Universiteit Leiden

Master Computer Science

Reinforcement learning for autonomous ferry navigation.

Name:	Marten Wiersma
Student ID:	s3200760
Date:	July 1, 2024
Specialisation:	Artificial Intelligence
1st supervisor:	Aske Plaat
2nd supervisor:	Álvaro Serra Gómez

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University

Abstract

Autonomous navigation within the maritime industry to improve safety and operational efficiency is an active and important area of research. In recent years, advanced methods leveraging reinforcement learning have started to find their way into the field of motion planning for autonomous surface vessels. We have experimented with combining *deep reinforcement learning (DRL)* and optimal control solvers like the *model predictive controller (MPC)* to create a local planner for an autonomous surface vessel. Additionally, we experimented with a DRL direct control approach and compared these two methods to an existing method that combines a classical graph-based search algorithm with an MPC. We found that the combination of DRL and MPC performed significantly better than the direct control approach and we showed similar path-planning capabilities to the classical graph-based search method. The experiments show promising results for obstacle avoidance in our simulated environment using our proposed method. This work provides a starting point for further development of a combined DRL and MPC-based local planner where future research can improve on theoretical safety guarantees and the ability to interact with other agents in a dynamic environment.

Acknowledgements

This thesis is conducted in collaboration with *Roboat* and supervised by the *Leiden Institute of Advanced Computer Science (LIACS)* at Leiden University to obtain a Master's degree in Computer Science. I am thankful for the opportunity to do this research project at this fascinating company and to everyone involved in this project.

First, I want to thank Aske Plaat for the guidance and feedback in all our meetings and overall support during the entire project. I appreciate the generous amount of time I could ask from you and the insights into the writing process.

Next, I would like to thank everyone at Roboat I have worked with during this project, especially Joshua Jordan and Julien Taveau for the supervision and the sparring sessions at Roboat and for providing all the necessary context for the project. I would also like to thank Jonathan Klein Shiphorst and Erwin Lodder for explaining the control systems related to the project and aiding in the search for interesting developments in the field.

Finally, I would like to thank Álvaro Gómez for the in-depth talks about the inner workings of the methods used in the thesis. Your support during the final stages of the project made an enormous difference and has taught me a lot in a very short amount of time.

Contents

1 Introduction

In the maritime sector, navigation can be a challenging task. Sailing a ship requires a balance between maneuvering efficiently yet safely, where factors like the trip duration, fuel consumption, and collision avoidance can contradict each other. Solving this problem generally requires the expertise of highly skilled skippers. Despite good education and training, human skippers sometimes still struggle with complex situations or bad environmental conditions which could lead to incidents with significant consequences (EMSA, 2023). For these reasons, research towards assistive technology for skippers is a long-existing field that aims to improve the safety and efficiency of maritime surface vessels (Calvert, 1960).

These support systems traditionally aid the skipper with one or more sub-tasks that can be categorized by a set of high-level tasks referred to as *Guidance Navigation and Control (GNC)* (Vagale et al., 2021). These tasks describe the complete process of controlling any vehicle or system that acts in an uncontrolled environment. An example of a GNC definition for maritime vessels is illustrated in Figure 1.

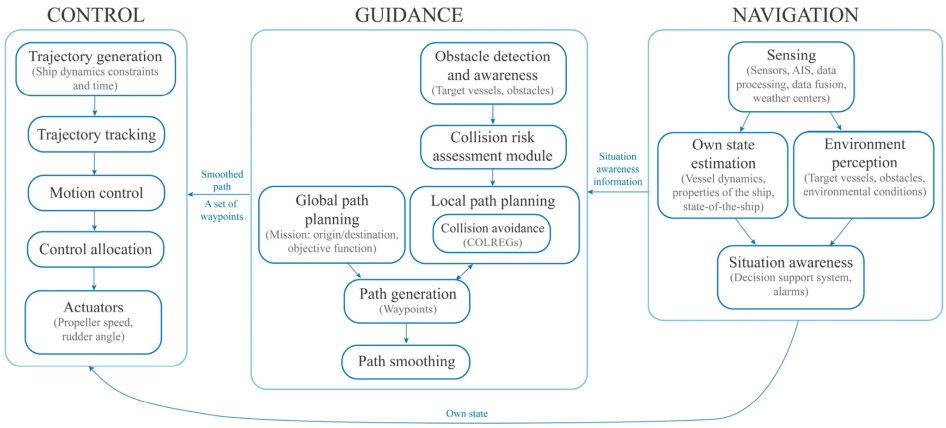


Figure 1: Guidance Navigation and Control architecture as proposed by Vagale et al. (2021). The architecture provides sub-tasks for each main task within the GNC model. The Navigation tasks are responsible for determining the state of the environment including the vessel itself. The Guidance tasks relate to the planning of a safe and feasible route. Finally, the control tasks ensure that this route is executed. This example should be considered a general architecture, where the categorization of each sub-task can deviate per implementation.

As technology progressed, maritime support systems became more advanced, providing advanced information like collision-free trajectories (Lazarowska, 2016). Such developments have also found their way into research towards fully autonomous vessels (DNV GL, 2014), where the system should be able to perform all GNC tasks without human intervention.

This work will focus on one of the GNC sub-tasks and is performed in collaboration with Roboat. Roboat is a company based in Amsterdam that develops an autonomy system that can be applied to any vessel as either a retrofit or a fully integrated part of the vessel. We will specifically look into the *local planning* applied to Roboat's autonomy solution.

We see an opportunity to apply reinforcement learning as part of the local planning mechanism for an autonomous vessel. This can be implemented in the current GNC structure, using reinforcement learning to learn a sub-goal recommendation policy for the currently implemented MPC controller, or by learning a policy combining planning and control as an end-to-end system. Therefore the main problem we will discuss is: *"What reinforcement learning algorithm works best for the autonomous navigation of a ferry in the city of Amsterdam"*. This will be addressed with the following research questions:

RQ1: How to model the problem in the MDP: end-to-end or separating planning and control?

RQ2: Which of the following methods is best suited for local planning based on an occupancy map: a DRL-based local planner or a graph-based local planner?

To answer these research questions, we will first implement the end-to-end method and the sub-goal recommendation approach and compare them experimentally to each other. Secondly, we will compare our proposed method to Roboat's current implementation.

The following section will discuss the background information related to Roboat's current implementation and references to related work on local path planning. Section 3 will cover the modeling and implementation of our contribution, including the MDP modeling and model architectures. Section 4 will describe the experimental setup used to evaluate the research questions. The results of these experiments are discussed in Section 5. Finally, we will review the results, conclude this work, and provide suggestions for future research in Sections 6 and 7.

2 Related Work

2.1 Roboat

This work is done in collaboration with Roboat. Roboat is a company based in Amsterdam developing navigation assistance and autonomy solutions for the maritime industry. The company is a spin-off of a research project conducted by Wang et al. (2020) together with the *Massachusetts Institute of Technology (MIT)*, and the *Amsterdam Institute for Advanced Metropolitan Solutions (AMS)*. Roboat's goal is to create a modular system that can be commissioned on any in-land vessel to provide various levels of autonomous capabilities.

For this project, we will limit this scope by only considering Roboat's current development vessels as illustrated in Figure 2. Lucy and Tony both have the same method of propulsion which will be further discussed in Section 2.1.1. Due to the identical method of propulsion, the methods that we will propose in Section 3, can be applied to both vessels.

(a) Lucy **(b)** Tony

Figure 2: Roboat's test vessels. *a)* An image of Lucy which is a small 6-person water-taxi designed for research and demonstration purposes. *b)* A render of Tony, a larger ferry designed with a maximum capacity of 35 people for supervised autonomous operation on the Seine in Paris.

Roboat's autonomy system currently resembles a GNC process similar to the architecture illustrated in Figure 1. This starts with the navigation module, which is responsible for determining the current state of the vessel and the environment. Both vessels are therefore equipped with two sensor modules. Each sensor module contains multiple cameras, a Lidar, GPS, and an *Internal Measurement Unit (IMU)*. The Navigation module uses these sensors to create an occupancy grid (Thrun et al., 2005) and an accurate measurement of the vessel's position, heading, and velocities. To enhance the occupancy grid, a YOLO-based instance segmentation model (Bolya et al., 2019) filters out any misclassified obstacles on the grid. The resulting grid is smoothed with a Gaussian kernel to reduce the impact of the sensor noise. This last stage provides the final image with a probabilistic characteristic indicating the probability of obstacles at each location on the map.

The guidance process uses this information to generate safe and efficient trajectories to the target location. This process is executed in two stages. First, a *global planner* applies the *probabilistic roadmap (PRM)* algorithm (Kavraki et al., 1996) to a static map of the area. This generates a set of waypoints from which one is selected as the current *target destination*. For the second stage, the local planner generates a safe trajectory by applying *A** on the occupancy grid, starting from the vessel's current position towards the *target destination*.

Finally, the *controller* converts this trajectory to a set of actuator control inputs. This final stage is implemented as a *Non-linear Model Predictive Controller (NMPC)* (Houska et al., 2011). The controller leverages a model of the vessel dynamics to optimize a series of control inputs, minimizing the expected deviation between the provided trajectory and the predicted trajectory that results from these control inputs.

As mentioned in Section 1, with this work we will focus specifically on the local planner and the interface between the local planner and the controller. For this purpose, we will assume that the navigation module will provide us with accurate occupancy maps and the current location, heading, and velocities of the vessel.

