IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 21, 2020, accepted February 16, 2020, date of publication February 27, 2020, date of current version March 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976586

Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning

EIVIND MEYER[1], HAAKON ROBINSON[1], ADIL RASHEED[1,2], AND OMER SAN[3]

[1]Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7034 Trondheim, Norway
[2]Department of Mathematics and Cybernetics, SINTEF Digital, 7031 Trondheim, Norway
[3]School of Mechanical and Aerospace Engineering, Oklahoma State University, Stillwater, OK 74078-5016, USA

Corresponding author: Adil Rasheed (adil.rasheed@ntnu.no)

This work was supported in part by the Norwegian Research Council, and in part by the industrial partners through DNV GL, Kongsberg, and Maritime Robotics of the Autosit Project under Grant 295033.

ABSTRACT In this article, we explore the feasibility of applying proximal policy optimization, a state-of-the-art deep reinforcement learning algorithm for continuous control tasks, on the dual-objective problem of controlling an underactuated autonomous surface vehicle to follow an a priori known path while avoiding collisions with non-moving obstacles along the way. The AI agent, which is equipped with multiple rangefinder sensors for obstacle detection, is trained and evaluated in a challenging, stochastically generated simulation environment based on the OpenAI gym Python toolkit. Notably, the agent is provided with real-time insight into its own reward function, allowing it to dynamically adapt its guidance strategy. Depending on its strategy, which ranges from radical path-adherence to radical obstacle avoidance, the trained agent achieves an episodic success rate close to 100%

INDEX TERMS Deep reinforcement learning, autonomous surface vehicle, collision avoidance, path following, machine learning controller.

I. INTRODUCTION

Autonomy offers surface vehicles the opportunity to improve the efficiency of transportation while still cutting down on greenhouse emissions. However, for safe and reliable autonomous surface vehicles (ASV), effective path planning is a pre-requisite which should cater to the two important tasks of path following and collision avoidance (COLAV). In the literature, a distinction is typically made between *reactive* and *deliberate* COLAV methods [1]. In short, reactive approaches, most notably artificial potential field methods [2]–[4], dynamic window methods [5]–[7], velocity obstacle methods [8], [9] and optimal control-based methods [10]–[14], base their guidance decisions on sensor readings from the local environment, whereas deliberate methods, among them popular graph-search algorithms such as A* [15] and Voronoi graphs [16], [17] as well as randomized approaches such as rapidly-exploring random tree [18] and probabilistic roadmap [19], exploit a priori known characteristics of the global environment in order to construct an optimal path in advance, which is to be followed using a low-level steering controller. By utilizing more data than just the current perception of the local neighborhood surrounding the agent, deliberate methods are generally more likely to converge to the intended goal, and less likely to suggest guidance strategies leading to dead ends, which is frequently observed with reactive methods due to local minima [20]. However, in the case where the environment is not perfectly known, as a result of either incomplete or uncertain mapping data or due to the environment having dynamic features, purely deliberate methods often fall short. To prevent this, such methods are often executed repeatably on a regular basis to adapt to discrepancies between recent sensor observations and the a priori belief state of the environment [20]. However, as this class of methods are computationally expensive by virtue of processing global environment data, this is sometimes rendered infeasible for real-world applications with limited processing power [21], especially as the problem of optimal path planning amid multiple obstacles is provably NP-hard [22]. Thus, a common approach is to utilize a

The associate editor coordinating the review of this manuscript and approving it for publication was Dalei Wu.

reactive algorithm, which is activated whenever the presence of a nearby obstacle is detected, as a fallback option for the global, deliberate path planner. Such *hybrid* architectures are intended to combine the strengths of reactive and deliberate approaches and have gained traction in recent years [23], [24]. The approach presented in this article is somewhat related to this; the existence of some a priori known nominal path is presumed, but following it strictly will invariantly lead to collisions with obstacles. Unlike other approaches, there is, however, no switching mechanism that activates some reactive fallback algorithm in dangerous situations. To this end, a reinforcement learning (RL) agent is trained to exhibit rational behaviour under such circumstances, i.e. following the path strictly only when it is deemed safe. Despite the vast amount of literature on the topic and the numerous different approaches, of which only a small subset has been mentioned here, it appears that, when applied to vehicles with nonholonomic and real-time constraints such as autonomous surface vehicles, no existing method is without drawbacks, whether it is unrealistic assumptions about the vessel dynamics (if not an outright neglect thereof), problems with scalability in terms of environment complexity (including the degrees of freedom, the number of obstacles as well as their shapes and their velocities), excessive computation time requirements in general, unrealistic assumptions of availability of measurements, the disregard for desirable output path properties such as continuity, smoothness, feasibility or even safety, an incompatibility with external environmental forces, a lack of determinism (which may or may not be deemed problematic), stability issues due to singularities or local minima leading to sub-optimal guidance strategies [25], [26].

RL is an area of machine learning (ML) of particular interest for control applications, such as the guidance of surface vessels under consideration here. Fundamentally, this ML paradigm is concerned with estimating the optimal behavior for an agent in an unknown, and potentially partly unobservable environment, relying on trial-and-error-like approaches in order to iteratively approximate the behavior policy that maximizes the agent's expected long-time reward in the environment. The field of RL has seen rapid development over the last few years, leading to many impressive achievements, such as playing chess and various other games at a level that is not only superhuman, but also overshadows previous AI approaches by a wide margin [27]–[29].

The focus of this paper is to explore how RL, given the recent advances in the field, can be applied to the guidance and control of ASV. Specifically, we look at the dual objectives of achieving the ability to follow a path constructed from a priori known way-points, while avoiding collision with obstacles along the way. In an end-to-end fashion, control signals for a simulated vessel are generated by a RL agent which, based on the readings from a rangefinder sensor suite which is attached to the vessel as well as rewards received from the environment, learns how to intelligently control the vessel in challenging obstacle avoidance scenarios. The resulting interplay between the environment, which incorporates the dynamics of the vessel itself, and the autonomous RL agent is illustrated in Figure 1.

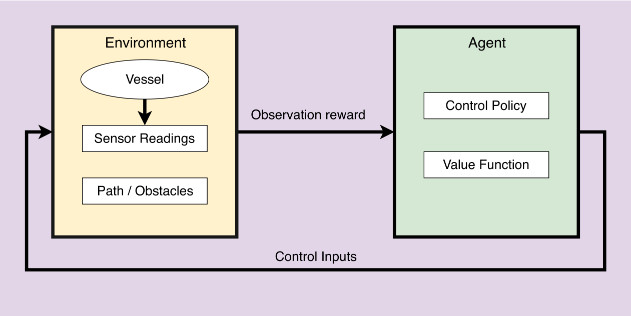


FIGURE 1. Block diagram illustrating the interaction between the environment and the RL agent.

For simplicity, we limit the scope of this work to non-moving obstacles of circular shapes. As RL methods are, model-free approaches, by their very nature, a positive result can bring significant value to the robotics and autonomous system field, where implementing a guidance system typically requires knowledge of the vessel dynamics, in the form of non-linear first-principle models with parameters that can only be determined experimentally at great cost.

II. THEORY

A. GUIDANCE AND CONTROL OF MARINE VESSELS

1) COORDINATE FRAMES

In order to model the dynamics of marine vessels, one must first define the coordinate frames forming the basis for the motion. A few coordinate frames typically used in control theory are of particular interest. The geographical North-East-Down (NED) reference frame $\{n\} = (x_n, y_n, z_n)$ forms a tangent plane to the Earth's surface, making it useful for terrestrial navigation. Here, the x_n-axis is directed north, the y_n-axis is directed east and the z_n-axis is directed towards the center of the earth.

The origin of the body-fixed reference frame $\{b\} = (x_b, y_b, z_b)$ is fixed to the current position of the vessel in the NED-frame, and its axes are aligned with the heading of the vessel such that x_b is the longitudinal axis, y_b is the transversal axis and z_b is the normal axis pointing downwards. It should be noted, that whenever the vessel is aligned with the water surface, a common assumption, z_b points in the same direction as z_n, i.e. towards the center of the Earth.

2) STATE VARIABLES

Following Society of Naval Architects and Marine Engineers (SNAME) notation [30], twelve variables are used for representing the vessel state. The state vector consists of the generalized coordinates $\boldsymbol{\eta} \triangleq [x^n, y^n, z^n, \phi, \theta, \psi]^T$, where the quantities in the bracket are North, East, Down positions in reference frame $\{n\}$, roll, pitch, yaw corresponding to a Euler angle zyx convention from $\{n\}$ to $\{b\}$ respectively, representing the pose of the vessel relative to the inertial frame. Also $\boldsymbol{\nu} \triangleq [u, v, w, p, q, r]^T$, where the quantities in the bracket are surge, sway, heave, roll rate, pitch rate and yaw rate respectively representing the vessel's translational and angular velocity in the body-frame.

3) DYNAMICS

Assumption 1 (Calm Sea): There is no ocean current, no wind and no waves and thus no external disturbances to the vessel.

In the general case, twelve coupled, first-order, nonlinear ordinary differential equations make up the vessel dynamics. In the absence of ocean currents, waves and wind, these can be expressed in a compact matrix-vector form as

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{B}\boldsymbol{f} &= \mathbf{M_{RB}}\dot{\boldsymbol{\nu}} + \mathbf{C_{RB}}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) \text{ (rigid-body, hydrostatic)} \\ &\quad + \mathbf{M_A}\dot{\boldsymbol{\nu}} + \mathbf{C_A}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} \text{ (hydrodynamic)} \end{aligned} \tag{1}$$

Here, $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$ is the transformation matrix from the body frame $\{b\}$ to the NED reference frame $\{n\}$. $\mathbf{M_{RB}}$ and $\mathbf{M_A}$ are the mass matrices representing rigid-body mass and added mass, respectively. Analogously, $\mathbf{C_{RB}}(\boldsymbol{\nu})$ and $\mathbf{C_A}(\boldsymbol{\nu})$ are matrices incorporating centripetal and Coriolis effects. Finally, $\mathbf{D}(\boldsymbol{\nu})$ is the damping matrix, $\boldsymbol{g}(\boldsymbol{\eta})$ contains the restoring forces and moments resulting from gravity and buoyancy, $\mathbf{B}$ is the actuator configuration matrix and $\boldsymbol{f}$ is the vector of control inputs.

4) 3-DOF MANEUVERING MODEL

In this subsection, the ASV assumptions and the resulting 3-DOF model is outlined.

Assumption 2 (State Space Restriction): The vessel is always located on the surface and thus there is no heave motion. Also, there is no pitching or rolling motion.

