X International Conference on Computational Methods in Marine Engineering
MARINE 2023
J. García-Espinosa, L. González, J.E. Gutiérrez & B. Serván-Camas (Eds)

A Generalized Reinforcement Learning Based Controller for Course-keeping of Ships in Waves

Afsin B. Bayezit[1*], Omer K. Kinaci[*], Bulent Duz[†], Douwe Rijpkema[†] and Bart Mak[†]

[*] Istanbul Technical University (ITU), İTÜ Ayazağa Kampüsü, 34469, Maslak, İstanbul, Türkiye, Web page: https://www.itu.edu.tr/

[†] Maritime Research Institute Netherlands (MARIN), Haagsteeg 2, 6708 PM, Wageningen, Gelderland, Netherlands Web page: https://www.marin.nl/

[1] Corresponding author: Afsin B. Bayezit, bayezit20@itu.edu.tr

ABSTRACT

In the maritime industry, traditional controllers such as PID are commonly used for ship motion control. However, recent research has demonstrated the efficacy of Reinforcement Learning methods in decision-making tasks. Considering the power of Machine Learning methods when dealing with highly non-linear problems, the use of Reinforcement Learning for ship motion control problems has started to attract more attention in recent studies. As such, this study proposes a Reinforcement Learning based controller for ship course-keeping in waves, which can operate under a range of sea conditions. Instead of directly showing the final version of the controller, we present a weak controller as a starting point and iteratively improve it through reward shaping. Our final controller outperforms the baseline LQR we use in terms of yaw error and rudder usage. We finalize the paper by presenting the behavioral differences of the designed controller and LQR. Our findings demonstrate the potential of RL for ship motion control, offering a promising alternative to traditional handcrafted methods.

Keywords: Course-keeping; Reinforcement Learning; Soft Actor-Critic; Reward shaping; Artificial intelligence; Ship motion control.

NOMENCLATURE

A_t	Action at time t
S_t	State at time t
R_t	Reward at time t
$\mathcal{A}$	Action space
$\mathcal{S}$	State space
$\mathcal{R}$	Reward space
π	Policy
π^*	The optimal policy
G_t	Expected return after time step t [-]
γ	Discount factor [-]
H	Entropy term [-]
β_w	Wave direction [deg]
$H_{1/3}$	Significant wave height [m]
T_p	Wave period [s]
δ	Rudder angle [deg]
$\dot{\delta}$	Rudder rate [deg/s]
ψ	Yaw angle [deg]
$\dot{\psi}$	Yaw rate [deg/s]
$\ddot{\psi}$	Yaw acceleration [deg/s^2]
ψ_{err}	Yaw error [deg]
ML	Machine Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
SAC	Soft Actor-Critic
PID Controller	Proportional-Integral-Derivative Controller
LQR	Linear Quadratic Regulator

1. INTRODUCTION

Reinforcement Learning (RL) is a subbranch of Machine Learning (ML) where the learner (agent) tries to map a state space to an action space in a way that maximizes some reward. RL has been successfully applied to a variety of decision-making tasks, ranging from complex board games like chess (Silver et al., 2017) to integrated circuit design (Wang et al., 2020).

In control problems, RL has shown promise as a model-free approach that does not rely on knowledge of the system's dynamics. Instead, it learns how to act through trial-and-error, making it well-suited for problems with nonlinear or unknown dynamics. Despite these potential benefits, the maritime industry continues to rely on traditional controllers, such as PID, for ship motion control.

While the use of ML in maritime domain is in its infancy, there are some works in the literature that investigate the feasibility of RL for ship motion control. Øvereng et al. (2021) tackles the problem of dynamic positioning using Deep RL (DRL) and shows that their agents can perform in real sea. Martinsen & Lekkas (2018) demonstrates that DRL agents can be successfully used for path following and collision avoidance.

Our study proposes an RL-based controller for course-keeping of ships in waves, using the state-of-the-

Figure 1: Profile and back views of 5415M frigate.

art Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018). We train an RL agent that can operate under a range of environmental conditions, where wave height, period, and direction may vary. To evaluate our agent's performance, we compare it to a Linear Quadratic Regulator (LQR) controller (Franklin et al., 1994) in terms of yaw error and rudder usage.

