Goal-seeking Navigation based on Multi-Agent Reinforcement Learning Approach

Abdul Muizz Abdul Jalil[1] and Mohd Ridzuan Ahmad*[2]

[1,2]Division of Control and Mechatronics Engineering, School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[1]abmuizz@gmail.com, [2]mdridzuan@utm.my

Article history

Received:
12th October 2021

Received in revised form:
18th October 2021

Accepted:
28th October 2021

Published online:
11th November 2021

*Corresponding author
mdridzuan@utm.my

Abstract

Navigation for the robot has numerous applications in industries such as agriculture, couriers, autonomous vehicle, and many more. Navigation seems a simple problem for humans but complex for machines. Robot navigation problems can be made up of mapping, localization, path planning, and motion control. But there are limitations when it comes to the real world. The nature of dynamics in the real world makes it hard to adapt to certain situations since a lot of conventional controllers cannot adapt. This is where machine learning can be used since it can adapt according to the situations it has learned. The closed-loop control systems have a feedback loop, and Reinforcement Learning is most suitable to develop where this paper explores how to develop a controller using deep reinforcement learning. The algorithm used is the Deep Deterministic Policy Gradient algorithm with multi-agents. The simulated robot inside the environment has a simple dynamics system for simplicity's sake for this proposed project, and the algorithm is designed to work with the environment developed by a third-party using OpenAI Gym. The robot is controlled using the neural network or controller network with the same as the multi-input multi-output system. The neural network was designed and also constructed using (Full abbreviation) LSTM with a fully connected neural network. The algorithm and evaluation of the model will be part of future works to achieve the objectives of the proposed project.

Keywords: *Reinforcement Learning, Deep Deterministic Policy Gradient, Multi-Agents, Control System, Navigation.*

1. Introduction

This section discusses the robot navigation of the past and current methods to solve robot navigation and the problem that needs to be solved, such as the complexity of the environment needs better controller using machine learning.

1.1. Background

Robot navigation has many applications in different industries, from agriculture, automotive to courier shipping, and more. Many mobile robots typically need navigation algorithms which are to move from one point to another point in environment space that typically has obstacles, and the robot has to be able to avoid obstacles while navigating to a goal [1]. Other things are the types of environments such as outdoor, semi-outdoor, indoor, and semi-indoor [2]. Navigation can be divided into geometric, topological, semantical, and perception models for

* *Corresponding author. mdridzuan@utm.my*

navigation [3]. Many researchers have proposed various solutions related to navigation problems, such as mapping by scanning the surroundings to produce an occupancy map represented by binary or probabilistic values for robot understanding of obstacles in environment space [4]–[6]. But mapping requires localization of the robot, which is to determine the position of the robot in a known map which obtains by feature extraction using vision or time of flight sensors for scanning the environment [7]. By determining robot localization, mapping the unknown environment is possible, then a path to goal position can be realized. Because the robot, while moving toward goal position, needs to know the environment and currently using simultaneous localization and mapping (SLAM) techniques for this problem which intends to do all related problems in multi-tasking such as mapping the environment while measuring the robot location in the environment [8]. But using SLAM came at a great cost which is computation time [9] which means it needs a high-performance system to be used which is not suitable when designing for certain applications, which is a real-time embedded system since this method deploys a high-level controller and a low-level controller.

Realistic speaking, the real world is dynamic by nature and should be solved in dynamic ways. Although the nature of dynamic for the simulated 2D environment might be difficult but can still be simulated in a static environment. Because limitations of using conventional controllers such as PID which are suitable for a low-level controller and not a high-level controller. Currently, it is solved by combining two different controllers, such as a high-level controller and a low-level controller, where a high-level controller is fed to the low-level controller rather than a controller with the ability to understand high-level of understanding the world surrounding and execute low-level instruction to the actuator. Recently researchers have been using machine learning to solve navigation problems such as supervised learning and reinforcement learning [10]. The suitable type of learning would be reinforcement learning since most of the algorithms are in the context of dynamic programming. But training this system can take longer times, but it can learn in off-policy, which can shorten the learning process since it learned from past random actions. The aims of this work are to identify different types of simulation environments, to develop a controller using a reinforcement learning algorithm, and, lastly, to develop a neural network structure. The scopes of the works are to simulate two dimensions of indoor space or environments, based on wheeled robot motion, and multi-agents of a maximum of 5 depending on environment sizes.