This assumption implies that the state variables z^n, ϕ, θ, w, p, q are all zero. Thus, we are left with the three generalized coordinates x^n, y^n and ψ and the body-frame velocities u, v and r. In this case, the transformation matrix $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$ is reduced to a basic rotation matrix $\mathbf{R}_{z,\psi}$ for a rotation of ψ around the z_n-axis as defined by

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Furthermore, since restoring forces are unimportant for 3-DOF maneuvering [31], we have that $\boldsymbol{g}(\boldsymbol{\eta}) = \mathbf{0}$. Also, by combining the corresponding rigid-body and added mass terms associated such that $\mathbf{M} = \mathbf{M_{RB}} + \mathbf{M_B}$ and $\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C_{RB}}(\boldsymbol{\nu}) + \mathbf{C_A}(\boldsymbol{\nu})$, we obtain the simpler 3-DOF state-space model

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{R}_{z,\psi}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \mathbf{B}\boldsymbol{f} \end{aligned} \tag{2}$$

where $\boldsymbol{\eta} \triangleq [x^n, y^n, \psi]^T$ and $\boldsymbol{\nu} \triangleq [u, v, r]^T$ and each matrix is 3x3.

Assumption 3 (Vessel Symmetry): The vessel is port-starboard symmetric.

Assumption 4 (Origin at the Centerline): The body-fixed reference frame $\{b\}$ is centered somewhere at the longitudinal centerline passing through the vessel's center of gravity.

Assumption 5 (Sway-Underactuation): There is no force input in sway, so the only control inputs are the surge thrust T_u and the yaw moment T_r.

Assumptions 3 and 4, which are commonly found in maneuvering theory applications, justify a sparser structure of the system matrices, where some non-diagonal elements are zeroed out. Also, from Assumption 5 we have that $\boldsymbol{f} \triangleq [T_u, T_r]^T$. The matrices and their numerical values are obtained from [31], where the model parameters were estimated experimentally for CyberShip II, a 1:70 scale replica of a supply ship, in a marine control laboratory.

B. REINFORCEMENT LEARNING

In this section, we will briefly review the RL paradigm and introduce the specific technique that our method builds on. For a more comprehensive coverage, the reader is advised to consult the book by Sutton and Barto [32].

Fundamentally, RL is an approach to let autonomous agents learn how to behave optimally in their environments. Using the phrase "let learn" instead of "teach" is not accidental; a defining feature of RL is that the learning is not instructive, as opposed to the related field of supervised learning. Instead, learning is achieved through a combination of exploration and evaluative feedback, which bears a close resemblance to the way in which humans and other animals learn [32]; they become gradually wiser by virtue of trial and error.

1) FUNDAMENTALS OF RL

At each discrete time-step of the learning process, the **agent**, which is operating within an **environment**, chooses an **action** u based on its current **state** s (also often referred to as *observation*). The way in which the specific action was chosen by the agent (i.e. the agent's strategy) is commonly referred to as the **policy** and denoted by π. Thus, the policy π can be thought of as a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from the state space to the action space. In order to learn, i.e. improve the policy π, the agent then receives a numerical **reward** r from the environment. The fundamental goal of the agent is to maximize its long-term reward (also known as the **return**), and updates to the agent's policy are intended to improve the agent's ability to do this. These concepts (i.e. *agents*, *environments*, *observations/states*, *policies*, *actions* and *rewards*) are fundamental to the study of RL.

Remark: The reward may not solely depend on the latest action made. An intuitively attractive action may have long-term repercussions. Similarly, an action which is unexciting in the short-term may be optimal in the long term. Delayed rewards are common in RL environments.

Remark: The policy need not be deterministic. In fact, in games such as rock–paper–scissors, the optimal policy is stochastic.

Remark: The actions need not be discrete. Traditionally, RL algorithm have been dealing with discrete action spaces, but recent advances in the field have led to state-of-the-art algorithms that are naturally compatible with continuous action spaces (i.e. do not involve the workaround of discretizing a continuous action space, which is undesirable for control applications [33]).

As the environment may be stochastic, it is common to think of the process as a Markov decision process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r(s_t, a_t)$, transition dynamics $p(s_{t+1}|s_t, a_t)$ and an initial state distribution $p(s_0)$ [34]. The combined MDP and agent formulation allows us to sample trajectories from the process by first sampling an initial state from $p(s_0)$, and then repeatedly sampling the agent's action a_t from its policy $\pi(s_t)$ and the next state s_{t+1} from $p(s_{t+1}|s_t, a_t)$. As the agent is rewarded at each time step, its total reward can be represented as

$$R_t \triangleq \sum_{i=t}^{\infty} r(s_i, a_i) \tag{3}$$

Remark: Analogous to discount functions used in the field of economics, it is common to introduce a discount factor $\gamma \in (0, 1]$ *to capture the agent's relative preference for short-term rewards mathematically and to ensure that the infinite sum of rewards will not diverge. The discounted sum of rewards is then given by* $\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$. *For concreteness in the following derivations, however, the discount factor is disregarded. This is justified by considering the discount factor as being already incorporated into the reward function, making it time-dependent.*

Due to the stochasticity of the environment, one must consider the expected sum of rewards to obtain a tractable formulation for optimization purposes. Thus, we can introduce the state-value function $V^{\pi}(s)$ and the action-value function $Q^{\pi}(s, a)$, two very related concepts. $V^{\pi}(s)$ represents the expected return from time t onwards given an initial state s, whereas $Q^{\pi}(s, a)$ represents the expected return from time t onwards **conditioned on the initial action** a_t.

$$V^{\pi}(s_t) \triangleq \mathbb{E}_{s_{i>=t}, a_{i>=t} \sim \pi}\left[R_t | s_t\right] \tag{4}$$

$$Q^{\pi}(s_t, a_t) \triangleq \mathbb{E}_{s_{i>=t}, a_{i>=t} \sim \pi}\left[R_t | s_t, a_t\right] \tag{5}$$

2) POLICY GRADIENTS

Whereas value-based methods are concerned with estimating the state-value function and then inferring the optimal policy, policy-based methods directly optimize the policy. For high-dimensional or continuous action spaces, policy-based methods are commonly considered to be the more efficient approach [35].

From now on, we consider the policy $\pi(\theta)$ to be stochastic (i.e. $\pi(\theta) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$) and assume that is defined by some differentiable function parametrized by θ, enabling us to optimize it through policy-gradient methods. In general, these methods are concerned with using gradient ascent approximations to gradually adjust the policy function parameterization vector in order to optimize the performance objective

$$J(\theta) \triangleq \mathbb{E}_{s_i, a_i \sim \pi(\boldsymbol{\theta})}\left[R_0\right] \tag{6}$$

More formally, policy-gradient methods approach gradient ascent by updating the parameter vector θ according to the approximation $\theta_{t+1} \leftarrow \alpha\theta_t + \widehat{\nabla_\theta J(\theta)}$, where $\widehat{\nabla_\theta J(\theta)}$ is a stochastic estimate of $\nabla_\theta J(\theta)$ satisfying $\mathbb{E}\left[\widehat{\nabla_\theta J(\theta)}\right] = \nabla_\theta J(\theta)$. Intuitively, the estimation of the policy gradient might be considered intractible, as the state transition dynamics, which affect the expected reward and hence our performance objective, are influenced by the agent's policy in an unknown fashion. However, the policy gradient theorem [36] establishes that the policy gradient $\nabla_\theta J(\theta)$ satisfies

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a \nabla_\theta \pi(a|s) Q^{\pi}(s, a) \tag{7}$$

Here, μ is the steady state distribution under π, i.e. $\mu(s) = \lim_{t\to\infty} Pr\{S_t = s | A_{0:t-1} \sim \pi\}$, where S_t and $A_{0:t-1}$ are random variables representing the state at time-step t, and the actions up to that point, respectively. Interestingly, the expression for the policy gradient does not contain the derivative $\nabla_\theta \mu(s)$, implying that approximating the gradient by sampling is feasible, because calculating the effect of updating the policy on the steady state distribution is not needed. By replacing the probability-weighted sum over all possible states in Equation 7 by an expectation of the random variable S_t under the current policy, we have that

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi\left[\sum_a \nabla_\theta \pi(a|S_t) Q^{\pi}(S_t, a)\right] \tag{8}$$

Similarly, we can replace the sum over all possible actions with an expectation of the random variable A_t after multiplying and dividing by the policy $\pi(a|S_t)$:

$$\begin{aligned} \nabla_\theta J(\theta) &\propto \mathbb{E}_\pi\left[\sum_a \frac{\pi(a|S_t)}{\pi(a|S_t)} \nabla_\theta \pi(a|S_t) Q^{\pi}(S_t, a)\right] \\ \nabla_\theta J(\theta) &\propto \mathbb{E}_\pi\left[\frac{\nabla_\theta \pi(A_t|S_t)}{\pi(A_t|S_t)} Q^{\pi}(S_t, A_t)\right] \end{aligned} \tag{9}$$

Furthermore, it follows from the identity $\nabla \ln x = \frac{\nabla x}{x}$ that

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi\left[\nabla_\theta \ln \pi(A_t|S_t) Q^{\pi}(S_t, A_t)\right] \tag{10}$$

Also, by considering that

$$\begin{aligned} \sum_a b(s) \nabla \pi(a|s) &= b(s) \nabla \sum_a \pi(a|s) \\ &= b(s) \nabla 1 = 0 \end{aligned} \tag{11}$$

it is straight-forward to see that one can replace the state-action value function $Q^{\pi}(s, a)$ in Equation 7 by $Q^{\pi}(s, a) - b(s)$, where the **baseline** function $b(s)$ can be an arbitrary function independent of the action a, without introducing a bias in the estimate. However, it can be shown that the variance of the estimator can be greatly reduced by introducing such a baseline. It is possible to calculate the optimal (i.e. variance-minimizing) baseline [37], but commonly the state value function V^π is used, yielding an almost optimal variance [38]. The resulting term is known as the advantage function:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{12}$$

which intuitively represents the expected improvement obtained by an action compared to the default behavior. Furthermore, by following the same steps as outlined above, we end up with the expression

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi\left[\nabla_\theta \log \pi(A_t|S_t) A^\pi(s,a)\right] \tag{13}$$

Thus, an unbiased empirical estimate based on N episodic trajectories (i.e. independent rollouts of the policy in the environment) of the policy gradient is

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty} \hat{A}_t^n \nabla_\theta \log \pi(a_t^n|s_t^n) \tag{14}$$

3) ADVANTAGE FUNCTION ESTIMATION

As both $Q^\pi(s,a)$ and $V^\pi(s)$ are unknown in general, it follows that $A^\pi(s,a)$ is also unknown. Thus, it is commonly replaced by an advantage estimator $\hat{A}^\pi(s,a)$. Various estimation methods have been developed for this purpose, but a particularly popular one is Generalized Advantage Estimation (GAE) as originally outlined in [38], which uses discounted temporal difference (TD) residuals of the state value function as the fundamental building blocks. For this, we reintroduce the discount parameter γ. However, even if γ corresponds to the discount factor discussed in the context of MDPs, we now consider it as a variance-reducing parameter in an undiscounted MDP. TD residuals [32], which are in widespread use within RL, and give a basic estimate of the advantage function, are defined by

$$\delta_t^V = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) \tag{15}$$

where $\hat{V}$ is an approximate value function. Whenever $\hat{V} = V^\pi$, i.e. our approximation equals the real value function, the estimate is actually unbiased. For practical purposes, however, this is unlikely to be the case, so a common approach is to look further ahead than just one step in order to reduce the bias. More formally, by defining $\hat{A}_t^{(k)}$ as the discounted sum of the k next TD residuals, we have that

$$\begin{aligned}
\hat{A}_t^{(1)} &= \delta_t^{\hat{V}} = -\hat{V}(s_t) + r_t + \gamma \hat{V}(s_{t+1}) \\
\hat{A}_t^{(2)} &= \delta_t^{\hat{V}} + \gamma\delta_{t+1}^{\hat{V}} = -\hat{V}(s_t) + r_t + \gamma r_{t+1} + \gamma^2 \hat{V}(s_{t+2}) \\
&\vdots \\
\hat{A}_t^{(k)} &= \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^{\hat{V}}
\end{aligned} \tag{16}$$

The defining feature of GAE is that, instead of choosing some k-step estimator $\hat{A}_t^{(k)}$, we use an exponentially weighted average of the k first estimators, letting $k \to \infty$. Thus, we have that

$$\hat{A}_t^{GAE(\gamma,\lambda)} \triangleq (1-\lambda)(\hat{A}_t^1 + \lambda\hat{A}_t^2 + \lambda^2\hat{A}_t^3 + \ldots) \tag{17}$$

which can be shown by insertion of the definition of $\hat{A}_t^{(k)}$ to equal

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^{\hat{V}} \tag{18}$$

Here, $\lambda \in [0, 1]$ serves as a trade-off parameter controlling the compromise between bias and variance in the advantage estimate; using a small value lowers the variance as the immediate TD residuals make up most of the estimate, whereas using a large value lowers the bias induced by inaccuracies in the value function approximation.