Rather than presenting the final agent directly, we start with a weaker version and iteratively improve it by addressing issues such as excessive actuator usage, overshooting, and steady-state error through reward shaping. In Section 2, we formally define the problem case, and in Section 3, we provide an overview of RL and explain the key features of SAC. We then present the design of the initial agent iteration and discuss the results in Section 4, showing how different issues were addressed through reward engineering. We also compare the final agent's performance and behavior with LQR. Finally, we conclude the paper with Section 5.

2. PROBLEM DEFINITION

The course-keeping problem involves maintaining a ship's course while minimizing rudder usage. For this study, we consider a ship operating in an environment with waves of following properties:

- any wave direction β_w,
- wave heights within $1.5m \leq H_{1/3} \leq 5.5m$ and
- wave periods within $8.5s \leq T_p \leq 12.5s$.

We use the 5415M frigate in our study. It is based on the public bare hull form DTMB 5415 (Simman 2008, n.d.), which has been widely used in previous research. Unlike the bare hull form 5415, 5415M has appendages such as bilge keels, stabilizers, rudders, propellers, shafts, and supportive struts. Table 1 presents the main particulars of the 5415M frigate, while Figure 1 shows a scaled model of the ship.

Table 1: Main particulars of DTMB 5415M.

Main Particular	Value (m)
L_{pp}	142
Draft aft	6.15
Draft fore	6.15
KG	71.51
GM	1.95
B_{WL}	19.06
k_{xx}	7.62
k_{yy}	35.5
k_{zz}	35.5

3. REINFORCEMENT LEARNING & CONTROLLER DESIGN

RL is a Machine Learning paradigm that deals with decision-making problems. A learner called the agent attempts to find an action-taking strategy (referred to as policy) that maximizes some reward.

Markov Decision Processes (MDPs) are a mathematical framework used to formalize Reinforcement Learning (RL) problems. An MDP consists of an agent, an environment, and their continuous interaction. The agent selects an action based on the current state, and the environment responds with a new state and a reward. The goal of the agent is to learn a policy that maximizes the expected return of reward over time (Sutton & Barto, 2018). The interaction between the agent and the environment in an MDP is illustrated in Figure 2.

Figure 2: Visualization of MDP.

To be more precise, the MDP can be described as follows. At each discrete time step t, the environment (anything outside the agent) can be in any state $S_t \in \mathcal{S}$. The agent observes the state S_t and chooses to perform some action $A_t \in \mathcal{A}$. After this interaction, the state transitions from the state S_t to some other state S_{t+1} and returns a reward $R_{t+1} \in \mathcal{R}(S_t, A_t)$. Through these interactions, the agent attempts to find some optimal policy π^* which maximizes the reward return G. Let us define G_t as the reward return following the time step t. When formulating G_t, instead of summing up all the reward return after time step t, the concept of discount factor is often used. This allows for the agent to prioritize immediate rewards more. The discount factor $0 \leq \gamma \leq 1$ can be reduced to make the agent more shortsighted or can be increased to make it more farsighted.

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

Practically speaking, it is not possible for an RL agent to find a policy that perfectly maximizes the return. This is why agents try to learn a policy that maximizes the expected return instead. Keeping this in mind, we could define a simple optimal policy as follows

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \underset{\tau \sim \pi}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] \tag{2}$$

where τ is the trajectory of state-action pairs.

3.1 Soft Actor-Critic

One of the main challenges of the RL problem is to find a balance between exploration and exploitation. An agent needs to exploit the information it already knows to be able to choose actions that return as much reward as possible. However, the agent also needs to explore the state space to be able to

find actions that return better rewards. Different RL algorithms use different methods in an attempt to find a good balance between exploration vs. exploitation in order to discover good policies.

One of the main features of the RL algorithm used in this paper, Soft Actor-Critic (SAC), is that it uses the concept of entropy. Within the context of SAC, the term entropy refers to the randomness of a policy. By maximizing a trade-off between reward and entropy, SAC tries to make sure that the agent is following a decent policy while not neglecting exploration. For SAC, the optimal policy can be formulated through

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \underset{\tau\sim\pi}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t (R_t + \alpha H_t) \right] \tag{3}$$

where H is the entropy term and $\alpha > 0$ is the trade-off coefficient. Through H, having a policy with a relatively high entropy is encouraged. SAC is an algorithm that manages to form a great balance between exploration vs. exploitation. It is sample-efficient and tolerant to changes in hyperparameters, which is why it has been chosen for this study.

