.1 Programs used 42
3.2 The task 43
3.2.1 Observation of state 43
3.3 Training Data 46
3.3.1 Data Gathering 46
3.3.2 Simulation of environment 47
3.4 Reinforcement Learning setup 51
3.4.1 Policy 52
3.4.2 Reward 54
3.5 Imitation Learning setup 56
3.5.1 Data Preprosessing 57
3.5.2 Discriminator 58

4 Results and Discussion 61
4.1 Validation 61
4.2 Results 62
4.2.1 Training 62
4.2.2 Expert Demonstrations 62
4.2.3 Rwsults using Positional Observations 64
4.2.4 Radar Observations 68
4.2.5 The Difference between Categorical and Gaussian Policy . 72
4.2.6 Summary of Results 72
4.3 Discussion 73
4.3.1 Observations and actions 73
4.3.2 GAN 74
4.3.3 End-to-end learning in practical use 75
4.3.4 RL vs IL 76

4.3.5 Model-free systems 76
4.4 Future Work 78

5 Conclusions **81**

A Additional Theory **83**
A.1 Maximum Entropy IRL 83
A.2 Deep Maximum Entropy IRL 84
A.3 Guided Cost Learning 85
A.3.1 The GCL Cost Optimization 87
A.3.2 The GCL Policy Optimization Step 88

References **89**

List of Tables

1.1 Summary of the abbreviations frequently used in this paper 5
1.2 A summary of the notation frequently used in this paper 6

4.1 Success rates of the different system setups. 73

List of Figures

1.1 Simplified illustration of our system 4

2.1 Simplified illustration of a biological neuron 9
2.2 Simplified illustration of an artificial neuron 9
2.3 Plot of three activation functions 11
2.4 Example of a Neural Network structure 12
2.5 Example of convolution of a kernel and an image 14
2.6 Simplified example of a CNN detection 15
2.7 Illustration of GAN 18
2.8 Agent interacting in an environment 21
2.9 Radar image example 38
2.10 Simplified illustration of a radar spoke 39

3.1 Simplified illustration of our system 42
3.2 Conceptual illustration of obstacle avoidance 43
3.3 Illustration of positional vectors as defined in our system 45
3.4 Simplified illustration of the environment module 47
3.5 Illustration of initial episode setup 48
3.6 Example of a stand-in radar image 50
3.7 Simplified illustration of the RL module of the system 51
3.8 Simplified illustration of the policy network 54
3.9 Simplified illustration of the IRL module of the system 57
3.10 Simplified illustration of our discriminator network 59

4.1 Expert actions from Figure 4.2 discretized 63

4.2 Example of an expert demonstration with continuous action space 63
4.3 Example of successful episode from validation of the RL setup with positional observations with Gaussian Policy 64
4.4 Example of successful episode from validation of the RL setup with positional observations with categorical policy 65
4.5 Rewards corresponding to the episode in Figure 4.4 65
4.6 Example of unsuccessful episode from validation of the IL setup and positional observations and Gaussian policy 66
4.7 Rewards corresponding to the episode in Figure 4.6 66
4.8 Example of successful episode from validation of the IL setup with positional observations with Categorical Policy 67
4.9 Example of episode from validation of the IL setup with radar image as the only observation 68
4.10 Example of successful episode from validation of the RL setup with radar observations with Gaussian policy 69
4.11 Example of successful episode from validation of the RL setup with radar observations with categorical policy 70
4.12 Example of unsuccessful episode from validation of the IL setup with radar observations with Gaussian policy 70
4.13 Example of successful episode from validation of the IL setup with radar observations with categorical policy 71
4.14 Test with waypoint following 80

A.1 An illustration of the GCL algorithm 86

Chapter 1

Introduction

Traditionally, USV steering has been performed through the use of a set of several modules, each designed and optimized separately before they are combined into the full system. For example, a scene understanding module may be designed to detect interesting objects such as other boats, which may be used by the navigation module to pinpoint its location relative to the USV. The guidance system then uses this information to generate new trajectories which in turn is delivered to and executed by the control system through the use of appropriate action parameters such as a change in rotor angle or a decrease in speed. While this approach has given good results it requires a high level of engineering, tuning, and knowledge in several disciplines, e.g. object detection from sensory input, knowledge of the system dynamics, path planning, and control system algorithms. The resulting system is also not optimized with respect to an overall objective function, but rather a collection of locally optimized subsystems.

An alternative approach advocates a tighter coupling between the sensor input and the actions taken by the vehicle. This project studies such an approach, often dubbed an end-to-end approach, where the system directly generates action parameters based on the sensory data, thus making the system overall more compact, efficient and accurate. Deep Neural Networks (ANN) offer a high level of expressive power. They are able to handle highly nonlinear relationships between their input and output (Schmidhuber; 2015; Goodfellow et al.; 2016), an ability which is necessary in order to perform end-to-end steering in USVs. Systems that couple deep ANNs with Reinforcement Learning (RL), have also proved to be able to learn complex

tasks (Silver et al.; 2016; Mnih et al.; 2015; Martinsen and Lekkas; 2018; Levine et al.; 2016; Cheng and Zhang; 2018; Kober et al.; 2013; Sutton and Barto; 2017). However, RL requires a known reward function to guide its training. We argue that such a function can be highly difficult to craft manually for USV steering as certain maneuverings do not express a clearly weighted cost structure (Abbeel and Ng; 2004). Because of this, combining RL with Inverse Reinforcement Learning (IRL), where the goal is to learn the reward function from a set of demonstrations[1], may be a preferable solution. This combination of RL and IRL form an Imitation Learning (IL) system that can learn from demonstrations. Though there are some examples of varying degrees of use of RL in USVs (Martinsen and Lekkas; 2018; Cheng and Zhang; 2018), we have not been able to find examples of the use of IL in USV steering.

In this project, we investigate the use of IL to achieve an end-to-end steering model. To this end, we implement a system that learns to perform obstacle avoidance through the use of radar-like images as observations. Due to its relation to RL, we also implement a system that uses purely RL, with a handcrafted reward function, and discuss and compare both results.

1.1 Previous works [2]

With the use of Artificial Neural Networks (ANNs), recent advancements in Reinforcement Learning (RL) have achieved impressive performances, rivaling, or even beating, that of humans. Examples lie in their use in playing Atari games (Mnih et al.; 2015) or in the program AlphaGo (Silver et al.; 2016), which famously beat the human world champion in the game of Go.

An example of deep learning in steering, is Bojarski et al. (2017), who mapped raw visual input to steering parameters of a car, quite successfully using their deep ANN PilotNet. However, this treats steering more like a deep ANN classification problem, a type of supervised learning. ANNs demand large datasets in order to be trained well (Goodfellow et al.; 2016) and as a supervised learning approach, the data must

[1] IRL does involve RL, i.e. learning a policy, usually as a means to optimize a reward function. However, in this thesis, we will treat them as two separate modules where IRL deals with learning the reward function while RL deals with learning the policy.

[2] This section, though written by us, was originally written for the preparatory project (Vedeler; 2018) preceding this thesis

be labeled before training. PilotNet trained on 6 hours worth of video and sensor data from a human driving a car. Such demands make this approach problematic in the case of USV steering, where large amounts of data is hard to come by or generate. In addition, because a small mistake on the part of the policy will place the system into states that lie outside of the distribution in the training data, supervised learning will in general not generate a policy with good long-horizon performance (Levine et al.; 2016).

Martinsen and Lekkas (2018) used a policy search RL approach, specifically the Deep Deterministic Policy Gradient method, to find the desired policy for straight-path following for an underactuated marine vessel exposed to unknown ocean currents. As described in the article, the approach is model-free, requiring no prior knowledge of the system it is assigned to control. Another example of RL in USV steering is Cheng and Zhang (2018) who proposes a deep RL approach for obstacle avoidance. However, as RL approaches, these require a pre-made reward function.

There are some examples of the use of Inverse Reinforcement Learning (IRL) in steering. One such example is Wulfmeier et al. (2016), who used a Maximum Entropy-based (Ziebart et al.; 2008), non-linear IRL framework with Fully Convolutional ANNs to represent the cost model underlying expert driving behavior. However, we found no examples of IRL in use in USVs, much less the use of the combination of both IRL and RL, known as a type of Imitation Learning (IL), in the steering of USVs.

1.2 Contribution and Background

The main goal of this thesis is to investigate how an end-to-end steering model for obstacle avoidance can be achieved through learning from demonstrated behavior. Specifically, we explored the case of finding a steering model which can generate the appropriate heading for the USV given a radar image observation. Our proposed method uses an Imitation Learning (IL) approach called GAIL (Ho and Ermon; 2016), illustrated in Figure 1.1.

The system uses trial based Reinforcement Learning (RL) to train a steering model in the form of a policy, to produce appropriate actions in order to accomplish a given task. It employs actions generated by the policy in the environment and receives feedback on how well it is performing considering the task at hand. In pure RL,

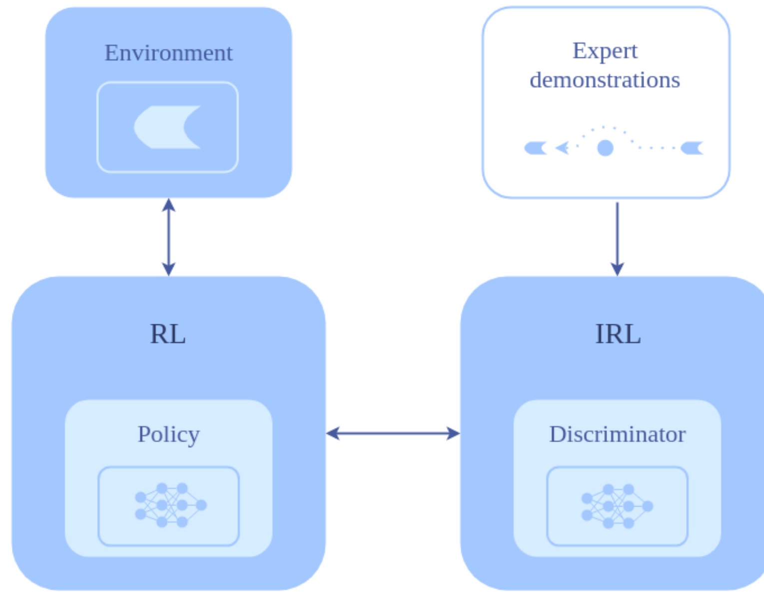


Figure 1.1: Simplified illustration of the Imitation Learning system combining Reinforcement Learning and Inverse Reinforcement Learning.

this feedback comes from a manually crafted reward function. Here, however, it comes from a separate Inverse Reinforcement Learning (IRL) module that learns the essence of the task by examining a set of demonstrations performed by an expert. This way, the IRL module provides feedback to the RL module using a discriminator function.

In our system, both the policy and the discriminator are parameterized by Artificial Neural Networks (ANNs) and are trained using Deep Learning (DL) approaches. The network structures are designed by us.

In addition to the exploration of IL in obstacle avoidance, we have also produced and tested a system of pure RL, complete with a reward function of our own design.

Our results are comprised of both an IL system and an RL system as well as the discussion and comparison of the different results. Through this, we show that both RL and IL may be suitable alternatives to the more traditional approach which separates scene understanding and navigation.

While we have found no examples of use of IL for USV steering, the use of RL in USV obstacle avoidance has been performed by Cheng and Zhang (2018), however in a different way. Cheng and Zhang (2018) uses a different setup and performs obstacle avoidance using a different type of observations comprised of the kinematic

state of the USV, the existence of an obstacle in a given radius and previous control behavior. We show that obstacle avoidance can be performed directly using radar-like images combined with its intended goal position without the agent having any other insight into the USVs state. However, we do note that, unlike Cheng and Zhang (2018), our experiments were conducted without current or other disturbances and that they used only one obstacle at a time.

Our implementation builds on two open source implementations: An implementation of the TRPO algorithm (Schulman et al.; 2015) from the RL codebase of OpenAI:SpinningUp (2018) and the discriminator training scheme from Fu (2018). We altered the two implementations and combined them into an implementation of the GAIL algorithm (Ho and Ermon; 2016). The code is implemented in python, using tensorflow (*Tensorflow home page*; 2017) for Artificial Neural Network (ANN) support. For training, we used four NVIDIA GeForce GTX 1080 GPU's.

All illustrations in this thesis were created using Draw.io unless stated otherwise.

1.3 Abbreviations and Notation

For convenience, we use a number of abbreviations in this paper. A list of the most notable abbreviations is included below in Table 1.1. In addition, we have included a table of notations that are frequently used in Table 1.2. We have chosen to use notation similar to that which is used in control theory.

Abbreviation	Full text
USV	Unmanned Surface Vehicle
DL	Deep Learning
ANN	Artificial Neural Network
GAN	Generative Adversarial Network
RL	Reinforcement Learning
IRL	Inverse Reinforcement Learning
IL	Imitation Learning

Table 1.1: Summary of the abbreviations frequently used in this paper

Symbol	Definition	Example/detail
$\boldsymbol{x}_t$	System state in a Markovian process at time step $t \in [0, T]$	The true state of a system or agent
$\boldsymbol{u}_t$	control or action at time step $t \in [0, T]$	E.g. a change in heading of a USV
$\boldsymbol{o}_t$	Observation at time step $t \in [0, T]$	E.g. a combination of sensor measurements LiDAR, camera image, etc.
τ	Trajectory: $\tau = (\boldsymbol{x}_0, \boldsymbol{u}_0), (\boldsymbol{x}_1, \boldsymbol{u}_1), ...(\boldsymbol{x}_t, \boldsymbol{u}_t)$	The sequence of state-action pairs in an episode
$r(x_t, u_t)$	Reward function that defines the goal of the task	May be parameterized as a Neural Network (NN), in which case ϕ denotes the network weights
$p(x_{t+1} \mid x_t, u_t)$	Unknown system dynamics, the transition function from one state to another	The physics of the environment of the agent
π_θ	Policy, parameterized by θ	May be parameterized as a Neural Network (NN), in which case θ denotes the network weights

Table 1.2: A summary of the notation frequently used in this paper

1.4 Outline

The thesis is organized as follows. In Chapter 2, a review of background information and relevant fields is presented, explaining the theory and algorithms that this project is built upon. Chapter 3 explains the details of our methods, how our system is constructed and why. The results and discussion of the testing is presented in Chapter 4. This chapter also presents a brief discussion of future work. Lastly, conclusions are presented in Chapter 5.

Chapter 2

Background and Theory

Machine Learning (ML) is a field concerned with computers ability to learn a task without following a set of rules or guidelines explicitly stated by the programmer. This approach to task solving is especially relevant for tasks that are more or less easy for humans to perform, but hard to describe formally, such as recognizing words or faces. In this, a function approximator is fitted to generate the correct output from a provided input.

In this chapter, relevant theory for machine learning will be presented. We will give an introduction to Deep Learning (DL) followed by an introduction to Generative Adversarial Networks (GANs). Later in this chapter, the principles of Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) will be presented as well as their use in Imitation Learning (IL). The last section of this chapter is dedicated to an introduction of the Unmanned Surface Vehicle (USV) used in this project and the sensor most relevant for this project.

2.1 Deep Learning

In Machine Learning, the computer attempts to find a behavioral function which can solve a task. Because this function is unknown, it must be approximated and a choice of parametrization must be made. A linear function $f(x) = ax + b$ is one very simple example of a function approximation, in which a conclusion, $f(x)$, results from an observation, x, and parameters, a and b, may be adjusted to improve the

performance of a certain task.

There are several ways with which relationships in data can be parametrized, and each method is suited for function approximation in different situations. The previous example is a simple one, with little expressive power. Another example is Gaussian processes, which are data efficient but struggle with expressing non-smooth curves and are slow to train (Zhang; 2017). Adhering to trends in Machine Learning and Reinforcement Learning (RL), we will in this section focus on Artificial Neural Networks (ANNs) and Deep Learning (DL), which possess great expressive power and scalability to large amounts of data.

In this section, we will present an introduction to ANNs before moving on to Deep Neural Networks which play an important part in some of the most impressive abilities of machine learning systems today, including in RL (Sutton and Barto; 2017). Furthermore, we will take a look at Convolutional Neural Networks (CNNs) an architecture type that builds on ANNs and which is used in our approach. The main source for this section is Goodfellow et al. (2016) unless otherwise specified. For deeper insight into ANNs and their use in RL, we refer to Goodfellow et al. (2016) and Sutton and Barto (2017) respectively.

2.1.1 Artificial Neural Networks

While many associate Deep Learning with new technological fields in artificial intelligence, the principles of Deep Learning date back to the 1940s. In order to understand Deep Learning, we will first take a look at the emergence of Artificial Neural Networks (ANNs).

ANNs were originally inspired by neuroscience and the human brain. The brain of humans and animals consist of a network of biological components called neurons, an example of which is illustrated in Figure 2.1. Simply put, a neuron has a body, called a soma, a tail, called an axon, and connectors known as dendrites. The tail of a neuron branches out and connect to the dendrites of other neurons like electrical wires. Each neuron collects signals passed to them through their dendrites and, depending on the strength of the signals it receives, sends out a signal of its own through its axon, to be used by other neurons. This way, each neuron is connected and receives signals from a number of neurons while sending its own signals to different neurons. Through experience, neurons will adapt to put more weight to

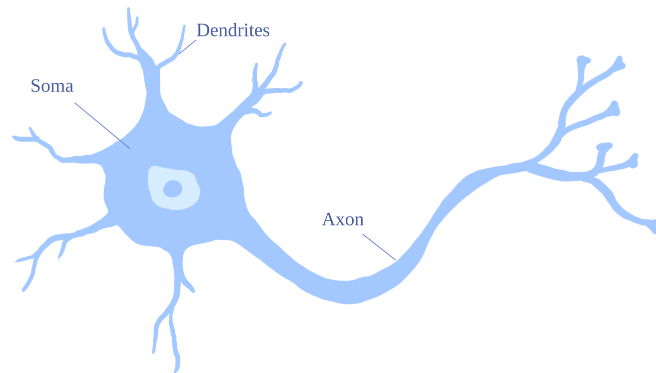


Figure 2.1: Simplified illustration of a biological neuron. The neuron was drawn using PaintTool SAI.

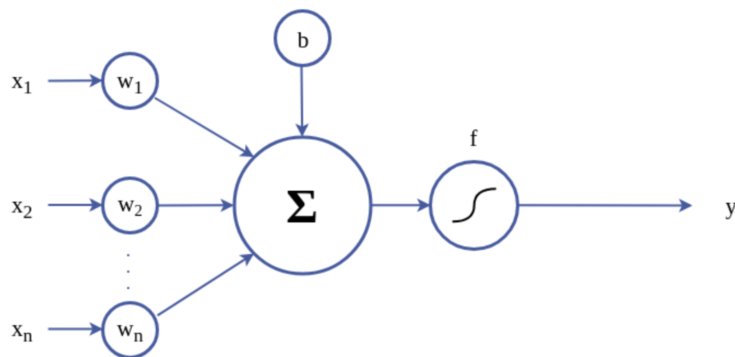


Figure 2.2: Simplified illustration of an artificial neuron

the signals of certain neurons they listen to, and less weight to others. Together, neurons are able to form complex behavior in both animals and humans, despite the individual neuron's relatively simple design. The scientists of the early days of Deep Learning took inspiration from this biological ingenuity and used similar principles to develop ANNs. For deeper insight into biological neurons, see Squire et al. (2008).

Like its biological counterpart, Artificial Neural Networks comprise of a network of neurons. Similarly to the biological neurons, these artificial neurons take a set of n input values, $x_1, x_2, ..., x_n$, and uses a set of n weights, $w_1, w_2, ..., w_n$, to calculate an output. In addition, a bias term, b, is also often used. This amounts to a linear

function

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \tag{2.1}$$

where $\mathbf{x}$ represents the vector of the input to the neuron, $\mathbf{w}$ the vector of weights, and b the bias term. By adjusting the weights and the bias, the output of the neuron can be fitted according to the user's desire.

The perceptron presented by Rosenblatt (1957), was the first model to learn the weights which defined two categories given examples of inputs from each category. The function of a perceptron neuron is given in (2.2).

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

The final output of the neuron is decided by an activation function, a function which takes the sum of all inputs multiplied with their respective weights and the bias, i.e. $\mathbf{w}^\top \mathbf{x} + b$, and computes an output, $y = f(\mathbf{w}^\top \mathbf{x} + b)$. The perceptron used a step function for activation, but a multitude of alternatives are in use today, depending on the task at hand. A step function will give a binary response, 0 or 1, a $tanh$ function provides a continuous scaling between -1 and 1, a $sigmoid$ function scales continuously between 0 and 1, and a $relu$ function will return the maximum of the input and 0. Graphs of the $sigmoid$, $tanh$ and $relu$ functions are presented in Figure 2.3 and a visual representation of an artificial neuron is presented in Figure 2.2.

Neurons can be grouped together in layers where the layer takes in an input vector, $\mathbf{x}$, and outputs an output vector $\mathbf{y}$ as a result of a weight matrix, $\mathbf{W}$, a bias vector $\mathbf{b}$, and an activation function, $f(\cdot)$.

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.3}$$

A network is built by passing the input through an input layer to a neuron layer, often called a hidden layer, and ultimately to an output layer. A network may contain only one such hidden layer or it may contain many, where the output of one layer is used as the input of the next. An example of such a network is provided in Figure 2.4.

