Navigation of Autonomous Vehicles using Reinforcement Learning with Generalized Advantage Estimation

Edwar Jacinto[1], Fernando Martínez[2], Fredy Martínez[3]
Universidad Distrital, Francisco José de Caldas
Bogotá D.C., Colombia

***Abstract*—This study proposes a reinforcement learning approach using Generalized Advantage Estimation (GAE) for autonomous vehicle navigation in complex environments. The method is based on the actor-critic framework, where the actor network predicts actions and the critic network estimates state values. GAE is used to compute the advantage of each action, which is then used to update the actor and critic networks. The approach was evaluated in a simulation of an autonomous vehicle navigating through challenging environments and it was found to effectively learn and improve navigation performance over time. The results suggest GAE as a promising direction for further research in autonomous vehicle navigation in complex environments.**

***Keywords*—Actor-critic; autonomous vehicles; generalized advantage estimation; navigation; reinforcement learning**

I. Introduction

Autonomous vehicles have the potential to revolutionize transportation by reducing accidents, improving efficiency, and providing access to mobility for those who may not be able to drive [1], [2]. However, their deployment is limited by their ability to navigate complex environments, which can be affected by various factors such as weather, traffic, and pedestrians [3]. In addition, traditional approaches to autonomous navigation often rely on hand-designed rules or pre-defined maps, which may need to be revised to handle the variety and unpredictability of real-world environments [4], [5]. This can lead to poor performance and even accidents in complex scenarios.

From the point of view of robotic applications, one of the main problems in autonomous navigation is localizing the robot's position and orientation in the environment [6], [7]. This is typically done using sensors such as cameras, lasers, and inertial measurement units (IMUs), which provide noisy and partial observations of the environment [8], [9]. The robot must then fuse these observations with a map of the environment and estimate its pose using techniques such as Kalman filters, particle filters, or SLAM (simultaneous localization and mapping). However, these techniques can suffer from drift, ambiguity, and inconsistency, especially in dynamic or cluttered environments, leading to erroneous or uncertain pose estimates [10].

Another problem is planning and executing safe, efficient, and feasible trajectories [11], [12]. The robot must consider various constraints and objectives, such as avoiding collisions, respecting traffic rules, minimizing energy consumption, and following a given path or mission [13]. This requires sophisticated algorithms that can reason about the robot's dynamics, kinematics, sensor models, and the environment's geometry, dynamics, and hazards [14]. These algorithms may include motion planners, path followers, and trajectory optimizers, which can be implemented using sampling-based planning, optimal control, and reinforcement learning techniques [15]. However, these techniques can be computationally intensive and may only sometimes find a solution, especially in complex or changing environments.

A third problem is adapting and learning from the environment [16]. The robot must learn and generalize from past experiences and observations to improve its performance, robustness, and flexibility [17]. This requires deep learning, transfer learning, and meta-learning, enabling the robot to learn features, models, and policies from data and transfer them to new tasks or situations [18], [19]. However, these techniques require large amounts of data and computation and may suffer from overfitting, generalization error, and sample efficiency.

To address this problem, a reinforcement learning approach using Generalized Advantage Estimation (GAE) to enable autonomous vehicles to learn to navigate complex environments is proposed [20], [21]. Reinforcement learning enables agents to learn by interacting with their environment and receiving feedback as rewards or penalties [22]. It has been successfully applied to various problems, including autonomous navigation [23]. However, traditional reinforcement learning approaches often require many interactions with the environment to learn effectively, which can be impractical in real-time scenarios such as autonomous navigation.

II. Background

GAE is a method for estimating the advantage of each action in a reinforcement learning algorithm, which is used to update the policy [24]. It was developed to address the problem of high variance in traditional reinforcement learning approaches, which can lead to slow learning and poor performance. GAE uses a linear combination of the value function and the reward to compute the advantage, which helps to reduce the variance and accelerate learning. It is effective in various environments, including robotic manipulation tasks [25].

