North American Journal of Engineering and Research

Est. 2020

Reinforcement Learning: Concepts and Practical Implementations

Kailash Alle

Email: kailashalle@gmail.com

Abstract

As a general-purpose approach, the Reinforcement Learning (RL) concept of practice demonstrating encouraging results. Foundation for resolving issues with decision-making in fields including robotics, gaming, and finance. Reducing challenges to learning and encouraging educators, researchers, and young students to use reinforcement learning (RL) as a natural tool for robotics problem solving are the goals of this effort. This work offers researchers, educators, and students at different levels (undergraduate, bachelor, master, and doctoral) an understandable, step-by-step formulation of an RL problem as well as an accessible interactive simulator. The interactive tool helps users become familiar with the main ideas of reinforcement learning, as well as how to formulate and apply problems using it. In this work, RL is applied to a robotics 2D navigation problem where the robot must try to achieve a goal point while avoiding collisions with objects. For instructional purposes, a navigational problem is easy to understand and practical because there is just one possible outcome—that is, either the goal is accomplished or not, a collision occurs or not. Owing to the dearth of freely available graphical interactive simulators in the field of reinforcement learning, this work integrates theoretical explanation with a user-friendly practical tool to aid comprehension. The outcomes showcased are generated using an open-source Python script designed to lower the learning curve in this cutting-edge robotics research area.

Keywords: Reinforcement Learning, Robotics, Gaming

Introduction

When the underlying workings of the environment in which they operate are unclear or incomplete, robotic applications including navigation and control can become more complex or costly. The ability to independently extract relevant knowledge from its environment is already present in a modern robotic system equipped with neural network-based perception algorithms [1, 2, 3, 4]. The acquired perceived knowledge must ultimately lead to environmental control activities.

Particularly, RL is becoming more and more popular as the obvious solution for robotics control since it gives robots the capacity to learn about complexity and unpredictability in the real environment. [6, [7], [8], [9], and [5]. Designed to Through experience and trial-and-error interaction with the outside world, model-free reinforcement learning creates a posteriori (empirical) knowledge. Thus, model-free approaches are favored over prior knowledge approaches like model-based approaches to convey the essence of interactive learning in order to fulfill the didactic purpose of this study. Furthermore, Sutton and Barto [10] suggest that tabular approaches are the easiest to use when determining the best values for minor issues with few states and actions, like the one this work addresses. Because of its ease of use and adaptability, the well-known algorithm known as Q-learning [11] is selected to solve the suggested case study [12], [13]. Action-values are kept and updated in a lookup table by a Q-learning algorithm.

In practice, learning entails modifying those action values repeatedly until the desired behavior is achieved. Because it is a tabular approach, this algorithm can only work with reasonably modest lookup tables. For aspiring researchers and students in the field of mobile robots, this paper presents the idea of reinforcement learning. The link between theory and application for beginners is not adequately explored in recent studies like [14], [15], and [16]. As an essential learning tool, [14] is one example; nonetheless, it takes a different route and does not adhere to the required visual exposition of the Markovian element in RL algorithms. Accordingly, the theory underlying this work is that the didactical goal is enhanced when the theoretical framework and the simulation findings work together.

Moreover, the robotics research community is free to utilize applications like the ones described in this study for learning and RL research. As previously mentioned, the examination of the Q-learning algorithm's simplicity and its didactic implementation are the focus of this work.

Current approaches, however, concentrate on broadening the use of tabular techniques such as approximated Q-learning and Deep Q-Networks (DQN) [17] and on enhancing anti-bias methods as suggested in DDQN [18] study. However, there are still updates and enhancements being made to Q-learning algorithms, with a primary focus on DQN. Examples of these are Quantile Regression Q-learning [19] and its extension, The work on Categorical DQN on [21] and Implicit Quantile Deep Q-learning [20] is done. In order to make decisions on high dimensional state and action spaces, RL robotics designers have recently focused on actor-critics-based algorithms, which have more complex solutions and occasionally call for two or more neural networks. These methods INCLUDE semi-model-based approaches like I2A [29], MBVE [30], and actor-critic methods like DDPG [22], A3C [23], TRPO [24], SAC [25], PPO [26], TD3 [27], and IMPALA [28]. By adding layers of intricacy, the state-of-the-art's sophistication makes it easier to solve complex problems. However, because it necessitates extensive mathematical understanding, it creates impediments to the learning of the foundational principles. Selecting traditional approaches such as Q-learning makes sense when the objective is to help novices understand reinforcement learning.

