Towards using Reinforcement Learning for Autonomous Docking of Unmanned Surface Vehicles

Martin Holen[1 [0000−0003−0967−221X]], Else-Line Malene Ruud[2], Narada Dilp Warakagoda[2,3 [0000−0002−9954−0431]], Morten Goodwin[1 [0000−0001−6331−702X]], Paal Engelstad[3], and Kristian Muri Knausgård[4 [0000−0003−4088−1642]]

[1] University of Agder, Centre for Artificial Intelligence Research
[2] Norwegian Defence Research Establishment (FFI)
[3] University of Oslo, Institute for Technology Systems
[4] University of Agder, Top Research Centre Mechatronics

Abstract. Providing full autonomy of Unmanned Surface Vehicles (USV) is a difficult goal to achieve. *Autonomous docking* is a subtask which is particularly difficult. The vessel has to distinguish between obstacles and the dock, and the obstacles can be either static or moving. In this paper, we developed a simulator using Reinforcement Learning (RL) to approach the problem.
We studied several scenarios for the task of docking of a USV in a simulator environment. The scenarios were defined with different sensor inputs and start-stop procedures, but with a simple shared reward function. The results show that the system managed to solve the task when the IMU (Inertial Measurement Unit) and GNSS (Global Navigation Satellite System) sensors were used to estimate the state, despite the simplicity of the reward function.

Keywords: Simulation · USV · Reinforcement Learning

1 Introduction

Autonomous vehicles have attracted significant interest in recent years, in diverse areas such as drones, self-driving cars and sea transport[33,3,27,22]. This paper deals with Unmanned Surface Vessels (USVs), an area which has displayed remarkable progress in the past few years. There are a few benefits of making the surface vehicles unmanned, such as removing the cost of the crew, and for shipping the possibility of removing crew areas such as cabins and bathrooms [16]; human factors are also removed [17], such as getting tired, and the ability to see in 360 degrees around the vessel, continuously.

One of the challenges associated with the development of USVs is that of docking, as the USV has to navigate through narrow spaces, avoid obstacles, localize itself and map the environment. Therefore, avoiding collisions while moving in and out of the dock autonomously is a difficult task. It is also important

DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

not to crash into other boats, the docks, or any other obstacles. If the docks are approached at a high speed, both the USV and the docks may be damaged.

In order to solve the task of docking we applied Reinforcement Learning (RL) which commonly used in video games and for controlling vehicles and more[20,8,18,9,13,35]. When performing RL, the state is given as the input to the RL algorithm. The output of the RL algorithm are called actions, which are used with the state, the next state and the reward to improve the reward it receives from the environment. State of the art RL algorithms include Deep-Q Networks (DQN) [20], Deep Deterministic Policy Gradient (DDPG) [18] and Proximal Policy Optimization (PPO) [26]. DQN has shown significant progress for video games[20], though it does not allow for continuous actions. Instead, these algorithms use discretized actions, meaning that the algorithm predicts the action from a given finite set of actions such as keyboard controls for a driving game. Continuous actions allow for a greater degree of control, examples of implementations can be found in [21,32,34,29]. DDPG and PPO can be applied to problems with continuous action spaces [18,26], allowing for more gradual control.

PPO is an algorithm which has the benefits of trust region policy optimizations where the estimations are done by the surrogate objectives. The surrogate objectives consist of two components and it chooses the pessimistic loss, as shown in Eq. 2, which makes sure the policy updates aren't too excessive [26]. When using an actor-critic architecture, the overall loss function would consist of the policy loss corresponding to the surrogate objective and the value function loss. This leads to Eq. 3, where the policy loss L^{CLIP} refers to Eq. 2, and value function loss L^{VF} refers to Eq. 1 where the value function loss is the squared error between the predicted value $V_\theta(st)$ and the target value V_t^{target}. The target value is assumed to be the total reward returned by the environment.