2.1.1 Vessel Dynamics

As discussed in the previous section, both of Roboat's development vessels use the same propulsion method. This propulsion method consists of 4 propellor-based actuators referred to as *thrusters*. The thrusters can generate a thrust i.e. *a force*, in a forward or backward direction relative to their orientation which is set up in an H-configuration as illustrated in Figure 3. This configuration is *over-actuated* meaning that even without one of the thrusters, the vessel is capable of *omnidirectional movement*, i.e., moving in any direction.

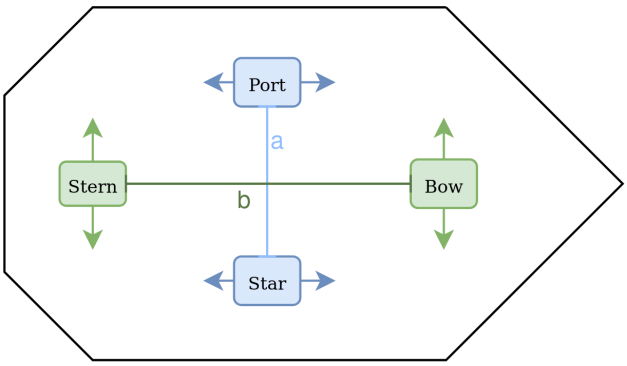


Figure 3: Schematic view of Roboats propulsion method. The four actuators labeled: Stern, Bow, Port, and Star, are placed in an H-configuration. The distance between the actuators is indicated by a and b. The arrow indicates the direction in which the actuators can provide a force.

This H-configuration is also used in one of the original Roboat papers (Wang et al.,

2020), which provides a set of differential equations describing the dynamics of such a vessel. We slightly modify the definition of the dynamics model from this paper to match the more recent implementation resulting in Equation 1.

$$\dot{\mathbf{x}} = \mathbf{T}(\mathbf{x})\mathbf{v} \tag{1a}$$

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{B}\mathbf{u}) - \mathbf{M}^{-1}\mathbf{D}(\mathbf{v})\mathbf{v} \tag{1b}$$

$$\dot{\mathbf{u}} = [u_{stern}, u_{bow}, u_{star}, u_{port}]^T \tag{1c}$$

where $\mathbf{x} = [x, y, \theta]^T$ represents the position and heading of the vessel relative to the world reference frame, $\mathbf{v} = [\dot{x}, \dot{y}, \omega]^T$ represents the velocities in the vessels reference frame. The transformation matrix $T(x)$ translates the state vector $\mathbf{x}$ to the vessel's reference frame. $\mathbf{u}$ represents the control vector where each element u represents the change in thrust applied by each thruster.

The remaining matrices, $\mathbf{M} \in \mathbb{R}^{3\times3}$, $\mathbf{B} \in \mathbb{R}^{4\times3}$, and $\mathbf{D} \in \mathbb{R}^{3\times3}$ represent the *mass and inertia matrix*, *control matrix*, and *drag matrix* respectively. These matrices define the vessel's characteristics and are different for Tony and Lucy.

2.2 Local Planning

Graph-based planning

As mentioned in Section 2.1, both the global and local planners of Roboat's current system are based on graph-based search algorithms. Within the maritime sector, graph-based planners are primarily used for global planning applications, however, there are also numerous studies applying them to local planning tasks (Ülkü Öztürk et al., 2022).

The main challenge of using graph-based methods for local planning tasks comes when dynamic obstacles and environmental effects on the vessel must be considered. Various studies have therefore resulted in variations on the well-known A* algorithm, considering multiple trajectories based on the probability of the effects of certain actions (Svec et al., 2011), or restricting the search space so that solutions can be found in real-time allowing to consider dynamic obstacles as static (Singh et al., 2018).

Directional methods

Another category of methods often used in robotics and the maritime industry are *directional methods*. These methods have in common that they suggest changes in heading and/or velocities relative to the agent's current position. The two methods most often used in the Maritime industry according to Ülkü Öztürk et al. (2022) are *Velocity Obstacles (VO)* and *Artificial Potential fields (APF)*.

The VO algorithm (Fiorini and Shiller, 1998) provides a geometric solution for determining when two objects will collide based on their relative velocities, and

what correction maneuvers are possible to avoid a collision. This method requires that the current location and velocities of the obstacles are known. This will generally not pose a problem for static obstacles since the relative velocities can be deduced from the OS's current velocity. However, for dynamic obstacles, this requires either multiple observations of the same object over time or, additional information from sources like the *Automated Identification System (AIS)* or an equivalent system.

APFs (Khatib, 1986) are often used due to their simplicity and high computational efficiency. The method computes a 'reaction force' based on a set of attractive and repelling forces artificially generated by the target and obstacles respectively. By modeling the equations that determine the strengths of these forces specific behavior can be determined.

Both APF and VO methods are active topics of research where recent works have proposed methods that take the naval *collision regulations (COLREGS)* into account (Kufoalor et al., 2018), (Lyu and Yin, 2019).

Optimisation based methods
The final category of methods formulates the local planning tasks as an optimization problem. Within this category, there is a wide range of different proposed algorithms like *Ant Colony Optimisation (ACO)* (Lyridis, 2021), *Particle Swarm Optimisation (PSO)* (Kang et al., 2018), and other methods related to *Genetic Algorithms (GA)*. The NMPC discussed earlier, can also be categorized as an optimisation-based method.

Roboat's current implementation of the NMPC mainly functions as a controller for generating the actuator inputs that respect the limits of the thrusters. However, similar hard- and soft constraints can be applied to spatial regions within the input space (Hagen et al., 2018). This method would provide guarantees of collision-free trajectories while respecting the vessel dynamics. The drawbacks of this method are the complexity of formulating the spatial constraints and the computational cost related to the prediction horizon. Larger prediction horizons increase the computational costs, yet generally result in higher quality of the predicted actions.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make optimal decisions by interacting with an environment. This can be done by formulating a problem as a *Markov Decision Process (MDP)* (Sutton and Barto, 2018). Figure 4 illustrates the MDP. Given a state s_t the agent takes an action a_t, which influences the environment. The environment will provide the system's next state s_{t+1}, including feedback on the performed action as a reward

r_{t+1}. This framework allows an agent to learn sequential decision processes by trial and error directly from observations from the environment.

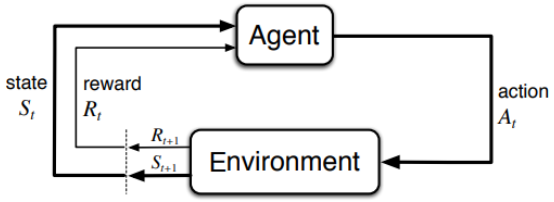


Figure 4: The agent-environment interaction as described by Sutton and Barto (2018).

With the rise of neural networks, the field of *Deep Reinforcement Learning (DRL)* emerged. Combining neural networks as function approximations with RL allows learning high-dimensional action and state spaces (Tesauro, 1994). With advances in network architectures like *Convolutional Neural Networks (CNN)*, it became possible to learn policies directly from sensor inputs to control inputs for the system (Mnih et al., 2013). Learning policies that directly map sensor data to control inputs are often referred to as *end-to-end* methods.

End-to-end

End-to-end DRL has been applied to various control problems over the last few years. Hwangbo et al. (2017) and Song et al. (2023) demonstrate the ability to control quadcopters based on their internal state. While OpenAI et al. (2019) provides an example of how visual input is converted into motor controls for a robotic hand.

Within the maritime sector Waltz and Okhrin (2023) recently, used an end-to-end approach to learn rudder control policies that adhere to the *Collision Regulations (COLREG)* for maritime surface vessels.

End-to-end approaches are interesting since they are relatively simple to implement, and can model complex dynamics and learn to account for external disturbances directly from the data. Therefore eliminating the need to engineer accurate analytical models.

DRL + MPC

While end-to-end approaches have demonstrated the ability to learn complex control policies, these methods are often quite sample-inefficient and lack certain analytical guarantees (Lin et al., 2021). Recent works have therefore started investigating ways to combine DRL and MPC.

Zhang et al. (2016) leverages an MPC during the training phase to learn a robust policy based on expert knowledge and prevent catastrophic failures during training. After training, the MPC is not used anymore and therefore none of the

benefits remain. Romero et al. (2024) solves this by incorporating the MPC as part of the agent's actor policy, providing a way to dynamically set the control parameters for the MPC.

The aforementioned methods primarily aim to leverage the model-based characteristics of the MPC to enhance the quality of the DRL policy predictions by preventing the need to learn an already-known dynamics model from scratch. However, these methods do not yet leverage the capability of an MPC to guarantee collision-free trajectories. Therefore Brito et al. (2021) suggests a method where the DRL agent recommends a sub-goal to an MPC. The MPC can include additional constraints to prevent the generation of control signals that lead to unsafe trajectories.

Based on the context provided by Roboat and the literature we discussed, we will experiment with the application of DRL in the context of a local planner. Specifically, we will implement an end-to-end method and a combination of DRL and MPC using the sub-goal recommendation strategy. In the next chapter, we will describe the setup for both of our methods.

3 Method

As discussed in Sections 1 and 2, we consider our agent to be controlling a maritime surface vessel within a 2D environment $\mathcal{W} \subset \mathbb{R}$, where both static and dynamic obstacles $\mathcal{O}_{\text{obst}} = \mathcal{O}^{\text{stat}}_{\text{obst}} \cup \mathcal{O}^{\text{dyn}}_{\text{obst}}$ can be present at any location. The vessel that the agent controls will be referred to as the *own ship (OS)*. Vessels controlled by other agents are referred to as *target ships (TS)* and are part of the dynamic obstacles $\mathcal{O}^{\text{dyn}}_{\text{obst}}$. Quays, buoys, and docks are examples of static obstacles $\mathcal{O}^{\text{stat}}_{\text{obst}}$. The location of the OS at time step t is defined by $\mathbf{p}_t = [x_t, y_t, \theta_t, \dot{x}_t, \dot{y}_t, \omega_t]^T$, where x_t, y_t represent the x and y locations, θ the heading, and $\dot{x}_t, \dot{y}_t, \omega_t$ the rate of change over each time step i.e. velocities of the earlier mentioned variables. A vector such as $\mathbf{p}_t$ that defines these six parameters will be called a *pose*.