Due to the recent advances made within deep learning (DL), a common approach is to use a deep neural network (DNN) for estimating the value function, which is trained on the discounted empirical returns. More specifically, the DNN state value estimator $\hat{V}_\theta(s_t)$, which is parametrized by θ_{VF}, is trained by minimizing the loss function

$$L_t^{VF}(\theta) = \hat{\mathbb{E}}_t\left[\hat{V}_\theta(s_t) - \sum_{i=t}^{\infty}\gamma^{i-t} r(s_i,a_i)\right] \tag{19}$$

where the expectation $\hat{\mathbb{E}}_t[\ldots]$ represents the empirical average obtained from a finite batch of samples. The reader is referred to [39] for a comprehensive introduction to DL, or to [40], which covers supervised machine learning, of which DL is a subfield.

4) A SURROGATE OBJECTIVE

Optimizing the performance objective directly using the empirical policy gradient approximation from Equation 14 is feasible; in fact, this constitutes the vanilla policy gradient algorithm originally proposed in [41]. However, it is well known that this approach has limitations due to a relatively low sample efficiency and thus suffers from a rather slow convergence time, as it requires an excessive number of samples for accurately estimating the policy gradient direction [42]. Accordingly, unless the step-size is chosen to be trivially small (yielding unacceptably slow convergence), it is not guaranteed that the policy update will improve the performance objective, which leads to the algorithm having poor stability and robustness characteristics [43].

Instead, recent state-of-the-art policy gradient methods such as Trust Region Policy Optimization (TRPO) [44] and its "successor" Proximal Policy Optimization [45] optimize a surrogate objective function which provides theoretical guarantees for policy improvement even under nontrivial step sizes. Fundamentally, these methods rely on the relative policy performance identity proven in [42], which states that the improvement in the performance objective $J(\theta)$ achieved by a policy update $\theta \to \theta'$ is equal to the expected advantage (ref. Equation 12) of the actions sampled from the new policy $\pi'_{\theta'}$ calculated with respect to the old policy π_θ. More formally, this translates to

$$J(\theta') - J(\theta) = \mathbb{E}_{\pi'_\theta}\left[\sum_t \gamma^t A^{\pi_\theta}(s_t, a_t)\right] \tag{20}$$

which is, albeit interesting, not practically useful as the expectation is defined under the next (i.e. unknown) policy $\pi_{\theta'}$, which we are obviously unable to sample trajectories from. However, Equation 20 can be rewritten and finally approximated by

$$\begin{aligned} &J(\theta') - J(\theta) \\ &\quad = \textstyle\sum_t \mathbb{E}_{s_t \sim \pi_{\theta'}}\left[\mathbb{E}_{a_t \sim \pi_{\theta'}}\left[\gamma^t A^{\pi_\theta}(s_t, a_t)\right]\right] \\ &\quad = \textstyle\sum_t \mathbb{E}_{s_t \sim \pi_{\theta'}}\left[\mathbb{E}_{a_t \sim \pi_\theta}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)\right]\right] \\ &\quad \approx \textstyle\sum_t \mathbb{E}_{s_t \sim \pi_\theta}\left[\mathbb{E}_{a_t \sim \pi_\theta}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)\right]\right] \end{aligned} \tag{21}$$

where the third and last steps can be seen as importance sampling and neglecting state distribution mismatch respectively. Loosely stated, the last approximation assumes that the change in the state distribution induced by a small update to the policy parameters is negligible. This is justified by theoretical guarantees imposing an upper bound to the distribution chance provided in [42]. This suggests that one can reliably optimize the *conservative policy iteration* surrogate objective

$$J^{CPI}(\theta') = \hat{\mathbb{E}}_t\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \hat{A}_t^{\pi_\theta}\right] \tag{22}$$

[42]. However, this approximation is only valid in a local neighborhood, requiring a carefully chosen step-size to avoid instability. In TRPO, this is achieved by maximizing $L^{CPI}(\theta')$ under a hard constraint on the KL divergence between the old and the new policy. However, as this is computationally expensive, the PPO algorithm refines this by integrating the constraint into the objective function by redefining the objective function to

$$\begin{aligned} J^{CLIP}(\theta') &= \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t^{\pi_\theta}, \operatorname{clip}_\epsilon(r_t(\theta))\hat{A}_t^{\pi_\theta}\right)\right] \\ \operatorname{clip}_\epsilon(x) &= \operatorname{clip}(x, 1-\epsilon, 1+\epsilon) \end{aligned} \tag{23}$$

where $r_t(\theta)$ is a shorthand notation for the probability ratio $\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$. The truncation of the probability ratio is motivated by a need to restrict $r_t(\theta)$ from moving outside of the interval $[1-\epsilon, 1+\epsilon]$. Also, the expectation is taken over the minimum of the clipped and unclipped objective, implying that the overall objective function is a lower bound of the original objective function $J^{CPI}(\theta')$. At each training iteration, the advantage estimates are computed over batches of trajectories collected from N_A concurrent actors, each of which executes the current policy π_θ for T timesteps. Afterwards, a stochastic gradient descent (SGD) update using the *Adam* optimizer [46] of minibatch size N_{MB} is performed for N_E epochs.

The PPO algorithms strikes a balance between ease of implementation and data efficiency, and is likely to perform well in a wide range of continuous environments without extensive hyperparameter tuning [45]. Sensitivity to hyperparameter choices is a frequently encountered problem for policy gradient methods [47], [48], and given the computation time required to train and test agents in a collision avoidance environment, this could be a detrimental bottleneck in our research.

Algorithm 1 Proximal Policy Optimisation

```
for iteration = 1, 2, ... do
    for actor = 1, 2, ... $N$ do
        For $T$ time-steps, execute policy $\pi_\theta$.
        Compute advantage estimates $\hat{A}_1, \ldots \hat{A}_T$
    for epoch = 1, 2, ... $N_E$ do
        Obtain mini batch of $N_{MB}$ samples from the $N_A T$ simulated time-steps.
        Perform SGD update from minibatch $(\mathbf{X}_{MB}, \mathbf{Y}_{MB})$.
        $\theta \leftarrow \theta'$
```

C. TOOLS AND LIBRARIES

The code implementation of our solution make use of the RL framework provided by the Python library **OpenAI Gym** [49], which was created for the purpose of standardizing the benchmarks used in RL research. It provides a easy-to-use framework for creating RL environments in which custom RL agents can be deployed and trained with minimal overhead.

Stable Baselines [50], another Python package, provides a large set of state-of-the-art parallelizable RL algorithms compatible with the OpenAI gym framework, including PPO. The algorithms are based on the original versions found in OpenAI Baselines [51], but Stable Baselines provides several improvements, including algorithm standardization and exhaustive documentation.

III. METHODOLOGY

In this section, we outline the specifics of our approach by defining the fundamental RL concepts as presented in Section II-B.1 according to the problem at hand and describe how the vessel's guidance capabilities are trained within the context of the RL framework Stable Baselines.

A. ENVIRONMENT

The environment in which we except the agent to perform is an ocean surface filled with obstacles, also containing an a priori known path that the agent is intended to follow while avoiding collisions. The vessel dynamics (ref. Section II-A.3) should, in fact, also be considered as a part of the environment, as it is outside of the agent's control. It is also critical that the environments in which the agent is trained pose a wide variety of challenges to the agent, so that the trained agent is able to generalize to unseen obstacle landscapes, potentially following a deployment on a vessel in the real world. Thus, we need a stochastic algorithm for generating training environments. If the environments are too easy or monotone (or a combination thereof), the agent will overfit to the training environments leading to undesired behavior when testing it in unseen, more complicated obstacle landscapes. For instance, if all obstacles are located very close to the path within the training environments, the trained agent may exhibit undesired behavior by always going around obstacles to avoid them, whereas an intelligent agent would simply ignore obstacles that are not in its way in order to stay on track. Also, if the obstacle density is too low, it is unlikely that the agent would perform well in a high-obstacle-density environment. To this end we suggest the procedure outlined in Algorithm 2 for generating new, independent training environments. Some randomly sampled environments generated from this algorithm can be seen in Figure 2. It is obvious that performing well within these environments (i.e. adhering to the planned path while avoiding collisions) necessitates a nontrivial guidance algorithm.