ANNs represent a complex and nonlinear function. According to the Universal

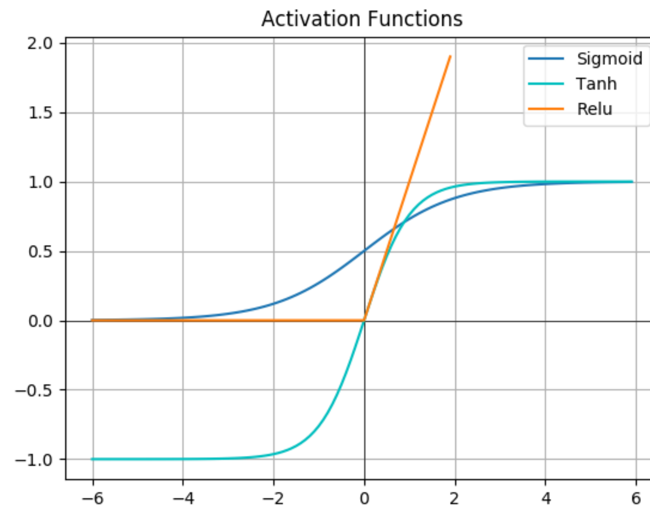


Figure 2.3: Plot of three functions often used as activation functions in ANNs

Approximation Theorem (Cybenko; 1981), a network with a single layer of a finite number of neurons can approximate any continuous function of a finite set of real variables. However, the parameters of the network, the weights, and biases, collectively labeled θ, must be fitted to the task. This process is called training the network and will be discussed later.

2.1.2 Deep Neural Networks

An ANN with more than one hidden layer is called a Multilayer Perceptron or a Deep Neural Network (DNN). It is through the use of such deep architectures and different activation functions that Deep Learning has achieved the impressive results seen in the later years (Silver et al.; 2016; Mnih et al.; 2015; Schmidhuber; 2015; Levine et al.; 2016; Kober et al.; 2013). Examples of their successes are many, such as Silver et al. (2016) and Mnih et al. (2015), who created networks that are able to play games of Go and Atari respectively on superhuman levels, or Lu et al. (2017) who provided a network which can transform sketches into photorealistic images.

Intuitively the notion of having a multitude of layers can be seen as piling simple concepts together in order to create something complex. Where one layer alone may be able to express something simple, building upon that expression may be able to express or deduce more. This works in the same way simple concepts like addition

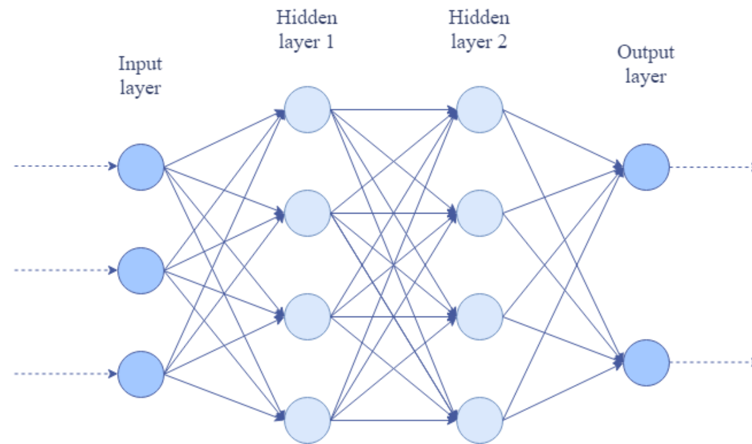


Figure 2.4: A Neural Network structure example where each colored dot represents a neuron in the network and the arrows the weighted connections between them. The input is passed to the network through the input layer, forwarded through a number of hidden layers, and finally, mapped to the output through the output layer. Input neurons are activated through sensors that are perceiving the environment, while other neurons are activated through weighted connections from previously activated neurons (Schmidhuber; 2015). In this example, the network consists of an input layer of size three, two hidden layers, both of size three and an output layer of size two, but these sizes may vary.

and multiplication can be piled to create complex and nonlinear equations which are able to describe highly complex concepts that lie beyond the expressive power of the individual pieces. Another example lies in how simple constructs of Lego pieces can together build large and intricate creations. One piece cannot represent much, but by using many pieces, a wall can be built, and from that, an entire building. On the other hand, the same pieces may be used to instead build a bridge. This example illustrates the expressive power that lies in DNN.

ANNs and deep learning, the process of adapting the weights of the deep NN to make it exhibit the desired behavior (Schmidhuber; 2015), have attracted widespread attention and recognition. As pointed out in Schmidhuber (2015), they have especially found success in tasks of visual nature, like image classification, and have accomplished impressive results in certain areas of reinforcement learning (Mnih et al.; 2015; Silver et al.; 2016; Sutton and Barto; 2017). In our case, their expressive power and compatibility with RL and IRL make them a good candidate for end-to-end learning of a steering model.

2.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), are a special subclass of ANNs in which at least one layer is convolutional, i.e. a layer that uses convolution in place of general matrix multiplication. Convolution is a mathematical operation. E.g. given a measurement function, $x(t)$, dependent on time, t, and a weighting function $w(a)$ where a is the age of the measurement, a weighted sum that provides more relevance to recent measurements could be gained by using the following operator (Goodfellow et al.; 2016):

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.4}$$

This operator is called convolution and Equation (2.4) presents the continuous single dimensional form. Equation (2.5) presents the discrete 2D form of the operator, which is the one used in CNN.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.5}$$

In CNNs, the weights which scale the input in a layer come in the form of kernels, sometimes referred to as filters. A 2D kernel is a matrix where each entry represents a weight of its input similarly to the weights of one fully connected neuron. In fact, convolving a kernel with a size equal to the input would be the same as applying one fully connected neuron to the input. However, these kernels are usually smaller than the input, allowing them to focus on local spacial features.

The input I, e.g. an image, is convolved by a kernel K as defined in Equation 2.5, resulting in an output image S where each entry equals a weighted sum of a piece of the input image. This process is equal to flipping the kernel and then moving it across the image, recording the weighted sum with each step. An example of one such step is illustrated in Figure 2.5. One convolutional layer may apply more than one kernel to its input. Because each convolutional operation outputs one 2D image, a layer may, therefore, produce a number of output 2D images, often called channels, from the same 2D input image.

The smaller size of the kernels, allow them to only account for local information in the input. In addition to this, differently weighted kernels will provide high values

Figure 2.5: An illustrated example of two steps of the convolution of a kernel, K, and an image, I. The convolution in this example is performed with a stride of 1 and without padding the image.

in different cases. E.g. where one kernel may output high values when convolved with a piece of an image that contains a horizontal edge, another may output high values when exposed to vertical edges. Using both of these kernels in one layer, a network may in a later layer combine this information in order to detect corners and more complex forms or textures. The weights and sizes of these kernels thus define a network's output. The sizes and number of the kernels in each layer are decided by the programmer, while the weights of each kernel are learned through training.

CNNs are used for processing data that has a known grid-like topology, such as images, and have proven highly effective in this regard. The application of deep CNN has allowed for breakthroughs in text recognition, scene labeling, image classification, object detection, object tracking, and more (Gu et al.; 2018). The latter two, object detection and object tracking, are especially valuable to us in the task of obstacle avoidance through the use of radar images.

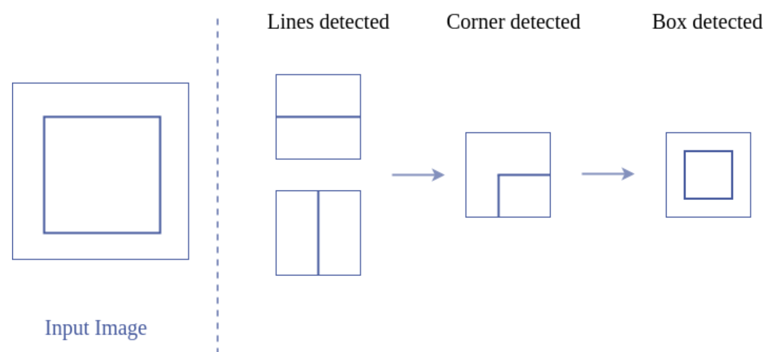


Figure 2.6: Conseptual illustration of the layered detection of a box in a CNN

Max-Pooling

CNNs often combine convolutional layers with max-pooling layers. Max-pooling is an operation in which a gridlike structure is downsampled by choosing the maximum value of a region to represent the region, effectively reducing the size of the grid in the process. In CNNs, pooling helps make the network invariant to small translations of the input, meaning that if an element in an image is moved by a small amount, the output of the pooling layer will mostly stay the same.

Batch Normalization

Another type of layer that often accompanies convolutional layers in CNNs is batch normalization (Ioffe and Szegedy; 2015). These layers normalize their input by subtracting the batch mean and dividing by the batch standard deviation. This mean and standard deviation is learned during the training process. The normalization provides a remedy against the problem of certain neurons overpowering others by leveling the playing field so to speak.

2.1.4 Training

A vital part of DL is the training of the ANNs. The parameters, θ, of the network must be fitted so that the input to the network is processed in a way befitting the network's task, whether this is the classification of an inputted image or generation

of an action based off an inputted observation as is the case in Reinforcement Learning.

The network is trained by repeatedly feeding input data from a training set into the network. For example, in supervised learning, such a training set consists of pairs of input and labels where the labels represent the true output, that which the network is to learn. The actual output of the network is then evaluated through a loss function which determines the error of the output for the given input. Adjustments are then made to the network parameters in order to reduce that loss. These adjustments are often performed using Gradient Descent through back-propagation of the error, as illustrated by this next example.

In the case of image classification, a supervised learning branch of DL, a task may be for an ANN to determine whether its input image depicts a cat or a dog. The image is passed through the network and produces two probabilities at its output, the probability of the image depicting a cat and the probability of it depicting a dog. In training, a loss may be found by calculating the difference in the true label of the image, it being a cat or a dog, and the suggested output. These losses are then back-propagated. This means that the gradientof the prediction error is passed backwards through the network, and the gradients of the error with respect to the weights of the network, are calculated. These gradients can then be used to update the weights of the network. The process is repeated for different inputs drawn from the training set.

2.2 GAN

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. (2014), inspired by two-player zero-sum games. In such a game, the sum of the gains of the two players is always zero and the win of one player is thus exactly matched by the loss of the other. The main idea behind GANs is to use two Artificial Neural Networks (ANNs) and, similarly to the two-player zero-sum game, pit them against each other in order for them to guide each other towards optimal behavior.

GANs are usually comprised of a generator and a discriminator which learn their individual tasks simultaneously. The generator's task is to capture the potential distribution of a set of real samples and generate new data. It thus aims to generate data which looks like it is part of the true set of samples, while in reality, it is

generated from a prompt, e.g. noise. The discriminator's task is to separate the real samples from the data generated by the discriminator as accurately as possible. Because of the nature of its task, the discriminator is often a binary classifier, though this does not need to true in all cases (Finn et al.; 2016b). As pointed out in Wang et al. (2017), the discriminator and generator can both adopt the structure of currently popular ANNs.

A practical example of the dynamics between the generator and the discriminator is one of an art forger and an art dealer. The art forger attempts to gain money by painting copies of expensive paintings while the art dealer attempts to distinguish these copied paintings from the genuine articles. The art dealer will over time improve their ability to distinguish between the copies and the real paintings and in turn, the art forger will have to improve their copies. Eventually, the art forger is able to capture the full distribution of the real paintings and the copies will be indistinguishable from the real paintings. In this scenario, the forger represents the generator, while the art dealer represents the discriminator. Their goals are intertwined such that for one to succeed, the other must fail. GANs can thus be viewed as two competing rivals. The win of one will motivate the other to do better until the roles reverse. This keeps going and the two rivals will this way push each other towards better results.

GANs have received much attention for their applicability and they have especially seen success in the fields of image and vision. Through the use of GANs, networks have learned to generate photorealistic photos of birds and faces, transform photos of scenery taken during summer time into a version during winter time, create high-definition images from low-definition images, and more (Goodfewllow; 2016; Wang et al.; 2017). GANs have also been used for tasks that do not involve images, such as a few cases of usage in Reinforcement Learning and Imitation Learning, though most examples of GAN usage are related to visual tasks (Wang et al.; 2017).

2.2.1 Training

Given a binary discriminator D, and a generator G, parameterized by the parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$ respectively, the training scheme of the GAN is defined by the following equation:

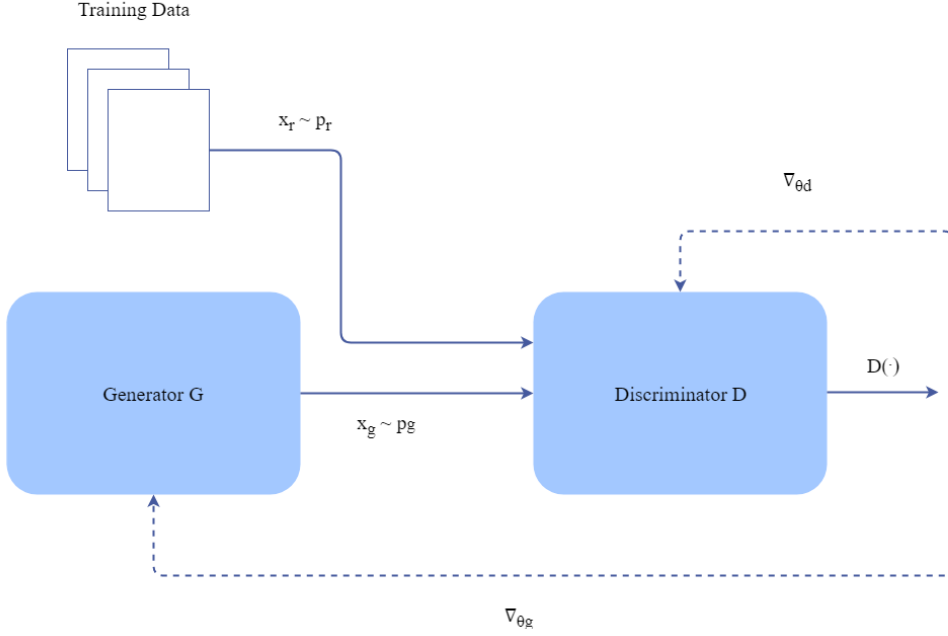


Figure 2.7: Illustration of a Genereative Adversarial Network training scheme

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \quad (2.6)$$

where $p_r(\mathbf{x})$ is the distribution of a training data set, i.e. the real distribution that the generator is tasked to learn, and $p_{\mathbf{z}}(\mathbf{z})$ is the distribution of the noise input to the generator.

In equation (2.6), $log(1 - D(G(\mathbf{z})))$ is the loss function for G. The loss of G is based on D's ability to correctly classify G's generated data, $G(\mathbf{z})$, and will thus be optimal at 0, when D classifies $G(\mathbf{z})$ as a real sample, $D(G(\mathbf{z})) = 1$. The loss function of D is $logD(\mathbf{x}) + log(1 - D(G(\mathbf{z})))$, i.e. to which degree D is able to categorize the real and generated samples correctly, $D(\mathbf{x}) = 1$, $D(G(\mathbf{z})) = 0$. In total, G attempts to minimize equation (2.6), while D attempts to maximize it, resulting in a tug of war between the two. The parameters of D and G are updated in turn, as shown in Algorithm 1. An illustration of a GAN training scheme is shown in Figure 2.7

Algorithm 1: Batch Training of GAN

Input : Data set representing the distribution p_r

1 **while** *training* **do**

2 Sample m generated samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from $p_z(\mathbf{z})$

3 Sample m real samples $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ from $p_r(\mathbf{x})$ Update D:

4

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i-1}^{m} \left[logD(\mathbf{x}^{(i)}) + log(1 - D(G(\mathbf{z}^{(\mathrm{i})}))) \right]$$

Update G:

5

$$\nabla_{\theta_g} \frac{1}{m} \sum_{j-1}^{m} log(1 - D(G(\mathbf{z}^{(j)})))$$

6 **end**

2.2.2 Advantages and Drawbacks

Goodfewllow (2016) point out that GANs enable machine learning to work with multi-modal outputs, i.e. the case of one single input corresponding to different, yet each acceptable, outputs. This is e.g. the case with many steering scenarios such as in our case of avoiding an obstacle, where one may choose to either turn to the left or the right with different degrees. Where approaches such as mean square error may only provide an average over the possible outcomes, which may be unsatisfactory on its own, GANs may understand that there are many possible outputs, each of which is distinctly different. This makes GAN a promising candidate for learning a steering model from demonstrated behavior.

As seen in the previous section, GANs train by simultaneous gradient descent, a process that converges for some games but not all. Goodfellow et al. (2014) showed that simultaneous gradient descent in min-max GAN games converges if the updates are made in function space. As pointed out in Goodfewllow (2016), updates are in practice made in parameter space, making the convexity properties that the proof relies on non-applicable. There are no theoretical arguments that GANs neither converge nor that they do not, and in practice, they often oscillate (Goodfewllow; 2016). One common way of harmful non-convergence in GANs is mode collapse, in

which the generator learns to map several different input values to the same output point(Goodfewllow; 2016). Goodfewllow (2016) point out that while complete mode collapse is rare, partial collapse is a common challenge of GANs.

2.3 Reinforcement Learning

As explained in Sutton and Barto (2017), Machine Learning (ML) is divided into three categories, Supervised Learning, Unsupervised Learning and Reinforcement Learning. In short, Supervised Learning concerns learning using data sets where each input example has a corresponding label or target, corresponding to the ground truth. Because these labels are known, training consists of comparing labels generated by the function in learning to the true labels, similarly to a teacher correcting a student's test. In Unsupervised Learning, the true labels of the examples in the data set are unknown. Algorithms of this kind handles datasets containing many features and learn useful properties of the structure of this dataset. While supervised learning can for example be used to separate rats from mice, unsupervised learning may be used to for example discover clusters in a data set, like correlations between the rodents diet and their activity level. Reinforcement Learning (RL) differs from the other two in that there is no pre-made data set used for training. Instead, training data is collected during the learning process. The supervision in RL is also indirect and often sparse. The control of a robot arm, learning to generate an appropriate action in a specific situation, is a typical Reinforcement Learning task.

Out of the three ML branches, RL is better suited for control tasks such as those considered in this project. In particular, with RL, we can avoid compounding errors, support multi step, forward looking decisions, and do away with the requirement of independently and identically distributed data (Tai et al.; 2016). Because of this, we here focus on this branch of ML. For further reading about supervised and unsupervised learning, we refer to Goodfellow et al. (2016).

This section will present some important topics in Reinforcement Learning (RL). We also introduce the use of Deep Learning (DL), presented in the previous section (Section 2.1), in RL as well as going more into detail of a specific deep RL algorithm which we use in our system. Further details on RL can be found in Sutton and Barto (2017), Gosavi (2009) and François-Lavet et al. (2018). We note that the first two subsections of this section, though written by us, were originally written for a report

as part of a preparatory project (Vedeler; 2018). As the theory has not changed, we have elected to only make slight modifications to these paragraphs.

2.3.1 The core idea of Reinforcement Learning

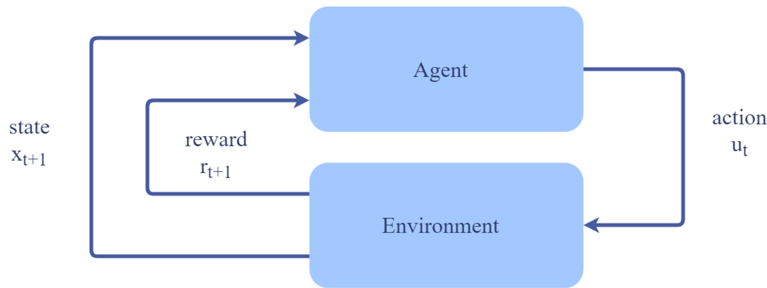


Figure 2.8: An illustration of the agent-environment interaction in Reinforcement Learning.

As defined in Sutton and Barto (2017); "Reinforcement learning is learning what to do — how to map situations to actions — so as to maximize a numerical reward signal." At its core, the idea of Reinforcement Learning (RL) is to approach learning in the highly intuitive way of most organisms we know of, learning by trial and error. In RL, the one making the decisions is called the agent while everything outside of the agent is called the environment The agent is not told what the goal is nor which actions to perform in order to accomplish it. It is simply put into its environment without prior knowledge of its workings. It must then discover the objective and the suitable approach by performing actions and observe the results that they cause. This feedback comes in the form of a numerical value from a given reward function and, like a toddler, given encouragement and scolding, the agent will eventually learn the desired behavior. A simple illustration of the RL process is presented in Figure 2.8.

2.3.2 Marcov Decision Process

The process of the agent interacting with its environment and the resulting reward is, in reinforcement learning, formulated as a Marcov Decision Process (MDP), a tuple $< \mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{R} >$. At each time step $t = 0, 1, 2, 3...$, the agent experiences some state of the environment, $\mathbf{x} \in \mathcal{X}$, and must decide on some action, $\mathbf{u} \in \mathcal{U}(\mathbf{x})$. The state is here defined as a unique characterization of the environment and the process of the task that the agent must solve. In practice, this is often expressed by a collection of

sensory input of the agent, such as e.g. the pose of a robotic arm combined with visual input (Finn et al.; 2016a). The actions can be used by the agent to control the system state. In our case, the case of an USV, such an action could be to apply a change in the steering parameters, e.g. change the heading of the USV. The choice of action results in the agent transitioning into a new state, $\mathbf{x}'$, and receiving a scalar reward, $r \in \mathcal{R}$, as a consequence of this transition. The transition is modeled by a transition function $\mathcal{P}$ which defines the probability, $p(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \in [0, 1]$, of ending up in state $\mathbf{x}'$ after performing action $\mathbf{u}$ in state $\mathbf{x}$. Meanwhile, $\mathcal{R}$, or $\mathcal{R}(\mathbf{x}, \mathbf{u}, \mathbf{x}')$, represents the reward function. It denotes the reward of transitioning from one state, $\mathbf{x}$, to another, $\mathbf{x}'$, with the use of action $\mathbf{u}$. The resulting reward may be positive or negative and is used to guide the agent towards solving the task. The sequence of state and action pairs, $\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, ..., \boldsymbol{x}_n, \boldsymbol{u}_n$ is often called the trajectory, τ.