The agent's main task is to navigate the OS from its *current pose* $\mathbf{p}_t$ to a *target pose* $\mathbf{d}_t$ provided by the *global planner*, without intersecting with any of the observed obstacles $\mathcal{O}_{\text{obst}}$. This can be modeled as an *MDP* as illustrated in Figure 5. The rest of this section will discuss the MDP modeling in more detail, starting with the representation of the observations, actions, and definitions of the reward and transition functions, followed by the agent's policy architecture, and training procedures.

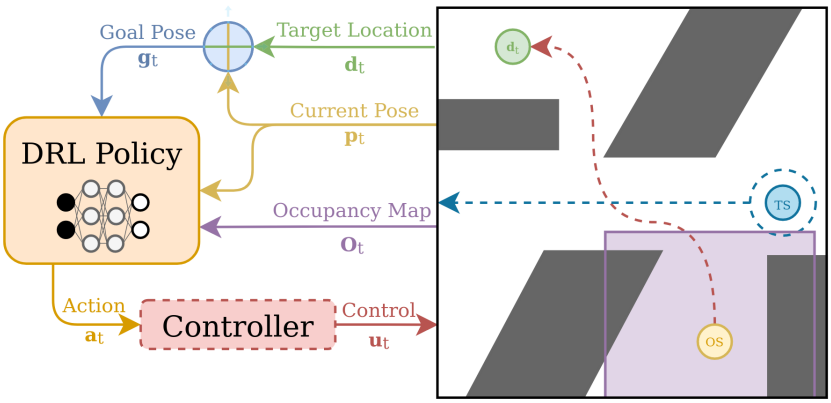


Figure 5: Conceptual view of the agent's observations and actions. The agent navigates the OS towards the target location $\mathbf{d}_t$. The destination $\mathbf{d}_t$, received from the global planner is represented in the world frame. This is converted to a goal $\mathbf{g}_t$ relative to the OS current pose $\mathbf{p}_t$. The agent also observes $\mathcal{O}_t$ which is an ego-centric occupancy grid including both static and dynamic obstacles. The agent provides actions $\mathbf{a}_t$ that can either be directly applied to the environment or processed by an optimization-based controller to a control input $\mathbf{u}_t$ as will be discussed in Section 3.1.2

3.1 MDP Modeling

The environment's *state* consists of $\mathbf{p}_t$, $\mathbf{d}_t$, and $\mathcal{O}_{\text{obst}}$. In this section's introduction, these elements were defined in the global coordinate frame to allow us to interact with the global planner. However, solving the local planning task greatly benefits from re-defining these elements relative to the agents' reference frame.

3.1.1 Observation

As illustrated in Figure 5, we first define the *goal pose* $\mathbf{g}_t = \mathbf{p}_t - \mathbf{d_t}$ as the difference between the current and target pose. Secondly, we convert $\mathbf{p}_t$ to a relative variant $\mathbf{p}'_t$ by simply setting the x and y components to zero. Finally, $\mathcal{O}_{\text{obst}}$ will be presented to the agent as an ego-centric probabilistic *occupancy grid* $\mathcal{O}_t$ (Thrun et al., 2005). This grid is a local representation of $\mathcal{W}$ where the value on each grid cell represents the probability of an obstacle at that coordinate. The occupancy grid is centered at the origin of the OS and oriented so that the x-,y-axis aligns with that of $\mathbf{p}'_t$.

Within the simulator, we obtain the probabilistic occupancy grid by rendering a 2D array where a value of 0 or 1 indicates the presence of an obstacle in a given location. We apply a Gaussian blur to create a smooth transition between the obstacles and free space, simulating a probability gradient near obstacles.

When we combine these components, the resulting state that the agent observes at each time step $\mathbf{s}_t$ is represented as:

$$\mathbf{s}_t = [\mathbf{p}'_t, \mathbf{g}_t, \mathcal{O}_t] \tag{2}$$

The translation of reference frames solves several challenges. Eliminating the relation to the world coordinates ensures that each observation is an independent and uncorrelated sample of the environment. This prevents the agent from *memorizing* specific features within the training environments which improves generalizability to previously unseen environments. Furthermore, the relation between states and actions will be simplified due to the alignment of the occupancy grid and the observed poses, where each grid cell correlates with a x, y value in both the $\mathbf{p}'_t$ and $\mathbf{g}_t$.

3.1.2 Actions and Transition Functions

We will be comparing two different methods of controlling the vessel. In both cases, the vessel's transition function can be defined as:

$$\mathbf{p}_{t+1} = f(\mathbf{p}_t, \mathbf{u}_t) \tag{3}$$

Where $f(\mathbf{p}_t, \mathbf{u}_t)$ represents the vessel dynamic as discussed in Section 2, and $\mathbf{u}_t$ is defined as:

$$\mathbf{u}_t = [u_{\text{stern}}, u_{\text{bow}}, u_{\text{star}}, u_{\text{port}}] \tag{4}$$

For the end-to-end control method this results in the following definition for the action: $\mathbf{a}_t = \mathbf{u}_t$, where the action a_t directly feeds into the vessel dynamics simulator.

The other approach uses the MPC to generate the thruster control inputs. In Section 2.1 we discussed that the MPC requires a reference trajectory over which the control inputs can be optimized. Predicting the entire reference trajectory would make the action space of the agent unnecessarily large. Instead, we only predict the final pose, referred to as the *sub-goal*. The remaining trajectory is then interpolated between the agent's current pose and the sub-goal pose as illustrated in Figure 6a. The details of the interpolation method used can be found in Appendix A.

We will define the recommended sub-goal relative to the agent resulting in the following definition of the agent's action:

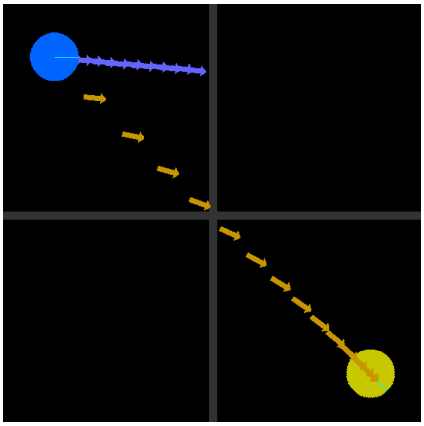
$$\mathbf{a}_t = [r_{t+n}, \Delta\theta_{t+n}], \tag{5}$$

where $\Delta\theta$ represents a change in the angle of the vessel and r is the distance at which the next sub-goal should be placed. $t + n$ indicates that the sub-goal represents the desired position at a fixed n steps into the future. Both r and $\delta\theta$ are bounded, resulting in a cone-shaped area in front of the vessel as illustrated in Figure 6b. This limits the action space and therefore reduces the complexity of the problem.

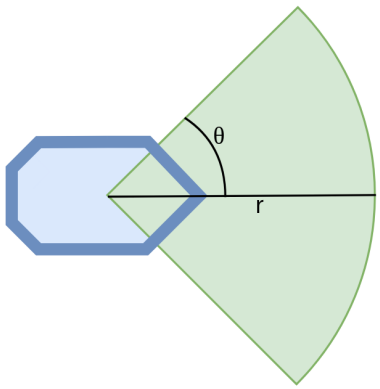
The recommended sub-goal results in the reference trajectory which is provided to the MPC that returns a set of control inputs U. When applied, these control inputs result in the trajectory illustrated as the purple trace in Figure 6a. This brings us to the definition of the control inputs while using the MPC:

$$\mathbf{u}_t = U_0, \tag{6}$$

where the first element of U is used as the control input for the vessel dynamics simulator.



(a) Interpolation of the reference trajectory based on a sub-goal



(b) Bounded area for the sub-goal recommendation

Figure 6: Sub-goal recommendation area and reference trajectory generation. *(a)* We see the agent indicated by the blue circle, the yellow circle is the recommended sub-goal. The orange trace of arrows is the interpolated reference trajectory. The purple trace of arrows is the predicted trajectory according to the MPC. *(b)* We see the area in which the agent can recommend a sub-goal. $\Delta\theta$ indicates the change in angle relative to the vessel's center. r indicates the distance between the vessels current position and the new sub-goal recommendation.

3.1.3 Reward

The main objective of the local planner is to guide the OS to the target provided by the global planner. Defining both $\mathbf{p}_t$ and $\mathbf{d}_t$ as poses allows us to measure the distance between these two poses. Therefore, minimizing the distance between $\mathbf{p}_t$ and $\mathbf{d}_t$ should maximize the reward. Additionally, due to the ego-centric nature of the observations, we can give the agent additional feedback based on whether the agent is moving closer, or further away from the target. Equation 7a therefore defines r_{step} as the progress towards the target by subtracting the Euclidian distances between $\mathbf{p}_t$ and $\mathbf{p}_{t+1}$ and $\mathbf{d}_t$.

