ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A GENERALIZED DEEP REINFORCEMENT LEARNING BASED CONTROLLER FOR HEADING KEEPING IN WAVES

M.Sc. THESIS

Afşin Baran BAYEZİT

Department of Ship Building and Ocean Engineering

Offshore Engineering Programme

JUNE 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A GENERALIZED DEEP REINFORCEMENT LEARNING BASED CONTROLLER FOR HEADING KEEPING IN WAVES

M.Sc. THESIS

Afşin Baran BAYEZİT
(508191229)

Department of Ship Building and Ocean Engineering

Offshore Engineering Programme

Thesis Advisor: Assoc. Dr. Ömer Kemal KINACI

JUNE 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DALGALI ORTAMDA YÖN TUTMA PROBLEMİ İÇİN GELİŞTİRİLMİŞ DERİN TAKVİYELİ ÖĞRENME TABANLI BİR KONTROLCÜ

YÜKSEK LİSANS TEZİ

Afşin Baran BAYEZİT
(508191229)

Gemi ve Deniz Teknolojisi Mühendisliği Anabilim Dalı

Açık Deniz Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ömer Kemal KINACI

HAZİRAN 2022

Afşin Baran Bayezit, a M.Sc. student of ITU Graduate School student ID 508191229 successfully defended the thesis entitled "A GENERALIZED DEEP REINFORCEMENT LEARNING BASED CONTROLLER FOR HEADING KEEPING IN WAVES", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Dr. Ömer Kemal KINACI** Istanbul Technical University	.....
Jury Members :	**Asst. Prof. İsmail BAYEZİT** Istanbul Technical University	.....
	Asst. Prof. Ferdi ÇAKICI Yıldız Technical University	.....

Date of Submission : 03 June 2022
Date of Defense : 21 June 2022

To my family

FOREWORD

Firstly, I would like to thank my advisor, Dr. Ömer Kemal Kınacı for his support not only for my thesis but in general for everything to do with my Master's program. Due to my undergraduate background in a relatively unrelated field to the marine scene, I have encountered many issues throughout my Master's program. I was able to able to overcome my shortcomings thanks to his help.

I am grateful to Dr. İsmail Bayezit, Rahman Bitirgen, and all the other members of the Model-Based Design and Control Laboratory of ITU for providing me with guidance, extra workspace, and extra hardware throughout my Master's education.

I also wish to thank the Maritime Research Institute of Netherland (MARIN) for providing me with an internship opportunity. The hardware and software they provided made this thesis possible. I am particularly grateful to Bülent Düz, Douwe Rijpkema, and Bart Mak of MARIN for their continuous support throughout my internship.

Finally, I thank my family for their continuous support and encouragement throughout my Master's education.

June 2022

Afşin Baran BAYEZİT
(Offshore Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

RL	**:** Reinforcement Learning
MDP	**:** Markov Decision Process
MC	**:** Monte Carlo
TD	**:** Temporal Difference
SAC	**:** Soft Actor-Critic
LQR	**:** Linear Quadratic Regulator
MAE	**:** Mean Absolute Error
RPM	**:** Rotation per Minute
SS	**:** Sea State
Hs	**:** Wave Height
Tp	**:** Wave Period
Dp	**:** Wave Direction

SYMBOLS

R	: Reward or R weight matrix of LQR
A	: Action
S	: State
G	: Gain
γ	: Discount rate
$\mathbb{E}[x]$	: Expected value of x
s'	: Next state
r'	: Next reward
π	: Policy
π_*	: Optimal policy
v	: State-value function
v_*	: Optimal state-value function
q	: Action-value function
q_*	: Optimal action-value function
Q	: Action-Value estimate or Q weight matrix of LQR
V	: State-Value estimate
δ	: TD error or rudder
$\dot{\delta}$	: Rudder rate
Δ	: Gradient
α	: Learning rate or tradeoff coefficient for SAC
ψ	: Yaw/Heading
ψ_{err}	: Yaw error
$\dot{\psi}$	: Yaw rate
$\ddot{\psi}$	: Yaw acceleration

LIST OF TABLES

LIST OF FIGURES

A GENERALIZED DEEP REINFORCEMENT LEARNING BASED CONTROLLER FOR HEADING KEEPING IN WAVES

SUMMARY

Reinforcement Learning (RL) is a machine learning method where a learner (the agent) tries to maximize a reward by learning how to act under different environmental circumstances. The agent looks at the state of its environment (through the state vector), takes an action, and then gets a reward and the next state of its environment. The agent improves its action-taking strategy (policy) with every action it experiments with.

RL methods have been used for many decision-making problems including control problems with promising results. Unlike many traditional control methods, a model-free RL doesn't need any environment dynamics to operate. This is especially beneficial for problems where the model dynamics are non-linear or not well-known. However, classical controllers are still the most used method of control for maritime applications.

Heading-keeping is a maritime control problem where a controller's objective is to keep the heading (yaw) angle of a vehicle constant. Generally speaking, the industry standard is to use traditional feedback controllers such as PID for this problem.

This study focuses on designing a generalized RL controller for the heading-keeping problem in waves. The study compares the designed RL controller to a traditional controller in terms of yaw error and rudder usage and observes that the designed RL-based controller performs better than the used traditional controller.

The first iterations of the RL agent had many issues. Unlike traditional controllers, the RL agents don't inherently recognize that in an idealized environment they can deal with waves coming from 0 and 180 degrees with almost zero rudder usage. On top of that, the first few developed agents had problems with excessive rudder usage, steady-state error, and overshooting behavior. All of these problems have been solved in the final iteration of the RL agent. Instead of just explaining the final agent, the thesis starts off with a weak RL agent and explains how it can be improved iteratively. This way the thesis explains how one might approach the problem of developing an RL-based controller.

The first section focuses on giving a rough summary of RL and the problem case, explains the purpose of the thesis, then talks about previous work over marine movement control in literature. Some detailed information about the used tools and simulation environment is also given here.

The second section introduces LQR controllers and designs an LQR controller for the heading keeping problem.

The third section explains RL in-depth to lay the foundation for the upcoming sections.

The fourth section starts with a naively designed simple RL agent and iteratively improves it. In each iteration of development, the agent is compared to the designed LQR controller, its weaknesses are analyzed, and the improvements for the next iteration are determined.

The fifth section summarizes the previous sections, explains the contributions of the thesis, and discusses possible future work.

DALGALI ORTAMDA YÖN TUTMA PROBLEMİ İÇİN GELİŞTİRİLMİŞ DERİN TAKVİYELİ ÖĞRENME TABANLI BİR KONTROLCÜ

ÖZET

Takviyeli Öğrenme (Reinforcement Learning ya da RL), bir öğrenicinin (agent) bir ödül (reward) fonksiyonunu maksimize etmek için hangi ortam koşulları altında nasıl eylemlerde (action) bulunması gerektiğini anlamaya çalıştığı bir makine öğrenimi yöntemidir. Agent, içinde bulunduğu ortamın çeşitli özelliklerini (state vector ya da durum vektörü üzerinden) gözlemler, bir eylemde bulunur ve ardından ortamdan bir ödül ve yeni ortam koşullarını alır. Denediği her eylemle agent eylem alma stratejisini (policy ya da politikasını) geliştirir. İyi tasarlanmış bir agent, yeterince eğitildiği taktirde iyi performans gösteren bir policy'ye yakınsayacaktır.

RL ve Derin RL (DRL) yöntemleri, birçok karar verme problemi için başarıyla kullanılmıştır. Örneğin RL agent'ları, Atari, satranç ve Go gibi bir çok oyunda insan üstü performans gösterebilmektedir. Bunun yanı sıra RL, robotik, finans ve hatta analog entegre devre tasarımı gibi çeşitli gerçek hayat uygulamalarında da oldukça umut vaadeden sonuçlar göstermiştir.

RL'in kullanılabileceği gerçek hayat uygulamalarından biri de kontroldür. Modelsiz RL (Model-Free RL), birçok geleneksel kontrol yönteminin aksine, herhangi bir ortam dinamiğine ihtiyaç duymadan çalışmaktadır. Model-Free RL'in bu özelliği, ortam dinamiklerinin lineer olmadığı veya iyi bilinmediği problemler için oldukça faydalıdır. Fakat sektörde denizcilik uygulamaları için hala en çok PID gibi klasik geri bildirimli kontolcüler kullanılmaktadır.

Denizcilik kontrol problemlerinden biri yön tutmadır. Bu kontrol probleminde, bir kontrolcü olabildiğince az enerji harcayarak bir su üstü aracının sapma açısını sabit tutmaya çalışır. Bu problem için, genel olarak sektörde PID benzeri geleneksel geri beslemeli kontrolcüler kullanılmaktadır. Bu çalışma, dalgalı deniz koşullarında yön tutma problemi için genelleştirilmiş bir RL kontrolcüsü tasarlamaya odaklanmıştır.

Kontrolcü, Maritime Research Institute of Netherlands'in (MARIN) simülasyon ortamı olan XSimulation ve modern bir RL algoritması olan SACda kullanan MARINRL isimli bir kurumiçi kütüphane kullanılarak geliştirilmiştir. SAC, actor-critic bir off-policy RL algoritmasıdır (Ayrıntılı bilgi Section 3'de verilmiştir). Bu algoritmayı diğer algoritmalardan ayırt eden en büyük özellik, entropi adı verilen bir kavram kullanıyor olmasıdır. SAC bağlamı içerisinde entropi, bir policy'nin rastgeleliği olarak nitelendirilebilir. Policy'nin rastgeleliğe yakın olması, agent'ın bir çok farklı çevresel koşulla karşılaşıp hızlı bir şekilde tecrübe edinebilmesini sağlar. Fakat policy fazla rastgele ise bu sefer agent'ın sıkça karşılacağı makul çevresel koşullarda edineceği tecrübe azalır. SAC, entropi kavramı yardımıyla, agent'ın policy'sini makullük sınırları içerisinde olabildiğince rastgele tutmaya çalışır. Bu

sayede SAC, çoğu diğer modern RL algoritmasından çok daha az veri ile çok daha başarılı policy'lere yakınsayabilmektedir. Deniz uygulamaları için gerçek bir geminin üzerinde bir RL agent'ı eğitmek çok pahalı olacağından, SAC'ın bu özelliği tez için oldukça önemlidir.

Bu çalışmada dalgalar lineer olarak modellenmiştir. Geliştirilen kontrolcü, kontrol edilen aracın pervane RPM'lerinin sabit tutulduğ ve deniz durumu 1 ve deniz durumu 6 arasında çalışacak şekilde tasarlanmıştır.

Kontrolcünün geliştirme sürecinde bir çok problemle karşılaşılmıştır. Aşırı dümen kullanımı, hedef sapma açısını ıskalayıp ters yönden sapma hatası alma ve ufak sabit kalıcı sapma hatalarını aşamama bu problemlerden sadece bir kaçıdır. En son geliştirilmiş RL agent bu hataların hepsi ile başedebilmek için farklı mekanizmalar kullanmıştır. Bu mekanizmalar agent'ın ödül (reward) fonksiyonuna yapılan çeşitli değişikleden oluşmaktadır.

Tasarlanan kontrolcülerin performansına bir referans noktası oluşturabilmek için bir Linear Quadratic Regulator (LQR) kontrolcüsü tasarlanmıştır. LQR, model tabanlı bir kontrolcüdür. İsminden anlaşılabileceği üzere, sistem dinamiklerini lineer bir şekilde modeller. Kontrolcünün davranışını ayarlamak için Q ve R ağırlık matrisleri kullanılır. Bu matrisler sırasıyla hatayı ve tahrik sistemi kullanımı (bu problem için dümen kullanımı) cezalandırır. Q'daki yüksek değerler kontrolcünün düşük hatalara öncelik vermesini sağlarken, R'daki yüksek değerler kontrolcünün tahrik sisteminin az kullanımına öncelik vermesini sağlar.

Tez, geliştirilen RL agent'ın son hali üzerinden konuşmak yerine, oldukça basit bir agent'dan başlayıp onu adım adım iyileştirmektedir. Her adımın sonunda, agent'ın performansı farklı deniz koşullarında uzun bir sürece değerlendirilmekte ve bu performans tasarlanan LQR kontrolcüsü ile karşılaştırılmaktadır. Böylelike yapılan her iyileştirmenin agent performansına olan etkisi açık bir şekilde döküman edilebilmiştir. Bununla gelecekte yapılacak denizcilik konulu RL çalışmalarına yön göstermek hedeflenmiştir.

En son geliştirilen RL agent'ın, sapma açısı hatası ve dümen kullanımı açısından tasarlanan LQR kontrolcüsünden daha iyi performans gösterdiği gözlemlenmiştir.

İlk bölüm, yön tutma problemi ve RL'in kaba bir özetini vermekte, tezin amacını açıklamakta ve tezin konusu üzerine yapılmış olan literatür araştırması hakkında bilgi vermektedir. Kullanılan araçlar ve simulasyon ortamı hakkında bazı ayrıntılı bilgiler de burada verilmiştir.

İkinci bölüm, RL agent'ın performansı ile karşılaştırılacak olan geleneksel LQR kontrolcüsünü tanıtmakta ve ardından bu çalışma için kullanılacak olan LQR kontrolcüsünün tasarımını anlatmaktadır.

Üçüncü bölüm, sonraki bölümler için bir temel oluşturmak amacıyla RL'i derinlemesine açıklamaktadır.

Dördüncü bölüm, basit bir RL agent ile başlamakta ve bu agent'ın adım adım iyileştirilme sürecini anlatmaktadır. Her adımda agent performansı tasarlanan LQR kontrolcüsü ile karşılaştırılmış, agent'ın zayıf yönleri analiz edilmiş ve bir sonraki adım için yapılacak olan iyileştimeler belirlenmiştir.

Beşinci bölüm önceki bölümleri özetlemekte, tezin literatüre olan katkılarını açıklamakta ve gelecekte yapılabilecek ilgili çalışmalar üzerine konuşmaktadır.

1. INTRODUCTION

Reinforcement Learning (RL) is a subbranch of machine learning where a learner (the agent) is expected to find which actions to take under what kind of environmental circumstances to maximize a reward. The agent does this by; observing the state of its environment (state vector), taking an action, and checking how desirable its actions were by checking the rewards it receives. As the agent experiments with more actions, it learns the behavior of its environment and sharpens its action-taking strategy (policy). With a sufficiently long training, a well-designed agent will converge to a sufficiently well-behaving policy [1].