RL is not alone in relying on a set of states, actions, and an underlying model. Both in this regard and others, RL share similarities with classical optimal control (Kober et al.; 2013). However, as pointed out by Kober et al. (2013), a key difference is the fact that optimal control assumes knowledge about this model is available, while RL does not. RL learns a task without prior knowledge of the inner workings of the environment in which it resides. $\mathcal{P}$ and $\mathcal{R}$ are both unknown to the agent. Only the string of states-actions pairs and the resulting rewards the repeated interactions produces, $\boldsymbol{x}_0, \boldsymbol{u}_0, r_0, \boldsymbol{x}_1, \boldsymbol{u}_1, r_1, \boldsymbol{x}_2, ...$, is observable by the agent. RL's ability to learn without a given model makes it an appealing choice when it comes to our task of steering an USV, as the dynamics of its environment is challenging.

An assumption of MDPs is that the current state provides enough information in order to choose the optimal action. In other words, the past states can be ignored when deciding the action. While this is unproblematic in some cases, an argument could be made that this does not properly make use of the temporal information that may be needed to perform the task in an optimal way (Chi and Mu; 2017). A task can be episodic, meaning it has an ending state that stops the process, or non-episodic, in which case the task could continue indefinitely.

2.3.3 Optimal Policy

As previously mentioned, RL learns how to map situations to actions. This mapping function is, in RL, called a policy, $\pi : \mathcal{X} \rightarrow \mathcal{U}$. When the agent acts in the environment, it does so based on the state to action mapping its current policy

provides it with. Thus, in order to solve its task, the agent's policy must be trained so that it maps a given state to an appropriate action, one that either successfully achieves the goal of the task or puts the agent a step closer to doing so. The goal is to learn a policy that solves the given task in an optimal way, i.e. to learn an optimal policy. In RL, the optimal policy, π^*, is learned by interacting with the environment and optimizing the policy of an agent such that it maximizes the reward it receives for its behavior.

The V-value function, $V^\pi(\mathbf{x})$ is a measure of the expected return of rewards given the agent is in state $\mathbf{x}$ and will follow the policy π for all future states. Thus, the V-value function is a measure of the value of a state under a specific policy. There are several ways of defining how future rewards should account for the current value of the state, three of which are named infinite horizon, finite horizon and discounted infinite horizon. In the infinite horizon approach all future rewards are used in calculating the value, while the finite horizon approach only accounts for the rewards within a certain number of time steps. As somewhat of a compromise between these two, the discounted infinite horizon approach accounts for all future rewards, but scales them such that the rewards that lie closer in time are regarded as more important. Using the discounted infinite horizon approach, the V-value function can be expressed as in equation (2.7) and equation (2.8), where the latter highlights the recursive property of the function.

$$V^\pi(\mathbf{x}) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | \mathbf{x}_t = \mathbf{x}, \pi\Big] \tag{2.7}$$

$$= \sum_{\mathbf{x}'} \mathcal{P}(\mathbf{x}, \pi(\mathbf{x}), \mathbf{x}')(\mathcal{R}(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma V^\pi(\mathbf{x}')) \tag{2.8}$$

In these equations, $\gamma \in [0, 1]$ denotes the discount factor. The closer to zero γ is, the more the agent values the immediate rewards over those further into the future. When $\gamma = 1$, the approach equals that of infinite horizon.

In addition to the V-value function, we have the Q-value function, $Q^\pi(\mathbf{x}, \mathbf{u})$. Much like the V-value function, the Q-value is a measure of the value of a state under a certain policy. However, in the Q-value function a separate action is provided. The Q-value represents the expected reward for the agent in state $\mathbf{x}$ provided it performs action $\mathbf{u}$ and from the next state, $\mathbf{x}'$, follows the policy. The action, $\mathbf{u}$, may be one

that does not follow from the policy, meaning the Q-value can be used to evaluate the value of an state-action pair for actions outside of its policy. Thus, the optimal policy can be obtained directly from the optimal Q-value function

$$\pi^*(\mathbf{x}) = \underset{\mathbf{u}\in\mathcal{U}}{\operatorname{argmax}}\, Q^*(\mathbf{x}, \mathbf{u}) \tag{2.9}$$

where $Q^*(\mathbf{x}, \mathbf{u}) = \max_{\pi\in\Pi} Q^\pi(\mathbf{x}, \mathbf{u})$ is the optimal Q-value function.

In addition to these two, we have the advantage function:

$$A^\pi(\mathbf{x}, \mathbf{u}) = Q^\pi(\mathbf{x}, \mathbf{u}) - V^\pi(\mathbf{x}) \tag{2.10}$$

The advantage describes how good the action, $\mathbf{u}$, is compared to following the policy π directly. In other words, it measures the value of changing the current policy so that it chooses the action $\mathbf{u}$ for the state $\mathbf{x}$.

These value functions are not readily known to the actor, and must thus be estimated during training. Like the policy, they must be represented by some form of function approximation that can be updated and optimized during training. One type of such representation which has performed well in this regard is the use of Artificial Neural Networks (ANNs).

2.3.4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines RL and Deep Learning (DL), using Artificial Neural Networks (ANNs) to express their policies. As previously mentioned, the use of DRL have accomplished impressive results in certain areas of reinforcement learning (Mnih et al.; 2015; Silver et al.; 2016; Sutton and Barto; 2017; Martinsen and Lekkas; 2018). Also in our case, the expressive power and compatibility with RL and make it a good candidate for end-to-end learning of a steering model.

Value based methods learns a value function and uses this function as a basis for the policy. Value based methods, most notably varieties of the Deep Q-networks (DQN) algorithm introduced by Mnih et al. (2015), have accomplished impressive tasks, like reaching super-human performance in Atari games. However, as explained in

François-Lavet et al. (2018), these types of algorithms are not well-suited for large and/or continuous action spaces and they cannot explicitly learn stochastic policies.

Another group of DRL methods are the Policy Gradient Methods (PGMs), which learn a parameterized policy, $\pi_{\boldsymbol{\theta}}$, that selects actions without consulting an estimated value function. A value function may still be used as a means to learn the policy, a subgroup of methods called actor-critic methods, but they are not required for the selection of the action. PGMs optimize a performance objective, $J(\pi_{\boldsymbol{\theta}})$, by finding a good policy, $\pi_{\boldsymbol{\theta}}$, using variants of stochastic gradient ascent with respect to the policy parameters $\boldsymbol{\theta}$. The performance objective is typically the expected cumulative reward (François-Lavet et al.; 2018). With α denoting the step size, or the learning rate, the general gradient ascent update rule is given in equation (2.11) (Sutton and Barto; 2017)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\pi_{\boldsymbol{\theta}_t}) \tag{2.11}$$

Actor-critic methods get their name from their two parts: the actor and the critic (Konda; 2002). The actor refers to the policy while the critic is the estimate of the value function. Both can be represented by ANNs (Mnih et al.; 2016). When updating the policy, actor-critic methods bases the changes on feedback from the critic.

Stochastic Policies

Policy Gradient Methods (PGMs) are usually performed on-policy, i.e. updating only using data collected with the most recent policy, as opposed to off-policy, meaning the update can be made on any collected data, even if the choices which led to the collected data does not adhere to the current policy. In practice, policies used in PGMs are thus usually non-deterministic so as to allow for exploration, resulting in the use of stochastic policies. Two of the most common types of stochastic policies are categorical policies, which can be used in discrete action spaces, and multivariate Gaussian policies, which are used in continuous action spaces. While the latter of the two is perhaps the more intuitive choice for use in a continuous task such as ours, we have performed testing with both and thus devote a short introduction to both of these forms of policy representation.

Multivariate Gaussian Policies, $\mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}))$ are described by a mean vector, μ,

and a covariance matrix, Σ. μ is calculated for the observed state by passing the observation through a neural network. The same may be true for Σ, but this is not always the case. An action sample, corresponding to the observed state can then be drawn from this distribution and training of the policy is equivalent to training the ANNs which produce the mean and, possibly, the covariance.

While Multivariate Gaussian Policies may produce actions that are continuous, categorical policies work with discrete action spaces. They map the input observation to one out of a fixed number of discrete categories where each of these categories have a separately specified, learned probability of being chosen. It is worth noting that even though the output of such a policy, the action, is discrete, the same need not be the case for the observation or state space. A discrete policy may still be used to solve a continuous task.

Policy gradient methods optimize their policy by promoting the actions which result in high rewards, making changes so that these favorable actions gain a high probability of occurring. Likewise, the actions which lead to low rewards are to be pushed towards a low probability of occurring.

2.3.5 Trust Region Policy Optimization

Policy gradient methods adjust the policy by updating with a step in the direction which yields the steepest ascent in expected reward. However, deciding the size of the step, often referred to as the learning rate, is not always a trivial matter. A large step may yield fast improvement, but it may also cause the update to step over the intended optima which may delay or even prevent convergence. In the worst case, the resulting policy could become destructive for the agent or its environment during training. On the other hand, an overly cautious step size may prolong the training time. This can be illustrated by a hiker walking towards the top of a cliff: Stepping too far will make them fall off the edge while taking too short steps will slow down the progress. An additional issue is the fact that the optimal step size, or learning rate, is not necessarily consistent. A learning rate with high performance at one point of the learning progress might prove disastrous in another.

Trust Region Policy Optimization (TRPO) (Schulman et al.; 2015) handles this issue by taking the largest step possible while still constraining the policy changes, limiting the parameter changes with a constraint on the difference between the

new and the old policy. In order to do this, TRPO uses a minorization-maximation (MM) algorithm (Hunter and Lange; 2004), trust region, importance sampling, and a conjugate gradient algorithm. The method, described in detail in Schulman et al. (2015), will be summarized in the following paragraphs.

The optimization objective

Let $\eta(\pi)$ express the expected discounted reward of the stochastic policy π. This is the objective that is to be optimized in order to attain an optimal policy.

$$\eta(\pi) = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots} \Big[\sum_{t=0}^{\infty} \gamma^k r_t \Big] \tag{2.12}$$

where r_t is the reward for state $\mathbf{x}_t$, $\mathbf{x}_0$ is the initial state sampled from the the distribution of initial states p_0, $\mathbf{u}_t$ is the action provided by the policy $\mathbf{u}_t \sim \pi(\mathbf{u}_t \,|\, \mathbf{x}_t)$, and the next state $\mathbf{x}_{t+1}$ results from the transition function $\mathbf{x}_{t+1} \sim \mathcal{P}(\mathbf{x}_{t+1} \,|\, \mathbf{x}, \mathbf{u})$

As proved in Kakade and Langford (2002), the expected return of another policy $\tilde{\pi}$ can be expressed by the accumulated advantage over π.

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\mathbf{x}_0, \mathbf{u}_0, \ldots \sim \tilde{\pi}} \Big[\sum_{t=0}^{\infty} A_\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big] \tag{2.13}$$

Where $A(\pi)(\mathbf{x}_t, \mathbf{u}_t)$ is the expected advantage, as described in equation (2.10).

By defining p_π as the discounted visitation frequencies, i.e. the frequency of which the states are visited under the policy π, $p_\pi(\mathbf{x}) := \mathcal{P}(\mathbf{x}_0 = \mathbf{x}) + \gamma \mathcal{P}(\mathbf{x}_1 = \mathbf{x}) + \gamma^2 \mathcal{P}(\mathbf{x}_2 = \mathbf{x}) + \ldots$, equation (2.13) can be expressed as a sum over states instead of time steps.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{\mathbf{x}} p_{\tilde{\pi}}(\mathbf{x}) \sum_{\mathbf{u}} \tilde{\pi}(\mathbf{u} \,|\, \mathbf{x}) A_\pi(\mathbf{x}, \mathbf{u}) \tag{2.14}$$

Because the visitation frequency cannot be negative, this confirms that a policy update $\pi \to \tilde{\pi}$ is guaranteed to either increase or retain the policy performance $\eta(\pi)$ if its expected advantage at every state is nonnegative, $\sum_{\mathbf{u}} \tilde{\pi}(\mathbf{u} \,|\, \mathbf{x}) A_\pi(\mathbf{x}, \mathbf{u}) \geq 0$. However, as pointed out by Schulman et al. (2015), when the components of (2.14)

are estimations, as is the case in most practical tasks, approximations and estimation errors makes the appearance of negative advantage values unavoidable.

Equation (2.14) is difficult to optimize directly due to its dependency on $p_{\tilde{\pi}}(\mathbf{x})$. Schulman et al. (2015) thus provides an alternative, local approximation to η which uses the visitation frequency of the old policy p_π instead of having to account for the changes that the new policy will introduce in the visitation frequency $p_{\tilde{\pi}}$.

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{\mathbf{x}} p_\pi(\mathbf{x}) \sum_{\mathbf{u}} \tilde{\pi}(\mathbf{u} \mid \mathbf{x}) A_\pi(\mathbf{x}, \mathbf{u}) \tag{2.15}$$

Though L_π is an approximation of η, when the policy is parameterized and differentiable such as an ANN, results of Kakade and Langford (2002) show that L_π matches η to first order. Because of this, a step that improves L_π, will also improve η.

The step size

In order for optimization of L_π to guarantee optimization policy improvement, the step size must be sufficiently small. Building on the work done in Kakade and Langford (2002), Schulman et al. (2015) shows that the following holds for the expected discounted reward η:

$$\begin{aligned} \eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C D_{KL}^{max}(\pi, \tilde{\pi}), \\ \text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2} \end{aligned} \tag{2.16}$$

where $\epsilon = \max_{\mathbf{x},\mathbf{u}} |A_\pi(\mathbf{x}, \mathbf{u})|$. For details and proofs for this result we refer to Schulman et al. (2015).

In equation (2.16), $D_{KL}^{max}(\pi, \tilde{\pi}) = \max_{\mathbf{u}} D_{KL}(\pi(\cdot \mid \mathbf{x}) || \tilde{\pi}(\cdot \mid \mathbf{x}))$ is the maximum KL-divergency. The KL-divergency represents the dissimilarity between two distributions (Zhang; 2017). Thus, it is intuitively sound that it would be usable in a lower bound.

By maximizing $L_\pi(\tilde{\pi}) - C D_{KL}^{max}(\pi, \tilde{\pi})$, the true objective η is guaranteed to be non-decreasing, thus training will cause monotonical improvement. This is, in fact,

a type of minorization-maximation (MM) algorithm (Hunter and Lange; 2004), where the maximation of an objective is achieved by estimating a lower bound that approximates the true objective at the current step but which is easier to use in calculations. At each step, this lower bound is then optimized and because the true objective is always larger than, the improvement to this true objective is guaranteed to be at least as great as the improvement made to the estimated bound.

Applying the results of equation (2.16) in the optimization of a policy parameterized by the parameters $\boldsymbol{\theta}$, yields the following update rule

$$\underset{\boldsymbol{\theta}}{\text{argmax}}[L_{\boldsymbol{\theta}_{old}}(\boldsymbol{\theta}) - CD_{KL}^{max}(\underset{old}{\boldsymbol{\theta}}, \boldsymbol{\theta})] \tag{2.17}$$

where $\boldsymbol{\theta}$ denotes the parameters that make up the policy, and $\boldsymbol{\theta}_{old}$ denotes the previous policy parameters which are to be improved upon.

Trust Region

Schulman et al. (2015) remarks that, in practice, using the update rule of equation (2.17) results in excessively small step size because of the penalty constant C, slowing down the learning of the policy. In order to allow for larger steps while still keeping the algorithm robust, Schulman et al. (2015) suggests the use of a trust region.

When applying a trust region, the maximum step size is determined based on the local accuracy, or level of trust, of the estimations in the objective function. This maximum step size yields a trust region. Once the trust region has been established, the objective can be maximized and a step can be taken to the maxima within the region.

Schulman et al. (2015) bases their trust region on the average KL-divergence, meaning the trust region is large for small changes in policy parameters and small for large changes in policy parameters. This results in the update rule of the TRPO algorithm: (Schulman et al.; 2015)

$$
\begin{aligned}
&\underset{\theta}{\operatorname{argmax}}\, L_{\theta_{old}}(\theta) \\
&\text{subject to } \bar{D}_{KL}^{p_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta
\end{aligned}
\tag{2.18}
$$

where $\bar{D}_{KL}^{p_{\theta_{old}}}(\theta_{old}, \theta)$ is the average KL-divergence. The use of average KL-divergence rather than true KL-divergence is motivated by the lesser number of constraints.

Intuitive summary

Intuitively, the expected improvement of a new policy can be approximated locally around the current policy, but with an accuracy which decreases when the new and current policy diverge. Establishing an upper bound for this error using KL-divergence yields a region in which the approximation can be trusted and thus when optimizing the local approximation within this trust region, the step is guaranteed to improve the policy. Repeating this process iteratively will eventually result in an optimal policy. TRPO also applies importance sampling in order to increase sampling efficiency and conjugate gradient optimization for computational efficiency.

Exploitation and Exploration

Policy methods can be categorized into on-policy methods and off-policy methods. While on-policy methods choose their actions from the current policy to be optimized during training, off-policy methods are free to use actions which do not come from the policy they are currently optimizing. This allows them e.g. to use random actions at random points during training. This randomness makes the agent explore its environment outside of the current path of the policy and perhaps discover better alternatives which can then be incorporated into the policy.

Because TRPO trains its policy, which is stochastic, in an on-policy way, it explores by sampling actions according to the latest version of its policy. Contrary to off-policy methods, TRPO, an on-policy method, explores by sampling actions according to the latest version of its policy. Because its policy is stochastic, some

randomness will be present, allowing for exploration of the environment. However, this randomness of its selected actions depends on initial conditions and the training procedure. The policy also typically becomes less random over the course of the training as a result of the TRPO rule which encourages the exploitation of the rewards which it has already found. This results in a policy that optimizes well over the rewards that it has discovered, however, it may also cause it to get trapped in a local optimum.

2.4 Inverse Reinforcement Learning

Reinforcement Learning (RL) relies heavily on feedback from a reward function as it provides the agent with the only exterior feedback on its behavior. In order for an agent to learn, a reward function must not only be known by the programmer, it must also capture the nature of the task. Formulating such a reward function is no trivial matter however, as many tasks may be difficult to describe in such a manner. In many cases, the reward function may be overly complicated, abstract or simply unknown to the programmer.

Inverse Reinforcement Learning (IRL) is a response to the issue of constructing a reward function. It is a relatively new field, in which the focus lies on the reward function, not the policy (Zhifei and Joo; 2012). In this section, we provide a short introduction to the IRL principles in order to build the necessary vocabulary for the next section.

2.4.1 The concept of IRL

As already mentioned, the need for a pre-constructed reward function is a drawback of RL because the design of such a function may be difficult to achieve, especially for complex tasks. E.g. in the case of driving a vehicle, even though we perform the task, there is no guarantee we will be able to accurately represent it in the form of a reward function.

In IRL, the construction of a reward function is achieved through observing an expert perform the task. The agent is given a set of demonstrations, $\mathcal{D} = \{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, ..., \boldsymbol{\tau}_n\}$, performed by an expert, where each demonstration consists of a set of state-action pairs, $\boldsymbol{\tau}_i = \{(\mathbf{x}_0, \mathbf{u}_0), (\mathbf{x}_1, \mathbf{u}_1), ...(\mathbf{x}_k, \mathbf{u}_k)\}$. The usage of recorded expert behavior is

a trait IRL shares with the behavioral cloning techniques. In behavioral cloning, the goal is to replicate these trajectories directly from the demonstrations, usually through supervised learning. However, this approach usually suffers from an insufficient number of training samples and poor generalization (Zhifei and Joo; 2012). Unlike behavioral cloning, IRL does not attempt to learn the behavioral directly, instead the goal is to infer the motivation behind the demonstrations or the underlying goal of the task. It learns the reward function that the expert would have used to achieve its performance.

As with RL, the reward function is assumed to able to succinctly represent the task and lead to optimal behavior (Zhifei and Joo; 2012). Thus, assuming the expert is performing the task, finding the reward function under which the expert behavior is optimal, IRL may recover a reward function that describes the task itself. Such a reward function contains the expert's experience in both the task and the dynamics of the environment in which it is performed. The latter part is especially important in situations with complex dynamics, such as in robotics, where extensive knowledge of such dynamics would be required in order to manually craft a descriptive reward function.

2.4.2 IRL algorithms

Two important advances in IRL were made with the papers on Maximum Entropy IRL (Ziebart et al.; 2008) and Deep Maximum Entropy IRL (Wulfmeier et al.; 2015), which introduced the use of entropy maximation and use of Artificial Neural Networks as a parametrization of the reward function respectively. For the sake of brevity, the specifics of these algorithms are left out of this thesis. However, for the sake of completion, a summary of them is included in Appendix A.1 and A.2.

An issue with learning a reward function is that it is difficult to evaluate the function directly. Most IRL algorithms perform this evaluation by optimizing a policy for the current iterations reward function, before comparing the performance of this policy with the expert demonstrations. Thus most such IRL algorithms contain an RL training loop which learns an optimal policy for every iteration of the IRL training loop. As pointed out in Finn et al. (2016a), this RL task can be a costly and complex task in itself, making such IRL algorithms expensive to run.

Guided Cost Learning (GCL) (Finn et al.; 2016a) differs from most IRL algorithms

in that it improves on the reward function inside a policy optimization loop instead of learning a policy in a reward function optimization loop. Because the algorithm does not complete a full RL procedure at each training iteration, the computational efficiency is drastically improved. In addition to this, the policy optimization of GCL fits a model to the dynamics, a model that the reward function optimization can benefit from. The algorithm, which for completion is summarized in Appendix A.3, learns both a policy and a reward function, and may thus also fall under the category of Imitation Learning which will be covered in the next section.