To incentivize the agent to take the shortest route possible we deduce the distance traveled between the two states as defined in Equation 7b. Lastly, we penalize the agent for coming into close proximity to an obstacle. This is measured by the probability of an obstacle at the agent's location based on the occupancy map.

$$r_{\text{step}} = \|\mathbf{p}_t - \mathbf{d}_t\|_2 - \|\mathbf{p}_{t+1} - \mathbf{d}_t\|_2 \tag{7a}$$

$$r_{\text{dist}} = \|\mathbf{p}_t - \mathbf{p}_{t+1}\|_2 \tag{7b}$$

$$r_{\text{obs}} = P_{obs}(\mathbf{p}_{t+1}) \tag{7c}$$

$$r(s_t, a_t) = r_{delta} - \alpha r_{dist} - \beta r_{obs} \tag{7d}$$

The complete reward function as defined by Equation 7d adds two additional parameters (α and β) to scale the relative importance of each component. Figure 7 illustrates the field generated by this reward function.

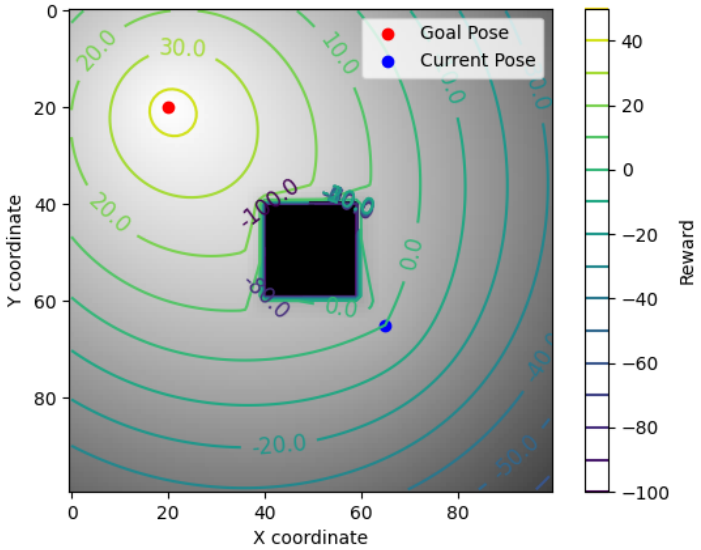


Figure 7: The field produced by the reward function based on the agent's current pose and the goal pose. For this illustration, $\alpha = 0.3$ and $\beta = 10$ were used. We see the contour lines at an interval of 10. The contour lines are elliptically shaped due to $\alpha > 0$. A second 0-contour line can be observed between the agent and the obstacle due to $\beta > 0$.

3.2 Model Architecture

The policy network architecture as illustrated in Figure 8 uses a CNN to extract the features from the occupancy map. The CNN architecture is derived from the *NatureCNN* architecture as formulated by Mnih et al. (2015). The final layer of this CNN is a fully connected layer that provides the feature embedding. This embedding is concatenated with the pose information of the OS p_t and the target pose g_t. The resulting vector is passed through a 2-layered MLP with 265 Neurons each. Finally, the architecture splits into an *actor head* responsible for defining the mean and standard deviations of the actions, and a *critic* representing the value of a specific state.

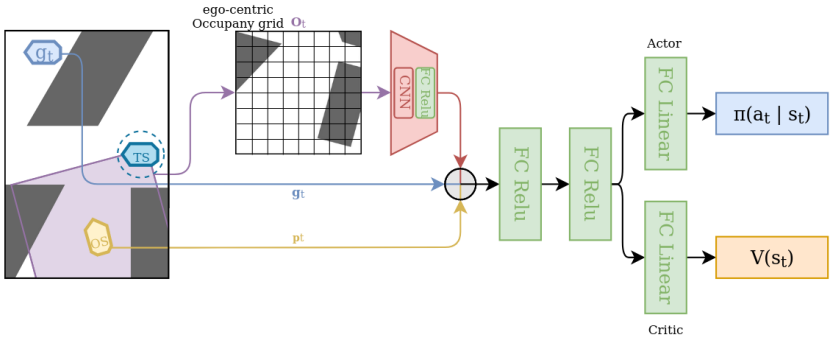


Figure 8: The policy network architecture. A CNN encoder followed by a fully connected layer with a ReLu activation processes the most recent occupancy map O_t. The resulting embedding is concatenated with g_t and p_t and passed through a two-layered MLP before splitting in the actor and critic heads.

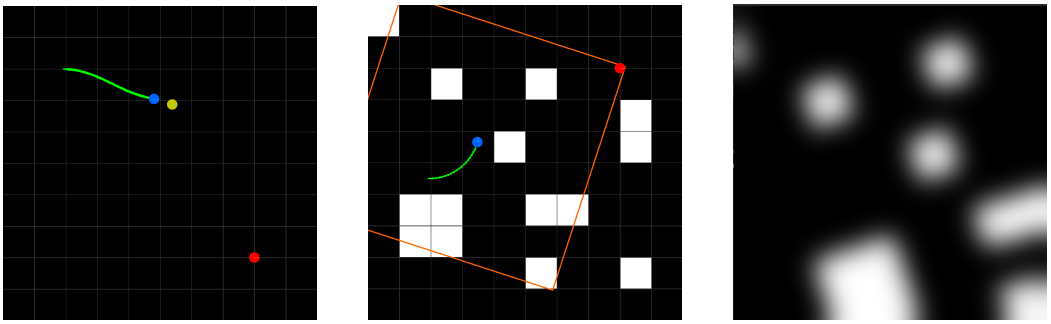
4 Experimental Setup

This section presents the experimental setup for comparing the different control methods and the performance of the models against Roboats' current implementation. We will discuss the different environments, as well as the training parameters.

4.1 Training Environments

We have constructed two environments to evaluate if our method can perform the local planning task. The first environment will contain no obstacles and will be used to evaluate the differences in learning a direct *control policy* i.e. end-to-end policy, and a sub-goal recommendation policy. The second environment will contain various pseudo-random obstacles for the agents to avoid.

Figures 9a and 9b represent the empty environment and the environment containing the obstacles respectively. Both environments represent the agent as a blue circle, a small line indicates the current heading of the agent. The goal and sub-goal are illustrated with similar red and yellow icons, including a line to indicate the desired heading. The red square around the agent in Figure 9b represents the area of the environment that is converted to the *occupancy map*. An example of the occupancy map is illustrated in Figure 9c. Obstacles are indicated as white surfaces. A collision is registered when the center of the agent intersects with one of these surfaces. Finally, the trail of green arrows following the agent illustrates the trajectory traversed by the agent.



(a) Empty Environment **(b)** Static Environment **(c)** Observation

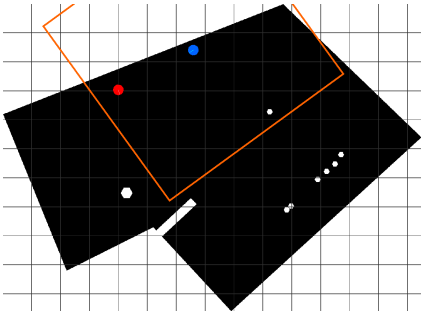
Figure 9: Variations illustrations of the training environment (a) An empty environment (b) Several pseudo-random white surfaces indicate obstacles (c) A rendering of the obstacle map.

4.2 Baselines and Evaluation Metrics

Separate agents will be trained to evaluate both the *direct-control policy* and the *sub-goal recommendation policy* as discussed in Section 3. Since the MPC can generate control inputs it is not strictly necessary for the agent to learn the vessel dynamics when using the MPC controller. Therefore, we will train two agents using the sub-goal recommendation policy. One version will be trained including the simulation of the vessel dynamics. While the other will transition directly to the location of the predicted sub-goal.

Both a quantitative and qualitative comparison between the agents will be conducted. The quantitative evaluation will compare the learning curves for the reward and success rates of the different agents. The qualitative study will investigate the abilities of the sub-goal recommendation policy trained without the vessel dynamics by reviewing the traversed trajectories of the agent.

The latter study will also include a comparison to Roboat's current implementation. This will be done using an environment based on Roboat's real-world test area as illustrated in Figure 10.



(a) Simulated Environment

(b) Satelite Reference

Figure 10: Environment based on Roboat's Real-World test area.

4.3 Training Procedure

We train our agents using Soft Actor-Critic (Haarnoja et al., 2018), as implemented by the Stable Baselines 3 framework (Raffin et al., 2021). The agent's pose is randomly initialized at the start of each episode in one of Section's 4.1 environments.

All agents are trained for $N_{\text{train}} = 1 * 10^6$ time steps. An episode length of $n_{\text{episode}} = 500$ is used for agents trained with the dynamics simulation enabled. The agent trained without the vessel dynamics is trained with $n_{\text{episode}} = 100$.

We use a form of curriculum learning (Bengio et al., 2009) to increase the complexity during training by gradually increasing the maximum distance r_{max} separating the initial position of the agent, and the generated target pose. r_{max} will increase according to the following schedule:

$$r_{\text{max}} = \begin{cases} r_{\text{init}} & \text{if } t < N_{\text{start}} \\ r_{\text{init}} + (r_{\text{final}} - r_{\text{init}}) \left(\frac{t - N_{\text{start}}}{N_{\text{end}} - N_{\text{start}}} \right)^{0.5} & \text{if } N_{\text{start}} \leq t \leq N_{\text{end}} \\ r_{\text{final}} & \text{if } t > N_{\text{end}} \end{cases} \tag{8}$$

Here r_{init} and r_{final} are the initial and final values for r_{max}. For the time t between $N_{\text{start}} \geq 0$ and $N_{\text{end}} \leq N_{\text{train}}$ the function smoothly transitions from r_{init} to r_{final}. The $\left(\frac{t - N_{\text{start}}}{N_{\text{end}} - N_{\text{start}}} \right)^{0.5}$ term scales this transition with a square root, therefore relatively more timesteps are spent on larger values for r_{max}. This compensates for the exponential increase in initial conditions as the maximum distance increases.