3.2 Agent Design

When designing an RL agent (assuming that an existing RL algorithm is being used), the main workload of the design lies in finding a good state space and designing a good reward function. While one might use experience and intuition as guidance, there is not an algorithm for figuring out the optimal state space or the reward function for a given problem case. Thus, starting with a simple reward function and a state space, determining their weaknesses and iterating over them is a key part of the design process.

3.2.1 Action and State Space

The agent samples it's actions from the action space. Since for our problem case the agent is only allowed to use the rudder, the action vectors have chosen to have the following format

$$A_t = [n_{\dot{\delta}}] \tag{4}$$

where $n_{\dot{\delta}} \in [-1, 1]$ is the rudder rate in percentage.

The state space could be seen as the agent's perception of the environment. When determining the state space, it is important to make sure that the information in state vector is sufficient and not redundant. Additionally, the state vector should not include elements that are unfeasible to obtain in real-life scenarios. For example, in a simulation environment, the agent could be provided with the exact sea state information, and intuitively one would expect this information to be extremely useful. However, it is not easy to figure out the sea state in real-life applications. So sea state should not be added to the state vector without proposing a reliable method of obtaining it.

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

$$S_t = [\psi_{err}, \delta, \dot{\psi}, \ddot{\psi}] \tag{5}$$

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.

3.2.2 Reward Shaping

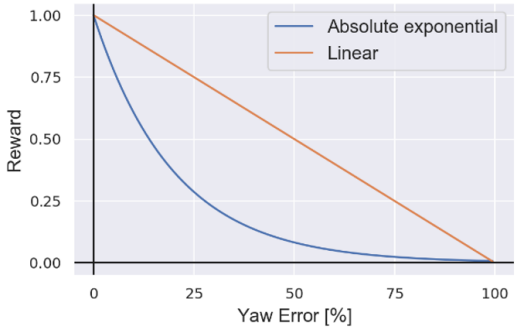
An agent conceptualizes the desirability of it's state through reward. So, by engineering a reward function that encourages good behavior and discourages bad behavior, one could expect to get a well-performing policy. In this study, we start with a simple reward function and it twice. In this section, we will only discuss the first iteration of the reward function.

The most straightforward reward function for our problem would be a linear function in which the yaw error is multiplied with some negative gain. However, we start off with the following absolute exponential function instead, in which the agent gets the most reward when it has zero yaw error

$$R = \exp(-c_\psi |\psi_{err}|) \tag{6}$$

where $\psi_{err} \in [0, 1]$ is the yaw error in percentage and c_ψ is the error scale. c_ψ can be used for adjusting the steepness function. The absolute exponential function has been inspired by Meyer et al. (2020), however we have chosen it for a different reason. Meyer et al. (2020) uses this function because they argue that having a lower slope when the error is high is more favorable for their problem case. We instead chose this function because we argue that having a strong gradient around zero error is helpful when dealing with steady-state error; an issue previous RL-based marine controllers struggled with (Meyer et al., 2020; Øvereng et al., 2021). As will be seen, the performance difference between the second agent and the third agent strongly supports this argument. The difference between the absolute exponential functions and linear functions have been demonstrated through some arbitrary functions in Figure 3.

Figure 3: Arbitrary linear and absolute exponential functions to demonstrate their differences. 100% yaw error corresponds to 180 degrees of yaw error.



4. SIMULATIONS & RESULTS

4.1 Training Setup

For the simulation environment, we use MARIN's in-house seakeeping and maneuvering software xSimulation (works similar to Quadvlieg & Rapuc, (2019)). The RL part of the simulations is handled through MARIN's MARINRL Python library. The library interfaces Stable Baselines' SAC agent (Hill et al., 2018) with xSimulation through a GYM (Brockman et al., 2016) environment, allowing RL-based controllers to be designed and evaluated easily. The environment setup and SAC hyperparameters can be seen in Table 2 and 3. Coefficients of each reward function have been tuned through a simple grid search.

Table 2: The simulation setup.

Parameter	Value	Unit
Sample time (dt)	0.2	s
Initial rudder angle	0	deg
Initial heading	0	deg
Initial surge velocity	10.29	m/s
$H_{1/3}$	random	m
T_p	random	s
β_w	random	deg

Table 3: Hyperparameters.