2.5 Imitation Learning

Imitation learning (IL) techniques aim to mimic human behavior in a given task, specifically by observing a teacher demonstrate this task (Hussein et al.; 2017). The goal of IL is not simply to reproduce the specific motions of the demonstrations but to learn a policy that can be generalized to unseen scenarios. The combination of Inverse Reinforcement Learning (IRL), to learn a reward function, and Reinforcement Learning (RL) to learn a policy from the reward function can thus be categorized as IL techniques.

In this section, we will compare IL, specifically the technique of combining IRL and RL, with the Deep Learning technique of Generative Adversarial Networks (GANs) introduced in Section 2.2. We will then proceed to summarize GAIL, an IL algorithm which exploits this connection.

2.5.1 GAN and Imitation Learning

In Section 2.2 we saw that in GAN, a generator, G, learns to imitate a (possibly unknown) distribution and that it does so by receiving feedback on its performance from a discriminator, D. The discriminator, meanwhile learns to distinguish between samples generated by G and samples of the real distribution represented by a data set. This relationship between D and G is highly similar to the relationship between a policy and a reward function in combined RL and IRL such as e.g. the GCL algorithm Finn et al. (2016a). The policy, which takes a state as input and outputs an action, should learn to imitate the distribution of the state-action pairs that the expert is working under and can thus assume the role of a generator in a GAN. The

reward function that the IRL part of the technique trains may then be represented by a discriminator, and thus provide feedback to the policy on whether or not it behaves as the expert would. One of the first to use this GAN representation of combined RL and IRL was Ho and Ermon (2016), through the GAIL algorithm.

2.5.2 GAIL

The similarities between the combination of IRL and RL and the combination of a discriminator and generator in GAN are many, as pointed out in both Ho and Ermon (2016) and Finn et al. (2016b). Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon; 2016) uses these similarities in order to learn an optimal policy using expert demonstrations. According to its authors, GAIL can scale to large state and action spaces.

In true GAN fashion, GAIL is a two-step algorithm. GAIL first updates the discriminator network, D, with an ADAM (Kingma and Ba; 2014) gradient step based on its ability to discern between samples from expert demonstrations and from the policy in training. This is followed by an update in the policy, using a TRPO updating step before the process is repeated. The use of TRPO as the updating procedure limits the change in policy, ensuring that a new policy does not stray too far from the next, as explained in Section 2.3.5. As explained in Ho and Ermon (2016), this step scheme ensures that divergence does not occur due to high noise estimating the gradient. Because the discriminator, which assumes the role of a reward function in an RL algorithm, is changing with every step the use of a constrained optimization step such as TRPO is crucial. If large steps were to be taken based on noisy and/or incorrect rewards, the policy learning could diverge or, in the case of real physical systems, adapt potentially dangerous behavior for the agent and its environment.

Algorithm 2(Ho and Ermon; 2016) shows the GAIL algorithm, denoting τ as the sequence of state-action pairs through an episode. While the method of parallel training of the networks is the same, the update rules are slightly different from the GAN algorithm presented in Algorithm 1. This will be explained shortly.

The Mathematics of GAIL

Similar to GAN, as a method of learning a generator that can imitate the distribution

Algorithm 2: GAIL

Input : Expert trajectories $\tau_E \sim \pi_E$

1 **for** $i = 0, 1, 2...$ **do**

2 Sample m generated samples $\{\tau^{(1)}, ..., \tau^{(m)}\}$ from π_{θ_i}

3 Update discriminator parameters from $\mathbf{w_i}$ to $\mathbf{w_{i+1}}$:

$$\mathbb{E}_{\tau_i}\left[\nabla_{\mathbf{w}} log(D_{\mathbf{w}}(\mathbf{x}, \mathbf{u}))\right] + \mathbb{E}_{\tau_E}\left[\nabla_{\mathbf{w}} log(1 - D_{\mathbf{w}}(\mathbf{x}, \mathbf{u}))\right] \tag{2.19}$$

4 Take a policy step from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$, using the TRPO rule with reward function $log(D_{\mathbf{w_{i+1}}}(\mathbf{x}, \mathbf{u}))$:

$$\begin{aligned} &\mathbb{E}_{\tau_i}\left[\nabla_{\theta} log \pi_{\theta}(\mathbf{x} \mid \mathbf{u}) Q(\mathbf{x}, \mathbf{u})\right] - \lambda \nabla_{\theta} H(\pi_{\theta}), \\ &\text{where } Q(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbb{E}_{\tau_i}\left[log(D_{\mathbf{w_{i+1}}}(\mathbf{x}, \mathbf{u})) | \mathbf{x_0} = \bar{\mathbf{x}}, \mathbf{u_0} = \bar{\mathbf{u}}\right] \end{aligned} \tag{2.20}$$

5 **end**

represented by some data set, IL can be described as a method of finding a policy that matches an experts occupancy measures, i.e. a distribution of state-action pairs which are generated by a policy π (Ho and Ermon; 2016). Mathematically, the occupancy measure is defined:

$$p_{\pi}(\mathbf{x}, \mathbf{u}) = \pi(\mathbf{u} \mid \mathbf{x}) \sum_{t=0}^{\infty} \gamma^t \mathcal{P}(\boldsymbol{x}_t = \mathbf{x} \mid \pi) \tag{2.21}$$

where $\mathcal{P}$ represents the dynamic model of the environment, the transition function, and γ is the discount factor.

GAIL uses a form of apprenticeship learning, a sub-class of IL learning methods. In traditional IRL, the goal is to find a function for which the expert's policy is the unique optimal solution so that the use of RL will yield the same policy. In apprenticeship learning, the goal is not to exactly match the policy of the expert, but to find one that solves the task at least as well, or even better, than the expert. In this regard, the task of IL can be expressed as finding a policy that matches the experts occupancy measure and not its policy directly.

$$\min_{\pi} -H(\pi) + \max_{c \in C} \mathbb{E}_{\pi}[c(\mathbf{x}, \mathbf{u})] - \mathbb{E}_{\pi_E}[c(\mathbf{x}, \mathbf{u})] \tag{2.22}$$

where $H(\pi)$ is the γ-discounted causal entropy, π_E is the expert policy and π is the policy which is to be trained. Here, $c \in C$ is a cost function, the negative of the term reward, $c = -r$. This optimization is equivalent to performing RL followed by IRL with a cost regularizer, ψ, which forces the IRL procedure to recover a cost function that allows the expert policy but is not restricted to it specifically.

Using this, Ho and Ermon (2016) developed a new IL algorithm which finds a policy whose occupancy measure minimize the divergence from the occupancy measure of the expert. They do this by treating the causal entropy, $H(\pi)$, as a regularizer for the policy, controlled by $\lambda \geq 0$ and introducing a regularizer for the cost in the form of $\psi_{GA}(c)$. Expressing the difference in occupancy measures of the expert and the agent as a Jensen-Shannon divergence $D_{JS}(p_\pi, p_{\pi_E} = D_{KL}(p_\pi||(p_\pi + p_E)/2) + D_{KL}(p_E||(p_\pi + p_E)/2)$ this IL algorithm is mathematically expressed:

$$\underset{\pi}{\text{minimize}}\, \psi^*_{GA}(p_\pi - p_{\pi_E}) - \lambda H(\pi) = D_{JS}(p_\pi, p_E) - \lambda H(\pi) \tag{2.23}$$

Thus, the cost regularizer ψ_{GA} must be optimized. Using this, the policy can be optimized such that it causes the least divergence from the occupancy of the expert.

Ho and Ermon (2016) expressed the cost regularizer, such that its optima can be found by the following maximation problem

$$\psi^*_{GA}(p_\pi - p_{\pi_E}) = \max_{D \in (0,1)^{\mathcal{X} \times \mathcal{U}}} \mathbb{E}_\pi[log(D(\mathbf{x}, \mathbf{u}))] + \mathbb{E}_{\pi_E}[log(1 - D(\mathbf{x}, \mathbf{u}))] \tag{2.24}$$

where D is a discriminator who's maximum range of output lies in the range of $(0, 1)^{\mathcal{X} \times \mathcal{U}}$. This formulation is highly similar to the GAN formulation of the discriminator from Section 2.2 and is indeed used for the update step (2.19) for the discriminator in the practical algorithm in Algorithm 2.

The full, practical algorithm of GAIL is thus to find the saddle point (π, D) of the expression

$$\mathbb{E}_\pi[log(D(\mathbf{x}, \mathbf{u}))] + \mathbb{E}_{\pi_E}[log(1 - D(\mathbf{x}, \mathbf{u}))] - \lambda H(\pi) \tag{2.25}$$

which is done by performing Algorithm 2.

The Qualitative Characteristics of GAIL

According to Ho and Ermon (2016), the algorithm is in general quite sample efficient in terms of expert data but requires a number of environmental interactions comparable to the TRPO algorithm. Because the TRPO algorithm is such a large part of GAIL, it is intuitive that it will inherit some of its characteristics. Because GAIL is a GAN method, the same is true for the characteristics of GAN. One example is the previously mentioned ability to handle multi-modal data, an important factor in USV steering.

GAIL is a model-free method, neither requiring a model beforehand nor fitting a model of the dynamics during training. As mentioned in Ho and Ermon (2016), this means that GAIL generally needs more environmental interaction than methods which use such models, such as e.g. GCL (Finn et al.; 2016a). On the other hand, it also means that GAIL can be used in cases where the dynamics are unknown or complex without suffering from potentially ill-fitted models.

It is worth noting that while the discriminator, D, gives feedback to a policy that is learned through an RL updating scheme, D is not a reward function as defined in RL. D does not give feedback that indicates how valuable an action or transition into the next state is for the goal, it returns a value corresponding to its prediction of the sample being generated or drawn from the expert respectively.

2.6 Unmanned Surface Vehicle

The USV used in our project is owned by the Norwegian Defence and Research Establishment (FFI). The boat is of 10.5 meters length and 3.5 meters width and its sensors include a pulse compression radar and a LiDAR. The USV moves through the water by the use of two waterjet based thrusters. Though the USV is a complex work of engineering, we do not need details on its workings for our use. This is because the GAIL system we have implemented is a model-free Imitation Learning (IL) algorithm. However, we do need a basic understanding of radar data measures, which is provided below.

Figure 2.9: Radar image from taken through the use of the USVs radar. The object to be avoided is represented by the oval dot in the lower middle of the image. This indicates that the object is straight ahead of the USV.

2.6.1 Radar

One of the sensors that the USV is equipped with is a Radar. A radar sensor measures distance through the use of short pulses of radio waves. Objects reflect the waves and by recording the direction of these reflections and the time it took for them to return, a distance measure can be acquired. The radar measures the distance to objects in an area of a certain radius. The radius of the area, i.e. the maximum viewing distance, depends on the specific radar. The same is also true for the angle of view. While some radars provide a full 360 degrees view of their surroundings, others provide only a section of this. The USV radar used in this project has a view of 295 degrees, a circle sector spanning from 147.5 degrees to the right of the USVs heading and to 147.5 degrees to the left of the USVs heading.

The measurements can be represented in the form of images, such as the one presented in Figure 2.9. Detected objects are shown in white and their distance from the sensor is given by their distance from the bottom line of the image. The data itself was collected through rotation, by repeatedly sending out radio wave pulses and collecting the reflections of the area in a section, called radar spoke. However, in this image format, each of these spokes is represented as a column of the image grid, making the image a panoramic view of the USVs surroundings. An illustration of a radar spoke and its representation as a column in an image is shown in Figure 2.10

In addition to the measured distances, the radar of our USV also records the heading of the USV at the time of the measurement. Both the distance image and the heading measurements are relevant for this project as they contain information that a human captain would base their steering decisions on during object avoidance.

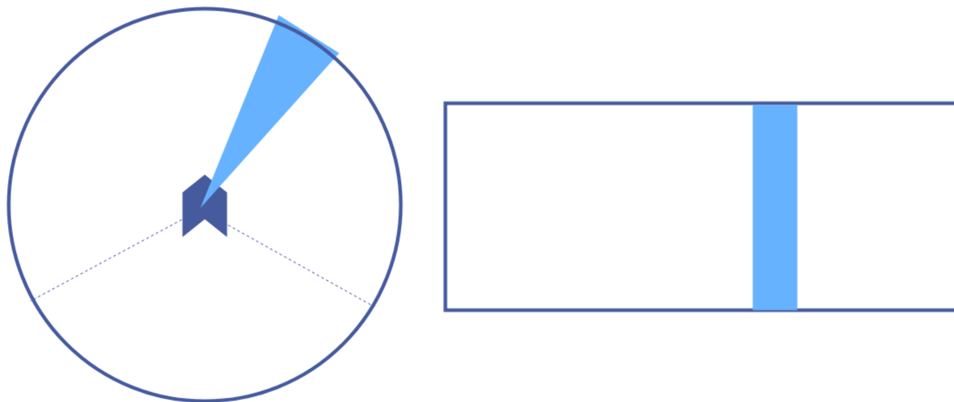


Figure 2.10: Simplified illustration of a radar spoke as a segment of a circle and its representation as a column in an image

Chapter 3

Method

Our goal is to learn an end-to-end steering model that can map sensor measurements to steering actions without requiring insight into the system's dynamics. In order to accomplish this, we consider Reinforcement Learning (RL) and Imitation Learning (IL). The GAIL algorithm combines RL with Inverse Reinforcement Learning (IRL) to form an IL algorithm with the clever training tactic of a GAN network. Using neural networks, GAIL is reported to be able to learn a policy from a relatively small set of demonstrations in a large state space completely model free. Motivated by this, we implemented a GAIL system for use in obstacle avoidance with an Unmanned Surface Vehicle (USV) and performed experiments using full IL. For comparison, we also implemented an RL system that learns using the TRPO algorithm and a handcrafted reward function, which does require more insight into the specifics of the tasks than IL. The RL only approach is still a model-free alternative to the traditional, modularized USV steering system, and it can be used in an end-to-end fashion as well. We implemented our system by using TRPO from OpenAI:SpinningUp (2018) and an implementation of the GAIL discriminator updating scheme from Fu (2018). We altered them to meet the needs for a full GAIL system and to accommodate for the specifications of our task.

A modularized illustration of the full system is presented in Figure 3.1. In this chapter, we will present the different modules as well as important aspects of our method. We present different test set-ups, including two different choices of observations and a set-up for training the full Imitation Learning system and one which uses solely Reinforcement Learning. Comparison and discussion of the

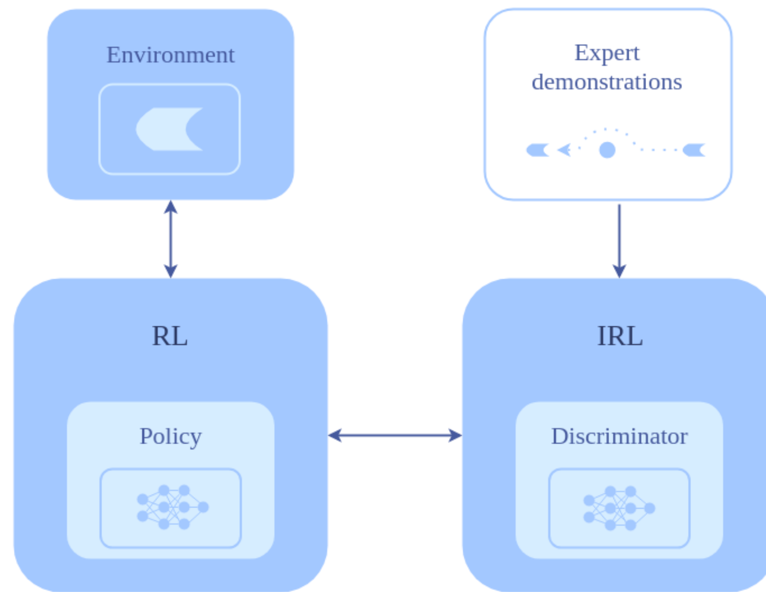


Figure 3.1: Simplified illustration of the Imitation Learning system combining Reinforcement Learning and Inverse Reinforcement Learning.

results are presented in the next chapter.

3.1 Programs used

Our system is programmed in the Python programming language, an easy and highly popular language that has an active community and a wide range of compatible open source libraries and implementations. In our implementation, we make use of the Tensorflow library (*Tensorflow home page*; 2017) which is an open source software library, originally developed by Google for high-performance numerical computing such as in the use of machine learning techniques like Neural Networks (NN). Tensorflow allows the user to construct a graph of data in the form of tensors and tensor operations in high-level python abstraction. A tensor is a regularized grid-like structure in which matrices and vectors are special cases. That is, matrices and vectors are 2D grids and 1D grids respectively but a tensor may represent a 3D grids or even 0D grids (scalars). In tensorflow, a tensor works as a symbolic handle to the output of a tensorflow operation, and the passing of tensors as input to other operations builds data flow connections. This way, data can be passed through multiple steps of computations and be manipulated efficiently in a predetermined fashion.

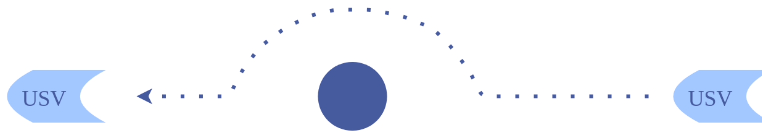


Figure 3.2: Conceptual illustration of obstacle avoidance. The dark circle represents the obstacle in question.

This way of programming is especially useful for machine learning purposes such as in the construction of Neural Networks. At runtime, tensorflow translates the operations specified by the user in python, to high-performance C++, providing its user with the ease of python coding while still keeping the benefits of highly optimized code.

3.2 The task

The task which we aimed to accomplish through Imitation Learning was that of object avoidance. We here define object avoidance as moving from an initial position to a goal position without coming into contact with an object positioned in the straight-line path between the initial and goal positions. This task requires the agent to travel towards the goal position, turn at an appropriate time to avoid the obstacle, and then turn back in order to reach the goal position. We deemed this to be an appropriate task for this project as it requires the use of sensor data and path planning in a dynamic setting. In this project, we thus aim to find a policy that accomplishes this task through Imitation Learning (IL), specifically through a GAN formulation of Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) combination. For comparison, we also included experiments using only RL with a manually crafted reward function for the same task.

3.2.1 Observation of state

We have until now referred to the agent's policy to be a mapping from the agents state to an action, $\mathbf{x} \in \mathcal{X} \rightarrow \mathbf{u} \in \mathcal{U}$. In practice, however, the agent will not have access to its whole state. Instead, only the information given through an observation, $\mathbf{o} \in \mathcal{O}$, of the state will be available. Thus, a more practical definition of a policy is a mapping from observation to action, $\mathbf{o} \in \mathcal{O} \rightarrow \mathbf{u} \in \mathcal{U}$. For our experiments, we

have used two types of observations: Relative positions[1] and radar images.

In physical systems, such as ours, an agent may use observations in the form of sensor data. This data must contain the information necessary for the agent to perform its task. E.g. an accelerometer, while an important sensor for control purposes, provides no information about an agents surroundings or upcoming obstacles. In order to be able to discover objects that lie a distance away, a distance measuring sensor must be used, such as radar or LiDAR images. Sensors providing both of these were available and could potentially be combined for a more complete insight into the state, however, we decided that using only one of the sensors would suffice for the scope of the task. We chose to use radar images, as they provide a long range view of the surroundings with accurate distances to obstacles, however, use of LiDAR would likely have been possible as well.

In an end-to-end system, the image does not need to be preprocessed by a scene understanding module before being passed to the policy. There is no need to detect an object and its position in a separate module before feeding the result to the policy because the policy should be able to deduce this information from the raw data by itself. However, mapping the appropriate action from such a large observation space is a more difficult task than if the important information was provided directly. We thus also included experiments with simpler observations in the form of the positions of the obstacle and the goal relative to the USV in order to compare the results.

Positional observations

One observation of the type using positional vectors, $\boldsymbol{o}_p$, consist of a vector shaped tensor on the form $[r_{pole}, \phi_{pole}, r_{goal}, \phi_{goal}]$. Here r_{pole} and ϕ_{pole} form the distance vector from the USV to the obstacle, the pole to be avoided, and r_{goal} and ϕ_{goal} mark the distance vector from the USV to the goal position. Both vectors are given in polar coordinates and are expressed relative to the USVs current heading. The reasoning behind the choice of the latter is twofold. Firstly, this mimics the way a human would observe an obstacle. A human captain would observe an obstacle

[1]Strictly speaking, relative position in this case is not an observation, because we measure only the position of the boat (GPS), unless the position of the object is known a priori, or position is estimated using a computer vision algorithm on the sensor data such as Radar/Lidar. Relative position together with the goal position can well be the full state. In the thesis, however, relative positions are considered as observations

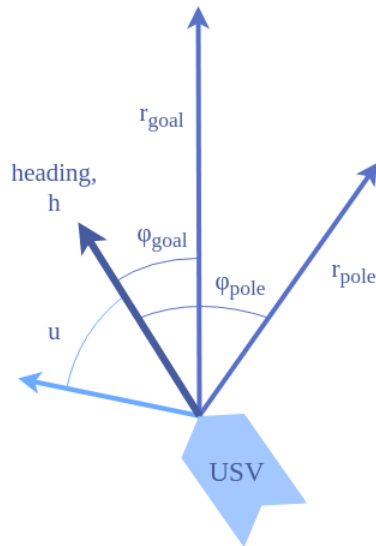


Figure 3.3: Illustration of positional vectors as defined in our system. All observational and action angles are given relative to the current heading.

relative to their own orientation, i.e. to their left or to their right, and swerve accordingly. This first-person view of the world is also the way a radar image expresses the data. Secondly, this makes the agent invariant to rotation, meaning the policy does not need to treat an obstacle it is approaching from the north differently than one it is approaching from the east. This makes training easier and the resulting policy more general. It is because of this same reasoning that we also express the action, a choice of reference heading, not as a new reference heading on the scale of 0 to 360 degrees, but rather as a degree of adjustment to the current heading. This type of observation and action is illustrated in Figure 3.3

Radar observations

While real radar measures were collected and studied, we did not end up using the images directly as observations. Instead, we chose to generate simplified substitutes for both the expert demonstrations and during training. The reasoning behind this will be presented in Section 3.3.2.