2. Literature Review

This chapter will review the different existent methods that are currently available, including the implementation which is relevant to this work. Firstly, to find suitable algorithms and to develop a controller using RL algorithms. Secondly, to design structures of neural networks using deep learning to meet the design requirements. Next, to determine the environment settings to be used in the deployment after training. And lastly, to evaluate the designed model.

2.1. The Reinforcement Learning Algorithm

The algorithm is chosen based on whether the robot in the environment has inputs spaces in continuous or discrete and outputs, which are the measurement states in continuous or discrete. This work considers robot dynamics and, therefore, will have continuous action (the inputs) and continuous observation state (the outputs) for the environment. There are several candidates such as Deep Q Network (DQN) [11], Asynchronous Advantage Actor-Critic (A3C) [12], Asynchronous Actor-critic (A2C) [13], Model-based approach [14]–[18], Deep Deterministic Policy Gradient (DDPG) [19]. Based on these papers, it is shown that the model-based approach has low adaptation on a dynamic environment and, therefore, will not be considered. Using DQN might work for the project, but it came with limitations since it is the extended version of Q-learning using the neural network as a function approximator of Q-learning, which are continuous actions that need to be discretized actions might not be a good idea. Next are the A2C and A3C since they were somewhat similar. These algorithms can be used on continuous action, and continuous states which involve many environments (using multi-core CPU or synchronous) in A2C and used independent networks on the parallel environment to tune the global network weights parameters where A3C used asynchronous ways means that rather than waiting for all parallel environment to be executed the action, it will execute without waiting for all parallel environment to tune the global network. This is somewhat similar to a centralized system where the main core control all the agents. Here, the agents are referred to as controllers which use deep learning where the robots refer to robots (or plants in the control system) inside the environment, and the agents will control the robots by sending actions such as arbitraries movement or continuous action such as voltage or position (for example closed-loop position controller) to the robots and therefore the agents will act as controller.

2.2. The Neural Network Structure

There are different types of artificial neural networks (ANN), such as multilayer perceptron (MLP), convolution neural network (CNN), recurrent neural network (RNN), and others [20][21]. Many AI researchers suggest using long short-term memory (LSTM), which is under RNN class means it is a type of neural that can store memories and these memories will change based on receiving inputs data [22]. Since humans used memories when to navigate are probably why several researchers suggest using LSTM, but using this network alone might not be a great idea since the structure of inputs and outputs need to be compatible such as the dimensions of action and measurable states of the environment [23]–[26]. Based on previous works, basic layers of the network can be constructed, such as input layer, LSTM layers, and output layers using dense layer. For Q-network, the dense layers are sufficient for this project. DDPG algorithm required 4 networks which are policy network, Q-network, target policy network, and target Q-network. The target network is an exact structure with main networks, but the weights of the network are updated using a different algorithm, as shown in the DDPG algorithm. The network can be constructed to form a reactive controller system [27] which should be the basis of a simple navigation problem.

2.3. The Environment

Choosing the environment for the agents to learn behaviors such as obstacles avoidance (dynamic and static), goal-seeking, and motion behaviors (linear and rotation movement) needs to be considered. Since the developed controller model required evaluation, therefore, needs environments for training while others for testing after training sessions are over. OpenAI Gym is a great tool, and many researchers have been using OpenAI Gym especially for developing RL algorithms for training and evaluation [22]. Several environments are used with the purpose of training and then testing on a few unseen environments to determine performance and reliability. The criteria for choosing must be based on the requirement needed, such as the ability to move based on input action to specific location serve as location point and also came with obstacles such as a walls-like maze. Several environments can be used, but the most suitable is the gym-miniworld [28] developed by a third-party using OpenAI Gym application programming interface (API), and although the design is simple, it can be customized to include others that are lacking, such as robot dynamics. These also have many different environments and can be used in this project to achieve the objectives and also within the scopes. Although the observable states are fully observable to be customized to our needs to works similar to partially observable arrays of ranging sensors.