Algorithm 2 Generate Path With Obstacles

```
Require:
  Number of obstacles $N_o \in \mathbb{N}_0$
  Number of path waypoints $N_w \in \mathbb{N}_0$
  Path length $L_p \in \mathbb{N}_0$
  Mean obstacle radius $\mu_r \in \mathbb{R}^+$
  Obstacle displacement distance standard deviation $\sigma_d \in \mathbb{R}^+$
  procedure GeneratePathColavEnvironment($N_o$, $N_w$, $L_p$, $\mu_r, \sigma_d$)
      Draw $\theta_{start}$ from $Uniform(0, 2\pi)$
      Path origin $\boldsymbol{p}_{start} \leftarrow 0.5\, L_p\, [\cos(\theta_{start}), \sin(\theta_{start})]^T$
      Goal position $\boldsymbol{p}_{end} \leftarrow -\boldsymbol{p}_{start}$
      Generate $N_w$ random waypoints between $\boldsymbol{p}_{start}$ and $\boldsymbol{p}_{end}$.
      Create smooth arc length parameterized path $\boldsymbol{p}_p(\bar{\omega}) = [x_p(\bar{\omega}), y_p(\bar{\omega})]^T$ using 1D Piecewise Cubic Hermite Interpolator (PCHIP) provided by Python library SciPy [52].
      repeat
          Draw arclength $\bar{\omega}_{obst}$ from $Uniform(0.1\, L_p, 0.9\, L_p)$.
          Draw obstacle displacement distance $d_{obst}$ from $\mathcal{N}(0, \sigma_d^2)$
          Path angle $\gamma_{obst} \leftarrow \text{atan2}\,(\boldsymbol{p}_p{}'(\bar{\omega}_{obst})_2, \boldsymbol{p}_p{}'(\bar{\omega}_{obst})_1)$
          Obstacle position $\boldsymbol{p}_{obst} \leftarrow \boldsymbol{p}_p(\bar{\omega}_{obst}) + d_{obst}[\cos(\gamma_{obst} - \frac{\pi}{2}), \sin(\gamma_{obst} - \frac{\pi}{2})]^T$
          Draw obstacle radius $r_{obst}$ from $Poisson(\mu_r)$.
          Add obstacle $(\boldsymbol{p}_{obst}, r_{obst})$ to environment
      until $N_0$ obstacles are created
```

In the current work the values of $N_o = 20$, $N_w = \mathcal{U}(2, 5)$, $L_p = 400$, $\mu_r = 30$, $\sigma_d = 150$ (where $\mathcal{U}$ is the uniform distribution) were used.

B. AGENT

Although the *agent*, within the context of RL, can be considered to be the vessel itself, it is more accurate to look at it as the guidance mechanism controlling the vessel, as its operation is limited to outputting the control signals that steer the vessel's actuators. As discussed in Section II-A.4, the available control signals are the surge thrust T_u, driving the vessel forward, and the yaw moment T_r, inducing a change in the vessel's heading. The RL agent's action, which it will output at each simulated time-step, is then defined as the vector $a = [T_u, T_r]^T$. Specifically, the action network, which we train by applying the PPO algorithm described in Section II-B.4, will output the control signals following a forward pass of the current observation vector through the nodes of the neural network. Also, the value network is trained simultaneously, facilitating estimation of the state value function $V(s)$ which is used for GAE as described in Section II-B.3. Deciding what constitutes a state s is of utmost importance; the information provided to the agent must be of sufficient fidelity for it to make rational guidance decisions, especially as the agent will be purely reactive, i.e. not able to let previous observations influence the current action. At the same time, by including too many features in the state definition, we risk overparameterization within the neural networks, which can lead to poor performance and excessive training time requirements [39]. Thus, a compromise must be reached, ensuring a sufficiently low-dimensional observation vector while still providing a sufficiently rich observation of the current environment. Having separate observation features representing path following performance and obstacle closeness is a natural choice.

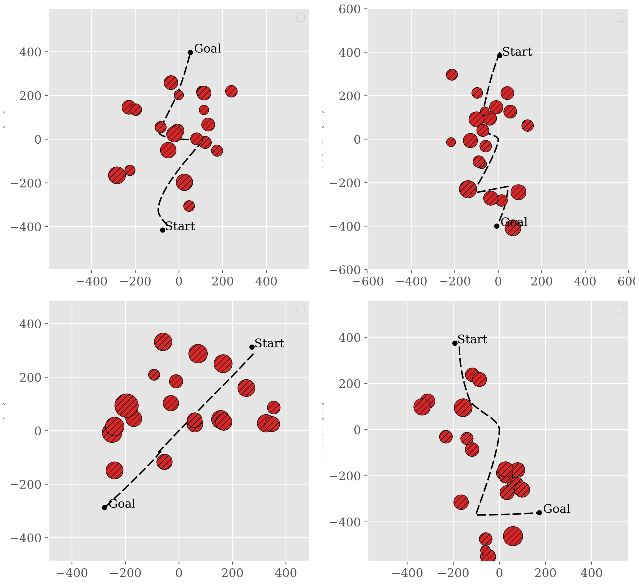


FIGURE 2. Four random samples of the stochastically generated path following scenario. Note that the scenario difficulty is highly varying.

1) PATH FOLLOWING

The agent needs to know how the vessel's current position and orientation aligns with the desired path. A few concepts often used for guidance purposes are useful in order to formalize this. First, we formally define the desired path as the one-dimensional manifold given by

$$\mathcal{P} \triangleq \left\{ \boldsymbol{p} \in \mathbb{R}^2 \mid \boldsymbol{p} = \boldsymbol{p}_p(\bar{\omega})\ \forall\ \bar{\omega} \in \mathbb{R}^+ \right\} \tag{24}$$

Accordingly, for any given $\bar{\omega}$, we can define a local path reference frame $\{p\}$ centered at $\boldsymbol{p}_p(\bar{\omega})$ whose x-axis has been rotated by the angle

$$\gamma_p(\bar{\omega}) \triangleq \operatorname{atan2}\left(y_p'(\bar{\omega}), x_p'(\bar{\omega})\right) \tag{25}$$

relative to the inertial NED-frame. Next, we consider the so-called look-ahead point $\boldsymbol{p}_p(\bar{\omega} + \Delta_{LA})$, where $\Delta_{LA} > 0$ is the look-ahead distance. In traditional path-following, look-ahead based steering, i.e. setting the look-ahead point direction as the desired course angle, is a commonly used guidance principle [53]. Based on the look-ahead point, we define the *course* error, i.e. the course change needed for the vessel to navigate straight towards the look-ahead point, as

$$\tilde{\chi}(t) \triangleq \operatorname{atan2}\left(\frac{y_p(\bar{\omega} + \Delta_{LA}) - y_p(\bar{\omega})}{x_p(\bar{\omega} + \Delta_{LA}) - x_p(\bar{\omega})}\right) - \chi(t) \tag{26}$$

where $\chi(t)$ is the vessel's current heading as defined in Section II-A.2. Furthermore, (as in [54]) given the current vessel position $\boldsymbol{p}(t)$ we can define the error vector $\boldsymbol{\epsilon}(t) \triangleq [s(t), e(t)]^T \in \mathbb{R}^2$, containing the *along-track* error $s(t)$ and the *cross-track* error $e(t)$ at time t, as

$$\boldsymbol{\epsilon}(t) = \mathbf{R}_{z,-\gamma_p(\bar{\omega})}(\boldsymbol{p}(t) - \boldsymbol{p}_p(\bar{\omega})) \tag{27}$$

A natural approach for updating the path variable $\bar{\omega}$ is to repeatedly calculate the value that yields the closest distance between the path and the vessel using Newton's method. Here, the fact that Newton's method only guarantees a local optimum is a useful feature, as it prevents sudden path variable jumps given that the previous path variable value is used as the initial guess [55]. Another approach is to update the path variable according to the differential equation

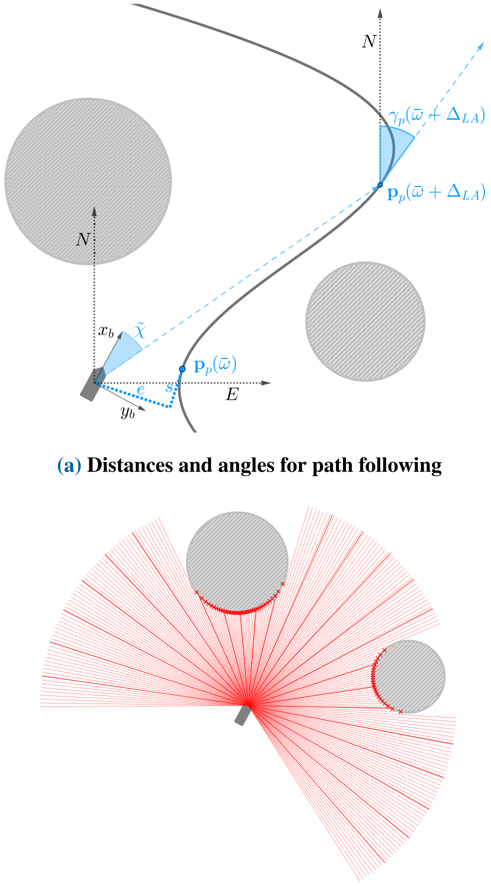
$$\dot{\bar{\omega}} = \sqrt{u^2 + v^2}\cos\tilde{\chi}(t) - \gamma_{\bar{\omega}} s(t) \tag{28}$$

where the along-track error coefficient $\gamma_{\bar{\omega}} > 0$ ensures that the absolute along-track error $|s(t)|$ will decrease. As this method is computationally faster, we chose to use it in our Python implementation. More specifically, in the current work $\gamma_{\bar{\omega}} = 0.05$ and $\Delta_{LA} = 100m$.

2) OBSTACLE DETECTION

Using rangefinder sensors as the basis for obstacle avoidance is a natural choice, as a reactive navigation system applied to a real-world vessel would typically use such a solution or a camera-based one. This realistic approach should enable a relatively straightforward transition from the simulated environment to a real one, given the availability of common rangefinder sensors such as lidar, radar or sonar.

In the setup used, $N = 225$ sensors with a total visual span of $S_s = \frac{4\pi}{3}$ radians (240 degrees) are arranged as illustrated in Figure 3b. The sensors are assumed to have a range of $S_r = 150$ meters, which was deemed sufficient given the relatively small size of the vessel. Obviously, with regards to the number of sensors, one must consider the trade-off between computation speed and sensor resolution. In the experiments conducted in this research project, 225 sensors were chosen, even if it is likely that a much lower number of sensors would yield similar performance. With regards to the visual span, it could be argued that providing 180 degree vision would be sufficient to achieve satisfactory collision avoidance, given the precondition of static obstacles. However, in the interest of avoiding sub-optimal performance due to a restrictive sensor suite configuration, the conservative choice of having 240 degree vision was made.


N
$\gamma_p(\bar{\omega} + \Delta_{LA})$
$\mathbf{p}_p(\bar{\omega} + \Delta_{LA})$
N
x_b
$\tilde{\chi}$
$\mathbf{p}_p(\bar{\omega})$
e
s
E
y_b

(a) Distances and angles for path following

(b) $N = 225$ rangefinder sensors partitioned into $d = 25$ sectors

FIGURE 3. Illustrations showing the parameters for path following and collision avoidance. (a) shows the cross-track error e, along-track error s, heading error $\tilde{\chi}$, path reference point $p_p(\bar{\omega})$, look-ahead point $p_p(\bar{\omega} + \Delta_{LA})$ and look-ahead path tangential angle $\gamma_p(\bar{\omega} + \Delta_{LA})$. In (b), the sensors are arranged in sectors, where the sensor measurements are pooled into a scalar values.