In the last few decades, RL and Deep RL (DRL) have been used for many decision-making problems with promising results. RL methods have shown a superhuman level of performance in games such as Atari [2], chess, and Go [3]. RL also has shown great potential for varying real-life applications such as robotics [4], finance [5], and even analog IC design [6].

RL has also been successfully used for many control problems (two marine control examples being [7] and [8]). Unlike many traditional control methods, model-free RL does not require any model dynamics to operate. It approximates the model dynamics through trial-and-error. This property is especially beneficial for problems where the model dynamics are non-linear or not well-known as one could develop an agent without needing to derive the dynamics of the model. However, for maritime applications, classical feedback controllers such as PID are still the dominant method of control [9].

Heading-keeping is one of the maritime control problems in which a controller is tasked with keeping the heading (yaw) angle of a surface vessel constant. For this problem, state-of-the-art methods are generally feedback controllers [9].

This thesis, explains the process of developing a generalized RL controller for heading-keeping under different sea conditions.

1.1 Purpose of Thesis

The developed RL agent has been evaluated for different wave directions, wave heights, and wave periods in a simulation environment. The RL agent has been demonstrated to be performing better than a base case traditional controller both in terms of rudder usage and yaw error.

The developed RL controller has been improved iteratively, where in each iteration a different weakness of the controller has been addressed. The process of improvement has been documented in detail.

The purpose of the thesis is to;

- Develop an RL agent that can compete with traditional controllers in the heading-keeping problem,
- And provide guidance for possible future RL applications in the field of marine by creating a detailed document that explains the process of developing an RL controller.

1.2 Literature Review

The invention of gyrocompass in 1908 allowed control systems to have access to yaw information, which led to the development of the first model-based ship controllers. And as global satellite navigation became more and more available, it became possible for many marine control problems (including heading keeping) to be reliably handled by autonomous systems. These initial feedback control systems have laid the foundation of today's autopilots.

In modern control systems, all sorts of techniques such as; PID and LQR control, fuzzy systems, different variations of Kalman filters, and even neural-networks are used. Generally speaking though, traditional feedback control methods are the dominant method of control in the industry [9].

However, the nonlinear nature of the environment dynamics is hard for traditional controllers such as PID or LQR to deal with. In addition, things such as the load on the vessel and environment dynamics can change over time. Traditional control methods have been improved upon by many works to tackle these issues. For example, [10] uses a Kalman-based adaptive controller for dynamic positioning under different sea conditions, however it requires a model. [11] uses a fuzzy-based PID controller for station keeping again under different environmental circumstances. However with this method, the maximum and minimum PID gains must be somehow decided on (usually empirically). Also the designers must decide what are the factors that should change the PID gains, and how the change in these factors will affect the PID gains.

While the attempts of using RL-based controllers in the field of marine are in their infancy, there are still many promising studies. [12] develops an RL-based dynamic positioning system, tests it in both simulation environment and sea, and compares it to traditional controllers. They see that the controller works well both in simulation and sea. However, they see that while the developed RL agent works better than traditional methods in the simulation environment, the same agent isn't able to beat the performance of traditional controllers in the sea. The inconsistencies between the idealized simulation environment and the sea are guessed to be the cause of this. [13] manages to get great results for an underactuated marine vessels in a straight path following problem using RL.

Using the knowledge gathered in the literature survey as a foundation, the thesis objective have been chosen and the rest of the study has been followed through.

1.3 Exact Definition of Problem Case and The Used Tools

In this section, the exact definition of the problem case will be given and the used tools will be explained. Generally speaking, this information would be given under a section that gives all the experiment results (such as section 5 in this thesis). But for this particular study, the agent is improved iteratively and the LQR controller is designed and tested before the design of the RL agent. This means that the problem definition is required even before giving the theoratical foudation of RL.

The readers can feel free to skip this section if they are simply interested in getting a rough idea of the main strategies used for developing the RL agent.

1.3.1 Problem case

As mentioned before, the studied problem case is heading keeping in waves. The objective is to keep the heading (or yaw) angle of a marine vehicle constant. For this study, the vehicle should be able to operate in waves with;

- Any wave direction (Dp),
- Wave amplitudes (Hs) of 1m to 5.5m,
- Wave period (Tp) of 8.5s to 12.5 seconds.

These numbers have been chosen by looking at waves that are not too unlikely to encounter in wave statistics of North Pacific [14].

The 5415M frigate has been used for this study. It is based on the public bare hull form 5415 [15], which is used in many past studies. 5415M has some appendages that the bare hull form 5415 doesn't have; bilge keels, stabilizers, rudders, propellers, shafts, and supportive struts of those shafts.

An image of a scaled model of 5415M can be seen in Figure 1.1. The main particulars of it can be seen in Table 1.1.

Figure 1.1 : A scaled down model of 5415M built by MARIN.

Table 1.1 : Main particulars of 5415M.

Main particular	Value	Unit
L_{PP}	142	m
Draft aft	6.15	m
Draft fore	6.15	m
KG	7.51	m
GM	1.95	m
B_{WL}	19.06	m
k_{XX}	7.62	m
k_{YY}	35.5	m
k_{ZZ}	35.5	m

1.3.2 Used tools

For the simulation environment, MARIN's XSimulation has been used. The waves have been modeled linearly. The environment has been solved using RK4. The simulation sample time has been chosen as 0.2 seconds.

RL part of the things have been handled by MARIN's MARINRL python library. The library interfaces Stable Baselines's SAC agent [16] with XSimulation through a gym [17] environment and allows for reward function and state vector to be modified easily.

2. BASELINE CONTROLLER

Two metrics have been used to evaluate the quality of developed RL controllers; yaw mean absolute error (MAE) and rudder usage. Rudder usage has been assessed by looking at the absolute area under the rudder time trace. The designed RL controllers also have been compared to a baseline traditional controller in terms of these metrics.

For this study, a simple Linear Quadratic Regulator (LQR) controller has been chosen as the baseline. This section will go over the basic principles of LQR and then will explain how an LQR controller has been designed for this problem case. For more information on LQR, please refer to [18].

2.1 Linear Quadratic Regulator (LQR)

To drive the rudder, LQR outputs the control signal

$$u = -Kx \tag{2.1}$$

in which K is the control gain that minimizes the cost function

$$\int_0^\infty (x^T Q x + u^T R u) dt \tag{2.2}$$

for the state space model

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2.3}$$

In these equations, x is the system state, u is the system input, and y is the system output. A, B, C, and D matrices determine the system behavior. The controller will try to pull the state to zero. The exact behavior of the controller is determined by the Q and R weight matrices which penalize the system error and actuator usage respectively.

The gain matrix K can be calculated through

$$K = R^{-1}B^T X \tag{2.4}$$

Which can be done after solving the Ricatti equation for X

$$A^T X + XA - XBR^{-1}B^T X + Q = 0 \tag{2.5}$$

The calculated gain will be optimal if the used linear model perfectly explains the system.

2.2 Design

The yaw response of a vehicle can be modeled as a function of rudder by a first order Nomoto model [9]. This model can be used as the state space model for the LQR:

$$\begin{bmatrix} \ddot{\psi} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{U}{T} & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\psi} \\ \psi \end{bmatrix} + \begin{bmatrix} \frac{KU^2}{T} \\ 0 \end{bmatrix} \delta \tag{2.6}$$

The yaw angle is given by ψ and the constant surge speed is given by U. The rudder angle δ is the free variable that the LQR (and also RL in the future) will use for heading control. The time constant $T = 82.2sec$ and the model gain $K = -6.4.10^{-3} rad/rad$ have been identified by MARIN in one of their prior projects. Note that this model is only used by LQR and not the simulation environment.

One of the weight matrices Q has been chosen as $Q = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$. Since only the last element of Q is nonzero, the controller is only punished for the yaw error but not for the rate-of-turn.

This setup will result in a gain matrix of $K = \begin{bmatrix} K_1, K_2 \end{bmatrix}$ where K_1 is the gain on rate-of-turn and K_2 is the gain on yaw error.

The behavior of this controller can be determined by changing R. Reducing R will make the controller more aggressive which can decrease the yaw MAE. But an unreasonably small R may damage the actuators or result in a rudder behavior that can not be realized by a real system. Increasing R will reduce the rudder usage. But an unreasonably large R may cause the yaw MAE to increase to an undesirable level.

To see the effect of different R values, LQRs with different R's has been used for the heading keeping of 5415M in XSimulation. The controllers have been tested on total of 210 different environments in which waves can have;

- Seven different phases,
- Dp of 0, 45, 90, 135, or 180 degrees,
- Hs of 3 or 5.5 meters,
- Tp of 8.5, 10.5, or 12.5 seconds.

The controllers spent a total of 1000 seconds in each environment. Their performance has been checked in terms of yaw MAE and rudder usage which can be seen in Figure 2.1.

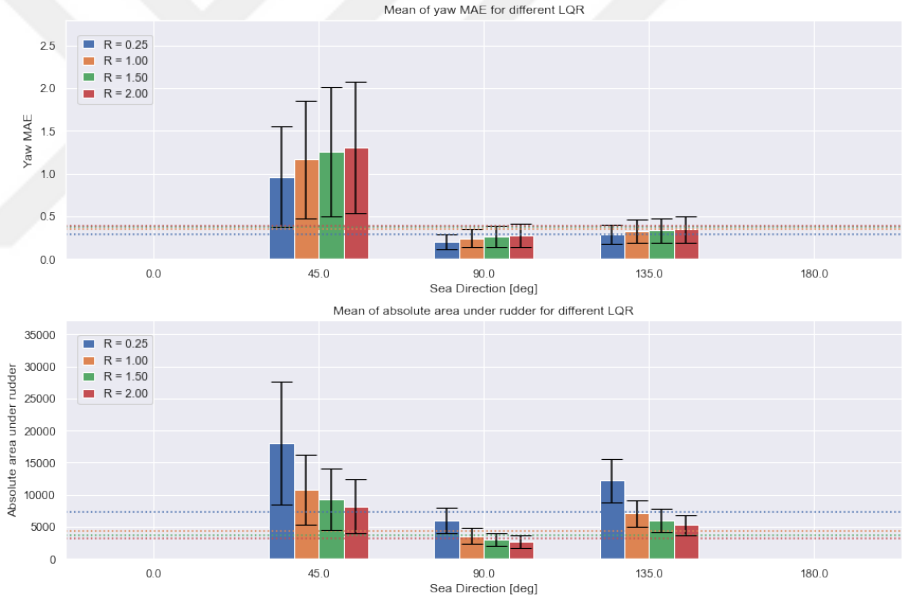

Figure 2.1 : LQRs with different R values.

In Figure 2.1, both the yaw and the rudder are in degrees. All simulations have been averaged for each wave direction. The average value of the metrics on each wave direction has been shown by their corresponding bars. The standard deviation of these metrics on each wave direction has been indicated by the error bars. The overall average of these metrics has been indicated by the horizontal dotted lines.

When the figure is observed, the relationship between R and the rudder can be seen clearly. For this study, $R = 1.0$ has been chosen after a simple grid search as it gives a good balance between rudder usage and yaw error.

3. REINFORCEMENT LEARNING (RL)

RL is a machine learning method that deals with decision-making problems. The learner, referred to as the agent, learns which actions to take under which circumstances by experimenting with different actions in its environment. The agent is able to assess the desirability of an action through a reward it receives from its environment after each action. The agent tries to converge to an action-taking strategy (or policy) that generates the most reward.

Fig. 3.1 can be seen to get a sense of where RL stands under AI.

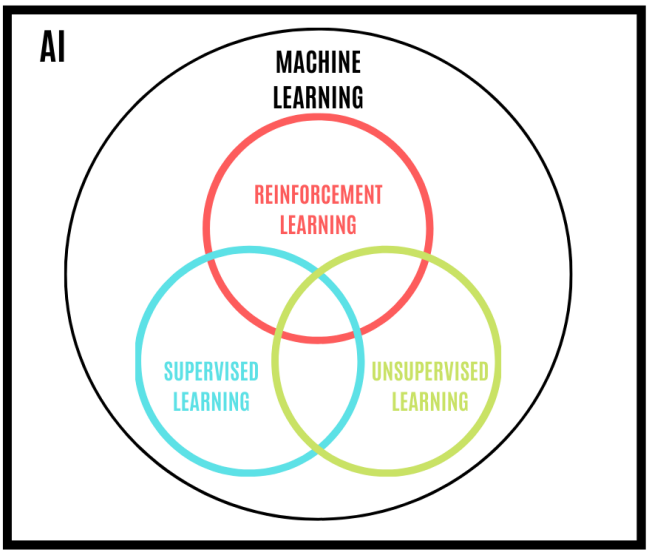

Figure 3.1 : Place of RL in AI.

This section explains the main concepts behind Reinforcement Learning (RL). Then it goes over the specific RL algorithm that is used for this study.

3.1 Markov Decision Processes (MDPs)

Markov Decision Processes or MDPs are the mathematically idealized form of the RL problem. Even though they are idealized, they still contain the main concepts of RL;

- A sequential decision-making problem,

- An agent that takes actions and receives rewards,
- And an agent that needs to find a balance between getting the most reward by using the best policy it has (exploitation) and sharpening its policy by taking potentially suboptimal unknown actions (exploration).

The thesis uses [1] as its main source for the information under this section and mainly follows the notation of this book.

3.1.1 MDP basics

MDPs model the interaction of a decision-maker and an environment. The decision-makers in the MDPs are referred to as agents. Anything outside the agent is referred to as the environment. The agent looks at the environment, decides on an action, and then sends this action to the environment. The environment uses the action to update itself, and then sends the new state of the environment to the agent along with a reward. Reward simply is a numerical value that the agent attempts to maximize over time. The agent and the environment may repeat this indefinitely or may stop after some time. This interaction has been visualized in Figure 3.2.

Figure 3.2 : Visualization of an MDP.

The agent-environment interaction happens at discrete time steps $t = 0, 1, 2, ...$ At a time step t, the environment sends $S_t \in \mathcal{S}$ to the agent. After observing S_t, the agent takes an action $A_t \in \mathcal{A}$ and sends it to the environment (action set A can be a function of state, ignored for simplicity). The environment uses this action A_t to transition into the next state S_{t+1}. It then sends the new state S_{t+1} and a numerical reward $R_{t+1} \in \mathcal{R}$ to the agent.

