Continuous Control of Mobile Robot Navigation

Kshitij Sharma
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
ksharma@wpi.edu

Kunal Nandanwar
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
kgnandanwar@wpi.edu

Uthiralakshmi Sivaraman
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
usivaraman@wpi.edu

***Abstract*—We describe a learning-based mapless motion planner that uses the objective location in reference to the mobile robot coordinate frame, the sparse 10-dimensional range results, and continuous steering commands as inputs and outputs, respectively. The obstacle map of the navigation environment is a key component of traditional motion planners for mobile ground robots equipped with laser range sensors, where both the extremely accurate laser sensor and the environment's effort to construct the obstacle map are crucial.**

Using a variety of deep reinforcement learning approaches, we show that a mapless motion planner can be trained from beginning to end without the use of manually made features or previous examples. Then, we compare the outcomes. This article investigates two deep reinforcement learning methods for mobile robot navigation and compares the Soft Actor-Critic (SAC) approach with the Deep Deterministic Policy Gradients (DDPG) algorithm in the same environment. To assist a robot in locating a target in an environment, both networks use the preceding linear and rotational velocity, relative location and angle of the mobile robot to the target, and 10 laser range data. Only a positive reward will be given to the agent when it reaches the goal, and a negative reward will be given if it collides with any other objects.

The recommended design was successfully applied in two simulated environments, and the results were utilized as a benchmark to compare the two techniques. It was discovered that the SAC algorithm outperforms the DDPG method when used for mobile robot navigation.

***Index Terms*—DDPG, Deep-RL, Navigation for mobile robots, SAC**

I. Introduction

Mobile robots have been increasingly used for many applications, from navigating through complex terrains (Fig. 1) to day-day floor cleaning (Fig. 2). Planning the motion of mobile robots in an unknown environment has been an interesting problem statement. Specifically, autonomous mobile robot navigation is a field of research that combines so many aspects of robotics: Path planning, state estimation (localization, mapping and world representation), prediction, motion planning and control. Traditional motion planners using simultaneous localization and mapping (SLAM) for state estimation are highly dependent on highly precise laser sensors and prior obstacle maps of the environment. The goal is moving towards a motion planner to navigate the non-holonomic mobile robot, collision-free, to the desired positions without requiring any obstacle map of the environment. Machine learning-based approaches have been increasingly used to tackle motion planning problems. Specifically, the reinforcement approach helps in learning the motion of the agent through the evaluation of cumulative reward. An extrapolation of reinforcement learning to a deep reinforcement learning approach facilitates learning from huge data generated from sensor findings for all kinds of environments.

Fig. 1. Robot Navigating through complex terrain [30]

Fig. 2. Floor cleaning robot [29]

Research on challenges involving the control of discrete systems [8] [9], continuous systems [10] [11], and more recently robotics [12] [13] has benefited greatly from Deep Reinforcement Learning (Deep-RL). Deep reinforcement learning has been used in a few robotics applications, such as a manipulation task created in a completely visible and stable environment [15]. Applications for mobile robots deal with impediments and physical environment interactions that complicate the problem at hand.

DeepRL approaches frequently discretized the activities to make an issue easier to solve and to stimulate new applications [14] [16]. Recent research also examines continuous control operations in the movement of mobile robots, with some intriguing outcomes. In broader sense, there are distinct types of reinforcement learning algorithms, value function based, policy based and combination of both. Policy based functions gives preferences to actions whereas value based functions

optimises the value. For our use case, we take Policy or combination approaches since there is more emphasise on actions. Among the policy gradient approaches, DDPG algorithm can adapt and perform well in simulated scenarios and is increasingly preferred for continuous action based planning . To contrast the SAC algorithm with the DDPG method, we employ a comparable issue using the SAC algorithm while making various changes to the network design and reward function. In order to employ two new environments on Gazebo, a fictitious mobile robot was installed there. The mobile robot model utilized in our work's simulation was the Turtlebot3 Burger. Both the algorithms are similar in the sense that both the algorithms emphasise on learning faster with lesser data and are well suited for learning stochastic policy.