Even if, in theory, a sufficiently large neural network is capable of representing any function with any degree of accuracy, including satisfactory mappings from sensor readings to collision-avoiding steering maneuvers in our case, there are no guarantees for either the feasibility of the required network size or the convergence of the optimization algorithm used for training the network [39]. Thus, forcing the action network to output the control signal based on 225 sensor readings (as well as the features intended for path-following) is unlikely to be a viable approach, given the complexity required for any satisfactory mapping between the full sensor suite to the steering signal. Instead, we propose three approaches for transforming the sensor readings into a reduced observation space from which a satisfactory policy mapping should be easier to achieve. As illustrated in Figure 3b, this involves partitioning the sensor suite into d disjoint sensor sets, hereafter referred to as *sectors*. First, we define the sensor *density* n as the number of sensors contained by one sector: $n \triangleq \frac{N}{d}$

Each sector is made up of neighboring sensors, so we can formally define the k^{th} sector, which we denote by $\mathcal{S}_k$, as

$$\mathcal{S}_k \triangleq \left\{x_{(k-1)\,n+1}, \ldots, x_{kn}\right\} \tag{29}$$

where x_i refers to the i^{th} sensor measurement according to a counter-clockwise indexing direction. This partitioning, which assumes that N is a multiple of d, is illustrated in Figure 3b.

Based on the concept of partitioning the sensor suites into sectors, we then seek to reduce the dimensionality of our observation vector. Instead of including each individual sensor measurement x_i in it, we provide a single scalar feature for each sector $\mathcal{S}_k$, effectively summarizing the local sensor readings within the sector. The resulting dimensionality reduction is quite significant; instead of having N sensor measurements in the observation vector, we now have only d features. What remains is the exact computation procedure by which a single scalar is outputted based on the current sensor readings within each sector. Always returning the minimum sensor reading within the sector, in the following referred to as *min pooling*, i.e. outputting the shortest measured obstacle distance within the sector, is a natural approach which yields a conservative and thereby safe observation vector. As can be seen in Figure 4, however, this approach might be overly restrictive in certain obstacle scenarios, where feasible passings in between obstacles are inappropriately overlooked. However, even if the opposite approach (*max pooling*) solves this problem, it is straightforward to see, e.g. in Figure 4b by considering the fact that the presence of a small, nearby obstacle in the leftmost sector is ignored, that it might lead to dangerous navigation strategies.

To alleviate the problems associated with min and max pooling mentioned above a new approach is required. A natural approach is to compute the maximum feasible travel distance within the sector, taking into account the location of the obstacle sensor readings as well as the width of the vessel. This requires us to iterate over the sensor readings in ascending order corresponding to the distance measurements, and for each resulting distance level check whether it is feasible for the vessel to advance beyond this level. As soon as the widest opening available within a distance level is deemed too narrow given the width of the vessel, the maximum feasible distance has been reached. A pseudocode implementation of this algorithm is provided as Algorithm 3.

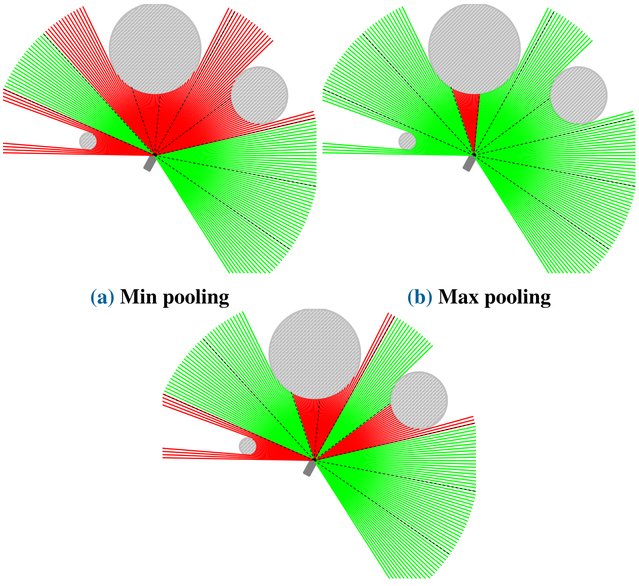

(a) Min pooling
(b) Max pooling
(c) Feasibility pooling

FIGURE 4. Pooling techniques for sensor dimensionality reduction. For the sectors colored green, the maximum distance S_r was outputted. It is obvious that min-pooling yields an overly restrictive observation vector, effectively telling the agent that a majority of the travel directions are blocked. On the other hand, max pooling yields overly optimistic estimates, potentially leading to dangerous situations.

TABLE 1. Sensor configuration.

Parameter	Description	Value
U_{max}	Maximum vessel speed	2 m/s
W	Vessel width	4 m
N	Number of sensors	225
S_s	Total visual span of sensors	240°
S_r	Maximum rangefinder distance	150 m
d	Number of sensor sectors	25

Having a runtime complexity of $\mathcal{O}(dn^2)$ when executed on the entire sensor suite, the feasibility pooling approach is slower than simple max or min pooling, both having the runtime complexity $\mathcal{O}(dn)$. However, in the simulated environment, the increased computation time, which is reported through empirical estimates in Figure 5 for $n = 9$, is negligible compared to the time needed to compute the interception points between the rangefinder rays and the obstacles.

Another interesting aspect to consider when comparing the pooling methods, is the sensitivity to sensor noise. A compelling metric for this is the degree to which the pooling output differs from the original noise-free output when normally distributed noise with standard deviation σ_w is applied to the sensors. Specifically, we report the root mean square of the differences between the original pooling outputs and the outputs obtained from the noise-affected measurements. The results for $\sigma_w \in \{1, \ldots, 30\}$ are presented in Figure 5b. Evidently, the proposed feasibility method for pooling is slightly more robust than the other variants.

Algorithm 3 Feasibility Pooling for Rangefinder

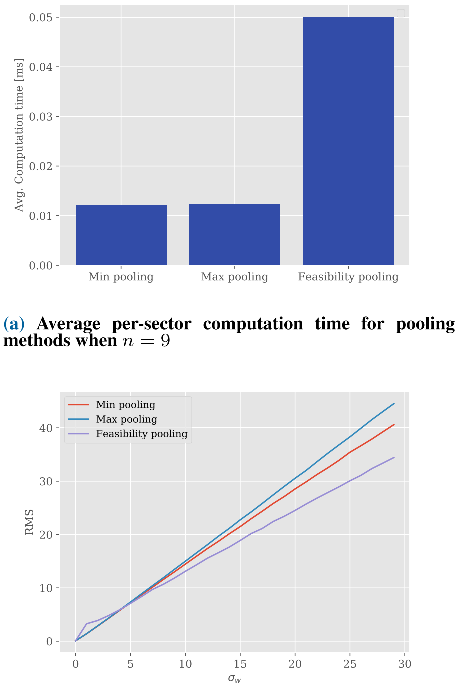
```
Require:
  Vessel width $W \in \mathbb{R}^+$
  Total number of sensors $N \in \mathbb{N}$
  Total sensor span $S_s \in [0, 2\pi]$
  Sensor rangefinder measurements for current sector $\boldsymbol{x} = \{x_1, \ldots, x_n\}$
  procedure FeasibilityPooling($\boldsymbol{x}$)
      Angle between neighboring sensors $\theta \leftarrow \frac{S_s}{N-1}$
      Initialize $\mathcal{I}$ to be the indices of $\boldsymbol{x}$ sorted in ascending order according to the measurements $x_i$
      for $i \in \mathcal{I}$ do
          Arc-length $d_i \leftarrow \theta x_i$
          Opening-width $y \leftarrow d_i/2$
          Opening was found $s_i \leftarrow false$
          for $j \leftarrow 0$ to $n$ do
              if $x_j > x_i$ then
                  $y \leftarrow y + d_i$
                  if $y > W$ then
                      $s_i \leftarrow true$
                      break
              else
                  $y \leftarrow y + d_i/2$
                  if $y > W$ then
                      $s_i \leftarrow true$
                      break
                  $y \leftarrow 0$
          if $s_i$ is false then return $x_i$
```

C. REWARDS

Any RL agent is motivated by the pursuit of maximum reward. Ideally, the agent should receive its reward at the end of the episode, after having either reached the goal position or collided. However, such a reward function is extremely sparse, leaving the agent with a near impossible learning task. This demonstrates the need of a continuous reward signal, guiding the agent to better performance. Given the complexity of the dual-objective task, as well as RL agents' tendency to misuse the reward function in any way possible, we had to design an appropriate reward function $r(t)$. This was paramount to the agent exhibiting the desired behavior after training. Given the dual nature of our objective, which is to follow the path while avoiding obstacles along the way, it is natural to reward the agent separately for its performance in these two domains.

Thus, we introduce the reward terms $r_{pf}(t)$ and $r_{oa}(t)$, being the reward components at time t representing the path-following and the obstacle-avoiding performance, respectively. Also, we introduce the weighting coefficient $\lambda \in [0, 1]$ to regulate the trade-off between the two competing objectives, leading to the preliminary reward function

$$r(t) = \lambda r_{pf}(t) + (1 - \lambda) r_{oa}(t) \tag{30}$$



(a) Average per-sector computation time for pooling methods when $n = 9$

(b) Robustness metric for pooling methods for $\sigma_w \in \{1, \ldots, 30\}$

FIGURE 5. Computational time and robustness of the different pooling approaches. The noise-affected measurements were clipped at zero to avoid negative values.

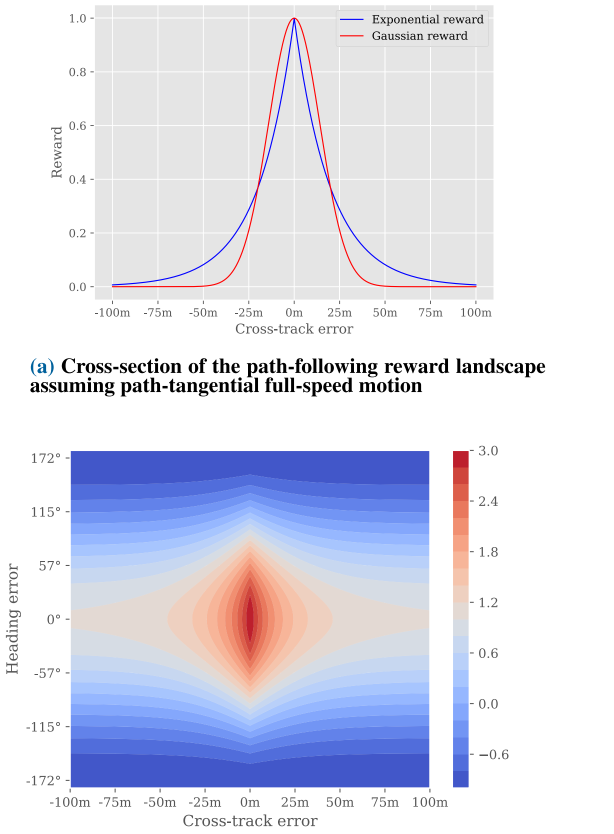
1) PATH FOLLOWING PERFORMANCE

A reasonable approach to incentivize adherence to the desired path is to reward the agent for minimizing the absolute cross-track error $e(t)$. In [55], a Gaussian reward function centered at $e(t) = 0$ with some reasonable standard deviation σ_e is used for this purpose. However, based on Figure 6a, we argue that the exponential $e^{-\gamma_e |y_e(t)|}$ has slightly more reasonable characteristics for this purpose due to its fatter tails, thus rewarding the agent for a slight improvement to an unsatisfactory location.