The agents trained to compare different methods will all use a static $r_{max} = 10$. Agents used for the qualitative studies will be trained with a curriculum schedule allowing for larger separations between the initial pose and the target destination. Table 1 describes the parameters used for this schedule and Appendix B discusses the effects of using this curriculum schedule.

The remaining parameters we need to define are:

- The size of the observed area as discussed in Section 4.1 is set to 75 meters in width and height.
- The maximum radius r and change in angle $\Delta\theta$ that forms the coned shape area as described in Figure 6b are set to $r \in [0, 10]$ and $\Delta\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ respectively.

Finally, unless stated otherwise, we will use the dynamics model for *Lucy* in all experiments.

Parameter	**Value**	**Unit**
N_{start}	$300 * 10^3$	
N_{end}	$800 * 10^3$	
r_{init}	10	meters (m)
r_{final}	50	meters (m)

Table 1: Curriculum schedule parameters

For the simulation of the vessel dynamics and solving the non-linear optimization control problem i.e. the NMPC solver, we use the Acados-simulator and -solver (Verschueren et al., 2021). To run the experiments we used either a laptop with an *Intel i7-13700H*, a *Nvidia RTX4060* with 8GB of VRAM and 32GB of RAM, or part of the Snellius cluster containing an *Intel Xeon Platinum 8360Y*, a *Nvidia A100* with 40GB of VRAM and requested 32GB of RAM.

5 Results

In this section, we will discuss the experimental results. For each environment, we will first discuss the quantitative results comparing the learning curves for each agent's reward and success rates. Each experiment is run 3 times, the resulting curves are averaged, and the standard deviation is displayed along with this average in each graph. Secondly, we will discuss qualitative results by presenting the trajectories as performed by the agents.

5.1 Empty Environment

As discussed in Section 4, we will first train our agents in an empty environment. To evaluate each method's performance on the basic navigational task.

Learning Curves

Figure 11 shows the learning curves for the rewards of the three agents trained in the empty environment. The sub-goal recommendation policy trained without a dynamics model represented by the blue curve quickly learns the task. This is expected since the agent's actions are always consistent with the transition to the next state.

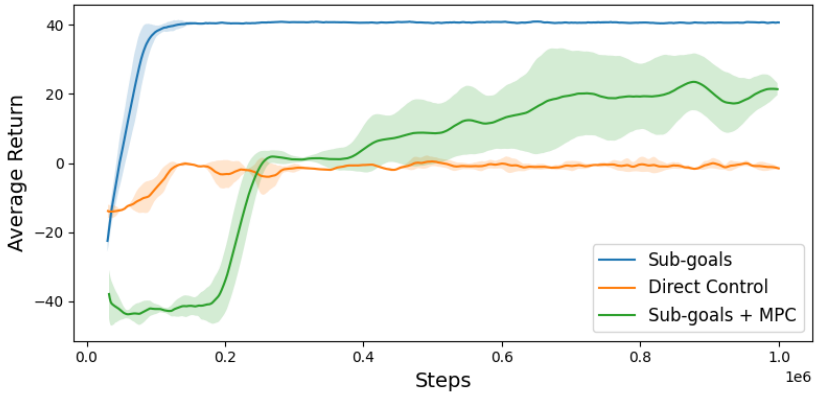


Figure 11: Average return during training in an empty environment. The blue, orange, and green lines show the learning curves of agents without vessel dynamics, direct-controlled, and MPC-controlled vessels respectively. On the x-axis, we see the number of training steps. The y-axis represents the average return using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation.

The other two agents perform notably worse. Learning the sub-goal recommendation policy along with the vessel dynamics proves to be more challenging than when not considering the dynamics. This is also expected for a similar reason as before. In this scenario, the effect of each sub-goal recommendation depends on all prior actions, while each action only has a small direct effect on the agent's

more complex state. Similarly, the *direct control* approach suffers even more from this since the action space is much larger, and the agent is required to learn the dynamics discussed in Section 2.1.

The learning curves of the dynamics-enabled sub-goal recommendation policy show some interesting behavior. Initially, it performs worse than any other policy, after which it quickly jumps up to a new plateau and slowly starts improving again. When we compare this reward curve to the success rate as illustrated in Figure 12. We notice that the jump to this intermediate plateau aligns with the start of the increase in success rate.

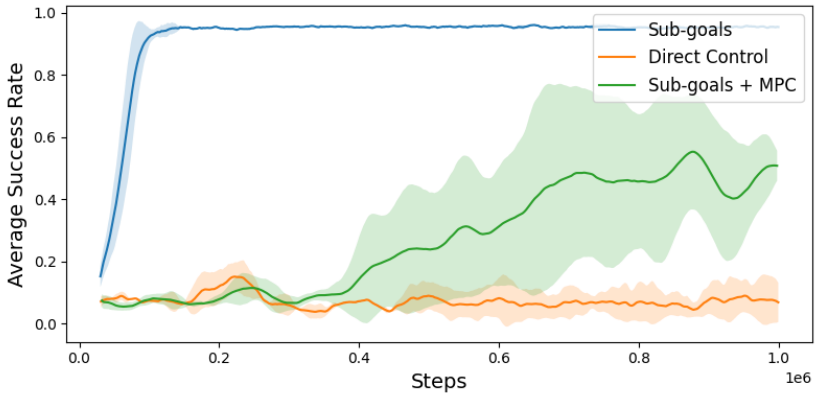
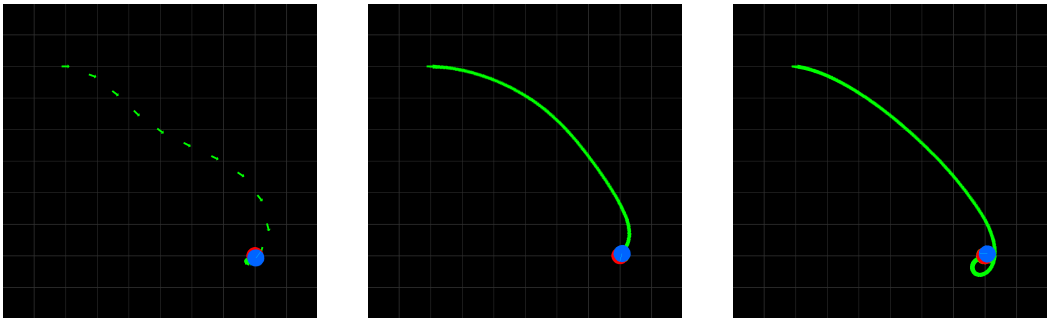


Figure 12: Average success rates during training in an empty environment. The blue, orange, and green lines show the learning curves of agents trained without vessel dynamics, direct-controlled, and MPC-controlled vessels respectively. On the x-axis, we see the number of training steps. The y-axis represents the percentage of rollouts in which the agent reached the target, i.e. the success rate, determined using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation.

Agent Performance

In Section 4.2 we discussed that the sub-goal recommendation method would not strictly require training with the vessel dynamics enabled. Instead, the learned policy of this agent should be able to navigate the vessel, regardless of the specific vessel characteristics. Figure 13 illustrates this by comparing the agent's trajectory while directly transitioning between sub-goals in Figure 13a and the trajectories performed by the same agent only now including the MPC and simulation of the vessel dynamics in Figures 13b and 13c. The latter two figures represent the trajectories while simulating the dynamics of *Lucy* and *Tony* respectively.

This demonstration confirms that the agent can navigate the vessel leveraging the MPC's dynamics model. Looking more closely at the results, we can see that when Tony's dynamics are used the agent first overshoots the target position,



(a) Direct transition between sub-goals

(b) Simulation of Lucy's vessel dynamics

(c) Simulation of Tony's vessel dynamics

Figure 13: A comparison of trajectories generated by the same agent *trained* without vessel dynamics.

forcing it to make a small additional curve to get back to the goal pose. The agent predicts a sub-goal assuming that it can reach this position. However, since the agent is not trained to take the current velocity and the inertia of the vessel into account, it cannot compensate for the additional effort required to make a course correction. We can also see that this effect is more pronounced with Tony's dynamics than with Lucy's, which can be explained by the size difference between Lucy and Tony as shown in Figure 2.

5.2 Static Environment

The second environment includes obstacles for the agent to avoid. Within this environment, the agent is required to learn what features are important from the occupancy map to prevent a collision.

Learning Curves

Figure 14 illustrates the average reward per rollout during training for all three agents. Similar to the previous results both the direct control method and sub-goal recommendation with vessel dynamics do not perform well within this environment. This can be expected since the task has become harder with the additional obstacles. This increase in task complexity is also reflected in the learning curve of the remaining agent where the increase in reward is less steep than in the previous experiment. Predictably, the only method able to reliably reach the target positions is the agent trained without the dynamics enabled. This is illustrated by the success rate curves in Figure 15.

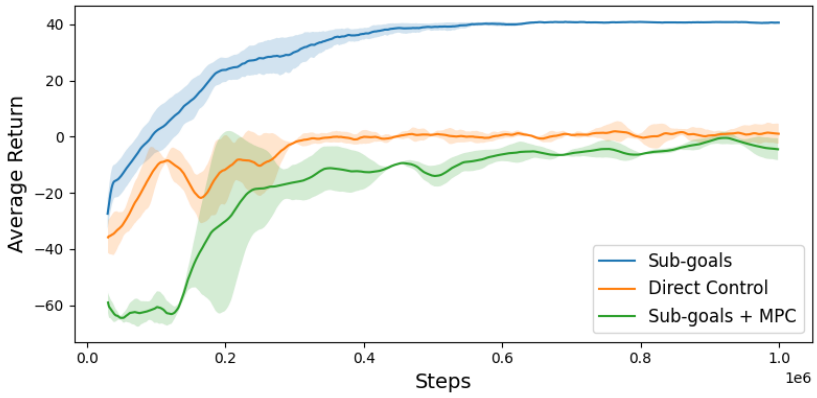


Figure 14: Average return during training in an environment with obstacles. The blue, orange, and green lines show the learning curves of agents trained without vessel dynamics, direct-controlled, and MPC-controlled vessels respectively. On the x-axis, we see the number of training steps. The y-axis represents the average return using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation.