To this end [26] propose Observational Imitation Learning (OIL), a novel imitation learning variant that supports online

training and automatic selection of optimal behavior by observing multiple imperfect teachers. [27] describe a generic navigation algorithm that uses data from sensors onboard the drone to guide the drone to the site of the problem. [28] propose a two-stage reinforcement learning (RL) based multi-UAV collision avoidance approach without explicitly modeling the uncertainty and noise in the environment. It is particularly an arduous task when handling multi-agent systems where the delay of one agent could spread to other agents. To resolve this problem [29] propose a novel framework to deal with delays as well as the non-stationary training issue of multi-agent tasks with model-free deep reinforcement learning. MIDAS uses an attention mechanism to handle an arbitrary number of other agents and includes a "driver-type" parameter to learn a single policy that aims of [30]. A neural network-based reactive controller is proposed for a quadrotor to fly autonomously in an unknown outdoor environment [31]. [32] aim to combine cloud robotics technologies with deep reinforcement learning to build a distributed training architecture and accelerate the learning procedure of autonomous systems. A deep reinforcement learning-based UANOA (USVs autonomous navigation and obstacle avoidance) method is proposed [33]. Nowadays, modern Deep-RL can be successfully applied to solve a wide range of complex decision-making tasks for many types of vehicles. Based on this context [34] propose the use of Deep-RL to perform autonomous mapless navigation for Hybrid Unmanned Aerial Underwater Vehicles (HUAUVs), robots that can operate in both, air or water media. Other influential work includes [35].

This work proposes a reinforcement learning approach using GAE for autonomous vehicles navigating complex environments. Our method is based on the actor-critic framework, where the actor-network predicts the actions to take, and the critic network estimates the value of each state. GAE is used to compute the advantage of each action, which is applied to update the actor and critic networks. Our method is evaluated on a simulation of an autonomous vehicle navigating through a series of challenging environments and show that it can learn to navigate effectively and improve its performance over time. Our results demonstrate the potential of GAE for enabling autonomous vehicles to navigate through complex environments and suggest that it could be a promising direction for further research.

III. Problem Statement

Autonomous vehicles have the potential to revolutionize transportation, but their ability to navigate complex environments is crucial for their practical deployment. Unfortunately, traditional approaches to autonomous navigation often rely on hand-designed rules or pre-defined maps, which may need to be revised to handle the variety and unpredictability of real-world environments. This can lead to poor performance and even accidents in complex scenarios.

Reinforcement learning is a promising approach for enabling autonomous vehicles to learn to navigate through complex environments. However, traditional reinforcement learning approaches often require many interactions with the environment to learn effectively, which can be impractical in real-time scenarios such as autonomous navigation. To address this problem, the use Generalized Advantage Estimation (GAE) to reduce the variance and accelerate learning in reinforcement learning for autonomous vehicles is proposed.

This research aims to develop a reinforcement learning approach using GAE for autonomous vehicles navigating through complex environments and evaluate its performance on a simulation of an autonomous vehicle navigating through a series of challenging environments. The aim is to demonstrate that our method can learn to navigate effectively and improve its performance over time and to show that GAE has the potential to be a promising direction for further research in this area.

A simplified example to illustrate the basic ideas of using GAE to enable an autonomous vehicle to navigate through a complex environment can be seen in Algorithm 1. In practice, additional factors such as perception, planning, and dealing with real-world constraints and uncertainties must be considered. In this example, the environment is represented by the $Environment$ class, which simulates the vehicle's interactions with the environment. The $actor\_network$ and $critic_netwo rk$ are neural networks that predict the actions to take and the value of each state, respectively. The Adam optimizer is used to update the networks based on the loss.

The algorithm's main loop runs through the episodes, where an episode represents one complete run through the environment. Within each episode, the algorithm runs through the timesteps, where a timestep represents one action taken by the vehicle. At each time step, the algorithm predicts the $action$ and the $value$ using the actor and critic networks, takes action, and observes the next $state$, $reward$, and $done$ flag. The $reward$ and the $value$ are stored in buffers, and the state is updated.

When the episode is complete, the algorithm computes the advantage using the GAE algorithm. It does this by first predicting the $value$ of the final state using the critic network and then using this value along with the rewards and values from the episode to compute the returns using the $compute\_gae$ function. The difference between $returns$ and $values$ then defines the $advantages$.