2.4. The Evaluation of the Model

During the training, it is important to know whether the model is training properly by looking at the increasing cumulative or average reward until it converges. When converge is occurred, and there is no increasing reward, and the agent is not able to reach specifications (not reached accuracy or loss), and this can be solved by increasing the hidden layers of the network. To evaluate the model for navigation is dependent on the success rate to reach the goal location. Quite similar to supervised learning, where the accuracy or loss to predict assigned labels can determine the performance of the model, but for application for navigation, especially for goal-seeking behavior, it can be determined by the total reached goal location with the total attempt. Based on other works [refs.], the accuracy can be reached somewhere 80% to 98%, but 95% accuracy is acceptable since humans can reach around 95% for different tasks, and it should be sufficient for our project purposes.

3. Research Methodology

The methods are chosen based on the literature review. Although DDPG was developed for a single agent but using it for multi-agents can still work as seen in A3C algorithms implementation. This section describes the details of the proposed method, diagram concerning the project, and resources to be used.

3.1. The Proposed Method

Below is the proposed method, which is the DDPG with multi-agents using a centralized global network and several independent networks for each agent. The DDPG algorithms required 4 networks, namely policy or actor-network, critic network, and target networks for actor and critic. The outputs of the actor-network

Table 1: The DDPG with a multi-agent algorithm

DDPG with multi-agents

Initialize the environment
Randomly Initialise critic-network Q($s,\ a|\theta^Q$), actor-network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target networks Q′ and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer, R
For episode=1, M do
Initialize multi-agents exploration
Receive initial observation, s_i
For time=1, T do
Select action (Run policy network) for every agent
Store for every agent $(s_i,\ a_i,\ r_i,\ s_{i+1})$ in R
Sample a random minibatch of N transitions from R
Set $y_i = r_i + \gamma Q'(s_{i+1},\ \mu'(s_{i+1}|\theta^\mu)|\theta^Q)$
Update critic by minimizing the loss, $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i,\ a_i|\theta^Q)\right)^2$
Update policy using sampled policy gradient
$\nabla_{\theta^\mu} \approx \frac{1}{N}\sum_i \nabla_a\, Q(s_i,\ a_i|\theta^Q)\nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$
Update the target networks
$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\,\theta^{Q'}$
$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\,\theta^{\mu'}$
End for
End for

will predict the next action for the robot, as shown in the block diagram. The environment and the agent neural networks need to be initialized first by predefined parameters such as training hyperparameters, neural networks structures, weights parameters, and others. After initializing the parameters, read the initial states and feed them into the model to obtain the action. Execute the action to obtain the next states and also how much the reward to the store in the memory. Then sample from memory to compute the Bellman equation to compute the loss function of Q-network that can adjust the weights using backpropagation. Then the DDPG algorithm can be executed with some modifications for multi-agents. In general, the data to be collected on various agents does not make the distinction on different agents, including its does not cooperative and competitive since each agent will have its goal location. Therefore, the data will be more diverse using a single environment, but many agents and the agents can also learn to perceive other agents as moving obstacles. Table 1 shows the proposed DDPG with a multi-agent algorithm.

The proposed robot-environment interface is shown in Figure 1 below to illustrate how different components operate as a single system to train or evaluate. The agent itself is a MIMO controller with states as input and action as output. For simplicity, the outputs model actions are linear movement and rotation movement such as move forward or backward, M_{FB} and rotate right or left, R_{RL}. The inputs are the positions in *x*-axis coordinate, *y*-axis coordinate, and *z*-axis orientation (P_X, P_Y, O_z). Other inputs are robot velocities such as linear velocity and angular velocity (V, Ω). The last pieces of feedback inputs are the array of obstacles distances from robot positions $(D_1, D_2, D_3, D_4, D_5)$. Although these are partially observing states, it should be enough to navigate to the goal while avoiding the obstacles similar to a reactive system.

