MASTER'S THESIS

Reinforcement Learning For Offshore Crane Set-down Operations

Author:
Mingcheng DING

Internal Supervisor:
dr. M. A. WIERING

External Supervisor:
Ruben de BRUIN
Marcelo Huet MUNOZ

Master of Artificial Intelligence

Bernoulli Institute
Faculty of Science and Engineering
UNIVERSITY OF GRONINGEN

and

HEEREMA MARINE CONTRACTORS

December 9, 2018

university of groningen / faculty of science and engineering / bernoulli institute

MARINE CONTRACTORS

Disclaimer

All numbers used in this report do not reflect the actual property of HMC assets but are considered as a good all-in-all representation of the reality. The usage of all images regarding HMC vessels and projects are restricted. They are not allowed to be used for any purpose without the permission of HMC.

UNIVERSITY OF GRONINGEN

Abstract

Faculty of Science and Engineering
Bernoulli Institute

Master of Artificial Intelligence

Reinforcement Learning For Offshore Crane Set-down Operations

by Mingcheng DING

Offshore activities are usually carried out in one of the worst working environments where vessels and objects are affected by weather all the time. One of the most common offshore operations is to set down a heavy object onto the deck of a floating vessel. A good set-down always requires a small impact force as well as a short distance to the target position. It can be quite challenging to achieve due to various reasons, such as ship motions, crane mechanics, and so forth. It takes years to train crane operators to make as many correct decisions as possible. Any small mistake might cause severe consequences. In this project, we investigated the feasibility of solving this practical offshore set-down problem using Reinforcement Learning (RL) techniques. As a feasibility study, we started from the simplest possible environment where only the heave motion and impact velocity are considered. Then, we gradually upgraded the simulation environment by adding more environmental and physical features with respect to a practical situation. The results under different environments bring us an overview of the possibilities and limitations of standard RL algorithms. We demonstrated that the methods suffer from the general challenges of RL, such as sparse rewards and sample efficiency in solving the long-term objective set-down problem. We tried various methods to work around this issue, such as transfer learning, hierarchical RL, and using simulation-based methods.

Acknowledgements

Thank you to everyone who helped me to finish this thesis in the last 9 months. First, I want to thank my supervisors within Heerema Marine Contractors, Ruben de Bruin and Marcelo Huet Munoz. They helped me to form the basic knowledge of the offshore engineering, and gave me so many smart ideas on designing the experiments and environments with their rich expertise in this field. Furthermore, I appreciate them helping me to quickly adapt to the new working environment within HMC as well as the city life in Leiden.

Secondly, I would like to thank my supervisor from the University of Groningen, dr. Marco Wiering. Marco was so willing to answer all of my questions (some questions now look really "stupid") and gave me a lot of useful advise on the theoretical part of this project. This was really important because those theories were in general not easy to be discussed within an engineering company like HMC.

I also want to thank again all aforementioned three people for reviewing and commenting on my report, which inspired me a lot on improving the content and the language of the report. Finally, I want to thank my girlfriend, Qing Liu, who stayed with me during my thesis and made me feel less lonely as a foreign expat and of course, as always, I appreciate the support from my parents all the way from China.

Mingcheng Ding
November 2018

Contents

List of Figures

List of Tables

List of Abbreviations

CoG	**C**enter **o**f **G**ravity
DAGGER	**DA**taset **AGGRE**gation
DNN	**D**eep **N**eural **N**etwork
DoF	**D**egree **o**f **F**reedom
DP	**D**ynamic **P**ositioning
DQFD	**D**eep **Q**-learning **F**rom **D**emonstration
DQN	**D**eep **Q** **N**etwork
DDQN	**D**ouble **D**eep **Q** **N**etwork
FA	**F**unction **A**pproximator
GAE	**G**eneralized **A**dvantage **E**stimator
HMC	**H**eerema **M**arine **C**ontractors
HRL	**H**ierarchical **R**einforcement **L**earning
IFT	**I**nverse **F**ourier **T**ransform
JONSWAP	**JO**int **N**orth **S**ea **WA**ve **P**roject
LSTM	**L**ong **S**hort **T**erm **M**emory
MC	**M**onte **C**arlo
MCTS	**M**onte **C**arlo **T**ree **S**earch
MDP	**M**arkov **D**ecision **P**rocess
MLP	**M**ulti **L**ayer **P**erceptron
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
PPO	**P**roximal **P**olicy **O**ptimization
RAO	**R**esponse **A**mplitude **O**perator
RL	**R**einforcement **L**earning
RNN	**R**ecurrent **N**eural **N**etwork
RPM	**R**evolutions **P**er **M**inute
SARSA	**S**tate **A**ction **R**eward **S**tate **A**ction
SMDP	**S**emi- **M**arkov **D**ecision **P**rocess
SSCV	**S**emi- **S**ubmersible **C**rane **V**essel
TD	**T**emporal **D**ifference
TL	**T**ransfer **L**earning
TRPO	**T**rust **R**egion **P**olicy **O**ptimization
UCT	**U**pper **C**onfidence **T**ree
WOW	**W**aiting **O**n **W**eather

Chapter 1

Introduction

1.1 Offshore Set-Down Operation

The term of commercial offshore industry was first introduced in the 1940s when the global demand for oil was booming, and the petroleum industry thus realized the opportunity of creating a whole new way of producing natural oil (Schempf, 2007). Typical offshore construction incorporates the installation and transportation of offshore structures in a marine environment. Modern offshore structures include fixed or submersible platforms, floating platforms, offshore wind power, and submarine pipelines (Reedy, 2012).

Offshore structures are installed by crane vessels with lifting capacities of up to 14,000 tons (Mouhandiz and Troost, 2013). Now modern crane vessels are semi-submersibles and have good stability, making them less sensitive to sea swells and harsh weather. Some vessels are equipped with more than one heavy-lifting crane, which allows operations to be done in tandem for heavier loads and better controls.

One of the most common offshore operations is to set down a heavy object onto a target position, which is either a floating vessel or a fixed platform. A good set-down always requires a small impact force as well as a short distance to the target position. In reality, a good set-down can be quite challenging to achieve due to various reasons, such as ship motions, crane mechanics, crew communication and so forth. Most of the actions and decisions during the set-down are made based on the spatial arrangement and dynamics between objects. The crane operator should not only be accurate at operating the effectors of the crane, but also capable of understanding ship motions in order to judge the best moment to take different actions. The set-down can only be done with excellent cooperation between these two factors. In this project, we simulated an ordinary heavy-lifting set-down operation where the load is simply a cubic suspended from the vessel (*Sleipnir*), and the target position is on the deck of a floating barge indicating by a horizontal guide. The motion of the vessel is only affected by the unidirectional wave and the actions taken during the operation.

1.2 Reinforcement Learning

Reinforcement learning (RL) is a field of machine learning that is aimed at training an artificial agent to achieve the maximum accumulated reward in a specific environment by taking

a sequence of actions (Wiering and van Otterlo, 2012). At every time step, the agent is situated in a particular state, and a reward is assigned after it takes an action and reaches the next state. The agent keeps interacting with the environment until it reaches the terminal state. An action is sampled from the policy which is a distribution over all possible actions given the current state. The policy can be represented by means of a look-up table, a neural network or any type of machine learning technique. This type of learning provides solutions to many classical control problems, such as the mountain-car and cart-pole problems.

Recently, a great deal of research in deep neural networks and computer vision has shown that RL has a strong potential to generate good policies even in high-dimensional, complex input spaces, such as images (Arulkumaran et al., 2017). During the set-down operation, multiple sensory signals are captured simultaneously. These time-series signals particularly describe the current situation of objects as well as the underlying dynamics of the environment. This creates the possibility of training such an agent in the nature of RL.

1.3 Heerema Marine Contractors

Heerema Marine Contractors (HMC) is the world's leading marine contractor based in Leiden, the Netherlands. The company is specialized in delivering high-quality solutions to issues related to transportation, installation, removal of offshore facilities. In addition, HMC occupies the entire supply chain from design to construction. HMC currently owns three of the world's top 10 crane vessels, which are "Thialf" (14,200- ton lifting capacity), "Balder" (8,100-ton lifting capacity) and "Aegir" (4,000-ton lifting capacity). The new flagship, "Sleipnir", is being assembled and will be introduced to the market in 2019.

1.4 Project Motivation and Scope

There are several motivations for this project. With regard to engineering, if the quality of set-down could be controlled consistently by the assisting machine learning algorithms, the engineering limits can become less restricted, meaning that HMC can use lighter bumpers for guiding the target and this extends the life of a barge. On the other hand, the offshore set-down is a highly empirical and complex activity that requires outstanding operating and perception skills. Even crane operators are unable to explain many of their actions, which they perform instinctively. Hence, the result of this project can contribute to gaining insight of their behaviors for further studies.

Therefore, this project is mainly focused on exploring to what extend the complex set-down operation can be simulated by using machine learning techniques especially in the field of RL. Since this whole research study was fully carried out from scratch, we began the study from an extremely simplified environment and gradually moved to a more realistic environment. Given the time and physical constraints, all the experiments were conducted exclusively in simulators provided under the license of HMC.

1.5 Research Questions

The main research question of this project is as follows:

How can machine learning techniques be combined to simulate manual set-down operations of offshore cranes?

There are two major concerns with regard to the main question: the simulation environment and the algorithm itself. To create a valid environment, it is necessary to understand how the objects and vessels are moving in reality. Furthermore, since the set-down is a physical procedure, it is necessary to make a metric for evaluating the result of the set-down such that the agent can improve based on this signal. The first two sub-questions are as follows:

1. What are the main factors and limits that form the offshore operating environment? How should we model and extend those limits for the simulation environment of Reinforcement Learning?
2. What is the metric and how should we shape the reward/quality of the set-down operation in terms of physical phenomena?

Due to the environmental setting, the result of a set-down is normally given at the end of every attempt, which causes a delayed credit assignment. Additionally, based on the way we built the environment, the dynamics of the environment and reward function are fully defined, which allows us to employ search-based methods. In this sense, the main research question is divided into the following sub-questions:

1. How can we deal with the sparse/delayed reward in each of the set-down simulations?
2. How can the performance of the agent be improved by learning with Monte Carlo tree search?
3. To what extent can the simulation environment be upgraded toward the real-world and what is an effective way to deal with partial observability in the environment?

1.6 Outline

The thesis contains eight chapters. Chapter 2 mainly covers the basics about the hydrodynamics and wave statistics in the simulation environment for the set-down. It contributes to the solution to the first sub-question. Chapter 3 reviews the theoretical frameworks of RL algorithms that are relevant to the experiments. Each of the chapters from Chapter 4 to Chapter 7 contains an introduction of the features, the experiments, and related results that answer the rest of the sub-questions. Chapter 8 includes a discussion and draws conclusions of this project.

Chapter 2

Offshore Environment and Set-Down Operation

An offshore set-down is usually executed between two independent structures. In the maritime environment, ocean surface waves cause motions on every floating structure in the sea, and thus the offshore operation has to take into consideration the motion of the effector as well as the ships themselves. In order to build such an environment, it is important to analyze the mathematical properties of the waves and how they affect ship motions.

2.1 Irregular Wave Statistics

Mathematically, irregular waves can be represented by linear superpositions of multiple regular wave components. Regular waves are harmonic waves traveling with kinematic and potential energy. Figure 2.1 shows a single harmonic wave component. The peak position is the *crest*, and the lowest point is the *trough*. The *amplitude* of the wave ζ_a is the distance from the mean water level to the crest. The wave height H is calculated vertically from crest to trough. For a sinusoidal wave, H is twice ζ_a. The *wave length* λ is measured from the distance between two consecutive crests. In the time domain, the wave length is described by the *wave period* T. The total energy per unit area of a regular wave can be represented by:

$$E = \frac{1}{2}\rho g \zeta_a^2, \tag{2.1}$$

where ρ and g are water density and gravitational acceleration.

The irregular wave elevation is generated by the linear summation along a series of sin and cos functions. Due to the superposition, the absolute wave period varies in every measurement. Statistically, irregular waves are described with a set of estimated variables over a certain period of time called sea states. In practice, the length of the recording should be at least 20 minutes being sampled every half second (Journée et al., 2000). The most commonly used variables are *significant wave height* H_s and *peak wave period* T_p. The significant wave height H_s is the mean of one-third of the highest wave heights in the recording. The H_s provides an good approximation of the most probable wave height in a time period. The T_p is the wave period with the most energy.

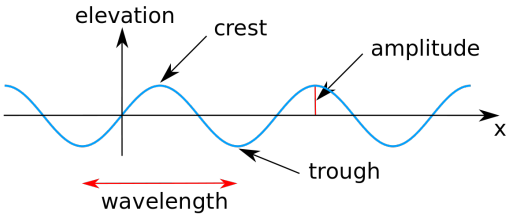


FIGURE 2.1: Description of regular sea wave.

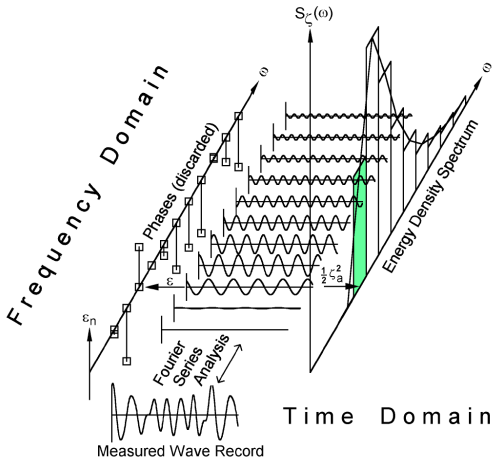


FIGURE 2.2: Wave energy spectrum describes energy per frequency per unit sea surface area

In principle, given a measured irregular wave for T seconds, if one can derive the coefficients of every frequency component, then this typical irregular wave can be re-generated by:

$$\zeta(t) = \Sigma_{n=1}^{N}(a_n cos(\frac{2\pi n}{T}t) + b_n sin(\frac{2\pi n}{T}t)), \tag{2.2}$$

where a_n and b_n are real and imaginary part of the coefficient. Notice that:

$$a_n cos(\frac{2\pi n}{T}t) + b_n sin(\frac{2\pi n}{T}t = A_n cos(\frac{2\pi n}{T}t - \beta_n), \tag{2.3}$$

where

$$A_n = \zeta_n = \sqrt{a_n^2 + b_n^2}, \beta_n = \epsilon_n = tan^{-1}(b_n, a_n), \tag{2.4}$$

$$\zeta(t) = \Sigma_{n=1}^{N}\zeta_n cos(\omega_n t - \epsilon_n), \tag{2.5}$$

in which ζ_n is the amplitude of component n. $\omega_n(\frac{2\pi}{T}n)$ and ϵ_n are the radian frequency and the phase angle of component n. Therefore, given a measured wave elevation for a long period, one can carry out a Fourier analysis to transform the wave into the frequency domain by plotting amplitude with respect to frequency, and thus results the wave amplitude spectrum. However, the instantaneous wave amplitude is a random variable that is parameterized by a Gaussian distribution with zero mean. By a small time shift, one might find a different series of ζ_n. It can be mitigated by calculating the mean of several squared amplitude components $\bar{\zeta_n^2}$. Multiplied with constants ρ and g (see equation 2.1), the wave spectrum can be expressed as the *wave energy spectrum* $S_\zeta(\omega_n)$ (see Figure 2.2) by:

$$S_\zeta(\omega_n)\Delta\omega = \Sigma_{\omega_n}^{\omega_n+\delta\omega}\frac{1}{2}\zeta_n^2(\omega), \tag{2.6}$$

where $\Delta\omega$ is the difference between two consecutive frequencies. If $\Delta\omega$ goes to 0, the energy spectrum extends to a continuous function defined by:

$$S_\zeta(\omega_n)d\omega = \frac{1}{2}\zeta_n^2. \tag{2.7}$$

Based on the energy spectrum from the measured wave, one can generate new wave elevations by using the inverse Fourier transform to compute amplitudes and assign a random phase angle ϵ_n to every wave component. The wave amplitude ζ_n can be calculated by:

$$\zeta_n = 2\sqrt{S_\zeta(\omega_n)\cdot\Delta\omega}, \tag{2.8}$$

where $\Delta\omega$ represents the interval between two discrete frequencies. Thus, an artificial series of wave elevation should carry the same energy as a measured one.

In practice, there are many theoretical energy spectra to represent ocean waves. The one used for this project is the Joint North Sea Wave Project (JONSWAP), which was carried out about 100 miles from the coast into the North Sea. The project began in 1968 and 1969 with an extensive measurement of energy waves. The formulation of the JONSWAP wave energy spectrum (Journée et al., 2000) requires two aforementioned sea-state variables, H_s and T_p,

$$S_\zeta(\omega) = \frac{320\cdot H_s^2}{T_p^4}\cdot\omega^{-5}\cdot e^{\frac{-1950}{T_p^4}\cdot\omega^-4}\cdot\gamma^A, \tag{2.9}$$

with

$$\gamma = 3.3, A = e^{-(\frac{\frac{\omega}{\omega_p}-1}{0.08\sqrt{2}})^2}, \omega_p = \frac{2\pi}{T_p}.$$

This can be considered as a tool to approximate the energy distribution of the waves in the North Sea given the intended sea state. It is extremely helpful for simulating the ship motions of an offshore activity that will take place in regions with the same sea state. The resultant spectrum is thus a function of ω, and is drastically influenced by the input sea states. Eventually, the wave elevation is generated by the linear superposition of every wave amplitude ζ_n derived by equation 2.8. It is worth noting that, in this project, all the waves are assumed to be unidirectional. In other words, all waves are coming from the same direction, which is called a long-crested sea.

2.2 Vessel Dynamics

Suppose a vessel is moving in a constant speed, the origin is at the center of gravity (CoG), the (x,y) plane is parallel to the sea surface at origin and z is pointing upwards. The coordinate system is called *steadily translating*, which follows the right-handed orthogonal rule.

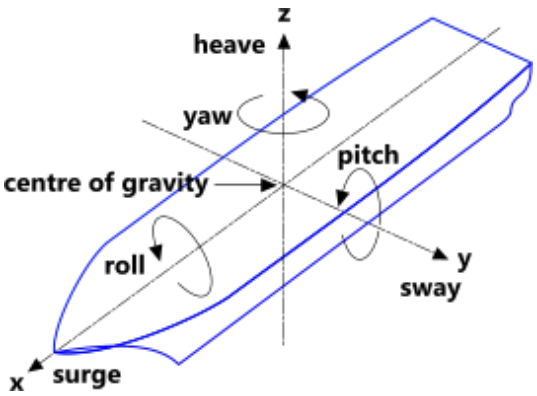


FIGURE 2.3: Definitions of ship motions in six degrees of freedom.

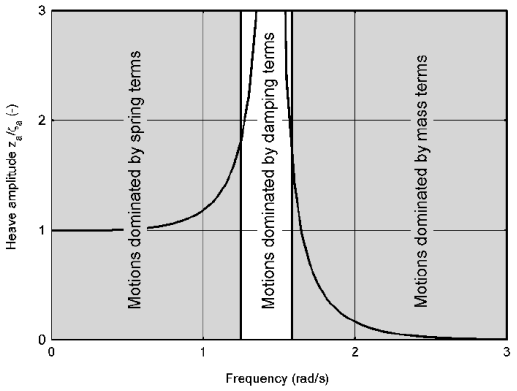


FIGURE 2.4: RAO is a transfer function of ω for each of six motions

The ship motions are defined in 6 degrees of freedom (DoF) (see Figure 2.3), which consist of three translations of CoG in the direction of the x-, y- and z-axes and three rotations with respect to the x-, y- and z-axes. Formally, we assume that waves are regular for now. Then, wave elevation at origin is defined as $\zeta(t) = \zeta_a cos(\omega t)$, where ζ_a is the wave amplitude. Then the motions at CoG are given as follows:

$$\begin{aligned} Surge : x &= x_a cos(\omega_t + \epsilon_{x\zeta}), \\ Sway : y &= y_a cos(\omega_t + \epsilon_{y\zeta}), \\ Heave : z &= z_a cos(\omega_t + \epsilon_{z\zeta}), \\ Roll : \phi &= \phi_a cos(\omega_t + \epsilon_{\phi\zeta}), \\ Pitch : \theta &= \theta_a cos(\omega_t + \epsilon_{\theta\zeta}), \\ Yaw : \psi &= \psi_a cos(\omega_t + \epsilon_{\psi\zeta}). \end{aligned} \tag{2.10}$$

The motion of each individual DoF has the same frequency as the encountering wave, but the actual motion amplitude is dependent on the response and phase shift (e.g., for heave motion, the amplitude and phase shift are z_a and $\epsilon_{z\zeta}$). The convention for phase shifting is that if the motion happens earlier than the wave elevation passes zero, then its corresponding phase shift is positive and negative otherwise. If a ship is a rigid body, when the six motions at CoG are determined, the motion at any location $P(x_b, y_b, z_b)$ on the ship is given by:

$$\begin{aligned} x_P &= x - y_b\psi + z_b\theta, \\ y_P &= y + x_b\psi - z_b\phi, \\ z_P &= z - x_b\theta + y_b\phi, \end{aligned} \tag{2.11}$$

where x, y, z, ϕ, θ, and ψ are surge, sway, heave, roll, pitch, and yaw motion at CoG of the vessel. For a set-down, the point of interest (PoI) is usually at the tip of the boom where

the heavy object is hung. Thus, it can be directly transferred from the motions of CoG. Therefore, the key issue is to calculate the motions of CoG. For the rest of this section, we only analyze the motion on the z-axis (heave). The other motions can be studied in similar ways.

According to linear wave theory, any irregular wave amplitude can be achieved by the summation over amplitudes of different regular waves. Recalling the derivation of wave energy spectra $S_\zeta(\omega)$, one could also derive the ship motion induced by irregular waves in terms of the motion response of every regular wave component. This is called the *motion response spectrum* $S_z(\omega)$ of the vessel. The motion response spectrum can be simply obtained from wave energy spectra via a transfer function,

$$\begin{aligned} S_z(\omega) \cdot d\omega &= \frac{1}{2} z_a^2(\omega) \\ &= |\frac{z_a}{\zeta_a}(\omega)|^2 \frac{1}{2} \zeta_a^2(\omega) \\ &= |\frac{z_a}{\zeta_a}(\omega)|^2 S_\zeta(\omega) d\omega. \end{aligned} \tag{2.12}$$

The transfer function $|\frac{z_a}{\zeta_a}(\omega)|^2$ is called the response amplitude characteristic. In offshore engineering, it is also known as the *Response Amplitude Operator* (RAO). In particular, the coefficients of the RAO are dependent on the hydromechanics properties of the vessel. It is essentially a function of regular wave frequencies ω_n as defined in the wave energy spectrum:

$$\frac{z_a}{\zeta_a} = e^{-kT} \sqrt{\frac{(c - a\omega^2) + (b\omega)^2}{(c - (m+a)\omega^2)^2 + (b\omega)^2}}, \tag{2.13}$$

where a, b, and c are added mass, damping, and stiffness coefficient. It outputs a relative ratio between the absolute motion response and the wave amplitude for a given regular wave frequency. For example, a heave RAO of 0.5 in a wave amplitude of 2m indicates that the vessel has an up-and-down motion (heave) from -1m to +1m from origin. A pitch RAO of 2 in the same wave means that the vessel has rotation around y-axis from -4 to + 4 degrees.

Knowing the motion response spectrum $S_z(\omega)$, the total motion can then be calculated by adding the motion responses of every individual wave component:

$$\begin{aligned} z_a(\omega) &= \sqrt{2 S_z(\omega) \cdot \Delta\omega}, \\ z(t) &= \Sigma_{n=1}^{N} z_n cos(\omega_n(t) - \epsilon_{z_n}). \end{aligned} \tag{2.14}$$

When the motion response spectra for all the six motions are presented, one can generate artificial time traces at CoG for the six DoFs. According to the principle of transformation, the time traces of motion of any PoI can be derived by equation 2.11. Hence, by far, we can fully generate the artificial wave and resultant vessels motions given a particular sea state in absence of any additional forces.

In summary, the pipeline of creating motions in the simulation environment is as follows: First, create a theoretical continuous wave energy spectrum (by the formulation of JONSWAP). Second, derive the RAOs of the vessel (which are directly available in HMC) and generate a response spectra transferred from the wave energy spectrum. Finally, generate the time traces of motions of interest by the inverse Fourier transform with random phases, as shown in equation 2.14.

As a preliminary study, in the simulation environment, the waves were unidirectional, which means that only up to three motions (two translation and one rotation) were applied on a vessel. Two translational motions are heave and surge in y-z plane, and the rotational motion is the pitch about x-axis. The RAOs were chosen from one of the crane vessels owned by HMC. Each of the vessel motions was expected to be non-periodical since the phases of regular wave components were randomized.

2.3 Crane Vessel Properties

Offshore construction is mainly completed by crane vessels. Figure 2.5 presents a general arrangement of one of the crane vessels owned by HMC-*Sleipnir* from the port side. Normally, the crane vessels used for heavy-lifting activities are semi-submersible crane vessels (SSCV).

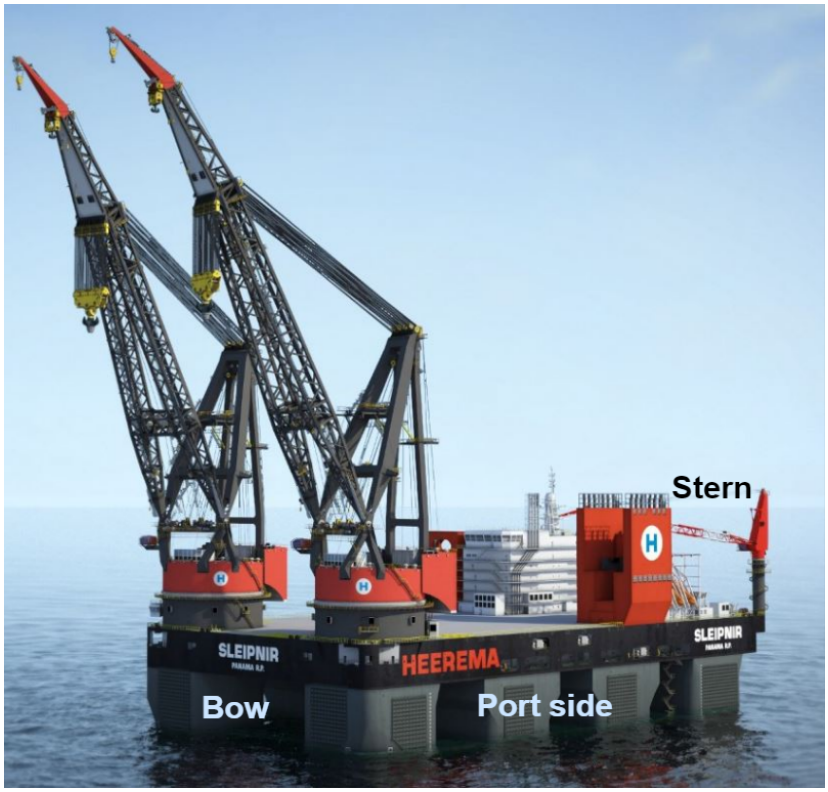


FIGURE 2.5: SSCV *Sleipnir*

Compared to normal floating vessels, the SSCVs offer better motions on the deck. Such response is obtained mainly by its hull shape of the pontoon under the water. The pontoons are connected to the four columns. Due to its water area plane, the SSCV has a different behavior than the hull of normal vessels. This makes SSCVs less responsive to worst sea waves. Additionally, SSCVs keep their own position by thrusters and dynamic position (DP) systems. These systems ensure the station keeping while lifting extremely heavy objects.