As any grayscale image, a radar image can be expressed as a 2D array of numbers, where each of the entries in this array adds to the observation space. The radar image thus provides a much larger observation space than the positional type observations.

In order to reduce the observation space somewhat and speed up operations with the image, the image size was reduced to a width of 119 pixels and a height of 50 pixels, which corresponds to the two dimensions of the 2D array.

Unsurprisingly, testing showed radar images on their own could only provide the position of the obstacle to be avoided, not the goal. The goal position is not a physical object and thus is not visible on the radar image. To prevent the agent from essentially going blind after passing the obstacle we decided to add the relative position of the goal to the image in a combined observation. In a practical perspective, we assume the goal position must be pre-assigned already through a path planning algorithm and the distance vector may be calculated from this and the USVs GPS position.

3.3 Training Data

While RL does not normally require a ready-made data set before training, it does require an environment for the agent to interact in and receive feedback from. In addition to this, as GAIL includes IRL as well as RL, a training set is still required in order to learn, specifically a set of expert demonstrations. Thus, in order to train our system, we needed both expert demonstrations and an environment in which to apply the policy.

3.3.1 Data Gathering

One of the results of the pre-project (Vedeler; 2018) was the gathering of expert USV demonstrations in preparation for this project. As explained in a pre-project report, we traveled to Horten, Norway, where we performed sampling of manual obstacle avoidance by the use of the USV, Odin, courtesy of the Norwegian Defence Research Establishment. During the demonstrations, the weather was dry and the water was relatively calm, which was beneficial to us as more challenging weather would introduce further challenges into an already difficult set of dynamics and task.

Each demonstrated episode consisted of the USV being manually steered directly towards an obstacle, before swerving either right or left to avoid collision and then steering back to the previous course. The obstacle in question was one of two poles

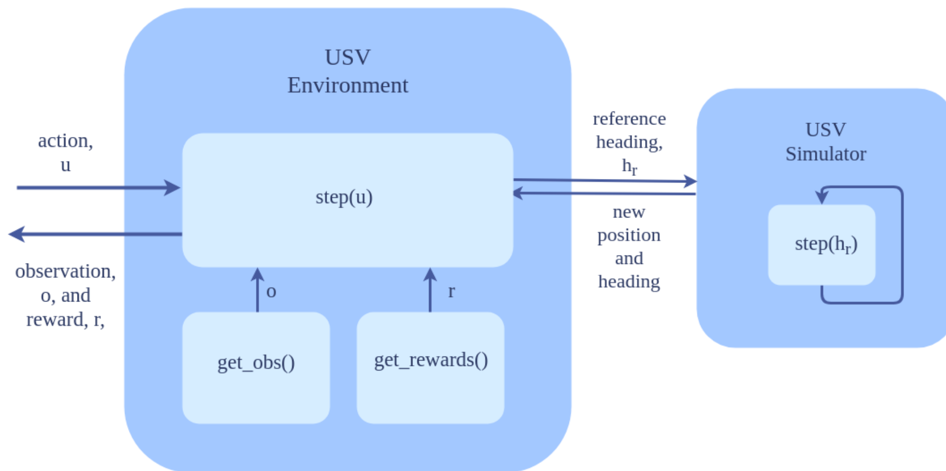


Figure 3.4: Simplified illustration of the environment module. The circle represents the goal position and the dot the pole, the obstacle to be avoided

present in the area, where we alternated between the two when using them for obstacle avoidance. The sensor data from the episodes was recorded in separate files. Each such episode file contained recorded radar images, heading measures and a note on which pole was used for the specific episode, in addition to other data which ultimately went unused, such as LiDAR readings.

The actuator input of the USV was not recorded during the demonstrations. The lack of actuator input means that the system cannot be completely end-to-end using IL in the sense that it does not produce a low-level control action in the form of thruster input. Instead, we chose the heading recorded by the radar compass, as the output. This heading may provide a reference value for a controller, making the system end-to-end only to the point of producing a high-level steering action.

3.3.2 Simulation of environment

In simple terms, GAIL learns by allowing the policy to act in an environment and then comparing these episodes with episodes performed by the expert in the same environment. However, in a physical system such as ours, this is not so straightforward. The USV is a large and expensive machine and allowing untrained policies to act through it would likely result in dangerous situations for both the USV and its surroundings. We thus cannot allow the policy to act in the true environment during training. Instead, we use a simulated environment.

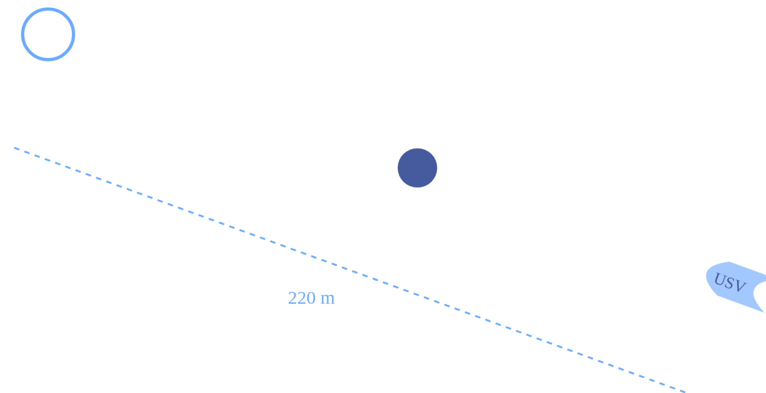


Figure 3.5: Illustration of initial episode setup

As per convention in RL, the environment handles the transition between the states, calculates the reward and provides the agent with a new observation for every step in an episode. A simplified illustration of this module, highlighting the step function, is shown in Figure 3.4.

Setting up the episode scenario

Before an episode is run, the `reset()` function of the module must be called. This resets the environment to an initial state from which the episode can take place. The environment contains the positions of the two poles used in the gathering of expert demonstrations and uses these as a base for the scenario's setup. When called, the `reset` function will choose one of the two poles at random and the chosen pole will act as the obstacle in the succeeding episode. The function then chooses a distance and an angle from which the USV is to approach. The angle is chosen from a uniform random distribution in the range of 0 to 360 degrees and the distance from the pole is chosen from a uniform random distribution in the range of 50 to 100 meters initial distance from the pole. The position of the USV is then set to this chosen position and its initial heading is set to face the pole, the obstacle to be avoided, similarly to the scenario of the expert demonstrations. Lastly, a goal position is placed 220 meters from the USVs initial position, on the other side of the obstacle. All the distances are chosen to approximate the setup of the expert demonstrations while still introducing some randomness into the setup. See Figure 3.5 for an illustration of the initial setup for an episode. After the setup is complete, an initial observation is generated and returned and the episode can begin.

Simulating one step

The `step(u)` function is called at each step of the episode in order to perform the action chosen by the policy, transition to a new state and receive a new observation of the new state. The function takes the action, u, which marks an adjustment that the agent wishes to perform on its heading. The environment adds this adjustment to the current heading of the USV, which itself is unknown to the agent, to get a new desired heading.

The environment acts as a stand-in for the actual physical environment and its dynamics. In order to simulate these dynamics of the system, we used a USV simulator provided us by FFI. This simulator simulates the dynamics of the USVs actuator controller and then simulates the dynamics of the physical system when the USV is moving in the water. Its step function thus takes a desired heading, performs one step of the controller, whose aim is to reach this value, and one step in the simulated physical system, before returning the new position and orientation of the USV. Reaching the desired heading is not an instant procedure and the simulators step function returns after only 0.1 seconds. In order to assure that, given a non-drastic action, the desired heading is reached, we thus perform 20 iterations of the simulators step function, effectively simulating 2 seconds of dynamics. When looking at the number of radar images that were recorded in each expert demonstration, a slower step time like this may actually be more realistic for a system which uses radar images for observations as the sampling frequency of the radar is not very high. We keep the speed of the simulated USV constant as was the case with the expert demonstrations.

After the simulation of the physical system has been performed, the environment module records the resulting position and heading of the USV. These values are then used to generate the observation, o, of the new state and to calculate a reward, r, to provide the agent. Which type of observation is generated depends on which is enabled.

Simulating radar images

Because GAIL compares expert demonstrations with episodes performed by the policy on the environment, it is important that the environment is as similar to

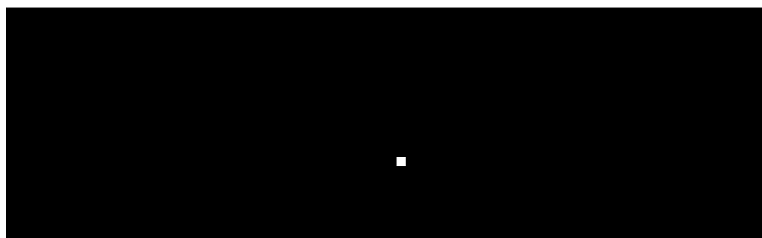


Figure 3.6: Example of a stand-in radar image generated for our task

the true environment as possible. If not, the discriminator may exploit the differences and base its predictions on this, which the policy cannot compensate for and thus effectively preventing any further learning. This means that the simulated environment and the observations it generates must be similar enough to fool the discriminator.

This fact proved difficult to achieve when using radar images as observations. In order to use the data collected from the radar on the true USV in the training, we would have to generate images that truly looked like those generated by a real radar in the same place as the expert demonstrations were performed. This generation would also have to be relatively quick as it would have to be performed at every timestep of every episode during training.

A fair amount of time went into studying the collected radar data and attempting to use it directly or indirectly in the generation of appropriate radar observations during simulation but this was ultimately scrapped due to the limited time and scope of the project. Instead, fully generated stand-ins were created to test the concept.

The stand-ins were constructed to be the simplest form of radar data. We created a black image and placed a box at the appropriate place according to the position of the obstacle relative to the USV and its heading. As with real radar images, the distance from the USV is represented through the row of a given pixel, while the angle is given by the column. We gave the USV a maximum viewing distance of 200 meters and a viewing angle spanning from -120 degrees to 120 degrees relative to its heading. This is smaller than the real radar images but it allows us to also make the images smaller and thus save computational time. A smaller view like this can be created from real radar images through cropping. We created images of 50-pixel height and 119-pixel width. Scaling an object's position to these ranges, they could then be represented by a white box on the black background. An example of a stand-in radar image generated through this method is presented in Figure 3.6.

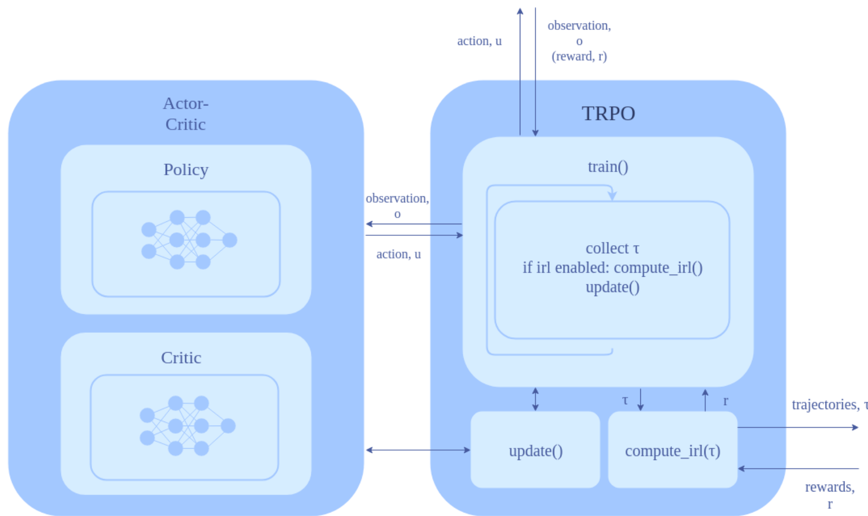


Figure 3.7: Simplified illustration of the RL module of the system, consisting of a modified TRPO algorithm. The module makes calls to the environment, sending an action u, and receiving the following observation, o, and a reward calculated by a manually crafted reward function. When IRL is enabled, it also calls the IRL module, sending the batch of trajectories, τ, and receiving the reward for the trajectories, calculated by the discriminator. The critic network has the same structure as our policy network, but lacks the stochasticity which the policy introduces.

3.4 Reinforcement Learning setup

A large part of the GAIL system is the RL module, which uses the TRPO algorithm (Schulman et al.; 2015) to learn a policy. We based this module on a TRPO implementation from OpenAI:SpinningUp (2018) but substantially altered the code to the point where only the module's `update()` function remained unaltered. This was done in order to accommodate the specifications of the task and to enable its use as part of a GAIL system. An overview of the module is given in Figure 3.7. Because the TRPO algorithm itself has already been explained in Section 2.3.5, we will here focus on things more specific to our scenario, such as the structure of our policy. By disabling the connection to the IRL module of the system, this module becomes a regular RL training system. We used this setup for testing of learning through RL.

3.4.1 Policy

We performed experiments with two types of policies, Multivariate Gaussian and Categorical. As explained in Section 2.3.4, both use a neural network, but where Gaussian policies output continuous results, Categorical policies output discrete. However, that does not mean that Categorical policies cannot be used in continuous systems. In fact, the use of a Categorical policy will limit the number of potential action values from theoretically infinite values to a discrete set. As we deemed such a limitation of action space to simplify the task somewhat we decided to perform experiments with this type of policy. The resolution of the Categorical Policy is however limited to a discrete number of categories, while a Gaussian policy is not. We decided thus to perform experiments with both policies in order to gauge their impact on the overall performance.

The networks used for the policies were equal for both the Categorical and Gaussian policy, with the exception of the output layer. For the Gaussian policy, we created an output function which assured the output of the network would be contained within a certain range. The output from the final layer was sent through a sigmoid function, scaling it between 0 and 1, before this value was scaled between a minimum action value and a maximum action value. We did this to assure that the output was in the correct range. For the Categorical policy, the network outputs a discrete number between 0 and 40, which is later scaled to a number between a minimum action value and a maximum action value. For both policies, the minimum and maximum action value were -30 and 30 degrees respectively. We chose this interval as a heading change of more in one timestep would be unrealistic. In fact, one could argue that even a 30-degree change in the heading would be drastic in one timestep. With this interval, the Categorical policy has a resolution of 1.5 degrees.

Positional Observations

The positional observation is a vector with 4 entries, representing the position of the obstacle and the goal, expressed relative to the USV. For this type of observation, we chose to use a fully connected network of two hidden layers, with 400 and 300 neurons each, inspired by Martinsen and Lekkas (2018). For the hidden layers, we use a $tanh$ activation function. Obviously, the input size of the policy network in this case is 4, whereas the output layer has the shape as described in the previous

section depending on whether it is categorical or Gaussian.

Radar Image Observations

The use of images as input lends itself to a Convolutional Neural Network (CNN) as described in Section 2.1.3. In constructing our CNN policy we took inspiration from AlexNet (Krizhevsky et al.; 2012), arguably the most well known CNN to date. AlexNet is a classification network used to classify images, labeling them as e.g. 'cat' or 'dog', and consists of five convolutional layers and three fully connected layers. Even though our task is not to classify per se, part of the network may be used for other purposes as well since the actual classification is only performed in the last layer of the network.

The images we are to use are much simpler than those used by Krizhevsky et al. (2012), being simplified binary radar images. We thus deemed it unnecessary to use the full AlexNet network and settled for only the first convolutional layer, followed by two fully connected layers. The first convolutional layer of AlexNet uses 96 kernels of size 11×11, and performs convolution with a stride of 4 with $relu$ activation. We downsized this to a convolutional layer of 10 kernels of size 7×7, also performing convolution with a stride of 4 and $relu$ activation. We did this because the information that is important in the image is not textures or shapes, but the position of pixels with non-zero values. The smaller number of kernels reduces the number of parameters in the network, reducing the training time as well. AlexNet follows up this convolution layer with a max-pooling layer, which helps make the output more invariant to small translations in elements of an image. In our case, however, the position of a pixel contains important information on the distance or direction of the object it may represent. For this reason, we chose to not include max-pooling in our network. The fully connected layers of AlexNet contain several thousand neurons each, but for our task, we settled on using the same number of neurons as in the network used for positional observation, i.e. 400 for the first layer and 300 for the second.

As mentioned earlier, the images in the radar observation case were not descriptive of the goal and so they were accompanied by a vector describing the goal position relative to the USV. However, because this position is a vector and not a 2D grid, it cannot be fed to the convolutional layer of the network together with the image.

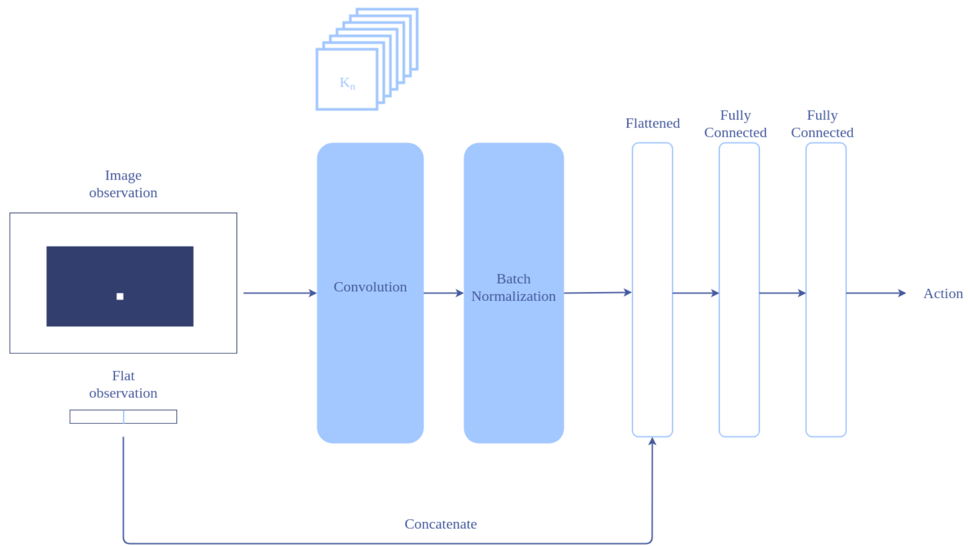


Figure 3.8: Simplified illustration of the policy network

Instead, we chose to perform a late input, merging the vector observation with the output of the convolutional part of the network before following suit with the fully connected layers.

When performing such a late input, we found that it was especially important to standardize this second input, i.e. scale the values to lie between 0 and 1. This was so that the new input, which could have values of hundreds of meters, did not overpower the output of the previous layer and change the focus to solely reaching the goal, ignoring the task of avoiding the obstacle. To make sure that the output of the convolution did not overpower the late input either we also added batch normalization and scaled the image input to the network. The network is illustrated in Figure 3.8

3.4.2 Reward

While GAIL does not need a manually crafted reward function, in order to run experiments with RL alone, we still need one. The reward function is implemented in the environment module. However, because it is relevant to the RL setup only, we will address it here.

In order to craft a reward function, we boiled down the task to its two most essential parts: Reach the goal position, and do not reach the pole position. The goal and

pole positions are thus both important values in our reward function but in different ways. We want the agent to be rewarded when the goal position is reached but to be punished should it reach the pole, i.e. crash. However, only providing a constant reward when the agent finds itself in one of these two positions or in a bound around them, is not sufficient.

Because the state space is so large, care had to be taken so that the agent would experience a reward that was sufficiently descriptive of the desired behavior during the whole episode, even at the initial state. TRPO is a policy gradient method, meaning it updates through gradient ascent/descent. Doing this is comparable to a person attempting to reach the highest point in an unknown terrain wearing a blindfold, they move in the steepest direction of the terrain they can currently feel. In RL, this terrain is a result of the reward function. However, if the agent starts in a spot where this terrain is flat, i.e. there is no change in the reward when transitioning from one state to the next, there is no clear direction in which to improve the policy and thus the training may not converge to an optimal policy. The lack of a sufficiently descriptive reward function may be countered by more extensive exploration, but this will likely prolong the training. A descriptive reward function was especially important because of the large state space in our environment and of TRPO's vulnerability to entrapment in suboptimal, local solutions of the task.

We thus chose to express our reward function using Gaussian functions, as these are smooth, symmetrical and differentiable functions who's output lies between 0 and 1. This makes them predictable and easy to work with. A Gaussian function is defined as:

$$g(x, \sigma, \mu) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}((x-\mu)/\sigma)^2} \tag{3.1}$$

where σ^2 is the variance, denoting the steepness of the bell-shaped curve, and μ is the mean, denoting the position of the maximum. By inserting a difference in position, $p - p'$, for x and setting $\mu = 0$ we get a function who's maximum arises when $p - p' = 0$ and gradually decreases as p moves away from p'. Using this, we define our reward function for this problem as:

$$r_{pole}(p_{USV}, p_{pole}) = -\frac{2}{\sigma_{pole}\sqrt{2\pi}} e^{-\frac{1}{2}((p_{USV}-p_{pole})/\sigma_{pole})^2} \tag{3.2}$$

$$r_{goal}(p_{USV}, p_{goal}) = -1 + \frac{1}{\sigma_{goal}\sqrt{2\pi}} e^{-\frac{1}{2}((p_{USV}-p_{pole})/\sigma_{goal})^2} \tag{3.3}$$

$$r_{goal}(p_{USV}, p_{pole}, p_{goal}) = r_{pole}(p_{USV}, p_{pole}) + r_{goal}(p_{USV}, p_{goal}) \tag{3.4}$$

where p_{USV}, p_{pole}, and p_{goal} are the positions of the USV, pole and goal respectively and π denotes the number, not the policy. We deem avoiding the pole as more important than quickly reaching the goal and thus weight this penalty double. This means $r_{pole}(p_{USV}, p_{pole}) \in (-2, 0)$ and $r_{goal}(p_{USV}, p_{goal}) \in (-1, 0)$, making $r_{goal}(p_{USV}, p_{pole}, p_{goal}) \in (-3, 0)$.