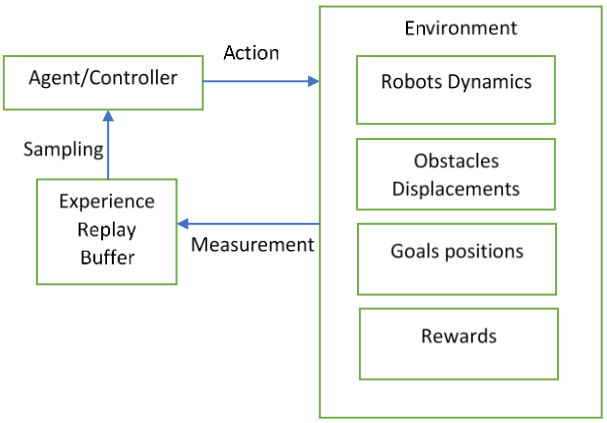

Figure 1. Block diagram of the robot-environment interface

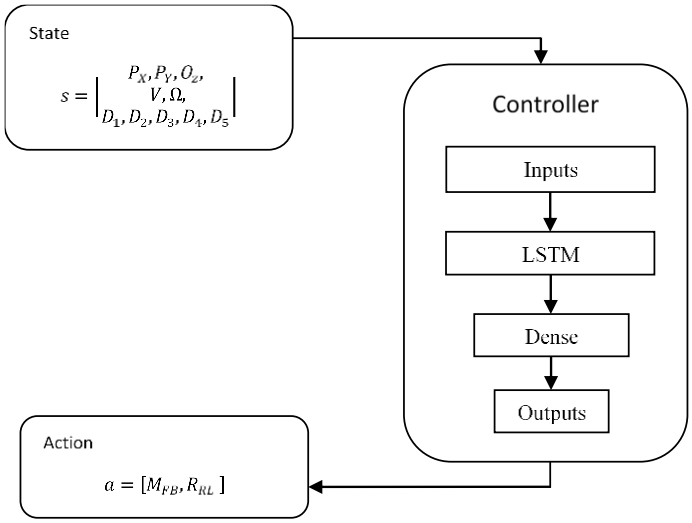

Figure 2. The Model Architecture

The structure of the network for the actor is shown in Figure 2 below. The LSTM cells which form hidden layers will have the same size as the input layer. More LSTM and dense layers will add if the performance of behavior does not meet the requirements. The dense layers can serve to process possible actions, which then feed into the environment. This structure takes inspiration from the feedback system and reactive system, which does not involve fully observed states.

3.2. Tools and Platform

The tools that were used are Anaconda [29] which is one of python distribution mainly used for programming the algorithm in python. Other tools such as OpenAI gym-miniworld for the environment that have different mazes and sizes pick to train or evaluate the model. The model itself used TensorFlow python API and was integrated as a part of the whole algorithm.