The purpose of this study is to demonstrate and evaluate Deep-RL network performance in challenges requiring the navigation of a mobile robot that starts in one location and moves to a target location within an environment. The DDPG and SAC algorithms are the Deep-RL methods that are being contrasted. To the best of our knowledge, this is one of the first articles to apply the SAC algorithm to mobile robots. Two alternative algorithms were used to investigate the navigation job of a mobile robot. These algorithms feature a generic actor network design with 14 inputs and 2 outputs. The 14 inputs are organized according to the 10 laser sensor readings, the prior linear and angular velocity, the distance from the target and the angle at which the mobile robot is facing it. The linear and rotational velocities are the actor network's outputs. These outputs are in charge of sending the robot's target-position-reaching control signals. It is anticipated that the intelligent agent's choice won't result in any item colliding with it on the way to the desired ultimate location.

There are seven sections to this report. A quick overview of the Deep-RL techniques for mobile robot navigation is provided in Section I. The studies of previous researchers in the field that served as inspiration for this one are described in Section II. The theoretical underpinnings of the algorithms employed in the experiments are presented in Section III. The tools, software, and environments used in this work are described in Section V. The methods used to teach the agent how to reach a target are illustrated in Section IV using the network topology and reward function. The methods employed and the outcomes attained in the Gazebo contexts are compared in Section VI. Finally, Section VII and Section VIII address the key outcomes and future work.

II. Related Work

While the majority of reinforcement learning algorithms use incentives that are predefined for the environment and take advantage of learned behaviours, some robotic navigation challenges call for an environment exploration map. Zhelo et al. in [24] devised a system of intrinsic incentives connected with a so-called "curiosity," rewarding the discovery of novel states, in order to encourage the model's exploration of the map. This approach promotes reconnaissance of the map, which helps Deep-RL agents perform better in terms of generalization.

Only tasks with distinct actions can use the DQN. Lillicrap et al. [10] suggested a deep deterministic policy gradient to extend it to continuous control (DDPG). DDPG served as the foundation for future methods including the soft-actor critic (SAC) algorithm as well as DeepRL applications in mobile robot navigation [21]. In [28], Haarnoja et al. used the SAC algorithm to solve issues involving locomotion and object handling as examples of how it may be used in robotics. An overview of the uses of reinforcement learning in robotics was supplied to us by Kober et al. [27], illuminating a route to the use of Deep-RL in robotics.

In an application that needed the robot to be able to explore an area, Tai et al. [25] introduced Deep-RL to robotic navigation, utilizing the DQN method with raw depth photos as input. They then contrasted the DQN technique with supervised learning and reinforcement learning. Tai et al. [16] tried an asynchronous Deep-RL algorithm with continuous control to develop a mapless motion planner in a subsequent study, and they were successful in navigating.

Another model was put up by Zhu et al. [14] to be used with Deep-RL to perform the task of steering a mobile robot. The model developed produced an action in a 3D environment from the input of the target picture and the present observation of states.

When there are humans involved, especially pedestrians, it is quite difficult to teach a car navigation system the standard social conventions. A DeepRL model that can quantify what to do and not do on the precise mechanism of human navigation was developed by Chen et al. [26]. They were successful in creating a time-effective navigation policy that adheres to accepted social standards and permits the management of a mobile robot in an area with plenty of foot traffic.

This research focuses on creating a mapless motion planner based on low-dimensional range measurements, following the same methodology as Tai et al. in [25] and our earlier work [23]. However, Deep-RL uses both deterministic and stochastic methods to handle the navigation challenges faced by mobile robots. We choose to use the DDPG algorithm in the deterministic method. We used a noise signal mixed with the algorithm's output since the agent must traverse the map in order to employ a deterministic approach. Similar to the DDPG strategy used in [25], but with an asynchronous approach that parallelizes the process among several robots that pool their policy changes simultaneously. It was chosen not to include the asynchronous process in our study since we only want to emphasize the deterministic and stochastic systems. It was decided to employ the SAC method, which does not require a noise signal, for the stochastic approach. These two methods are contrasted in order to determine which one performs better for the resolution of our problem in terms of the reward function and the successful path to the targets on the settings suggested.