In a finite MDP, the sets of state, action, and reward ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) have a finite number of elements. S_t and R_t have a discrete probability distribution and they *only* rely on S_{t-1} and A_{t-1}. The probability of a specific $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ to occur can at time t can be calculated through $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ defined as

$$p(s', r|s, a) \doteq \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{3.1}$$

Where $s', s \in \mathcal{S}$, $a \in \mathcal{A}$, and $r \in \mathcal{R}$. p is a deterministic function which describes the behavior of the environment. For every s and a pair, it gives the probability distribution

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{3.2}$$

As mentioned before, in MDPs, S_t and R_t *only* depend on S_{t-1} and A_{t-1}. This means that in these idealized processes, the S_t and R_t do not depend on the past but only depend on the present. If the state has perfect information over transitioning from a given S_{t-1} to S_t, then the state is said to have Markov property. A state having this property can be mathematically shown by the following

$$\mathbb{P}(S_{t+1}|S_t, A_t) = \mathbb{P}(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ...) \tag{3.3}$$

When equation 3.3 holds, we can say that the state has the Markov property as there is no need for the previous state-action pairs for deriving the correct probability.

3.1.2 Tasks and rewards

For some tasks, the agent-environment interaction naturally ends at a final time step T. These tasks are referred to as *episodic tasks*. In such tasks, the time between $t = 0$ and $t = T$ is called an *episode*. The last state in episodic tasks is referred to as the *terminal state*.

The agent needs to receive a reward signal from the environment occasionally to be able to assess the quality of its actions. For many problem cases, the environment sends a reward to the agent after every action. However there are exceptions such as

chess, where the agent receives a reward at the end of each episode (game) instead of receiving a reward after every move.

Excluding these exceptions, the agent will receive a reward $R_t \in \mathbb{R}$ after each action. The agent's objective is to maximize the cumulative reward it receives. This means that the agent should not only focus on the immediate reward, but it also should think about the total reward it is going to get in the long term.

RL attempts to maximize cumulative reward by maximizing the expected return G. In the case of an episodic task, the expected return at a time step t, G_t, simply can be described by

$$G_t \doteq R_{t+1} + R_{t+2} + ... + R_T \tag{3.4}$$

However, for many problems, the agent-environment interaction doesn't end after some predetermined time. Instead, the interaction is indefinite. For such problems, equation 3.4 doesn't work well as the return can be infinite. To avoid this issue, the concept of emphdiscounting can be used. Discounting makes it so that the agent prioritize immediate reward over future rewards by turning equation 3.4 into following

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.5}$$

where $0 \leq \gamma \leq 1$ is the *discount rate*. One can rearrange this equation to be recursive as follows

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{3.6}$$

As γ approaches 0, the agent will get more and more short-sighted. And as γ approaches 1, the agent will prioritize the future rewards more.

3.1.3 Policies and value functions

Policies, notated by π, are an agent's action-taking strategy. Mathematically, they map states to the probability of taking specific actions

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \tag{3.7}$$

equation 3.7, gives the probability of taking the action $a \in \mathcal{A}$ when the state is $s \in \mathcal{S}$ at time t. For each s, π gives a probability distribution of $a \in \mathcal{A}$. Since the agent must take an action, $\sum_a \pi(a|s) = 1$ must hold.

To be able to determine the desirability of a state, a policy π will utilize the value functions. The *state-value function* under policy π, denoted by v_π, is defined as the expected return if the agent follows current policy π after state s

$$v_\pi(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] = \mathbb{E}_\pi[G_t|S_t = s] \tag{3.8}$$

for all $s \in \mathcal{S}$, for any t, and where $\mathbb{E}_\pi[.]$ is the expected value of a random variable given that the agent always follows the policy π. A terminal state, if exists, will always have the value of 0. This function sometimes will be simply referred to as the value function.

The *action-value function* under policy π, dentoted by q_π, is defined by the expected return if the agent takes the action a and then follows current policy π after state s.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{3.9}$$

Using these equations, one could derive the following equation using the relationship between the value function and the action-value function

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{3.10}$$

For state s and an action $a \in \mathcal{A}$. This equation shows that the state-value of a state s is the same as the sum of all action-values weighted by the possibility of their respective actions occurrence.

3.1.4 Bellman equations

Just like equation 3.6, equation 3.10 can also be written in a recursive form. For any policy π, the value of a state s can be written as a function of the value of its possible successors

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi)[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.11}$$

This equation is referred to as the *Bellman equation* or *Bellman expectation equation* for v_π.

Similarly, the action-value function can also be written in a recursive form

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s']] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.12}$$

This equation is referred to as the *Bellman equation* or *Bellman expectation equation* for q_π.

3.1.5 Optimality

An agent will try to improve its policy as it gets more and more experience. Mathematically speaking, a policy π is considered better than or equal to some other policy π' if its expected return is equal to or greater than π' for all states

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v'_\pi(s) \forall s \in \mathcal{S} \tag{3.13}$$

For any given problem at least one *optimal policy*, i.e. a policy that is better than or equal to all others, must exist. These policies are denoted by π_*. All optimal policies share the same state-value function v_*, called *optimal state-value function*. It is defined as

$$v_*(s) \doteq \max_\pi v_\pi(s) \forall s \in \mathcal{S} \tag{3.14}$$

They also share the same *optimal action-value function* q_*

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{3.15}$$

This equation can be written as

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_s = s, A_t = a] \tag{3.16}$$

When the optimal value functions are discovered, the MDP is solved. Note that even though there can be multiple optimal policies, there can only be one optimal state-value and action-value function as they are shared by all possible optimal policies.

It becomes easy to find the optimal policy after finding the optimal value functions. The optimal state-value function can be used for a one-step search in state space can be used to find all the optimal actions for each state. The optimal action-value function can be used for finding an optimal policy can be written as

$$\pi_*(s,a) = \begin{cases} 1, & a = \underset{a \in \mathcal{A}}{argmax} q_*(s,a) \\ 0, & otherwise \end{cases} \tag{3.17}$$

where *argmax* finds the argument that maximizes a function (set of the argument below, the function to maximize to the right).

Since v_* is a value function, it must satisfy the Bellman equations and thus can be written as

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned} \tag{3.18}$$

The last two lines of equation 3.18 are different forms of *Bellman optimality equation* for v_*. Let us do the same thing for the action-value functions to derive the two different forms of *Bellam optimality equation* for q_*

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{3.19}$$

Notice how these functions are independent of any policy as every optimal policy for a problem will have the same optimal value functions.

3.2 Learning Methods

Now that the foundation of RL has been laid, the thesis will go over different methods of learning without the model of the environment. The thesis has chosen Soft Actor-Critic (SAC) as its algorithm since it is a model-free state-of-the-art method that is quite sample-efficient. However, actor-critic methods lie in between value-based and policy-based methods (see Figure 3.3), so the thesis will explain these concepts first.

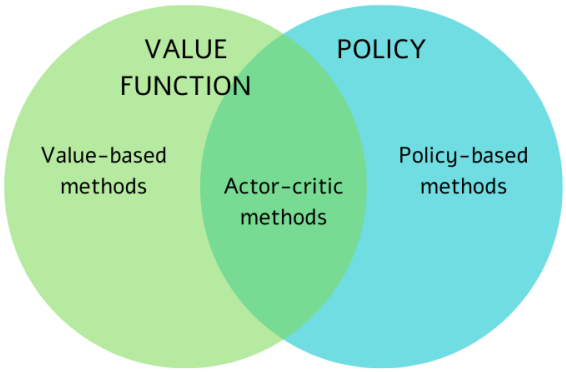

Figure 3.3 : Value-based, policy-based, and actor-critic methods.

3.2.1 Value-based methods

3.2.1.1 Monte Carlo (MC)

Monte Carlo (MC) methods are *model-free methods* which means that they don't try to directly model the environment. They run for many episodes, then average out the returns to estimate the value functions. They can only work with episodic tasks.

This subsection aims to give a brief overview of MC methods instead of giving an extensive overview of how MC can be used for control. Because of this, this subsection will simply go over how one can use MC for *preditction*, i.e. for computing v_π for an arbitrary π.

MC slightly modifies equation 3.8, and replaces the expected return with an empirical mean return. The method hopes that with enough episodes, the mean return should converge to the expected return. Using this idea, the MC methods will look at the mean return for every encountered state after running for a while.

When a state s is encountered in an episode, it is referred to as a *visit* to s. The *first-visit MCs* only use the returns from the first visits to a state s whereas *every-visit MCs* uses the returns from every visit to a state s. An overview of the first-vist MC can be seen in algorithm 1.

Algorithm 1 First-visit MC prediction

Input: A policy π
Output: A value function V (attempts converge to v_π)

$N(s) \leftarrow 0 \; \forall s \in (S)$
$returns\_sum(s) \leftarrow 0 \; \forall s \in (S)$
Loop for each episode:
    Generate an episode using π
    $G \leftarrow 0$
    Loop for each t:
        $G \leftarrow \gamma G + R_{t+1}$
        If $N(S_t)$ is not 1:
            $N(S_t) \leftarrow N(S_t) + 1$
            $returns\_sum(s) \leftarrow returns\_sum(s) + G$
$V(s) \leftarrow returns\_sum(s)/N(s) \; \forall s \in (S)$

Generally speaking, two main modifications are made to this algorithm. First of all, there is no need for keeping the sum of returns for all states. Instead, one could calculate the means iteratively. Means $\eta_1, \eta_2, ..., \eta_k$ of a sequence $x_1, x_2, ..., x_k$ can be computed through

$$\begin{aligned} \eta_k &= \frac{1}{k}\sum_{i=1}^{k} x_i \\ &= \frac{1}{k}\left(x_k + \sum_{i=1}^{k-1} x_i\right) \\ &= \frac{1}{k}(x_k + (k-1)\eta_{k-1}) \\ &= \eta_{k-1} + \frac{1}{k}(x_k + \eta_{k-1}) \end{aligned} \tag{3.20}$$

Using this idea one can calculate $V(S_t)$ iteratively through

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)) \tag{3.21}$$

The second modification that can be made is to replace the $\frac{1}{N(S_t)}$ with a constant *learning rate*, notated by α

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{3.22}$$

Using this, one can give more weight to the recent return (Section 2.5 [1]). In non-stationary problems (in which the environment varies over time), this is quite useful since the more recent experiences of the agent will have more information about the current state of the environment.

In equation 3.21 and equation 3.21, G_t is referred to as the *MC target* and $(G_t - V(S_t))$ as the *MC error*. The V is updated every episode so that it gets closer to the MC target, using MC error as a direction. Learning rate is used for configuring how big the steps towards the MC target should be.

3.2.1.2 Temporal-Difference (TD) learning

Temporal-Difference (TD) attempts to solve one of the weaknesses of MC: MC must wait until the end of an episode to update $V(S_t)$. TD updates the $V(S_t)$ every time step. To be able to achieve this, TD modifies equation 3.22 so that it uses an estimate of the return instead of the actual return

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.23}$$

When the environment returns the reward R_{t+1} to the agent, TD uses this reward and the estimate of $V(S_{t+1})$ to update V. Which effectively makes the TD target $R_{t+1} + \gamma V(S_{t+1})$. The $[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$ in the equation is referred to as *TD error* and is notated by δ. Note that equation 3.23 is used for a specific kind of TD called $one - stepTD$. The v_π estimation algorithm for one-step TD can be seen in algorithm 2.

Algorithm 2 one-step TD v_π estimation

Input: A policy π
Output: A value function V (attempts converge to v_π)
Algorithm arguments: $small \alpha > 0$

Initialize $V(s)\ \forall s \in (S)$ to random values except $V(terminal) = 0$
Loop for each episode:
    Initialize S
    Loop for each time step:
        $A \leftarrow$ action given by π for S
        Send A to the environment
        Receive R and S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
        Break if S is terminal

A comparison between MC and TD can be seen in Table 3.1.

Table 3.1 : A comparison between MC and TD.

MC	TD
Model-free	Model-free
Updates after each episode	Updates after each time step
Uses the actual return	Uses an estimate of the value function
Can only be used for episodic tasks	Can be used for episodic or continuous tasks
Can be sample-efficient	Usually more sample-efficient than MC
Has good convergence	Doesn't always have good convergence (see Section 6.2 in [1])

3.2.1.3 Control using different TD

This section will go over two TD control algorithms: *Sarsa* and *Q-learning*. The TD algorithm operate using the concept of *generalized policy iteration* (GPI). GPI consists of two steps; policy evaluation where the policy value function is estimated and policy improvement where the policy is improved using this estimated value function. A policy is considered to be converged if the policy improvement step stops causing changes.

Let us start with Sarsa first. The TD update rule for action-value function is derived as it can be used for finding which action leads to the highest value state

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \tag{3.24}$$

Action-value function is updated using equation 3.24 each time step. $Q(S_{t+1},A_{t+1})$ is zero if S_{t+1} is terminal. The name Sarsa comes from the fact that algorithm utilizes all the $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ terms. The use of an ε-greedy (explained below) Sarsa for control has been showed in an algorithmic form in algorithm 3.