In addition to the policy and value function losses, we use an extra loss $S_\pi(...)$ representing the entropy of the policy as shown in 3.

$$L^{VF}(\theta) = (V_\theta(st) - V_t^{targ})^2 \tag{1}$$

$$\begin{aligned} L^{CLIP}(\theta) =& \hat{E}_t[min(r_t(\theta)\hat{A}_t \\ &+ clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \end{aligned} \tag{2}$$

$$\begin{aligned} L_t^{CLIP+VF+S}(\theta) =& \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) \\ &+ c_2 S[\pi_\theta](st)] \end{aligned} \tag{3}$$

There has been significant research on the use of RL for autonomous vehicles, for purposes such as reducing noisy rudder behavior [19], autonomous parking [35] as well as using Deep RL for controlling an Autonomous Underwater Vehicle while being affected by outside disturbances [8]. There is also some work exploring the use of RL algorithms instead of traditional control algorithms [28,13], where it was observed that RL algorithms are able to predict actions similar to the optimal control algorithms. They are also able to deal with disturbances observed.

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

Based on the literature review conducted, there are only a few studies that employ RL to docking of USVs [4,9]. This opens up the need to develop a RL based simulator which can be used to study an autonomous docking system. The main contributions of our work is the development of a new simulator for docking of USVs. This simulator provides a "standardized" way of creating and testing RL algorithms for docking. It was developed in Unity with the **socketio** module,[5] to control the USV with a programming language such as Python, C# or C++. We have provided the simulator as an open-source software.[6]

Our work also demonstrates that it is possible to train a RL algorithm for the task of autonomous docking of USVs using our simulator. The paper proposes a reward function for this purpose, which is a major challenge in applying RL for industrial purposes. The solution is developed and validated with the simulator.

This paper is organized as follows: Section 2 presents our methods for implementing the simulator and the agents for the environment. Results of our experiments are presented and discussed in Section 3 followed by conclusions in Section 4.

2 Methods

In this section, we will discuss the environment, the agent, and the reward function created for our simulator. With the environment giving the state to the agent. While the agent generates the actions to perform in the environment. The reward function is used to update the agent, giving it feedback about its performance.

2.1 Environment

The environment the agent interacts with is simple, containing land, water, docks. With the water being the area the USV moves across, land is the obstacle to avoid, and the goals are the docks. For the agent to learn how to control the USV it gets a reward from the reward functions. The higher the reward, the more the USV got to or moved towards the docks.

2.2 Agent

The agent is the entity that interacts with the environment.[7] It gets information about the environment through the given sensors and generates actions to change the state of the environment. To train this agent we use the PPO algorithm on a simple Neural Network (NN) [26]. The reason for choosing PPO is that it is among the best algorithms for training in a continuous action space.

[5] https://github.com/udacity/self-driving-car-sim
[6] https://anonymous.4open.science/r/boatSimulator-3605/
[7] https://anonymous.4open.science/r/SteeringDockingPaper-98BC/README.md

Neural Network The Neural Network (NN) used in the study was a simple neural network with 3 layers, similar to that from ALVINN [24]. It has an input layer, a hidden layer and an output layer and all these layers are implemented as fully connected layers. For Scenario 1,2 and 3, the input layer has 256*256*3 units, the hidden layer has 64 units and the output layer has 3 units. In Scenario 4, it has an input layer with 6 units, while the hidden layer and output layer has the same number of units as in previous scenarios. The first three scenarios have 12 583 104 weights, while scenario 4 has 576 weights to update. Between each layer there was a *tanh* activation, suggested by [2]. They are trained until they converged, or for 1000 updates of the neural network parameters. Convergence implies that the reward was unchanged for 100 episodes.

2.3 Reward function

One of the challenges with using RL for non trivial applications is to define an appropriate reward function[4] for the model. It is imperative for the reward function to capture the realistic behavior of the vehicle. The reward function defined in this study is such that it has three main components. Firstly, if the USV completes the episode by going to a dock the agent receives a reward of +1000. Secondly, the previous distance to the current dock and the new distance is used to calculate how far it traveled, as shown in Eq. 4

$$reward_i = distance_{i-1} - distance_i \tag{4}$$

Thirdly, a reward of -0.01 is given for each action performed, and this is meant to decrease the amount of stalling done by the vehicle, i.e. sitting in the same location. Using this reward function, we can pick and choose, to attempt to find the best fit for each scenario.

3 Simulator

We have developed a simulator to study Reinforcement learning based approach for docking of USVs. There are several commercially available and open-source software for simulating physics-based environments, like OpenAI Gym[5], Gazebo[15], Unity[14] and Unreal engine[11]. Our simulator, was built using the Unity game engine[14] for physics and visualization, SocketIO[25] was used for communication between the environment and the agent, the flow of which can be seen in Fig. 1b. Based on the simulator a set of scenarios were created where an increasing complexity from one approach to the next.