On the deck, there are two main cranes with heavy-lifting capacity. The crane is installed at the corner of the deck (bow side). For heavy-lifting activities, the load is hung by *blocks* (see Figure 2.6). Depending on the weight and technical specifications of the load, there are three blocks for different purposes. The whip block has the fastest speed and the longest reach with the least lifting capacity. The main block offers the largest lifting capacity, but its operating speed is the slowest. The auxiliary block is faster than the main block, but its lifting capacity is smaller.

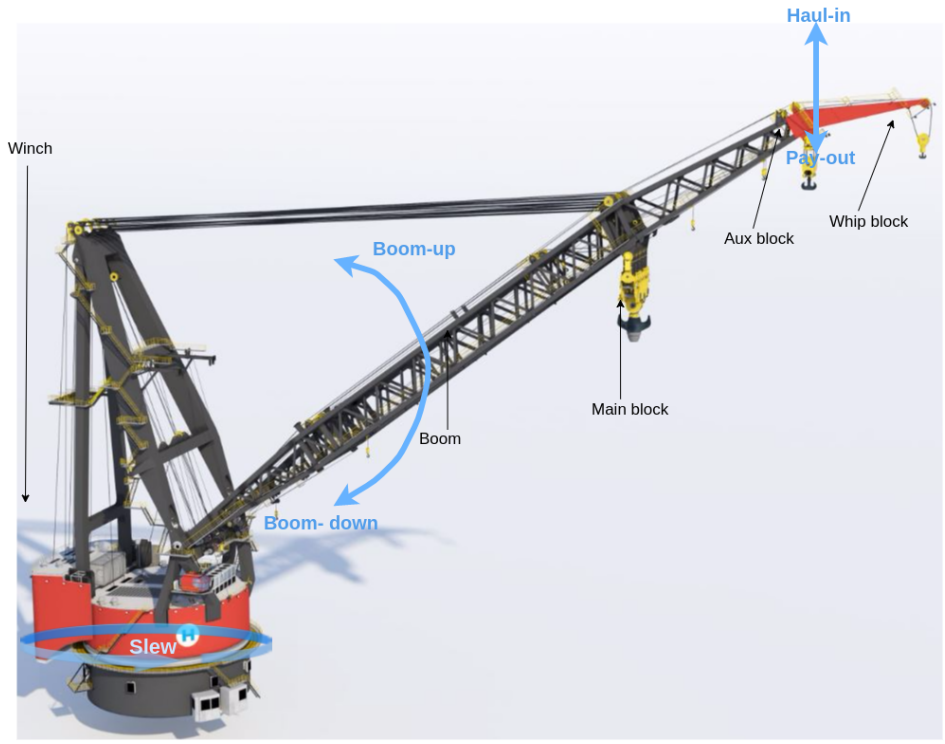


FIGURE 2.6: Heavy-lifting crane on *Sleipnir*

Blocks are connected to the boom by hoist wires. Hoist wires are reeled at the *winches* on the backside of crane cabin. The action to lower the block is called *payout*, where the winch extends length of hoist wire to the block; conversely, for the lifting operation, the crane operator has to *haul-in* the hoist wire. The maximum payout/haul-in speed of each block is dependent on the angular velocity of the connected winch. Since electrical motors drive the winch, it takes a small amount of time to actually reach max speed. The delayed time for reaching maximum RPM is referred to as *ramp-up time*. Ramp-up time varies between different blocks. For real applications where ship motions are presented, it is critical to consider

the dynamics within the period of ramp-up, which requires an additional prediction of ship motions.

The blocks are suspended from the boom. The boom can move independently to go up (boom up) or down (boom down). In this project, we assume that the position of the boom always remains fixed. Furthermore, the boom can be rotated about the z-axis with respect to the deck, which is known as *slewing*. With the rotation in both directions, it transports the load to the target position of the set-down. Note that the slewing speed is also affected by the ramp-up time.

In the simulation environment, for every time step, the agent is only allowed to operate one of five actions: payout, haul-in, slew left or slew right, and, of course, do nothing. The ramp-up and motion are based on HMC vessels. More details with concern to the environmental configurations are covered in Chapter 4 to Chapter 7.

2.4 Prerequisites of Set-Down

A real set-down operation is presented in Figure 2.8. In practice, the completion of a set-down comprises several pre-requisites. First of all, the weather condition. The weather conditions determine the workability of an entire project schedule. The downtime due to the weather conditions is known as *weather down time* or *waiting on weather* (WOW). The WOW is primarily affected by the Metocean conditions, which consist of the aforementioned sea states, wind and the long-term forecasting of currents. The sea state is a statistical description of the wave characteristics over a long period. Typically, the sea state will not change for about three hours, and it takes few days to undergo radical changes.

In HMC, the workability is assessed by assigning operability criteria to activities. Typical ways to assign the criteria include using provided sea-state forecasting and vessel motion responses. For heavy-lifting activities, the most commonly used criterion is $H_s T_p^2$. Depending on the weight of load and type of barge, this limit is normally set to 150, which essentially constrains the value of T_p (peak period) to be relatively small; thus, wave peaks come more often. The intuition of this limit has to be combined with the RAO (see Figure 2.4). Assume the wave is regular in a long period and remains steady. Then, the value on the y-axis represents the relative ratio of heave response with respect to the wave amplitude. Assume a ship is a spring-mass system; the responses of low frequencies are dominated by spring coefficients, which are essentially the hydrostatic properties of the vessel. The responses for higher wave frequencies are affected by the added-mass term which is similar to the mass of a vessel. As shown in Figure 2.4, the higher the wave frequency is, the less response the vessel produces. Nevertheless, if the wave frequency is low, the vessel has the risk of falling into the response range dominated by the damping term, which causes the vessel resonance. It is not ideal because it produces huge heave motions. Therefore, higher frequency waves are preferred for the sake of ship motions. The high frequency is equivalent to shorter wave periods. In practice, a SSCV is still able to maintain good motion response against high

amplitude waves as long as the wave period is short. Therefore, the peak period (T_p) has a higher impact on operability criteria, and we normally take the term of $H_s T_p^2$ for a quick assessment.

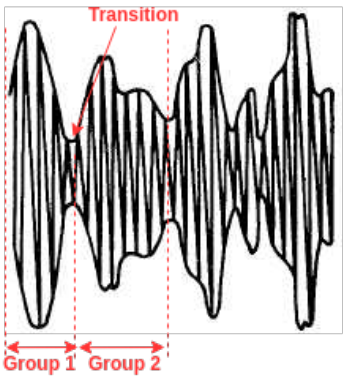


FIGURE 2.7: Wave envelope

FIGURE 2.8: Photo of a real execution. (Left) suspended module on crane. (Right) Module on barge for transport. [Photo: Bo B. Randulff og Jan Arne Wold/ Equinor.

Second, any irregular wave owns the shape of a wave envelope (see Figure 2.7). The motion response to irregular waves can be split into several wave groups with distinct lengths and transitions. Suppose we connect crests and troughs all together by lines. It turns out that the distance between the upper and lower outline is varying on time. This information is particularly useful in set-down operations. If the vessel response is too high, it implies that the vessel is probably in the middle of the current wave group. Then, the better option is to wait for the transition of the next wave group because the response might be lower there. Therefore, the recognition of the wave patterns and knowing exactly when a favorable moment will come make the offshore set-down more challenging. Meanwhile, many actions have to be taken in advance in order to be able to utilize the favorable moment.

2.5 Set-Down Simulation Environment

The target of the set-down is to transfer the heavy object (yellow) onto the *barge* (the platform in green in Figure 2.9). The horizontal positional guide (in blue) is known as the *bumper*, which the crane operator is allowed to bounce against for keeping positioning. However, the maximum allowed contact on the bumper is strictly constrained according to the engineering design of the bumper in HMC. The dimension of the load is 20m (l) x 20m (w) x 40m (h), and the initial distance between the bottom side and the barge deck is about 17m. The dimension of the bumper is 40m (l) x 5m (w) x 5m (h). At the beginning of every episode, a random initial swing amplitude is posed at the hoist wire in y-z plane to ensure that the initial position is always different.

In this project, we have created four different simulation environments with distinct levels of simplification. As a feasibility study carried out completely from scratch, we started the experiments under the simplest possible environment, namely *basic 1D* (see Chapter

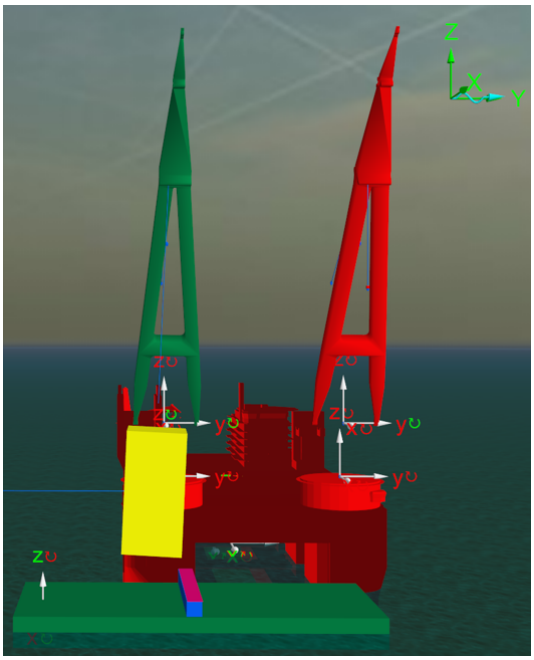


FIGURE 2.9: Visualization of advanced 2D simulation environment

4). In basic 1D, we assume that waves only produced heave motions on a vessel and the ramp-up times of payout and slewing were completely ignored. Moreover, in *advanced 1D* environment (see Chapter 5), we maintained the constraints on vessel motion but took into account the ramp-up and evaluation on re-impacts, which significantly increased the difficulty of the problem. Furthermore, in Chapter 6 and Chapter 7, we continued challenging the agent in more sophisticated 2D environments where lateral motions were enabled. More details about the simplifications and differences of environments are elaborated on from Chapter 4 to Chapter 7. A quick glimpse of the 2D environment is presented in Figure 2.9.

2.6 Evaluation Metrics

The evaluation of a set-down operation consists of multiple factors, the first of which is the set-down precision. In most set-down cases, the load is expected to be placed on a specific location of the barge, which is especially important when the barge deck is relatively full or the load mode to be sea-fastened to certain details. Any unexpected collisions against other objects should be avoided, which explains engineers' concerns about the distance that the load deviates from the targeted position. In practice, positional guides (bumpers) are placed around the targeted position on the barge. The bumper allows the crane operator to bounce against it to make the position better. For the 1d environment, we assumed that the load was always on top of the targeted position, and all objects only had heave motions. In that sense, the set-down precision did not apply. For 2d environments, however, the set-down precision was evaluated by the distance between the lower right-hand corner of the load and the left edge of the bumper.

The second evaluation concerns the impact force. The impact force could occur on both the barge and the bumper whenever there is contact. Most offshore structures are quite sensitive to impacts (i.e., the blades of wind turbines) and the structures must be well preserved before the installation. Therefore, the magnitude of impact force is essentially the most important factor for a set-down.

Under ideal conditions, a set-down can be seen as an elastic response problem. At the moment of contact, we assumed that the barge was perfectly elastic, and it was fully converted to incoming kinetic energy E_k to spring deformation energy E_p with stiffness k:

$$E_k = E_p, \tag{2.15}$$

provided,

$$\begin{aligned} E_k &= \frac{1}{2}mv^2, \\ F &= kx, \\ E_p &= \int_0^{x_{max}} F dx = \int_0^{x_{max}} kx dx = \frac{1}{2}kx_{max}^2. \end{aligned}$$

Since $F_{max} = kx_{max}$, we get the following equations:

$$\begin{aligned} \frac{1}{2}mv^2 &= \frac{1}{2}kx_{max}^2, \\ \frac{1}{2}mv^2 &= \frac{F_{max}^2}{2k}, \\ F &= v\sqrt{mk}. \end{aligned} \tag{2.16}$$

For a specific operation, m and k are constants. According to Equation 2.16, the impact force is linearly dependent on the relative velocity at the moment of contact and sub-linearly on the stiffness of the barge. In 1D environments, we assumed that objects only had relative heave motions, therefore, the impact force can be simplified by the impact velocity, which is essentially the sum of the distances traveled by the load and the barge within a single time step during the contact. Based on the HMC experience, the common limit on the vertical impact velocity for a set-down is 0.4 m/s.

For 2D environments, measuring the contact is slightly more complicated because the object has velocities in two directions with a certain amount of momentum and inertia. We read the measurement directly from the analyzing toolboxes provided by the simulator. We evaluated the max contact force applied to the bumper and the barge separately. In HMC, there are strict engineering limits on the maximum allowed impact force on the bumper, which is 10% of the total mass of the load.

2.7 Practicalities

In reality, there are some pitfalls that might drastically influence the quality of a set-down. These thoughts can be very useful to shape the reward function for training the agent.

One of the most common phenomena during this process is the pendulum swing of the load. Because the load is suspended from the hoist wire with huge weight and inertia, the swing motion occurs easily even with small actions or movements at the crane tip. It is always the first priority to slow down the swing in order to reduce the horizontal impact force. However, it is tricky because in some cases, due to the ship motions caused by waves, the swing still occurs automatically even if the crane operator has not taken any action, which makes it harder to stabilize the load in offshore environments. In addition, the actual position of the vessel is also affected by the actions of the crane operator (e.g., induced roll motion due to the slewing of the crane).

Another issue is re-impact. Re-impacts are the follow-up contacts between the load and the barge when the set-down is completed. The re-impact is caused by the difference in the motions of the load and the barge after the first impact. Suppose the load contacts the barge when the barge deck is just about to descend. If the barge deck descends much faster than the subsequent payout of the load, a gap will appear, which may eventually lead to unexpected impact forces. In practice, the crane operator always switches to the maximum payout speed as soon as the load contacts the barge to ensure that the length of extra payouts in hoist wire can cover the distance that the barge travels after the set-down. However, the pay-out speed is significantly slower than wave induced motion.

However, under 2d environments, the re-impact is even more harmful in the sense of horizontal motions. For heavy loads, a sudden upward motion occurs on the crane vessel as soon as the vessel loses the weight of the load. If the hoist wire is still connected to the load, it suddenly lifts the load and pulls the load away from the target potision in both vertical and horizontal directions. Suppose the load is very close to the bumper with its own pitch motion. A huge impact force occurs when they collide with the highest horizontal relative velocity in the opposite direction. Therefore, the abnormally high impact forces on bumpers are sometimes due to the re-impact in the horizontal direction, which needs to be strictly controlled by the "10%" criterion.

Chapter 3

Reinforcement Learning

The core of this project is about training an agent to complete the set-down. In this chapter, we briefly review the theoretical framework of RL that contributed to the methods used to train the agent.

3.1 General Definition

Reinforcement learning is about mapping states to actions in order to maximize the reward given from the environment by sequential decision making. The problem that RL solves can be formally described as a *Markov decision process (MDP)*. The solution to an MDP is a general rule to select an action that leads to the maximum reward signal of the episode. In particular, the *Markov property* specifies that the environment is fully observable, which means that the future state is independent of past states given the current state. However, it does not hold for most practical applications.

Formally, an MDP is a tuple in the form of $< S, A, P, R, \gamma >$, where S is a finite subset of all possible states in the environment, A is a finite set of all actions that are valid in the environment, P is a state transition probability matrix that provides the probability of entering the next state from a given current state and action, R is a reward function of entering the current state given the action, and γ is a discount factor ranging from 0 to 1. The value 1 simply implies that all rewards should be treated equally, and 0 means that only the next reward is relevant. The common selection is between 0.9 and 1.

The goal of solving the MDP is to find a *policy* $\pi(a|s)$ to choose actions that lead to the maximum expected return. A stochastic policy is a probability density function over all actions given by $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. There are two distinguished classes of algorithms that are used to solve MDPs: the *value-based* and *policy-based* approaches. For the value-based approach, a policy is derived from a *value function* $V_\pi(s)$ of states. The value of a state refers to the expected return starting from the current state s until the end of the episode by following the current policy π. Alternatively, a policy can be derived from an *action-value function* $Q_\pi(s, a)$, which estimates the expected return from state s, taking action a, and following the current policy π. The value and action-value functions can be decomposed into instant reward R_s^a of taking action a at state s plus the value of the next

state $V(s')$ that eventually satisfies the Bellman expectation equation:

$$\begin{aligned} V_\pi(s) &= \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')), \\ Q_\pi(s,a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') Q_\pi(s',a'). \end{aligned} \tag{3.1}$$

Accordingly, given an MDP, one can derive its value-function. This procedure is known as *prediction*. If the transitions of MDP $P_{ss'}^a$ are known, iterative methods such as dynamic programming can be applied. If they are unknown, methods such as Monte Carlo (MC) and temporal difference (TD) learning (Sutton, 1988) are applicable, which are classified as *Model-free* methods. However, the actual solution to an MDP is to find the **optimal** value function $V_*(s)$ or $Q_*(s,a)$ provided an initial policy, which is commonly known as a *control* problem. The solution to the control problem satisfies the *Bellman optimally equation*:

$$\begin{aligned} V_*(s) &= \max_a (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')), \\ Q_*(s,a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} Q_*(s',a'). \end{aligned} \tag{3.2}$$

and specifies the best possible performance in the MDP. The other class of algorithms is policy-based methods. In contrast to value-based methods in which the optimal policy $\pi_*(a|s)$ is found by maximizing over all $Q(s,a)$, policy-based methods directly find the optimal parameterization of the policy $\pi_\theta(s,a) = P[a|s,\theta]$. The action can thus be sampled directly from the prediction of the algorithm.

3.2 Value-Based Methods

Recall that the value function is the expected return of a state $V_\pi(s) = \mathbb{E}[R_t|S_t = s]$. The return is defined as the total discounted reward from time t until the end of the episode:

$$R_t = r_t + \gamma r_t + \gamma^2 r_{t+2} + ... + \gamma^{T-1} r_{t+T-1}, \tag{3.3}$$

and the new value of $V(s)$ is updated toward the difference in true outcome versus the estimation. There are two critical ways of describing the error term, which are *Monte Carlo (MC)* and *temporal difference (TD)*. The former uses the empirical mean return instead of expected return G_t. Formally, in order to update the value of state s, the error is calculated by the true return of the episode minus the old value of state s and the error is scaled by dividing by the number of visitations $N(s)$ of state s:

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(R_t - V(s_t)), \tag{3.4}$$

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)). \tag{3.5}$$

which requires the algorithm to complete the entire episode. On the other hand, TD computes the error by introducing a target value (TD target) (see equation 3.5), which is the discounted expected return from s' plus the reward of entering s' from s, and α is the learning rate. The TD error is simply a difference between a more realistic estimation (TD target) and the current estimation ($V(s)$).

There are mainly three significant differences between MC and TD updates. First, the MC method must wait until the end of an episode in order to know the true return, whereas TD can perform an update for every time step as long as a value function exists. Hence, MC is only applicable for episodic environments. The second difference concerns the bias and variance trade-off. The MC method introduces huge variance because the error for the update is composed by the true value, which is computed from a long horizon. It might be possible that one episode has a much stronger influence than the others such that the algorithm might just update toward it and get closer to optimum. However, TD methods are biased because the TD error measures the difference between two estimated terms. Therefore, the TD error is less accurate, especially when the value function is just initialized. Finally, TD methods are more useful for states with Markov properties because the TD error merely compares with two successive states. Conversely, due to using the future trajectories to compute the return, MC methods are more effective in non-Markovian environments. One way to facilitate the advantages of both methods involves using λ *returns (TD(λ))*. The intuition is that, for evaluating the value of state s_t, instead of only bootstrapping one step (TD target) or waiting for the true return (MC), we accumulate n step immediate rewards from R_t to R_{t+n} plus the estimated return onwards $V(s_{t+n+1})$:

$$R_t^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}), \tag{3.6}$$

$$\begin{aligned} R_t^{\lambda} &= (1-\lambda) \sum_{n=1} \lambda^{(n-1)} R_t^{(n)}, \\ V(s_t) &\leftarrow V(s_t) + \alpha(R_t^{\lambda} - V(s_t)). \end{aligned} \tag{3.7}$$

Then, we take a weighted average in terms of λ over a number of different *"n-step"* returns, which eventually leads to a TD(λ) error (see equation 3.7). The factor λ determines how fast the importance of long-term return is decayed. When λ is 1, it becomes standard Monte Carlo and pure TD when λ is 0.

For most problems, we want to achieve an optimal policy based on an initialization. In the sense of value-based methods, the optimal policy can be derived from the optimal value function $V_*(s)$. Suppose we have obtained a value function $V_\pi(s)$ based on policy π, and the MDP (model) is unknown, then the only way to derive the improved policy is to select actions greedy over the action-value function $\pi(s) = \text{argmax}_{a \in A} Q(s,a)$. This policy iteration is known as *model-free control*. When the policy is far from optimal, especially for the beginning, it is important to encourage exploration over all possible actions. An overall of common exploration methods was given in (Wiering, 1999). For the value function, one of the common ways is $\epsilon - greedy$ exploration. It chooses an action at random with probability

ϵ and chooses an action with the highest action value $a = \text{argmax}_a Q(s, a)$ with probability $1 - \epsilon$. According to the policy improvement theorem (Jaakkola et al., 1995), if the $\epsilon - greedy$ policy π is improved, then the equivalent value function V_π is also improved.

For control problems, it is more straightforward to use TD methods than MC methods. This is simply because the TD target can be directly plugged into Bellman equations for policy evaluation, and the update can be performed in every time step, which is much more promising in terms of sample efficiency. Depending on the rules of updating Q-values (action-value of a state $Q(s, a)$) for policy evaluation, there are mainly two different classes of methods. Recalling the TD target of the value function in equation 3.5, which consists of an immediate reward r and an estimation of expected return onwards, then we convert it in terms of state-action values, which immediately raises two options. For estimating the return from state s', if we stick to the state-action value indicated by the current policy $Q(s', a')$, we will get an *on-policy* evaluation rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)), \tag{3.8}$$

which satisfies the Bellman expectation equation (equation 3.1), which is also known as *state-action-reward-state-action (SARSA)* (Sutton, 1996). Another on-policy method is *QV-learning* (Wiering, 2005). In QV-learning, a separate value function V is learned in addition to the action-value function Q, and in contrast to SARSA, the action-value function Q used for computing the TD target is replaced by the independent value function V, which results in better estimations of state-value function by using more experiences (Wiering and van Hasselt, 2009). On the contrary, suppose we completely ignore the current policy and choose the maximal Q value of state s' for calculating the TD target; then, the policy evaluation becomes *off-policy*:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)), \tag{3.9}$$

satisfying the Bellman optimality equation (equation 3.2). The updating rule of the action-value function in equation 3.9 is known as *Q-learning* (Watkins and Dayan, 1992). The convergence to the optimal action-value function of SARSA and Q-learning are theoretically guaranteed for tabular cases (Sutton and Barto, 1998).

In practice, considering the dimensionality of the state space, the value functions are normally represented by differentiable function approximations instead of a look-up table. The value function $V(s)$ is parameterized by a set of weights θ such that $V_\pi(s) \approx V_\theta(s)$. One of the most popular choices of the function approximation is a neural network. The weights are updated toward the gradient of error in the approximated value function. In practice, the true value for computing the error in the objective function is substituted by a target return in a TD or MC fashion. Due to the non-linearities of the activation functions of neural networks, the convergence of TD approaches is not guaranteed (Sutton and Barto, 1998). It is widely accepted in Supervised Learning that the gradient update over a mini-batch

is much more efficient than having an on-line update. Therefore, for off-policy methods, an experience buffer is created to store the experience transitions (s, a, r, s') that the agent has experienced. Then, a small subset of all experiences is sampled to compute the gradient. This procedure is known as *experience replay* (Lin, 1992), which largely decouples correlations in the sequence of observations.

3.3 Policy-Based Methods

Another branch of RL algorithms is policy-based methods. Contrary to value-based methods, which obtain the policy from value functions, policy-based methods do not use value functions but directly return the policy $\pi(s, a)$ with a parameterization of input states $\pi(s, a) = \mathbb{P}(a|s, \theta)$. One of the advantages of policy-based methods is their effectiveness in high-dimensional and continuous action spaces. Recall that value-based control algorithms exclusively compute the state-action value for all possible actions given a state. If the action space is continuous, then the number of actions is infinite, which makes it less efficient to learn the optimal policy. This typically happens in robotic control where the policy is the rotation angle of a joint. Furthermore, the policy gradient method is guaranteed to converge to a local optimum given a differentiable function approximation (Sutton, McAllester, et al., 2000). However, value-based functions have no convergence properties using non-linear function approximations and sometimes might even diverge.