However, this alone does not reflect our desire for the agent to actually make progress along the path. This can be achieved by multiplying by the velocity component in the desired course direction given by $\sqrt{u^2 + v^2} \cos \tilde{\chi}(t)$, yielding negative rewards if the agent is tracking backwards, and zero reward if it is vessel course in a direction perpendicular to the path. Finally, we note that, if the agent is standing still, or if the course error is ±90°, it will receive zero reward regardless of the cross-track error, which is not desired. Similarly, when the cross-track error grows large, it receive zero reward regardless of the speed or course error. Thus, we add constant multiplier terms 1 and end up with the path-following reward function

$$r_{pf}(t) = -1 + \left(\frac{\sqrt{u^2+v^2}}{U_{max}} \cos \tilde{\chi}(t) + 1 \right) \left(e^{-\gamma_e |y_e(t)|} + 1 \right) \tag{31}$$

where U_{max} is the maximum vessel speed.



(a) Cross-section of the path-following reward landscape assuming path-tangential full-speed motion

(b) Path-following reward function assuming full-speed motion

FIGURE 6. Cross-section and level curves for the path-following reward function for $\gamma_e = 0.05$.

Remark: Note that, for added flexibility, it is possible to replace the 1 *multipliers by some customizable coefficients. However, for the sake of parametric simplicity, we decide to use* 1.

2) OBSTACLE AVOIDANCE PERFORMANCE

In order to encourage obstacle-avoiding behavior, penalizing the agent for the *closeness* of nearby obstacles in a strictly increasing manner seems natural. Having access to the sensor measurements outlined in Section III-B.2 at each timestep, we use these as surrogates for obstacle distances through which the agent is penalized. By noting that the severity of obstacle closeness intuitively does not increase linearly with distance, but instead increases in some more or less exponential manner, and that the severity of obstacle closeness depends on the orientation of the vessel with regards to the obstacle in such a manner that obstacles located behind the vessel are of much lower importance than obstacles that are right in front of the vessel, is it easy to see that the term $(1 + |\gamma_\theta \theta_i|)^{-1}(\gamma_x \max(x_i, \epsilon_x)^2)^{-1}$, where θ_i is the vessel-relative angle of sensor i such that a forward-pointing sensor has angle 0, exhibits the desirable properties for penalizing the vessel based on the i^{th} sensor reading. This reward function is plotted in Figure 7.

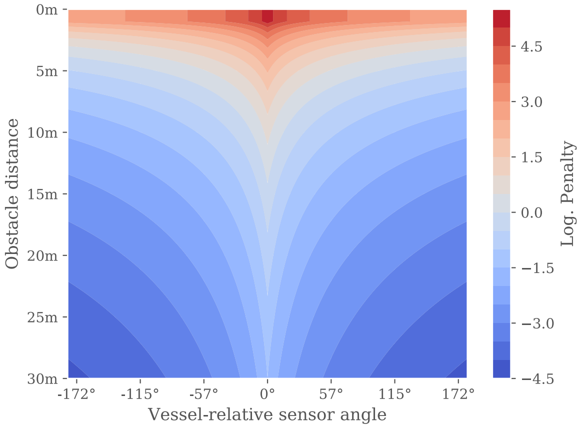


FIGURE 7. Obstacle closeness penalty as a function of vessel-relative sensor angle and obstacle distance, imposing a maximum penalty for obstacles located right in front of the vessel.

In order to to cancel the dependency on the specific sensor suite configuration, i.e. the number of sensors and their vessel-relative angles, that arises when this penalty term is summed over all sensors, we use a weighted average to define our obstacle-avoidance reward function such that

$$r_{oa}(t) = -\frac{\sum_{i=1}^{N}(1 + |\gamma_\theta \theta_i|)^{-1}(\gamma_x \max(x_i, \epsilon_x)^2)^{-1}}{\sum_{i=1}^{N}(1 + |\gamma_\theta \theta_i|)^{-1}} \tag{32}$$

where $\epsilon_x > 0$ is a small constant removing the singularity at $x_i = 0$.

3) TOTAL REWARD

In order to discourage the agent from simply standing still at a safe location, which would yield a reward of zero given the preliminary reward function, we impose a constant living penalty $r_{exists} < 0$ to the overall reward function. A simple way of setting this parameter is to assume that, given a total absence of nearby obstacles and perfect vessel alignment with the path, the agent should receive a zero reward when moving at a lower than speed $\alpha_r U_{max}$, where $\alpha_r \in (0, 1)$ is a constant parameter. This gives us

$$\begin{aligned} r_{exists} + \lambda\left(\left(\frac{\alpha_r U_{max}}{U_{max}} + 1\right)(1+1) - 1\right) &= 0 \\ r_{exists} &= -\lambda(2\alpha_r + 1) \end{aligned} \tag{33}$$

Also, in the interest of having bounded rewards, we enforce a lower bound activated upon collisions by defining the total reward

$$r(t) = \begin{cases} (1-\lambda)\, r_{collision} & \text{(if collision)} \\ \lambda r_{pf}(t) + (1-\lambda)\, r_{oa}(t) + r_{exists} & \text{(otherwise)} \end{cases} \tag{34}$$

Deciding the optimal value for the trade-off parameter λ is a nontrivial endeavour. This touches upon the fundamental challenge tackled in this project, namely how to avoid obstacles while without deviating unnecessarily from the desired trajectory. Thus, we initialize it randomly at each reset of the

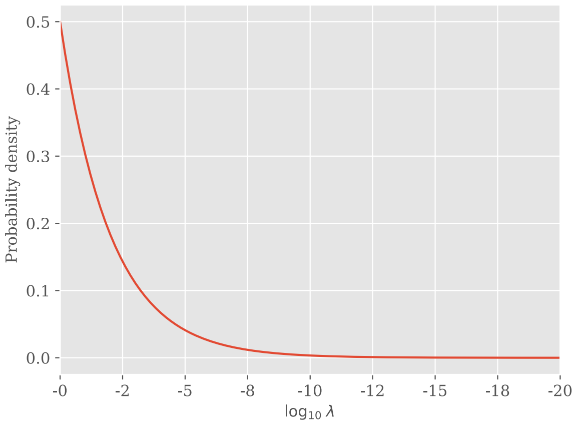


FIGURE 8. Gamma-distribution with parameters $\alpha_\lambda = 1$, $\beta_\lambda = 2$ from which $-\log_{10}\lambda$ is drawn.

TABLE 2. Observation vector *s* at timestep *t*.

Observation feature	Definition
Surge velocity	$u^{(t)}$
Sway velocity	$v^{(t)}$
Yaw rate	$r^{(t)}$
Look-ahead course error	$\gamma_p(\bar{\omega}^{(t)} + \Delta_{LA}) - \chi^{(t)}$
Course error	$\tilde{\chi}^{(t)}$
Cross-track error	$e^{(t)}$
Reward trade-off parameter	$\log_{10}\lambda^{(t)}$
Obstacle closeness, first sector	$1 - \frac{1}{S_r}\text{FeasibilityPooling}(\boldsymbol{x} = \{x_1, \dots x_d\})$
$\vdots$	
Obstacle closeness, last sector	$1 - \frac{1}{S_r}\text{FeasibilityPooling}(\boldsymbol{x} = \{x_{N-d}, \dots x_N\})$

environment by sampling it from a probability distribution. In order to familiarize the agent with different degrees of radical collision avoidance strategies ($\lambda \to 0$), which is useful in dead-end scenarios where the correct behavior is to ignore the desire for path adherence in order to escape the situation, we sample $\log_{10}\lambda$ from a gamma distribution such that

$$-\log_{10}\lambda \sim Gamma(\alpha_\lambda, \beta_\lambda) \tag{35}$$

In order to let the agent base its guidance strategy on the current λ, we include $\log_{10}\lambda$ as an additional observation feature. The reward parameters used in the current work is given by $\alpha_\lambda = 1.0$, $\beta_\lambda = 2.0$, $\gamma_e = 0.05$, $\gamma_\theta = 4.0$, $\gamma_x = 0.005$, $\epsilon_x = 1.0m$, $\alpha_r = 0.1$, $r_{collision} = -2000$.

The complete observation vector, which in the context of RL represents the state s, contains features representing the position and orientation of the vessel with regards to the path as well as the pooled sensor readings and the logarithm of the current trade-off parameter λ.

D. TRAINING

The RL agent is trained using the PPO algorithm (ref. Algorithm 1) implemented in the Python library *Stable Baselines* [50], with the hyperparameters given by $\gamma = 0.999$, $T = 1024$, $N_A = 8$, $K = 10^6$, $\eta = 0.0002$, $N_{MB} = 32$, $\lambda = 0.95$, $c_1 = 0.5$, $c_2 = 0.01$, $\epsilon = 0.2$. The action and value function networks were implemented as fully-connected neural networks, both using the tanh(.) activation function and consisting of with two hidden layers with 64 nodes. We simulate the vessel dynamics using the fifth order Runge-Kutta-Fahlberg method [56] using the timestep $\Delta t = 0.1s$. Whenever the vessel either reaches the goal $\boldsymbol{p}_{end}$, collides with an obstacle or reaches a cumulative negative reward exceeding -5000, the environment is reset according to Algorithm 2.

TABLE 3. List of reward trade-off test values.

Agent index	λ
1	1.0
2	0.9
3	0.5
4	0.1
5	0.01
6	0.001
7	0.0001
8	0.00001
9	0.000001

E. EVALUATION

We analyze the agent's performance based on quantitative as well as qualitative testing. Evaluating how the value of the reward trade-off parameter λ, which is fed to the agent as an observation feature, influences the guidance behavior is of particular interest. Specifically, we test the agent with the values listed in Table 3, including both radical path adherence (i.e. $\lambda = 1$) as well as various shades of radical obstacle avoidance strategies (i.e. $\lambda \to 0$).

1) QUANTITATIVE TESTING

In order to obtain statistically significant evidence for the guidance ability of the trained agent, we simulate the agent's behavior in 100 random environments generated stochastically according to Algorithm 2. We then report the performance criteria in terms of success rate, average cross-track error and average episode length. In the current context, the success rate is defined as the percentage of episodes in which the agent reached the goal, average cross-track error is defined as the average deviation from path in meters, average episode length is the average length of episode in seconds.

F. QUALITATIVE TESTING

In addition to the statistical evaluation, we observe the agents' behavior in the test scenarios shown in Figure 9.

G. COMPARISON WITH ALTERNATIVE RL ALGORITHMS

In order to assess the performance of the PPO algorithm on this guidance problem, we train the agent using several other frequently cited model-free policy gradient algorithms, a class of RL algorithms known for excelling at continuous control tasks [48]. Deep Deterministic Policy Gradient (DDPG) [33], Actor Critic using Kronecker-Factored Trust

TABLE 4. Quantitative test results obtained from 100 episode simulations per agent.