Algorithm 3 ε-greedy Sarsa for control

Algorithm arguments: $small\alpha > 0$, small $\varepsilon > 0$

Initialize $Q(s,a)\ \forall s \in (S), \forall a \in (A)$ to random values except $Q(terminal,.) = 0$
Loop for each episode:
    Initialize S
    Choose A looking at S and using Q (with an ε-greedy approach)
    Loop for each time step:
        Send A to the environment
        Receive R and S'
        Choose A' looking at S' and using Q (with an ε-greedy approach)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R_{t+1} + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S', A \leftarrow A'$
        Break if S is terminal

It can be seen that q_π is updated every time step for the policy π. The agent has an ε chance of choosing a random policy, but it has a $1-\varepsilon$ chance of acting *greedily*, i.e. choosing the action that maximizes the action-value function. An agent said to be acting ε-greedily if it follows this approach. Mathemtaically, the policy that formulates this behavior can be described with the following (assuming there are m possible actions)

$$\pi(s,a) = \begin{cases} \varepsilon/m + 1 - \varepsilon, & a_* = \underset{a \in \mathcal{A}}{argmax} Q(s,a) \\ \varepsilon/m, & otherwise \end{cases} \tag{3.25}$$

In Q-learning, the update is made with the assumption that the agent will always choose the action that maximizes the $Q(s,a)$. But when taking an action it follows an ε-greedy policy with respect to $Q(s,a)$. So the policy used for updating Q and policy used

choosing actions are different. These sorts of methods are said to be *off-policy* where as methods such as Sarsa that use the same policy for updating Q and choosing actions are said to be *on-policy*. Update rule for Q-learning is given in equation 3.26, the algorithm for Q-learning control is given in algorithm 4.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \underset{a}{max} Q(S',a) - Q(S_t,A_t)] \tag{3.26}$$

Algorithm 4 ε-greedy Q-learning for control

Algorithm arguments: *small* $\alpha > 0$, small $\varepsilon > 0$

Initialize $Q(s,a)\ \forall s \in (S), \forall a \in (A)$ to random values except $Q(terminal,.) = 0$
Loop for each episode:
    Initialize S
    Loop for each time step:
        Choose A looking at S and using Q (with an ε-greedy approach)
        Send A to the environment
        Receive R and S'
        $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \underset{a}{max} Q(S',a) - Q(S_t,A_t)]$
        $S \leftarrow S'$
        Break if S is terminal

3.2.2 Policy-based methods

In value-based method, the agent's policy is obtained by a learned a value function. But policy-based methods do not learn a value function (with some exceptions), they learn the policy directly. Policy-based methods parameterize their policies and they choose actions without the need of a value function:

$$\pi(a|s,\theta) = \mathbb{P}(A_t = a|S_t = s, \theta_t = \theta) \tag{3.27}$$

where θ is the *parameter vector*. These methods define a function parameter vector, $J(\theta)$, to measure the quality of a given policy. So the RL problem turns into an optimization problem where the learner tries to maximize J. One method of finding the θ that maximizes J is to use *gradient ascent*

$$\theta_{t+1} = \theta_t + \alpha \widehat{\Delta_\theta J(\theta_t)} \tag{3.28}$$

where $\widehat{\Delta_\theta J(\theta_t)}$ term is the stochastic gradient with respect to θ. The methods that follow this approach are referred to as *policy gradient methods* (See Section 13.2 in [1] for theoretical details).

The function J is defined differently for episodic and continuous tasks. For episodic tasks it can be defined as follows

$$J(\theta) \doteq v_{\pi_\theta}(s_0) \tag{3.29}$$

where v_{π_θ} is the policy that uses θ and s_0 is the initial state of the episode. Eq. 3.29 tells that the quality of a policy is measured by the expected total reward (discounted if $\gamma \neq 1$) starting from s_0.

3.2.2.1 REINFORCE

REINFORCE algorithm is a policy gradient method [19]. In reinforce, θ update is as follows

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\Delta_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} = \theta_t + \alpha G_t \Delta_\theta ln\, \pi(A_t|S_t,\theta) \tag{3.30}$$

REINFORCE is a MC algorithm since it uses G_t thus requires the entire episode trajectory. The application of REINFORCE can be seen in an algorithmic form in algorithm Refalg:reinforce.

Algorithm 5 REINFORCE: MC Policy Gradient Control

Input: a parameterized policy $\pi(a|s,\theta)(mustbedifferentiable)$
Algorithm arguments: $small\alpha > 0$

Initialize θ arbitarily
Loop for each episode:
Generate the entire episode trajectory: $S_0, A_0, R_1, ...S_{T-1}, A_{T-1}, R_T$ following π
For each time step t of the episode:
$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
$\theta \leftarrow \alpha \gamma^t G \Delta_\theta ln\, \pi(A_t|S_t,\theta)$

REINFORCE has the main features of MC methods; it has good convergence but it is slow and not that sample-efficient. To make it learn faster, one common approach is to modifying the update rule adding a baseline gain:

$$\theta_{t+1} \doteq \theta_t + \alpha(G_t - b(S_t))\Delta_\theta ln\,\pi(A_t|S_t,\theta) \tag{3.31}$$

The baseline is a function of state that helps the method distinguish bad actions from the good ones. One choice of a baseline is the state-value estimate $\hat{v}(S_t,w)$ where $w \in \mathbb{R}^m$ is the weight vector which is also learned (for example with an MC method). The new algorithm can be seen in algorithm 6

Algorithm 6 REINFORCE with baseline

Input: a parameterized policy $\pi(a|s,\theta)$ and a state-value function $\hat{v}(s,w)$ (both must be differentiable)

Algorithm arguments: $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize θ and w arbitarily

Loop for each episode:

Generate the entire episode trajectory: $S_0,A_0,R_1,...S_{T-1},A_{T-1},R_T$ following π

For each time step t of the episode:

$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

$\delta \leftarrow G - \hat{v}(S_t,w)$

$w \leftarrow w + \alpha^w \delta \Delta_w \hat{v}(S_t,w)$

$\theta \leftarrow \alpha^\theta \gamma^t G \Delta_\theta ln\,\pi(A_t|S_t,\theta)$

3.2.3 Actor-critic methods

Actor-critic methods can be thought of as a TD version of policy gradient. They consist of an actor and a critic. The actor takes actions and handles the policy convergence, whereas the critic assesses the quality of the taken actions. The critic updates the parameter vector of the value function w. The actor improves its policy by updating the parameter vector policy θ with the help of the critic.

Unlike REINFORCE, actor-critic methods use an iterative one-step return (with exception of some n-step actor-critics):

$$\theta_{t+1} \doteq \theta_t + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1},w) - \hat{v}(S_t,w))\Delta_\theta ln\pi(A_t|S_t,\theta) \tag{3.32}$$

where the term $\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)$ is the one-step TD error. It can be seen that transitioning from REINFORCE to actor-critic methods is really similar to transitioning from MC to one-step TD. An example one-step actor-critic design can be seen in Algorithm 7.

Algorithm 7 One-step actor-critic

Input: a parameterized policy $\pi(a|s,\theta)$ and a state-value function $\hat{v}(s,w)$ (both must be differentiable)

Algorithm arguments: $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize θ and w arbitarily
Loop for each episode:
Initialize S
$I \leftarrow 1$
While S is not terminal:
$A \sim \pi(.|s,\theta)$
Send A to the environment
Receive S' and R
$\delta \leftarrow R + \gamma\hat{v}(S',w) - \hat{v}(S,w)$
$w \leftarrow w + \alpha^w \delta \Delta_w \hat{v}(S,w)$
$\theta \leftarrow \theta + \alpha^\theta I \delta \Delta_\theta ln\, \pi(A|S,\theta)$
$I \leftarrow \gamma I$
$S \leftarrow S'$

3.2.4 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [20] is an off-policy actor-critic algorithm. It has been used for this study because it is a sample-efficient state-of-the-art method. The algorithm tries to maximize a trade-off between exploration and exploitation, using the concept of entropy.

Within the context of SAC, entropy can be defined as the randomness of a policy. If the probability of choosing actions is governed by white noise, it would be said that the entropy is the highest. In the other hand if the probability of choosing an action is not probabilistic at all, the entropy would be zero. SAC attempts to find a sweet spot in entropy where the policy is random enough to explore a lot while still isn't too random to not be able to finish the task at hand.

Entropy can be added to an RL framework through, for example, adding a negative weighted entropy to the loss function. This would make it so that higher entropies decrease the loss and thus encouraged.

In SAC, the policy tries to maximize the expected return *and* entropy:

$$J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha H(\pi_\theta(.|s_t))] \tag{3.33}$$

Where H is the entropy term and α is the trade-off coefficient.

The SACs main futures are as follows

- It is model-free
- It is actor-critic
- It is off-policy
- It maximizes a trade-off between exploration and exploitation, using entropy

4. HEADING KEEPING IN WAVES USING RL

This section aims to design a SAC-based controller for heading keeping in waves. The exact definition of the problem and the used tools are given under Section 1.3. After choosing an RL algorithm (in this case SAC), the main workload of the development lies in choosing a good state vector and designing a good reward function.

This section will start off by explaining the used action and space vector. After this, the thesis will propose a really simple reward function and iteratively improve it. For every reward function, the performance of the developed agent will be evaluated in terms of yaw MAE and rudder usage, and then will be compared to the LQR controller designed in Section 2. For each reward function, seven different agents will be trained to see if the reward function relies on a lucky start to perform well.

4.1 State & Action Vectors

The state vector is used as an input to the actor and the critic. It essentially defines the perception of the agent. So when deciding the elements of the state vector, one should make sure that; the information that the agent receives is sufficient and that there is no redundancy in the state vector. Also, it is important that the state vector doesn't have elements that are hard to obtain in real-life scenarios. For example, in a simulation environment, the agent could be provided with the exact sea state information. Intiutively, one would expect this information to be extremely useful to the agent. However, it is a very hard task to obtain the sea state in a real-life application. So one should avoid adding it to the state vector unless they also have a reliable method of obtaining the sea state.

Keeping this in mind, the following state vector has been chosen

$$s_t = [\psi_{err}, \delta, \dot{\psi}, \ddot{\psi}] \tag{4.1}$$

where ψ_{err} is the yaw error, δ is the rudder angle, $\dot{\psi}$ is the yaw rate, and $\ddot{\psi}$ is the yaw acceleration.

The action vector has only one element

$$a_t = [n_{\dot{\psi}}] \tag{4.2}$$

where $n_{\dot{\delta}} \in [-100\%, 100\%]$ is the rudder rate in terms of percentage.

4.2 Reward Function #1

Reward functions can be a function of anything and they can be as complicated as the designer wishes. However, just like the state vector, one should avoid using information that can't be reliably obtained in a real-life situation. Also keeping the reward function as intuitive as possible allows for the reward function to be configured and modified easily.

One simple starting point for designing a reward function is to either give negative rewards for higher errors or give positive rewards for lower errors to encourage low errors. The most straightforward way of doing this would be to give a constant reward to the agent if the error is lower than some constant value as follows

$$R = \begin{cases} c_{reward}, & \psi_{err} < c_{err} \\ 0, & otherwise \end{cases} \tag{4.3}$$

where c_{reward} is some constant reward and c_{err} is the constant error threshold. When we look at the literature (for example [8]), it can be seen that such functions don't give great results. If the reward function doesn't change for different ψ_{err} under c_{err}, the agent will have no incentive for discovering a policy that can get errors lower than c_{err}. Also, the agent will not have a strong sense of "direction" when updating its policy. This is because it is really difficult to differentiate between two different policies that can achieve $\psi_{err} < c_{err}$ or two different policies that can't achieve $\psi_{err} < c_{err}$, as in both cases the policies will be indistinguishable in terms of reward.

A better idea would be to use a continuous function. For example a linear reward function such as the following can be chosen

$$R = c_{max} - c_{slope}\psi_{err} \tag{4.4}$$

where the agent gets rewarded more and more as the ψ_{err} decreases and c_{slope} determines how strongly the punishment increases as ψ_{err} increases. An example of this reward function can be seen in Figure 4.1.

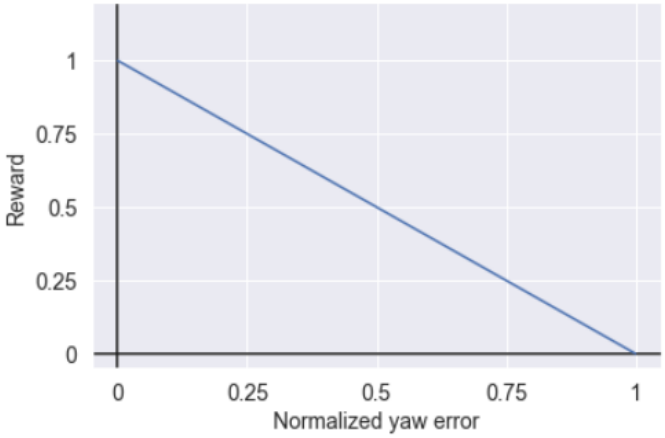

Figure 4.1 : A linear reward function where $c_{max} = 1$, $c_{slope} = 1$, and ψ_{err} has been normalized between 0 and 1.

A linear function is not the only choice for a simple continuous reward function. For example, many works in the literature utilize a reward function with a gaussian behavior, [8] and [7] being two examples. Such a reward function would look like this:

$$R = c_{scale} exp\left(-\frac{\psi_{err}^2}{2c_{std}^2}\right) \tag{4.5}$$

where the agents gets rewarded more and more as it lowers the ψ_{err}, c_{scale} is the scale of the reward, and c_{std} is the standard deviation of the bell. An example of this reward function can be seen in Figure 4.2.

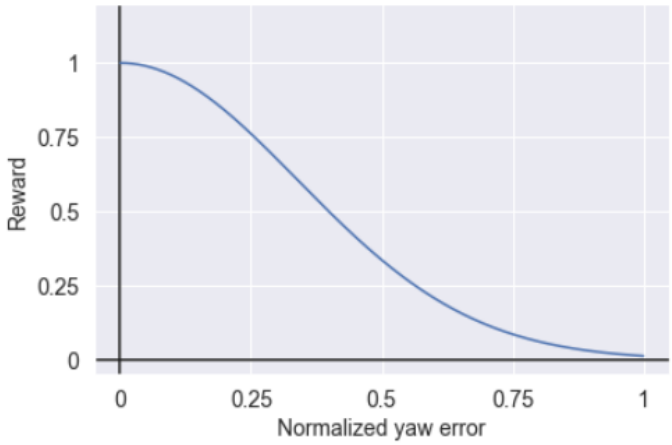

Figure 4.2 : A gaussian reward function where $c_{scale} = 1$, $c_{std} = 0.6$, and ψ_{err} has been normalized between 0 and 1.

However, [21] argues that, at least for their problem case, an absolute exponential function is a better candidate since it has better gradient around its tails. With this property, they are hoping to give more information to the agent about desirability of its state even when the error is relatively high. An absolute exponential reward function would look as the following

$$R = c_{scale}\left(-c_{err\_scale}|\psi_{err}|\right) \tag{4.6}$$

where c_{scale} is the scale of the reward function and c_{err_scale} is the scale of the yaw error. A comparison between an absolute exponential function and a gaussian function with a relatively lower standard deviation can be seen in Figure 4.3.

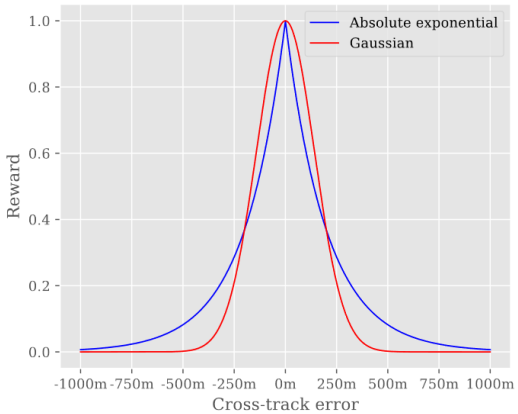

Figure 4.3 : A comparison between an absolute exponential function and a gaussian (taken from [21]).