A public repository offers an interactive simulator that can help with comprehension of the concepts and difficulties [31]. By using mathematical explanations or pseudo code alone, the tool makes implementation details that could be hard to understand visible. By using the program, one can experiment with Q-learning and change the navigational problem's environment-related characteristics. What this article contributes is as follows:

Describe the construction of an RL system through an easy-to-follow, sequential approach.

Give instructors, researchers, and students an easy-to-use software application that can be used to solve robotics challenges using reinforcement learning.

Encourage and hasten the robotics industry's use of RL techniques.

This is the structure of the paper: Reinforcement learning theory is presented in part II, which is fundamental. The framework, findings, and case study on navigational robotics are presented in Section III. The constructed interactive simulator is shown in Section IV. Finally, a quick summary of this research's findings is provided in section V.

History

The pertinent history of reinforcement learning is revealed in this section. The basic components of the RL paradigm are initially presented, including the establishment of goals and rewards, the agent-environment interaction, and the Markov Decision Process, which is the issue an RL agent must resolve in order to learn a behavior. The Q-learning algorithm is finally introduced by (i) outlining its primary characteristics, such as being an Off-Policy and Temporal-Difference method that is employed in control situations, and by (ii) outlining its convergence qualities.

Environment-Agent Relationship

Sutton and Barto [32] define reinforcement learning (RL) as the "process of learning a behavior through interaction to achieve a goal." The agent is defined as the creature that learns. Every time step t, the agent assesses the current state st (location, battery level, velocity, etc.) and selects an action at (such as moving forward, jumping, or remaining still) based on the last1 taught behavior (policy) πt, which is denoted by at = πt(at |st). When an action is applied at, the environment reacts by sending a feedback scalar reward signal (rt+1 ∈ ℜ) that indicates the action's immediate value.2.
Finally, the agent repeats the procedure, beginning at the following stage, st+1, after learning how to behave from the encounter.
A discrete time sequence with t = 0, 1, 2, 3,... is used to describe this agent-environment interaction.
The earlier explanation is illustrated by the image 1.

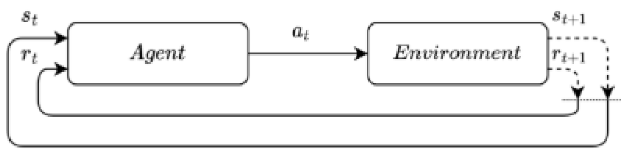
Prizes and Objectives

By employing incentive signals, the conventional reinforcement learning framework aims to codify the agent's objective.

A reward is only a scalar value, rt ∈ ℜ. The agent's primary goal is to maximize the total reward over the long term.Three [33]

Finding the ideal strategy π⋆} that maximizes the predicted cumulative reward is hence the aim of reinforcement learning (RL) as follows:

Gt is the cumulative reward, which is defined as follows: π⋆.= arg max πE [Gt], (1)
Gt = rt+1 + Άrt+2 + γ}2 rt+3 +... = X∞ k=0 γ k rt+k+1, (2)
The behavior of the agent at timestep t is indicated by the subscript of t in the policy πt. Based on the knowledge gathered up until timestep t, the agent selects a course of action. According to convention, the reward signal is thought to be received in timestep t + 1.
3A theory called the reward hypothesis.



where the discount factor, γ, values the uncertainty of future rewards and is in the interval [0, 1]. Gt is kept from adding up to infinity (diverging) via a mathematical technique called discounting. Selecting the appropriate γ value is essential to the learning process. The agent becomes more "myopic," focusing on maximizing projected immediate rewards, when the discount factor is small (γ → 0).

Conversely, when selecting big discount factors (γ → 1), the agent becomes more forward-thinking and prioritizes benefits in the future. Usually, this γ value is adjusted through trial and error to match the behavior of the intended agent.