Agent	λ	Success Rate	Avg. Cross-track Error	Avg. Episode Length
1	1	97%	34.92 m	1001 s
2	0.9	97%	36.56 m	1028 s
3	0.5	99%	38.15 m	1024 s
4	0.1	100%	49.13 m	1077 s
5	0.01	100%	63.95 m	1062 s
6	0.001	100%	68.36 m	1238 s
7	0.0001	100%	72.99 m	1480 s
8	0.00001	100%	70.40 m	1469 s
9	0.000001	100%	70.51 m	1212 s

Region (ACKTR) [57] and Asynchronous Advantage Actor Critic (A3C) [58] are all available in the **Stable Baselines** library, and their quantitative test results will be included as benchmarks for the performance of the PPO agent.

IV. RESULTS AND DISCUSSIONS

In this chapter, we present the test results obtained from training and testing the agent and discuss the findings.

A. TRAINING PROCESS

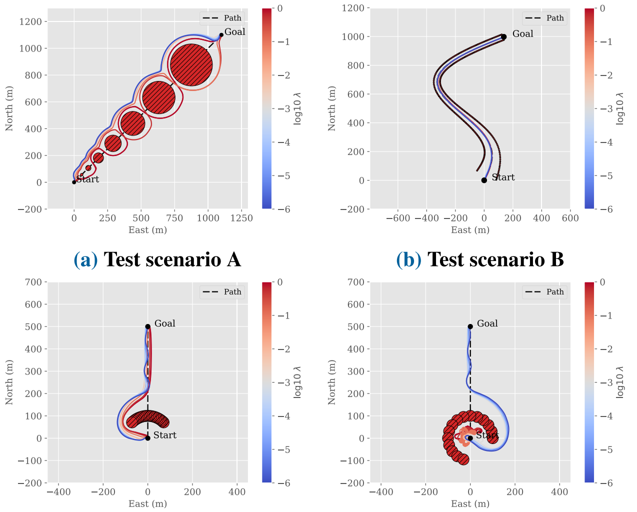
We train the agent for 3903 episodes, corresponding to more than 5 million simulated time-steps of length $\Delta t = 0.1\ s$. At this point, all the metrics used for monitoring the training progress had stabilized. The training process, which, for the purpose of faster convergence, ran 8 parallel simulation environments, took approximately 48 hours on a Intel Core i7-8550U CPU.

B. TEST RESULTS

As outlined, each value of λ was tested for 100 episodes, all of which took place in a randomly generated path following environments according to Algorithm 2. Of course, a larger sample size is always better for quantitative evaluation, but in the interest of time, 100 test episodes for each λ value was a reasonable compromise. Clearly, the calculation of the interception points between the rangefinder rays and the obstacles is the most computationally expensive part of the simulation. Thus, the simulation can be made orders of magnitude faster by lowering the sampling rate of the sensors, but we decided to perform the testing without any restrictions to the sensor suite. The observed test results are displayed in Table 4.

Additionally, we simulated each agent in the four outlined qualitative test scenarios. Except for scenario B, in which all agents chose more or less exactly the same trajectory, the other scenarios clearly reflect the differences between the agents. The agents' trajectories in each test scenario are plotted in Figure 9.

The PPO agent was clearly superior to the other RL algorithms that were tested, which, despite unquestionably exhibiting different kinds of behavior, all must be classified as failures when applied to this task. The trained A3C agent is the least competent one, mindlessly guiding the vessel in an arbitrary direction until it collides. The ACKTR agent appears to master the path following task, but frequently collides. The DDPG agent rarely collides, but does not follow the path and often ends up going in circles. A comparison of all four algorithms is provided in Figure 10, where the trained agents are simulated in a randomly generated scenario. This illustrates the superior performance exhibited by the PPO agent. It should be noted, however, that only the default set of hyper-parameters found in the **Stable Baselines** package were tested for the other RL algorithms.



(a) Test scenario A

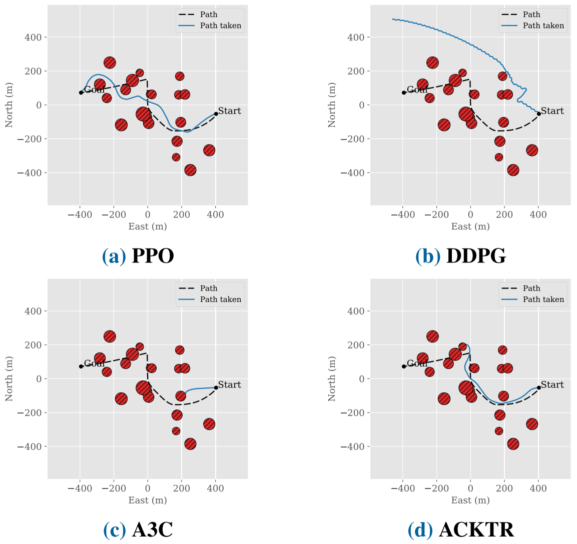
(b) Test scenario B

(c) Test scenario C

(d) Test scenario D

FIGURE 9. Agent trajectories in qualitative test scenarios when λ parameter is varied. The behaviour in terms of collision avoidance is significantly modulated.

Based on the results, it seems clear that a reactive RL agent is capable of becoming proficient at the combined path-following / collision-avoidance task after being trained using the state-of-the-art PPO algorithm. Prior to conducting any experiments, our assumption was the decreasing λ, and thus decreasing the degree to which the agent would prioritize path-adherence over collision avoidance, would lead to a higher success rate. Also, our expectation was that this performance increase would come at the expense of the agent's path following performance, leading to an increase in the average cross-track error. The results show a clear and reliable trend, supporting our hypothesis. In fact, as seen



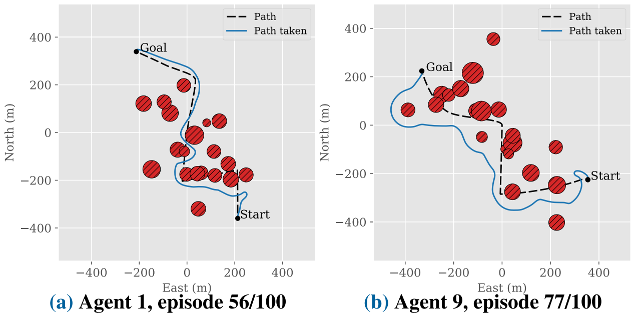
(a) PPO

(b) DDPG

(c) A3C

(d) ACKTR

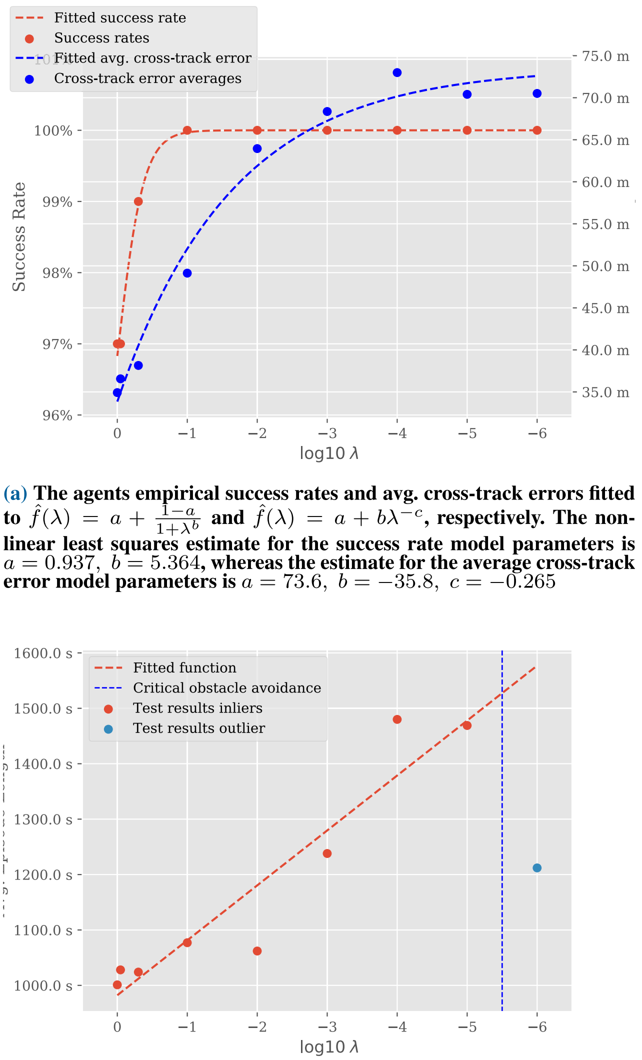
FIGURE 10. Comparison of agent trajectories in randomly generated scenario for different RL algorithms. All agents were given λ = 1. Only the PPO agent managed to reach the goal.



(a) Agent 1, episode 56/100

(b) Agent 9, episode 77/100

FIGURE 11. Example trajectories highlighting the different in guidance strategies for extreme values of the trade-off parameter λ. Evidently, the radical obstacle avoidance agent, where λ was set to 10^{-6}, clearly exhibits a more defensive behavior, basically avoiding the entire cluster of obstacles surrounding the path b. More impressively, the radical path adherence agent, with λ = 1, follows the path closely while avoiding the obstacles blocking it a.



(a) The agents empirical success rates and avg. cross-track errors fitted to $\hat{f}(\lambda) = a + \frac{1-a}{1+\lambda^b}$ and $\hat{f}(\lambda) = a + b\lambda^{-c}$, respectively. The non-linear least squares estimate for the success rate model parameters is $a = 0.937,\ b = 5.364$, whereas the estimate for the average cross-track error model parameters is $a = 73.6,\ b = -35.8,\ c = -0.265$

(b) The agent's empirical average episode length fitted to $\hat{f}(\lambda) = a - b\log_{10}\lambda$. The non-linear least squares estimate for the model parameters is $a = 982,\ b = 99.1$. The point marked as an outlier was excluded from the regression, as there is an obvious explanation as to what might cause a drop in average episode length when λ gets very small: due to the resulting radical collision avoidance strategy, the agent will tend to simply avoid the entire cluster of obstacles, instead of avoiding individual obstacles. Thus, the log-linear model will only be valid up to a certain point. In the figure, this validity threshold is labelled as the critical obstacle avoidance

FIGURE 12. Empirical success rate.

in Table 4, the collision avoidance rate stabilizes at 100% when λ is sufficiently small. Figure 11, which features two episodes extracted from the training process, clearly illustrates why a small λ will lead to a lower collision rate, but also cause a significant worsening in path following performance. From plotting the test metrics against λ, it becomes clear that the trends can be described mathematically by simple parametric functions of λ. After deciding on suitable parameterizations, we use the Levenberg-Marquardt curve-fit method provided by Python library SciPy [52] in order to obtain a non-linear least squares estimate for the model parameters. The fitted models for our evaluation metrics can be visualized in Figure 12a and Figure 12b. The fitted parametric models allow us to generalize the observed results to unseen values of λ.

V. CONCLUSION

In this work, we have demonstrated that RL is a viable approach to the challenging dual-objective problem of controlling a vessel to follow a path given by a priori known way-points while avoiding obstacles along the way without relying on a map. More specifically, we have shown that the state-of-the-art PPO algorithm converges to a policy that yields intelligent guidance behavior under the presence of non-moving obstacles surrounding and blocking the desired path.