Then actor and critic losses are calculated. Actor loss is a measure of the quality of the action selection. It is computed using the log probability of the action taken, as predicted by the actor-network, and the advantage, as estimated by the GAE. The advantage represents the excess reward obtained from an action over the baseline value, and it reflects the relative importance of the action in the long run. The log probability represents the confidence of the actor-network in the action and reflects the risk of the action in the short run. The actor loss is defined as the negative dot product of these two quantities, which indicates the trade-off between exploration and exploitation. The actor loss is minimized during training to improve the action selection of the robot. Mathematically, the actor loss is defined as (Eq. 1):

$$actor\_loss = -mean\left(log\_probs \times advantages\right) \quad (1)$$

where log_probs is a tensor of log probabilities, and $advantages$ is a tensor of advantages.

Algorithm 1 Pseudocode for Reinforcement Learning Algorithm using GAE

```
# Initialize the actor and critic networks
actor = Actor_network()
critic = critic_network()

# Initialize the optimizer and the GAE parameters
optimizer = Adam(actor.parameters(), lr=learning_rate)
gamma = 0.99 # Discount factor for future rewards
beta = 0.95 # Weight factor for the advantage

# Loop through the episodes
for episode in range(num_episodes): do
    # Initialize the environment and the state
    env = Environment()
    state = env.reset()

    # Initialize the reward, value, and advantage buffers
    rewards = []
    values = []
    advantages = []

    # Loop through the timesteps
    while not env.done: do
        # Predict the action and the value using
        # the actor and critic networks
        action = actor(state)
        value = critic(state)

        # Take the action and observe the next state,
        # reward, and done flag
        next_state, reward, done = env.step(action)

        # Store the reward and the value
        rewards.append(reward)
        values.append(value)

        # Update the state
        state = next_state
    end while
    # Compute the advantage using GAE
    next_value = critic(state)
    returns = computate_gae(next_value, rewards)
    returns = computate_gae(values, gamma, beta)
    advantages = returns − values

    # Compute the actor and critic losses
    actor_loss = (-log_probs * advantages).mean()
    critic_loss = advantages.pow(2).mean()

    # Backpropagate the losses and update the actor
    # and critic networks
    loss = actor_loss + critic_loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
end for
```

The critic loss is a measure of the quality of the value estimation. It is computed using the advantage, as estimated by the GAE, and the value, as predicted by the critic network. The advantage represents the excess reward obtained from a sequence of actions over the baseline value, and it reflects the relative importance of the actions in the long run. The value represents the expected reward obtained from a state or action, and it reflects the long-term potential of the state or action. The critic loss is defined as the mean squared error between these two quantities, which indicates the deviation of the value from the true advantage. The critic loss is minimized during training to improve the value estimation of the robot. Mathematically, the critic loss is defined as (Eq. 2):

$$critic\_loss = mean\left(advantages.pow\left(2\right)\right) \tag{2}$$

where advantages is a tensor of advantages.

Finally, losses are backpropagated, and the model weights are updated. The losses are backpropagated through the model's computation graph to compute the model weights' gradients concerning the losses. The gradients are accumulated over an episode's timesteps and are used to update the model weights using the optimizer algorithm. The optimizer algorithm performs stochastic gradient descent on the model weights, using the gradients as the updated direction and the learning rate as the updated step size.

IV. Methods

The algorithm is developed for a small autonomous robot under the following considerations that replicate the functional characteristics of our working platform (the robot could move to any of the four adjacent cells in the grid by selecting one of the following actions: up, down, left, or right). [36]:

- The robot has a state space of size four, representing the state of the robot in the environment.
- The robot has an action space of size two, which represents the actions that the robot can take in the environment.
- The robot can be rewarded for performing a specific action in the environment.
- The robot's state can change after taking action in the environment.
- The robot's episode can be terminated (done flag set to True) based on a particular environmental condition.
- The robot has an actor-network, which takes in the current state of the robot and outputs a probability distribution over the possible actions.
- The robot has a critic-network, which takes in the current state of the robot and outputs a value estimate for the current state.