We are looking for an optimal policy that maximizes the expected return of the episode. The objective function is thus simply $J(\theta) = V^{\pi_\theta}(s_0)$, where s_0 is the initial state of an episode. Then, a single update can be achieved by ascending the gradient of J with respect to θ. According to the policy gradient theorem (Sutton, McAllester, et al., 2000), the gradient of any policy object function can be represented by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)], \tag{3.10}$$

which is the expectation of the product of the gradient of the policy value and the long-term return given state and action $Q^{\pi_\theta}(s, a)$. Therefore, suppose we update the policy by only one episode and use an unbiased Monte Carlo return for $Q^{\pi_\theta}(s, a)$, then the update rule for the policy function is given in equation 3.11, which formulates the REINFORCE algorithm (Williams, 1992) with MC return:

$$\theta \leftarrow \theta + \alpha \nabla_\pi log \pi_\theta(s_t, a_t) R_t. \tag{3.11}$$

The vanilla REINFORCE suffers from the same issue as MC control methods, which is the high variance due to the unbiased return. In light of the policy gradient theorem in equation 3.10, a natural idea is to replace the long-term expected reward of the action, $Q^{\pi_\theta}(s, a)$, with a function approximation with a parameterization of w, that is, $Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$, which leads to the class of *actor-critic* algorithms (Konda and Tsitsiklis, 2000). The

actor is parameterized by θ for generating the policy given the state, and for the critic, an action-value function approximation is used to estimate the long-term return of the current state and action. The actor is updated using the estimation from the critic while the critic is updated using the TD error (equation 3.5). The actor-critic system approximates the policy gradient in the following equation:

$$\begin{aligned} \delta &= r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a), \\ \theta &= \theta + \alpha \nabla_\theta log \pi_\theta(s, a) Q_w(s, a), \\ w &\leftarrow w + \alpha \delta \nabla_w Q(s, a). \end{aligned} \tag{3.12}$$

This introduces bias due to the nature of value function approximation. According to the compatible function approximation theorem (Sutton, McAllester, et al., 2000), if the error of a critic is small enough in estimating the expected long-reward return, then the convergence property of the method will follow the *exact policy gradient*. Therefore, the variance of the standard actor-critic method can be further degraded by actually reducing the bias in approximating the policy gradient, that is, the bias in the critic. Such reductions fall into the definition of using an *advantage function*. The advantage function intuitively quantifies the extra gain of reward of taking action a at state s over the average such that the benefit of a particular action over others are clarified. This is done by subtracting the *baseline* $B(s)$ from the critic. A common selection of baseline is the value-function of state s with another set of weights, e.g., $V_v(s)$ (see equation 3.13). Given the chain rule of $\nabla_\theta log f(\theta)|_{\theta_{old}}$, the policy update can be represented in terms of *importance sampling* (equation 3.14), which is the ratio $r_t = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ in proportion to differences between the output of new and old policies given the same state s_t:

$$A^{\pi_\theta}(s, a) = Q_w(s, a) - V_v(s), \tag{3.13}$$

$$\begin{aligned} \nabla_\theta J &= \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) A^{\pi_\theta}(s, a)], \\ \nabla_{\theta_{old}} J &= \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A^{\pi_{\theta_{old}}}(s_t, a_t)]. \end{aligned} \tag{3.14}$$

As stated in the section above, a more appealing way to estimate the return in the advantage function is TD(λ), which estimates the return of a state-action value by the weighted average over multiple roll-outs. The results in (Mnih, Badia, et al., 2016) show that combining TD(λ) and the value function in approximating the policy gradient leads to much better convergence in practice.

Besides using advantages functions, van Hasselt and Wiering (2007) introduced the *Continuous Actor Learning Automaton (CACLA)* that uses the sign of the temporal difference error instead of its magnitude. Cacla can be implemented directly in the continuous action space, especially for the problem where the distance to the optimal action is more informative than the estimation of the value improvement. CACLA is thus invariant to the scaling of reward functions, and the results in (van Hasselt and Wiering, 2007) show that CACLA outperforms

other policy-based methods in continuous action spaces.

Above all, policy-based methods are, in general, limited by sample efficiency and step size. Due to the nature of on-policy methods, it is only possible to perform one gradient update for each sampled trajectory; that is, the samples are discarded as soon as the gradient is computed. Additionally, if training episodes are generated in a random manner but the algorithm uses a fixed step size, then the algorithm might converge to a local minimum caused by the updates on bad trajectories and may not be able to converge.

3.4 Recent Advances in RL

3.4.1 Value Function

The recent advances of RL comprise deep neural networks (DNN) and RL algorithms. The former harnesses a deep network topology to extract high-level features and generates relevant abstractions of the input state. This allows RL algorithms to solve very complex tasks. Simultaneously, advances in RL algorithms also improve the performance of many baseline algorithms.

One of the first algorithms to couple DNN and value-based methods is the deep Q-network (DQN) (Mnih, Kavukcuoglu, et al., 2015). The core of DQN is model-free, off-policy Q-learning using the value function approximation; DQN was initially applied to play Atari games by taking graphical inputs as human do. The inputs are first processed by convolution layers, which are widely used in image classification DNNs such as CNNs (LeCun et al., 1998) to extract high-level features from the input. Note that, instead of separately computing a Q value for every action, the FA of DQN computes all Q values at once. Like other value-function approximation methods, the objective of DQN (see equation 3.15) is to minimize squared temporal difference (TD) errors. Due to the non-linearity of the function approximator, the convergence of the control problem using Q-learning is unstable and not guaranteed. The policy is pretty sensitive to small updates of the value function, and the correlations between observations might increase the variance of the update. Therefore, DQN degrades the correlations between inputs by using an experience replay buffer D initially proposed by Lin (1992). The gradients of a batch are calculated by the TD error of randomly sampled transitions (s, a, s', r):

$$\mathbb{L}(\theta) = \mathbb{E}_{s,a,s',r\ D}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]. \tag{3.15}$$

Additionally, DQN uses two value functions for calculating targets and estimations separately, and they are updated asynchronously. The benefit of introducing a separate target network is to stabilize correlations between target and value estimations. Intuitively, the value network is a student to imitate the target network. It will be easier for a student θ to learn if the distribution of target network θ^- remains fixed for a while than if it changes all the time. For every input, DQN stacks multiple frames so as to extract high-order features

such as motions due to the partial observability in the nature of image input. There are a couple of enhancements based on vanilla DQN. Since standard Q-learning takes maximum operators for updating the function approximation (see equation 3.9), it involves an optimistic bias in calculating the error if the true value is rather small. Therefore, an incorrect action might get learned too fast in the beginning, which is known as maximization bias (van Hasselt, 2010). A natural idea is to replace the max operator in the target value by a less biased estimation. Thanks to the two-network framework of DQN, double DQN (van Hasselt, Guez, et al., 2016) removes the bias by calculating the TD target (Q_2) using the action indicated by the value network (Q_1):

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]. \quad (3.16)$$

Moreover, for every update, batches are sampled uniformly from the buffer. For problems with rewards that are delayed or given seldomly, the importance of each sample is largely in difference. One way is to correct the sampling probability in proportion to the magnitude of TD error (equation 3.5). Therefore, effective samples are seen more often. In order to avoid overly large step sizes, Schaul et al. (2015) attempted to correct the step size by importance sampling. Furthermore, since actions do not always have huge impacts on rewards, it is also particularly common that the selection of actions does not make a huge difference since all the actions lead to a similar future. Based on this intuition, Wang et al. (2015) proposed dueling DQN, which estimates the Q value for every action by aggregating separate estimations of the state value and the advantage value. The advantage function essentially measures how much more important an action is than others. By providing this dueling aggregating module, Wang et al. (2015) achieved a substantial enhancement over vanilla DQN.

3.4.2 Policy Gradient

With regard to policy-based methods, the standard policy gradient suffers from two issues: sample efficiency and step size. For the vanilla policy gradient, trajectories are generated following the on-policy manner, which only allows the trajectory to be used once and discarded right after. On the other hand, the standard policy gradient, in general, is high in variance because the gradient is computed based on correlated trajectories. If the gradient update is derived from poor samples, the policy will probably be shifted to a local minimum with a large step size. However, too small a step size will slow down the optimization procedure.

Schulman, Levine, et al. (2015) successfully addressed the issue of step size by introducing a *trust region*. Intuitively, we assume that any policy update within the trust region does not dramatically change the policy distribution. The goal of policy optimization is to keep improving the expected long-term return $R(\pi)$ of policy π. Given the definition of *advantage* in equation 3.13, the improvement of the new policy $R(\tilde{\pi}) - R(\pi)$ is essentially

the expected advantage value of taking actions sampled from the new policy $a_t \sim \tilde{\pi}(a_t|s_t)$ on the trajectory of old policy π:

$$R(\tilde{\pi}) = R(\pi) + \mathbb{E}_t[\sum_{t=0} \gamma^t A_\pi(s_t, a_t)]. \tag{3.17}$$

Assuming the state visitations $\rho_\pi(s)$ of the old π and new policies $\tilde{\pi}$ are close, a local approximation to the expected return (given in equation 3.17) of the new policy $L_\pi(\tilde{\pi})$ can be formed by:

$$L_\pi(\tilde{\pi}) = R(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi, \tag{3.18}$$

$$R(\tilde{\pi}) \leq L_\pi(\tilde{\pi}) + C D_{KL}^{max}(\pi, \tilde{\pi}). \tag{3.19}$$

Kakade and Langford (2002) proved that the expected return of a policy $R(\tilde{\pi})$ is guaranteed to increase monotonically as long as a policy update improves the right-hand side of equation 3.19, and the improvement is bounded by a distance measure (KL divergence D_{KL}) of the policy update. Therefore, in order to improve the lower bound, the *"surrogate"* loss objective function for *trust region policy optimization (TRPO)* is proposed in equation 3.20. The objective is to maximize the expected advantage of the new policy while minimizing the KL penalty of two consecutive policy updates. In that sense, a trust region constrains the largest allowed change of one policy update:

$$\underset{\theta}{\text{maximize}} \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A^{\pi_\theta}(s_t, a_t) - \beta D_{KL}[\pi_{\theta_{old}}(s_t), \pi_\theta(s_t)]]. \tag{3.20}$$

A novel extension called *proximal policy optimization (PPO)* proposed by Schulman, Wolski, et al. (2017) significantly reduces the computation on the conjugate gradient in equation 3.20. Moreover, PPO replaces the constraint on KL divergence in the surrogate loss of TRPO by a clipping function given by: 3.21.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(i_t(\theta)\hat{A}_t, clip(i_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]. \tag{3.21}$$

It clips the policy update (product of importance sampling $i_t = (\pi_\theta(a_t|s_t))/(\pi_{\theta_{old}}(a_t|s_t))$ and advantage) by a hyperparameter ϵ within the range $1-\epsilon$ to $1+\epsilon$. The clipping allows the change of the policy within a range and essentially restricts that the new policy does not change too much from the old policy, which satisfies the original idea of a *trust region*. Accordingly, multiple steps of stochastic gradient ascent are actually possible on this surrogate objective, which improves the sample inefficiency issue of standard policy gradient methods.

One of the key techniques in the variance reduction of policy gradient methods is the use of the advantage function. One popular choice is the *finite-horizon estimator* initially introduced in (Mnih, Badia, et al., 2016), where the state-action value in the advantage function

is estimated by the sum of the discounted reward in a finite horizon $l = T - t$ from time t to terminal T (see equation 3.22). However, the variance is still high when the horizon is very long.

$$\hat{A}_t^{(l)} = r_t + \gamma r_{t+1} + ... + \gamma^{l-1} r_{t+l-1} + \gamma^l V(s_{t+l}) - V(s_t). \tag{3.22}$$

Motivated by the forward view of the TD(λ) backup, Schulman, Moritz, et al. (2015) introduced the *generalized advantage estimator*. The state-action value is instead estimated by the weighted summation over multiple finite-horizon estimations with various lengths:

$$\begin{aligned} \delta_t^V &= r_t + \gamma V(s_{t+1}) - V(s_t), \\ \hat{A}_t^{(l)} &= \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + ... + (\gamma\lambda)^{l-1}\delta_{t+l-1} \end{aligned} \tag{3.23}$$

$$\begin{aligned} \hat{A}_t &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ...) \\ \hat{A}_t &= \sum_{l=0} (\lambda\gamma)^l \delta_{t+l}^V. \end{aligned} \tag{3.24}$$

Each finite-horizon estimation with length l is discounted by λ^l, which results in equation 3.24.

3.4.3 Hierarchical Reinforcement Learning

For many goal-driven problems, the reward for feedback about agent behavior can be very sparse, requiring the agent to have levels of abstraction of the environment and use hierarchical policies. Sutton, Precup, et al. (1999) introduced the concept of *option* for the generalization of primitive actions in an MDP setting. The option consists of a policy with the validity from its initial state until the terminal state. As soon as an option is activated, all actions are selected according to the policy of the option until termination. Any MDP that includes the selection of options is called a *semi-Markov decision process* (SMDP), which allows each action to take various amounts of time.

Many researchers have worked on such hierarchical policies for complex problems. Wiering and Schmidhuber (1997) proposed *HQ-learning* that learns ordered sequences of sub-policies for each of the sub-goals. The sub-goal is a specific state that is chosen according its corresponding HQ-table. Every HQ-table is updated toward the expectation of the future reward of the subsequent sub-goal. Contrary to HQ-learning, Dayan and Hinton (1993) initially introduced the idea of feudal control that decomposes the hierarchy into "super-manager and sub-managers" at levels with different explicit goals. Managers reward and select sub-tasks for sub-managers all the way down to primitive actions. Each sub-task refers to its own MDP. The manager recursively satisfies its super-manager until the goal of the highest-level manager is satisfied. Dietterich (1998) extended feudal control by introducing the MAXQ method that represents the optimal value function of an MDP by the value functions of the MDP at lower levels and ensures the Markov property for sub-tasks at each level of the hierarchy. The FeUdal network (FuN) (Vezhnevets et al., 2017) and hierarchical DQN (h-DQN) (Kulkarni et al., 2016) incorporate the hierarchical RL with deep

networks. A meta-controller computes sub-goals based on a recursively optimal policy, and corresponding controllers try to achieve maximum intrinsic rewards from the sub-goal. As a variation of DQN, h-DQN uses action-value functions for both the meta-controller and the controller while FuN adopts recurrent networks for all controllers and derives policy-by-policy gradient methods, as in the option-critic method (Bacon et al., 2017). They both represent substantial improvements on *Montezuma's Revenge*, one of the most challenging games on ATARI, where rewards are given rarely in particular states. Alternatively, the H-DRLN given in (Tessler et al., 2017) demonstrates a better policy can be achieved if the controller is allowed to select from all primitive actions as well as from pre-trained skills that are good at solving the sub-tasks of a complex problem.

3.4.4 RL with Monte Carlo Tree Search

Monte Carlo tree search (MCTS) methods have already significantly aided in solving complex problems that require sequential decisions and planning. Numbers of MCTS variants have shown great success in playing many combinatorial games, such as Go, Hex, as well as some other non-game applications like combinatorial optimization and scheduling tasks (Browne et al., 2012).

In contrast to traditional exhaustive tree search algorithms, such as Minimax, MCTS builds a search tree within a certain computational limit. Every node of the tree represents an action and is associated with a value that represents the expected total reward of taking this action unitl it is terminal. Although action values are estimated by random simulations, MCTS enables the estimations to be achieved efficiently with respect to the computation limit. In principle, one might perform a complete MCTS for the selection of every action. An MCTS consists of four stages. 1) **Selection**: Starting from the root node s, a tree policy π_{tree} is applied recursively to picking child nodes (s, a) until it reaches an expandable node that has unexplored children. 2) **Expansion**: Add a few nodes at the expandable node according to the tree policy. 3) **Simulation**: Run several simulations starting from nodes that are unexplored or expanded in the **Expansion** stage until the terminal state. Note that the policy used in the simulation $\pi_{rollout}$ is different than **selection**. 4) **Backpropagation**: Update the state-action value of a node $Q(s, a)$ by averaging the rewards of all simulations that include this node. In other words, the value of a node reflects the average rewards of taking the action over several simulations. The most popular choice of the tree policy is the *upper confidence bound for trees* (UCT) (Kocsis and Szepesvári, 2006). Instead of always selecting child nodes by maximum state-action value $Q(s, a)$, UCT encourages exploration with concern to the number of visitations:

$$\pi_{UCT}(s) = \underset{a_i}{\operatorname{argmax}}\, Q(s, a_i) + c\sqrt{\frac{2 \ln n(s)}{n(s, a_i)}}, \tag{3.25}$$

where $n(s)$ is the number of visitation of node s, and $n(s, a_i)$ is the number of times that child node (s, a_i) is selected. Equation 3.25 reduces the chance of a child node with good

value being selected too often while keeping the probability of being selected for nodes that are less explored.

The recent advances in RL have achieved remarkable improvements by incorporating MCTS. Silver, Sutton, et al. (2008) introduce the Dyna-2 framework, which selects actions for TD learning by MCTS with the minimum of initial knowledge of the model. For every state s, MCTS is implemented given the current transition model, and the value function $Q(s, a)$ is updated toward the TD error derived from the action a selected by MCTS. The transition model is then updated according to the observed subsequent state s' such that planning and learning happen concurrently. Instead of updating the value function, the AlphaGo Zero algorithm proposed in (Silver, Schrittwieser, et al., 2017) shows that it is of great effectiveness to just maximize the similarity between policy and MCTS search results in a way analogous to supervised learning. Similar to its predecessor AlphaGo in (Silver, Huang, et al., 2016), the AlphaGo Zero improves the efficiency of MCTS with regard to breadth and depth by the policy and value networks. The policy network indicates the prior for the reduction of child nodes for selection, and the value function predicts the expected return such that it truncates the depth of the simulation. AlphaGo has had great success in translating human knowledge into a policy network that can be used for MCTS, and AlphaGo Zero directly takes the output of MCTS for training, which results in a much simpler and better performance. AlphaZero, as proposed in (Silver, Hubert, et al., 2017), further generalizes the algorithm of AlphaGo Zero for some other board games.

3.4.5 Imitation Learning

The goal of imitation learning is to train a policy that tries to mimic the behavior of demonstrators/experts given the distribution of states in the testing environment. The simplest approach is reducing the imitation learning to a supervised learning problem (i.e., behavior cloning) that minimizes the difference between model predictions and an expert's action along the trajectories generated by the expert. This is impractical because it fails to generalize to the states that are not presented in the training set, and the policy is unable to recover from making a mistake. Ross and Bagnell (2010) argued that, in the worst case, the performance of naive behavior cloning degrades superlinearly with respect to the horizon of the trajectory. The *forward training algorithm* proposed in (Ross and Bagnell, 2010) manages to reduce the growth of the performance loss to linear complexity. It maintains individual policies exclusively for every time step on a trajectory. For every time step, the algorithm queries for the action made by the expert and updates its learning policy, which minimizes the difference between actions. Therefore, the dataset actually includes the corrections made by the expert for states encountered by the training policies. However, it still fails to be applicable for problems with longer horizons.

As a more robust approach, the algorithm *dataset aggregation (DAgger)* proposed in (Ross, Gordon, et al., 2011) demonstrates that a better performance guarantee can be achieved in

a stationary policy. Unlike the forward training algorithm, DAgger only maintains a single deterministic policy that trains on a dataset that is iteratively aggregated by experiences generated by the policy mixed with the learning and expert policy. Therefore, the dataset contains states induced by the learning policy with the feedback by the expert and states resulting from the expert policy. Alternatively, as a variant of DQN, *deep Q-learning from demonstration (DQFD)* proposed by Hester et al. (2017) shows the effectiveness of directly adding a supervised loss to the objective function of DQN. In addition to the agent experience buffer, experiences of the expert are stored in a separated expert buffer and remain unchanged. For each batch update, experiences from the agent and the expert are sampled simultaneously. The agent is asked to give actions on states sampled from the expert buffer and this results in a supervised loss with respect to the expert's action. Adding such a term leads to a faster learning procedure as well as higher average accumulated rewards for some control problems.

3.5 Transfer Learning

In reality, different problems might require similar skills for making decisions, and it is a big waste that the agent has to learn every similar problem from scratch. Therefore, the motivation of transfer learning (TL) is to transfer the knowledge of a source task to a related target task for a better training process. Taylor and Stone (2009) argued that the evaluation metrics of a TL method consist of jump-start rewards, asymptotic rewards, total accumulated rewards, and the transfer ratio (i.e., the ratio of total accumulated rewards gained by using TL and not using TL).

One of the common classes of TL is transferring between tasks sharing the same state and action variables. In that sense, the allowed differences between tasks can be initial/terminal states, reward functions, and the transition model of the MDP. Selfridge et al. (1985) discovered that knowledge can be transferred to an MDP with slightly different transition dynamics. They found that the agent balances a heavy pole faster if it starts off learning on a lighter pole and then switches to a heavier one once the easier problem is solved. Similarly, Asada et al. (1994) demonstrated that the knowledge of reaching the goal state from an easy initial state can be transferred to solving a problem that starts from a much harder initial state, which is also called starting-point method in (Torrey and Shavlik, 2009). Moreover, Atkeson and Santamaria (1997) transferred the source task to a target task by only changing the reward function, which resulted in a significant improvement on jump-start and asymptotic rewards. With regard to batch RL methods, Lazaric (2008) showed that it is useful to manually add the experiences from source tasks that are relevant to solving target tasks to the replay batch for updating the target-task policy. In that sense, useful experiences are available earlier than with random exploration.

3.6 Partial Observability

Suppose the state of an existing MDP satisfies the Markov property such that:

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, ...s_t]. \tag{3.26}$$

In practice, the true states are not always available due to perception devices; therefore the agent only receives an observation o_t determined by a function of the current state $Z^a_{so} = \mathbb{P}[o|s, a]$ such that $o_t \sim Z^a_{s_t o_t}$. Then, the next state s_{t+1} is no longer fully dependent on the agent's current observation such that the agent might be unable to provide optimal actions based on this partial observability:

$$\mathbb{P}[o_{t+1}|o_t] \neq \mathbb{P}[o_{t+1}|o_1, ...o_t]. \tag{3.27}$$

Formally, this is known as a *POMDP* formulated by the tuple $< S, A, O, P, R, Z, \gamma >$, where S, A, P, R, γ is the same as in an MDP, O is a set of observations, and Z is an observation function.

Since the agent does not have access to the true state S_t of the environment, in the POMDP setting, the agent uses a belief state $b(s_t|H_t)$ as a probabilistic estimation of the current state S_t given the *history* H_t, that is, all the observation-action pairs $h_t = a_1, o_1, ...a_{t-1}, o_{t-1}$ from the past. Note that the MDP can be seen as a special case of POMDP, assuming that the observation is identical to the state and that the observation function always returns true states to the agent. The optimal value function of a POMDP is then obtained by a policy $\pi(h_t, a)$ that maximizes the expected return onwards given the history h_t:

$$V^*(h_t) = \max_\pi \mathbb{E}[R_t|h_t]. \tag{3.28}$$

A common choice is to facilitate the internal memory state of *recurrent neural network* (RNN) (Hochreiter and Schmidhuber, 1997) to maintain the belief state for the estimation of the true state.

Chapter 4

Basic 1D

4.1 Basic 1D Environment

Basic 1D produces the simplest abstraction of a set-down operation. In this environment, the load and the barge were initially separated by 7 meters. The load (block) and the barge are represented by two points only with relative heave motions in the time domain (see Figure 4.1), and their mass and stiffness are completed ignored and set to 1 (refer to equation 2.16). In addition, we assume the heave response of the barge is 1 to all frequencies such that the barge position is always identical to the wave amplitude generated by a specific sea state, and the load is not affected by waves.

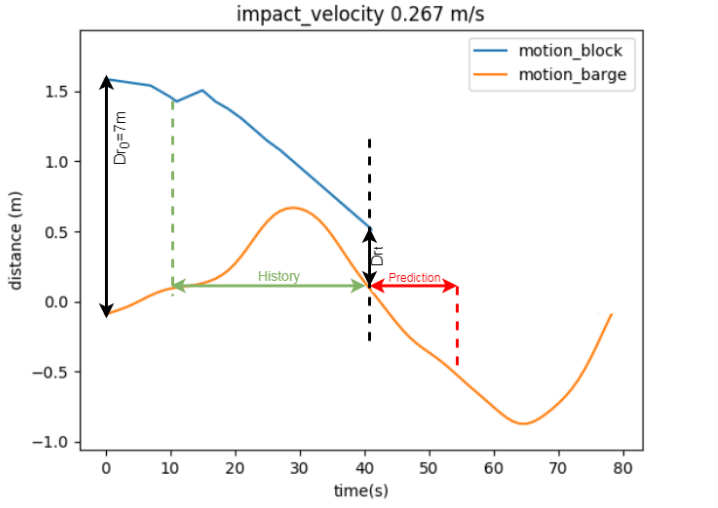


FIGURE 4.1: Basic 1D set-down environment.

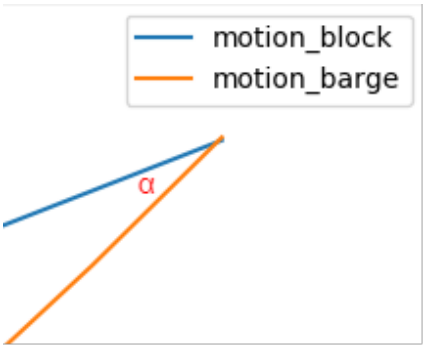


FIGURE 4.2: Episode ends when distance of block and barge is negative. Key is to reduce contact angle α before they collide.

In every time step, the environment calculates a relative distance D_r between the load and the barge by subtracting the adjusted length of the hoist wire and the barge position from the initial distance. Then, the agent is allowed to select a payout speed to change the hoist wire length for the next time step. Note that the ramp-up time is not considered in this environment, so the hoist wire length is adjusted immediately according to the agent's action. This procedure repeats until the relative distance is reduced to be negative (See Figure 4.2), in which case we assume a set-down episode terminates.

The goal is to ensure that the change of the relative distance D_r (thus the velocity) is small in the time step where D_r is changed from positive to negative. In other words, the agent needs to reduce the angle between the blue and orange curves before they collide (see Figure 4.2). If the barge is static, it is extremely easy to achieve by always lowering the load at the smallest payout speed. However, it becomes much more tricky when the barge is moving.

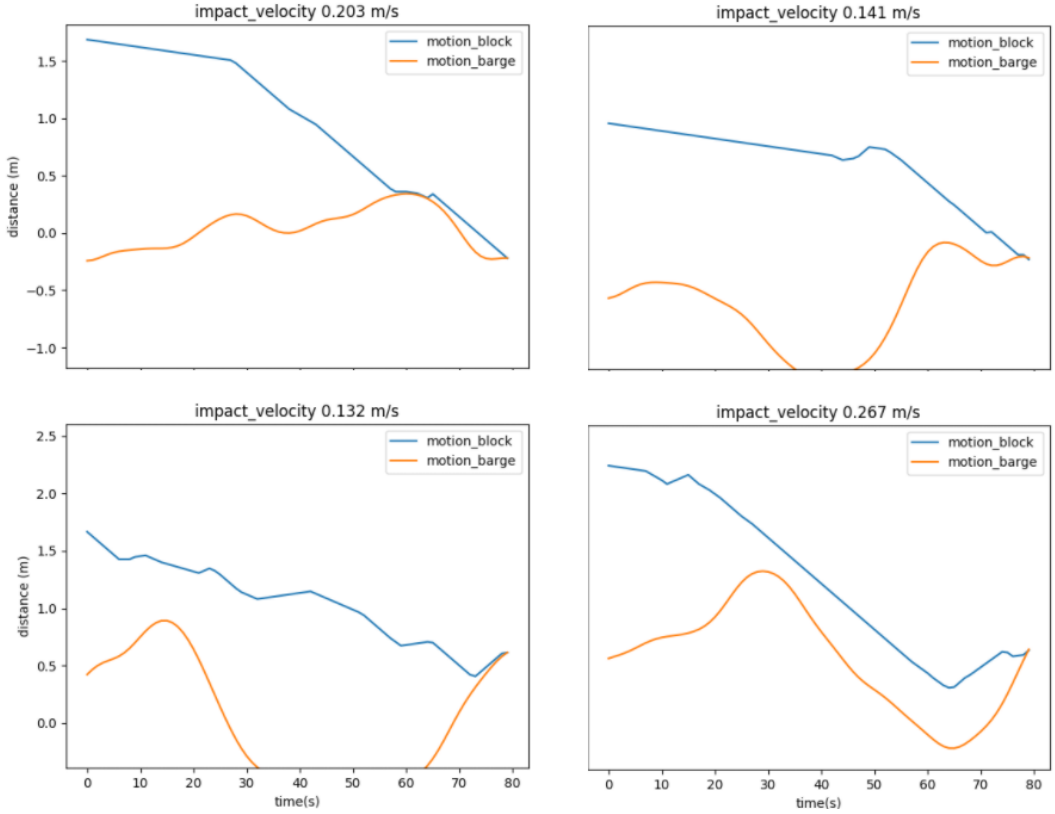


FIGURE 4.3: Examples of good set-down episodes.