This thesis advocates for the use of an absolute exponential function for an entirely different reason: steady-state error. A system is said to have a stead-state error when there is a constant error as the limit in time goes to infinity. For our case, if the controller isn't able to overcome some constant yaw error and maintains this constant error throughout the episode, we can say that we have steady-state error. The previous RL studies in MARIN suggest that if a mechanism for fighting steady-state error is not implemented, the system will have steady-state error for this problem cases.

The thesis believes that the extreme gradient around zero yaw error in absolute exponential function will encourage the agent into finding policies that can overcome steady-state error. This is because the reward the agent gets improves drastically as the error goes to zero. This belief is proven to be correct in the third iteration of the reward function.

Considering all of this, the study decided to use a simple absolute exponential function for the first iteration of the reward function.

4.2.1 Training setup

As mentioned before, SAC has been chosen as the RL algorithm to use. Used SAC hyperparameters can be seen in Table 4.2. For the state and action, the vectors given in Section 4.1 have been used. For the reward function, a simple absolute exponential function given at equation 4.6 with $c_{scale} = 1$ and $c_{err\_scale} = 5$ has been used. c_{scale} has been chosen as 1 so that the reward scale can be tuned easily if needed. c_{err_scale} has been chosen with a simple grid search. The yaw error that feeds into this function has been normalized between 0 and 1. The reward function can be seen in Figure 4.4.

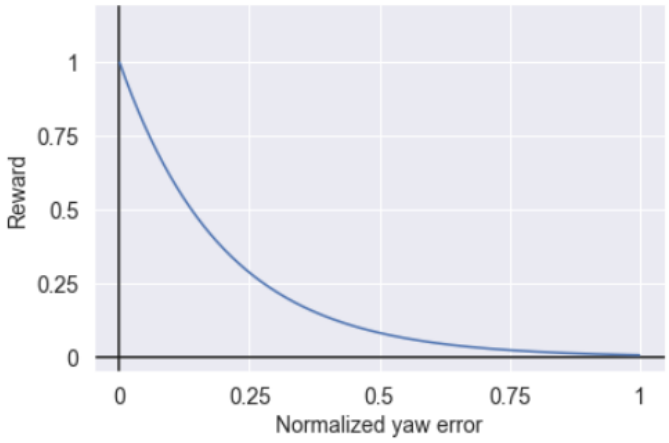

Figure 4.4 : The used absolute exponential function.

To make sure that the agent performance doesn't rely on a lucky initial start, seven different agents have been trained with the exact same setup. Each agent have been trained for 2000 episodes. For each episode, the environment choses a random wave phase, Hs, Tp, and Dp using the boundaries stated in Section 1.3. The best performing agent has been chosen by getting the peak of moving window average of reward where the window size is 100. The parameters of the environment can be seen in Table 4.1.

The exact tools used for the environment has been explained under the Section 1.3.

Table 4.1 : Environment setup.

Parameter	Value	Unit
Sample time (dt)	0.2	seconds
Initial rudder angle	0	degrees
Initial heading	0	degrees
Initial body fixed forward velocity	10.29	m/s
Hs	random	m
Tp	random	s
Dp	random	degrees

4.2.2 Reward function #1 - results

The trained agents have been evaluated the same way with how the designed LQR has been evaluated in Section 2. The overall performance of the seven agents have been compared to LQR in Figure 4.5. Note that for the RL agents, the error bars *do not* indicate the standard deviation of different runs, but it indicates the standard deviation

Table 4.2 : SAC hyperparameters.

Parameter	Value
optimizer	Adam [22]
learning rate	3×10^{-4}
discount	0.9
replay buffer size	5×10^{4}
number of hidden layers	2
number of neurons per layer	64
number of samples per minipatch	256
activation function	tanh
target smoothing coefficient	5×10^{-3}
target update interval	1
gradient steps	1

of *average performance* off all seven agents. So the standard deviation information for different evaluations on the same seadirection have been lost in this figure.

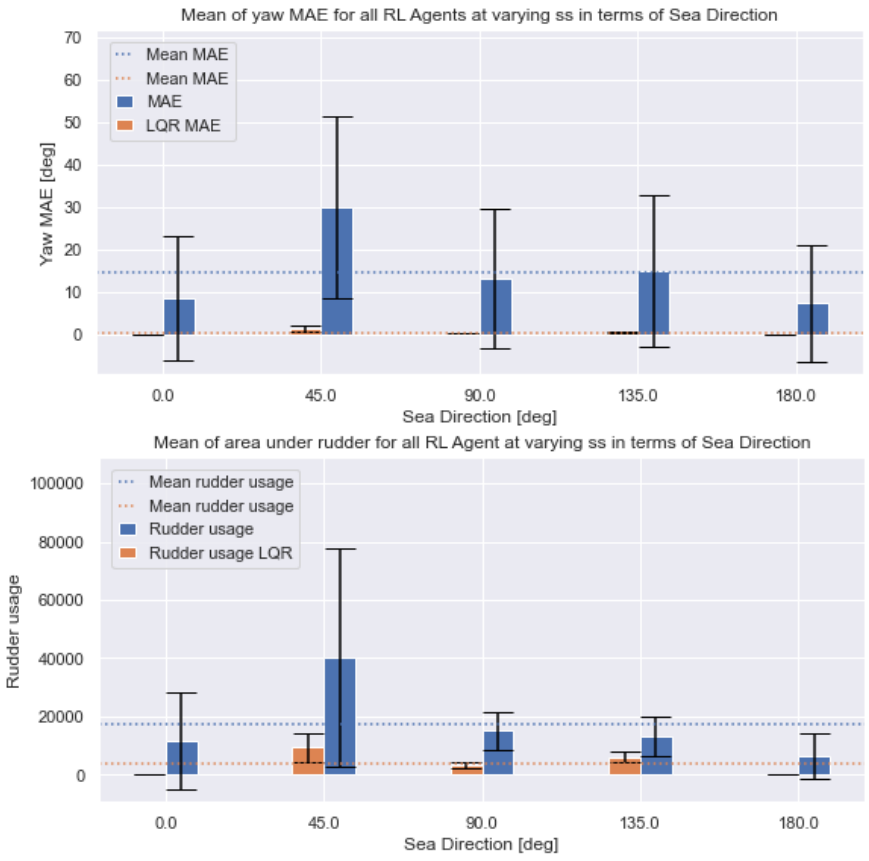

Figure 4.5 : The performance of agents trained on the first reward function, compared to LQR.

When this figure is observed, it can be clearly seen that this reward function can not , produce agnets that can deal with this problem case consistently. And because of how weak some of the agents were in these evaluations (see Figure 4.6), this figure doesn't give us a lot of information about what can be done to be able to improve the agent.

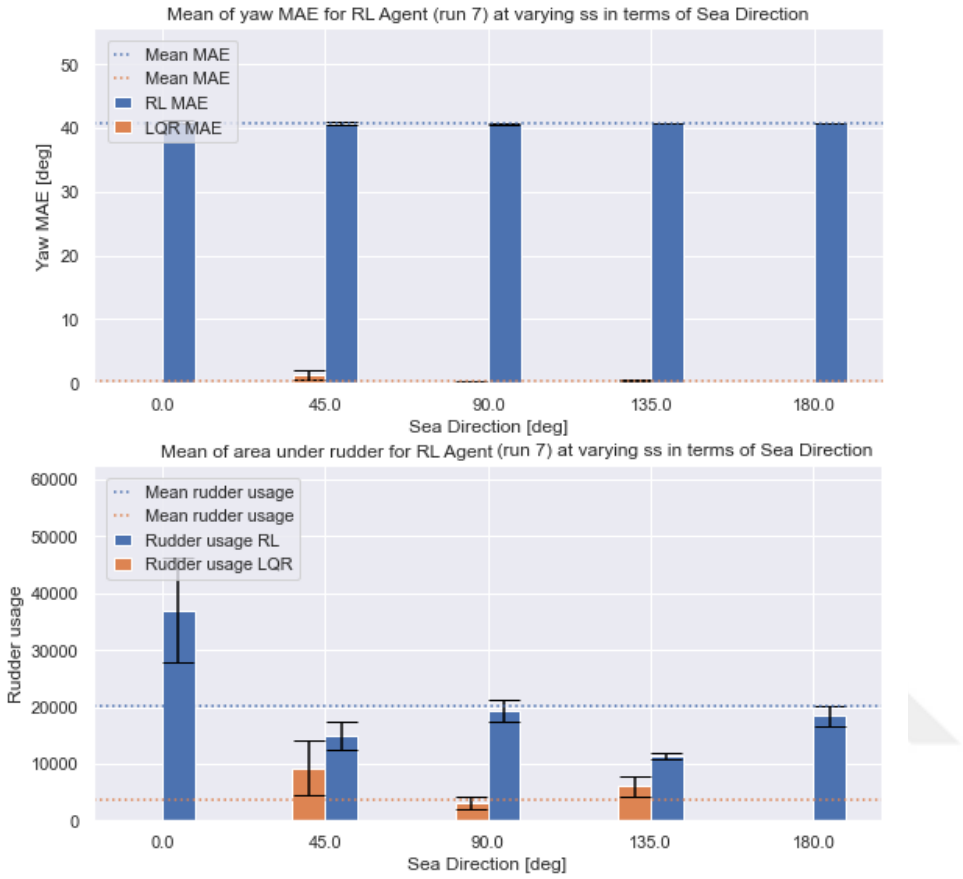

Figure 4.6 : The weakest agent produced by the first reward function.

To be able to get an idea on what to do next, let us observe the same plot, but this time comparing LQR and the best agent we have in terms of yaw MAE (Figure 4.7). Note that in this figure, just like LQR, the error bars show the standard deviation of performance for different the evaluations in a wave direction.

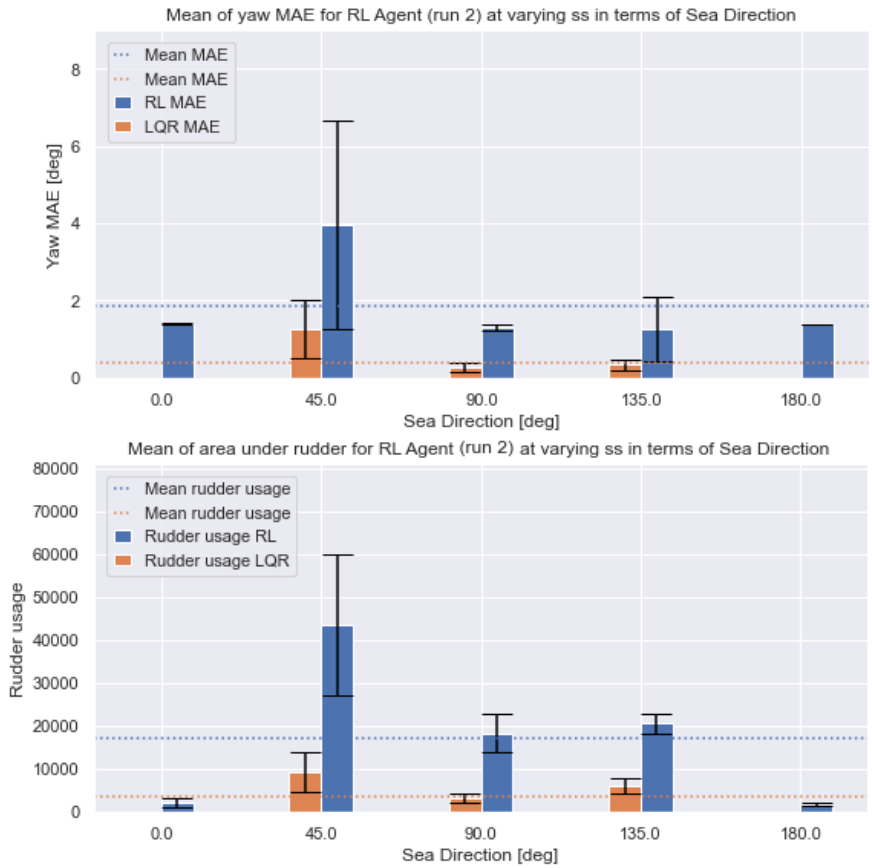

Figure 4.7 : The performance of a specific agent trained on the first reward function, compared to LQR.

When this figure is observed, it can be seen that we have at least one RL agent that can actually somewhat do an okay job at heading keeping even though it is weaker than LQR. Both the RL agent and LQR seems to be struggling in the same wave directions. One important thing to note here is that the agent isn't able to understand that it can do heading keeping with minimal rudder effort with waves that come from 0 degrees and 180 degrees.

Now let us observe some time traces to understand what the agent is struggling with. The study has detected three main weaknesses with the behavior of the agent; steady-state error, excessive rudder usage, and overshooting.

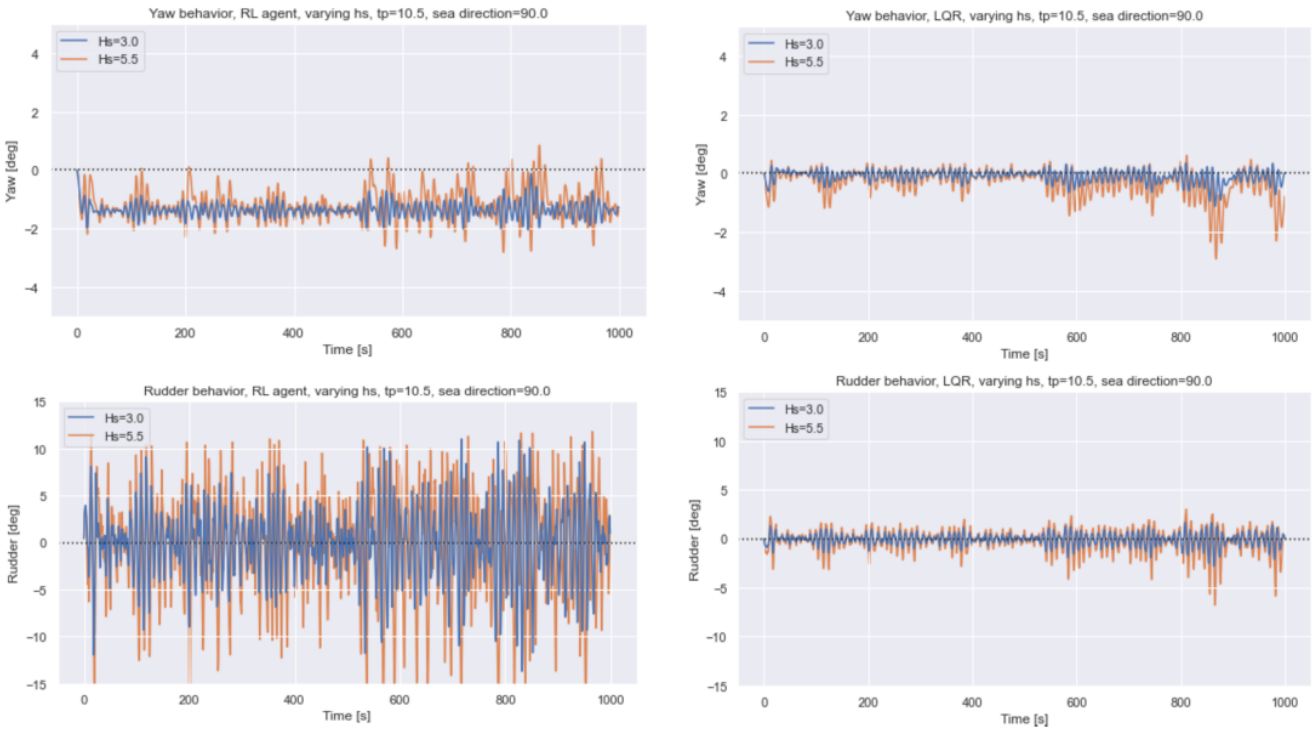

Figure 4.8 : Best agent and LQR under some specific sea conditions: excessive rudder usage and steady-state error.