Python and PyTorch are used for the implementation. The first step consists of importing the libraries and defining algorithm parameters, such as the number of episodes (1000) and the learning rate (0.001). Next, the actor and critic networks is

defined using PyTorch. The actor-network is a simple feedforward neural network with four input units (corresponding to the state size), two output units (corresponding to the action size), and two hidden layers with eight units each. The critic network is also a simple feedforward neural network with four input units (corresponding to the state size) and one output unit (corresponding to the value estimate). Both networks use ReLU activation functions.

Then, a function called $compute\_gae()$ is defined to compute the generalized advantage estimate for a given set of rewards, values, and discount factor. This function takes in the following value estimate, the rewards, the values, the discount factor, and the weight factor for the advantage, and returns the generalized advantage estimates.

The next step would be to define the environment with which the robot will interact. This can be done by creating a class for the environment, which should have methods for resetting the environment, stepping through the environment with an action, and checking if the episode is done. The environment should also have necessary attributes, such as the state and reward at each timestep and other relevant information. For example, in the case of a small robot navigating through a maze, the environment may have a 2D grid representation of the maze, with the state being the current position of the robot and the reward being based on the distance to the goal.

Finally, the algorithm's main loop is implemented, consisting of running through a series of episodes. In each episode, the environment is restarted, and the state is initialized. At each time step, the actor and critic networks are used to predict the action and the value, respectively, and the action is performed on the environment. The reward and value are stored in buffers. Once the episode is terminated (either by a termination condition or by reaching a maximum number of timesteps), the GAE algorithm is used to compute the rewards. For this purpose, the function $compute\_gae()$ is used, which takes the following value (predicted by the critical network), the rewards, the values (predicted by the critical network at each timestep), the discount factor ($gamma$) and the weight factor ($beta$). In the end, it returns the calculated payoffs. The losses of the actor and the critic are calculated using the advantages and the stored probabilities and values. These losses are backpropagated through the networks, and the optimizer updates the networks. This process is repeated for the specified number of episodes, updating the actor and critic networks at each iteration based on the observed rewards and advantages.

V. Results

The experimental setup was designed entirely in Python 3.8.16 (GCC 7.5.0) and respected the motion constraints of our ARMOS TurtleBot robot as well as the experimental navigation environment. The robot was modeled as an agent of the environment. The Deep-RL algorithm with GEA was also implemented with Python and the PyTorch 1.13.0 library.

For all training episodes, the initial position of the vehicle was set at the origin of the environment (lower left corner), with coordinates (0.0, 0.0), with its front pointing towards the positive x-axis. In each episode, a randomly generated target point (final navigation destination) was created in the environment that the agent had to reach. In case of encountering an obstacle, or a boundary of the environment, the agent turns its position at a random angle and continues to move forward. The episode ends if the vehicle has reached a total of 1000 steps in that episode. Reaching the target point does not terminate the episode.

The Adam optimizer was used to train the actor and critical neural networks, with a learning rate of 0.001 for each method. In both instances, a 256-piece minibatch size was chosen. According to the reward values the agents attained, a training cap of 1000 episodes was established. In order to ensure a robust and effective learning algorithm, an exploration rate of 0.5 was employed for the actor network and 0.2 for the critic network.

The reward value of one of the tests developed can be seen in Fig. 1. It can be observed that it takes on average a little more than 200 episodes to start learning the task, but after this stage, it increases almost steadily the reward value until it reaches close to the maximum saturation value.