Engineering the agent's observation vector, as well as the reward function, involved the design and implementation of several novel ideas, including the Feasibility Pooling algorithm for intelligent real-time sensor suite dimensionality reduction. By augmenting the agent's observation vector by the reward trade-off parameter λ, and thus enabling the agent to adapt to changes in its reward function, we have demonstrated experimentally that the agent is capable of adjusting its guidance strategy (i.e. its preference of path-adherence as opposed to collision avoidance) based on the λ value that is fed to its observation vector.

By means of extensive testing, we have observed that, even in challenging test environments with high obstacles densities, the agent's success rate is in the high 90s when λ is set such that it induces a strict path adherence bias, and close to 100% when a more defensive strategy is chosen. It is worth mentioning that here, we simply studied the impact of λ on the performance of the agent. It would be desirable to actually learn the optimal value of λ. This is outside the scope of our current work. However, one approach could be to learn this parameter from the Automatic Identification System (AIS) data.

A weakness of these algorithms is that they rely heavily on deep neural networks which contains a massive number of trained parameters, the interpretation of which is immensely challenging. This flaw prevents a wholehearted acceptance of these algorithms for safety critical applications. However, the current work does demonstrate the possibility of programming intelligence into these safety critical applications.

REFERENCES

[1] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton, NJ, USA: Princeton Univ. Press, 2012.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, Jul. 2016, doi: 10.1177/027836498600500106.

[3] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, Jun. 1991.

[4] D. Panagou, "Motion planning and collision avoidance using navigation vector fields," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 2513–2518.

[5] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[6] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 1, May 1999, pp. 341–346.

[7] B.-O.-H. Eriksen, M. Breivik, K. Y. Pettersen, and M. S. Wiig, "A modified dynamic window algorithm for horizontal collision avoidance for AUVs," in *Proc. IEEE Conf. Control Appl. (CCA)*, Sep. 2016, pp. 499–506.

[8] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, Jul. 1998. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijrr/ijrr17.html#FioriniS98

[9] D. K. M. Kufoalor, E. F. Brekke, and T. A. Johansen, "Proactive collision avoidance for ASVs using a dynamic reciprocal velocity obstacles method," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 2402–2409.

[10] Y. Chen, H. Peng, and J. Grizzle, "Obstacle avoidance for low-speed autonomous vehicles with barrier function," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 1, pp. 194–206, Jan. 2018.

[11] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Trans. Autom. Control*, vol. 50, no. 7, pp. 947–957, Jul. 2005.

[12] B. H. Eriksen, M. Breivik, E. F. Wilthil, A. L. Flåten, and E. F. Brekke, "The branching-course model predictive control algorithm for maritime collision avoidance," *J. Field Robot.*, vol. 36, no. 7, pp. 1222–1249, Aug. 2019.

[13] I. B. Hagen, D. K. M. Kufoalor, E. F. Brekke, and T. A. Johansen, "MPC-based collision avoidance strategy for existing marine vessel guidance systems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 7618–7623.

[14] G. Bitar, M. Breivik, and A. M. Lekkas, "Energy-optimized path planning for autonomous ferries," *IFAC-PapersOnLine*, vol. 51, no. 29, pp. 389–394, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896318321451

[15] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[16] M. Candeloro, A. M. Lekkas, A. J. Sørensen, and T. I. Fossen, "Continuous curvature path planning using Voronoi diagrams and Fermat's spirals," *IFAC Proc. Volumes*, vol. 46, no. 33, pp. 132–137, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S147466701646146X

[17] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin, "Path planning for mobile robot navigation using Voronoi diagram and fast marching," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 2376–2381.

[18] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[19] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Sep. 1996.

[20] O. A. G. Loe, "Collision avoidance for unmanned surface vehicles," M.S. thesis, Norwegian Univ. Sci. Technol., Trondheim, Norway, 2008.

[21] M. S. Wiig, "Collision avoidance and path following for underactuated marine vehicles," Ph.D. dissertation, Dept. Eng. Cybern., Norwegian Univ. Sci. Technol., Trondheim, Norway, 2019.

[22] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proc. 28th Annu. Symp. Found. Comput. Sci. (SFCS)*. Washington, DC, USA: IEEE Computer Society, Oct. 1987, pp. 49–60, doi: 10.1109/SFCS.1987.42.

[23] E. Serigstad, B.-O.-H. Eriksen, and M. Breivik, "Hybrid collision avoidance for autonomous surface vehicles," *IFAC-PapersOnLine*, vol. 51, no. 29, pp. 1–7, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896318321499

[24] B.-O. H. Eriksen, G. Bitar, M. Breivik, and A. M. Lekkas, "Hybrid collision avoidance for ASVs compliant with COLREGs rules 8 and 13-17," 2019, *arXiv:1907.00198*. [Online]. Available: http://arxiv.org/abs/1907.00198

[25] Z. Yan, Y. Zhao, S. Hou, H. Zhang, and Y. Zheng, "Obstacle avoidance for unmanned undersea vehicle in unknown unstructured environment," *Math. Problems Eng.*, vol. 2013, pp. 1–12, Nov. 2013.

[26] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, Apr. 1991, pp. 1398–1404.

[27] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017, *arXiv:1712.01815*. [Online]. Available: https://arxiv.org/abs/1712.01815

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[29] O. Vinyals. (2019). *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

[30] Society of Naval Architects and Marine Engineers (U.S.). Technical and Research Committee, *Nomenclature for Treating the Motion of a Submerged Body Through a Fluid: Report of the American Towing Tank Conference* (Technical and Research Bulletin). Jersey City, NJ, USA: Society of Naval Architects and Marine Engineers, 1950. [Online]. Available: https://books.google.no/books?id=VqNFGwAACAAJ

[31] R. Skjetne, Ø. Smogeli, and T. I. Fossen, "Modeling, identification, and adaptive maneuvering of CyberShip II: A complete design with experiments," *IFAC Proc. Volumes*, vol. 37, no. 10, pp. 203–208, Jul. 2004.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: https://arxiv.org/abs/1509.02971

[34] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmont, MA, USA: Athena Scientific, 2000.

[35] L. Tai, J. Zhang, M. Liu, J. Boedecker, and W. Burgard, "A survey of deep network solutions for learning control in robotics: From reinforcement to imitation," 2016, *arXiv:1612.07139*. [Online]. Available: https://arxiv.org/abs/1612.07139

[36] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst*, 2000, pp. 1–7.

[37] L. Weaver and N. Tao, "The optimal reward baseline for gradient-based reinforcement learning," 2013, *arXiv:1301.2315*. [Online]. Available: https://arxiv.org/abs/1301.2315

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," Jun. 2015, *arXiv:1506.02438*. [Online]. Available: https://arxiv.org/abs/1506.02438

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[40] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Germany: Springer-Verlag, 2006.

[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992, doi: 10.1007/BF00992696.

[42] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn. (ICML)*, San Francisco, CA, USA: Morgan Kaufmann, 2002, pp. 267–274. [Online]. Available: http://dl.acm.org/citation.cfm?id=645531.656005

[43] J. Schulman. (2016). *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. [Online]. Available: https://www.semanticscholar.org/paper/Optimizing-Expectations%3A-From-Deep-Reinforcement-to-Schulman/ e31692a74427b58b6154e37da7535e142ceceb4b

[44] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015, *arXiv:1502.05477*. [Online]. Available: http://arxiv.org/abs/1502.05477

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: https://arxiv.org/abs/1707.06347

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[47] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," 2018, *arXiv:1809.07731*. [Online]. Available: https://arxiv.org/abs/1809.07731

[48] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2017, *arXiv:1709.06560*. [Online]. Available: https://arxiv.org/abs/1709.06560

[49] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

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

[51] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. (2017). *Openai Baselines*. [Online]. Available: https://github.com/openai/baselines

[52] P. Virtanen *et al.*, "SciPy 1.0-Fundamental algorithms for scientific computing in Python," 2019, *arXiv:1907.10121*. [Online]. Available: http://arxiv.org/abs/1907.10121

[53] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Chichester, U.K.: Wiley, 2011.

[54] M. Breivik and T. I. Fossen, "Guidance laws for autonomous underwater vehicles," in *Underwater Vehicles*, A. V. Inzartsev, Ed. Rijeka, Croatia: IntechOpen, 2009, ch. 4, doi: 10.5772/6696.

[55] A. B. Martinsen, "End-to-end training for path following and control of marine vehicles," Dept. Eng. Cybern., Norwegian Univ. Sci. Technol., Trondheim, Norway, Tech. Rep., 2018.

[56] E. Fehlberg, "Klassische Runge-Kutta-Formeln vierter und niedrigerer ordnung mit schrittweiten-kontrolle und ihre anwendung auf Wärmeleitungsprobleme," *Computing*, vol. 6, nos. 1–2, pp. 61–71, Mar. 1970, doi: 10.1007/BF02241732.

[57] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. NIPS*, 2017, pp. 5279–5288.

[58] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," Feb. 2016, *arXiv:1602.01783*. [Online]. Available: https://arxiv.org/abs/1602.01783

EIVIND MEYER is currently working on the master's thesis and completing the five-year integrated master's degree in cybernetics and robotics with the Norwegian University of Science and Technology (NTNU), Trondheim. Having specialized in real time systems, his research interest focuses on adopting state-of-the-art artificial intelligence methods for autonomous vehicle control.

HAAKON ROBINSON received the bachelor's degree in physics and the master's degree in cybernetics and robotics from NTNU, in 2015 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Norwegian University of Science and Technology (NTNU). His current work investigates the overlap between modern machine learning techniques and established methods within modeling and control, with a focus on improving the interpretability and behavioural guarantees of hybrid models that combine first principle models and data-driven components.

ADIL RASHEED received the bachelor's degree in mechanical engineering and the master's degree in thermal and fluids engineering from IIT Bombay, and the Ph.D. degree in multiscale modeling of urban climate from the Swiss Federal Institute of Technology Lausanne. He is currently a Professor of big data cybernetics with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, where he is working to develop novel hybrid methods at the intersection of big data, physics-driven modeling, and data-driven modeling in the context of real-time automation and control. He is currently a part-time Senior Scientist with the Department of Mathematics and Cybernetics, SINTEF Digital, where he led the Computational Sciences and Engineering Group, from 2012 to 2018.

OMER SAN received the bachelor's degree in aeronautical engineering from Istanbul Technical University, in 2005, the master's degree in aerospace engineering from Old Dominion University, in 2007, and the Ph.D. degree in engineering mechanics from Virginia Tech, in 2012. He held a postdoctoral position at Virginia Tech, from 2012 to 2014, and then the University of Notre Dame, IN, USA, from 2014 to 2015. He has been an Assistant Professor of mechanical and aerospace engineering with Oklahoma State University, Stillwater, OK, USA, since 2015. He was a recipient of the U.S. Department of Energy 2018 Early Career Research Program Award in Applied Mathematics. His field of study is centered upon the development, analysis, and applications of advanced computational methods in science and engineering with a particular emphasis on fluid dynamics across a variety of spatial and temporal scales.