Figure 4.3 displays a few examples of possible good set-down attempts with regard to the basic 1D environment. It indicates that a good set-down is quite dependent on the shape of the barge motion and the way the agent to prepare itself to be involved in a position that potentially leads to a good set-down.

The settings of the environment are summarized in Table 4.1. Note that the wave of every episode is randomly generated from the JONSWAP spectra created by the same sea state. Therefore, the wave elevation differs between episodes but not the statistical property. Note that, in basic 1D, we assumed the action space was discrete, in order to meet the range of the actual pay-out speed of the auxiliary block (-5/30 to 5/30 m/s), the numerator of the speed can only be an integer between -5 and 5, and the denominator is always 30.

In this environment, the goal is to reduce the impact velocity v_i. Based on the limit according to HMC experience, we designed the reward function in terms of v_i in equation 4.1. We consider any episode with $v_i > 0.3$ unacceptable, and such episodes receive a penalty.

Setting	Value
Initial D_r	7m
Time step	0.2s
Episode length	300s
Hoist speed range	from -5/30 to 5/30 m/s
Size of action space	11
Input space	11 (2s future motion+current D_r) or 151 (30s history+current D_r)
Sea state	Hs:1.5m, Tp: 15s

TABLE 4.1: Environment settings of basic 1D.

Setting	Value
Learning rate	0.0001
Gamma	0.99
Exploration ϵ	0.5-0.1
ϵ decay	to 0.1 in 50000 steps
Batch size	32
Target update frequency	10000
Replay buffer	50000
Update frequency	1
Num hidden layers/units	1/100
Activation	Sigmoid
Optimizer	Stochastic Gradient Descent (SGD)

TABLE 4.2: Hyperparameters for training end-to-end.

The v_i was simply derived from the sum of the last positive D_r and the absolute of the first negative D_r, and then we divided the sum by time step 0.2, which is equivalent to the angle between the blue and orange curves in Figure 4.2.

$$r = \begin{cases} \max(200, \frac{10}{v_i}), & \text{if } 0 < v_i < 0.3, \\ -30, & \text{if } v_i > 0.3, \\ -0.01, & \text{else}, \end{cases} \tag{4.1}$$

where v_i impact velocity.

4.2 Training for End-to-End

The goal of the first experiment was to learn to set down directly from 7m. Considering the hoist speed and time step, it actually requires a long sequence of actions in order to achieve the final reward. In that sense, the real useful reward is very much delayed, and the reward assignment is very sparse. Hence, we named it *end-to-end set-down*. We prepared two distinct state variable inputs (see Figure 4.1). Both contained the current relative distance D_{r_t}, but

with one knowing the barge heave history for the most recent 30s (2 wave periods) and for the other one, the agent was fed by the true barge motion of 2 seconds in the future (1/7 of single wave period). For both inputs, the time step is 0.2s. Note that, since we pre-generated the wave elevation array for every episode, the "true" wave prediction could be obtained by simply extracting subsequent elements in the wave time trace from the current time step.

We used vanilla double Q-learning with experience replay and target network, as proposed in (van Hasselt, Guez, et al., 2016), with the hyperparameter provided in Table 4.2 for training. The pseudocode of double q-learning can be found in Appendix A.1. We tested two agents on 500 random episodes with no exploration. In addition, for benchmarks, we also created a "monkey" that always selects the maximum payout speed regardless of states for the benchmark.

The results are presented in Figure 4.4. The figure shows the probability density of impact velocity over 500 testing episodes. Apparently, both agents managed to obtain a better policy than just constantly fully speeding down. The peak of the histogram is shifted to the left of 0.3, indicating that the majority of the testing episodes are quite acceptable.

4.3 Results

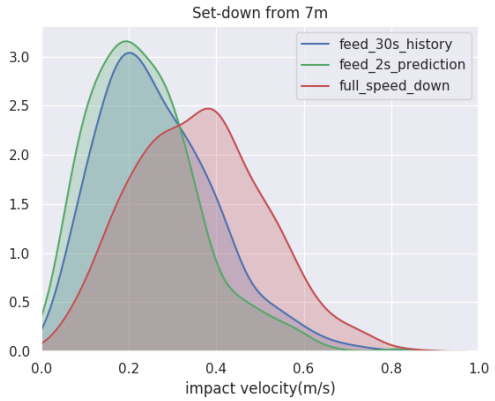


FIGURE 4.4: Results of end-to-end training of three agents.

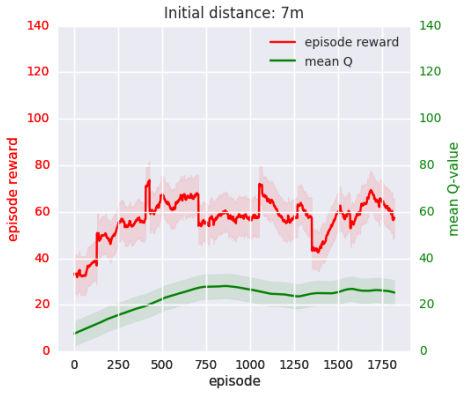


FIGURE 4.5: Accumulated reward and state-action value during for training agent knowing "future".

However, it is notable that the state did not have an impact on the results because both agents performed equally well on the 500 testing cases. So the agent is not doing anything useful with the state. Furthermore, the reward and mean state-action value curves in Figure 4.5 indicate that the agent converged to a policy at early stage of the training. Since the trick was to reduce relative velocity between objects, using a low payout speed already guaranteed the reduction as long as the barge was slow. Therefore, we made another comparison between the agent and another "monkey" with a constant payout at a low speed (see Figure 4.6). It turns out that the agent's policy was very similar to the new monkey. This implies that the learned policy is probably as simple as just getting closer to the barge as slowly as possible and then performing some random actions regardless of the information in the

input. Most likely, the action is dominated only by the relative distance provided in the input. The agent failed to utilize the "future" we provided or predict the "future" based on the history.

The algorithm probably suffered from the sparse rewards that led to the early convergence to a local optimal policy. The agent only received a reward for the last action before the terminal, whereas all immediate steps were not actively rewarded. Since we basically used Q-learning with experience replay, it failed to maintain the sample efficiency when most of the experiences in the buffer were not rewarded.

4.4 Learning Sub-Tasks

Considering a regular set-down procedure, it can be divided into three different sub-tasks (see Figure 4.7). First, when the load is far away from the barge, the crane operator usually implements constant payout regardless of barge motion (*full speed down*). Second, as soon as they are getting closer, the crane operator becomes more patient and starts searching for a proper moment for set-down while keeping the load close to the barge (*following*). Finally, he recognizes the possibility and completes the set-down. Therefore, the set-down skills are mostly required only when the load and the barge are very close. Therefore, we

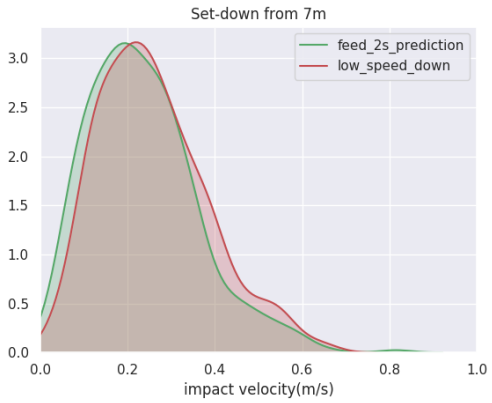


FIGURE 4.6: Comparing to a slower "monkey".

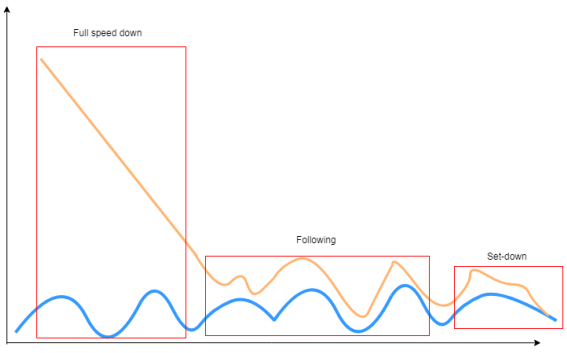


FIGURE 4.7: End-to-end set-down decomposition.

further simplified the end-to-end set-down by using a closer starting position. A shorter initial distance allows the agent to take significantly fewer steps to finish an episode and get rewarded sooner. The chance of having experiences with effective rewards being sampled is thus higher. Following the same hyperparameters provided in Table 4.2, the results of learning an "easier" set-down are presented in Figures 4.8, 4.9, and 4.10. This confirms that in this simplified setting, we achieved a more efficient learning progress than with an end-to-end set-down. Figure 4.10 shows a robust improvement on the accumulated rewards along the episode. The policy of the new agent outperforms the end-to-end agent at the very beginning of the training. Furthermore, the data in Figure 4.8 indicate a clear distinction between different input state variables. Given the same initial distance of 1m, the agent knowing

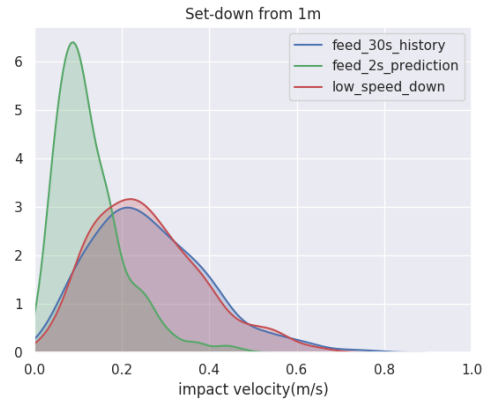


FIGURE 4.8: Histogram of impact velocity of 1m initial distance.

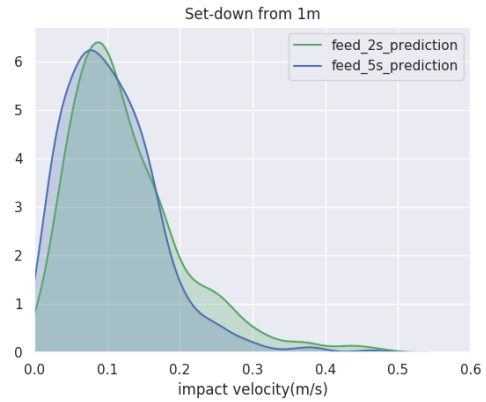


FIGURE 4.9: Effect of various prediction lengths provided to agent.

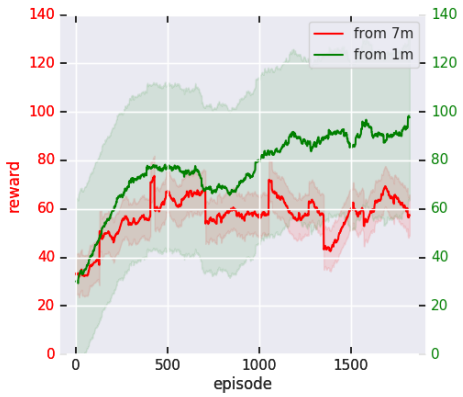


FIGURE 4.10: Reward curves of training from different initial positions.

about the history still struggled to deal with state inputs. However, the agent fed the prediction started to harness this information to make better decisions. The examples given in Figure 4.3 are attempts made by this agent with quite reasonable decisions. Counterintuitively, a longer prediction did not seem to improve results much more in this experiment (see Figure 4.9). Given different lengths of prediction, agents performed almost equally.

We further obtained an agent with substantial improvements (see Table 4.3) by feeding extra information about a upper "boundary", which is the offset of barge motion by 2m. The boundary specifies a range that encourages the agent to stay in (see Figure 4.11).

State variables	Mean impact velocity(m/s)
2s prediction + 2m boundary	0.0732 ± 0.0521
2s prediction	0.126 ± 0.0786
5s prediction	0.119 ± 0.0649
30s history	0.265 ± 0.134

TABLE 4.3: Set-down from 1m using different input state variables.

In addition, we trained the agent to stay within a "safe-zone" that followed the barge motion without hitting it. As shown in Figure 4.11, the maximum allowed margin (between the blue and green curves) was determined by a vertical limit to the barge position by 1m. The reward was given for every time step where the agent did not hit the barge or the limit. The "following" sub-task actually helped the agent always to retain a good starting position before utilizing the aforementioned "set-down" sub-task, which resembles the second stage in Figure 4.7. In general, it is easy to learn because the reward is assigned to every action and is less dependent on actions from history.

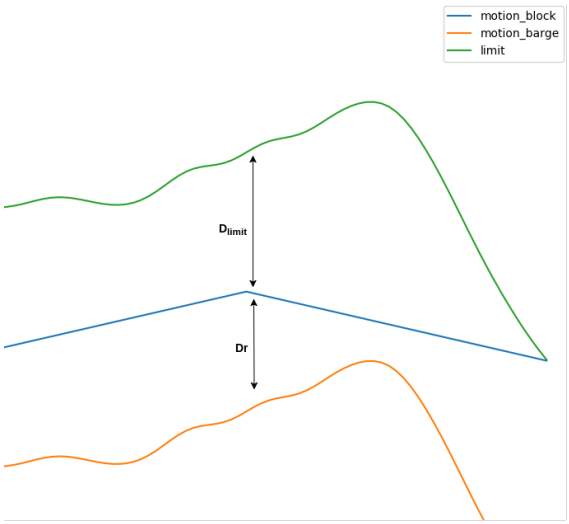


FIGURE 4.11: "Following" sub-task, agent (blue) collides with the limit (green), episode ends.

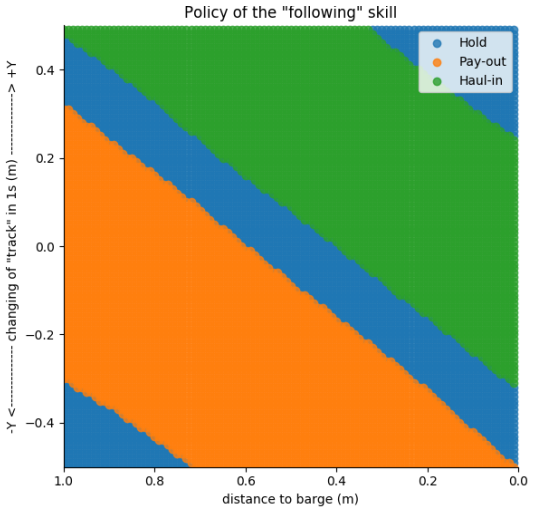


FIGURE 4.12: Visualization of "following" policy.

$$r_{barge} = \min(10, \frac{1}{D_r}), \tag{4.2}$$

$$r_{middle} = \min(10, \frac{2}{|D_r - D_{limit}|}), \tag{4.3}$$

where:

D_r = relative distance between the load and the barge
D_{limit} = distance to the middle of the "safe-zone"

The goal is to stay inside the "safe-zone" for as many time steps as possible. We proposed two reward functions with different focuses. The first idea was to reward the agent by the relative distance D_r between the load and the barge (see equation 4.2), and the other was to reward based on the distance to the middle of the "safe-zone" $|D_r - D_{limit}|$ so as to balance the distances to the boundaries (see equation 4.3). They both received -100 for hitting boundaries. We tested the agent under different input observations. Full observations consisted of 2-second "true" predictions on both the barge and the limit. We also tested the observation that excluded the prediction on the limit.

Observation	Reward function	Mean survival time (s)
barge	r_{middle}	138.31 ± 42.32
barge	r_{barge}	168.45 ± 47.83
barge+limit	r_{middle}	192.74 ± 23.92
barge+limit	r_{barge}	179.86 ± 44.63

TABLE 4.4: "Following" barge with different settings.

Table 4.4 summarizes the results for the different state and reward configurations. In general, knowing about both boundaries is very useful in order to complete the "following" even if the limit is just a simple offset. The reward on the distance to the "middle" is more effective than only on the relative distance, which makes sense because the agent can then prepare better for any significant changes. Furthermore, note that the result in the first row of Table 4.4 is worse than the those of the others. Since r_{middle} is fully explained by D_r and D_{limit}, this implies that a better reward is a function of input variables, and the performance will be influenced if the reward function includes information which is inaccessible by the agent.

It is also interesting to visualize the policy (see Figure 4.12). The x-axis stands for the distance to the barge, the y-axis indicates to which direction the "safe zone" is deviating from the center. It is obvious that the agent generally follows the change of the safe zone. It tries to get away from boundaries as fast as possible when they are about to contact.

4.5 Back to End-to-End

Our original ambition was to set-down from 7m, and the preceding sections showed that every sub-task is feasible to learn individually. Finally, there should be a method to leverage the sub-policies in a more generalized setting. Motivated by the HRL and TL approaches described in Chapter 3, we considered two distinct methods: TL and using hierarchy.

Firstly, we investigated the feasibility of using the hierarchy following the framework of *h-DQN* (Kulkarni et al., 2016). The idea was to decompose a complex task into sub-tasks, and we trained a *meta-controller* to select one of the sub-tasks to solve at different states. Each sub-task was associated to a particular *controller* that provided the *option* for the sub-task until it reached its goal state. When a controller was selected, the *meta-controller* gave it the complete mastery until it completed the sub-task. Intuitively, the meta-controller learned a master-policy of all the sub-policies/options. The controller learned a sub-policy over all primitive actions of a specific sub-task. In theory, controllers were updated through *intrinsic rewards*, which were only valid for the sub-task, and the meta-controller was rewarded by *extrinsic rewards*. The goal of the meta-controller is to maximize the expected discounted extrinsic rewards as standard value-based methods.

Sub-tasks	Intrinsic reward	Termination condition
Full speed down	equation 4.2	1 second
Full speed down	equation 4.2	2 seconds
Full speed down	equation 4.2	3 seconds
Following	equation 4.3	1 second
Following	equation 4.3	2 seconds
Following	equation 4.3	3 seconds
Set-down	equation 4.1	$Dr < 0$

TABLE 4.5: List of options for controller

Following the idea of hDQN, we defined the controllers and meta-controllers in Table 4.5 based on our original assumption in Figure 4.7. The original hDQN requires clear distinctions between sub-tasks in terms of goal states. For the end-to-end set-down, the distinctions are less straightforward. Instead, we set the goal states of *following* and *full speed down* sub-tasks to have different lengths of duration. Depending on the motion of the barge, the meta-controller was allowed to select one of seven sub-tasks to perform. We initialized the controllers for *following* and *set-down* sub-tasks by the pre-trained models in section 4.4. Finally, the extrinsic reward for the meta-controller was the same as the reward function of the training *set-down* sub-task.

The learning progress results of hDQN are shown in Figure 4.13, and two figures in the middle show the height and the moment of selecting the *set-down* controller. After a few episodes, the meta-controller figured out that it is better to select the set-down option as the distance is becoming smaller. This makes senses because the policy of the set-down controller was trained particularly for the short distances, which is more effective if used from the same initialization point.

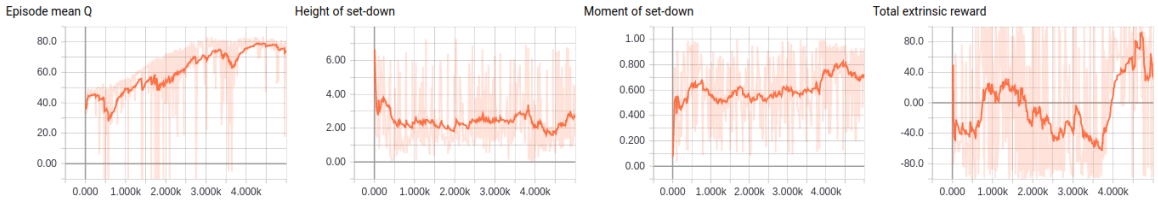


FIGURE 4.13: Mean state-action value of selecting options, height that agent decides to use set-down option, moment of an episode that set-down option is selected, total extrinsic reward

Alternatively, regarding TL, as proposed by Asada et al. (1994), we transferred the old policy by setting a more difficult starting position, but we did not change the MDP of the environment. We initialized the weights of the new agent by the policy of setting down from 1m and started a new training session where the new initial position was set to 7m.

In fact, the idea of *learning from easier tasks* worked very well (see Figure 4.14). Figure 4.14 illustrates a moving average of the accumulated rewards along episodes. The reward at the very beginning of TL is already much higher than the maximum reward of the end-to-end learning agent, which indicates that the old set-down policy generalizes quite well to a

FIGURE 4.14: Reward curve of transfer set-down sub-task back to end-to-end.

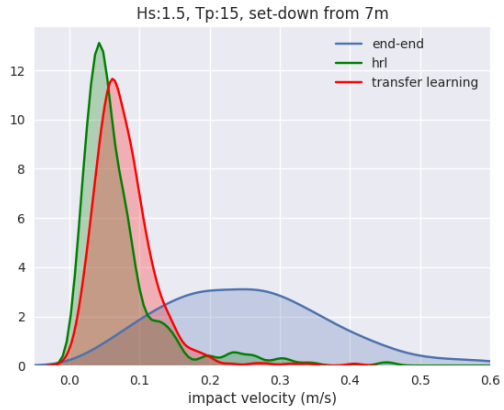


FIGURE 4.15: Histogram of impact velocity by TL, hDQN and end-to-end.

different initialization within a few episodes. In other words, through TL, the agent quickly learns to proceed to a familiar part of the state space from different parts of the state space without radically changing the policy distribution.

4.6 Conclusion

In Figure 4.7, it is noticeable that both TL- and HRL-based methods significantly outperformed end-to-end learning. In terms of asymptotic performance, hDQN performed slightly better because all sub-policies were already hierarchically optimal before the training. Then, the algorithm was able to reach the recursive optimality without changing the sub-policies too much. For TL, instead, the policy was changed in order to fit a different initial position, which might reduce the level of performance. Nevertheless, with respect to the computational complexity of training, TL is more efficient because it only maintains one parameterization of policy, whereas hDQN requires one for the controllers of each sub-task. The promising result of TL shows the potential of improving the learning progress for more dedicated environments where multiple skills are required. These sub-skills can be trained in sequence or separately with a more efficient reward function that only focus on achieving the goal of sub-skills.

Chapter 5

Advanced 1D

5.1 Advanced 1D Environment

In the basic 1D environment, simplifications were implemented to reduce the difficulty for the agent, which makes basic 1D extremely impractical for use in real situations. In new environment, substantial changes were made on top of basic 1D (see Table 5.2) such that it reflected some of the practical concerns. From now on, we will call this environment *advanced 1D*. In advanced 1D, the motion was still the heave motion between the vessel and the load, and the load was instead hanging 3m above the barge.

The first substantial change was including the ship response amplitude operator (RAO). We obtained the RAO of the relative motion between the crane tip and the barge and subsequently translated the wave spectrum defined by sea-state variables into a response spectrum. Then, we generated the relative heave motion in the time domain by IFT. By adding the motion of the SSCV, we could simulate the effect of a second ship in the environment and the effect of their relative motion between each other. In practice, a good set-down moment is always located at the transition between two wave groups where the relative motion is smaller (see Figure 5.1). In advanced 1D, we set Hs to 1.5 and Tp to 8, which were carefully chosen based on the operability assessment criteria (see section 2.4) at HMC.

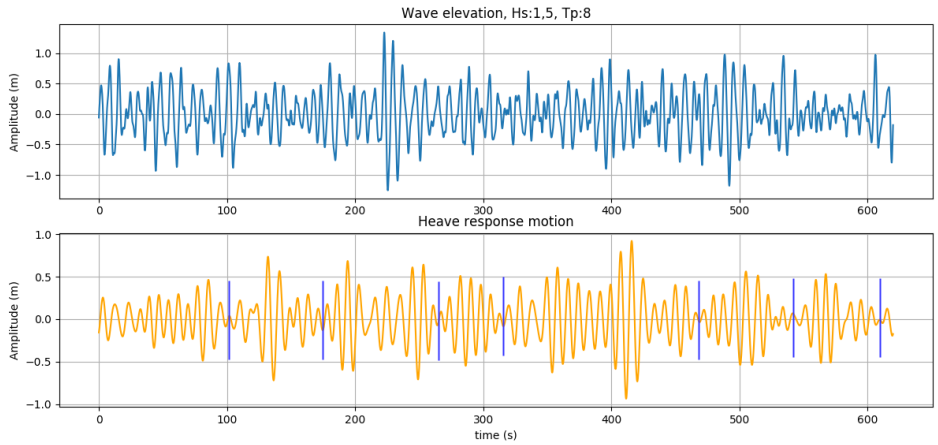


FIGURE 5.1: Wave elevation and corresponding heave motion responses. Heave motions can be separated by the number of wave groups, indicated by blue vertical lines. Relative motions are smaller at those transitions between wave groups.

% max speed	Ramp-up (s)	End speed (m/s)
0% to 50%	3s	2.5/30
50% to 100%	3s	5/30
100% to 50%	3s	2.5/30
50% to 0%	3s	0

TABLE 5.1: Ramp-up time of changing hoist speeds.

In addition, we included the ramp-up time (see Table 5.1). As introduced in section 2.3, the ramp-up plays a role as the delay in reaching the intent payout/haul-in speed. In basic 1D, we assumed that the hoist moved in the selected speed as soon as the action was taken, which is unrealistic. Instead, in the crane cabin, the operator controls the hoist speed by manipulating a joystick. The further the joystick is pushed, the faster the speed the hoist will reach with a longer ramp-up. In this case, the joystick is similar to a gas / brake pedal. Pushing forward accelerates and pulling backward decelerates. In practice, before executing a maneuver with the joystick, the crane operator first determines a speed by imagining the outcome of taking that speed with regard to ramp-up and relative motions. In many cases, common speed choices are 50% and 100% of max speed. In that sense, we limited the action space of advanced 1D to only three actions: *gas*, *brake*, and *hold*. Based on Table 5.1, the *gas* action linearly increases the payout speed by 50% of max speed in 3 seconds, and the *brake* slows down the payout by 50% of max speed in 3 seconds. The *gas* and the *brake* are no longer be used after reaching maximum speed. The *hold* only lasts for a single time step (0.2s).