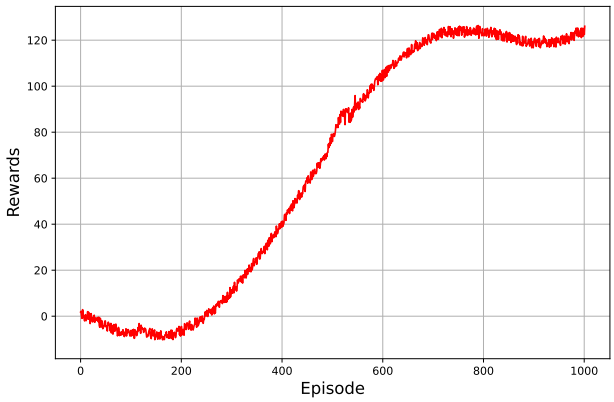

Fig. 1. Moving of the reward over 1000 episodes of the training

Our algorithm has been extensively validated and has consistently demonstrated accurate and effective agent behavior in the environment. The agents were able to learn the features of the environment and generalize their knowledge to successfully navigate without the use of external maps. In addition, they were able to adapt and learn how to avoid obstacles to reach their target destination. These results demonstrate the effectiveness of our algorithm in training intelligent agents to navigate complex environments in a lifelike simulation. This has important practical implications as it shows that our approach can be used to train agents for real-world navigation tasks without the need for external maps. Overall, our findings highlight the potential of our algorithm as a powerful tool for comprehending and learning complex behaviors in simulated environments.

VI. Discussion

The results of this study demonstrate the effectiveness of using a Deep-RL algorithm with GEA for navigation tasks in a simulated environment for a TurtleBot robot. The experimental setup, implemented in Python and utilizing the PyTorch library, successfully modeled the robot as an agent of the environment

and respected the motion constraints of the TurtleBot. The Adam optimizer was used to train the actor and critic neural networks, with a learning rate of 0.001 and a minibatch size of 256. The exploration rate was set to 0.5 for the actor network and 0.2 for the critic network.

The results show that the agent was able to effectively navigate to the randomly generated target points in the environment, while also avoiding obstacles and boundaries. It was observed that it took on average a little more than 200 episodes for the agent to start learning the task, but after this stage, the reward value increases almost steadily until it reaches close to the maximum saturation value. This suggests that the agent is able to learn the navigation task in a relatively short amount of time and perform well in the environment.

One limitation of this study is that the experiments were conducted in a simulated environment, which may not fully reflect the complexities of a real-world environment. Additionally, the agent's decision-making process was based on a predetermined set of rules, which may not be optimal in all situations. However, the study's results are still a promising step towards the development of intelligent robots that can navigate autonomously in complex environments.

In conclusion, this study has successfully demonstrated the effectiveness of using a Deep-RL algorithm with GEA for navigation tasks in a simulated environment for a TurtleBot robot. The results show that the agent is able to learn the navigation task in a relatively short amount of time and perform well in the environment. Future work can further improve the algorithm's performance by incorporating more realistic environments, and exploring other decision-making methods.

VII. Conclusion

In this paper, a reinforcement learning algorithm for a small robot using generalized advantage estimation (GAE) is presented. The algorithm was implemented in Python and was designed to enable the robot to navigate and interact with its environment to maximize its reward. The robot was designed to operate in a simple 2D environment. The environment consisted of a grid of cells, each of which could be occupied by the robot or contain an obstacle. The robot could move to any of the four adjacent cells in the grid by selecting one of the following actions: up, down, left, or right. At each timestep, the robot received a reward based on its current position and the presence or absence of obstacles in its environment. To enable the robot to learn how to navigate its environment and maximize its reward, a reinforcement learning algorithm using GAE is implemented. The algorithm consisted of two key components: an actor-network and a critic network. The actor network was responsible for predicting the action the robot should take at each timestep based on the current state of the environment. The critic network was responsible for predicting the value of each state, which was used to estimate the expected future reward of each action. A combination of supervised learning and reinforcement learning to train the actor and critic networks is used. Specifically, the actor-network to predict the action the robot should take at each timestep based on the current state of the environment is used. The actor-network was trained using supervised learning, with the input being the current state of the environment and the output being the predicted action. The critic network was trained using reinforcement learning, with the input being the current state of the environment and the output being the predicted value.

The algorithm was run for a series of episodes, resetting the environment and initializing the state at the beginning of each episode. At each timestep, the actor and critic networks predict the action and value, respectively, and the action is performed on the environment. The reward and value are stored in buffers. Once the episode is terminated, the GAE algorithm calculates the payoffs, and the losses for the actor and critic networks are calculated. These losses are backpropagated and used to update the networks. The results of our experiments show that the robot can successfully navigate through the environment and reach the goal state, with the average reward increasing throughout training. It was observed that the losses of actor and critic networks decrease as training progresses, indicating that the networks are learning effectively.