Figure 4.8 compares the behavior of the best RL agent (again in terms of yaw MAE) and LQR under some specific sea condition. When the behavior of the RL agent is compared to LQR, it can be clearly seen that the agent uses the rudder a lot, and has a steady-state error.

When we observe Figure 4.9, the overshooting behavior can be seen.

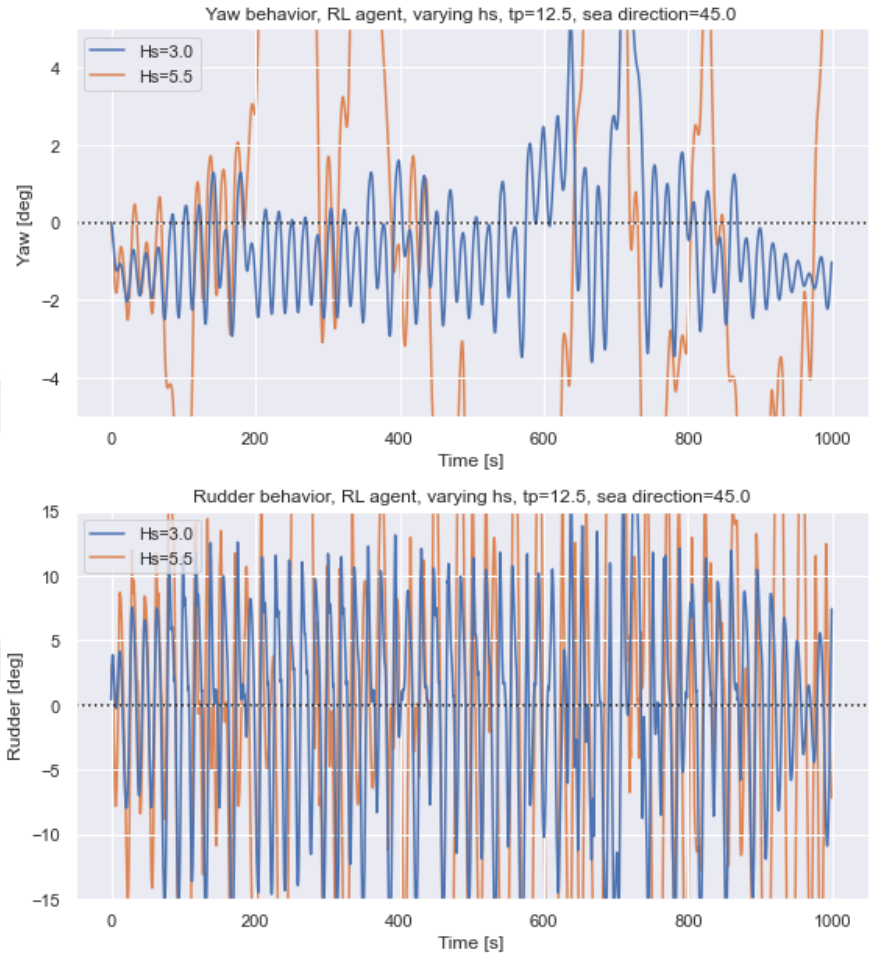

Figure 4.9 : The performance of agents trained on the first reward function, compared to LQR.

4.3 Reward Function #2

This iteration of the reward function will only focus on overshooting behavior and excessive rudder usage. To fight the these two problems, the reward function has been split into two sections; the positive reward signal and the negative reward signal. While the positive reward signal is used for encouraging low errors, the negative reward signal is used for discouraging the mentioned two problematic behavior.

First of all, let us define our reward function as follows

$$R = R_{encourage} - R_{punish} \tag{4.7}$$

For this iteration of the reward function, let us use the previous reward function as $R_{encourage}$

$$R_{encourage} = c_{scale} exp\left(-c_{err\_scale}|\psi_{err}|\right) \tag{4.8}$$

Now for the R_{punish}. Let us split R_{punish} into two parts

$$R_{punish} = R_{rudder} + R_{overshoot} \tag{4.9}$$

where R_{rudder} punishes the excessive rudder usage and $R_{overshoot}$ punishes the overshooting behavior. To fight with excessive rudder usage, rudder angle and the rudder rate can be punished

$$R_{rudder} = c_{rudder} R_{encourage} \delta + c_{rudder\_rate} \dot{\delta} \tag{4.10}$$

where c_{rudder} and c_{rudder_rate} are used for deciding the strength pf the punishment, δ is the rudder angle and $\dot{\delta}$ is the rudder rate. For now, ignore the $R_{encourage}$ term as it will be explained shortly.

To fight the overshooting behavior, the error rate can be punished. This would work the same way the D term works in a PID controller. Yaw rate could be considered as the error rate for our problem.

$$R_{overshoot} = c_{yaw\_rate} R_{encourage} \dot{\psi} \tag{4.11}$$

where $\dot{\psi}$ is the yaw rate, and c_{yaw_rate} is the punishment scale.

Now let us explain the $R_{encourage}$ encourage term in some of the punishment components. When we think about for example rudder rate, it is always a good idea to punish it since excessive rudder rate can damage the actuators and generally speaking won't be that useful. However, for something such as rudder angle, it is not alway good to punish it since we would like to use high rudder angles when the error is high. The same goes for the yaw rate too, we would like to have high yaw rates when the yaw error is high. Considering this, negative reward that comes from these two properties has been scaled by $R_{encourage}$. When the error is high, high rudder and yaw rate will not be punished that much since $R_{encourage}$ will be almost zero. But when the error is low, the increase in $R_{encourage}$ will make it so that the punishment that comes from the rudder angle and yaw rate is high.

Let us observe the effect how these punishments affect the shape of the reward function. In Figure 4.10, an example version of this reward function has been demonstrated when the rudder rate is high but rudder angle and yaw rate are zero. It can be seen that the entire function shifts down in this case.

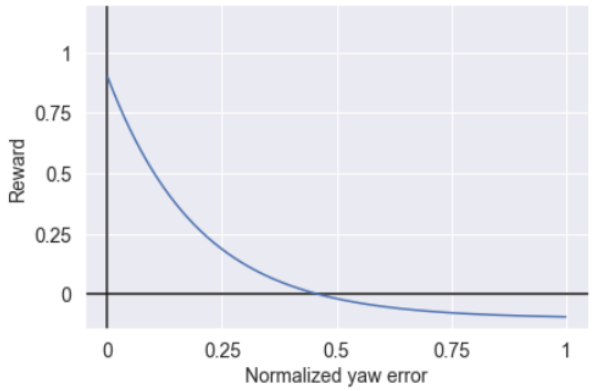

Figure 4.10 : The reward function after rudder rate punishment.

In Figure 4.11, an example version of this reward function has been demonstrated when the rudder angle is high but rudder rate and yaw rate are zero. It can be seen that the agent isn't being punished for having a high rudder angle when the error is high.

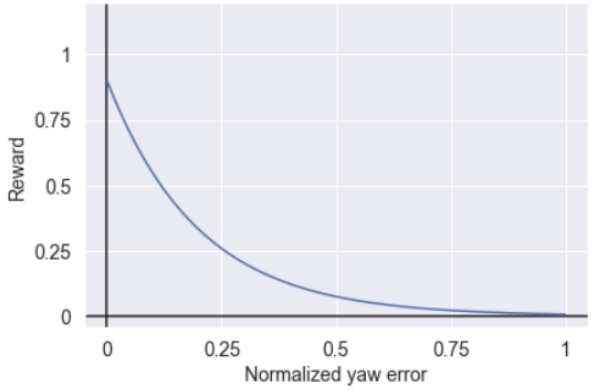

Figure 4.11 : The reward function after rudder punishment.

4.3.1 Reward function #2 -results

The reward function designed in section has been used. The reward parameters has been chosen as $c_{yaw\_rate} = 0.1$, $c_{rudder\_rate} = 0.3$, and $c_{rudder} = 0.1$ after a grid search. The agents have been trained the exact way with the previous ones. The overall performance of these agents have been compared to LQR in Figure 4.12. Again, for this plot the standard deviation information for different runs have been lost.

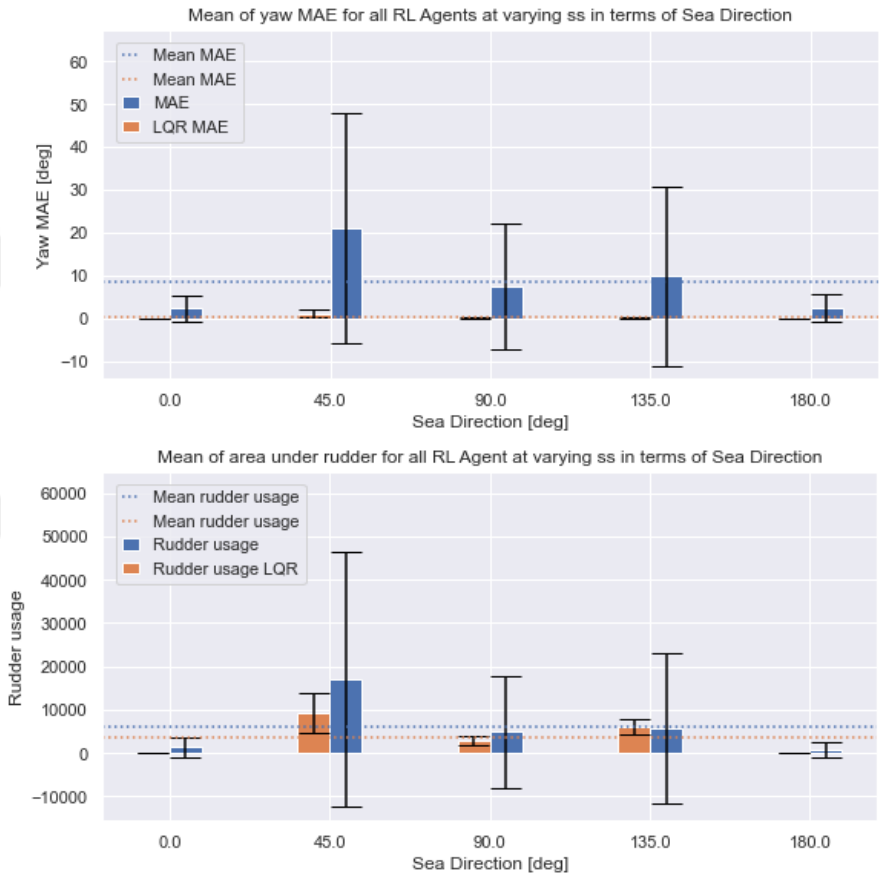

Figure 4.12 : The performance of agents trained on the second reward function, compared to LQR.

When compared to Figure 4.5, it can be seen that the agents have improved a lot in terms of rudder usage. When we look at the best agent in terms of the yaw MAE (Figure 4.13), we surprisingly see that the agent beats LQR in terms of rudder usage.

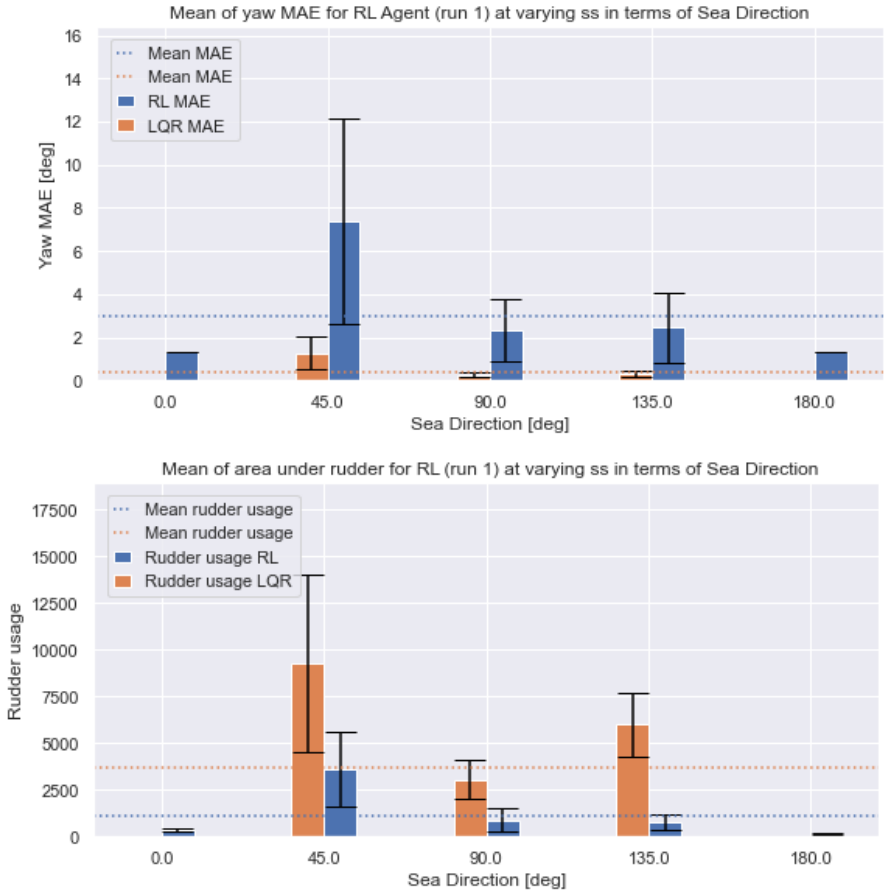

Figure 4.13 : The best agent (in terms of yaw MAE) beats LQR in terms of rudder.