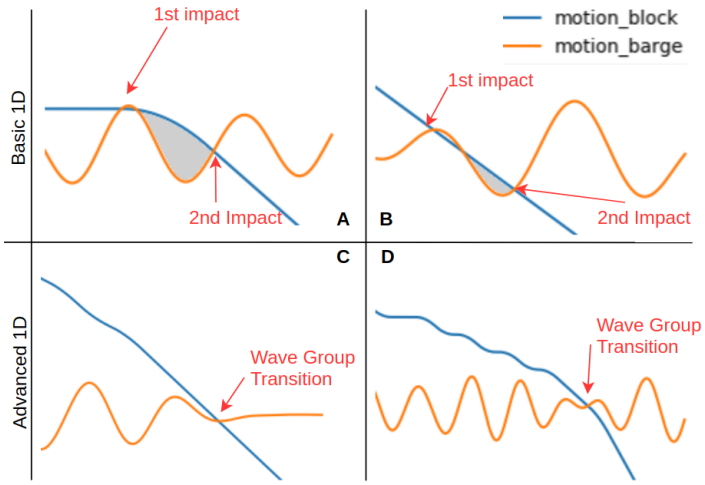


FIGURE 5.2: Plots A and B are good examples of set-down in basic 1D but unacceptable in advanced 1D because they cause new spacing between the load and the barge (gray area) again due to barge motions, which yields larger impact velocities at re-impact (see set-down angle as explained in Figure 4.2). In C and D, the agent managed to avoid re-impacts: the trick is to find a position where the load is faster than the barge motion.

Most importantly, advanced 1D also included an evaluation of re-impacts. As introduced in section 2.4, slacking for heavy loads takes a long time, the re-impacts provide additional contacts due to the barge motion after the first contact, and re-impacts sometimes could get out of control and damage the load. In addition, re-impacts also cause safety issues as there are many people on the deck. In 1D environments, in order to avoid re-impacts, there should not be a positive relative distance after the first contact. In reality, what the crane operators normally do is always take maximum payout speed as soon as the load touches the barge such that the increment of the length of the hoist wire is fast enough to cover any barge motion within the slacking period. An illustration is presented in Figure 5.2, where the agent needs to reduce the gray area as much as possible. In that sense, many of the successful episodes in basic 1D are no longer acceptable because they failed to avoid re-impacts.

However, avoiding re-impacts and resulting smaller impact velocities are somehow contradictory. As shown in Figure 5.2, the safety zone for small impact velocity does not always guarantee no re-impacts and vice versa. In that sense, the advanced 1D environment is more challenging than basic 1D because it requires the agent to have a better understanding of relative motions with a much lower control frequency.

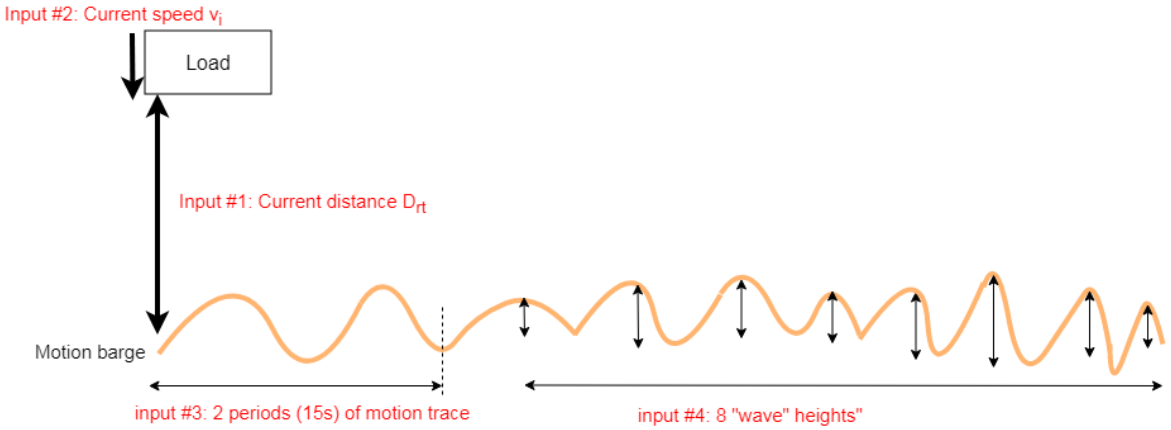


FIGURE 5.3: Input state variables include current speed, relative distance, and true motion prediction for 15s and 8 "wave heights."

The agent's input state variables are visualized in Figure 5.3. As stated before, identifying transitions between wave groups is a key tool to find proper set-down moments. We "spoiled" the agent by feeding "true" future motions. In practice, a wave group normally lasts for 10 wave periods. We therefore prepared the information about the future motion in two formats. For the nearest two periods of about 15 seconds, we fed the actual time trace of the motion, which is similar to basic 1D. The remaining eight periods were represented by eight corresponding wave heights. Intuitively, for set-down, only the nearest future is relevant, so we supplied the full detail. For recognizing wave group transitions in the future, wave heights are the most useful information.

It is worth noting that, in practice, haul-in is not used very often in set-down. This is the case simply because the total ramp-up time from haul-in mode to maximum payout

Setting	Value
Initial D_r	3m
Time step	0.2s
Episode length	300s
Hoist speeds:	0, 4.5/60, 9/60
Actions	Gas (speed up by 4.5/60), brake (slow down by 4.5/60), hold
Ramp-up	1.5/60 m/s^2
Action duration	3s for gas and brake, 0.2s for hold
Sea state	Hs:1.5m, Tp: 8s
Response model	RAO between barge and crane tip

TABLE 5.2: Environmental settings of advanced 1D.

will certainly cause re-impacts (e.g.,. see bottom right example in Figure 4.3). Therefore, in advanced 1D, we excluded haul-in mode, and the agent switched to the maximum payout mode as soon as the first contact was made.

Finally, based on the reward function of set-down in basic 1D, we added an additional penalty on re-impact with proportion to the integral of the relative distance after first contact (the gray area in Figure 5.2). Since the task becomes harder, we slightly loosened the threshold on impact velocity by 0.5 m/s. The reward function is defined by:

$$r = \begin{cases} \max(200, 10/v_i), & \text{if } 0 < v_i < 0.5, \\ -30, & \text{if } v_i > 0.5, \\ 10 \cdot A_r, & \text{if } A_r > 0. \end{cases} \tag{5.1}$$

where:

A_r = area of grey region in Figure 5.2, magnitude of re-impact
v_i = impact velocity

5.2 Results

Based on the hyperparameters suggested in Table 5.3, the results were positive in general. As shown in Figure 5.5 and Table 5.4, the agent obtained a better policy than simply using full speed down. Over 500 testing attempts, the agent improved the mean impact velocity by 50%. Note that the agent's distribution is right tailed, which probably implies that the agent prioritizes avoiding re-impacts over large impact velocities because the allowed velocity was increased to 0.5 m/s.

In Figure 5.6, examples of good trails are displayed with the agent's actions along the trajectory. Apparently, the agent generally made correct decisions. When the agent saw there was an opportunity nearby, it took aggressive actions to get closer (see plots B and C in Figure 5.6). In addition, when the waves were pretty strong, the agent waited many steps

Setting	Value
Learning rate	0.001
Gamma	0.99
Exploration ϵ	0.5-0.1
ϵ decay	to 0.1 in 50000 steps
Batch size	32
Target update frequency	10000
Replay buffer	50000
Update frequency	1
Num hidden layers/units	1/200
Activation	Sigmoid
Optimizer	Stochastic Gradient Descent (SGD)
Algorithm	Double Q-Learning

TABLE 5.3: Hyperparameters for training in advanced 1D.

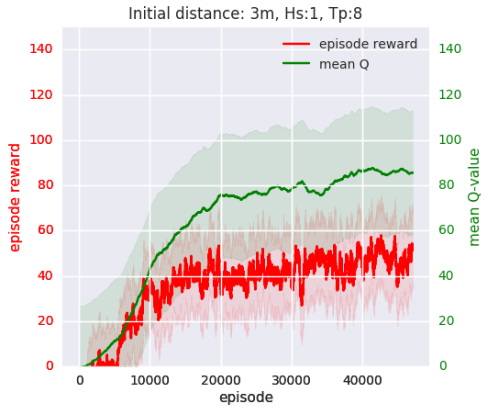


FIGURE 5.4: Learning progress in advanced 1D.

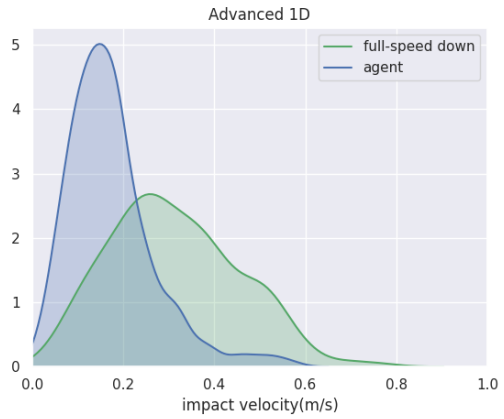


FIGURE 5.5: Histogram of impact velocity of 500 testing episodes by agent and "Monkey".

until it saw the chance (see A and D in Figure 5.6). The agent prepared itself to set down on those transitions of wave groups where the motion amplitude was in general lower than that of other positions. Unsurprisingly, these are the most probable positions where rewards on re-impacts and impact velocities do not contradict each other.

5.3 Improving by MCTS

The results in Table 5.4 indicate that the agent still caused a notable number of re-impacts, which accounted for 8% of the total episodes. Meanwhile, many actions of the examples shown in the Appendix B are obviously not optimal where the agent should have waited longer or responded more quickly. For that reason, the current policy seems still far from satisfactory. For many board games (e.g., Go and Othello), it is almost a common choice to use simulation-based search methods for policy improvement (Browne et al., 2012). One of the most fundamental requirements of executing simulations is the presence of a transition

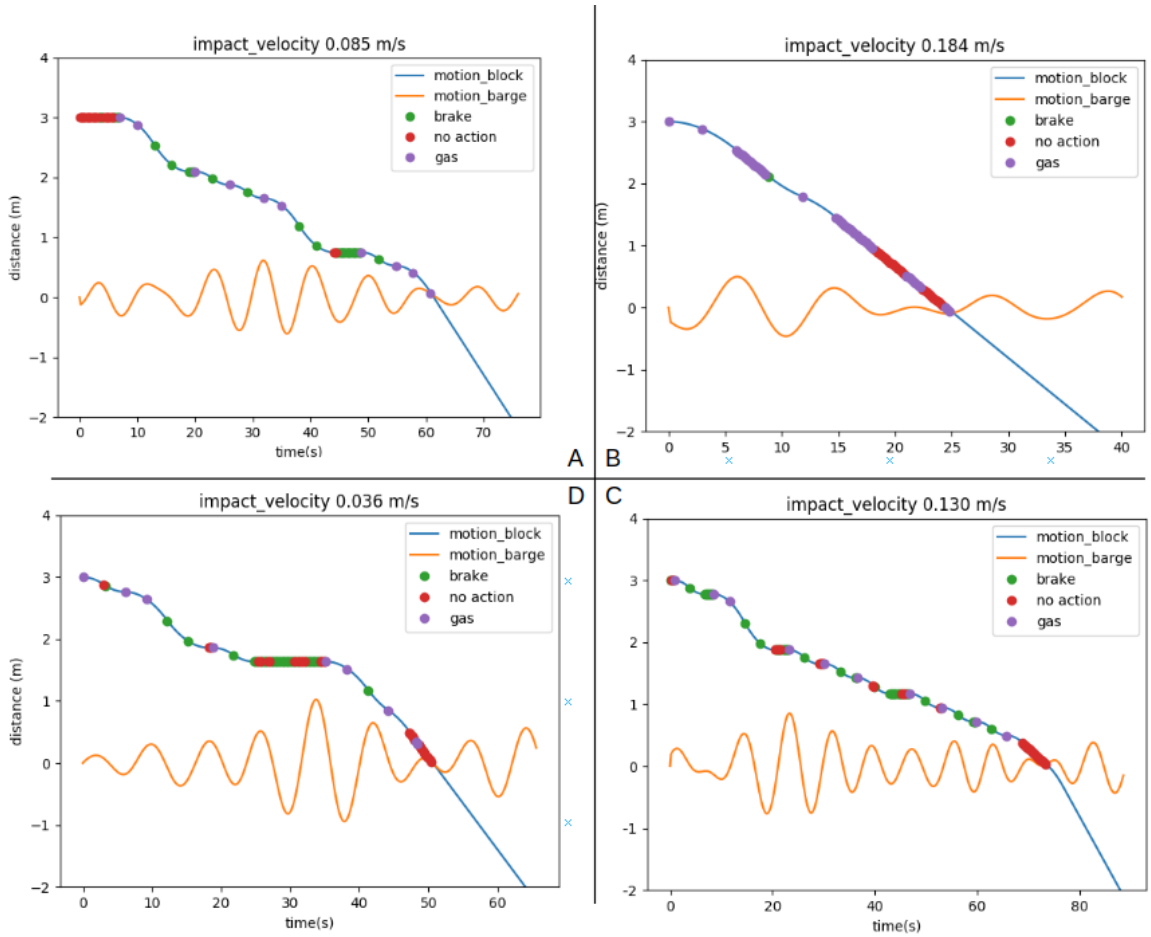


FIGURE 5.6: Examples of good set-down attempts with small impact velocities and no re-impacts.

	Mean impact velocity (m/s)	**Number of re-impacts**
"Monkey"	0.341 ± 0.14	121
Q-Learning agent	0.176 ± 0.085	35

TABLE 5.4: Results for 500 testing episodes in advanced 1D

function of the MDP $s' = \mathbb{P}[s'|a, s]$ and a reward function $r = \mathbb{E}[r'|s]$. Therefore, in advanced 1D, the search methods can be carried out because the barge motion is predefined and not affected by the agent's actions such that the transition model and the reward function are fully determined.

Motivated by the *AlphaGo* algorithm (Silver, Huang, et al., 2016), during the testing, we executed Monte Carlo tree search (MCTS) to improve state-action value estimations. For the *roll-out* policy, we used the policy obtained from section 5.2. For the selection and expansion stage, the child nodes were selected by UCT (see equation 3.25). Instead of using the extra value function, we simply backpropagated state-action values by the true return of simulations. Deviating from the standard tree search, we only executed MCTS at the moments of interest (see in Figure 5.7).

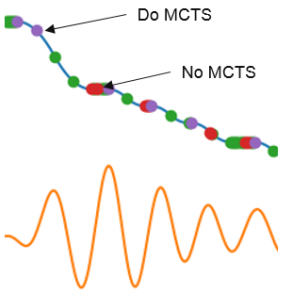


FIGURE 5.7: MCTS is executed when the policy suggests changing speed.

Setting	Value
Max depth	2
UCT exploration constant	1
Reward re-scale	[0,1]
Simulations per search	100
Roll-out policy	Achieved in 5.2

TABLE 5.5: Settings of MCTS

Num. searches	Num. re-impacts	Mean imp. vel. (m/s)
0	35	0.176 ± 0.085
10	0	0.129 ± 0.095
20	0	0.123 ± 0.104
30	0	0.118 ± 0.108
>30	0	0.108 ± 0.072

TABLE 5.6: Results on 500 testing episodes executing a different number of MCTS searches per episode.

Specifically, at every time step, we first sampled an action from the roll-out policy. Then, the MCTS was carried out only if the sampled action would suggest to speed up or slow down. This is mainly because most of the failures are due to incorrect decisions about changing speeds, and they last for 3 seconds (see the examples in Appendix B), which are more worthy of re-evaluation than the "waiting" action, which only lasts for 0.2 seconds.

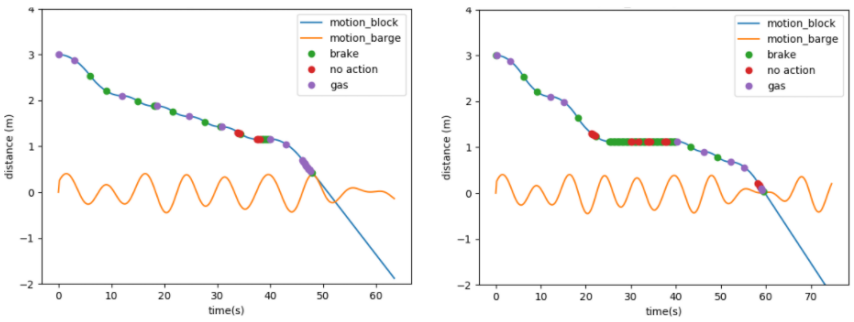


FIGURE 5.8: Given the same barge motion, the agent (left) failed to avoid re-impact, but MCTS (right) succeeded.

Based on the settings in Table 5.5, we tested the performance of combining MCTS with respect to the maximum number of searches allowed per episode (see Table 5.6). It turns out that the policy had already significantly improved after only 10 total searches. Meanwhile, using more searches does not produce much better results. Using 30 or more MCTS only improved the impact velocity by 0.015, which probably implies that the search implemented at the beginning guided the agent to an advantageous position where the agent could rely

Num. searches	Num. re-impacts	Mean imp. vel. (m/s)
0	35	0.176 ± 0.085
5	16	0.174 ± 0.181
10	9	0.182 ± 0.205
15	5	0.173 ± 0.184

TABLE 5.7: Results on 500 episodes that allow MCTS only when the relative distance is smaller than 1 meter.

on its own actions from that position onwards. An example is shown in Figure 5.8, and the MCTS's choice is on the right. It is obvious that a lot of gas actions from the agent were rejected by MCTS. Instead, it encouraged the agent to wait longer for a better position.

Moreover, in order to know how well MCTS can recover from the mistakes made by the agent's policy, we did another experiment that only executed MCTS when the relative distance between the load and the barge was small. The load was firstly transported by the agent's policy until the distance was smaller than 1m then MCTS was used for selecting actions. The results are presented in Table 5.7. Interestingly, MCTS only avoids re-impacts with more searches, but it does not improve impact velocity. This is probably because the agent wrongly entered a strong wave group on its own where it was only possible to avoid re-impacts. Therefore, it is more effective to plan thoroughly at the beginning instead of asking for MCTS when the agent is already in a difficult situation.

5.4 Improving by Imitation Learning

Although using MCTS improves the overall performance, it is impractical to execute MCTS in real-time applications. We would rather have a model that imitates the decisions made by MCTS. In that sense, MCTS can be considered as an expert providing good demonstrations to the learning agent.

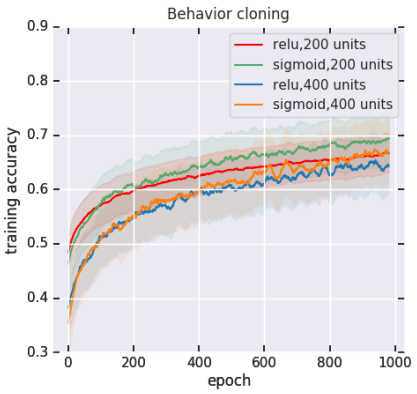


FIGURE 5.9: Training accuracy curves of different network topologies.

Setting	Value
Num of samples	123994
Batch size	32
Learning rate	1e-4
Num of layers	1
Optimizer	Adam
Total epoch	1000
Loss	Categorical cross-entropy
Dropout	0.2

TABLE 5.8: Hyperparameters for behavior cloning.

Given the expert data, the simplest approach is to learn through behavior cloning, where a model is trained to map state variables to the expert's action or the action-value function to end up with classification or regression problems. Guo et al. (2014) discovered that such methods were very effective in some Atari games. They collected expert data by running offline MCTS planning and used cross-entropy function for the loss. They found that the classification model achieves higher online testing scores than the regression model. Following this concept, we prepared a dataset that contains 2,000 episodes (123,994 experiences) completed by executing MCTS with the configuration in Table 5.5 and launched the training session based on Table 5.8. The learning curves presented in Figure 5.9 indicate that the training converged at nearly 70% accuracy. According to the testing results displayed in Table 5.10, the behavior cloning is even worse than the policy obtained in section 5.2. As argued by Ross and Bagnell (2010), naive behavior cloning can easily fail because the network is only trained under the distribution of the expert's policy. The agent is unable to recover from the state, which is either induced by a mistake or unseen in the training set.

Therefore, we implemented the *dataset aggregation* (DAgger) algorithm. DAgger is supposed to achieve a "no regret" property (Ross, Gordon, et al., 2011) by aggregating data that contain all previous experiences by the expert and the learner. The pseudocode of DAgger is presented in Appendix A.2. We started with an initial dataset that was used for training behavior cloning and initialized the learner by the model of behavior cloning (iteration 0). We then ran another 2,000 episodes using the learner and query MCTS for every time step. We augmented the dataset by the MCTS action (expert) whenever it differed from the learner's action. The action was decided by stochastic sampling with a 50% chance for both the learner and the expert. Hence, we maintained a dataset that contains the trajectories of the expert as well as mistakes by the learner with corrections provided by the expert. The learner was then updated through supervised learning on the aggregated dataset using the same hyperparameters as those in Table 5.8 (iteration 1).

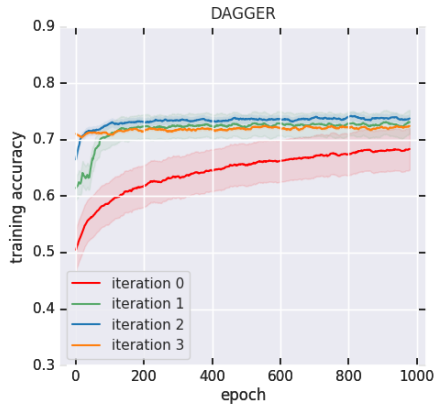


FIGURE 5.10: Training accuracy curves of iterations.

Model	New entries
Before training	123994
Iteration 0 (Behavior cloning)	11970
Iteration 1	8920
Iteration 2	5570
Iteration 3	5420

TABLE 5.9: Number of new entries being appended to the dataset by the model. Note that an entry is added only if expert and model provide different actions.

We completed 3 iterations of this procedure. As depicted in Table 5.9, the model of iteration 3 produced almost as many new entries as iteration 2 to the dataset; that is, the model of

the new iteration resulted in the same amount of mistakes as in the previous iteration. Thus, the model did not improve. The learning curves of the DAgger iterations are presented in Figure 5.10. Most of the learning happened in iteration 0 and iteration 1, and it contributed mostly to the new entries of the dataset. However, the learning curves do not seem very promising since they all converged at 70%, which is still similar to the results produced by behavior cloning.

Apart from supervised learning, we also tested the possibility of directly imitating the underlying policy of the expert. We simply stored the expert's experience in the agent's replay buffer and sampled mini-batches to compute the TD error using the same hyperparameters displayed in Table 5.3.

Surprisingly, the mean state-action value is improved but the reward largely decreased (see Figure 5.11). Therefore, the agent expected to achieve a higher reward by actually taking wrong actions. This probably means that the agent is getting closer to the expert's action-value function, which is only workable in these demonstration episodes. Because the update of action-value function merely occurs through samples from the expert's experience, it fails to interact with the current learning environment. Therefore, it probably over-fits to the distribution of the expert's policy while failing to generalize to the unseen states.

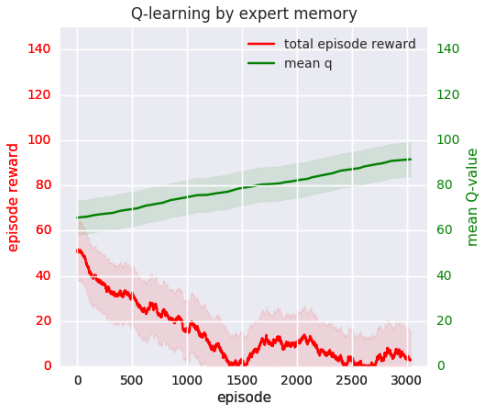


FIGURE 5.11: Q learning by replaying expert's memory

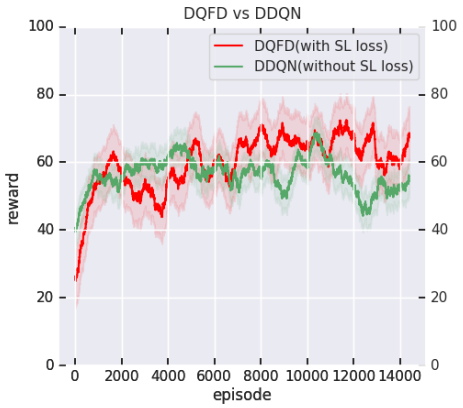


FIGURE 5.12: Accumulated rewards of DQFD and Double Q-Learning. DQFD slightly outperforms Double Q-Learning in the end.

Eventually, we implemented the DQFD algorithm (the pseudocode is presented in Appendix A.3), which facilitates the concept of supervised learning in the context of RL. This was mostly done by introducing a supervised loss term J_E in the object function (see equation 5.3). In every replay batch, a fixed partition $(s_{E1}, a_{E1}, s_{E2}, a_{E2}, ...)$ of the expert's dataset was sampled together with the regular agent's experiences. The expert dataset contains the state s_E and corresponding action a_E determined by the expert. For every sampled S_E from the expert dataset, the supervised loss J_E was achieved by subtracting the state-action value

of the agent's choice and the expert's choice:

$$J_E = \max_{a \in A}[Q(s_E, a) + m] - Q(s_E, a_E), \tag{5.2}$$

where m is a constant margin set to 0.8, The overall loss function is given by:

$$J(Q) = J_{DQ} + \lambda_1 J_E + \lambda_2 L_2(Q), \tag{5.3}$$

where J_{DQ} is the standard double Q learning loss (equation 3.16), and it is regularized by L_2-norm. Therefore, the supervised loss provides additional feedback on actions that are familiarized by the expert, which enables interactions between the agent and the environment while querying the expert. We used the dataset aggregated during the last iteration by the DAgger algorithm as the expert dataset (155,874 entries) and kept the proportion of the expert data at 20% in every mini-batch. We compared the learning curve of DQFD and vanilla Double Q-learning (see Figure 5.12). Both algorithms were initialized by the same policy obtained from section 5.2. The result shows that Double Q-learning did not improve further on its initial policy when the reward of DQFD was low at the beginning but eventually exceeded Double Q-learning.

Method	**Num. re-impacts**	**Mean imp. vel. (m/s)**
Double Q-learning	35	0.176 ± 0.085
MCTS	0	0.108 ± 0.072
Behavior Cloning	40	0.189 ± 0.101
DAgger	30	0.162 ± 0.095
Q-learning with expert's experience replay	168	0.392 ± 0.35
DQFD	40	0.154 ± 0.07

TABLE 5.10: Results of 500 testings of all imitation methods proposed in this section. Note that the Double Q-learning agent is the policy achieved in section 5.2, and MCTS is best configuration according to Table 5.6

The summary of all imitation methods is presented in Table 5.10. Obviously, none of the methods provided a substantial improvement on the original policy. The DAgger and DQFD were slightly better in terms of impact velocity but still failed to significantly reduce the number of re-impacts. Regarding MCTS, the action was chosen by the averaged return of rollouts, so the action was not necessarily fully dependent on the current state. Therefore, the action could even differ given the same state variables such that the distribution of the expert dataset could be inconsistent, which makes it difficult to imitate.