4. Preliminary Results

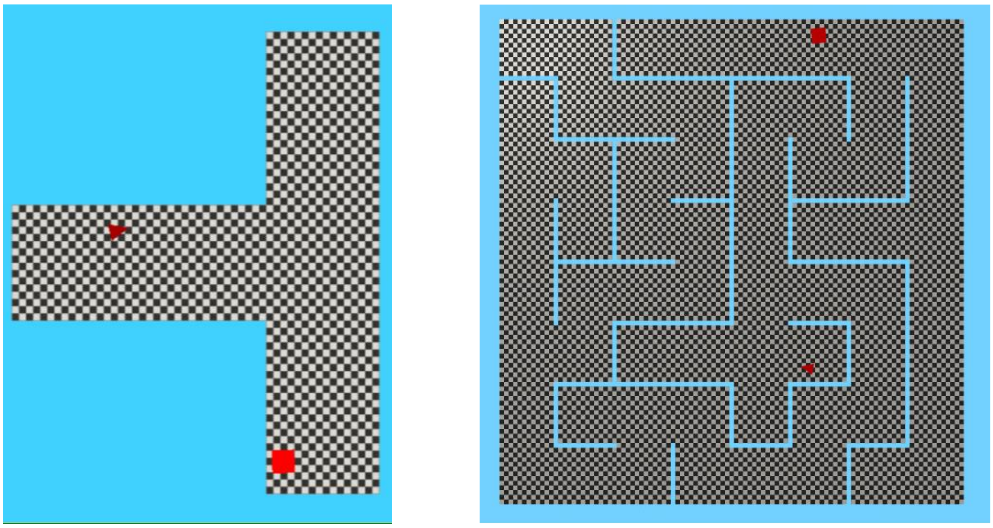

Figure 3. Training (left) and Testing (right) Environments

The results were carried out according to the objectives concerning the environment for robot navigation which are suitable such as the complexity of the mazes and sizes. The neural network architecture can also be easily designed according to the shape we want. The results are shown below in terms of environment and neural networks that can integrate into the algorithm.

4.1. The Environments

The environment to access reliability such as robot movement toward the goal. When it reaches the goal, it will reset and restart the new robot position and goal location. The dynamics of the robot are simple such as moving forward, backward, and rotating right, left. The fully observable top-down/overhead view is available to be used as measurable states but only needs partially observable for the proposed project. Other states were also available such as position in x-coordinate, y-coordinate, and z-coordinate. The angle or the orientation is also included, but other states that are lacking are robot velocities and obstacles displacement, but these can measure from image top view of the environment. These environments, as shown in Figure 3, should have enough space to deploy multi-agents.

4.2. The Agents Models

The structure ANN such as LSTM and MLP or dense layers as the proposed method, there are two ways to do and below is achieved using functional API. These will form a basis for the controller as more layers will be added for future works. Figures 4 and 5 are the model summary to be used for the proposed project. The input layers are the signal from measured states and feedforward to the LSTM layer. The LSTM will have the same cells as observed states for simplicity and feedforward to dense layer. The dense layer also has the same neuron dimension as LSTM. Then lastly, to reduce the last dense layer to two for the actions.

```
Model: "actor network"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 10, 1)]           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 10)                480
_________________________________________________________________
dense_20 (Dense)             (None, 10)                110
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 22
_________________________________________________________________
tf.math.multiply_4 (TFOpLamb (None, 2)                 0
=================================================================
Total params: 612
Trainable params: 612
Non-trainable params: 0
_________________________________________________________________
```

Figure 4. The Policy Network of The Model

```
Model: "critic network"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_20 (InputLayer)           [(None, 10)]         0
__________________________________________________________________________________________________
input_21 (InputLayer)           [(None, 2)]          0
__________________________________________________________________________________________________
dense_22 (Dense)                (None, 32)           352         input_20[0][0]
__________________________________________________________________________________________________
dense_23 (Dense)                (None, 32)           96          input_21[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 64)           0           dense_22[0][0]
                                                                 dense_23[0][0]
__________________________________________________________________________________________________
dense_24 (Dense)                (None, 128)          8320        concatenate_3[0][0]
__________________________________________________________________________________________________
dense_25 (Dense)                (None, 1)            129         dense_24[0][0]
==================================================================================================
Total params: 8,897
Trainable params: 8,897
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5. The Actor Network of The Model

The critic network, which is Q-network shown in Figure 5, will have a much bigger size than the actor-network or controller network. The output from the actor and the critic network is important for the DDPG algorithm. There are two inputs which are states and actions, which then feed into separate dense layers. These two dense layers are then fed into concatenate layer, which links them together as a single dimension output to feed into the dense layer and reduce the output dimension to one. This out the value called Q-value, which is related to Q-learning. The target networks will have the same structure as the actor and critic networks.

4.3. The Experience Replay Buffer

```
class Buffer:
    def __init__(self, buffer_capacity=100000, batch_size=64):
        # Number of "experiences" to store at max
        self.buffer_capacity = buffer_capacity
        # Num of tuples to train on.
        self.batch_size = batch_size