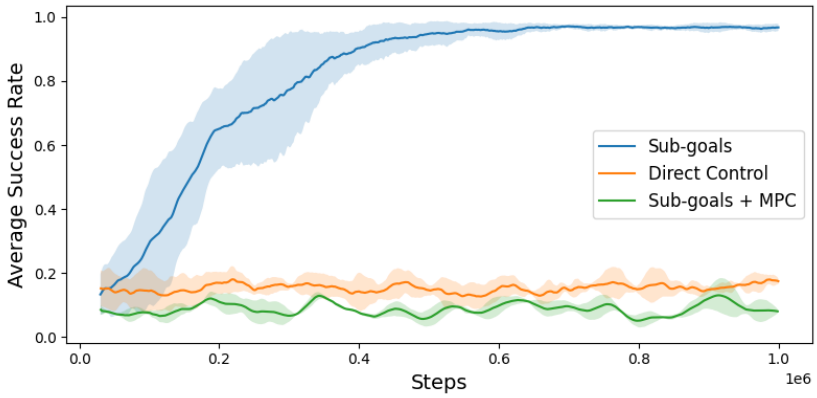
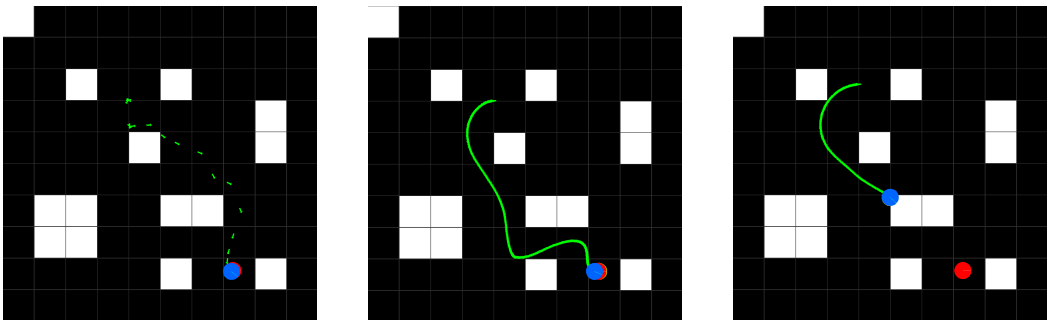


Figure 15: Average success rate during training in an environment with obstacles. The blue, orange, and green lines show the learning curves of agents trained, without vessel dynamics, direct-controlled, and MPC-controlled vessels respectively. On the x-axis, we see the number of training steps. The y-axis represents the percentage of rollouts in which the agent reached the target, i.e. the success rate, determined using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation.

Agent Performance

Figure 16 demonstrates the ability of the sub-goal recommendation method to guide the vessel through the environment while avoiding obstacles. Similar to the example of the empty environment, the figures display an agent trained without the vessel dynamics.



(a) Direct transitions between sub-goals

(b) Successful trajectory with vessel dynamics

(c) Failed trajectory with vessel dynamics

Figure 16: A comparison of trajectories generated by an agent trained without vessel dynamics in an environment with obstacles.

In Figures 16a and 16b the agent is initialized in the same pose, with the same target pose. The only difference between the two examples is the simulation of the vessel dynamics. We can see that the agent uses a different route due to the changed transitions between the states, yet can still navigate towards the target pose. It even prevents a collision with the obstacle on the bottom center. However, Figure 16c shows that this method does not provide a guaranteed collision-free trajectory. In this case, the agent found itself in a position where nearly the entire action space would result in a sub-goal that intersects with the obstacle. Along with some residual motion from the previous states, this resulted in a collision.

5.3 Roboat Environment

The final experiment will compare the current A* implementation to the proposed DRL method. For this experiment, we used an agent trained *without* the vessel dynamics and in the environment with static obstacles discussed in Sections 4.1 and 5.2. Figure 17 illustrates the results of this comparison. The red waypoints create a parkour that forces the agent to avoid obstacles between waypoints 2 and 3, and waypoints 4 and 5. In blue we see the trajectory as a result of Roboat's current A* + MPC implementation. We compare this trajectory to the DRL + MPC method represented by the green and orange traces.

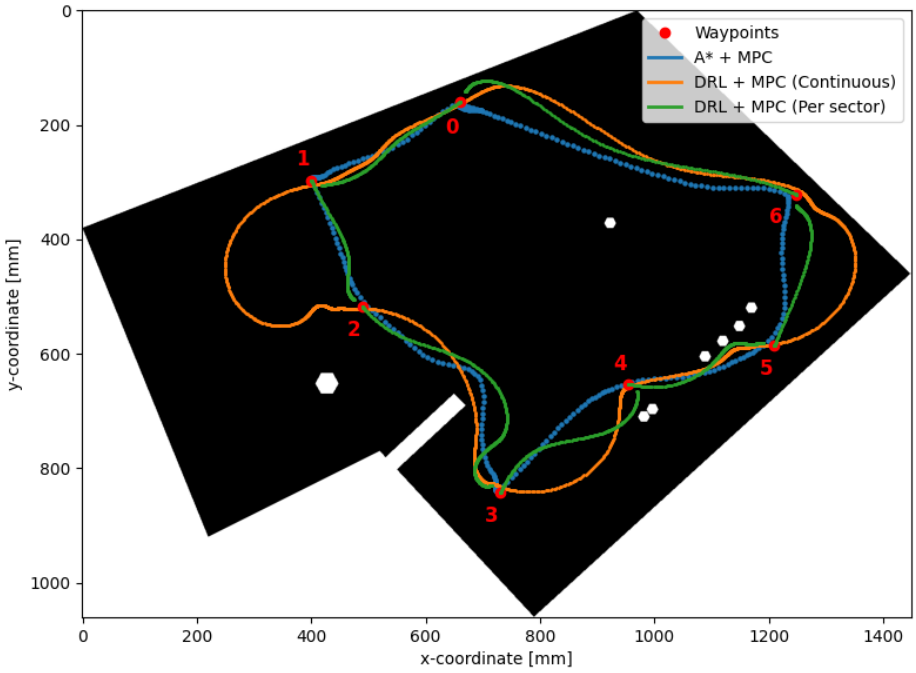


Figure 17: A comparison between the proposed DRL method and the current A* implementation. In red we see the waypoints that substitute the global planner inputs. The blue trace is the resulting trajectory using the current A* implementation. The orange and green traces illustrate the performance of the DRL method. The green trace is created by reinitializing the agent upon reaching a waypoint to reset all the velocities. The orange trace demonstrates the trajectory without re-initialization.

We can see that the DRL+MPC method can traverse the parkour without colliding with any of the obstacles present in the environment. This demonstrates that our agent has the ability to generalize to previously unseen environments. For the initial experiment, we let the agent execute the entire parkour without resetting the vessel's state once a waypoint had been reached. This *continous* run is illustrated by the orange trace.

Since the agent was trained without the vessel dynamics, it does not consider the

current velocity and inertia of the vessel. The effects of this can be best observed from the trace between waypoints 1 and 2 where the vessel has an initial speed and heading resulting from traveling between waypoints 0 and 1. The resulting trace makes a large arch where the minimum turn radius is produced by the shape of the action space and the generation of the reference trajectory for the MPC as discussed in Section 3.1.2.

Since the A* method briefly holds its position to reorient itself on each waypoint, a better comparison with the DRL method can be made if we do something similar. Therefore, we ran a second experiment where we reinitialized the vessel state each time a waypoint was reached. This resulted in the green trajectory in Figure 17. We can see that the resulting curves in sections between waypoints 0, 1, and 2 are the most similar to the baseline. The sector from waypoint 2 to 3 starts with a very similar path, however, as the vessel picks up speed the trajectories start deviating and the same overshooting behavior as discussed in Section 5.1 can be observed near the end.

6 Discussion

We found that a DRL agent trained to recommend sub-goals might be a viable strategy for local planning based on occupancy maps. However, this approach suffers from some limitations. First, When we train the agent without the dynamics model, it is implied that the vessel can move in a straight line to the new sub-goal. Therefore the agent will learn to predict sub-goal recommendations that will avoid collisions assuming a straight trajectory. However, the MPC optimizes a set of control inputs that *minimizes* the *deviation* between a reference, and a feasible trajectory while subjected to a set of constraints applied to the dynamics model. This can lead to quite significant deviations between the reference and predicted trajectory. Therefore this method cannot *guarantee* collision-free trajectories.

It might be possible to reduce the probability of the MPC generating an unsafe trajectory, by training the agent along with the predictions of the MPC. Therefore the agent can take additional information like the velocities into account. We observed that this task is much more challenging and might require significantly more training steps or an alternative training strategy. Even in this scenario, since the MPC is responsible for computing the thruster control inputs, as long as no additional constraints are applied while solving the optimal control problem there will always be a possibility of generating an unsafe trajectory.

Secondly, by limiting the area in which a sub-goal can be predicted we also limit the number of options the agent has to navigate the vessel. In this cone-shaped configuration of the action space, we do not use the full omnidirectional capabilities of the vessel. As a result, a minimal turn radii emerges and the agent cannot travel backwards. Redefining the action space by predicting a relative x,y, and heading coordinates for the sub-goal could provide more flexibility but will introduce more complexity.