The USV is controlled by applying forces at the rear of the USV, simulating a physical vehicle. The following sections describe the sensors and control in more detail.

3.1 Sensors

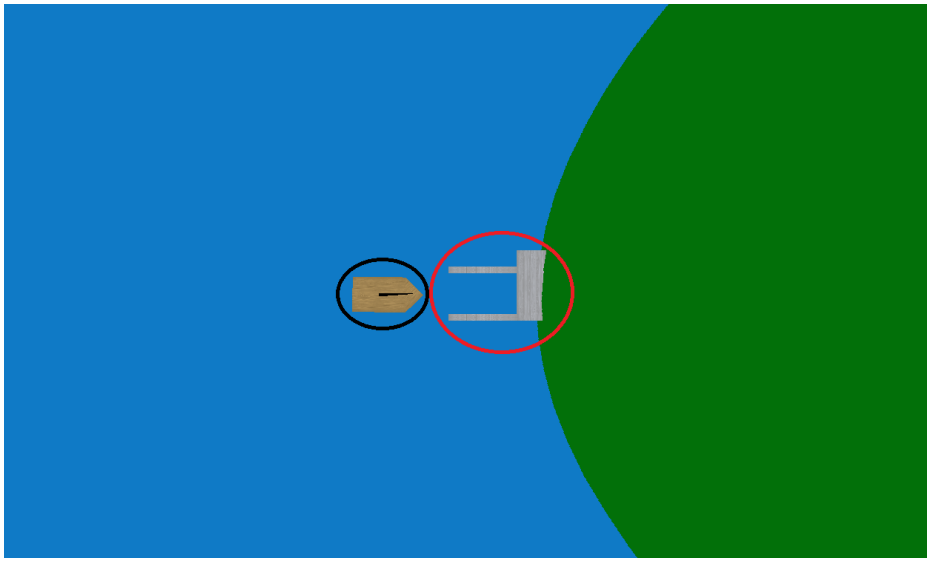
For an RL algorithm to understand its environment it needs sensors, which in our case includes a camera, an inertial measurement unit (IMU) and a global navigation satellite system (GNSS) sensor. The camera is a built-in sensor in the Unity GameEngine. It captures an RGB image, and is an idealized sensor, meaning that it will provide a noise-free picture of the scene. The camera is rotated about the Z-axis, showing a top-down overview of the USV, dock, ocean and the land, which lie on the x-y plane, simulating a local map made using Lidar/RADAR or a detailed prior map.

Next we have an IMU sensor, which is a built-in sensor of the Unity implemented as a GameObject. A GameObject is an object in the Unity environment which can hold other GameObjects, scripts and other entities. A given GameObject is associated with a transformation, which includes its position, rotation and scale [31]. The IMU sensor includes the velocity and angular velocity components along the x, y, and z axes. The GNSS, also a built-in GameObject is an idealized sensor which gives the exact position in (x, y, z) coordinates and rotation of the USV, the notation of which can be found in Tab. 1.

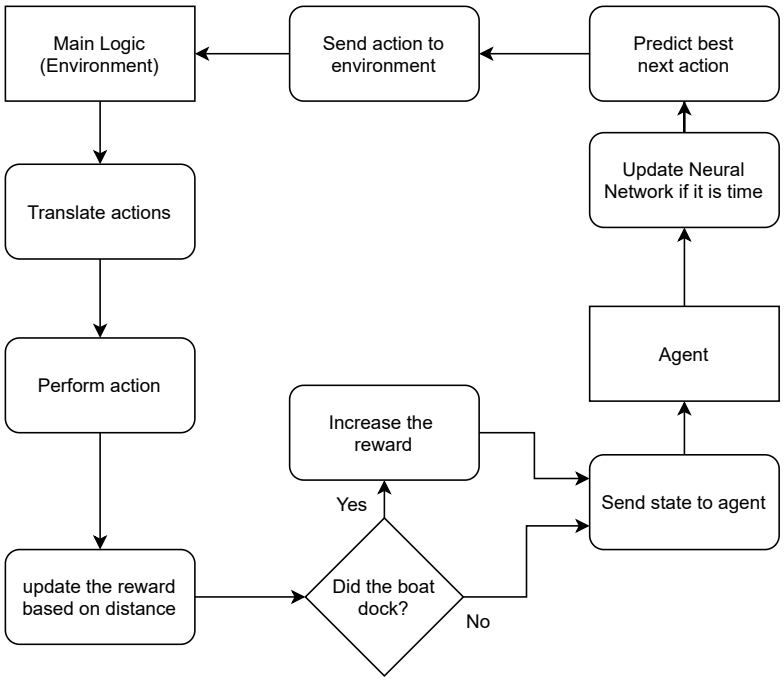
Table 1: DOF = Degree of freedom, Mot = motion in *axis, Rot = Rotation around the *-axis, FM = Forces and Moments, LAV = Linear and Angular velocities, PEA = Positions and Euler angles