        # Its tells us num of times record() was called.
        self.buffer_counter = 0

        # Instead of list of tuples as the exp.replay concept go
        # We use different np.arrays for each tuple element
        self.state_buffer = np.zeros((self.buffer_capacity, num_states))
        self.action_buffer = np.zeros((self.buffer_capacity, num_actions))
        self.reward_buffer = np.zeros((self.buffer_capacity, 1))
        self.next_state_buffer = np.zeros((self.buffer_capacity, num_states))

    # Takes (s,a,r,s') obervation tuple as input
    def record(self, obs_tuple):
        # Set index to zero if buffer_capacity is exceeded,
        # replacing old records
        index = self.buffer_counter % self.buffer_capacity

        self.state_buffer[index] = obs_tuple[0]
        self.action_buffer[index] = obs_tuple[1]
        self.reward_buffer[index] = obs_tuple[2]
        self.next_state_buffer[index] = obs_tuple[3]

        self.buffer_counter += 1
```

Figure 6. The Experience Replay Buffer

Every action simulated in the environment is stored in a replay buffer, as shown in Figure 6. In terms of progress, this part is finished and linked together with the neural network model and the environment. The function of this class will simplify the programming when linked to the main algorithm. The buffer store known as Markov process decision and the minibatch from this data will be used in the policy gradient equation to find the change of weights to be used for backpropagation.

4.4. Discussion

The controller using ANN has been successfully constructed using TensorFlow and can be designed as need it. Since the model uses functional API rather than sequential API, it was easier to construct in the code, especially for the critic network, since it is two networks of two inputs of actions and states and merge them as one-dimension array values.

The environment overall works as expected that the robot can move inside the environment space using a simple command such as moving forward, backward, rotating right, left. The lacking feature in the environment will be part of future works. Every movement or time step is observed by calling function in OpenAI Gym. The outputs values of the states will be used to store in replay buffer. The reward function is simple, but in order to develop certain behaviors of the robot, the reward function will be formulated in future works. The environment will operate in top view, which gives a 2D view that can be regarded as a 2D environment.

The main algorithm can be executed in future works by linking them together since most of the individual part has almost been completed. This can be used as part of future works to be planned the experiment. There is currently a lack of an environment that can be used to simulate different types of mobile robots, such as a multi-wheeled robot, which is the reason why a lot of works need to be done on a simulated wheeled robot as a part of the project.

5. Conclusions and Future Works

The outcomes of this research are to develop the controller, the environment, the DDPG algorithm with multi-agents using various API according to the proposed method that has been successfully carried out. To design a controller in conventional ways such as PID controller has limitations, but those limitations can be overcome using an intelligent system such as deep learning and will play a crucial role in future applications in dealing with a complex system to the controller.

For future works, the proposed algorithm will be linked with the environment to experiment. After that, train the developed model and observe the robot behavior. If robot behaviors do not react as expected or do not meet the requirements, then some adjustments are needed, such as increasing the size of networks or modifying the reward function. After all the conditions are good, then the evaluation will be carried out in an unseen environment to be able to determine the performance, such as the accuracy to reach the goal location.

Acknowledgments

I would like to thank everyone who was involved in my academic terms to complete this work. I also want to thank my parent, who supported me in pursuing my passion in the field of robotics. Lastly, my supervisor for the advice and guidance throughout the research process. Thank you for all your support.

References

[1] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering*, vol. 6, no. 1. 2019.

[2] J. Yan, A. A. Diakité, and S. Zlatanova, "A generic space definition framework to support seamless indoor/outdoor navigation systems," *Trans. GIS*, vol. 23, no. 6, pp. 1273–1295, 2019.