Besides these limitations, we should also note that the minimal turn radii and the overall behavior of the agent during deployment with the MPC enabled are largely determined by the method used for generating the reference trajectory. For example, smaller turn radii or faster acceleration are possible by using different interpolation strategies. Training the agent without enabling the MPC therefore allows you to influence the behavior characteristics of the vessel afterwards.

7 Conclusion

Within this work, we discussed different strategies for using DRL for local planning based on occupancy maps. We proposed a policy network architecture and trained multiple agents to compare the different strategies to each other as well as to Roboat's current implementation.

To answer our first research question, how to model the problem in the MDP, we compared an end-to-end control approach and a method learning sub-goal recommendations for an MPC to separate the planning and control tasks. We found that learning an end-to-end policy for direct control of the thrusters is not feasible in the current setup. Therefore the alternative of learning to predict sub-goals from which a reference trajectory is generated for an MPC proved better suited for this task. We demonstrated that a model of the target vessel's dynamics is not strictly required to learn the sub-goal recommendation policy. Although the learned policy can avoid obstacles, we observed that this does not yet guarantee collision-free trajectories.

The second research question aimed to determine how a DRL method would compare to the current A*-based implementation of the local planner. Considering the current implementations of both methods, we observed that the A* planner can produce a more efficient trajectory within the test area. Nevertheless, we demonstrated that the proposed DRL + MPC method can generalize to previously unseen environments and was able to navigate the vessel.

Therefore we can conclude that the separation of planning and control is the better reinforcement learning strategy for autonomous navigation of a ferry in the city of Amsterdam. This yields a DRL algorithm that learns a sub-goal recommendation policy which in combination with an MPC is able to navigate the target vessel.

To improve the performance of the DRL + MPC method, future work could consider one or more of the following topics. A better training procedure that considers the full vessel state including the velocities or the embedding of collision avoidance constraints into the MPC to guarantee safe trajectories Additionally, this method can be extended with a temporal memory structure like *frame stacking* or an *LSTM*, allowing it to consider the movement of other agents in a dynamic environment.

A Reference Trajectory Interpolation

In this section, we will discuss the generation of the reference trajectory for the MPC based on the provided sub-goal. Reference trajectory for the MPC consists of a pose definition for every timestep within the prediction horizon. A pose is defined as $p_t = \{x, y, \theta, \dot{x}, \dot{y}, \omega\}$ where x and y determine the location of pose p on a grid, θ provides the angle and $\dot{x}$, $\dot{y}$, and ω the velocities of each component at that time step t. The trajectories are always computed relative to the origin where x, y, and θ are 0. The sub-goal $s_{t+n} = \{x, y, \theta\}$ is therefore always provided relative to this origin and considered the desired location at the end of the prediction horizon n steps into the future.

Figure 18 provides an example for the generated trajectory for sub-goal $s_{t+n} = \{5, 8, -\frac{\pi}{2}\}$ with $n = 30$.

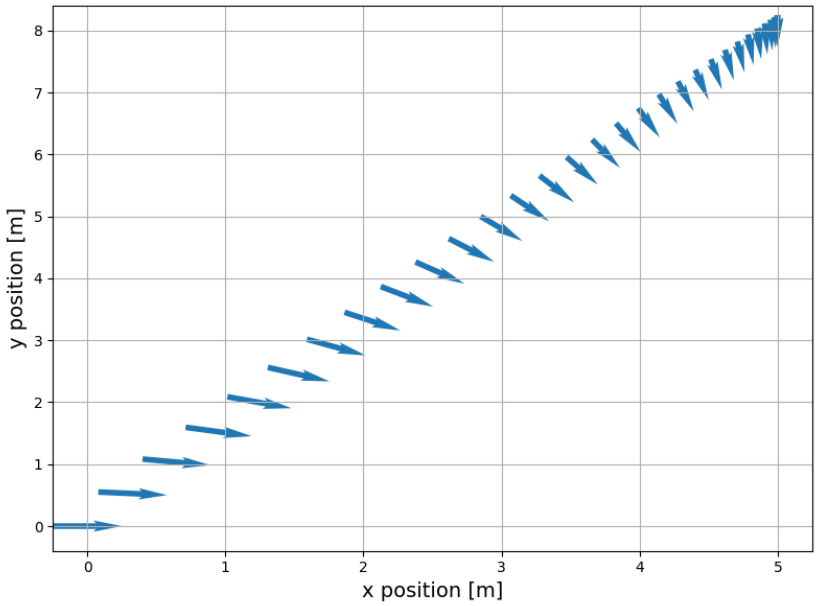


Figure 18: The interpolated path from the origin to the provided sub-goal at $x = 5$, $y = 8$ and $\theta = -\frac{\pi}{2}$. Every arrow indicates the intermediate position and heading at an interval of 100 milliseconds.

This interpolated path is computed in several steps. First, the x- and y-positions are interpolated with a *Piecewise Cubic Hermite Interpolating Polynomial (Pchip-Interpolator)* with a duplicated point at the final time step as illustrated in Figure 19. The duplication of the final time step achieves a smooth transition with a stronger acceleration at the beginning of the trajectory. The PchipInterpolator is used over a *Cubic Spline Interpolator* to prevent overshooting or oscillations.

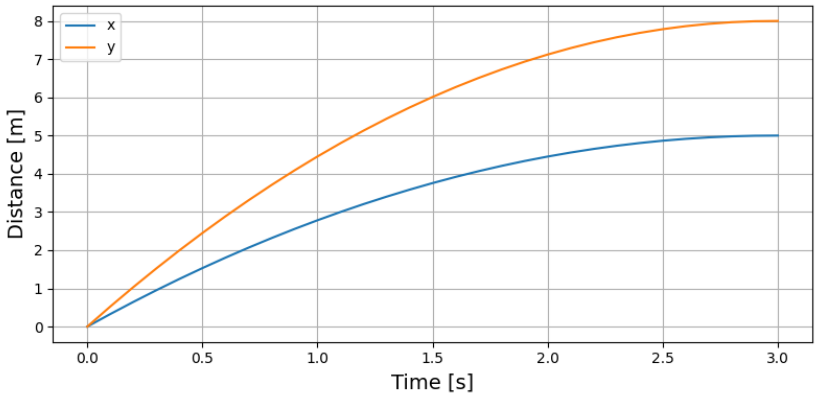


Figure 19: The distance traveled along the x- and y-axis over time.

Secondly, the angle is interpolated in a similar method also using the PchipInterpolator as shown in Figure 20. Additionally, the sin and cosine of the heading are computed and added to the reference trajectory. While solving the optimal control problem, the MPC uses these values to address the modular nature of the heading.

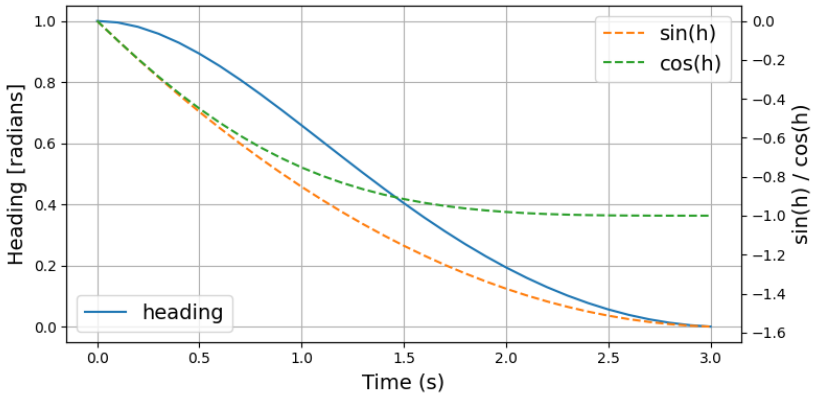


Figure 20: The heading θ over time, displayed along with the sin and cosin of the heading.

Finally, the velocities for each pose in the reference trajectory are to be determined. This is done by taking the derivatives of the x, y, and heading curves as shown above. Since the $\dot{x}$ and $\dot{y}$ are defined relative to the current heading θ, the derivates are translated from to a relative coordinate frame, resulting in the curves as displayed Figure 21.

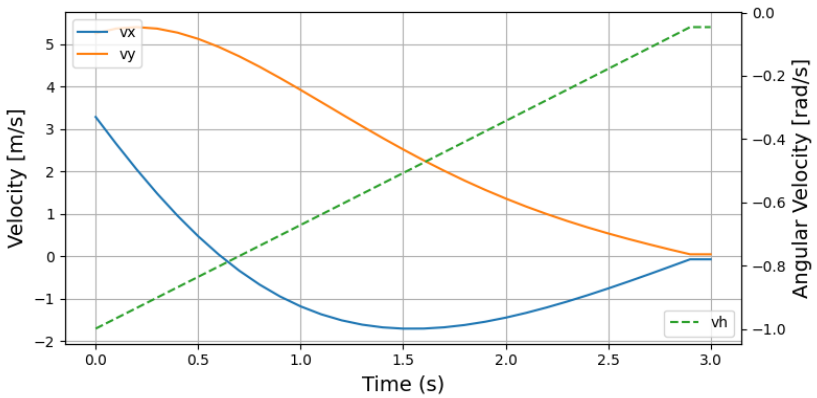


Figure 21: The velocities in x, y and θ over time. In this graph, vx and vy are the velocities relative to the current heading h at each timestep.

B Curriculum Learning

We discussed the use of a curriculum learning method to train our agents when a larger distance between the initial state and the target pose is required. This section will evaluate the benefits of using such a curriculum schedule.