DOF		FM	LAV	PEA
1	Mot xb axis (surge)	X	u	x^n
2	Mot yb axis (sway)	Y	v	y^n
3	Mot zb axis (heave)	Z	w	z^n
4	Rot xb-axis (roll)	K	p	ϕ
5	Rot yb-axis (pitch)	N	q	θ
6	Rot zb-axis (yaw)	Z	r	ψ

DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38



(a) The black circle shows the boat. The red circle showing the dock. While the green is land, and blue is the ocean



(b) Shows a flow chart of the environment and agent

Fig. 1: Shows the environment, as well as a flow chart of how it functions

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

3.2 Control

Actions from the agent (see Eq. 5) are applied to the USV as shown in Fig. 2. For the actions (X, Y, r) to be applied, they first have to be translated to forces. X is the accelerative force, Y is the rotational force, and r is either -1 or 1 used to change between reverse and moving forwards. As Unity does not take into account the bearing of the object, meaning that as the USV turns, the agent will have to take the angle into account when making its prediction. We use the rotation along the z axis of the USV (ψ), to calculate which direction to send the forces. This then means that the forces would be added from the rear to the front of the boat when its rotation is 0, but when its rotation is 90 the same force would go from left (port) to the right (starboard).

The forces to the body are calculated from the three actions (see Eq. 5) to the F^{body} (see Eq. 6). We then use the rotation of USV to calculate the rotational matrix (see Eq. 7) for the Unity coordinate system. F^{body} and R_ψ are then used to calculate the forces (F^{ENU}) which are directly applied to the USV (see Eq. 8). The notation of the control equations can be found in Tab. 1.

$$actions = [X \quad Y \quad r]^T \tag{5}$$

$$F^{body} = [rX \quad Y]^T \tag{6}$$

$$R_\psi = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \tag{7}$$

$$F^{ENU} = R_\psi^T * F^{body} \tag{8}$$

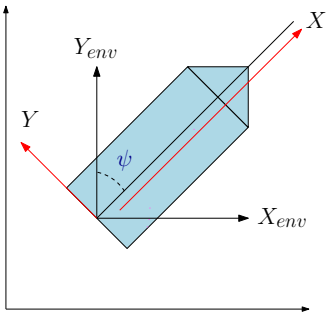


Fig. 2: Forces (X, Y) applied on the USV, given the predicted force (X_{env}, Y_{env})

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

4 Experiments and Results

This section focuses on the results of each of the four scenarios outlined below. We trained the agent in each scenario, until they converged or for 1000 parameter updates (backpropagations). The number of backpropagations were chosen due to the time constraint.

In order to test our algorithm in the environment, a few different scenarios were defined each with a differing degree of complexity. We have considered a total of four scenarios, one of which does not include islands (which might hinder the movement of the boat), and the rest include islands. For the latter three scenarios we use a top-down idealized camera sensor, which is hanging in the air while following the vessel.

These four scenarios differ quite a bit, each with its own reward functions and reset system, an overview of which can be found in Tab. 2. The last scenario has a different environment setting. In the following we describe each scenario in more detail.

Table 2: Shows the different scenarios implemented. In the table the following X, O actions indicate, X - Yes, O - No.
ResetDocked means that the USV is Reset when it is close enough to the dock.
ResetOffMap, Resets the USV when it goes off the map.
Reset25m means that the USV is Reset when it drives away from its Reset position of 10m-15m

Scenario	Image	Islands	State	ResetDocked	ResetOffMap	Reset25m
Scenario 1	X	X	O	O	X	O
Scenario 2	X	X	O	O	X	O
Scenario 3	X	X	O	X	X	X
Scenario 4	O	O	X	X	X	X

Scenario 1 is a simple scenario in which the USV is reset to its spawning location if it moves off the map. The reward in this scenario is based on the generic reward function described in section 2.3. If it reaches the goal, it is given a new dock as its current goal, towards which it has to navigate again. For its state, it uses a top-down idealized camera, with an arrow pointing towards the current goal as shown in Fig. 1a.