5.5 Conclusion

In advanced 1d, we extended the basic 1d environment by the realistic relative motion and the ramp-up time for each of the actions. We also took the magnitude of the re-impact into

account. The standard Q-learning was not consistent at reducing re-impacts while it was fundamentally improved by applying the tree-search method. We therefore tried to imitate the behavior of the search-based method so as to have the policy working for a real-time case. However, the improvement of the imitation learning was quite unremarkable. It could be that the input features are still insufficient to determine the optimal action or that the reward function of the re-impact is not appropriate.

Chapter 6

Basic 2D

6.1 Basic 2D Environment

For this section, we extended the environment to two dimensional space. A general visualization of this environment in given in Figure 6.1. In contrast to the 1D environments, the load and the barge are presented in different shapes. The goal is to set down the load as close to the left of the *bumper* as possible while reducing the impact force as much as possible on the bumper and the barge in both the vertical and horizontal directions. Since objects are no longer points, the agent has to consider geometries and the dynamics of the load. Notably, the barge here is not affected by waves, and it keeps standing-still; however, the hoist is moving with respect to crane vessel's RAO on heave (z-axis) and sway (y-axis). In addition, ramp-up time is ignored. Therefore, we named the environment *basic 2D*.

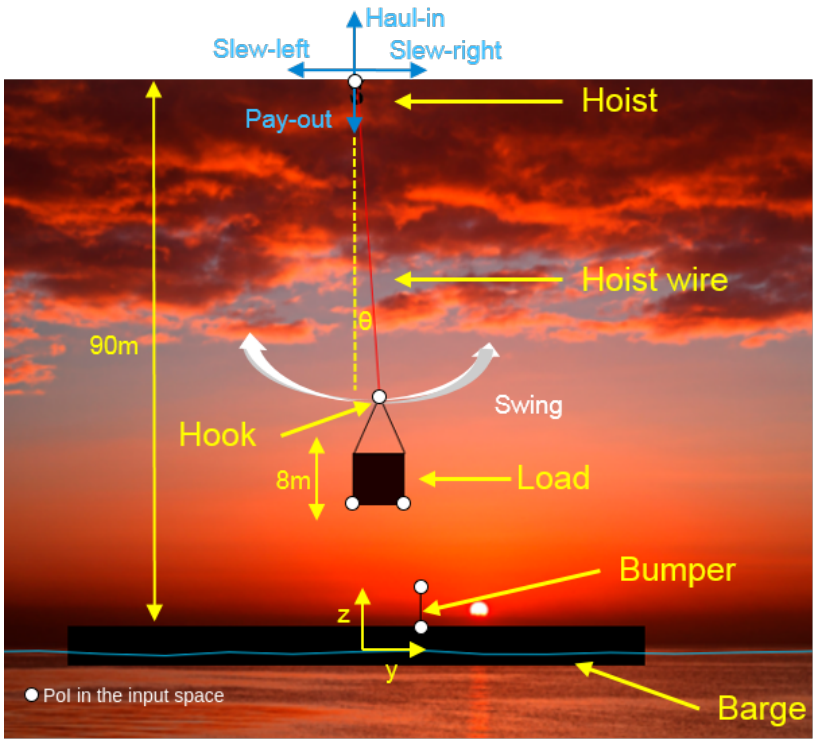


FIGURE 6.1: Visualization of basic 2D environment.

Specifically, the load was an 8m x 8m square hanging by a crane hoist located 90 meters above the barge. The origin was located at the intersection of the barge and the hook extension line (see Figure 6.1). For every episode, the 10m-bumper was placed perpendicularly at a random location. We also set a random swing amplitude θ representing the angle between the hoist wire and a vertical line for a different initial loading position (see Figure 6.1). The action space contains five actions: payout, haul-in, slew-left, slew-right, and hold. Payout and haul-in are the same as those in the 1D environment, and slew-left/right moves the hoist laterally. For the sake of simplicity, we ignored the ramp-up time for every action, and we assumed all actions were operating at the same speed (see Table 6.3), which resembles the properties of the auxiliary hoist used in the 1D environment.

We used the *pymunk* library to define physical objects in the environment and compute impact forces between objects. The episodes terminated whenever there was a continuous impact load exerted on the barge for 5 seconds. Instead of measuring the impact velocity and subsequent relative motions, we evaluated the set-down by looking at the maximum impact force during the whole episode, which allowed us to include the influence of re-impacts. The time step of simulation was set at 0.2 seconds.

One of the added challenges in this environment is the control of lateral movement. The set-down can be considered as a pendulum system, where any action causing a change in the length of the hoist wire or the hoist position will have an impact on the period of the pendulum. In practice, due to the vessel motion and hoist movement, the swing might emerge in any moment. It is crucial to avoid the swing motion for any unexpected contacts, which requires an explicit skill to control the lateral movement of the load. In that sense, it is considerably difficult to directly solve the problem solely with the information provided about the set-down goal. As a result, the problem might suffer from the same issue as in basic 1D, (i.e., the sparse reward and long-term objective credit assignment). Based on the results of TL in basic 1d, we defined necessary skills for have a good set-down and transferred the knowledge of easier skills to harder skills.

6.2 Skills Transfer

Inspired by real set-down operations, the set-down in 2D can be divided into several skills (see Table 6.1). In contrast to basic 1D, skills under this division are implicitly correlated, and

Skill	**Goal**	**Requirement**
Stabilizing	"Kill" swing motion	NA
Aligning	Find and align with bumper	No swing motion
Approaching	Stay right above bumper	Stay aligned with bumper
Set-down	Be close to bumper and small impact force	Approaching

TABLE 6.1: 2D set-down sub-skills definitions.

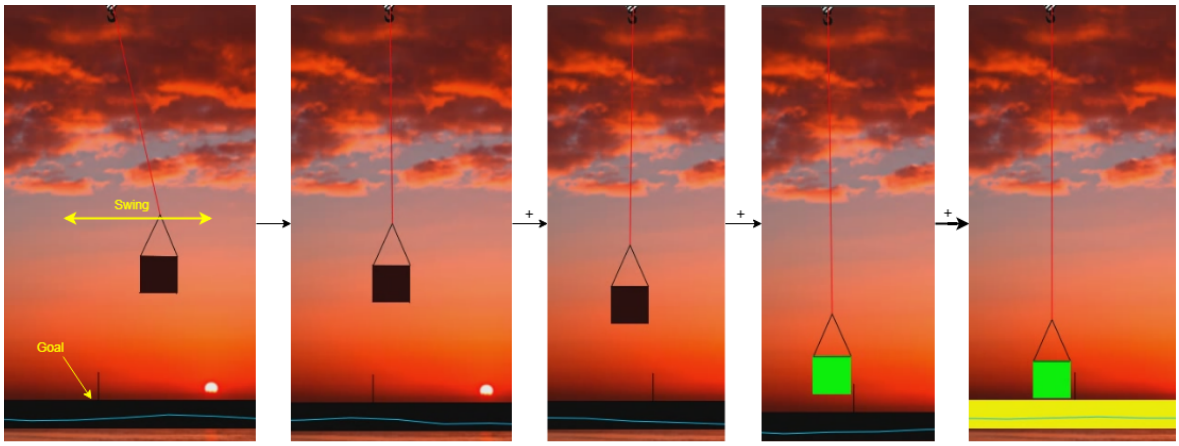


FIGURE 6.2: Visualization of skill transferring.

advanced skills can only be learned with the mastery of low-level skills. For instance, the result of the *aligning* skill is only acceptable if the swing motion is killed. For the *approaching* skill, the load can not move horizontally away from the bumper. Otherwise, a good set-down distance can not be achieved. A clear visualization is presented in Figure 6.2. In relation to the value function of a skill, the high-valued parts of the states of an advanced skill are essentially the subset of the high-valued states of a lower-level skill. The agent was encouraged to shape the value function of advanced skills starting from the lower level ones. Unsurprisingly, such logic aligns with the nature of TL.

6.2.1 Training Details

We implemented TL following the order in Table 6.1. The transfer was achieved by changing the reward function from the source task to the target task while fixing the state and action space (Atkeson and Santamaria, 1997). The details about the state and action space are described in Tables 6.2 and 6.3. Note that the coordinate system follows the illustration in Figure 6.1, where y_n stands for the y coordinate of PoI n, and v_y stands for the velocity along the y-axis. The initialization of every episode is presented in Table 6.4.

PoI	State variable
Hoist	y_h
Hook	θ_{ho}, y_{ho}, v_y^{ho}, v_z^{ho}
Load lower right	y_{lr}, z_{lr}
Load lower left	y_{ll}, z_{ll}
Bumper lower	y_{bl}, z_{bl}
Bumper upper	y_{bu}, z_{bu}

TABLE 6.2: State space of basic 2D.

Action	Definition	Speed(m/s)
Payout	Extend hoist wire	0.15 (5/30)
Haul-in	Shorten hoist wire	0.15
Slew-left	Move hoist left	0.15
Slew-right	Move hoist right	0.15
Hold	0	

TABLE 6.3: Discrete action space of basic 2D.

In addition to Double Q-learning, we included the state-of-the-art policy gradient algorithm, proximal policy optimization (PPO) (Schulman, Wolski, et al., 2017). This algorithm

Setting	Value
State space	13
Action space	5
Episode length	1,500 time steps
Time step	0.2 second
Initial swing amplitude θ	10 to 15 degrees
Initial hoist wire length	55 to 65 m
Initial bumper position	-10 to +10m away from origin
Sea state	Hs:1.5, Tp:8
RAO	Crane vessel heave, sway, surge

TABLE 6.4: Environmental settings for basic 2D.

specifies the "trust region" by a clipping constant on the advantage value, which allows multiple epochs of batch updates in every gradient update (the pseudocode is presented in Appendix A.5). The advantage value was estimated by the generalized advantage estimator (GAE) (Schulman, Moritz, et al., 2015) based on the $TD(\lambda)$ return. The main motivation for using the policy-based method here was the better convergence property. The proposed sub-skills are quite straightforward and simple, implying that the loss function could be smooth without a complex local concave. The convergence property of the policy gradient enables faster updates toward local optima. The hyperparameters of Double Q-learning and PPO for training sub-skills are provided in Tables 6.5 and 6.6.

Setting	value
Learning rate	0.0001
γ	0.99
Exploration ϵ	0.5-0.1
ϵ decay	to 0.1 in 500k steps
Batch size	32
Target update frequency	100k
Replay buffer	500k
Update frequency	1
Num hidden layers/units	2/64
Activation	sigmoid
Optimizer	Adam

TABLE 6.5: Hyperparameters for Double Q-learning.

Setting	value
Learning rate	0.0001
Discount factor γ	0.99
Clipping factor ϵ	0.2
Trajectory length	1500
Num mini-batches	32
Num epochs	10
λ return	0.95
Num hidden layers/units	2/64
Activation	tanh
Optimizer	Adam

TABLE 6.6: Hyperparameters for PPO.

6.2.2 Stabilizing

The goal of stabilizing is to slow down ("kill") the swing motion induced by any action. In other words, given a random initial state, the load should be stabilized at the equilibrium position of a pendulum with a low horizontal velocity as quickly as possible. We thus

defined the reward function based on the goal (see equation 6.2), where θ_{ho} is the swing amplitude defined in Figure 6.1, and v_y^{ho} is the horizontal velocity of the hook. We encouraged the agent to kill swing without moving too far away from the origin. We initialized every episode by a random swing amplitude between 10 to 15 degrees and a random hoist wire length between 50 and 60 meters. We carefully configured the general damping of the pendulum system in the basic 2d such that the swing wouldn't stop by itself for at least 1,500 time steps. This prevents the agent to cheat by just waiting for the swing being killed by itself. The condition for the stabilization is:

$$|\theta_{ho}| < 1 \text{ and } v_y^{ho} < 1, \tag{6.1}$$

with the reward function:

$$r = \begin{cases} 1, & \text{if Eqn 6.1,} \\ -1, & \text{if } |y_h| > 5, \\ 0, & \text{else.} \end{cases} \tag{6.2}$$

In fact, the problem is quite simple; The optimal action is fully determined by the relative position and speed, which are θ_{ho}, y_{ho}, v_y^{ho} and y_h. Figure 6.3 displays the reward curves of PPO and Double Q-learning when only feeding those four state variables. Both algorithms worked but PPO learned much faster than Double Q-learning and led to a much higher asymptotic reward. Since the episode length was set at 1,500 time steps, the curve indicates that PPO was able to stabilize the load within 400 steps.

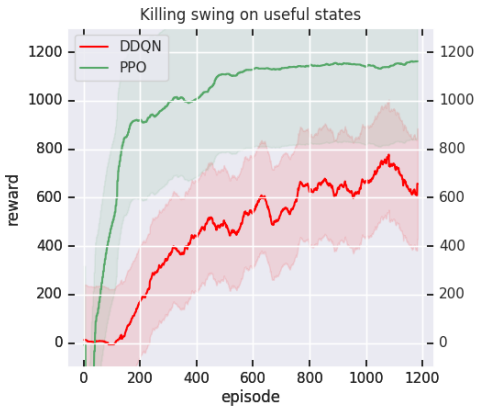


FIGURE 6.3: Reward curve on training stabilizing with only most useful state variables.

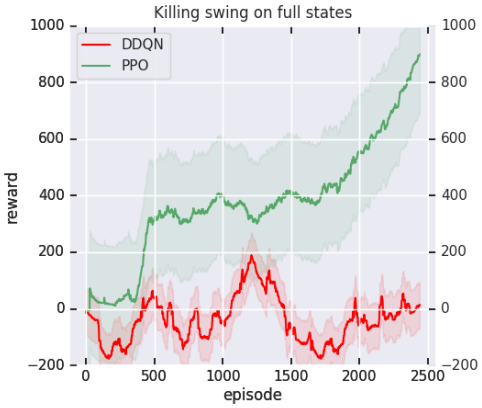


FIGURE 6.4: Reward curve of PPO and Double Q-learning on training stabilizing with full state variables.

However, when the algorithms were fed by all state variables in Table 6.2 without changing hyperparameters, they both had difficulties in terms of feature selection (see Figure 6.4). Nevertheless, PPO eventually managed to solve the problem, whereas Double Q-learning failed to reproduce the same results as in the previous experiment.

6.2.3 Aligning

The aligning skill is used to minimize the horizontal distance between the lower right corner of the load and the bumper while satisfying the condition on stabilization (see equation 6.1). Note that for every episode, we randomly placed the bumper on the barge between -10 and 10m away from the origin. From that point on, we fed the agent with all state variables displayed in Table 6.2.

We proposed two types of reward function (see equations 6.4 and 6.5). The step reward was assigned as soon as the load and the barge were close d_{lb} horizontally. The continuous reward incorporated the distance measurement inversely. The function approximation was initialized by the weights of the stabilizing skill. Therefore, the agent was already capable of killing swing. The condition for the alignment is:

$$\begin{aligned} &d_{lb} = y_{bl} - y_{lr}, \\ &0 < d_{lb} < 2 \text{ and Eqn } 6.1, \end{aligned} \tag{6.3}$$

with the reward function:

$$r = 1, \qquad \text{if Eqn } 6.3, \tag{6.4}$$

$$r = \min(1, \frac{2}{d_{lb}}), \quad \text{if Eqn } 6.1. \tag{6.5}$$

As presented in Figure 6.5, the continuous reward function led to a more robust enhance-

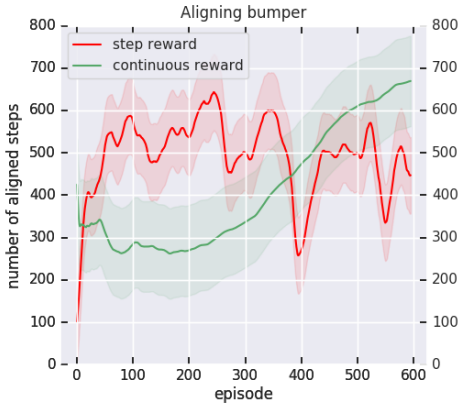


FIGURE 6.5: Learning curve of PPO based on stabilizing skill with two types of reward function. Note that the y-axis represents the number of steps of an episode where agents satisfy the same aligning requirements ($0 < d_{lb} < 2$ and no swing).

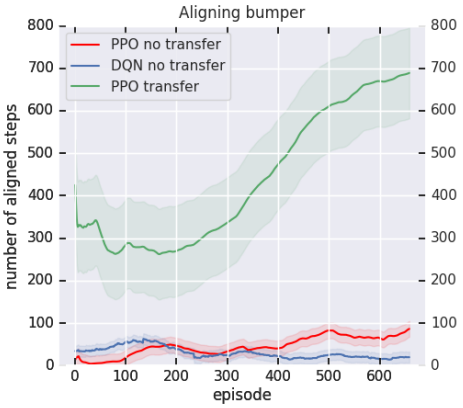


FIGURE 6.6: Learning curves of Double Q-learning and PPO training from scratch and PPO training with the initialization of stabilizing skill.

ment of performance, but the step reward seemed to be faster but less stable, and the policy switched between local optima dramatically. Moreover, the results in Figure 6.6 reveal a significant influence on transferring the skill when we compare the learning progress of TL and non-TL approaches. This implies that the agent found the "sweet zone" of the new reward

function on the state space much faster with a good initialization on the policy. Moreover, running PPO from scratch required many more episodes to start improving, whereas Double Q-learning just failed.

6.2.4 Approaching

Next, we tested the transfer from the aligning to the approaching skill by adding extra criteria on height h_{lb} from the load's lower right corner z_{lr} to bumper's lowest point z_{bl} (see equation 6.6). For every episode, we randomly initialized swing amplitude, hoist wire length, and bumper position.

$$h_{lb} = z_{lr} - z_{bl}, \tag{6.6}$$

$$r = 1, \qquad \text{if Eqn 6.3 and } h_{lb} < 5, \tag{6.7}$$

$$r = \min(1, 1 - \frac{h_{lb}}{100}), \quad \text{if Eqn 6.3.} \tag{6.8}$$

Similarly, we proposed two reward functions, as we did for training aligning. As shown in Figure 6.7, the continuous reward function (see equation 6.8) was slightly better because it assigned the reward more frequently than the step reward (see equation 6.7). The *shaping reward* provides information on the level of goodness of an action, which might be useful during backpropagation.

FIGURE 6.7: Number of steps in which the agent satisfies $h_{lb} < 5$.

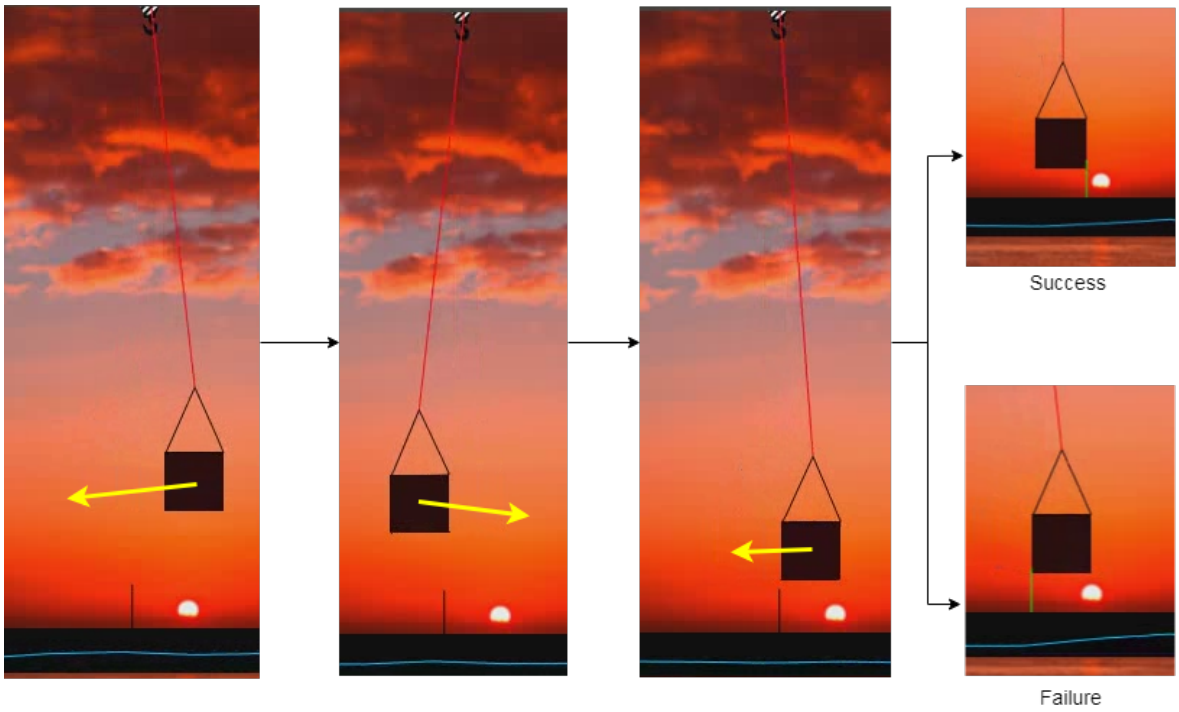


FIGURE 6.8: A typical example of policy learned directly from end-to-end; the agent is approaching the barge without the control of lateral movements, which is very likely to cause unexpected contacts on the bumper, leading to a poor set-down. The bumper turns green when it gets hit.

6.3 Results

We executed the final transfer based on approaching skill, which is subject to the reward on the max impact force F_{max} of the episode:

$$r = (\frac{5000}{F_{max}})^2. \tag{6.9}$$

The lower the max force is, the higher the reward the agent receives. As a reference, we also trained a PPO agent completely from scratch. It used the reward in equation 6.9 plus a shaping reward (see equation 6.5) because the distance is also a strict requirement.

As seen in Table 6.7 and Figure 6.9, the policy achieved by TL significantly outperformed the end-to-end method regarding the maximum impact force. Interestingly, we realized that the distribution of the end-to-end method in Figure 6.9 is split into two clusters, which are very scattered by the resultant distance to the bumper (y-axis). This is mainly due to the policy that the agent learned. Figure 6.8 illustrates a typical set-down attempt done using the end-to-end policy, where the agent completely neglects to slow down the swing motion during payout. As the swing is continuous, it is very likely that one of the lower corners of the load will hit the tip of the bumper at the highest excursion within a pendulum cycle. The fortunate outcome is that the load hits the left side of the bumper such that a good distance to bumper can be maintained (top right corner in Figure 6.8). However, if the load first hits the right side of the bumper and does not recover, the load simply stays on the

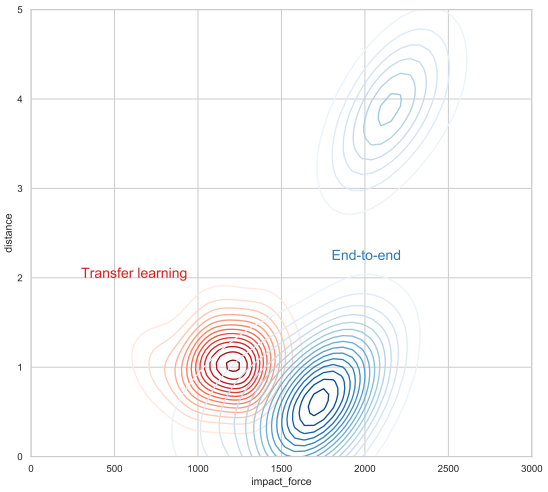


FIGURE 6.9: Distribution of impact force and distance of two methods over 500 testing episodes.

Method	Mean Max Impact Force	Mean Distance (m)
End-to-end	1819.77	1.045
TL	1163.61	1.183

TABLE 6.7: Results of two training methods on 500 testing episodes with random initial positions.

right side until the end, which results in a big margin to the target (bottom right corner in Figure 6.8) and the two clusters in Figure 6.9. Moreover, both situations fail to reduce the impact on the bumper because the lateral movement of load is rarely under control by the policy until it hits the bumper. In contrast, the policy derived by TL does not suffer from the issue because it is pre-trained in multiple supplementary skills that maintains a better distribution of states for the set-down.

6.4 Partial Observability

In practice, due to the precision of sensing equipment, many of the states involve error, noise or even the availabilities that are not ensured. Many crane operators still highly rely on their visual perceptions to infer the movement of objects. Although crane operators are unable to determine the actual value of every state, such as velocity and acceleration, they still know, with a reasonable accuracy, the direction or trend in which a state variable is changing.

Motivated by this fact, we proposed an experiment for training the aforementioned stabilizing skill but with insufficient state variables. The system dynamics is no longer determined by input state variables, resulting in the POMDP problem. For the input, we hid the hook velocity v_y^{ho} such that agent only saw geometrical information, including the hoist position y_h, the swing amplitude of the hook θ_{ho}, and the hook position y_{ho}, which resembles what the human eyes actually sees.

We tested two approaches by either stacking the four most recent frames of observations or replacing the MLP of the policy network of PPO by 128 single-layer LSTM units. The results are shown in Figure 6.10. It is obvious that the agent with only geometrical state variables (in blue) failed to achieve as good a policy as the agent with information about full

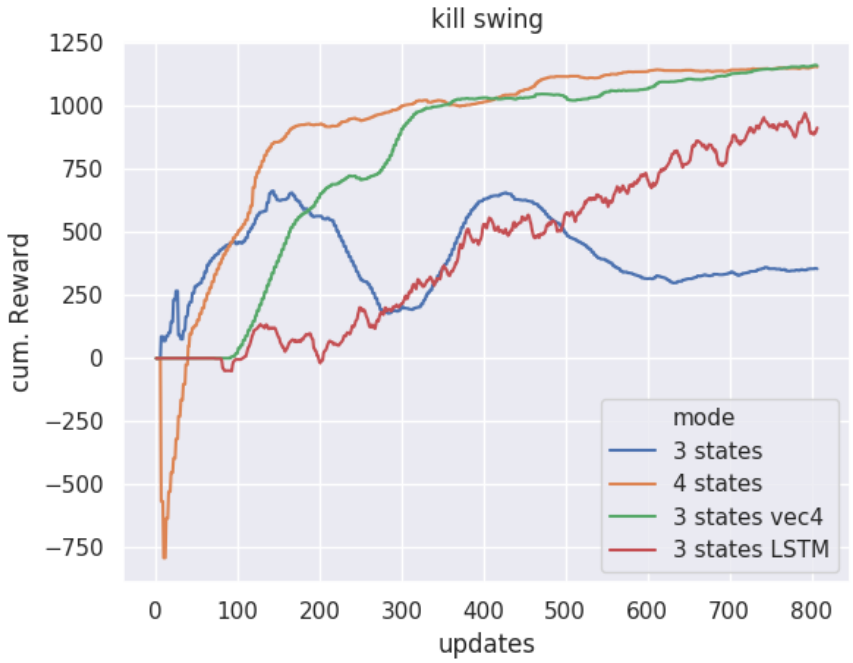


FIGURE 6.10: Reward curves of different methods of learning the stabilizing skill with partial observability.

states (in yellow). The stacking method fell slightly behind the full state agent by a number of episodes, a situation that was probably influenced by the increased number of parameters in the input layer. The recurrent version of PPO also indicated a promising result along a calm reward curve. Both approaches were able to infer the dynamics given the geometrical inputs. A recurrent network is more biologically plausible but less efficient than stacking observations.