Parameter	Value
Optimizer	Adam (Kingma & Ba, 2014)
Learning rate	3×10^{-4}
Discount factor	0.9
Replay buffer size	5×10^{-4}
Hidden layers	2
Neurons per layer	64
Samples per minibatch	256
Activation function	tanh
Target smoothing coefficient	5×10^{-3}
Target smoothing interval	1
Gradient steps	1

For each reward function, seven agents have been trained for 1500 episodes where one episode length is 300 seconds and values of wave direction, height and period are randomly chosen.

4.2 Evaluation Method

For each training run, moving window average of rewards over the last 100 episodes has been observed. The version of the agent that achieves the highest peak in this average has been chosen as the best performer. These best performing agents then have been evaluated 210 simulations which has waves with; seven different phases, β_w of 0, 45, 90, 135, and 180 degrees, $H_{1/3}$ of 3 and 5.5 meters, T_p of 8.5, 10.5 and 12.5 seconds. For each wave combination, the agent behaviors have been simulated for 1000 seconds. The results have been evaluated in terms Mean Absolute Error (MAE) of yaw and the total

rudder usage.

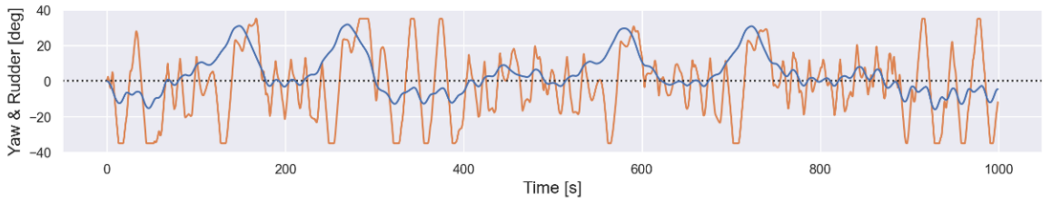
The obtained results also have been compared with an LQR controller. The designed LQR controller models the yaw response of the ship using a first order Nomoto model (Fossen, 2011). The performance of LQR's with different cost tunings have been demonstrated (by fixing the Q and changing R) to see how the RL agents perform compared to different LQR tunings.

4.3 The First Reward Function

Equation 6 has been used as the first reward function with $c_\psi = 5$.

This iteration of agents has not been able to converge to a well-behaving policy. Through the analysis of simulation results, it has been observed that this iteration of agents uses the rudder excessively and overshoot the zero error consistently. An example simulation which demonstrates both of these behaviors can be seen in Figure 4.

Figure 4: Yaw and rudder behavior in time domain for the first reward function demonstrated.



4.4 Fighting Rudder Abuse and Overshoots

Considering the problems in the previous iteration, three new terms have been added to the reward function. The new reward function can be seen below.

$$
\begin{aligned}
R &= R_\psi + R_\delta + R_\psi R_{\dot{\delta}} + R_\psi R_r \\
R_\psi &= exp(-c_\psi |\psi_{err}|) \\
R_\delta &= -c_\delta |\delta| \\
R_{\dot{\delta}} &= -c_{\dot{\delta}} |\dot{\delta}| \\
R_{\dot{\psi}} &= -c_{\dot{\psi}} |r|
\end{aligned}
\tag{7}
$$

Here, δ is the rudder angle, $\dot{\delta}$ is the rudder rate, $\dot{\psi}$ is the yaw rate and c_δ, $c_{\dot{\delta}}$ and $c_{\dot{\psi}}$ are the reward coefficients. The δ, $\dot{\delta}$ and $\dot{\psi}$ are normalized between zero and one to make the tuning more intuitive.

R_δ and $R_{\dot{\delta}}$ terms keep the actuator use reasonable whereas the $R_{\dot{\psi}}$ term, inspired by the D term of PID, fights overshoots. R_δ and $R_{\dot{\psi}}$ terms are scaled by the R_ψ to reduce the punishment of high δ and $\dot{\psi}$ when the error is high. The effects of the punishment terms on the reward have been demonstrated on Figure 5.

The reward coefficients have been chosen as $c_\delta = 0.1$, $c_{\dot{\delta}} = 0.3$ and $c_{\dot{\psi}} = 0.1$ and the previous coefficient has been kept the same. The agents trained with this reward function have been able to learn a policy that has acceptable performance. However, they were no match to LQR mainly due to a persistent steady-state error issue which can be seen on an example simulation in Figure 6.