Scenario 2 is also a simple one, which has a similar reset policy as in Scenario 1. Its reward function has a similar rule as in Scenario 1, where a reward is awarded based on the traveled distance towards the dock. In addition, when attempting to apply an action, this scenario penalizes any actions which is not directing the USV towards the current goal. When an action is bearing towards the goal, with a maximum deviation of 45 degrees, or the predicted force X is lower than 0.5, the forces are applied to the vessel; otherwise the action is not

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

applied and the agent receives a penalty. If it reaches the goal, it is given a new dock as its current goal, towards which it has to navigate again. For its state, it uses a top-down idealized camera, with an arrow pointing towards the current goal as shown in Fig. 1a.

Scenario 3 resets the USV for moving off the map, moving further than 25m away from the goal or reaching its goal. If the USV is reset or is spawning, it is placed within 45 degrees of the current goal and at a distance of 10-15m. This means that the USV is reset close to the opening of the docks. If it reaches the goal, it is given a new dock as its current goal, which it has to navigate towards it again. The reward function of this scenario is identical to that of Scenario 1. For its state, it uses a top-down idealized camera, with an arrow pointing towards the current goal as shown in Fig. 1a.

Scenario 4 is somewhat more simplistic than the other scenarios as it does not include any islands which might act as obstacles. Its reset strategy and reward function are identical those of Scenario 3. For its state, it uses the IMU and GNSS data, specifically the distance in the x and y axis, z rotation, z angular velocity, surge and sway as shown in Fig. 3.

An overview of the scenarios can be found in Tab. 2.

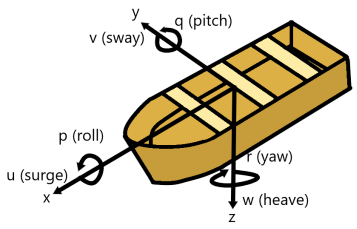


Fig. 3: The linear and rotational motion of the boat with respect to the coordinate system of the model, adopted from Fossen [30,12]

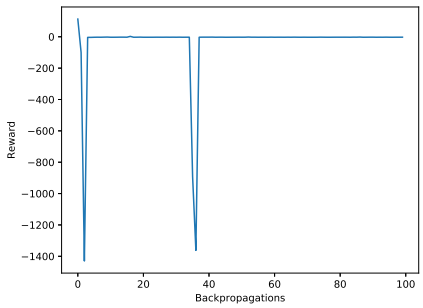
Any reward greater than 700 achieved by the system implies that the USV reached the goal, at least once. When the system has reached the goal, and has performed more than 2000 actions, the NN parameters are updated, otherwise the system continues to the next goal. With fewer experiences than the hyperparameter batch size when updating the NN is not able to sample the memory.

Fig. 4d shows that Scenario 4 converged after 100 backpropagations, while Fig. 4c shows that the algorithm did not learn much during this training, but it kept trying new things. Fig. 4a and 4b represent scenario 1 and scenario 2 respectively, it is observed that they did not learn much, staying around a reward of -2.9, and deviating only 3 times around this score.

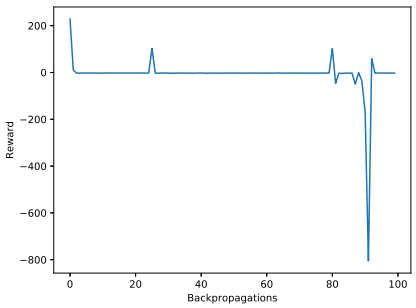
Tab. 3 shows the performance between backpropagation 100-200, where scenario 4 had the highest maximum, minimum and average reward. Tab. 4 shows

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

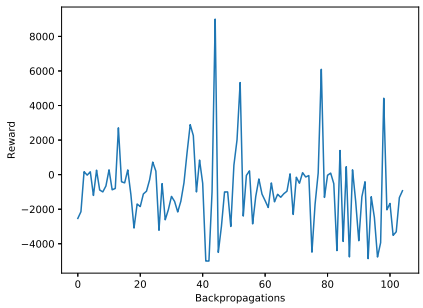
that there was little to no performance increase among scenario 1,2 and 3, though scenario 2 seemed to be most negatively affected.