The process of creating an appropriate reward function is a crucial factor in determining if an RL application is successful. [34]. because shoddy design could lead to undesirable or constrained behavior by diverting the agent's learning process. Theoretically, an agent's reward function is how the RL designer tells it what to do, as opposed to how to accomplish a certain objective [33]. If a goal can be established by domain knowledge and intuition, the reward function is typically created by the user by hand [35]. Gradually increasing task complexity necessitates more indirect approaches, such as learning reward functions rather than designing them by hand [36], [37], [38].

Resolving the Markov Question

A Markov Decision Process (MDP) is the foundation for the theoretical framework known as reinforcement learning (RL). A decision-making problem is modeled mathematically using an MDP. A regulated discrete stochastic chain of state transitions makes up a decision-making issue. It is governed by the fact that the agent choose an action an in an action space A to transit to a new state s in a state space S. It is stochastic in the sense that state action transitions have a certain probability, p(s′|s, a). It is discrete because every transition is a distinct event that takes place at a certain timestep (t). RL is a legitimate method for solving a problem if it can be expressed as an MDP. In specifics, a tuple defines an MDP. S, A, R, p, γ , in where S stands for the set of states, A for the set of acts, and R for the set of benefits. If S, A, and R are finite, then the following discrete probability distributions, p: S × R × S × A → [0, 1], accurately represent the likelihood, at each timestep t, of transiting to a later state (st+1 ∈ S) and earning a reward (rt+1 ∈ R) following action at given current state st:

p(s ′ ,r|s, a) .= Pr{st+1 = s ′ ,rt+1 = r|st = s, at = a}.

Significantly more, the likelihood of transitioning to a new state s ′ given s and an is determined by the expected state-transition probability function, which is defined as follows:

p(s ′ |s, a) .= Pr{st+1 = s ′ |st = s, at = a} = X r∈R p(s ′ ,

= X r∈R p(s ′ ,r|s, a).

The Markov condition must hold in order for the dynamics shown in (3) to hold. In particular, just the prior state, st, and action at—rather than other past states and actions—are used to completely characterize the probability of transiting from st to st+1. In other words, Pr{st+1|st}.= Pr{st+1|s0,...,st} is the knowledge that the state must contain regarding everything that occurred in previous interactions that potentially have an impact on the future.

Action

The complete definition of behavior π is provided by a probability distribution4 over actions given states, that is

$$\pi(a|s) \doteq \Pr\{a_t = a|s_t = s\}.$$

An action is selected based on its likelihood given the state, presuming that a state contains all the information the agent needs to know about its current circumstances. From a conceptual standpoint, the effective probability transition between states is a function of both the environment, which is outside the agent's control, and its decisions. Put differently, the likelihood of moving from sto s′ is determined by multiplying the policy function for every action that is feasible by the chance of the environment's transition dynamics probability function, p(s′|s, a) (described in 4).

The Quality of The Action

As with all RL techniques, the agent must be aware of how effectively the policy is working at any given time in order to make improvements. at the specified time. Each RL approach is unique in the way that the policy varies based on experience. It is obvious that the predicted future benefit that a policy can receive determines how beneficial it is.

A value function describes the value of a policy. In specific terms, the state-value function of a policy vπ can be expressed as the predicted (discounted) cumulative reward that an agent could receive if they were to follow the policy π for all time after reaching states.

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t | s_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s\right], \quad \forall s \in \mathcal{S}.$$

A unique variety of value function in addition to the state-value function is the action-value function (qπ), which is defined as the predicted future reward following policy π for all time after an action is taken in a particular state.

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t | s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s, a_t = a\right].$$

An Ideal Practical Action

An agent's ultimate goal is to act in an optimal manner. To reach optimality, one must identify the best policies, π⋆, and the corresponding value functions, v⋆ and q⋆. For all policies, the maximum state-value and action-value functions are known as the optimal state-value (v⋆) and action-value (q⋆) functions.

$$v_\star(s) \doteq \max_\pi v_\pi(s) \text{ and } q_\star(s, a) \doteq \max_\pi q_\pi(s, a)$$

Methods To Resolved

An agent aims to solve the MDP it is placed in at the end. The agent employs a basic equation known as the Bellman Equation to solve it by having complete knowledge of the MDP, which is knowing P an s, s ′, and Ra s. The helpful concept that the value of a state is divided into two halves lends credence to this equation:

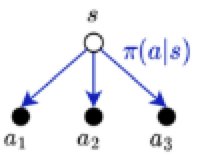
Rewards immediately: rt+1,

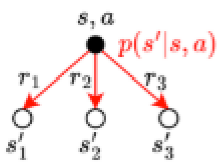
The successor state's discounted value is represented by γvπ (st+1).