Figure 5: Effects of punishment on reward with arbitrary coefficients. Rudder rate is punished even when the error is high whereas the rudder punishment has no visible effect on the reward until the error gets low. Yaw rate punishment is similar to the rudder punishment.

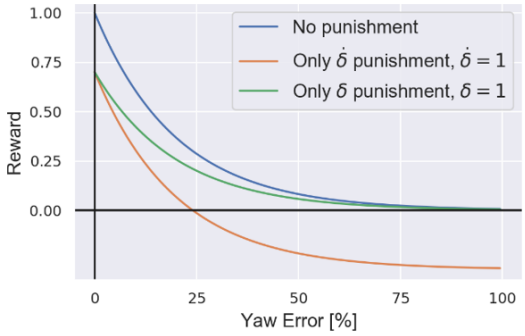
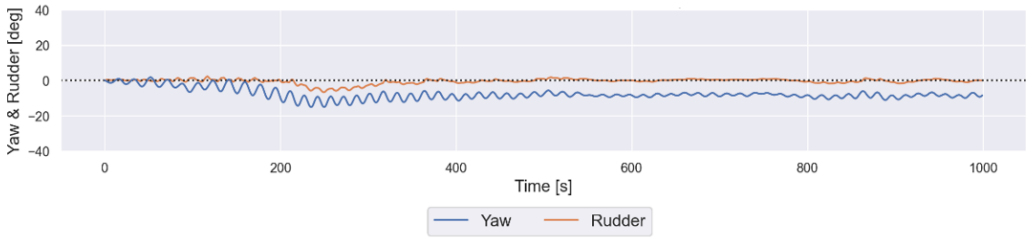


Figure 6: Yaw and rudder behavior in time domain for the second reward function demonstrated.



4.5 Fixing Steady-state Error

To address the steady-state error, the gradient around zero error has been made even stronger. This has been achieved by splitting R_ψ into two parts for lower and higher errors as follows

$$R_\psi = \begin{cases} \exp(-c_{\psi low}|\psi_{err}|), \text{for } \psi_{err} \leq \psi_{border} \\ r_{border} \exp(-c_{\psi high}|\psi_{err} - \psi_{border}|), \text{otherwise} \end{cases} \tag{8}$$

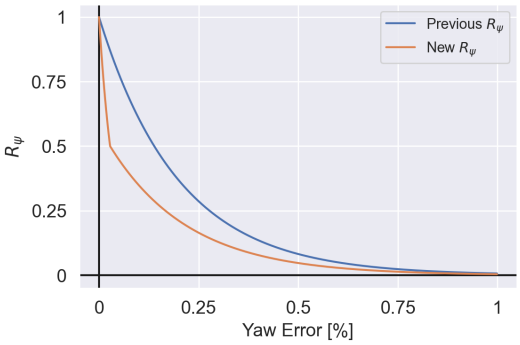
where $c_{\psi low}$ and $c_{\psi high}$ determine the steepness of the reward for lower and higher errors, ψ_{border} and r_{border} determine where the transition between two functions will occur in the yaw error and reward axes, respectively. There is only one $c_{\psi low}$ that satisfies the condition of two functions touching each other at $\psi_{err} = \psi_{border}$ and $R_t = r_{border}$ which can be calculated through

$$c_{\psi low} = -\frac{\ln(r_{border})}{\psi_{border}} \tag{9}$$

whereas $c_{\psi high}$ can be chosen freely.

The coefficients have been chosen as $c_{\psi high} = 5$, $\psi_{border} = 5/180$ (5 degrees normalized), $r_{border} = 0.5$ and the previous coefficients have been kept the same. The chosen border coefficients effectively makes sure that the agent is denied 50% of the maximum reward until it achieves at least 5° yaw error. The differences between the new R_ψ and the previous R_ψ has been demonstrated in Figure 7.

Figure 7: The new R_ψ compared to the previous one.



The agents trained with this reward function have been able to consistently outperform the designed LQR controller. The average performance of all of these seven agents have been compared to LQR in Figure 8.

Figure 8: The final agents compared to LQR. Yaw MAE on the left, total the rudder usage on the right. For the RL agents, the bars show the average performance of all seven trained agents and the error bars show the standard deviation between different agents.