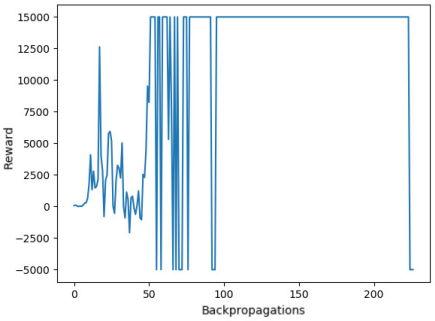
(a) Scenario 1 - Rewards being averaged over 10 backpropagations



(b) Scenario 2 - Rewards being averaged over 10 backpropagations



(c) Scenario 3 - Rewards being averaged over 10 backpropagations



(d) Scenario 4 - Converged after only about 100 episodes

Fig. 4: The results of the different scenarios

The Scenarios show that there is a difference in the reward awarded in different scenarios. More specifically, Scenario 1 and 2 did very poorly with essentially no improvement or learning. For Scenario 4 we see a very quick improvement, while Scenario 3 attempted to update the NN, but did not converge over the 1000 episodes.

In the different scenarios, the agent has got different types of rewards. In Scenario 3 and 4 the agents were given positive rewards more often, in Scenario 1 and 2 the agents have got rewards only occasionally. The results are consistent with these patterns of reward functions showing that Scenario 4 learnt more, which can be attributed to the feedback and simplicity of the environment.

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

Table 3: The nMax is the maximum number of times it reached dock, min is the lowest reward, and AVG is the average reward. These are done for episode 100-200, when scenario 4 converged.

Scenario	nMax	Min	AVG
Scenario 1	0.0	-3.1177	-2.144
Scenario 2	0.0	-3.0	-2.4031
Scenario 3	15.0	-5000	-525.2318
Scenario 4	15.0	15000	15000

Table 4: The nMax is the maximum number of times it reached dock, min is the lowest reward, and avg is the average reward. These are done for the last 100 episodes of each scenario.

Scenario	nMax	Min	AVG
Scenario 1	0.0	-2.9913	-2.7341
Scenario 2	1.0522	-1686.1361	-93.3627
Scenario 3	11.74	-5000	-525.2318
Scenario 4	15.0	-5000	14800

Scenario 4 achieved the highest reward consistently after only about 100 backpropagations, with some exploration at the end.

Scenario 3 was observed to be varying quite a lot, implying that it was still training, with some very large variations giving the USV between the maximum and the minimum reward, though the average reward did not seem to improve much. Given more training time we assume we could have increased the performance on this scenario. This behavior can be attributed to its higher complexity than that of Scenario 4 due to the camera being its sensor versus IMU and GNSS. The scenario of moving from one part of the map to the next, is a more complex scenario, and so Scenario 1 and Scenario 2 will take significantly longer to train.

If we intend to improve Scenario 1 and 2, we may perform something similar to the approach in [10], where transfer learning is applied starting from from the easiest scenario to the most difficult one, i.e. from Scenario 4 through Scenarios 3 and 2 to Scenario 1. Scenario 4 has an important advantage over the other scenarios due to the nature of its state-space where the agent has direct access to distance in x and y along with the angle of rotation to the goal, compared to having to learn this just from the bearings of an arrow pointing towards the goal in other scenarios. In Scenario 1 and 2 the USV has to move from one dock to the next after it has reached a dock, and they are not spawned in the vicinity of the docks, making them more complex scenarios. If given enough time, however, Scenario 2 should get a better feedback than Scenario 1, as Scenario 1 allows for actions which moves it away from the next dock, unlike Scenario 2 which simply penalizes such actions while not performing them.

Dette er en postprint-versjon/This is a postprint version.
DOI til publisert versjon/DOI to published version: 10.1007/978-3-031-08223-8_38

5 Conclusion and Further Work

In this work, we developed a simulator which could be used for RL. The contribution of this work is three-fold. Firstly we present an open-source Unity based boat simulator for development of USVs. Secondly, application of RL to the docking problem was demonstrated in several scenarios, thirdly a reward function to give feedback to the agents has been defined and studied. The performance of this simulator was tested in 4 different training scenarios, using IMU and GNSS or a top-down camera as the sensors. Each of these scenarios had a different variant of the reward function and the reset systems.