In the figures below we see the learning curves of 2 policies trained in the environment with obstacles. One policy is trained using a static value of $r_{\text{max}} = 50$ which is the maximum value considered during our test. The other is trained using the curriculum schedule as described in Section 4.3. Again for each curve, the experiment was performed 3 times and the figures display the average and standard deviations.

Looking at the average reward achieved during training in Figure 22. We see little difference in the average performance of each trained policy. We do observe that the learning is a bit more stable for the policy trained with the curriculum schedule as indicated by the smaller standard deviation.

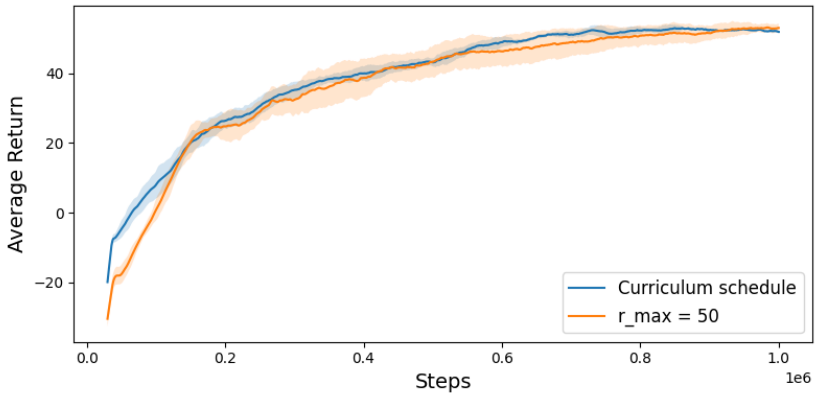


Figure 22: Average return during training. The blue and orange lines represent the policies trained with a curriculum schedule and a fixed value for r_{max} respectively. The x-axis represents the number of training steps and the y-axis the average return using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation

We see a larger difference in performance when we visualize the average success rate in Figure 23. Using the curriculum learning schedule, we see that the success rate achieves its final performance much quicker than the policy trained with the fixed value for r_{max}. Again, we also observe more stable training when using the schedule.

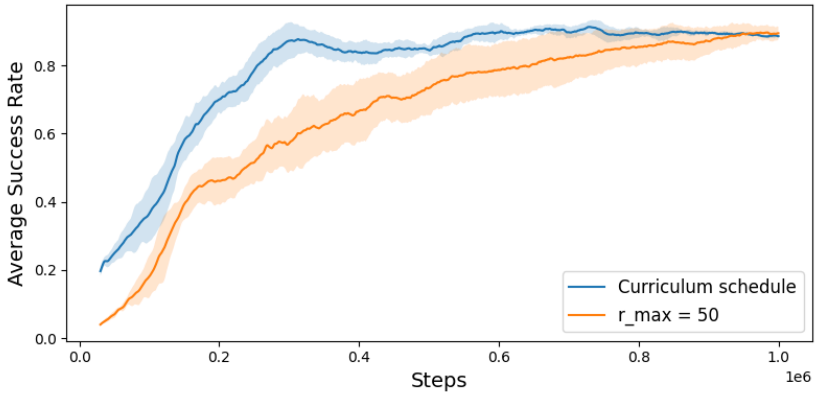


Figure 23: Average success rate during training. The blue and orange lines represent the policies trained with a curriculum schedule and a fixed value for r_{max} respectively. The x-axis represents the number of training steps and the y-axis is the average success rate using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation

The final interesting metric in this scenario is the average episode length achieved by the policy visualized in Figure 24. This indicates, how quickly an agent travels from the initial pose to the target pose. A lower value, generally means that the agent found a more efficient route. Here we see that the policy trained with the curriculum schedule on average always achieves a lower episode length. The temporary increase in episode length around $350 * 10^3$ steps aligns with the start of the increase in r_{max} as discussed in Section 4.3.

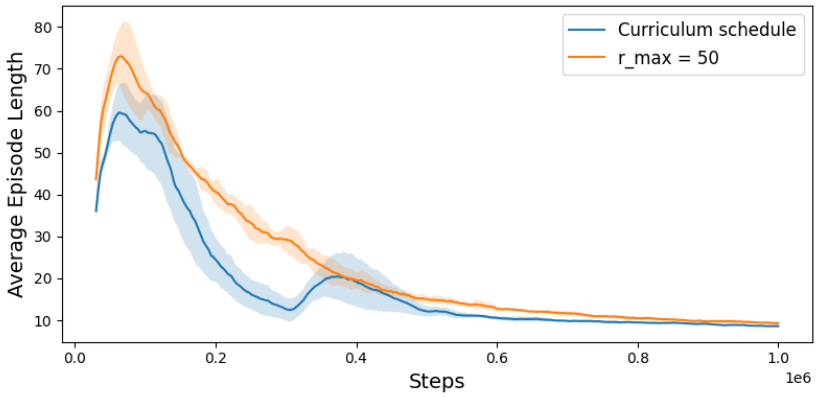


Figure 24: Average episode length during training. The blue and orange lines represent the policies trained with a curriculum schedule and a fixed value for $r_{\max}$ respectively. The x-axis represents the number of training steps and the y-axis is the average episode length using a window of 100 episodes. Each curve is averaged over 3 runs where the shaded area represents the standard deviation

From these results, we can conclude that the addition of the curriculum schedule does not have a significant impact on the final performance of the learned policies. However, in certain scenarios, it can aid in faster convergence and more stable learning.

References

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. *Proceedings of the 26th Annual International Conference on Machine Learning*.

Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). Yolact: Real-time instance segmentation. https://arxiv.org/abs/1904.02689.

Brito, B., Everett, M., How, J. P., and Alonso-Mora, J. (2021). Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):4616–4623.

Calvert, E. S. (1960). Manœuvres to ensure the avoidance of collision. *Journal of navigation*, 13(2):127–137.

DNV GL (2014). Autonomous ships are coming. https://www.dnv.no/karriere/employee-stories/interview-articles/revolt/.

EMSA (2023). Annual overview of marine casualties and incidents 2023. https://www.emsa.europa.eu/publications/item/5052-annual-overview-of-marine-casualties-and-incidents.html.

Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The International journal of robotics research*, 17(7):760–772.

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.

Hagen, I. B., Kufoalor, D. K. M., Brekke, E. F., and Johansen, T. A. (2018). Mpc-based collision avoidance strategy for existing marine vessel guidance systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7618–7623.

Houska, B., Ferreau, H., and Diehl, M. (2011). ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE robotics and automation letters*, 2(4):2096–2103.

Kang, Y.-T., Chen, W.-J., Zhu, D.-Q., Wang, J.-H., and Xie, Q.-M. (2018). Collision avoidance path planning for ships by particle swarm optimization. *Journal of Marine Science and Technology*, 26(6):777–786.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on robotics and automation*, 12(4):566–580.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International journal of robotics research*, 5(1):90–98.

Kufoalor, D. K., Brekke, E. F., and Johansen, T. A. (2018). Proactive collision avoidance for asvs using a dynamic reciprocal velocity obstacles method. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2402–2409.

Lazarowska, A. (2016). A trajectory base method for ship's safe path planning. *Procedia Computer Science*, 96:1022–1031.

Lin, Y., McPhee, J., and Azad, N. L. (2021). Comparison of deep reinforcement learning and model predictive control for adaptive cruise control. *IEEE Transactions on Intelligent Vehicles*, 6(2):221–231.

Lyridis, D. V. (2021). An improved ant colony optimization algorithm for unmanned surface vehicle local path planning with multi-modality constraints. *Ocean Engineering*, 241:109890.

Lyu, H. and Yin, Y. (2019). Colregs-constrained real-time path planning for autonomous ships using modified artificial potential fields. *Journal of Navigation*, 72(3):588–608.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2019). Learning dexterous in-hand manipulation. https://arxiv.org/abs/1808.00177.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Romero, A., Song, Y., and Scaramuzza, D. (2024). Actor-critic model predictive control. https://arxiv.org/abs/2306.09852.

Singh, Y., Sharma, S., Sutton, R., Hatton, D., and Khan, A. (2018). A constrained a* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents. *Ocean Engineering*, 169:187–201.

Song, Y., Romero, A., Müller, M., Koltun, V., and Scaramuzza, D. (2023). Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82).

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: an introduction*. The MIT Press, Cambridge, Massachusetts.

Svec, P., Schwartz, M., Thakur, A., and Gupta, S. K. (2011). Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1154–1159.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press.

Vagale, A., Oucheikh, R., Bye, R. T., Osen, O. L., and Fossen, T. I. (2021). Path planning and collision avoidance for autonomous surface vehicles i: A review. *Journal of Marine Science and Technology*, 26(4):1292–1306.

Verschueren, R., Frison, G., Kouzoupis, D., Frey, J., van Duijkeren, N., Zanelli, A., Novoselnik, B., Albin, T., Quirynen, R., and Diehl, M. (2021). acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*.

Waltz, M. and Okhrin, O. (2023). Spatial–temporal recurrent reinforcement learning for autonomous ships. *Neural Networks*, 165:634–653.

Wang, W., Shan, T., Leoni, P., Fernandez-Gutierrez, D., Meyers, D., Ratti, C., and Rus, D. (2020). Roboat ii: A novel autonomous surface vessel for urban environments. https://arxiv.org/abs/2007.10220.

Zhang, T., Kahn, G., Levine, S., and Abbeel, P. (2016). Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. https://arxiv.org/abs/1509.06791.

Ülkü Öztürk, Akdağ, M., and Ayabakan, T. (2022). A review of path planning algorithms in maritime autonomous surface ships: Navigation safety perspective. *Ocean Engineering*, 251:111010.