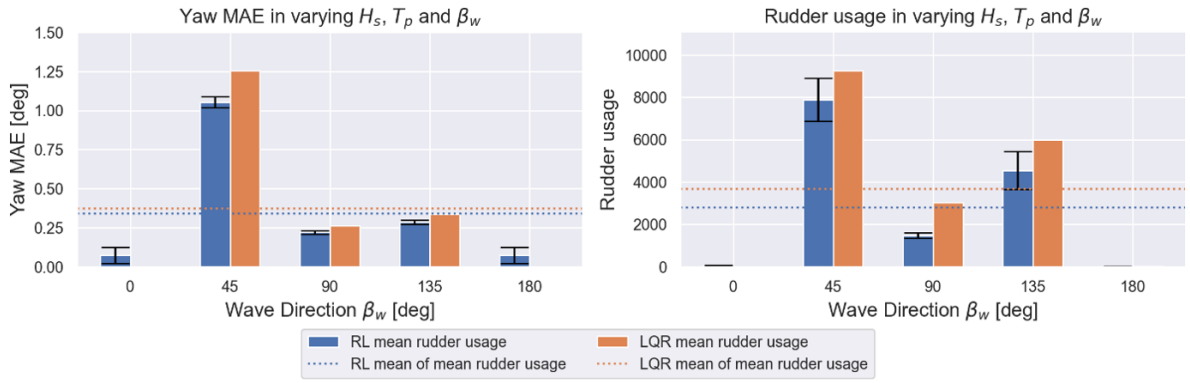
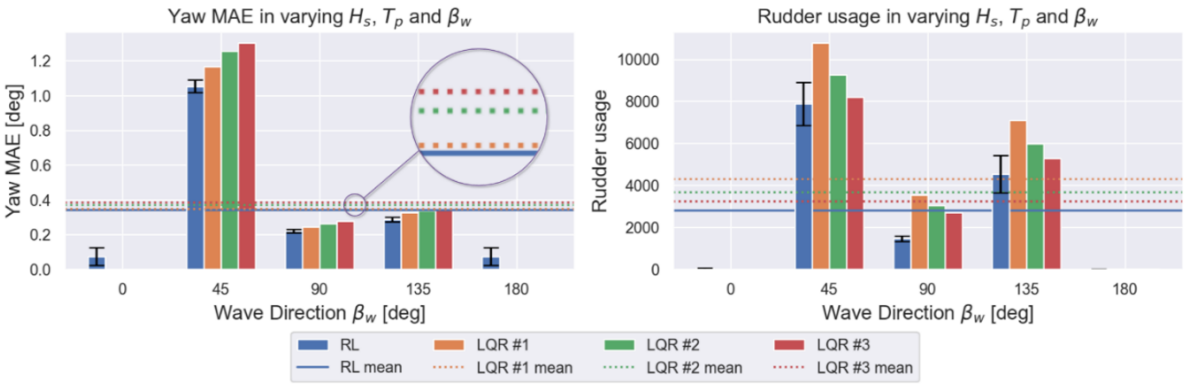


Figure 9: Comparing different LQR tunings to the RL agents.



Performance of LQRs with different tunings have been compared to RL agents in Figure 9. By observing LQR #1 in this figure, it can be seen that to be able to reach MAE similar to RL agents, LQR almost needs twice the rudder usage of RL agents. Through the comparison LQR #3 and RL