Overall, our implementation of the GAE algorithm for the small robot demonstrates its effectiveness in learning to navigate through an environment and reach a specific goal state. Furthermore, this algorithm can be extended to other reinforcement learning tasks in various environments, such as control and decision-making.

Acknowledgment

This work was supported by the Universidad Distrital Francisco José de Caldas, specifically by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank all the students and researchers of the research group ARMOS for their support in the development of this work.

References

[1] K. P. Divakarla, A. Emadi, S. Razavi, S. Habibi, and F. Yan, "A review of autonomous vehicle technology landscape," *International Journal of Electric and Hybrid Vehicles*, vol. 11, no. 4, p. 320, 2019.

[2] S. Jain, N. J. Ahuja, P. Srikanth, K. V. Bhadane, B. Nagaiah, A. Kumar, and C. Konstantinou, "Blockchain and autonomous vehicles: Recent advances and future directions," *IEEE Access*, vol. 9, no. 2021, pp. 130 264–130 328, 2021.

[3] S. Koul and A. Eydgahi, "The impact of social influence, technophobia, and perceived safety on autonomous vehicle technology adoption," *Periodica Polytechnica Transportation Engineering*, vol. 48, no. 2, pp. 133–142, 2019.

[4] R. B. Issa, M. Das, M. S. Rahman, M. Barua, M. K. Rhaman, K. S. N. Ripon, and M. G. R. Alam, "Double deep q-learning and faster r-CNN-based autonomous vehicle navigation and obstacle avoidance in dynamic environment," *Sensors*, vol. 21, no. 4, p. 1468, 2021.

[5] D. Miculescu and S. Karaman, "Polling-systems-based autonomous vehicle coordination in traffic intersections with no traffic signals," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 680–694, 2020.

[6] W. Lin, X. Ren, J. Hu, Y. He, Z. Li, and M. Tong, "Fast, robust and accurate posture detection algorithm based on kalman filter and SSD for AGV," *Neurocomputing*, vol. 316, no. 2018, pp. 306–312, nov 2018.

[7] X.-B. Jin, T.-L. Su, J.-L. Kong, Y.-T. Bai, B.-B. Miao, and C. Dou, "State-of-the-art mobile intelligence: Enabling robots to move like humans by estimating mobility with artificial intelligence," *Applied Sciences*, vol. 8, no. 3, p. 379, 2018.

[8] H. Min, X. Wu, C. Cheng, and X. Zhao, "Kinematic and dynamic vehicle model-assisted global positioning method for autonomous vehicles with low-cost GPS/camera/in-vehicle sensors," *Sensors*, vol. 19, no. 24, p. 5430, 2019.

[9] C. Fan, Z. Zhou, X. He, Y. Fan, L. Zhang, X. Wu, and X. Hu, "Bio-inspired multisensor navigation system based on the skylight compass and visual place recognition for unmanned aerial vehicles," *IEEE Sensors Journal*, vol. 22, no. 15, pp. 15 419–15 428, 2022.

[10] F. Maurelli, S. Krupiński, X. Xiang, and Y. Petillot, "AUV localisation: a review of passive and active techniques," *International Journal of Intelligent Robotics and Applications*, vol. 6, no. 2, pp. 246–269, 2021.

[11] T. Elmokadem and A. V. Savkin, "Towards fully autonomous UAVs: A survey," *Sensors*, vol. 21, no. 18, p. 6223, 2021.

[12] X. Liu, G. V. Nardari, F. C. Ojeda, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. F. Romero, C. J. Taylor, and V. Kumar, "Large-scale autonomous flight with real-time semantic SLAM under dense forest canopy," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5512–5519, 2022.

[13] E. Pairet, J. D. Hernandez, M. Carreras, Y. Petillot, and M. Lahijanian, "Online mapping and motion planning under uncertainty for safe navigation in unknown environments," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3356–3378, 2022.

[14] I. Mir, F. Gul, S. Mir, M. A. Khan, N. Saeed, L. Abualigah, B. Abuhaija, and A. H. Gandomi, "A survey of trajectory planning techniques for autonomous systems," *Electronics*, vol. 11, no. 18, p. 2801, 2022.