However, when we look at the time traces (Figure 4.14), it can be seen that the problem with the steady state error persists as it hasn't been addressed by the changes.

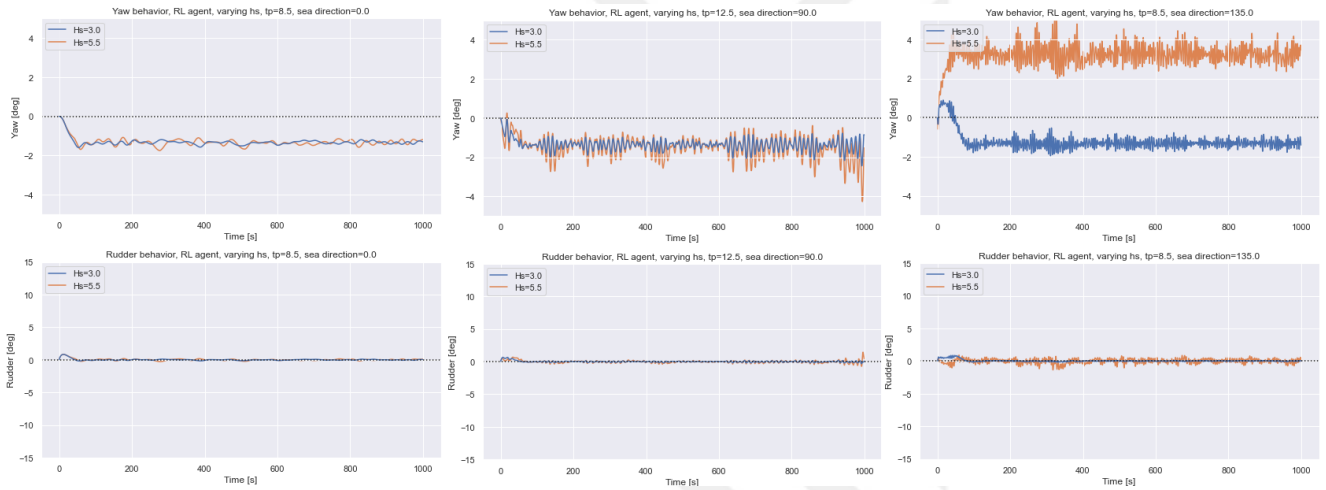

Figure 4.14 : Best agent and LQR under some specific sea conditions: steady-state error only.

4.4 Reward Function #3

To fight the steady state error, the gradient in $R_{encourage}$ around 0 degree error has been increased drastically by turning it into a joint function. The newly designed $R_{encourage}$ consists of two different exponential function where the stronger exponential function is chosen when the error is less then some predetermined value.

$$R_{encourage} = \begin{cases} exp\left(-c_1 \psi_{err}\right), & \psi_{err} < \psi_{border} \\ r_{border} exp\left(-c_2(\psi_{err} - \psi_{border})\right) & otherwise \end{cases} \tag{4.12}$$

where c_1 is the error scale for the low errors, c_2 is the error scale for the higher errors, r_{border} is where the the transition between two functions will occur in reward axis and ψ_{border} is where the transition will occur in the error axis.

Looking at the scale of the steady-state error in different runs, ψ_{border} has been chosen as 5 degrees. To make the gradient as drastic as possible, r_{border} has been chosen as 0.5 which effectively means that the agent isn't able to get the 50% of maximum possible error if it doesn't have less than $\psi_{border} = 5^\circ$ of error. c_2 has been chosen as 5 since it is the value used for the previous reward functions. After deciding these parameters, c_1 can be calculated through

$$c_1 = \frac{ln(r_{border})}{\psi_{border}} \tag{4.13}$$

Using the final iteration of the reward function new agents have been trained. The same reward parameters as the previous one has been chosen after a grid search. The agents have been trained the exact way with the previous ones. The comparison of overall performance of the new agents and LQR can be seen in Figure 4.15.

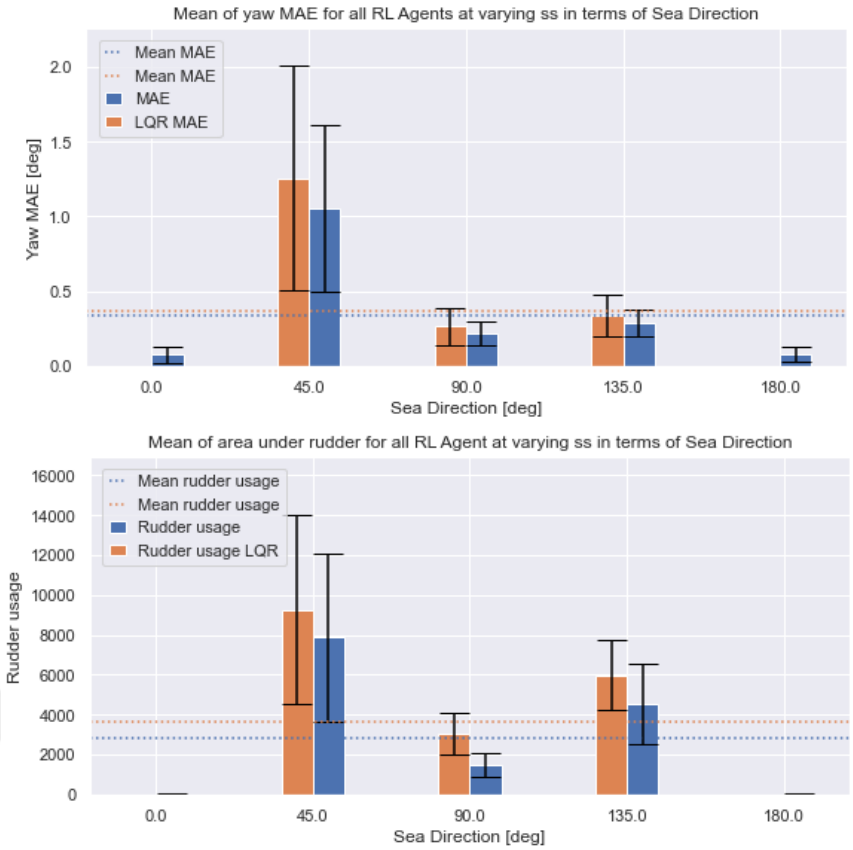

Figure 4.15 : The final agents compared to LQR.

Looking at this figure, it can be seen that the agent outperforms LQR both in terms of rudder usage and yaw MAE. If we try to increase the R value of the LQR in an attempts to beat the agent in terms of rudder usage, the gap between the LQR and RL agent in MAE increases (Figure 4.16). If we try to decrease the R to beat RL in terms of MAE, the rudder usage of LQR increases drastically (Figure 4.17).

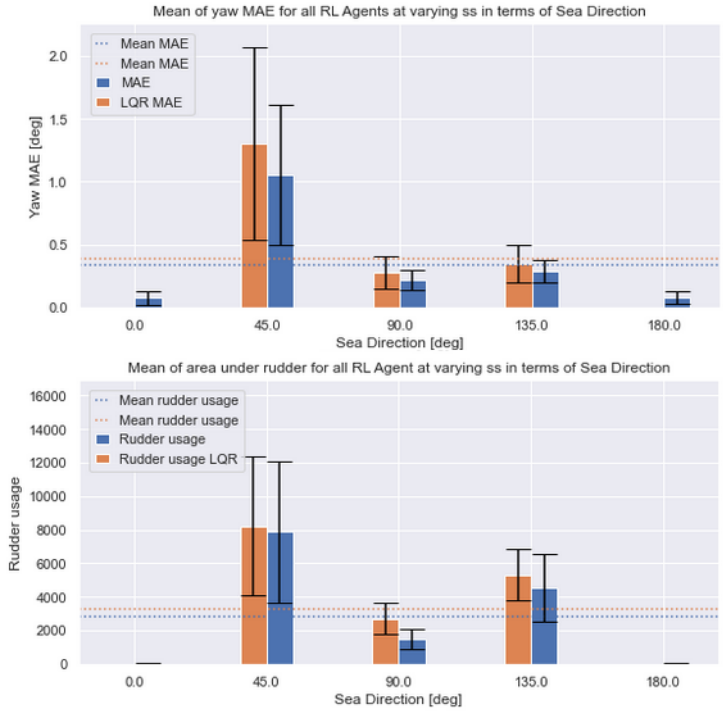

Figure 4.16 : The final RL compared to an LQR with a high *R*.

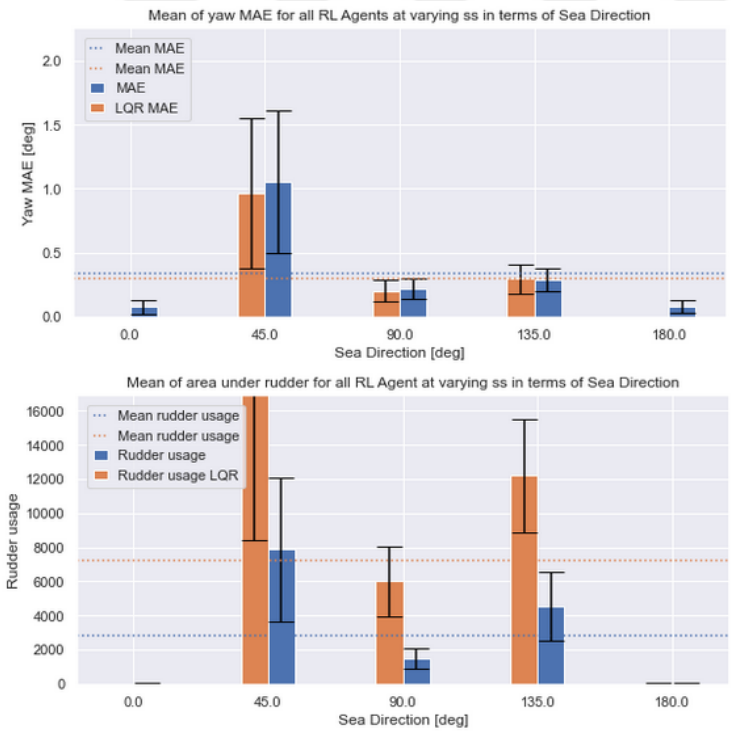

Figure 4.17 : The final RL compared to an LQR with a low *R*.

When time traces are observed, one of the strength that RL has over LQR seems to be that RL can have an overall low frequency rudder trend and have a high frequency

behavior on top of that. Whereas LQR seems to have a high frequency, really steady, periodic behavior. This gives an edge to RL over LQR in some of the complicated circumstances. Some examples of this can be observed in Figure 4.18 and especially with Figure 4.19.

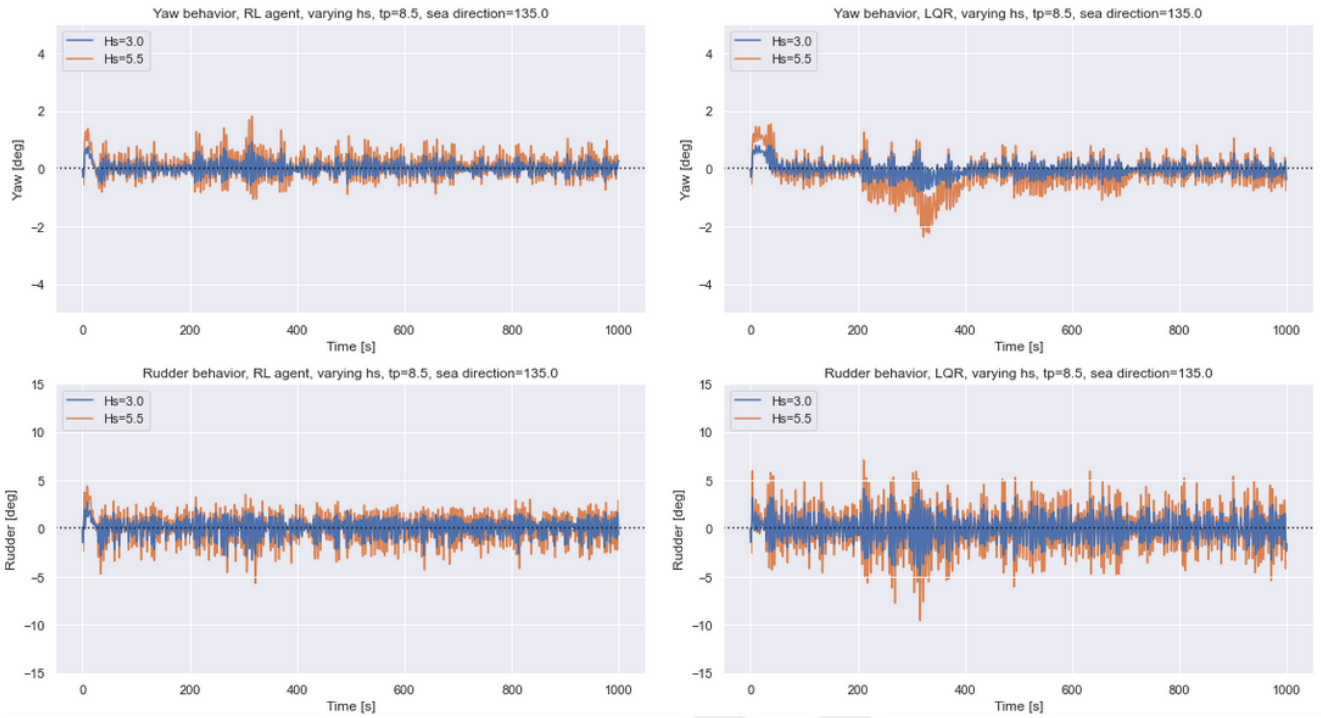

Figure 4.18 : RL is able to have a high frequency behavior over low frequency trend - first example.