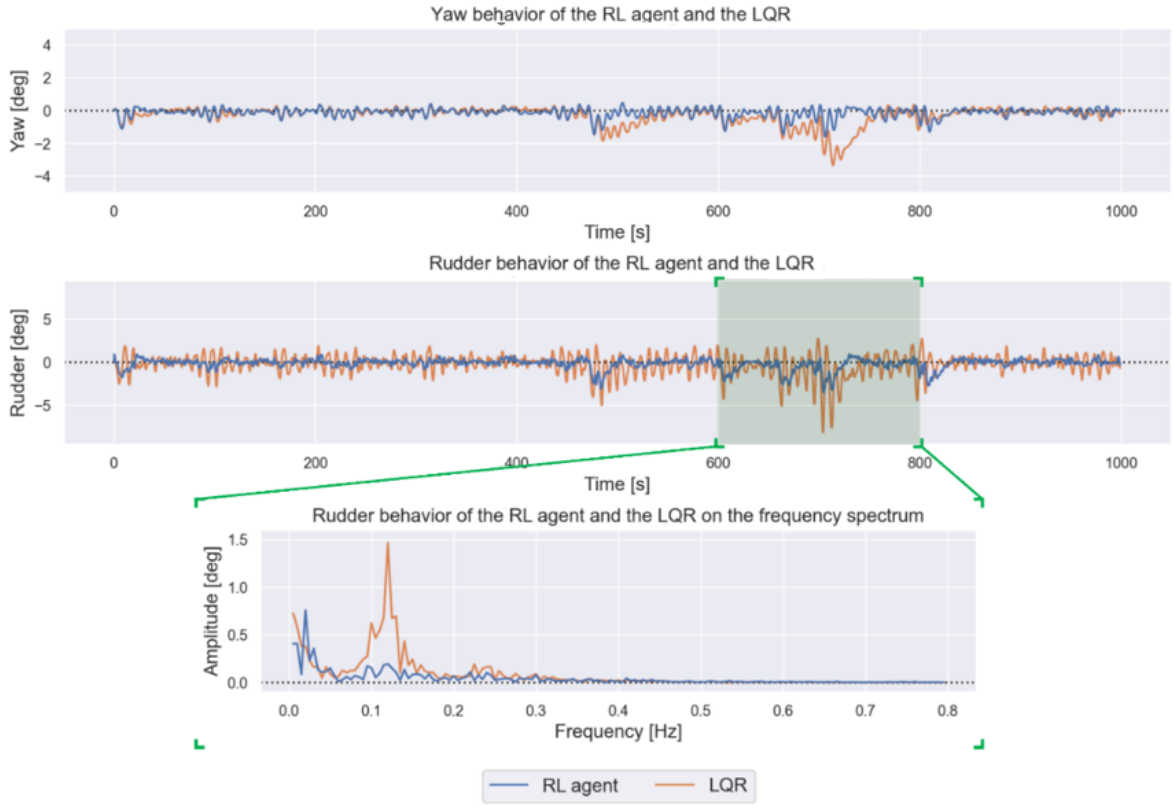
agents, it can be noticed that LQR needs to have a way higher MAE than RL agents to be able to attempt to beat RL agents in terms of rudder usage.

4.6 Behavioral Differences Between RL and LQR

Figure 10 compares the yaw and rudder behavior of RL and LQR controllers in both the time and frequency domains in a representative simulation. The frequency domain plot show that the LQR controller has two dominant frequencies, while the RL agent has only one. The frequencies of both controllers match around zero-frequency; however, Although the frequencies of both controllers match around 0 Hz, LQR has an additional higher dominant frequency at around 0.1 Hz, which results in excessive rudder usage compared to RL, as seen in the middle graph.

Further analysis of simulations, such as the one shown in Figure 10, has revealed that the consistent higher frequency peaks in the rudder response of LQR match with the wave encounter frequency. As noted by works such as (Fossen, 2011), a desirable controller should avoid responding to first-order wave forces, which correspond to encounter frequency, and instead, should try to counteract second-order slow wave drift forces. The final RL agents were able to learn this behavior without any explicit programming.

Figure 10: Rudder usage of agents and LQR compared. Yaw at the top, rudder in time domain in the middle, rudder in frequency domain (t=600s-800s) at the bottom.