6.5 Conclusion

In basic 2d, we discovered the possibility of decomposing a "2d" set-down into sub-skills with specific reward functions. Furthermore, it is of great effective if these sub-skills are related, and we found the policy of a more difficult sub-skill could be converged faster provided the initialization by the policy of an relevant easier sub-skill. In particular, the policy-based method was more efficient than the value-based method because of the better convergence property. Finally, we compared the progress of different methods in dealing with partial observability on solving the "kill" swing task. Both LSTM method and the frame stacking method were able to infer the missing information and provide fairly reasonable policies.

Chapter 7

Advanced 2D

7.1 Advanced 2D Environment

In contrast to basic 2D, the most significant upgrades under this environment were the vessel motions and the realistic ramp-up times of hoisting and slewing. Here, the motions of the barge and the crane vessel were concurrently affected by irregular waves in more DoFs and the change of mass distributions. The load accounts for a significant portion of the vessel weight, and any big movement of the load can lead to a change in the orientation of the vessel. Furthermore, the physical effects of colliding were also enabled in order to model the resultant motions of objects by impact forces. The environment resembled the most realistic features with regard to practical situations and was considered as the most difficult environment. We named it *advanced 2D*. We built advanced 2D in the Orcaflex GUI, which is one of the most used packages for the dynamic analysis of offshore structures.

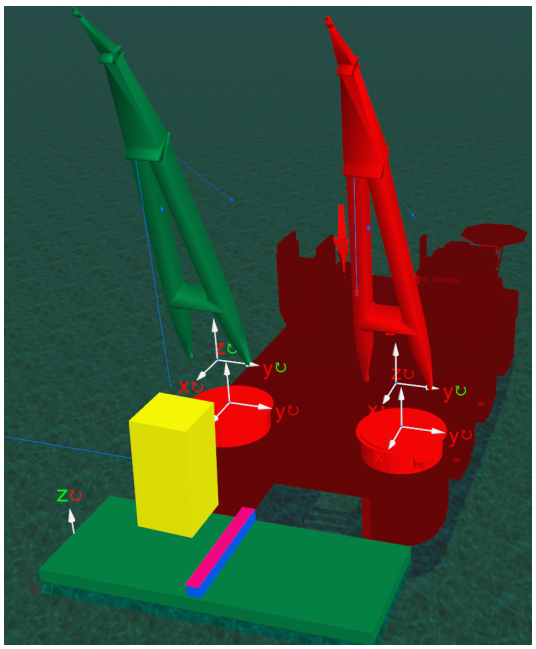


FIGURE 7.1: Visualization of Advanced 2D simulation environment.

Motion	Definition	Barge	Crane vessel
Surge	Translation along x	no	no
Sway	Translation along y	no	no
Heave	Translation along z	yes	yes
Roll	Rotation about x	no	yes
Pitch	Rotation about y	yes	yes
Yaw	Rotation about z	no	no

TABLE 7.1: Allowed motion in advanced 2D. Note that the definitions of motion follow the convention on the **local coordinate system** shown in Figure 7.2. Unidirectional wave will not cause the crane vessel's pitch, but it actually happens as soon as the load is all transferred to the barge.

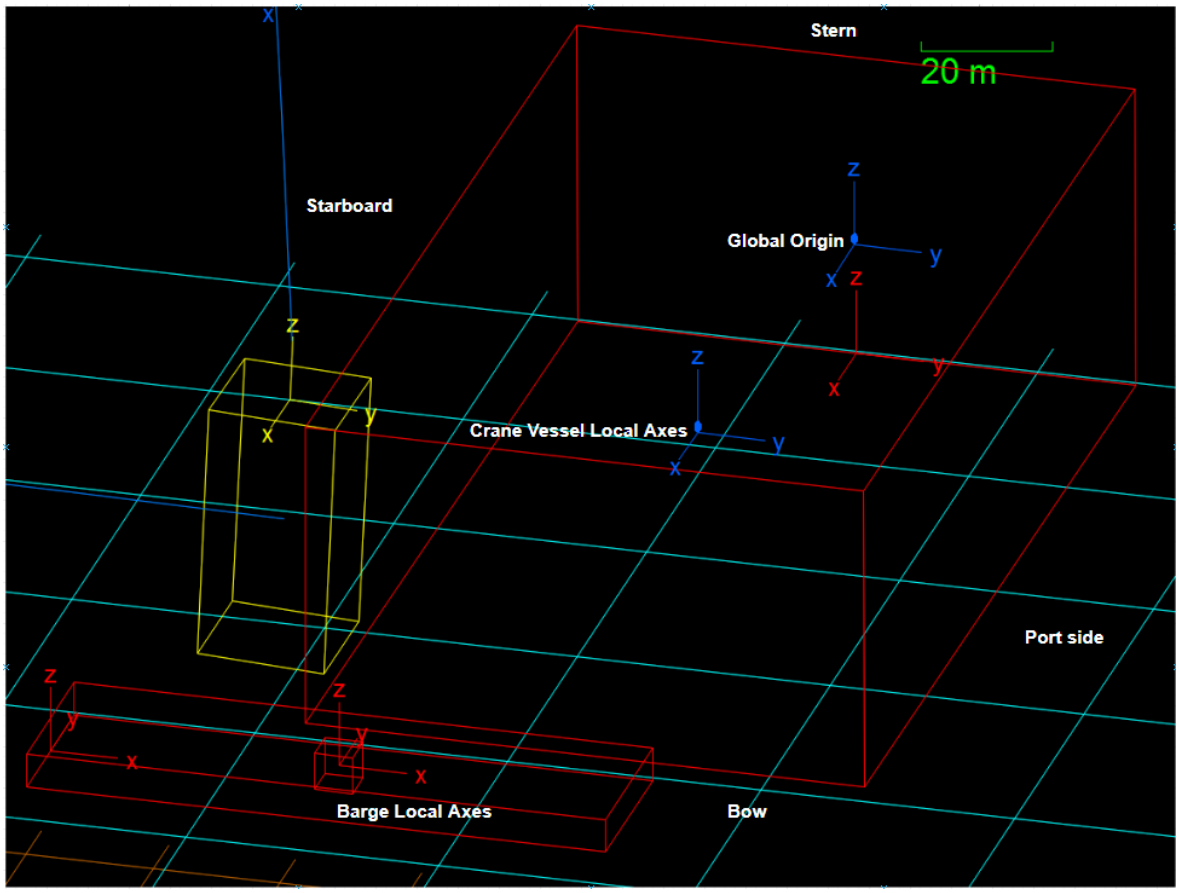


FIGURE 7.2: Local axes of barge and crane vessel.

The general visualization of advanced 2D is presented in Figure 7.1. The global axes are displayed in the upper right-hand corner of Figure 7.1, where the global origin is located at the center of the stern (refer to Figure 2.5) of the crane vessel. The global y-axis is pointing to the port side from the global origin. The global x-axis is pointing to bow of the vessel. The z-axis is straight up from the origin.

Regarding the sea state, in order to resemble only 2D motions in the y-z plane, we limited the wave to travel only unidirectionally along the global y-axis. This limit excluded the motions on translation on the global x-axis (surge) and rotation on the global y- and z-axes (pitch and yaw). A summary of allowed motions on vessels is presented in Table 7.1. Note that the motions are defined with respect to the vessel's local axes, and the local axes of the barge are rotated 90 degrees around the global z-axis. Therefore, the barge pitch is essentially the rotation about the x-axis in the global axes, which behaves as the roll of the crane vessel. Besides that, we assumed that the *crane tuggers* were working perfectly such that the load does not rotate around its own z-axis due to slewing. For advanced 2D, we used the same sea state to generate irregular waves throughout all the episodes. The sea states were carefully chosen to be Hs 1.5 and Tp 8s, a decision that follows the workability assessment methodology used by HMC projects.

The physical properties of the load and barge were unchanged throughout the whole

PoI	Abbr.	State
Load lower right	llr	y_{llr}, z_{llr} v_y^{llr}, v_z^{llr}
Load lower left	lll	y_{lll}, z_{lll}
Aux. hoist	hoi	v_y^{hoi}, y_{hoi}
Hoist wire	hw	ϕ_{hw}, T_{hw}
Aux. hook	ho	v_y^{ho}
Bumper upper left	buul	y_{buul}, z_{buul}
Bumper lower left	bull	y_{bull}, z_{bull} v_y^{bull}, v_z^{bull}
Slewing		v_{slew}
Hoisting		v_{hoist}
Barge CoG	bcog	ω_{bcog},θ_{bcog}

TABLE 7.2: State space of advanced 2D. Definition of components follows Figure 4.1. Note that ϕ stands for swing amplitude in the y-z plane as in basic 2D, θ_{bcog} stands for the pitch of the barge on its local axes and T_{hw} represents the instantaneous tension in the hoist wire.

Action	Definition	Ramp-up
Payout	increase hoisting	+0.025 m/s^2
Haul-in	reduce hoisting	-0.025 m/s^2
Slew-left	increase slewing	+0.3 deg/s^2
Slew-right	reduce slewing	-0.3 deg/s^2
Hold	do nothing	0

TABLE 7.3: Discrete action space of advanced 2D. Note that the range of the hoisting speed is [-0.15,+0.15] m/s, for slewing [-1.8,+1.8] deg/s.

experiment, which particularly ensured that the dynamics of the MDP would remain unchanged (see Table 7.4). However, the agent still has to deal with the changing of the vessel motion response due to the relative position of the load with respect to the crane and the slewing/hoisting actions. This largely increased the difficulty of the problem. For this particular weight of load, the auxiliary block was selected. The action spaces were the same as those in basic 2D. However, we included ramp-up time for reaching maximum speed, which fulfilled the property of the auxiliary block as used in advanced 1D (see Table 5.1). Contrary to advanced 1D, we no longer froze the action until it reached a certain payout speed; instead, new actions could be taken in every time step. The effect of actions is now adapted to increasing/decreasing the hoisting/slewing speed by a ramp-up acceleration (see Table 7.3). Regarding the slewing actions, the max slewing speed is 1.8 deg/s, and the ramp-up time is about 5s.

Similar to basic 2D, we fed the agent the coordinates and dynamics of the relevant PoIs in the y-z plane. We did not include all the possible dynamics (i.e., v_z^{lll} and v_y^{lll}) because we believed the agent would be able to infer that information based on simple geometrical transformations from provided states. Since barge motion is also affected by waves in advanced 2D, we fed the agent with the pitch θ and the angular velocity ω of the barge as well as the velocities of the bumper so as to support the agent in reducing the impact force. Moreover, we included the hoist wire tension T_{hw} as an indication of degree of completion of the set-down. In practice, the set-down can be considered completed only if the hoist wire is slack, and its tension is neglectable compared to the load weight. The detail of input state variables is presented in 7.2.

Setting	Value
Load mass	556t
Load size	20m (x) x 20m (y) x 40m (z)
Barge size	46m (x) x 120m (y) x 10m (z)
Barge critical damping ratio	8%
Barge stiffness	500e3
Bumper size	45m (x) x 5m (y) x 5m (z)
Bumper critical damping ratio	12%
Bumper stiffness	50e3
Hs	1.5
Tp	5
Initial y between hoist and barge CoG $y_{hoi} - y_{bcog}$	7.8m
Initial z between load and barge $z_{llr} - z_{bull}$	$17 \sim 20m$
Initial hoist wire length	75m
Initial bumper position	$-5m \sim 5m$ away from Barge CoG
Initial swing amplitude ϕ_{ah}	$-2.5 \sim +2.5$ deg
Timeout	250s
Time step	0.2s
Explicit inner time step	0.01s

TABLE 7.4: Environmental setting of advanced 2D.

Finally, we set up simulation episodes based on the Table 7.4. We fixed the relative positions between the barge and the crane vessel, as well as the hoist wire length for every episode; however, we provided a random initial swing amplitude and bumper position for different starting positions. The ultimate goal in advanced 2D was similar to that of basic 2D regarding the impact force and set-down precision from a initial swing amplitude. However, the difficulty increased significantly mainly because of the ramp-up and vessel responses on irregular waves as well as during the transportation of the load. HMC has a specific engineering design criteria for the maximum allowed force on bumpers with respect to the load weight. Therefore, special attention was paid to the impact force on the bumper because it is, in practice, more sensitive to impacts than barges are.

7.2 Skill Transfer

In basic 2D, we realized that it is of great effectiveness to decompose the set-down into interrelated sub-skills and start training from the simplest skill. In advanced 2D, we explored the feasibility of using the same methodology on more complex dynamics and a larger input state space. We therefore followed the same sub-skills definitions used in basic 2d (see Table 6.1 in Chapter 6) but with newly designed reward functions.

Goal	Reward
Stabilizing Easy	+1, if Eqn 7.3
Stabilizing Hard	+1, if Eqn 7.4
Aligning	$\min(1, \frac{1}{\lvert d \rvert})$, if Eqn 7.4
Approaching Easy	$\frac{17-h}{10}$, if Eqn 7.4 and $0 < d < 3$ $2 + d$, if $d < -2$ and $h < 7$
Approaching Hard	$\frac{17-h}{10}$, if Eqn 7.4 and $0 < d < 2$ $1.5 + d$, if $d < -1.5$ and $h < 7$

TABLE 7.5: TL for advanced 2D.

7.2.1 Design of Reward Functions

The reward functions for training sub-skills are presented in Table 7.5. Regarding stabilizing skills, we prepared two rewards with distinct difficulties in terms of the allowed swing amplitude and the horizontal velocity of the hook. We started training with easier rewards and then switched to harder ones. We defined the vertical distance between bumper lower left and load lower right by:

$$h = z_{llr} - z_{buul}. \tag{7.1}$$

Horizontal distance between bumper lower left and load lower right:

$$d = y_{buul} - y_{llr}. \tag{7.2}$$

Stabilized condition:

$$\text{Easy mode: } |\phi_{hw}| < 1.5 \text{ and } |v_y^{ho}| < 0.5, \tag{7.3}$$

$$\text{Hard mode: } |\phi_{hw}| < 1 \text{ and } |v_y^{ho}| < 0.2. \tag{7.4}$$

The results in basic 2d showed that continuous rewards are more effective in distance-control problems because of the information on the goodness of actions they provide. In advanced 2D, we calculated all positive rewards linearly with respect to either distance or height. Notably, for the approaching skill, we gave the agent negative rewards $2 - d$ whenever there was an overlap between the load bottom surface and bumper top surface because, in reality, the load was never allowed to step onto the bumper, and we punished the agent

for any potential situation that might make it happen. For every time step, we applied a constant penalty of -10 to any positive impact force measured by Orcaflex on the barge or the bumper. Moreover, we excluded the reward for the final set-down for now because all sub-skills before the set-down belong to the continuous control, whereas the set-down is an episodic task, and it requires a completely different focus on state variables, which could drastically change the policy.

7.2.2 Asynchronous Training

For training each sub-skill, we included asynchronous methods, an approach proposed by Mnih, Badia, et al. (2016) in the implementation of A3C. For each of the policy iterations, we initialized n workers by the policy of the previous iteration. We used them to run n simulations concurrently and stored all experiences along the trajectories of every worker. Then, we updated the policy by stochastic batch samplings over newly added experiences for every fixed amount of total steps. The beauty of trust region methods is that they allow multiple gradient updates within a single policy iteration. We used PPO for all the training in advanced 2D. The hyperparameters are displayed in Table 7.6.

Setting	**value**
Num worker	4
Learning rate	0.00003
Discount factor γ	0.99
Clipping factor ϵ	0.2
Trajectory length	1250
Num mini-batches	50
Num epochs	10
λ return	0.95
Num hidden layers/units	2/64
Activation	tanh
Optimizer	Adam

TABLE 7.6: Hyperparameters for PPO.

7.2.3 Results of Transfer Learning

We executed the training procedure following the order of reward functions in Table 7.5. The initial weights of the policy were initialized by the policy of previous skill. The learning curves of all sub-skills are presented in Figures 7.3, 7.4 and 7.5. New skills revealed positive learning progress based on the parent skills, although they seemed to struggle at the beginning due to the new reward function. Notably, for the stabilizing skill, the pre-training on the easier stabilizing task (equation 7.3) enabled the agent to achieve a higher starting point on the same problem with a more difficult reward function (equation 7.4) and also resulted in a high reward given for the same amount of training resources. For the approaching

skill, although the starting point was worse than the end of its pre-trained policy because of the new reward (see the position where the reward function was switched in Figure 7.5), it caught up quickly and eventually achieved significant improvements.

FIGURE 7.3: Reward curves of training the stabilizing skill. The agent in red was continuously rewarded by the strict reward function, whereas the agent in green started from the easier reward function and then switched to the strict one.

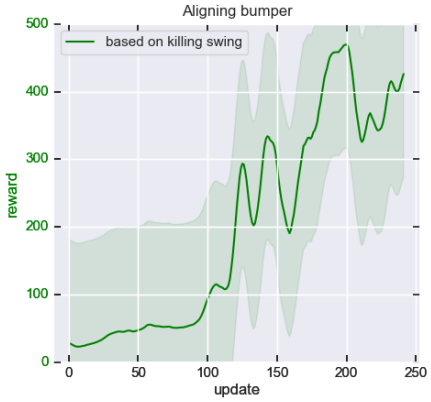


FIGURE 7.4: Learning curve of training the aligning skill.

The effect of the policy of the approaching skill is shown in Figure 7.6. We observed that the agent eventually stabilized the load at a very interesting position, where the load was just right below the top surface of the bumper and a small horizontal distance away from the bumper. The distance provided sufficient space to take actions in response to the bumper motion while still satisfying the reward function of the approaching skill. The vertical distance prevented the impact from the barge as well as the risk of stepping over the bumper. We tested the approaching policy for 500 episodes with a random initial position that lasted 250s. The agent, on average, was able to stay in the position for 146.6 ± 12.5s, as shown in Figure 7.6.

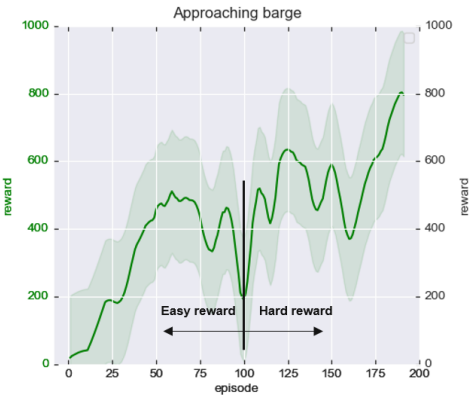


FIGURE 7.5: Learning curves of approaching skill.

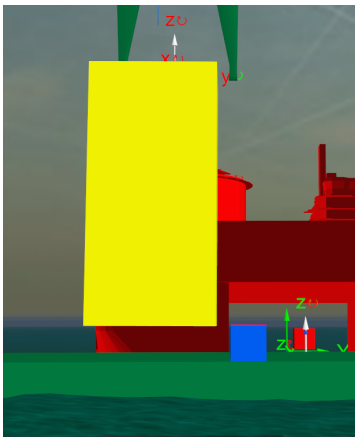


FIGURE 7.6: Stabilized position of using approaching skill.

7.3 Separate Set-Down

As illustrated in basic 1D, it would be more efficient to learn set-down from an easier starting position than by using end-to-end, especially as we have already achieved a good approaching policy that enables the agent to stabilize the load very near the barge and the bumper. Therefore, for the last part, we focused on training the agent to set down from the position shown in Figure 7.6 assuming the rest of the episode was perfectly controlled by the approaching skill.

7.3.1 Reward Functions

In advanced 2D, we defined the reward r^t for every time step by the summation of set-down reward r_s, barge impact penalty r_{Fy} and bumper impact penalty r_{Fz},

$$r^t = r_s^t + r_{Fy}^t + r_{Fz}^t. \tag{7.5}$$

The set-down is done only if the tension in the hoist wire nearly disappears such that the load weight is fully exerted on the barge. We therefore gave the positive reward on every time step when the set-down was completed until the episode ended. The reward is influenced by the set-down position d as defined by equation 7.2. The closer the load is to the bumper, the higher the step reward assigned to the agent becomes:

$$r_s = \begin{cases} \frac{1}{10} \cdot \min(1, \frac{1}{d}), & \text{if } 0 < d < 7 \text{ and } T_{hw} < 100, \\ 0, & \text{else.} \end{cases} \tag{7.6}$$

One of the challenges in using Orcaflex is that all simulations have a predefined length. To prevent the reward from increasing with earlier set-down attempts, we applied r_s only for 60 seconds after the set-down was completed and discarded the rest. It balanced the sum of positive step rewards on whatever set-down moments. Otherwise, the agent would always take risky actions to set down as early as possible to get a much higher sum of positive rewards. The special condition on hoist wire tension in equation 7.6 also encouraged the agent to avoid re-impacts. Because during a re-impact, hoist tension would increase again, the agent would no longer receive positive step rewards.

Let F_z denote the impact force on the barge along the z-axis. Since the load weight is 556t, for completing a set-down, the minimum F_z could not be smaller than 5,560kN. Thus, we set the margin at 6,000kN as the maximum allowed impact force on the barge. For every time step, any measured F_z higher than 6,000 was penalized (see equation 7.7).

$$r_z = \min(0, 6000 - F_z)/20, \tag{7.7}$$

$$r_y = \max(-10, min(500 - F_y, 0)/50), \tag{7.8}$$

$$r_y = 10 \min(0, 0.4 - (v_y^{lly} - v_y^{bull})). \tag{7.9}$$

Similarly, in order to prevent horizontal re-impacts, the engineering design criteria of bumpers at HMC indicates that the maximum impact force on bumper F_y should not be higher than 10% of the total load weight, which leads to the maximum allowed force (500kN) in the reward function for the bumper (see equation 7.8). In addition, the equivalent of avoiding a huge impact force on the bumper prevents a high relative velocity between the load and the bumper along the y-axis. We proposed an alternative reward function in terms of the relative velocity (see equation 7.9).

7.3.2 Auxiliary Task Loss

Although the lateral control is still a key to a good set-down, it does not directly contribute to the goal of the set-down. If we include the reward for the approaching skill in the reward function, the agent stops exploring new positive rewards because it has already satisfied the reward from the approaching skill, however, we also do not want to completely forget the approaching skill. A compromise is to use the auxiliary task loss, as proposed by Jaderberg et al. (2016) such that agent is asked to balance between episodic rewards and the rewards of the auxiliary tasks. Besides the policy gradient loss of PPO, we included a loss of an auxiliary task (see equation 7.10) associated with the value function for estimating the advantage of PPO:

$$L(\theta) = L^{\theta}_{pg} + L^{\theta}_{vr} + L^{\theta}_{v_{aux}}. \tag{7.10}$$

Specifically, L_{pg} is the ordinary clipping loss of PPO (see equation 3.21). L_{vr} is the TD error of the value function with respect to the n-step return of the true reward function. $L_{v_{aux}}$ denotes the TD error associated with the reward for auxiliary tasks. We used the reward function of the approaching skill (see last row in Table 7.5) as the return in the auxiliary task loss term. We initialized the policy by the weights of the approaching skill.

7.3.3 Results

We tested the effectiveness of adding the auxiliary task loss by plotting the distance between the load and bumper d after set-down was completed (see Figure 7.7). Because the policy was first initialized by the weights of approaching skill, the set-down distance did not show much difference between loss functions. However, as the training continued, the policy began to be influenced by the reward function, and the standard PPO loss failed to maintain the horizontal position of the load, which resulted in huge oscillations. Instead, the auxiliary task loss still controlled the changing of horizontal distance within a reasonable range throughout the whole training.

Then, we compared the policies based on different rewards with concern for the impact force on the bumper proposed in equations 7.8 and 7.9. As shown in Figure 7.8, neither of the two reward functions indicated significant improvement on reward; however, it is obvious that the reward associated with force was less stable than the reward based on relative velocity. It is logical in the sense that the force is dependent on all inputs, but is probably

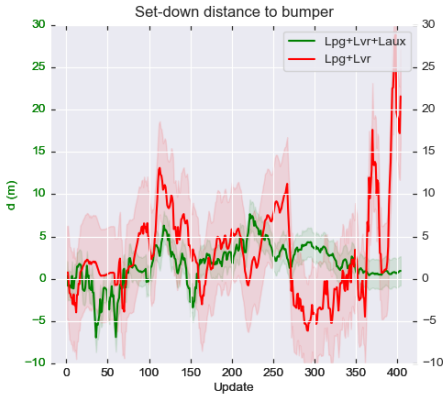


FIGURE 7.7: Horizontal distance between load and bumper along policy updates.

FIGURE 7.8: Reward curves of two distinct reward functions.

not directly indicated by any of them, whereas for the reward on the relative velocity, it can be derived directly from the input, which makes it easier to be understood by the agent.

Moreover, in Figures 7.9 and 7.10, we visualized the distribution of maximum impact forces on the barge (x-axis) and the bumper (y-axis) made by policies based on two reward functions. We noticed that, in Figure 7.9, most impact forces on the bumper satisfy the "10%" criterion, and the majority of the impact loads on barge is below 120% of the mass of the load. Compared with Figure 7.10, the reward based on the relative velocity produces much fewer outliers than the reward based on forces. This confirms again that the training for the set-down would be more effective if we replaced the real goal with a straightforward equivalent that can be easily accessed from input state variables.

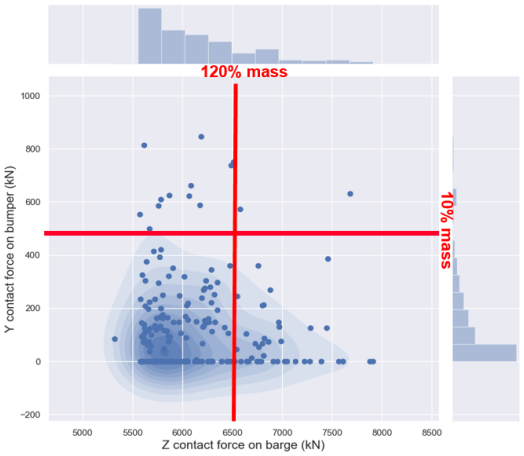


FIGURE 7.9: 200 episodes done with the policy rewarded by "high" relative velocity. The horizontal red line specifies the engineering limits on the impact force on bumper. The vertical red line shows the majority of the impact force on barge is below 120% of the load mass.

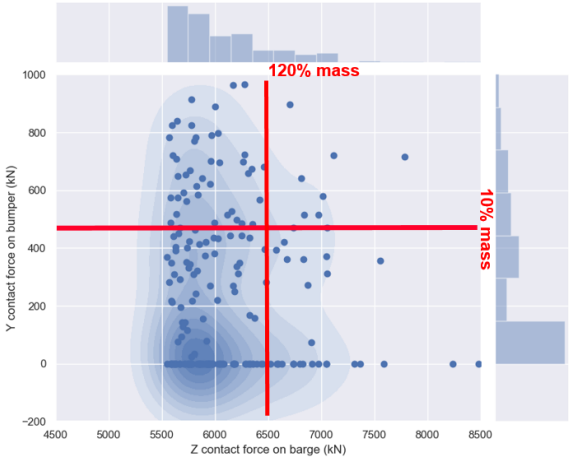


FIGURE 7.10: 200 episodes done with the policy rewarded by forces on the bumper.