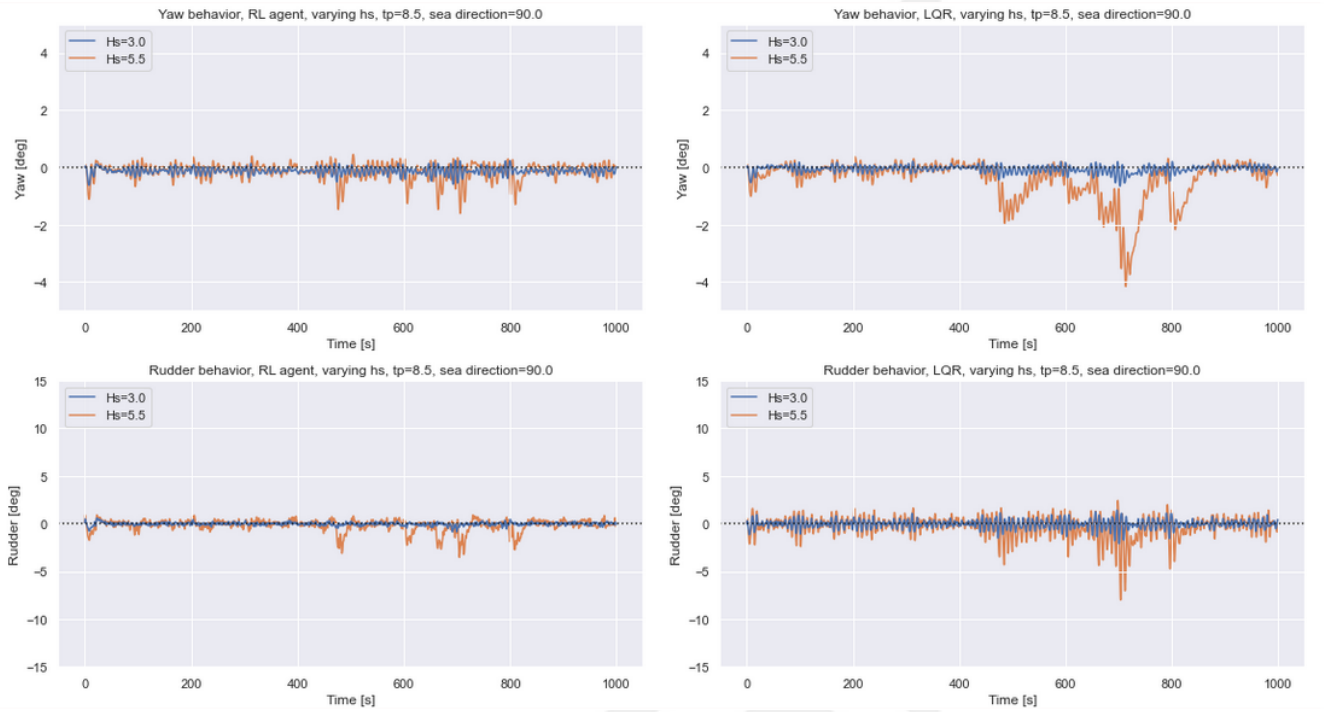

Figure 4.19 : RL is able to have a high frequency behavior over low frequency trend - second example.

5. CONCLUSION

In this thesis, an RL-based controller has been proposed for the heading keeping of a ship. The potential benefits of using an RL-based controller over a traditional one have been discussed. To see if the designed RL-based controller can actually be compared to a traditional controller, an LQR controller has been designed.

After giving an extensive overview of RL methods and explaining the specific algorithm that the study uses, the thesis starts off by creating a simple RL agent, and iteratively improves it. In each iteration of the improvements, the RL agents have been compared to the designed LQR in terms of yaw MAE and rudder usage. The final iteration of the RL has been seen to beat the designed LQR both in terms of yaw MAE and rudder usage.

To conclude all the improvements, one evaluation of all the designed controllers has been visualized in Figure 5.1. When this figure is observed, it can be seen that the first iteration of the RL agent has unnecessarily high rudder usage and isn't able to beat a steady-state error. In different runs, it also has been seen that this agent has an overshooting behavior. The next iteration of the agent attempts to fight the rudder usage and overshooting problem and manages to succeed. However in this iteration, the steady-state error becomes much more prevalent. The final iteration of the RL agents manage successfully beat the steady-state error issue without losing the advantages of the previous iteration.

While still an emerging field, the use of machine learning methods in marine movement control shows promising results. This study designs an machine learning based controller to check the feasibility of such controllers and attempts to provide guidance for possible future work.

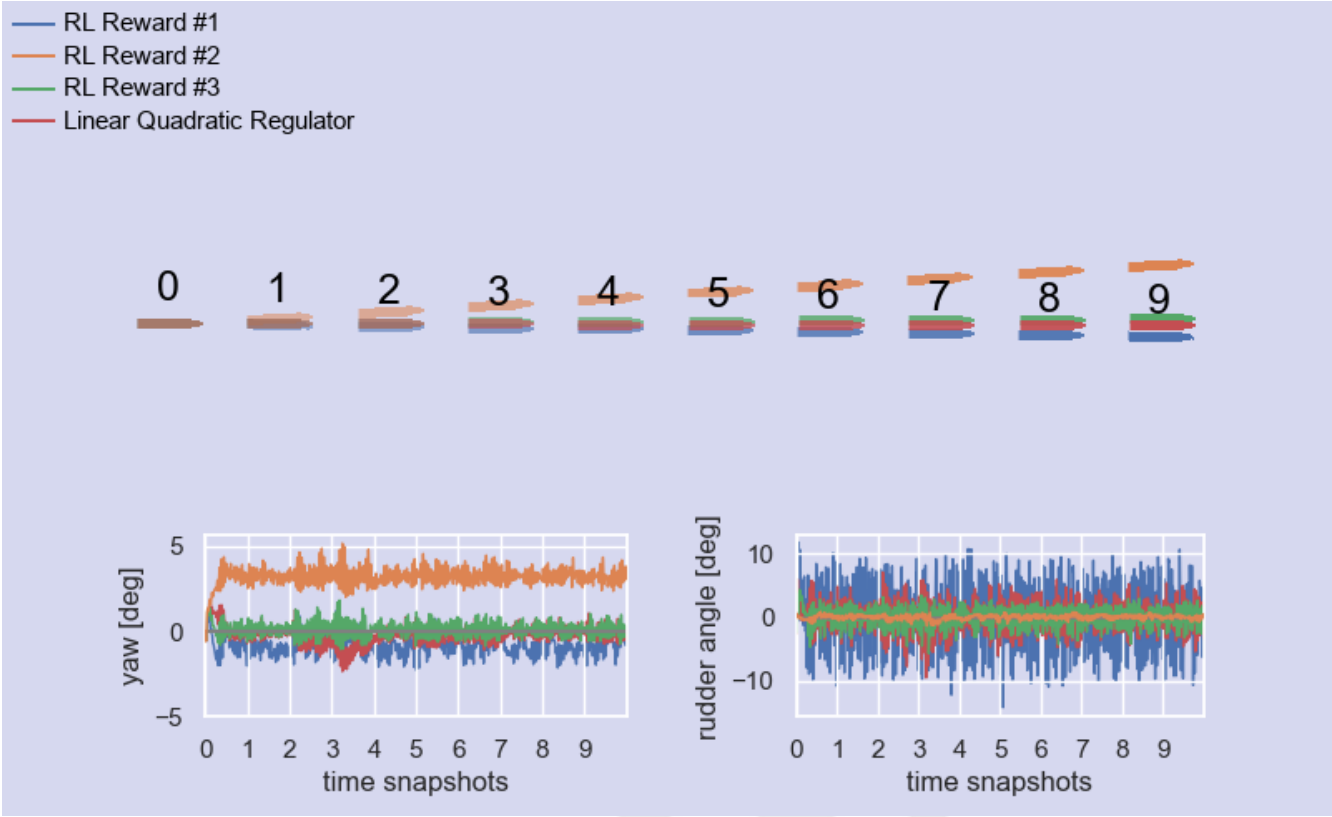

Figure 5.1 : RL is able to have a high frequency behavior over low frequency trend - second example.

5.1 Future Work

5.1.1 Reward shape

This study only studies reward functions with an exponential behavior. While the high gradient around zero error might be a desirable characteristic to fight steady-state error as shown by this study, it can be valuable to investigate reward functions of different shapes.

5.1.2 Drift reduction

The current reward function doesn't have any terms for reducing drift. A possible future work could involve adding a term for drift reduction to the reward function.

5.1.3 An agent for varying propeller RPM

The current agent only works on one specific propeller RPM. The agent could be further generalized by making it able to work for different propeller RPMs. One simple attempt could be to simply train an agent which has the exact same configuration as the last developed agent except it also has RPM in its state vector.

5.1.4 RL in nonlinear waves

In this study, the used wave model is linear. Considering how well the neural network based learners perform in nonlinear environments, the performance difference between RL and traditional controllers can potentially increase a lot if a nonlinear wave model is used.

Considering how computationally expensive it is to simulate nonlinear environments compared to linear environments, the author decided to use linear waves for this study. However now that there is already a working model, it is much more feasible to test and improve this agency configuration in a nonlinear environment within a reasonable

amount of time. In fact, the same trained models could be taken and trained for a while in nonlinear waves.

REFERENCES

[1] **Sutton, A.S. and Barto., A.G.** (2018). *Reinforcement Learning: An Introduction. MIT Press, Cambridge*, MIT Press, Cambridge, MA.

[2] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Rusu, A.**, **Veness, J.**, **Bellemare, M.**, **Graves, A.**, **Riedmiller, M.**, **Fidjeland, A.**, **Ostrovski, G.**, **Petersen, S.**, **Beattie, C.**, **Sadik, A.**, **Antonoglou, I.**, **King, H.**, **Kumaran, D.**, **Wierstra, D.**, **Legg, S. and Hassabis, D.** (2015). Human-level control through deep reinforcement learning, *Nature*, *518*, 529–33.

[3] **Silver, D.**, **Hubert, T.**, **Schrittwieser, J.**, **Antonoglou, I.**, **Lai, M.**, **Guez, A.**, **Lanctot, M.**, **Sifre, L.**, **Kumaran, D.**, **Graepel, T.**, **Lillicrap, T.**, **Simonyan, K. and Hassabis, D.** (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, *Science*, *362*, 1140–1144.

[4] **Levine, S.**, **Finn, C.**, **Darrell, T. and Abbeel, P.** (2015). End-to-End Training of Deep Visuomotor Policies, *17*.

[5] **Deng, Y.**, **Bao, F.**, **Kong, Y.**, **Ren, Z. and Dai, Q.** (2016). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading, *IEEE Transactions on Neural Networks and Learning Systems*, *28*, 1–12.

[6] **Wang, H.**, **Yang, J.**, **Lee, H.S. and Han, S.** (2018). *Learning to Design Circuits*.

[7] **Øvereng, S.S.**, **Nguyen, D.T. and Hamre, G.** (2021). Dynamic Positioning using Deep Reinforcement Learning, *Ocean Engineering*, *235*, 109433, `https://www.sciencedirect.com/science/article/pii/S0029801821008398`.

[8] **Martinsen, A.B. and Lekkas, A.** (2018). Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning, volume 51, pp.329–334.

[9] **Fossen, T.** (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*.

[10] **Tannuri, E.**, **Kubota, L. and Pesce, C.** (2006). Adaptive techniques applied to offshore dynamic positioning systems, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *28*.

[11] **Xu, S.**, **Yang, J. and Wang, L.** (2019). A Fuzzy Rule Based PID Controller for Dynamic Positioning of Vessels in Variable Environmental Disturbances, *Journal of Marine Science and Technology*.

[12] **Øvereng, S.S.**, **Nguyen, D.T. and Hamre, G.** (2021). Dynamic Positioning using Deep Reinforcement Learning, *Ocean Engineering*, *235*, 109433.

[13] **Martinsen, A.B. and Lekkas, A.M.** (2018). Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning, *IFAC-PapersOnLine*, *51*(29), 329–334, 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.

[14] **Mak, B. and Duz, B.** (2019). SHIP AS A WAVE BUOY - USING SIMULATED DATA TO TRAIN NEURAL NETWORKS FOR REAL TIME ESTIMATION OF RELATIVE WAVE DIRECTION.

[15] *US Navy Combatant, DTMB 5415*, `http://www.simman2008.dk/5415/combatant.html`, accessed: 2022-04-15.

[16] *SAC*, `https://stable-baselines.readthedocs.io/en/master/modules/sac.html`, accessed: 2022-04-16.

[17] *Gym documentation*, `https://www.gymlibrary.ml/`, accessed: 2022-04-16.

[18] **Kwakernaak, H. and Sivan, R.** (1972). *Linear Optimal Control Systems*, John Wiley & Sons, Inc., USA.

[19] **Williams, R.J.** (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Reinforcement Learning*, 5–32.

[20] **Haarnoja, T.**, **Zhou, A.**, **Abbeel, P. and Levine, S.** (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, `https://arxiv.org/abs/1801.01290`.

[21] **Meyer, E.**, **Heiberg, A.**, **Rasheed, A. and San, O.** (2020). COLREG-Compliant Collision Avoidance for Unmanned Surface Vehicle Using Deep Reinforcement Learning, *IEEE Access*, *8*, 165344–165364.

[22] **Kingma, D.P. and Ba, J.** (2014). *Adam: A Method for Stochastic Optimization*, `https://arxiv.org/abs/1412.6980`.

CURRICULUM VITAE

Name: Afşin Baran BAYEZİT

EDUCATION:

- **B.Sc.** 2019, Yeditepe University, Engineering Faculty, Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2017, ITU Model-based Design and Control Lab, Intern, Embedded programming for implementations of different control algorithms on a quadcopter
- 2017, Yeditepe University Computer Science Lab, Research Volunteer, Working on the design of a multi-core processor under Sezer Gören Uğurdar. Mainly test benches
- 2017-2018, ITU Model-based Design and Control Lab, Intern, Embedded programming support for implementations of different control algorithms on a quadcopter
- 2018-2019, ITU Model-based Design and Control Lab, Intern, Embedded programming for implementations of different control algorithms on a reaction wheel
- 2019, graduation from the Computer Engineering undergraduate program of Yeditepe University
- 2019-2021, TÜBİTAK 1001 Project in ITU, Project Member, Embedded programming on a model-sized ship that is designed for hydroacoustic experiments
- 2021-2022, Maritime Research Institute of Netherlands, Intern, Designing a Reinforcement Learning based controller for course keeping

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Omer Kemal Kinaci, Cihad Delen, Rahman Bitirgen, Afsin Baran Bayezit, Ismail Bayezit, Deniz Ozturk, Burak Gunguder** (2021). Free-running tests for DTC self-propulsion – An investigation of lateral forces due to the rudder and the propeller. *Applied Ocean Research*, November 2021