[3] R. Barber, J. Crespo, C. Gómez, A. C. Hernámdez, and M. Galli, "Mobile Robot Navigation in Indoor Environments: Geometric, Topological, and Semantic Navigation," in *Applications of Mobile Robots*, IntechOpen, 2019.

[4] S. Sattaratnamai, N. Niparnan, and A. Sudsang, "Urban Navigation System with Multiple Sub-Maps and Multiple Sub-Navigators," *IEEE Access*, vol. 8, pp. 99974–99989, 2020.

[5] A. A. Ravankar, A. Ravankar, T. Emaru, and Y. Kobayashi, "A hybrid topological mapping and navigation method for large area robot mapping," *2017 56th Annu. Conf. Soc. Instrum. Control Eng. Japan, SICE 2017*, vol. 2017-Novem, pp. 1104–1107, 2017.

[6] D. De Gregorio and L. Di Stefano, "SkiMap: An efficient mapping framework for robot navigation," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017, pp. 2569–2576.

[7] X. Chen, S. Wang, B. Zhang, and L. Luo, "Multi-feature fusion tree trunk detection and orchard mobile robot localization using camera/ultrasonic sensors," *Comput. Electron. Agric.*, vol. 147, pp. 91–108, 2018.

[8] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving," *IEEE Trans. Intell. Veh.*, vol. 2, no. 3, pp. 194–220, 2017.

[9] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM problem: A survey," *Front. Artif. Intell. Appl.*, vol. 184, no. 1, pp. 363–371, 2008.

[10] S. Aggarwal, K. Sharma, and M. Priyadarshini, "Robot navigation: Review of techniques and research challenges," in *Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, 2016, pp. 3660–3665.

[11] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning."

[12] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 4, pp. 2850–2869, Feb. 2016.

[13] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 5280–5289, Aug. 2017.

[14] M. Gheisarnejad and M. H. Khooban, "An Intelligent Non-Integer PID Controller-Based Deep Reinforcement Learning: Implementation and Experimental Results," *IEEE Trans. Ind. Electron.*, vol. 68, no. 4, pp. 3609–3618, 2021.

[15] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-Level Control of a Quadrotor with Deep Model-Based Reinforcement Learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4224–4230, 2019.

[16] T. G. Thuruthel, E. Falotico, S. S. Sant'anna, F. Renda, C. Laschi, and T. George Thuruthel, "Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators Soft Multi-body Dynamics View project Model Based Reinforcement Learning for Closed Loop Dynamic Control of Soft Robotic Manipulators," *Ieeexplore.Ieee.Org*.

[17] N. O. Lambert, C. B. Schindler, D. S. Drew, and K. S. J. Pister, "Nonholonomic Yaw Control of an Underactuated Flying Robot with Model-Based Reinforcement Learning," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 455–461, 2021.

[18] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based Reinforcement Learning: A Survey."

[19] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[20] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, Nov. 2018.

[21] A.-N. Sharkawy, "Principle of Neural Network and Its Main Types: Review," *J. Adv. Appl. Comput. Math.*, vol. 7, no. 1, pp. 8–19, Aug. 2020.

[22] K. Smagulova and A. P. James, "A survey on LSTM memristive neural network architectures and applications," *Eur. Phys. J. Spec. Top. 2019 22810*, vol. 228, no. 10, pp. 2313–2324, Oct. 2019.

[23] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. Van De Panne, "Feedback Control for Cassie with Deep Reinforcement Learning," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 1241–1246, 2018.

[24] W. Zhang, N. Liu, and Y. Zhang, "Learn to Navigate Maplessly with Varied LiDAR Configurations: A Support Point-Based Approach," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1918–1925, 2021.

[25] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, pp. 31–36, 2017.

[26] S. Bohez, T. Verbelen, E. De Coninck, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "Sensor fusion for robot control through deep reinforcement learning," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, no. 1, pp. 2365–2370, 2017.

[27] A. Pore and G. Aragon-Camarasa, "On Simple Reactive Neural Networks for Behaviour-Based Reinforcement Learning," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 9781728173, no. January, pp. 7477–7483, 2020.

[28] M. Chevalier-Boisvert, "gym-miniworld environment for OpenAI Gym," *GitHub*, 2018. [Online]. Available: https://github.com/maximecb/gym-miniworld. [Accessed: 30-Jun-2021].

[29] "Anaconda | Individual Edition." [Online]. Available: https://www.anaconda.com/products/individual. [Accessed: 11-Jun-2021].