Because the danger of the pole is only a factor in close proximity, and we did not want the agent to take a too large detour around it, we chose $\sigma_{pole} = 5$ meters. As mentioned previously in this section, we want the goal to attract the agent even from far away. For this reason we chose $\sigma_{goal} = 100$ meters.

3.5 Imitation Learning setup

When the full IL setup is enabled, the RL module does not use the rewards calculated by the reward function when training its policy. Instead, it uses the rewards received by the IRL module. The discriminator network, which calculates the reward based on the observation-action pairs of a trajectory, is trained through the GAIL updating scheme. The implementation is based on an implementation of the discriminator update scheme of GAIL by Fu (2018), altered to accommodate for our task. An overview of the module is given in Figure 3.9.

The full IRL setup, presented in Figure 3.1, uses the same policy as the RL setup explained in the previous section. In this section, we will thus focus on the discriminator network which this model introduces and the expert data set which it uses during training.

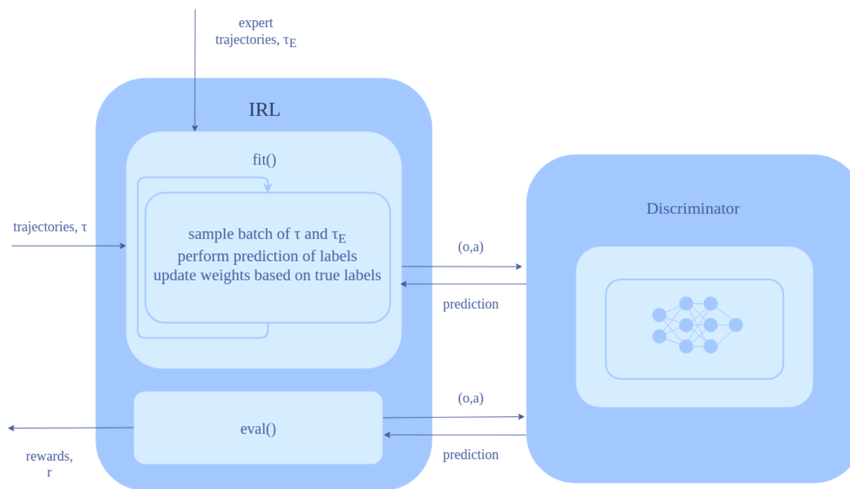


Figure 3.9: Simplified illustration of the IRL module of the system, consisting of the GAIL discriminator update rule.

3.5.1 Data Preprosessing

While the expert data was gathered before the commencing of this project, the data collected could not be used in its raw form. Thus, time was spent writing a program to preprocess the data collected and construct the expert trajectories which would be used during training.

The positional observations were constructed from the USVs position, recorded through GPS, by calculating the vector from the USV to the static position of the pole and the goal for every step in the trajectory. When performing the demonstrations, the expert had not been aiming for a specific target position after passing the goal. Thus, we chose the last position of each trajectory as the goal position. The pole positions had been recorded separately.

The different sensors on the USV have different sample frequencies, resulting in measurements that are neither in synch nor of the same number. In order to construct the expert trajectories, the observations recorded from the GPS, and actions from the radar compass, had to be matched in time. For this, we wrote a program that matched them through the timestamps attached to each measurement.

The heading recorded during the expert demonstrations is not in actuality a reference, but rather a measurement of the heading measured from the radar compass. However, by down-sampling extensively it can be assumed that using the simulated heading

controller, the USV will have reached the next value in the time between two samples. Thus, the heading measured at the next time step can be used as an approximate to the real reference value. From this, a change in reference was calculated by comparing the two headings, resulting in a change in the heading which we defined as the expert's action. Doing this, we were able to recreate the paths of the expert demonstrations using the expert's actions in our simulated environment, albeit at a lower speed than during the recording of the expert demonstrations.

Depending on what type of observation, the trajectories would then be further processed. For example, when using radar observations, these had to be generated using the same method as during training to assure the discriminator would not learn to discern them through a difference in generation. When using a Categorical policy, the actions had to be discretized.

3.5.2 Discriminator

Unlike the policy, which takes an observation and outputs an action, the discriminator takes both the observation and the action of a timestep and outputs a prediction of a label, i.e. if the observation-action pair comes from an expert or the agent in training. As with the policy, the network differs depending on the observation type.

Positional Observations

The discriminator network for the positional observations is highly similar to the policy network, i.e. two hidden layers of 400 and 300 neurons each. The only difference is the input layer which takes the observation and action of a timestep concatenated into one tensor.

Radar Observations

The discriminator network for the Radar based observations is highly similar to the policy network using those same observations. Because part of the observation is an image and part is a vector, the network performs convolution on the image, flattens the result and concatenates the result with the vector observation before sending them through two fully connected layers. This part is equal to the policy network.

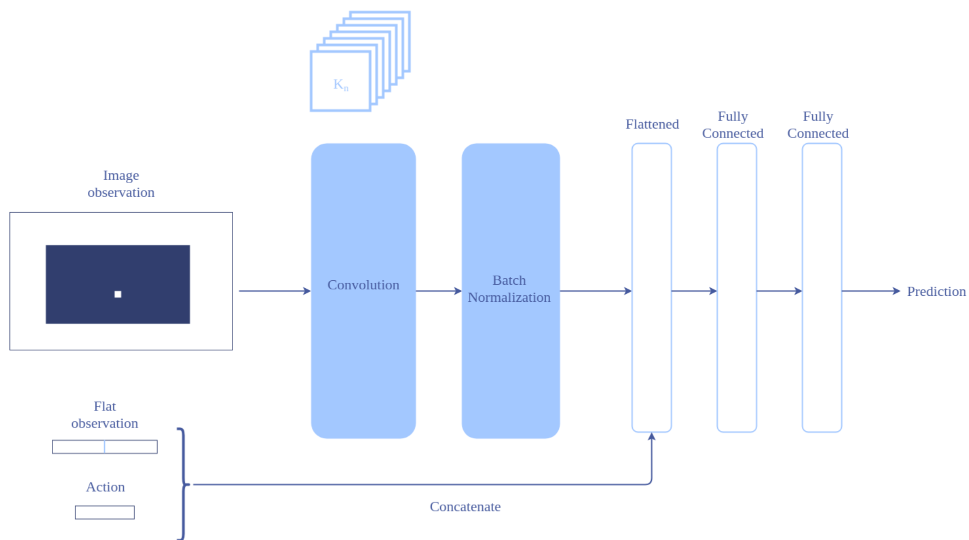


Figure 3.10: Simplified illustration of our discriminator network for radar observations

However, the discriminator also uses the action, one single value per step in our case, as input to its network. As with the vector part of the observation, the action cannot be merged with the image directly before the convolutional layer. Thus, it is also inputted at a later point in the network. This discriminator network is illustrated in Figure 3.10

Chapter 4

Results and Discussion

In this chapter, we present the results of each of our setups, using their best performances after training. We also present a discussion of our results and of the use of Imitation Learning (IL). Finally, we present our ideas for future work.

4.1 Validation

Validation was performed by executing the policy and measuring the results. 15 episodes were performed, each with a different, but fixed, starting position. For each of these episodes, a random pole was chosen from the two available to work as the obstacle. The USV was then placed from 50 to 100 meters away from the pole, making sure that the 15 test episodes spanned the range of possible distances. The USV was placed at an angle spanning from 0 to 360 degrees from the pole, facing it. Then the goal placed 220 meters from the USV, at the opposite side of the pole. We performed validations each 100th episode during the course of the training.

In Reinforcement Learning, unlike in Supervised Learning, there is no separate validation data set and training data set. The agent trains by being exposed to the environment, just like it would be at validation time. Our validation setup was thus very similar to the training scenario, but more regularized so that we could more easily compare the results.

We deemed an episode a quantitative success if the agent was able to get within 10 meters from the goal while never being closer to the pole than 10 meters at any point during the episode. Because steering includes more than this quantitative measure, however, we also plotted the path taken by the agent, the observations, the actions and the rewards recorded for the episode in order to also evaluate more qualitative aspects. E.g. since we operated with fixed length validation episodes, some validations ended before the USV reached the goal position. Whenever it was clear that the agent was heading towards the goal state and would have reached it within only a few steps after the episode ended, we deemed it a successful episode.

4.2 Results

4.2.1 Training

In Deep Learning (DL) projects such as this, training time can be an issue. Training deep networks are time-consuming and this limits the number of experiments that may be performed. Our results might have improved through longer training sessions and more experimentation. However, with the amount of training time available to us for training and testing when the simulator arrived and the number of set-ups to test, we did not have enough time to prolong the training sessions or perform more experiments. For reference, 5000 iterations of training with radar observations took approximately 12 hours. As recommended for TRPO in Schulman et al. (2015), the RL optimization was performed through the conjugate gradient method, while the IRL module used Adam optimization.

4.2.2 Expert Demonstrations

After preprocessing the raw data from the expert demonstrations, we plotted them and examined them. Out of the 46 demonstrations collected, we deemed 35 of them good enough to use as our expert data set. An example from this set is shown in Figure 4.2. The expert aims straight for the obstacle, makes a decisive turn, and proceeds to turn back to its original path. The curve of the path is smooth. Though the actions may look noisy, keep in mind that each entry is a correction in heading and that from one timestep to another the USV is given time to reach the new, corrected, heading as long as the demanded change is not too drastic.

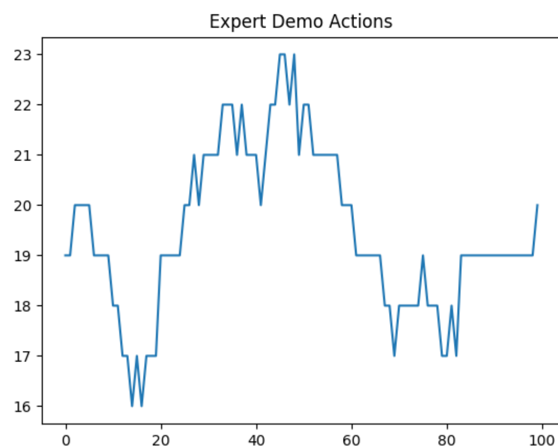
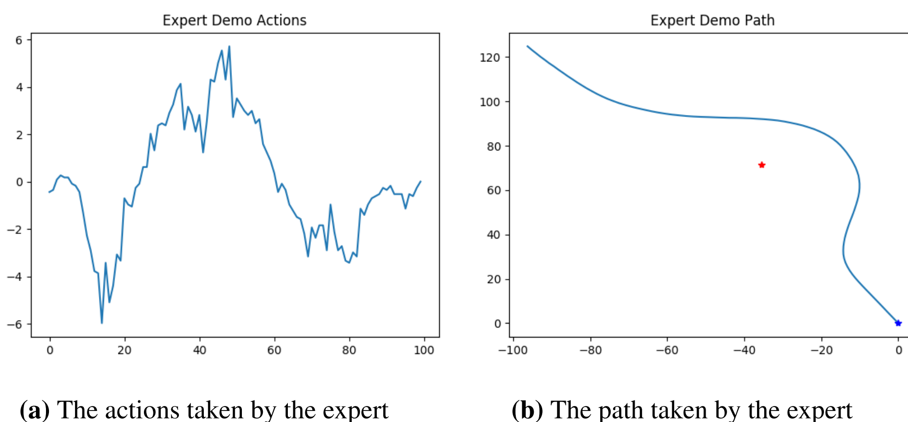


Figure 4.1: Expert actions from Figure 4.2 discretized

This path is steered by a human, and it is a similar steering pattern which we wished to re-create. Because there was no given goal position when the expert paths were recorded, we chose to mark the final position of the USV as the goal position in each expert episode.

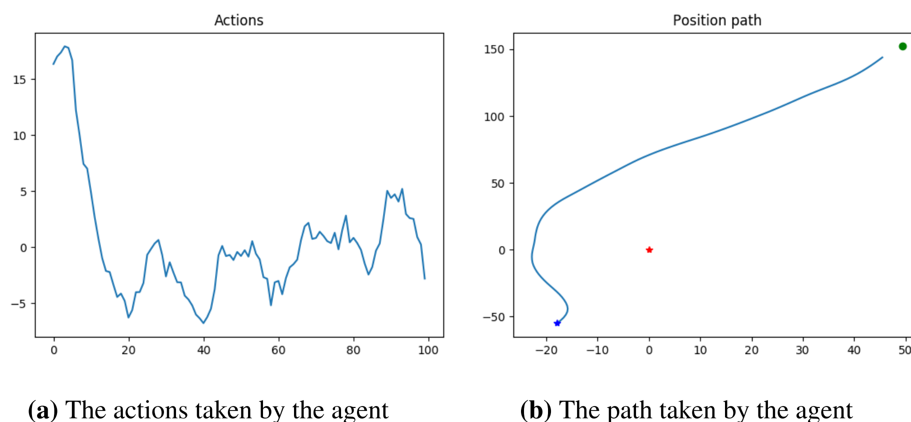


(a) The actions taken by the expert

(b) The path taken by the expert

Figure 4.2: Example of an expert demonstration with continuous action space. In (b), the blue star marks the starting position, and the red star marks the obstacle.

4.2.3 Rwsults using Positional Observations

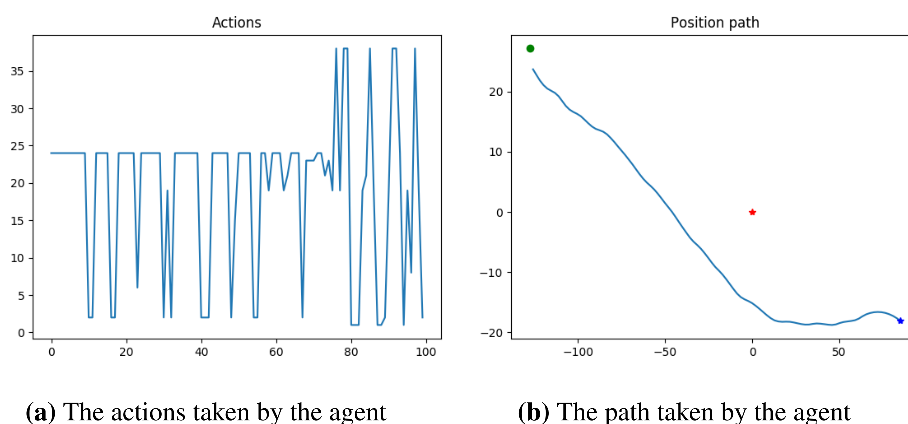


(a) The actions taken by the agent **(b)** The path taken by the agent

Figure 4.3: Example of successful episode from validation of the RL setup with positional observations with Gaussian Policy. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

RL setup with Gaussian policy

After approximately 2400 RL training iterations the agent using positional observations and Gaussian policy performed all 15 validation tests successfully. As shown in Figure 4.3, an example of a successful episode, the agent is able to maneuver around the obstacle and converge on the goal position. Compared to the expert demonstration in Figure 4.2 the turn is slightly larger and the shape of its actions is different. The expert's actions marked a decisive turn to the left and then to the right before a slower adjustment to the right to get back on the path. The agent, however, shows oscillations in its action and chooses a more direct path to the goal position after avoiding the obstacle. This effectiveness is logical when considering the reward function the agent attempts to optimize for.



(a) The actions taken by the agent

(b) The path taken by the agent

Figure 4.4: Example of successful episode from validation of the RL setup with positional observations with categorical policy. In (a) the vertical axis marks the action category, where 20 equals a 0 degree change in heading and 40 equals a 30 degree change. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

RL setup with categorical policy

The categorical policy seemed to learn faster and performed all 15 validation episodes successfully after 1000 iterations. However, its actions were more radical, creating oscillating movements in the path as shown in Figure 4.4. This may partly be a more extreme case of the symptom spotted in the Gaussian policy in Figure 4.3 and while this behavior is unfavorable, it is also interesting. As can be concluded from Figure 4.5, the oscillations do not impact the reward the agent gets when using the reward function we have constructed and thus has no incentive to correct it. While this likely could have been solved by assigning a penalty for large changes in actions, it highlights the problem of crafting and tuning a reward function manually. When defining a task, even one as simple as this one, much thought must be put into the reward function for RL to be able to accomplish the task. We would also like to point out that while a penalty on rapid changes in the categorical RL policy may have lessened the larger oscillations of the agent, removing the smoother oscillations

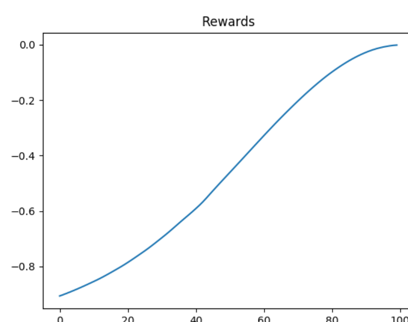
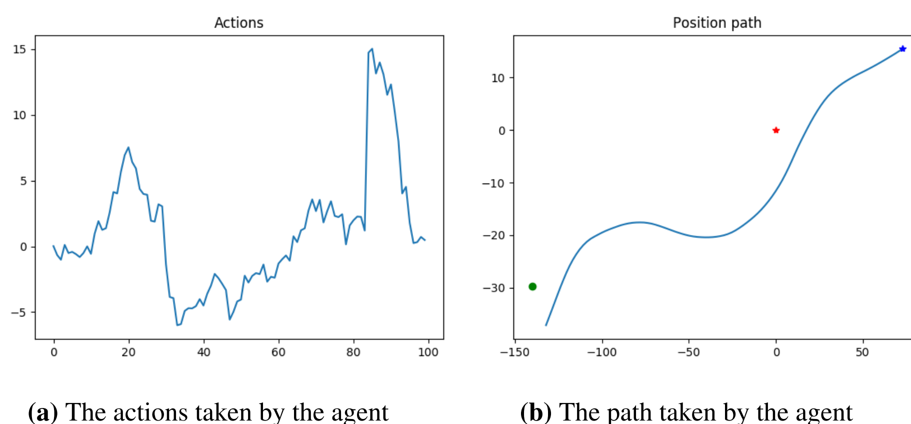


Figure 4.5: Rewards corresponding to the episode in Figure 4.4



(a) The actions taken by the agent

(b) The path taken by the agent

Figure 4.6: Example of unsuccessful episode from validation of the IL setup with positional observations with Gaussian policy. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

of the Gaussian RL policy in Figure 4.3 may prove more challenging.

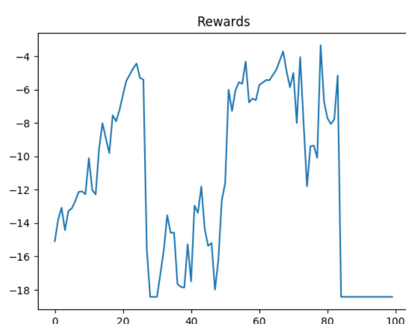
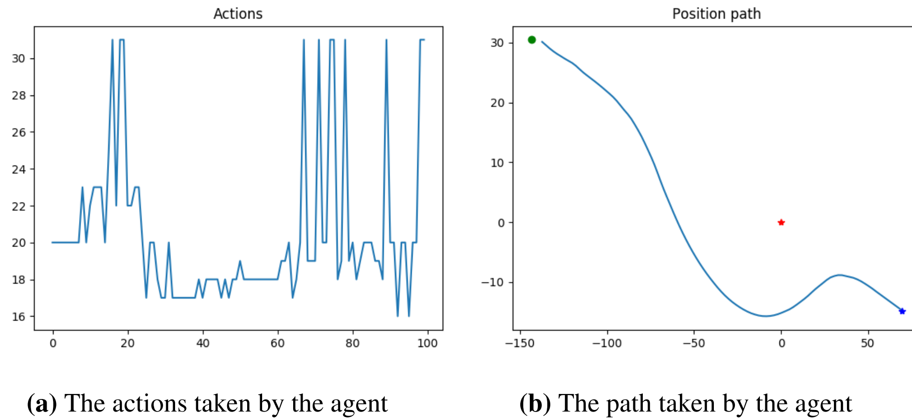


Figure 4.7: Rewards corresponding to the episode in Figure 4.6

IL setup with Gaussian policy

The Gaussian policy IL setup peaked at 1700 training iterations with all but two of the episodes successful. Though these two were close, they did not quite reach the goal position. The agent did not seem to be able to amend this flaw with further training. This agent also seemed to steer a bit closer to the pole than the previous agents, something which also caused a few unsuccessful episodes at later iterations.