III. Technical Background

Deep reinforcement learning's complexity may be reduced to the control of an agent in a setting that aims to optimize its reward function. On a lot of Atari video games, the deep Q-network (DQN) method developed in [8] [17] worked well. With just a raw pixel picture input to predict the agent's activity, our system was able to perform at a level comparable to a human. Despite these successes, the DQN could only address the complicated observation spaces utilizing discrete action spaces. However, continuous action spaces must be used while handling a variety of robotics activities that require control. So when starting with continuous domain jobs, other methods must be used.

A. Deep Deterministic Policy Gradient

In a continuous set of action policies, the deep deterministic policy gradient (DDPG) algorithm concurrently learns a Q-function and a policy using an actor-critic technique [10]. The Bellman equation and off-policy data are used to learn the Q-function, which is then used to learn the policy. Two neural networks are used in this algorithm: one represents the actor network, which is in charge of learning the policy, and the other represents the critic network, which roughly approximates the Q function. The two neural networks provide a temporal-difference error signal for each time step and are in charge of action prediction for the present observation state. The actor network's input is the observation made in the current state, and its output is a continuous action space value determined by the policy. The Q-function estimates the present condition and the action indicated by the actor for the critic.

The agent's exploration of its surroundings poses one of the major difficulties in learning in continuous action space. Because the DDPG is a deterministic approach, an exploration policy μ' must be developed to allow the agent to explore the environment effectively.

This is acquired by adding samples from a noise process N to the actor-network policy, defined as:

$$\mu' = \mu(s_t) + N \tag{1}$$

where N is a noise that has been chosen to fit the workplace. The most popular method for producing temporally correlated exploration efficiency in physical control issues was the Ornstein-Uhlenbeck process [18], although more recent findings indicate that uncorrelated, mean-zero Gaussian noise can function flawlessly [19].

B. Soft Actor-Critic

The Soft Actor-Critic (SAC), which is also known as a stochastic actor-critic, is a technique that relies on approximation functions that may learn continuous action space rules [21]. Therefore, we must construct a useful approximation to a soft policy iteration over a large continuous domain. The soft policy iteration is an algorithm that alternates between evaluating and improving policies while learning the best maximum entropy policies. Using the highest entropy, the policy assessment phase seeks to identify the precise value function for our present policy. The policy distribution is updated for the policy enhancement step such that it conforms to the exponential distribution for the current Q-function. In order to estimate a policy for the actor-network, the SAC method uses a neural network as a function. A value network is also used to approximate the state's value, and a critic network is used to approximate the Q-value. These three networks produce a temporal-difference error signal for each time step as well as the action prediction for the current state.

Additionally, the SAC sets a goal to maximize the benefits of the system while also attempting to increase the entropy of the policy. The term "entropy" describes how erratic a variable might be. The method features zero entropy, which encourages the agent to explore if a random variable still assumes a single value.

It is often advised to employ a replay memory that can store the experiences provided by the agent throughout the training session in order to train and evaluate the policy function arising from the actor-network, the value function, and the Q function arising from the critic network [9]. Due to the large number of temporally associated simulated trajectories, which might result in the accumulation of a significant number of variances when approximating the more accurate Q-function to the issue, it is required to store the experiences. The states, behaviours, incentives, and novel states that an agent experimented with in the earlier episodes are saved into a memory buffer and designated as experiences. The experiences from earlier episodes are randomly picked for the algorithm's learning after a sufficient number of events have been stored in memory. In the training phase, the temporal correlations between various episodes can be stopped using this replay memory strategy. The performance of the off-policy Deep-RL algorithms has significantly improved, despite having a short memory. No matter how much Deep-RL can improve application capabilities, using replay memory can slow down an agent's learning process.

The Q-learning methods employ target networks in addition to replay memory [20]. Since reducing the loss brings the Q-function closer to its aim, the expression refers to it as the target. The aim, however, is dependent on the exact same training settings θ. The primary network's weights are represented by θ. The minimizing loss becomes unstable as a result. Therefore, the target network—an additional network with a time delay—was used as the solution. The target network's weight update is defined as:

$$\theta' \leftarrow \tau\theta + (1-\theta)\theta' \tag{2}$$

with $\tau << 1$. Employing a Poliak-averaging [22] to update its parameters, which has an impact on the learning stability of the Deep-RL algorithms, indicates that the target network is just a clone of the main network.