Eventually, we investigated the general strategy that the agent had learned. Let 6,000

Imp. Velocity v_{z_0}	**$F_{z_{max}}$>6000 Kn**		**$F_{z_{max}}$<6000 Kn**		**Total Episodes**	
	"Monkey"	Agent	"Monkey"	Agent	"Monkey"	Agent
> 0.4	160	**30**	14	**2**	174	**32**
> 0.2 and < 0.4	14	**55**	9	**44**	23	**99**
< 0.2	1	**25**	2	**44**	3	**69**

TABLE 7.7: The number of "good" ($F < 6000Kn$) and "bad" ($F > 6000kN$) set-down attempts w.r.t. the vertical impact velocity on the barge for 200 episodes done by a continuous payout policy and agent with "set-down" skill.

kN be the threshold for classifying the quality of the set-down. Define the impact velocity as the vertical relative velocity between two PoIs from the load and the barge, respectively, that first contact v_{z0}. There is a correlation between the impact velocity and the maximum impact force $F_{z_{max}}$ (see Table 7.7). The higher the first impact velocity is, the higher the chance a set-down with larger impact force will happen. With that regard, as presented in Table 7.7, the agent significantly reduced the number of higher first vertical impact velocities, which implicitly brings a better set-down moment with respect to the relative motion. It echoes the conclusion drawn from the results in the 1D environment.

FIGURE 7.11: Distribution of the set-down position w.r.t. the barge impact load for 200 testing episodes.

Regarding the distance to the bumper, the "set-down" skill managed to keep the distance within 0.5 and 0.8 meters on average (see Figure 7.11).

7.4 Final Model

From any initial position, the "approaching" skill worked very well to transport and stabilize the load at the position used as the starting position for the learning of the "set-down" skill. Therefore, regarding the final model for solving the end-to-end set-down problem as defined in Table 7.4, we decided to execute a "hard switch" between the "approaching" skill and the "set-down" skill at the position where the load is 5 meters above the barge. In a more

general setting, this switching moment can be optimized using hierarchical RL. Given the time constraint on this project, we continued with this "hard switch". We further tested the agent for 200 episodes from 17m with the random initial swing, and we achieved the very similar result as the exclusive "set-down" skill (see Table 7.8). This indicates that the "approaching" skill is very effective at transferring the load from any initial position to the position where the "set-down" skill is familiar to carry on.

Imp. Velocity v_{z_0}	$F_{z_{max}}$**>6000 Kn**	$F_{z_{max}}$**<6000 kn**	**Total Episodes**
> 0.4	27	3	30
> 0.2 and < 0.4	59	44	**103**
< 0.2	22	45	**67**

TABLE 7.8: The number of "good" ($F < 6000Kn$) and "bad" ($F > 6000kN$) set-down attempts w.r.t. the vertical impact velocity on the barge for 200 end-to-end episodes with initial swing by the agent using the "hard switch" plan.

Chapter 8

Discussion and Conclusion

8.1 Conclusion

In this project, we investigated the feasibility of solving the practical offshore set-down problem using RL techniques. As a feasibility study, we started from the simplest possible environment where only the heave motion and impact velocity are considered. Then, we gradually upgraded the set-down environment on top of the previous one by adding carefully chosen features referring to the realistic set-down operation. The results under different environments bring us an overview of the possibilities and limitations of standard RL algorithms. We demonstrated that the methods suffer from the general challenges of RL, such as sparse rewards and sample efficiency in solving the long-term objective set-down problem. We tried various methods to work around this issue, such as transfer learning, hierarchical RL, and shaping rewards. Moreover, the results of using searching methods and recurrent neural networks provided us the option to include model-based methods with a deeper network topology.

8.2 Answers to Research Questions

1.What are the main factors and limits that form the offshore operating environment? How should we model and extend those limits for the simulation environment of Reinforcement Learning?

In an ideal situation, the offshore environment is mostly determined by the sea states and corresponding ship motions. The link between waves and the resultant motion is the RAO of a vessel. In this project, we assumed that irregular waves are unidirectional along the vessel, and the sea states remain unchanged. We used the JONSWAP spectrum to generate the wave energy spectra for the chosen sea state regarding the workability assessment at HMC. We transferred the wave spectra into the motion response spectrum by the RAO of an HMC vessel. Finally, we executed the inverse Fourier transfer with a random phase angle to generate action ship motion in the time domain. For 1D environments, the ship motion is only limited to 1 DoF (heave). For 2D environments, we included sway, heave and pitch because they all effect in the 2D plane.

2.What is the metric and how should we determine the quality of the set-down operation in terms of physical measurements?

In practice, the important criteria of the set-down are impact force and set-down precision. The impact force describes the maximum force applied to the barge as well as the bumper during the set-down operation. That maximum usually occurs when a load contacts the barge or the bumper. In 1D environments, the impact forces are represented by the impact velocity, which stands for the relative velocity between the load and the barge right before they contact. The key challenge of the project was to find the proper solutions to reduce the impact velocity. Furthermore, in 2D environments, we also considered the set-down precision, which is essentially a measurement of the distance between the load and the bumper. In reality, it is extremely important, especially when the barge deck is full and the margin is quite limited.

3. How should we deal with the sparse/delayed reward in each of the set-down simulations?

In this project, we addressed this issue mainly by TL and shaping rewards. Sparse and delayed rewards are, in general, the most challenging part of this study. The set-down is mostly long-term and objective-oriented. However, it does require multiple skills during the set-down, but they are not always correlated. We made clear distinctions between the focuses and skills at different stages of the set-down. We trained every sub-skill separately by shaping rewards. The shaping reward helps to decompose a long-term objective into unit rewards associated to every action. Furthermore, we found that it was quite effective to execute TL between multiple skills because many of the sub-skills are interrelated. There was also a possibility to implement hierarchical RL, which is useful for more separated sub-skills with clear distinctions between each other.

4.How can the performance of the agent be improved by learning with Monte Carlo tree search? Monte Carlo tree search (MCTS) can be very powerful with the combination of a valid roll-out policy. We experimented with MCTS mainly in the 1D environment where the simulator provides the transitions of system dynamics. We used standard RL to train a roll-out policy and executed MCTS that runs multiple simulations using the roll-out policy. It leads to great improvements even without a large number of simulations. We further tested the feasibility of imitation learning based on the trajectories of MCTS, but it did not provide promising results. This is probably due to the inconsistent policy of MCTS. The actions indicated by MCTS are not fully dependent on states but also on the results of simulations. However, the goal of imitation learning is to map the observed state to the expert's action. Therefore, it is less effective to consider MCTS as the source of demonstration data under this setting.

5.To what extent can the simulation environment be upgraded toward the real-world and what is the effective way to deal with partial observability in the environment? The simulation environment can be upgraded by increasing the complexity of object motions

and action spaces. In this project, we upgraded the environment starting from unidirectional to 3 DoFs and then upgraded the action space by adding practical constraints, such as ramp-up time and action duration. However, there is still a huge gap between the simulation and a real crane operator cabin, which can be investigated further. In addition, we tested the feasibility of inferring the dynamics of a partially observable environment for a simple stabilizing problem. Both frame stacking and recurrent function approximations are workable, which left open the option to test frames of images as the input instead of fully relying on time-series sensor data.

8.3 Further Research

The first suggestion for the future work relates to the design of reward functions. The set-down is generally a long-term, objective problem on which we have spent a lot of time trying different shaping rewards, but they are not guaranteed to be optimal. It would be more effective to have the agent itself learn the reward function from interactions with the environment. Possible techniques could include Inverse RL. Second, in the project, only model-free algorithms were tested. Given the simulator of the environment, it would be possible to learn the transition probability between states in order to improve the approximation of the value function by roll-outs. Third, for the input state variables, we could combine the images from different perspectives for a better explanation of the geometrical information of PoIs. Then, deep networks could be considered for use. Finally, the environment could be upgraded further by including the motions of the remaining DoFs, and we could use different shapes of bumpers for training in order to improve the generalization of the learned policy.

Bibliography

Arulkumaran, K. et al. (2017). "A brief survey of deep reinforcement learning". In: *arXiv preprint arXiv:1708.05866*.

Asada, M. et al. (1994). "Vision-based behavior acquisition for a shooting robot by using a reinforcement learning". In: *Proc. of IAPR/IEEE Workshop on Visual Behaviors*, pp. 112–118.

Atkeson, C. G. and Santamaria, J. C. (1997). "A comparison of direct and model-based reinforcement learning". In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE, pp. 3557–3564.

Bacon, P.-L., Harb, J., and Precup, D. (2017). "The Option-Critic Architecture." In: *AAAI*, pp. 1726–1734.

Browne, C. B. et al. (2012). "A survey of Monte Carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Dayan, P. and Hinton, G. E. (1993). "Feudal reinforcement learning". In: *Advances in neural information processing systems*, pp. 271–278.

Dietterich, T. G. (1998). "The MAXQ Method for Hierarchical Reinforcement Learning". In: *Proc. of the fifteenth international conference on machine learning*, pp. 118–126.

Guo, X. et al. (2014). "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning". In: *Advances in neural information processing systems*, pp. 3338–3346.

Hester, T. et al. (2017). "Learning from Demonstrations for Real World Reinforcement Learning". In: *arXiv preprint*, 1704.03732v1 [cs.AI].

Hochreiter, S. and Schmidhuber, J. (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). "Reinforcement learning algorithm for partially observable Markov decision problems". In: *Advances in neural information processing systems*, pp. 345–352.

Jaderberg, M. et al. (2016). "Reinforcement learning with unsupervised auxiliary tasks". In: *arXiv preprint arXiv:1611.05397*.

Journée, J., Massie, W., and Huijsmans, R. (2000). *Offshore Hydromechanics: Course OE4630*. TU Delft.

Kakade, S. and Langford, J. (2002). "Approximately optimal approximate reinforcement learning". In: *ICML*. Vol. 2, pp. 267–274.

Kocsis, L. and Szepesvári, C. (2006). "Bandit based Monte-Carlo planning". In: *European conference on machine learning*. Springer, pp. 282–293.

Konda, V. R. and Tsitsiklis, J. N. (2000). "Actor-critic algorithms". In: *Advances in neural information processing systems*, pp. 1008–1014.

Kulkarni, T. D. et al. (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*, pp. 3675–3683.

Lazaric, A. (2008). "Knowledge transfer in reinforcement learning". PhD thesis. Politecnico di Milano.

LeCun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lin, L.-J. (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8.3-4, pp. 293–321.

Mnih, V., Badia, A. P., et al. (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518, pp. 529–533.

Mouhandiz, A.-A. el and Troost, S. (2013). "Subsea Templates Installation at North Sea using HLV THIALF". In: *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers.

Reedy, M. A. el (2012). *Offshore structures: design, construction and maintenance*. Gulf Professional Publishing.

Ross, S. and Bagnell, D. (2010). "Efficient reductions for imitation learning". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668.

Ross, S., Gordon, G., and Bagnell, D. (2011). "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635.

Schaul, T. et al. (2015). "Prioritized Experience Replay". In: *arXiv preprint arXiv:1511.05952*.

Schempf, J. F. (2007). *Pioneering offshore: the early years*. PennWell Custom Pub.

Schulman, J., Levine, S., et al. (2015). "Trust region policy optimization". In: *International Conference on Machine Learning*, pp. 1889–1897.

Schulman, J., Moritz, P., et al. (2015). "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., et al. (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Selfridge, O. G., Sutton, R. S., and Barto, A. G. (1985). "Training and Tracking in Robotics." In: *IJCAI*, pp. 670–672.

Silver, D., Huang, A., et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–489.

Silver, D., Hubert, T., et al. (2017). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815*.

Silver, D., Schrittwieser, J., et al. (2017). "Mastering the game of Go without human knowledge". In: *Nature* 550, pp. 354–359.

Silver, D., Sutton, R. S., and Müller, M. (2008). "Sample-based learning and search with permanent and transient memories". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 968–975.

Sutton, R. S. (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

— (1996). "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems*, pp. 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

Sutton, R. S., McAllester, D. A., et al. (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pp. 1057–1063.

Sutton, R. S., Precup, D., and Singh, S. (1999). *Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning*. Tech. rep., pp. 181–211.

Taylor, M. E. and Stone, P. (2009). "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *Journal of Machine Learning Research* 10, pp. 1633–1685.

Tessler, C. et al. (2017). "A Deep Hierarchical Approach to Lifelong Learning in Minecraft." In: *AAAI*. Vol. 3, p. 6.

Torrey, L. and Shavlik, J. (2009). "Transfer Learning". In: *Machine Learning*, pp. 1–22.

van Hasselt, H. (2010). "Double Q-learning". In: *Advances in Neural Information Processing Systems*, pp. 2613–2621.

van Hasselt, H., Guez, A., and Silver, D. (2016). "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 2. Phoenix, AZ, p. 5.

van Hasselt, H. and Wiering, M. A. (2007). "Reinforcement learning in continuous action spaces". In: *IEEE Internation Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 272–279.

Vezhnevets, A. S. et al. (2017). "Feudal networks for hierarchical reinforcement learning". In: *arXiv preprint arXiv:1703.01161*.

Wang, Z. et al. (2015). "Dueling network architectures for deep reinforcement learning". In: *arXiv preprint arXiv:1511.06581*.

Watkins, C. J. and Dayan, P. (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292.

Wiering, M. A. (1999). "Explorations in efficient reinforcement learning". PhD thesis. University of Amsterdam.

— (2005). "QV (lambda)-learning: A new on-policy reinforcement learning algrithm". In: *Proceedings of the 7th European Workshop on Reinforcement Learning*, pp. 17–18.

Wiering, M. A. and Schmidhuber, J. (1997). "HQ-learning". In: *Adaptive Behavior* 6.2, pp. 219–246.

Wiering, M. A. and van Hasselt, H. (2009). "The QV family compared to other reinforcement learning algorithms". In: *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*. IEEE, pp. 101–108.

Wiering, M. A. and van Otterlo, M. (2012). "Reinforcement learning". In: *Adaptation, learning, and optimization* 12, p. 51.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4, pp. 229–256.

Appendix A

Pseudocodes

A.1 Double Q-Learning

Double Q-learning

Initialize: Replay Memory D, two action-value functions $Q^{(\theta)}, Q^{(\theta^-)}$ with initial weights $\theta = \theta^-$
Initialize: Replay Memory capacity N_r, batch size N_b, update target frequency N
for episode $i = 0$ to M **do**
  for t=1,T **do**
    With probability ϵ select random action a_t
    Otherwise $a_t = \underset{a}{\text{argmax}}\, Q(s_t, a; \theta)$
    Take a_t and receive reward r_t and enter s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random N_b transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j, & \text{if episode ends at } j+1 \\ r_j + \gamma Q(s_{j+1}, \max\limits_{a'} Q(s_{j+1}, a'; \theta); \theta^-), & \text{otherwise} \end{cases}$
    Execute a gradient descent step w.r.t. θ on $\mathbb{E}[(y_j - Q(s_j, a_j; \theta))^2]$
    For every N steps, reset $\theta = \theta^-$
  end for
end for

A.2 Dataset Aggregation

DAGGER

Initialize: trajectory data-set $D \leftarrow \emptyset$, expert policy π^*
Initialize: student policy $\hat{\pi}(s;\theta)$
for Iteration $i = 0$ to N **do**
  for $t = 0$ to T **do**
    With probability β select action by student $a_t = \hat{\pi}_i(s_t;\theta)$
    Otherwise action selected from expert $a_t = \pi^*(s_t)$
    Aggregate data-set by expert's action: $D \leftarrow D \cup (s_t, \pi^*(s_t))$
  end for
  Gradient updates w.r.t. θ to train $\hat{\pi}_{i+1}$ on D by supervised learning
end for

A.3 Deep Q-learning from Demonstration

DQFD

Initialize: Expert data-set buffer D_E, agent replay buffer D_{replay}
Initialize: Two action-value functions $Q^{(\theta)}, Q^{(\theta^-)}$ with initial weights $\theta = \theta^-$
Initialize: Pre-training gradient updates K
Initialize: Batch size N_b, update target frequency N
for pre-training step $k = 0$ to K **do**
  Sample a mini-batch of N_b expert transitions $(s_j, a_j.r_j, s_{J+1})$ from D_E
  compute J_{DQ} as y_j in Double Q-learning
  compute $J_E = \max_a[Q(s_j, a; \theta)] - Q(s, a_j; \theta)$
  compute a gradient descent step on $J_{DQ} + \lambda_1 J_E + \lambda_2 J_{l2-norm}$ w.r.t. θ
  if k **mod** N=0 **then** $\theta = \theta^-$
  end if
end for
for episode $i = 0$ to M **do**
  for t=1,T **do**
    With probability ϵ select random action a_t
    Otherwise $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$
    Take a_t and receive reward r_t and enter s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D_{replay}
    Sample $0.8 \cdot N_b$ transitions (s_j, a_j, r_j, s_{j+1}) from D_{replay}
    Sample $0.2 \cdot N_b$ transitions $(s_j^e, a_j^e, r_j^e, s_{j+1}^e)$ from D_E
    compute J_{DQ} as y_j in Double Q-learning
    Set $l(a_j^e, a) = \begin{cases} 0, & \text{if } a_j^e = a \\ 0.8, & \text{otherwise} \end{cases}$
    compute $J_E = \max_a[Q(s_j^e, a; \theta) - l(a_j^e, a)] - Q(s, a_j^e; \theta)$
    compute a gradient descent step on $J_{DQ} + \lambda_1 J_E + \lambda_2 J_{l2-norm}$ w.r.t. θ
    For every N steps, reset $\theta = \theta^-$
  end for
end for

A.4 Generalized Advantage Estimator

$$\hat{A}_t^{(1)} := \delta_t^{(V)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(2)} := \delta_t^{(V)} + \gamma\delta_{t+1}^{(V)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$
$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^{(V)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t)$$
$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^{(V)} = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$$

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(2)} + ...) \\
&= (1-\lambda)(\delta_t^{(V)} + \lambda(\delta_t^{(V)} + \gamma\delta_{t+1}^{(V)}) + \lambda^2(\delta_t^{(V)} + \gamma\delta_{t+1}^{(V)} + \gamma^2\delta_{t+2}^{(V)}) + ...) \\
&= (1-\lambda)(\delta_t^{(V)}(1 + \lambda + \lambda^2 + ...) + \gamma\delta_{t+1}^{(V)}(1 + \lambda + \lambda^2 + \lambda^3 ...) + ...) \\
&= (1-\lambda)(\delta_t^{(V)}(\frac{1}{1-\lambda}) + \gamma\delta_{t+1}^{(V)}(\frac{1}{1-\lambda}) + \gamma^2\delta_{t+2}^{(V)}(\frac{1}{1-\lambda}) + ...) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^{V}
\end{aligned}$$

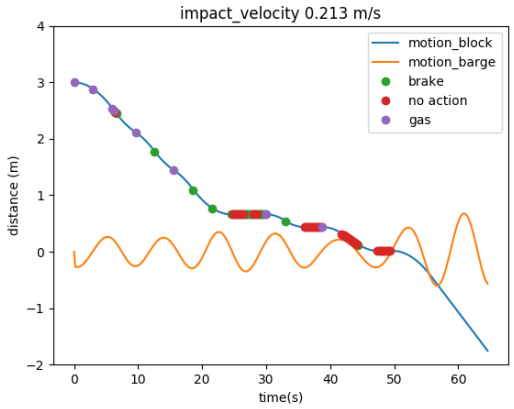
A.5 Proximal Policy Optimization

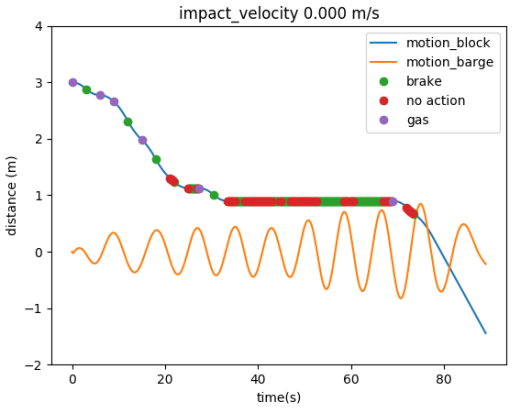
PPO

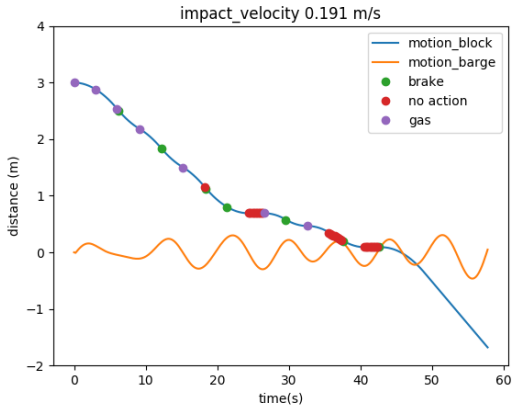
Initialize: Trajectory length of single environment per gradient update T, number of epochs per gradient update K, number of actors N.
Initialize: Replay Memory D with capacity $N * T$, value function $V(s;\theta)$, policy π with parameter θ, minibatch size $M < NT$
for iteration $i = 0$ to ∞ **do**
  for $n = 1$ to N **do**
    for $t = 1$ to T **do**
      Run π_θ for T steps, and store transition $(s_t, a_t, r_t, s_{t+1}, \pi(s_t;\theta))$ in D
    end for
    for $t = 1$ to T **do**
      Compute $\hat{A}_t^{GAE(\gamma,\lambda)}$ and $V_t(s;\theta)$, add them to the entry of t in D
    end for
  end for
  for $k = 1$ to K **do**
    Sample a mini-batch of M transitions $(s_j, a_j, r_j, s_{j+1}, \pi(s_j;\theta), \hat{A}_j^{GAE(\gamma,\lambda)}, V_j(s))$
    Let $k_j = [\frac{\pi_\theta(a_j|s_j)}{\pi_{\theta_{old}}(a_j|s_j)}]$
    Let $L_j^{CLIP} = \hat{\mathbb{E}}_j[min(k_j(\theta)\hat{A}_j^{GAE(\gamma,\lambda)}, clip(k_j(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_j^{GAE(\gamma,\lambda)})]$
    Compute surrogate loss $L = \hat{\mathbb{E}}_j[L_j^{CLIP} - c_1 L_j^{VF}]$, where $L_j^{VF} = \delta_j^{(V(s_j;\theta))}$
    Execute a gradient descent step w.r.t. θ on surrogate L
  end for
  Reset $D \leftarrow \emptyset$
end for

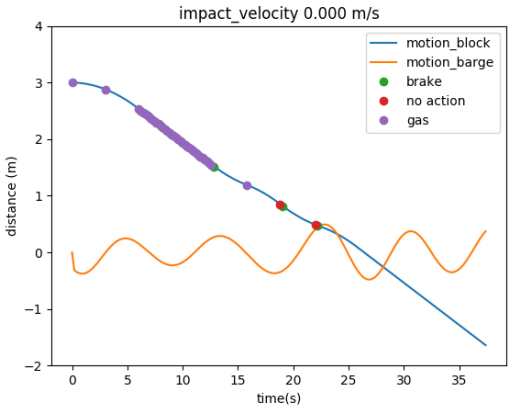
Appendix B

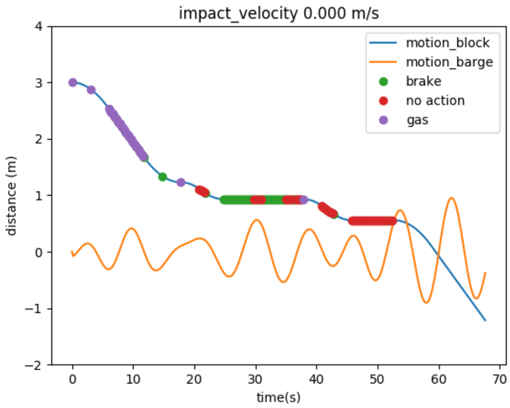
More Examples of Bad Attempts in Advanced 1D

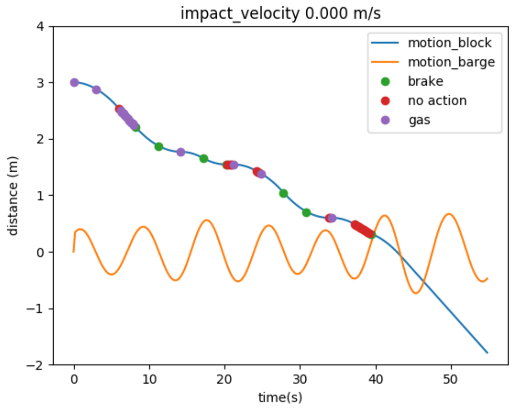


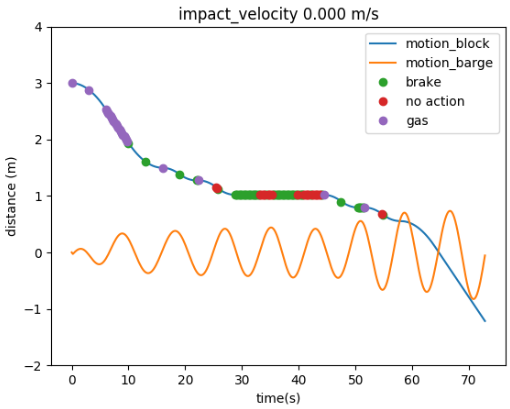


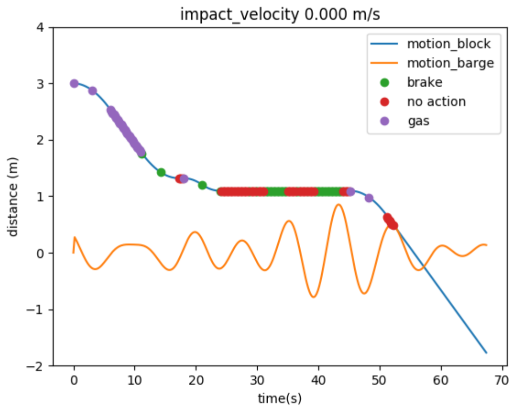





impact_velocity 0.000 m/s
motion_block
motion_barge
brake
no action
gas
distance (m)
time(s)


impact_velocity 0.000 m/s
motion_block
motion_barge
brake
no action
gas
distance (m)
time(s)


impact_velocity 0.000 m/s
motion_block
motion_barge
brake
no action
gas
distance (m)
time(s)


impact_velocity 0.000 m/s
motion_block
motion_barge
brake
no action
gas
distance (m)
time(s)