[15] Y. A. Younes and M. Barczyk, "Optimal motion planning in GPS-denied environments using nonlinear model predictive horizon," *Sensors*, vol. 21, no. 16, p. 5547, 2021.

[16] A. Haydari and Y. Yilmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 11–32, 2022.

[17] K.-C. Ma, L. Liu, H. K. Heidarsson, and G. S. Sukhatme, "Data-driven learning and planning for environmental sampling," *Journal of Field Robotics*, vol. 35, no. 5, pp. 643–661, 2017.

[18] F. Martínez, F. Martínez, and E. Jacinto, "Visual identification and similarity measures used for on-line motion planning of autonomous robots in unknown environments," in *Eighth International Conference on Graphic and Image Processing (ICGIP 2016)*, 2016, pp. 321–325.

[19] F. Martínez, C. Hernández, and A. Rendón, "A study on machine learning models for convergence time predictions in reactive navigation strategies," *Contemporary Engineering Sciences*, vol. 10, no. 25, pp. 1223–1232, 2017.

[20] Z. Fei, Y. Wang, J. Wang, K. Liu, B. Huang, and P. Tan, "A new noise network and gradient parallelisation-based asynchronous advantage actor-critic algorithm," *IET Cyber-Systems and Robotics*, vol. 4, no. 3, pp. 175–188, 2022.

[21] B. Peng, M. F. Keskin, B. Kulcsár, and H. Wymeersch, "Connected autonomous vehicles for improving mixed traffic efficiency in unsignalized intersections with deep reinforcement learning," *Communications in Transportation Research*, vol. 1, no. 2021, p. 100017, 2021.

[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2009.

[23] M. L. Littman, "Reinforcement learning improves behaviour from evaluative feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, no. 1707.06347, pp. 1–12, 2017.

[25] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *arXiv*, no. 1707.01495, pp. 1–15, 2017.

[26] G. Li, M. Müller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Oil: Observational imitation learning," *arXiv*, pp. 1–21, 2018.

[27] V. J. Hodge, R. Hawkins, and R. Alexander, "Deep reinforcement learning for drone navigation using sensor data," *Neural Computing and Applications*, vol. 33, no. 6, pp. 2015–2033, 2020.

[28] D. Wang, T. Fan, T. Han, and J. Pan, "A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3098–3105, 2020.

[29] B. Chen, M. Xu, Z. Liu, L. Li, and D. Zhao, "Delay-aware multi-agent reinforcement learning for cooperative and competitive environments," *arXiv*, pp. 1–11, 2020.

[30] X. Chen and P. Chaudhari, "Midas: Multi-agent interaction-aware decision-making with adaptive strategies for urban autonomous navigation," *arXiv*, pp. 1–16, 2020.

[31] L. He, A. Nabil, and B. Song, "Explainable deep reinforcement learning for uav autonomous navigation," *arXiv*, pp. 1–12, 2020.

[32] J. Fang, Q. Sun, Y. Chen, and Y. Tang, "Quadrotor navigation in dynamic environments with deep reinforcement learning," *Assembly Automation*, vol. 41, no. 3, pp. 254–262, 2021.

[33] N. Yan, S. Huang, and C. Kong, "Reinforcement learning-based autonomous navigation and obstacle avoidance for USVs under partially observable conditions," *Mathematical Problems in Engineering*, vol. 2021, no. 2021, pp. 1–13, 2021.

[34] R. B. Grando, J. C. de Jesus, V. A. Kich, A. H. Kolling, N. P. Bortoluzzi, P. M. Pinheiro, A. A. Neto, and P. L. J. Drews-Jr, "Deep reinforcement learning for mapless navigation of a hybrid aerial underwater vehicle with medium transition," *arXiv*, pp. 1–7, 2021.

[35] M. Li, P. Sankaran, M. Kuhl, A. Ganguly, A. Kwasinski, and R. Ptucha, "Simulation analysis of a deep reinforcement learning approach for task selection by autonomous material handling vehicles," in *Winter Simulation Conference (WSC 2018)*, 2018.

[36] F. Martínez, "Turtlebot3 robot operation for navigation applications using ros," *Tekhnê*, vol. 18, no. 2, pp. 19–24, 2021.