Our results showed that Scenario 4 converged very quickly while Scenario 1, 2 and 3 took quite long to train. We conclude that Scenario 4 achieved its goal of docking, after updating parameters of the agent through only about 100 backpropagations. It was also concluded that scenario 1 and 2 struggled due to their complexity, and may require more feedback or a longer training time.

To improve the training of Scenario 1,2,3, we could use the state data similar to Scenario 4 in addition to the camera. For Scenario 1 and 2 we could, for example, make checkpoints to provide additional feedback when the USV gets closer to them. It would also be interesting to train these algorithms for a longer period of time and observe the changes. Another approach is to transfer the knowledge gathered in a simpler scenario to a more complex scenario through transfer learning. As well as developing the simulator further, implementing different propulsion methods, sensors and further developing the reward function. And lastly it would be possible to move the simulation to a Gym environment [6] using Gym-unity [1] to train with the standard algorithms from baselines [23] or dopamine [7].

References

1. ml-agents/gym-unity at main · Unity-Technologies/ml-agents, https://github.com/Unity-Technologies/ml-agents/tree/main/gym-unity
2. Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., Bachem, O.: What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study (6 2020), http://arxiv.org/abs/2006.05990
3. Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L., Berriel, R., Paixão, T.M., Mutz, F., de Paula Veronese, L., Oliveira-Santos, T., De Souza, A.F.: Self-driving cars: A survey. Expert Systems with Applications **165**, 113816 (3 2021). https://doi.org/10.1016/J.ESWA.2020.113816
4. Bjering Strand, H.: Autonomous Docking Control System for the Otter USV: A Machine Learning Approach (2020), https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2780950
5. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Openai, W.Z.: OpenAI Gym (6 2016), https://arxiv.org/abs/1606.01540v1
6. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (6 2016), https://arxiv.org/abs/1606.01540v1

7. Castro, P.S., Moitra, S., Gelada, C., Kumar, S., Bellemare, M.G.: Dopamine: A Research Framework for Deep Reinforcement Learning (12 2018), https://arxiv.org/abs/1812.06110v1
8. Cui, R., Yang, C., Li, Y., Sharma, S.: Adaptive Neural Network Control of AUVs with Control Input Nonlinearities Using Reinforcement Learning. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(6), 1019–1029 (6 2017). https://doi.org/10.1109/TSMC.2016.2645699
9. Cui, Y., Osaki, S., Matsubara, T.: Autonomous boat driving system using sample-efficient model predictive control-based reinforcement learning approach. Journal of Field Robotics **38**(3), 331–354 (5 2021). https://doi.org/10.1002/ROB.21990
10. Dosovitskiy, A., Ros, G., Codevilla, F., López, A., Koltun, V.: CARLA: An Open Urban Driving Simulator. Conference on robot learning pp. 1–16 (2017)
11. Epic Games, I.: The most powerful real-time 3D creation tool - Unreal Engine, https://www.unrealengine.com/en-US/
12. Fossen, T.I.: Handbook of marine craft hydrodynamics and motion control, 2nd Edition. Wiley, 2nd edn. (4 2021)
13. Gaudet, B., Linares, R., Furfaro, R.: Deep reinforcement learning for six degree-of-freedom planetary landing. Advances in Space Research **65**(7), 1723–1741 (4 2020). https://doi.org/10.1016/J.ASR.2019.12.030
14. Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D.: Unity: A General Platform for Intelligent Agents (9 2018), https://arxiv.org/abs/1809.02627v2
15. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) **3**, 2149–2154 (2004). https://doi.org/10.1109/IROS.2004.1389727
16. Kretschmann, L., Burmeister, H.C., Jahn, C.: Analyzing the economic benefit of unmanned autonomous ships: An exploratory cost-comparison between an autonomous and a conventional bulk carrier. Research in Transportation Business & Management **25**, 76–86 (12 2017). https://doi.org/10.1016/J.RTBM.2017.06.002
17. Kyriakidis, M., Winter, J.C.F.d., Stanton, N., Bellet, T., Arem, B.v., Brookhuis, K., Martens, M.H., Bengler, K., Andersson, J., Merat, N., Reed, N., Flament, M., Hagenzieker, M., Happee, R.: A human factors perspective on automated driving. https://doi.org/10.1080/1463922X.2017.1293187 **20**(3), 223–249 (5 2017). https://doi.org/10.1080/1463922X.2017.1293187, https://www.tandfonline.com/doi/abs/10.1080/1463922X.2017.1293187
18. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings (9 2015), https://arxiv.org/abs/1509.02971v6
19. Martinsen, A.B., Lekkas, A.M.: Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning. IFAC-PapersOnLine **51**(29), 329–334 (1 2018). https://doi.org/10.1016/J.IFACOL.2018.09.502
20. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 2015 518:7540 **518**(7540), 529–533 (2 2015). https://doi.org/10.1038/nature14236, https://www.nature.com/articles/nature14236