An example of an episode where the agent missed the target is shown in Figure 4.6. Here we see that the agent has a spike in the action which steers it off course at the end of the episode. It is quite peculiar that this flaw in the policy is not corrected as the spike is penalized by the discriminator, which we can defer from comparing the time of the action spike in Figure 4.6a with the drop in reward in Figure 4.7. One possibility is that this is caused by the TRPO algorithm getting stuck in a local optimum and/or that this behavior remains from a previous iteration of the



(a) The actions taken by the agent

(b) The path taken by the agent

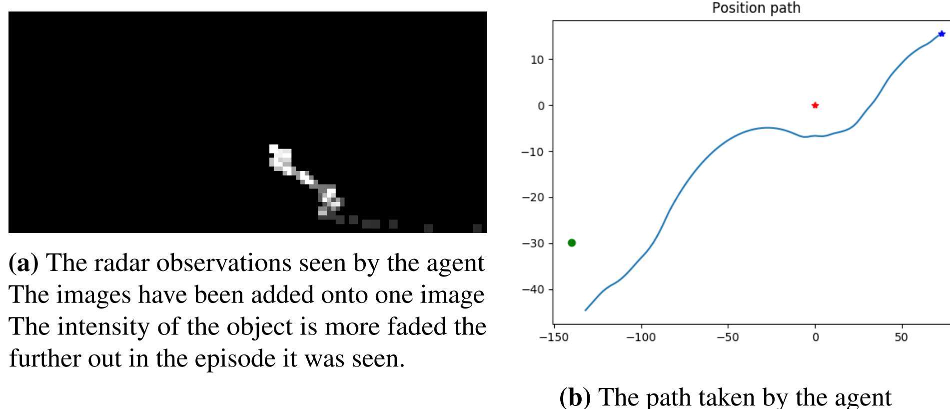
Figure 4.8: Example of successful episode from validation of the IL setup with positional observations with Categorical Policy. In (a) the vertical axis marks the action category, where 20 equals a 0 degree change in heading and 40 equals a 30 degree change. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

discriminator's untrained feedback. From the theory, explained in Section 2.3.5, we know that TRPO should update so that it monotonically improved the policy. However, this may not apply when the reward it is updating under is changing such as is the case in GAIL.

From Figure 4.6b, we can see that the shape more closely resembles the expert's and that it steers back to the original path before continuing. The shape of the action graph, with the exception of the spike at the end, also closely resembles the experts. Like the expert, it marks three distinct and gradual turning motions. We can also see that the path is smoother and lacks the oscillations seen in the previous two scenarios.

IL setup with categorical policy

The agent using the IL setup with categorical policy produced successful episodes on all 15 validations after 2100 iterations. Figure 4.8 presents an example of a successful episode. Like the previous categorical policy, it produced some large changes through its actions, but unlike before, here they produced no oscillations. Here they seemed to help the USV make sharper turns and course corrections while it otherwise kept close to zero change. When considering both quality and quantity this agent seems to have the best performance on this observation type.



(a) The radar observations seen by the agent The images have been added onto one image The intensity of the object is more faded the further out in the episode it was seen.

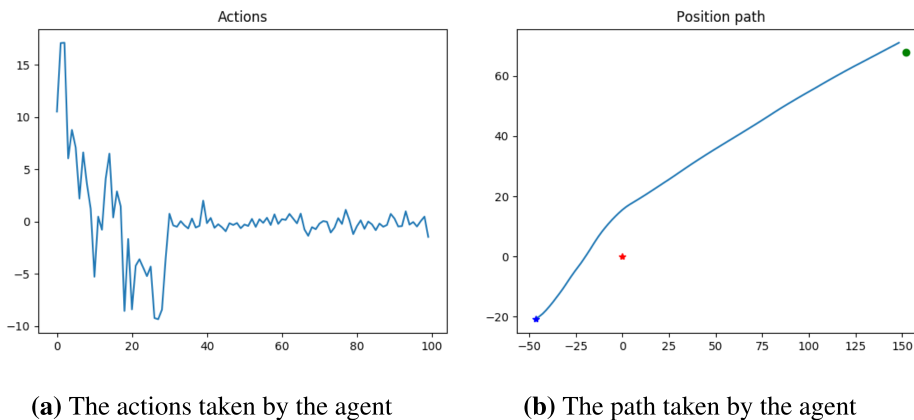
(b) The path taken by the agent

Figure 4.9: Example of episode from validation of the IL setup with radar image as the only observation. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

4.2.4 Radar Observations

As mentioned previously, before we included the position of the goal into the radar observations, we first performed experiments using only radar images as observations. As seen by the example from one of the IL setup validations in Figure 4.9, we found that the agent struggled with reaching the goal position, resulting in few to none successful validation episodes. This is not surprising as the moment the obstacle disappears out of the agents 240-degree view angle, the agent essentially goes blind. The radar observations provide no information on where it should head in relation to the goal position. While this experiment did not yield successful behavior in relation to the overall task, it did indicate that our networks were able to extract the position of the obstacle from the images and avoid collision. These experiments led to our decision to include the goal positional vector in the radar observation setup as explained in Section 3.4.1.

RL setup with Gaussian policy With the new observations, the RL setup using a Gaussian policy was able to perform 15 out of 15 validation episodes successfully after only 1300 iterations. An example is shown in Figure 4.10. Out of all of our experimental setups, this is likely the most consistent. During all 15 validation episodes, the agent had a very consistent minimum distance to the pole, varying with less than one meter regardless of the initial distance the agent had to the pole. The same consistency was also present considering the agent's ability to reach the



(a) The actions taken by the agent

(b) The path taken by the agent

Figure 4.10: Example of successful episode from validation of the RL setup with radar observations with Gaussian policy. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

goal position.

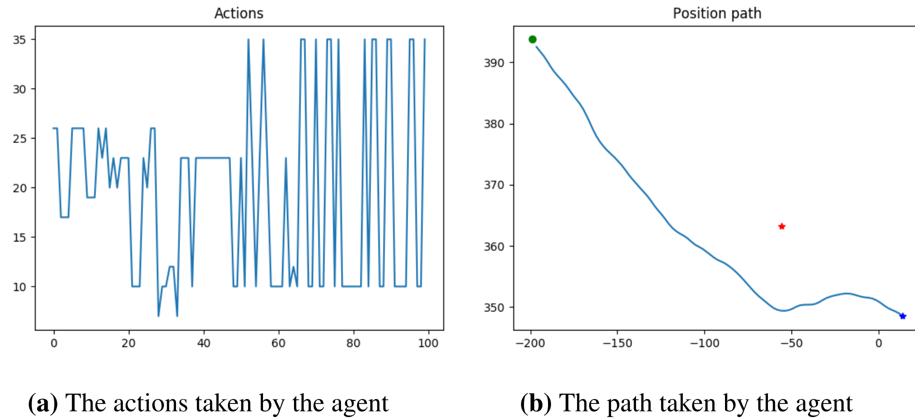
From Figure 4.10, we can also see that the agent no longer follows an oscillating path and that it has more stable actions. In fact, the actions seem to keep close to a zero change in heading, making for a highly efficient path albeit one less similar to that of the expert which the RL agent does not have access to.

RL setup with categorical policy

The RL setup with a categorical policy needed more training before reaching 15 successful validation episodes, specifically 4300 iterations. As seen by the example in Figure 4.11, the result is an agent that manages to avoid the pole and reach the goal in a similar fashion as with the positional type observations in Figure 4.4. The oscillations in the actions and path also remained.

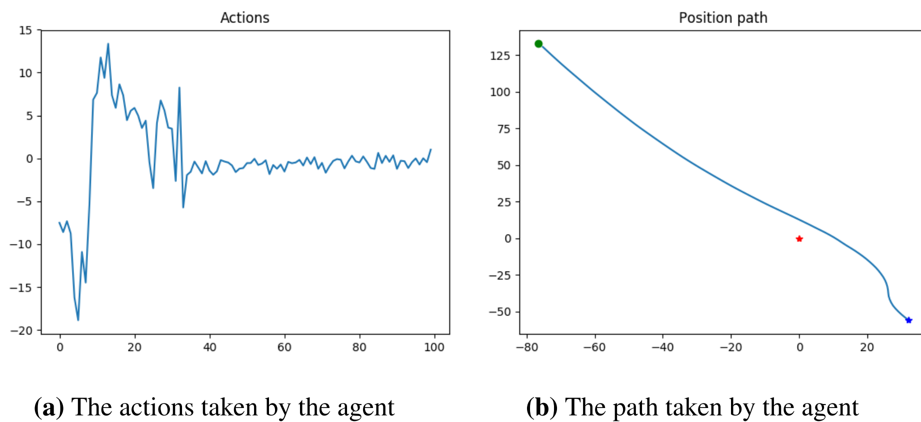
IL setup with Gaussian policy

The IL Gaussian policy at its best managed only 8 successful episodes out of 15 validation episodes at 3700 iterations. The unsuccessful episodes came as a result of the agent steering too close to the pole, coming as close as a 5-meter distance from the obstacle at the worst run. While this would likely not have caused a collision due to the obstacle's size, it is dangerously close. Plots of this particular episode is presented in Figure 4.12. What is interesting though, is that this was the only agent who chose to alternate between passing the obstacle on the right and passing it on



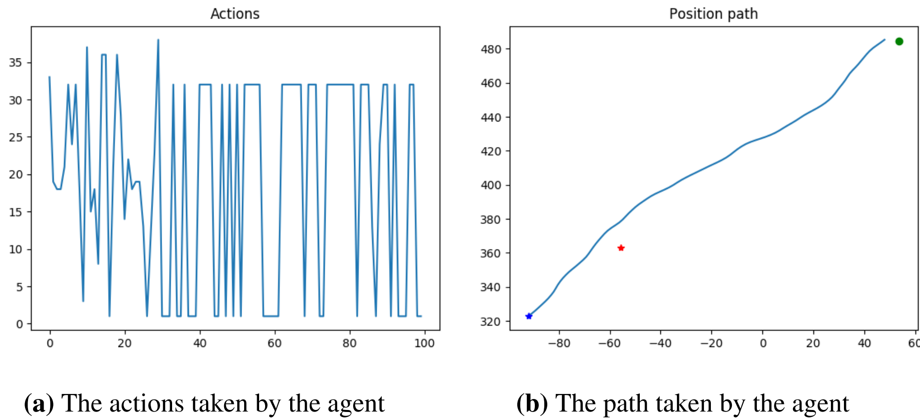
(a) The actions taken by the agent **(b)** The path taken by the agent

Figure 4.11: Example of successful episode from validation of the RL setup with radar observations with categorical policy. In (a) the vertical axis marks the action category, where 20 equals a 0 degree change in heading and 40 equals a 30 degree change. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.



(a) The actions taken by the agent **(b)** The path taken by the agent

Figure 4.12: Example of unsuccessful episode from validation of the IL setup with radar observations with Gaussian policy. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.



(a) The actions taken by the agent

(b) The path taken by the agent

Figure 4.13: Example of successful episode from validation of the IL setup with radar observations with categorical policy. In (a) the vertical axis marks the action category, where 20 equals a 0 degree change in heading and 40 equals a 30 degree change. In (b), the blue star marks the starting position, the green dot marks the goal position and the red star marks the obstacle.

the left. The expert also alternated between the two. With a starting distance of 70 meters and lower from the obstacle the agent chose to the right, while at starting distances of higher than 70, it chose to turn towards the left. Interestingly, it was at these lower starting distances, that it also failed the episodes. While this agent had the least successes in our quantitative definition of success, other than returning to its original path too quickly, its qualities were favorable. The paths were smooth and showed no noticeable oscillations and the actions are reminiscent of the expert's actions as seen in Figure 4.2a, though with a stronger tendency towards a straight path to the goal.

IL setup with categorical policy

The IL agent with categorical policy succeeded in 13 out of 15 validation episodes after reaching the maximum training iterations of 5000 iterations. Like with the Gaussian IL agent, the unsuccessful episodes came as a result of the agent steering too close to the obstacle, though not to the same degree. Here the agent reached a minimum distance to the obstacle of 9.6 and 6.8 meters in the two unsuccessful episodes. Plots of the least successful episode are given in Figure 4.13. For both the successful and the unsuccessful episodes, we observed oscillations in the agent's path, similar to its RL counterpart in Figure 4.11.

4.2.5 The Difference between Categorical and Gaussian Policy

We initially believed that because of its discretization of the action space, a categorical policy would simplify the problem and learn faster. However, from our results, it seems this is not always the case. The categorical policy was the quickest to learn in the case of the positional vector observations. Yet, using radar observations, the RL agent using the Gaussian policy was by far the quickest learner and seemed to produce the most consistent results overall.

When considering all previously mentioned experiments, the categorical policy did overall perform a greater number of successful episodes than the Gaussian policy. However, the policy also seemed to be overall more susceptible to oscillations and produced action graphs that looked less like that of the expert's in Figure 4.1.

One reason for its susceptibility to oscillations may be that the categories in this policy are treated as separate, non-related labels. Their numbers and their order do not necessarily mean anything, just as a label of 'dog' has no inherent relationship with a label of 'flower'. For example, the category corresponding to zero change in heading does not necessarily lie closer in action space to the one corresponding to 3 degrees of change than the one corresponding to 30. We have chosen to label the actions in rising order so that category 0 corresponds to a -30-degree change while category 39 corresponds to a 30-degree change. However, the agent would have acted the same if we instead chose to label them in random order, such as category 0 corresponding to a 27-degree change while category 1 corresponding to a -25-degree change and so on.

Gaussian policies, however, are continuous and so the actions relate to each other differently. An output of 0 is inherently closer to an output of 2 than 10 in the action space. That does not mean that jumps in outputs cannot happen, but they require a greater process to be learned.

4.2.6 Summary of Results

Overall, the results of our experiments have been mostly positive. Though some unsuccessful episodes were recorded, bringing the lowest succession rate down to 53.3%, all set-ups showed a clear grasp of the task, even when their performances were not accurate enough to be marked successful. The quantitative performances of all 8 set-ups are summarized in Table 4.1

Setup	Success rate
RL-gaussian-positionalObs	100%
RL-categorical-positionalObs	100%
IL-gaussian-positionalObs	86.6%
IL-categorical-positionalObs	100%
RL-gaussian-radarObs	100%
RL-categorical-radarObs	100%
IL-gaussian-radarObs	53.3%
IL-categorical-radarObs	86.6%

Table 4.1: Success rates of the different system setups.

As mentioned above, the IL setup with Gaussian policy acts mostly like the expert demonstrations, and it was the only setup to not show any noticeable oscillating behavior in any of the experiments. However, it has the least success rate on both positional observations and on radar observations. Overall IL, which has to learn more than only the policy, is the more difficult problem. This seems to show in its results as it offers less accuracy. Meanwhile, the RL setup with Gaussian policy has the most consistent and efficient paths, though they are not as similar to expert's.

In the end, one would likely have to make the decision whether the goal is to act as the expert or act the most accurate. Either way, we have shown it is possible to use an end-to-end approach to learn object avoidance with a USV. Though the results were slightly better for RL in terms of quantitative success measures, the IL setup achieves competitive and promising results that might have been improved upon with further testing.

4.3 Discussion

4.3.1 Observations and actions

From our results, we observed that the starting angle, i.e. from which direction (North, South, etc.) the USV approaches the pole, is irrelevant to its performance. The same seems to be the case with which of the two poles the agent is tasked to avoid. This is unsurprising as both poles are represented equally in the observations and because all locations provided to the agent, whether through radar observations

or positional vectors, are relative to its position and heading. This makes the system invariant to starting positions, though not to starting headings. The data set of expert demonstrations all started facing towards the pole. We thus elected to do the same during training both for the full IL setup and for the RL setup.

In our experiments, we used two types of observations, the relative position of the obstacle, that would have needed the use of some sort of scene understanding module, and radar-like images, which worked as a stand-in for downsampled raw radar measurements. Both of these were coupled with the relative position of the goal position, which could be provided by the use of measurements of the USVs position through a GPS and the predetermined position of a goal position. While the IL agents displayed weaker results using radar observations than with the sparser positional observations, the RL agents performed better. We found this interesting because the radar images were generated using only the positional vectors of the obstacles relative to the USV, the very same information which the agent is fed directly in the positional observations. The radar observations thus do not include more information about the agent's state than the positional observations. However, it is possible that the use of images to represent the information combined with our CNN network design for the policy enables better use of the information.

The IL agents, unlike the RL agents, performed worse using radar observations, yet they employ the same observation-policy combinations as their RL counterparts. This may suggest that the problem lies in the discriminator and its feedback to the policy. The discriminator and policy networks are highly similar. However, the discriminator also uses the agent's and the expert's actions when performing its task, in addition to their observations. Unlike the observations, these actions are not normalized before they are inputted into the network. Thus they may have overpowered the other input features and partly hindered the learning process. While this is likely not the sole reason of the IL agents performing being weaker than the RL agents, correcting this flaw in the discriminator, may have improved upon the performance. Sadly, due to the lack of time, we could not test this simple improvement.

4.3.2 GAN

As mentioned in Section 2.2, there are no proofs that GANs will eventually converge and in practice, the generator and the discriminator may end up playing cat and

mouse. In our experiments, we saw that the number of successful episodes varied over the course of the run, increasing and decreasing sometimes drastically from one validation to the next. While this did not seem to be as much of an issue with RL, it was especially noticeable in IL agents. This may be an issue of non-convergence, where the accuracy of the performance would not converge. The overall trajectory of aiming for the goal while swerving for the obstacle did not change after the agent had grasped it. The non-convergence lied mostly in the finesse with which the agent performed the task.

4.3.3 End-to-end learning in practical use

In the traditional modularized USV-setting, a scene-understanding module, a path planning module and a path following module would all be implemented directly onto the USV. With the use of DL systems such as ours, all these modules could be replaced by one policy. Though our system does comprise of several modules, they serve the purpose of training that policy and would thus not directly be implemented on the USV outside of training. This end-to-end policy would drastically reduce the complexity of the USVs system at run-time. However, this, of course, comes at the expense of a time-consuming training session before the policy can be put into use.

An issue with the idea of end-to-end learning using raw data is the potentially massive amount of raw data the system may need to process, especially when combining the raw data of several sensors. For example, it would be reasonable to assume that the combined use of LiDAR data and radar data may provide more information to the agent in each observation, thus the agent may select better actions. However, the sensors output large images that will result in a large observation-space. A large observation-space does not only increase the training time, but it also increases the computational time of the policy. When used in situations that require real-time output this may potentially become an issue. We chose to use images of smaller dimensions than that the ones directly outputted by a radar. This corresponds to downsampling of the raw sensor output which is likely a necessary preprocessing step of the observation data.

When combining data from different sensors into singular observations, the different sampling frequencies of the sensors must be taken into account. This is relevant for both policies learned through RL and IL alike.

4.3.4 RL vs IL

Our results show that both our RL setup and our IL setup is able to grasp the task at hand. However, RL did seem to perform with greater accuracy when performing the task. This indicates that, while our system is able to deduce the underlying goal of the demonstrations, it does not provide the same finesse as its RL counterpart.

The RL agents learned using a simple reward function of our own design. In this type of object avoidance, the crafting of a reward function thus seems to be non-problematic, raising the question of whether using an IL system demands more work than it is worth. However, there are a plethora of other tasks in which a reward function is not as simply crafted as in our case. Designing a reward function for larger and more irregular obstacles may in example prove more difficult than for small regularized obstacles. Another task which may benefit from an IL approach rather than an RL approach may be the docking of the USV, which require a series of finer movements. In this obstacle avoidance scenario, however, we were able to construct a reward function for the task and thus conclude that RL performs better.

We have repeatedly pointed out the problems of defining a reward function for certain tasks, and how IL is released from this issue by introducing the use of expert demonstrations. However, we also need to address the problem of acquiring these expert demonstrations. Though the expert demonstrations relieve the programmer of the task of creating a function to describe the task, it does not relieve them of work. The execution of the task, recording of observations and the construction of demonstration episodes from this, is time-consuming and requires access to the physical agent. The use of expert demonstrations also imposes a set of restrictions on the RL part of the system. For example, the agent's observation type must be equal to the expert's and must be produced through a simulation that is realistic enough to fool the discriminator. These factors mean IL systems are not short-cuts. In some cases, creating and testing a reward function may be the easier route. On other cases, the reward structure may lie so far beyond reach that the IL approach would be the better option.

4.3.5 Model-free systems

Our system uses a policy parameterized by an ANN, specifically, a deep ANN, whose parameters are learned without the use of a model or need of knowledge

about the dynamics of the system. This may be seen as both a positive and a negative aspect.

ANNs are often seen as black boxes. They contain a high number of variables, many in the magnitudes beyond millions, which are fitted to the task guided only by the data they observe and without the direct influence or intentions of the programmer. While this may help avoid the need for a descriptive model of the solution of the task, it also makes the entire process less transparent. It is often hard to know or explain on what basis an ANN draws its conclusions. This is especially true for deeper networks because the initial input is processed in so many stages it quickly becomes unmanageable for a human. Even the maker of the network is often not able to pinpoint what qualities of the input that spurred the specific output or exactly why a certain result was wrongly generated. This may pose some problems when the output of the ANN is unfavorable like we saw in one of our experiments using the IL setup on positional observations. We observed that the policy output suddenly spiked, causing the agent to miss its goal. However, it is unclear as of why this is.

A positive side of model-free DL systems like GAIL and TRPO is that we do not need intricate knowledge about the physics of the system. In this thesis, no focus has been placed on the environments dynamics and physical properties because the policy will learn the behavior without the programmer having to account for them. We did rely on a simulator to train the system, but with the simulator working as a black box for both the system and the programmer. The treatment of the environment as a black box is perhaps even more consistent for IL than for RL, as the programmer may choose or need to account for some dynamics in the manual construction of a reward function. That being said, we still chose to construct our ANNs based on properties of the sensor data in order to make it as small as possible. In this sense, the need to adapt a system to the sensor it relies on is a factor our system shares with the more traditional, modularized, control systems.

Unlike the classification of visual images, we are not looking for textures, colors or whether or not an individual has pointy ears. Our data is radar-based and we wish it to tell us two things: If there is an obstacle, and where is it - whether it is close enough to act. We create our network so that it can exploit the information present in the data. We do not need many convolutional layers to combine textures and shapes, and we do not need to make it invariant to translation. In fact, in our images, the translation carries important information that we want to be extracted. This is part of the reason why our CNNs have only one convolutional layer, few

kernels, and no pooling layers.