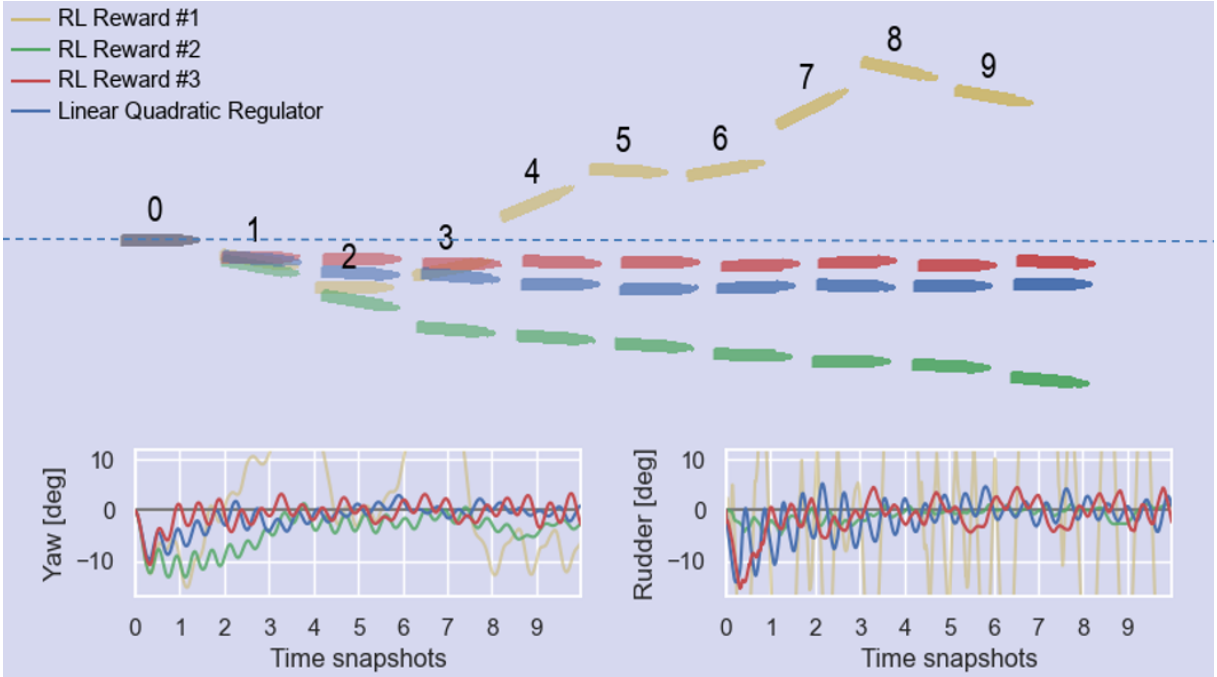


4.7 Overall Comparison of Different Reward Functions

The study demonstrates three iterations of RL agents. Only the reward function has been changed between the iterations. The behavior of these three iterations have been demonstrated in an example simulation at Figure 11. The figure also shows the behavior of LQR under the same sea conditions to be used as a baseline.

The figure clearly demonstrates the weaknesses and strengths of each iteration. The naive design of the first iteration is seen to cause excessive rudder usage and overshoots. The new terms added in the second iteration are demonstrated to help the agent learn how to fight these issues. However it is seen to struggle with steady-state error. The final agent iteration is demonstrated to be able to fix this issue with further improvements. The LQR's tendency of having a high frequency periodic rudder usage can also be seen when it is compared to third agent.

Figure 11: The performance of different agent iterations. The distances have been exaggerated for clarity.



5. CONCLUSIONS

In this study, an RL-based controller has been proposed to maintain a ship's course under external disturbances. The potential benefits of using an RL-based controller over a traditional one have been shown and discussed. The study starts with a weak controller and improves it iteratively to demonstrate how reward shaping can be used to fight common issues controllers face. The final iteration of our RL-based controller has been shown to outperform our baseline LQR controller in terms of yaw error and rudder usage, even after further tuning.

Our study contributes to the emerging field of ML methods in ship motion control and provides guidance for future research. In particular, future work could explore the use of model-based RL to improve sample efficiency and investigate the effectiveness of our approach for different ship geometries and varying speeds. Overall, our results demonstrate the promising potential of RL-based controllers for ship motion control.

ACKNOWLEDGEMENTS

We would like to thank MARIN for providing the us with the necessary software and hardware for our simulations.

REFERENCES

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. ArXiv Preprint ArXiv:1606.01540.

Fossen, T. I. (2011). Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley.

Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (1994). Feedback control of dynamic systems. Addison-Wesley.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv preprint arXiv:1801.01290.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., & Wu, Y. (2018). Stable Baselines [GitHub repository]. GitHub. Retrieved from https://github.com/hill-a/stable-baselines

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.

Martinsen, A. B., & Lekkas, A. M. (2018). Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning. IFAC-PapersOnLine, 51(29), 329-334.

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

Øvereng, S. S., Nguyen, D. T., & Hamre, G. (2021). Dynamic Positioning using Deep Reinforcement Learning. Ocean Engineering, 235, 109433.

Quadvlieg, F. H. H. A., & Rapuc, S. (2019, October). A pragmatic method to simulate maneuvering in waves. In SNAME Maritime Convention SMC, Washington, USA.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv preprint arXiv:1712.01815.

Simman 2008. (n.d.). US Navy Combatant, DTMB 5415 geometry and conditions. Retrieved from http://www.simman2008.dk/5415/5415_geometry.htm

Sutton, A. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

Wang, H., Yang, J., Lee, H.-S., & Han, S. (2020). Learning to Design Circuits. arXiv preprint arXiv:1812.02734.