21. Moerland, T.M., Broekens, J., Plaat, A., Jonker, C.M.: A0C: Alpha Zero in Continuous Action Space (5 2018), https://arxiv.org/abs/1805.09613v1
22. Mousazadeh, H., Jafarbiglu, H., Abdolmaleki, H., Omrani, E., Monhaseri, F., Abdollahzadeh, M.r., Mohammadi-Aghdam, A., Kiapei, A., Salmani-Zakaria, Y., Makhsoos, A.: Developing a navigation, guidance and obstacle avoidance algorithm for an Unmanned Surface Vehicle (USV) by algorithms fusion. Ocean Engineering **159**, 56–65 (7 2018). https://doi.org/10.1016/J.OCEANENG.2018.04.018
23. OpenAI: openai/baselines: OpenAI Baselines: high-quality implementations of reinforcement learning algorithms, https://github.com/openai/baselines
24. Pomerleau, D.A.: ALVINN: AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK. Advances in Neural Information Processing Systems (1989), https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf
25. Rai, R.: Socket. IO Real-Time Web Application Development - Rohit Rai - Google Books. Packt Publishing Ltd., Birmingham, 1st edn. (2 2013), https://books.google.no/books?id=YgdbZbkTDkoC&pg=PT37&dq=socket+io&lr=&source=gbs_selected_pages&cad=2#v=onepage&q=socket%20io&f=false
26. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms (7 2017), https://arxiv.org/abs/1707.06347v2
27. Shao, G., Ma, Y., Malekian, R., Yan, X., Li, Z.: A novel cooperative platform design for coupled USV-UAV systems. IEEE Transactions on Industrial Informatics **15**(9), 4913–4922 (9 2019). https://doi.org/10.1109/TII.2019.2912024
28. Shuai, Y., Li, G., Cheng, X., Skulstad, R., Xu, J., Liu, H., Zhang, H.: An efficient neural-network based approach to automatic ship docking. Ocean Engineering **191**, 106514 (11 2019). https://doi.org/10.1016/J.OCEANENG.2019.106514
29. Tang, Y., Agrawal, S.: Discretizing Continuous Action Space for On-Policy Optimization. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 5981–5988 (4 2020). https://doi.org/10.1609/AAAI.V34I04.6059, https://ojs.aaai.org/index.php/AAAI/article/view/6059
30. Thor I. Fossen: Lecture Notes: TTK 4190 Guidance, Navigation and Control of vehicles, https://www.fossen.biz/wiley/pdf/Ch1.pdf
31. Unity: Unity - Manual: GameObjects, https://docs.unity3d.com/Manual/GameObjects.html
32. Van Hasselt, H., Wiering, M.A.: Reinforcement learning in continuous action spaces. Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL 2007 pp. 272–279 (2007). https://doi.org/10.1109/ADPRL.2007.368199
33. Vásárhelyi, G., Virágh, C., Somorjai, G., Nepusz, T., Eiben, A.E., Vicsek, T.: Optimized flocking of autonomous drones in confined environments. Science Robotics **3**(20) (7 2018). https://doi.org/10.1126/SCIROBOTICS.AAT3536/SUPPL{_}FILE/AAT3536{_}SM.PDF, https://www.science.org/doi/abs/10.1126/scirobotics.aat3536
34. Veelen, M.v., Spreij, P.: Evolution in games with a continuous action space Matthijs van Veelen · Peter Spreij (2008). https://doi.org/10.1007/s00199-008-0338-8
35. Zhang, P., Xiong, L., Yu, Z., Fang, P., Yan, S., Yao, J., Zhou, Y.: Reinforcement Learning-Based End-to-End Parking for Automatic Parking System. Sensors 2019, Vol. 19, Page 3996 **19**(18), 3996 (9 2019). https://doi.org/10.3390/S19183996, https://www.mdpi.com/1424-8220/19/18/3996/htmhttps://www.mdpi.com/1424-8220/19/18/3996