The expected cumulative reward under a given policy (π) is represented by the Bellman Expectation Equation, while the expected cumulative reward by performing optimally (π⋆) is represented by the Bellman Optimality Equation. These two forms of Bellman Equations are different from one another.

Appraising The Action

Finding qπ and vπ through backup operations is the process of solving an MDP. Transferring value information from its successor states to a starting point in any state is known as "backing up." Because value functions have recursive relationships, a backup method that uses the dynamics of the environment p(s ′ | s, a) is used to find the current values.





π(s|a), as well as the discounted value of the successor state (γvπ (s ′) or γqπ (s ′, a ′)). The following is the breakdown of vπ using the definition from (6):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma\left(r_{t+2} + \gamma r_{t+3} + \cdots\right) | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma G_{t+1} | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s\right]. \end{aligned}$$

Similarly, by using (7), q_π can be decomposed as follows:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right]. \quad (10)$$

Concretely, v_π (as depicted by the Figure 2a) is the weighted sum over all action-values, that is,

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a). \quad (11)$$

In the same way, q_π (as depicted by the Figure 2b) is equal to the immediate reward plus a discounted weighted sum over all successor state-values as follows:

$$q_\pi(s, a) = r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v_\pi(s'), \quad (12)$$

Finally, the Bellman Expectation Equations are obtained by merging the relationships from (11) and (12). The Bellman Expectation Equation for $v_\pi(s)$ (as depicted by the Figure 3a) is defined as

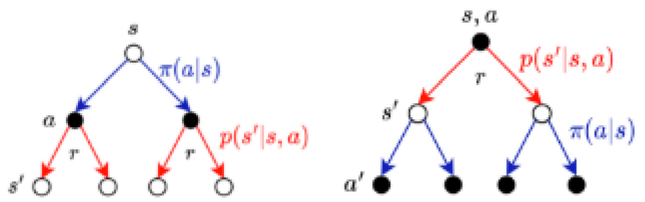
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v_\pi(s')\right), \quad (13)$$

and the Bellman Expectation Equation for $q_\pi(s, a)$ (as depicted by the Figure 3b) is defined as

$$q_\pi(s, a) = r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a'). \quad (14)$$

Actions Ideal

When acting greedily with respect to the optimal action-value function q⋆, optimal performance is determined by selecting the best course of action at every given moment. Taking action involves maximizing over the q⋆.



that results in a higher predicted return for us. Therefore, the following optimum policy π⋆ defines optimal behavior:

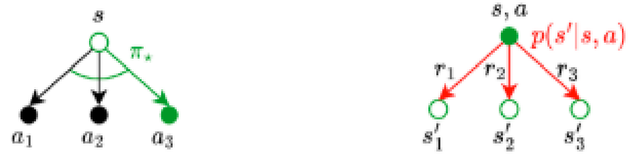
$$\pi_\star(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_\star(s,a) \\ 0 & \text{otherwise} \end{cases} \qquad (15)$$

This means that, according to (9), the best action-value is identical to the optimal state-value, v⋆, as shown in Figure 4a:

$$v_\star(s) = \max_{a \in \mathcal{A}(s)} \mathbb{E}\left[r_{t+1} + \gamma v_\star(s_{t+1}) | s_t = s, a_t = a\right]. \qquad (16)$$

Furthermore, according to (10), the ideal action-value (shown in Figure 4b) q⋆ is

$$q_\star(s,a) = \mathbb{E}\left[r + \gamma \max_{a' \in \mathcal{A}(s)} q_\star(s',a') | s_t = s, a_t = a\right], \qquad (17)$$