4.4 Future Work

We have here only used substitutes for real radar images, due to the problem of generating realistic radar images residing outside of our scope. For future work, however, this may be a natural next step. One possible way of simulating radar measurements could be to transform a map of the surrounding area into a radar image. This could e.g. be done by training a GAN in a similar fashion as the GAN network of Lu et al. (2017) which generates photo-realistic images from sketches. Another alternative could be to simply calculate the angle and distance to every nearby object from a map or some other database. The addition of noise in the measurement would also be relevant as noisy measurements happen from time to time in real radar data. Introducing more variation in the scenarios could also be a natural next step, for example in the size and type of obstacles to be avoided. This could eventually lead to the avoidance of moving obstacles.

In our task, we have simply needed to extract two things from the images: whether there was an object present, and where it was. We had no use for sensitivity to texture or shapes, such as a network trying to determine whether an individual has fur and if its ears are triangular or round. Thus we did not see the need for a large number of layers that, combined, could detect such features. In more complex scenarios, however, where the agent must determine whether it is observing an actual object or simply noise, and what size the object in question is, more layers to the networks would likely be necessary. The agent could distinguish objects from noise based on their texture, and it would have to determine its path based on the size and shape of the obstacle to be avoided.

The goal position which we have used in our scenarios as the point the agent should attempt to reach is in many ways equal to a waypoint. In marine path-following, waypoints often mark the path that the ship should follow. Path following may then be executed through e.g. trying to keep as close to the line between two points as possible. Using such waypoints as goal positions, our system could potentially be used as an alternative to such a path following technique. The system may then also offer automatic obstacle avoidance without the need to generate new waypoints around the obstacle, reducing the number of waypoints. Because our

implementation of GAIL does not generate actions based on a trajectory, but rather on one observation alone, it has no sense of history. The switching of one waypoint as goal position to the next would thus likely not be problematic. A quick test was performed to illustrate the concept using the RL-gaussian-radarObs policy from Table 4.1. No further training was performed before the test was conducted. The resulting path is shown in Figure 4.14.

In order for this extension of use to work properly, one would likely need to include training episodes with no obstacle present, so that the agent is familiarized with this scenario and learns to aim straight for the goal when nothing obstructs its path. It would likely also benefit from introducing starting positions where the USV is not already facing the goal position as this is a likely scenario when switching from one waypoint to another.

Future work could also include the implementation of a fully end-to-end system in the sense of a policy that maps the raw sensor data to a low-level actuator input such as the input of the USVs thrusters. The use of GAIL for acquiring an end-to-end policy could be extended to other USV tasks as well, such as e.g. docking.

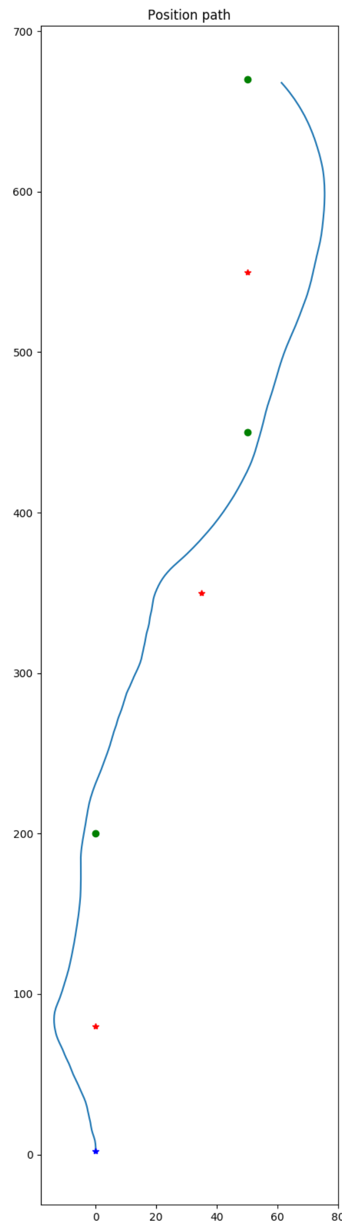


Figure 4.14: Test with waypoint following using the Gaussian policy trained with RL, using radar observations. The green dots mark the waypoints and the red stars mark obstacles the agent must avoid. The numbers along the axis mark the distance from the starting position in meters.

Chapter 5

Conclusions

We have in this project presented two systems that both learn an end-to-end steering model, one learning through Reinforcement Learning (RL) and the use of a manually crafted reward function, and one learning through Imitation Learning (IL) and the use of a set of expert demonstrations. Both of these systems use Deep Learning techniques to learn a policy that maps input observations to steering actions. We have tested our systems on two types of observations, the most notable of which is the use of radar-like images. Both systems show a clear understanding of the task at hand and are able to steer towards a target position while avoiding collision with an obstacle.

The RL system performed at the highest accuracy overall, scoring 100% on our pre-determined success measure. While the problem of learning from demonstrated behavior seems to be the more difficult task, resulting in lower accuracy, the IL system produces results that indicate it is able to grasp the concept of the task and that in many ways are on par with the RL system. We deem this promising for future use in tasks that are not as easily described by a reward function. While object avoidance using radar observations seem to be a task which can be described by a manually crafted reward function without many difficulties, other USV tasks may not be as simple to capture. These tasks may benefit more from the IL approach.

From our results, we conclude that our end-to-end steering model is able to perform the task of obstacle avoidance using radar observations all without the need for a model or insight into the dynamics of the system. Our choice of observations and

action formulation also make our system invariant to the position and past actions of the USV. Such a system could thus potentially be further developed into a path following approach using waypoints with automatic obstacle avoidance.

Appendix A

Additional Theory

This appendix contains additional theory included for the sake of completion and for the readers support. Sections A.1, A.2 and A.3 were originally written by us for the pre-project report preceding this thesis. In this section, the terminology cost, c, is used. Cost is defined as negative reward, $c = -r$

A.1 Maximum Entropy IRL

A challenge in IRL is that the task of finding a cost function that describes the observed behavior is underdefined. Several cost functions might describe the same behavior. For example, a red ball is moved to the left, and onto a green cloth. The motivation behind this action might not easily be discerned. Its movement could be motivated by some rule that red always should go on top of green, or it could be that the ball should not be in direct contact with the table, or it could be that the ball should always move to the left and the cloth was simply there. Although this is a simple example, it illustrates an important issue with learning from observations.

Maximum entropy IRL (Ziebart et al.; 2008) attempts to ease the issue of the underdefined problem in IRL. By making use of the principle of maximum entropy (Jaynes; 1957), they present a probabilistic approach. Intuitively, the algorithm assumes that the optimal policy is the most likely to appear in the dataset and the algorithm will favor the reward function that maximizes the likelihood of the observed expert distribution. This is done by formulating a probabilistic problem.

With τ representing a trajectory, $\tau = \{s_1, a_1, ..., s_t, a_t, ..., s_T\}$ and $R_\phi(\tau)$ the hidden reward for the trajectory, $R_\phi(\tau) = \sum_t r(s_t, a_t)$ where ϕ denotes the reward weights.

$$p(\tau) = \frac{1}{Z} exp(R_\phi(\tau)) \tag{A.1}$$

Z is the partition function, $Z = \int exp(R_\phi(\tau))d\tau$ a normalization constant gained by summing $exp(R(\phi))$ over all paths. A benefit of this is the implicit handling of the uncertainty and noise of the observed path, making the approach more robust.

The expression of the trajectory probability comes as a result of the assumption that trajectories with higher rewards will have exponentially higher probability of occurring in the set of trajectories than those yielding lower rewards. I.e. it expresses the wish to find a reward function such that if the expert were to use it, the likelihood of it receiving high rewards would be exponentially larger than it receiving low rewards for its demonstrations, maximizing the likelihood of the occurrence of the observed trajectories. The weights of the reward function can thus be found through maximizing over the set of demonstrations $\mathcal{D} : \{\tau_i\} \sim \pi^*$

$$\phi^* = \underset{\phi}{\operatorname{argmax}} \sum_{\tau \in \mathcal{D}} \log p_{r_\phi}(\tau) \tag{A.2}$$

The reward function is assumed to be linear. As such, the gradient of the log-likelihood function can be found by the use of dynamic programming.

Formulating or estimating a partition function can be challenging, especially in higher dimension problems or when the dynamics are unknown. Despite this, the Maximum Entropy approach has been heavily used and built upon in the IRL field. In its work on the distribution of trajectories, the algorithm not only provides a way to handle the issue of the underdefined problem but it also somewhat facilitates another issue of IRL. In contrast to the IRL approaches proceeding it, Ziebart et al. (2008) does not assume the expert observations are optimal.

A.2 Deep Maximum Entropy IRL

Deep Maximum Entropy IRL (DeepIRL) (Wulfmeier et al.; 2015), builds upon the Maximum Entropy IRL, using Neural Networks (NN) to approximate the reward

function. In their paper, they advocate the use of NNs in IRL because their layered structure allows for a compact representation of highly nonlinear functions through composition and reuse of results from previous layers. In addition, they point out how NNs computational complexity allows them to scale well to problems with large state spaces and complex cost structures. NNs thus opens the door for end-to-end learning, where the cost may be learned from raw sensor data rather than from hand-crafted features.

Wulfmeier et al. (2015) suggests a Fully Convolutional Neural Network (FCNN) which calculates the cost of each state-action pair. This current cost function is then used to calculate a policy before the policy is propagated through the MDP. From this, the frequency of which each state is visited, can be calculated and compared to that of the demonstrations. Finally, the results can be backpropagated and the wights of the network can be updated before possibly re-iterating the process.

The paper indicated promising results that outperformed methods such as the Gaussian Process IRL of Levine et al. (2011) when it came to complex situations where higher order relationships between features were of large importance. However, like most IRL approaches, DeepIRL uses the transition function in its calculation of the cost and thus requires it to be known beforehand. Another limitation of Wulfmeier et al. (2015) is that they use dynamic programming that traverses over the whole state space to calculate the partition function. In large state spaces, this is highly problematic from a computational view.

A.3 Guided Cost Learning

Motivated by the prospect of using IRL in fields where the dynamics are complex or unknown, such as in robotics, Finn et al. (2016a) presents the Guided Cost Learning algorithm (GCL). GCL uses an adaptive IRL approach that tests the current cost function and policy through training in the environment, samples this and uses the samples to update its environmental model and policy. The cost function is then updated through a sample based approximation of Entropy IRL (Ziebart et al.; 2008) which, like Wulfmeier et al. (2015), expresses the cost function by the use of a Neural Network.

A regular challenge in IRL is how to evaluate the cost function. As pointed out in Zhifei and Joo (2012) and Finn et al. (2016a), a common approach is to find

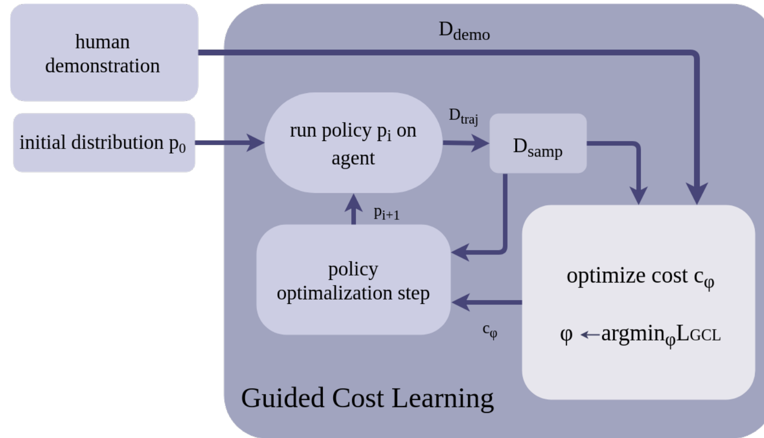


Figure A.1: An illustration of the GCL algorithm. i represents the time step, and ϕ represents the cost parameters.

the optimal policy for the current cost function, a task which has been dubbed 'the forward problem' and compare the actions of this policy to that of the expert. The aforementioned authors note that standard IRL solutions solve this forward problem inside the cost optimizing loop. They argue that finding an optimal policy for a cost function can in itself be a complex and costly task that requires intricate knowledge of the dynamics of the system.

Thus, instead of finding a policy inside the loop searching for the cost, the approach of Finn et al. (2016a) is to find the cost in a loop searching for the policy. In other words, it constructs a cost function that can motivate the expert's behavior as a step in optimizing a policy that can imitate that same behavior. The policy is then tested in the environment and samples are collected. These samples are used to improve the cost function that gives rise to the policy. This way, knowledge of the system dynamics is not required as the agent will estimate them by itself when creating its current policy in the environment. This also grants it the benefit of building knowledge of the dynamics that is closer to the true local dynamics only for the areas where it is most important. GCL gets its name from the fact that the optimizing of the policy guides the sampling towards regions with lower cost.

A result of this approach is that it yields not only the cost but also a trajectory function, $q(\tau)$, that corresponds to a time-varying linear-Gaussian controller which can be used to execute the learned behavior.

A.3.1 The GCL Cost Optimization

In the GCL algorithm, the loop updating the cost function is located inside the loop that performs the policy search. GCL builds upon Ziebart et al. (2008) and assumes the demonstrated trajectories, τ_i, are drawn from the distribution

$$p(\tau) = \frac{1}{Z} exp(-c_\phi(\tau)) \tag{A.3}$$

where $c_\phi(\tau) = \sum_t c_\phi(x_t, u_t)$, the sum of the cost for each state, x_t, and action, u_t, pair that make up the trajectory. Because the partition function, $Z = \int exp(R_\psi(\tau))d\tau$, is unknown, it is estimated from samples of a background distribution, $q(\tau)$ which is estimated as part of the policy optimizing part of the GCL algorithm.

The GCL objective can then be expressed using the set of N demonstrations, $\mathcal{D}_{demo}$, and M samples, $\mathcal{D}_{samp}$ as denoted below (Finn et al.; 2016a)

$$\mathcal{L}_{GCL}(\phi) = \frac{1}{N} \sum_{\tau_i \in \mathcal{D}_{demo}} c_\phi(\tau_i) + \log Z \tag{A.4}$$

$$\approx \frac{1}{N} \sum_{\tau_i \in \mathcal{D}_{demo}} c_\phi(\tau_i) + \log \frac{1}{M} \sum_{\tau_j \in \mathcal{D}_{samp}} \frac{exp(-c_\phi(\tau_j)}{q(\tau_j)} \tag{A.5}$$

Finn et al. (2016a) then compute the corresponding gradients, with respect to the cost parameters, ϕ, and denoting $w_j = \frac{exp(-c_\phi(\tau_j)}{p(\tau_j)}$ and $Z = \sum_j w_j$

$$\frac{d\mathcal{L}_{GCL}}{d\phi} = \frac{1}{N} \sum_{\tau_i \in \mathcal{D}_{demo}} \frac{dc_\phi}{d\phi}(\tau_i) - \frac{1}{Z} \sum_{\tau_j \in \mathcal{D}_{samp}} w_j \frac{dc_\phi}{d\phi}(\tau_j) \tag{A.6}$$

The parameters of the cost function, ϕ, can thus be updated iteratively, using this gradient. As the cost function is represented by a NN, Finn et al. (2016a) back propagate $-\frac{w_j}{Z}$ for each trajectory $\tau_j \in \mathcal{D}_{samp}$ and $\frac{1}{N}$ for each trajectory $\tau_i \in \mathcal{D}_{demo}$ and update the weights of the NN based on the resulting gradients.

A.3.2 The GCL Policy Optimization Step

GCL optimize both a cost function and a policy. As part of the algorithm, a policy is optimized as a step in estimating the background distribution $p(\tau)$ used in the cost optimization. As described in Finn et al. (2016a), this step in the GCL algorithm uses the method of Levine and Abbeel (2014). Their method is highly similar to the GPS algorithm described in Section **??**. Similarly to the GPS approach described in Levine et al. (2016), the policy optimization procedure consists of iteratively generating samples from $p(u_t|x_t)$, fitting the dynamics, $p(x_{t+1}|x_t, u_t)$, to these samples, and updating $p(u_t|x_t)$ under these fitted dynamics. One difference between the method of Levine and Abbeel (2014) and Levine et al. (2016) is that the latter offers the simultaneous learning of a global, observation-based policy π_θ.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning, *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, pp. 1–8.

Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L. D. and Muller, U. (2017). Explaining how a deep neural network trained with end-to-end learning steers a car, *CoRR* **abs/1704.07911**.

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

Chi, L. and Mu, Y. (2017). Learning end-to-end autonomous steering model from spatial and temporal visual cues, *VSCC 2017 - Proceedings of the Workshop on Visual Analysis in Smart and Connected Communities, co-located with MM 2017*, pp. 9–16.

Cybenko, G. (1981). Approximations by superpositions of sigmoidal functions, *Mathematics of Control, Signals, and Systems* **2**: 303–314.

Finn, C., Christiano, P. F., Abbeel, P. and Levine, S. (2016b). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models, *CoRR* .

Finn, C., Levine, S. and Abbeel, P. (2016a). Guided cost learning: Deep inverse optimal control via policy optimization, *33rd International Conference on Machine Learning, ICML 2016*, Vol. 1, pp. 95–107.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. and Pineau, J. (2018). An introduction to deep reinforcement learning, *CoRR* .

Fu, J. (2018). Implementation of gail's discriminator update scheme. Software available from https://github.com/justinjfu/inverse_rl/blob/master/inverse_rl/models/imitation_learning.py. **URL:** *https://github.com/justinjfu/inverse_rl/blob/master/inverse_rl/models/imitation_learning.py*

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

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative adversarial nets, *Advances in Neural Information Processing Systems* **27**: 2672–2680.

Goodfewllow, I. (2016). Nips 2016 tutorial: generative adversarial networks.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances, *INFORMS Journal on Computing* **21**(2): 178–192.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J. and Chen, T. (2018). Recent advances in convolutional neural networks, *Pattern Recognition* **77**: 354–377.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning, *CoRR* .

Hunter, D. R. and Lange, K. (2004). A tutorial on MM algorithms, *The American Statistician* **58**(1): 30–37.

Hussein, A., Gaber, M. M., Elyan, E. and Jayne, C. (2017). Imitation learning: A survey of learning methods, *ACM Computing Surveys* **50**(2).

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, *32nd International Conference on Machine Learning, ICML 2015*, Vol. 1, pp. 448–456.

Jaynes, E. (1957). Information theory and statistical mechanics, *Physical Review* **108**(2): 171.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning, p. 267–274.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization, *International Conference on Learning Representations* .

Kober, J., Bagnell, J. A. and Peters, J. (2013). Reinforcement learning in robotics: A survey, *International Journal of Robotics Research* **32**(11): 1238–1274.

Konda, V. (2002). *Actor-critic Algorithms*, PhD thesis, Cambridge, MA, USA.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, Vol. 2, pp. 1097–1105.

Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics, *Advances in Neural Information Processing Systems*, Vol. 2, pp. 1071–1079.

Levine, S., Finn, C., Darrell, T. and Abbeel, P. (2016). End-to-end training of deep visuomotor policies, *Journal of Machine Learning Research* **17**.

Levine, S., Popović, Z. and Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*.

Lu, Y., Wu, S., Tai, Y. and Tang, C. (2017). Sketch-to-image generation using deep contextual completion, *CoRR* .

Martinsen, A. B. and Lekkas, A. M. (2018). Straight-path following for underactuated marine vessels using deep reinforcement learning, *IFAC-PapersOnLine* **51**(29): 329–334.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning, *CoRR* .

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). Human-level control through deep reinforcement learning, *Nature* **518**(7540): 529–533.

OpenAI:SpinningUp (2018). Implementation of the trpo algorithm. Implemented by by Josh Achiam.
URL: *https://github.com/openai/spinningup/blob/master/spinup/algos/trpo/trpo.py*

Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton, *Cornell Aeronautical Laboratory* .

Schmidhuber, J. (2015). Deep learning in neural networks: An overview, *Neural Networks* **61**: 85–117.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search, *Nature* **529**(7587): 484–489.

Squire, L., Berg, D., Bloom, F., du Lac, S., Ghosh, A. and Spitzer, N. (2008). *Fundamental Neuroscience, 3rd edition*, Elsevier.

Sutton, R. S. and Barto, A. G. (2017). *Reinforcement Learning: An Introduction, 2nd edition*, MIT Press Ltd.

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

Tensorflow home page (2017). [Online; accessed 13-December-2018].
URL: *https://www.tensorflow.org/*

Vedeler, A. S. (2018). Pre project: Learning an end-to-end steering model for an unmanned surface vehicle.

Wang, K., Gou, C., Duan, Y., Lin, Y., Zheng, X. and Wang, F.-Y. (2017). Generative adversarial networks: introduction and outlook, *IEEE/CAA Journal of Automatica Sinica* **4**(4): 588–598.

Wulfmeier, M., Ondruska, P. and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning, *CoRR* .

Wulfmeier, M., Wang, D. Z. and Posner, I. (2016). Watch this: Scalable cost-function learning for path planning in urban environments, *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2089–2095.

Zhang, M. (2017). Model-based reinforcement learning. [Online; posted 27-September-2017].
URL: *https://michaelrzhang.github.io/model-based-rl*

Zhifei, S. and Joo, E. M. (2012). A survey of inverse reinforcement learning techniques, *International Journal of Intelligent Computing and Cybernetics* **5**(3): 293–311.

Ziebart, B. D., Maas, A., Bagnell, J. A. and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning, *Proceedings of the National Conference on Artificial Intelligence*, Vol. 3, pp. 1433–1438.

Alexandra Skau Vedeler

Learning an End-to-End Steering Model for an Unmanned Surface Vehicle

NTNU

Kunnskap for en bedre verden

FFI Forsvarets forskningsinstitutt

Norwegian Defence Research Establishment