Lastly, by combining the connections from (16) and (17), the Bellman Optimality Equations are produced. The definition of the Bellman Optimization Equation for v⋆ (as shown in Figure 5a) is as follows:

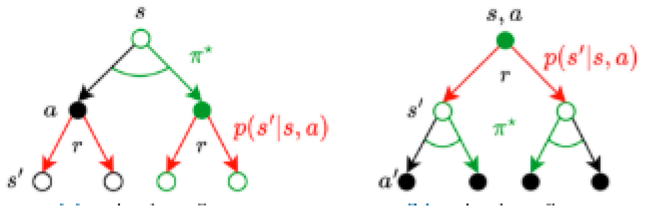
$$v_\star(s) = \max_{a' \in \mathcal{A}(s)} r_{t+1} + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\star(s'). \qquad (18)$$

Furthermore, the Bellman Optimality Equation for q⋆ (as shown in Figure 5b) has the following definition:

$$q_\star(s,a) = r_{t+1} + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}(s)} q_\star(s',a'). \qquad (19)$$

Q-Science

The creation of the popular Model-Free Off-Policy Temporal-Difference Control algorithm, known as Q-learning, was a significant advancement in the early days of reinforcement learning [11], [39].



Managing With RL: A Textbook Case Study

In this part, a case study of a 2D navigational robotics issue is presented. It is easy and practical to use a navigational task for waypoint-goal achieving with or without obstacle avoidance [31] for instructional purposes since it has comprehensible outputs.

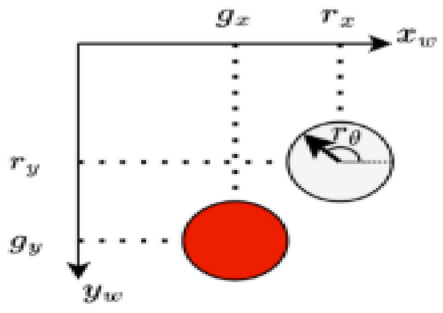
It involves figuring out how to get from one place in a grid-like environment to another while dodging any obstacles that may be there. The problem is separated into two tasks, each with a different environment configuration, from an incremental perspective: a collision-free job that needs goal reaching in an environment devoid of obstacles, and a collision-avoidance task that needs goal reaching in an environment scattered with obstacles. Q-learning is utilized to answer this case study, and the following discrete phases address the appropriate step-by-step problem formulation:

State design: specify the state that can encode the awareness needed by the agent to observe its surroundings and recognize its objective;

Selecting the appropriate actions to enable exploration and collision avoidance is known as action design.

Determining a function that more accurately captures the goals of the task in light of the agent's circumstances at each timestep is reward design.

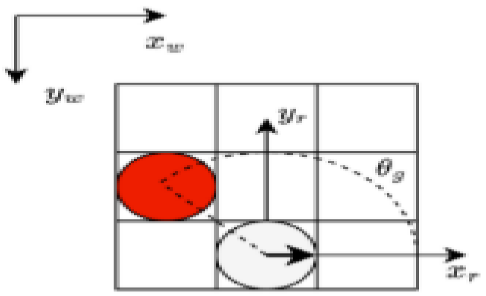
Ultimately, the agent needs to pick up a policy that guides the robot toward a waypoint-goal while dodging obstacles along the way. The robot's linear velocity will always be positive, constant, and non-zero. Discrete steering actions, or angular velocities, are the possible control actions. Because they don't show any dynamism during training, the goal and the obstacles are static. Furthermore, keep in mind that discrete techniques like Q-learning are susceptible to the dimensionality curse, which can lead to high computing costs if left unchecked [10].

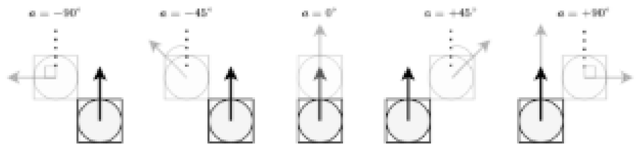


To create tiny lookup tables, state and action spaces must therefore be comparatively small. As a result, a thorough design of small state and action spaces is carried out to guarantee the viability of this case study against this specific caveat.

Activity Designing

With a constant, positive, and uncontrollable linear velocity, the agent advances automatically one cell in the direction of its orientation aθ for each timestep. You can only control its angular velocity. As a result, the set of steering actions around the agent's rotational axis, represented by the formula A = {−90◦, −45◦, 0◦, 45◦, 90◦}, is called the action space.





The agent then moves by the following kinematics:

$$\begin{bmatrix} r_x[t+1] \\ r_y[t+1] \\ r_\theta[t+1] \end{bmatrix} = \begin{bmatrix} r_x[t] + v\cos(r_\theta[t] + \frac{a}{2}) \\ r_y[t] + v\sin(r_\theta[t] + \frac{a}{2}) \\ r_\theta[t] + a \end{bmatrix},$$

Conclusion

With a focus on robotics specifically, this effort aims to encourage and support the use of reinforcement learning (RL) as a logical next step in the development of autonomous systems. In particular, it presents a clear and practical robotics case study where the principles are given with care, and it teaches the main theoretical concepts underpinning reinforcement learning.

The case study's ease of use makes it possible to achieve the targeted educational goal without drawing attention away from the main goal of the project, which is to help newcomers understand real language. An interactive simulator is also available, enabling the user to observe how reasonable and highly reliant RL-based results are on small adjustments to their basic settings.

Ultimately, this work differs from other scientific works and open source repositories due to its theoretical foundation, clear, step-by-step formulations, and interactive software tool availability.

more case study applications, cross-platform (web, mobile) compatibility, visual exposition of training metrics, support for more Q-learning variations or other RL algorithms, and other expansions of the simulator would be beneficial for future development.

With a focus on robotics specifically, this effort aims to encourage and support the use of reinforcement learning (RL) as a logical next step in the development of autonomous systems. In particular, it presents a clear and practical robotics case study where the principles are given with care, and it teaches the main theoretical concepts underpinning reinforcement learning.

The case study's ease of use makes it possible to achieve the targeted educational goal without drawing attention away from the main goal of the project, which is to help newcomers understand real language. An interactive simulator is also available, enabling the user to observe how reasonable and

highly reliant RL-based results are on small adjustments to their basic settings.

Ultimately, this work differs from other scientific works and open source repositories due to its theoretical foundation, clear, step-by-step formulations, and interactive software tool availability.

more case study applications, cross-platform (web, mobile) compatibility, visual exposition of training metrics, support for more Q-learning variations or other RL algorithms, and other expansions of the simulator would be beneficial for future development.

References:

[1] M. I. Pereira, P. N. Leite, and A. M. Pinto, “Detecting docking-based structures for persistent ASVs using a volumetric neural network,” in Proc. Global Oceans, 2020, pp. 1–6.

[2] P. N. Leite and A. M. Pinto, “Exploiting motion perception in depth estimation through a lightweight convolutional neural network,” IEEE Access, vol. 9, pp. 76056–76068, 2021.

[3] M. I. Pereira, R. M. Claro, P. N. Leite, and A. M. Pinto, “Advancing autonomous surface vehicles: A 3D perception system for the recognition and assessment of docking-based structures,” IEEE Access, vol. 9, pp. 53030–53045, 2021.

[4] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, “Learning to walk via deep reinforcement learning,” 2018, arXiv:1812.11103.

[5] M. Everett, Y. F. Chen, and J. P. How, “Collision avoidance in pedestrianrich environments with deep reinforcement learning,” IEEE Access, vol. 9, pp. 10357–10377, 2021.

[6] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, “Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning,” IEEE Access, vol. 7, pp. 165262–165278, 2019

[7] X. Cao, C. Sun, and M. Yan, “Target search control of AUV in underwater environment with deep reinforcement learning,” IEEE Access, vol. 7, pp. 96549–96559, 2019.

[8] X. Wang, M. C. Gursoy, T. Erpek, and Y. E. Sagduyu, “Learningbased UAV path planning for data collection with integrated collision avoidance,” IEEE Internet Things J., vol. 9, no. 17, pp. 16663–16676, Sep. 2022.

[9] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, “How to train your robot with deep reinforcement learning: Lessons we have learned,” Int. J. Robot. Res., vol. 40, nos. 4–5, pp. 698–721, Apr. 2021.

[10] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.