IV. Methodology

A mobile robot control system that can create its own motion plan on an environment without any prior information

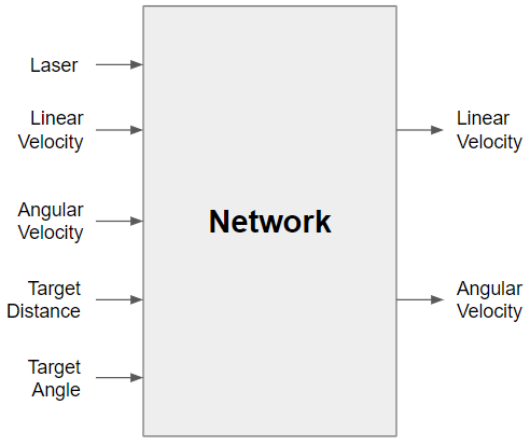

Fig. 3. Network inputs and outputs

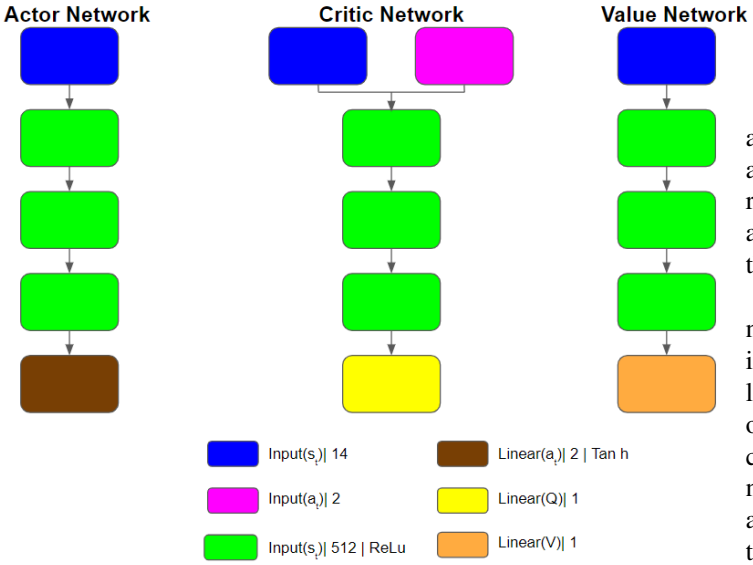

Fig. 4. SAC network structure model

is suggested in this study. Its movement equation is expressed as follows:

$$v_t = f(x_t, p_t, v_{t-1}) \tag{3}$$

where x_t is the raw information from the sensor readings, p_t is the relative location in angles and distance to the goal in the map, and v_{t-1} is the velocity applied in a previous step to the mobile robot. These variables, previously defined are the current state st that the agent will use for training. When s_t passes through a function, in this case, the network, the expected result is an action for the current state.

A. Network Structure

As seen in Fig. 3, the neural network comprises 14 inputs, of which 10 are laser sensor readings, 2 are prior angular and linear velocity messages provided to the agent, and the last 2

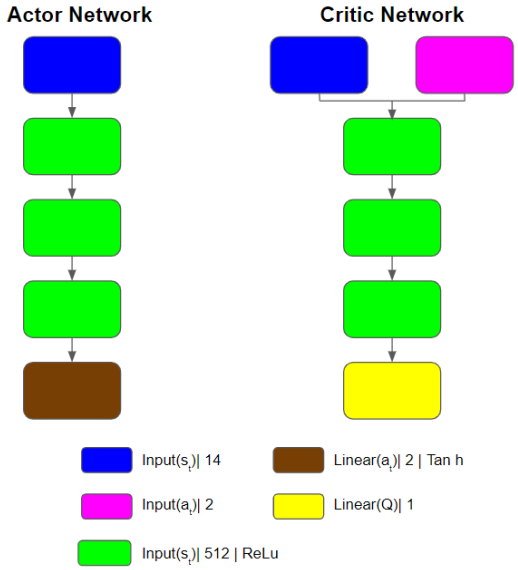

Fig. 5. DDPG network structure model

are the relative location of the robot, together with its angle and scalar distance from the target. In relation to the moving robot, the laser readings are collected at an angle between 90 and 90 degrees. The robot, as shown in Fig. 4, will receive the linear and angular velocities from the network.

Fig. 4 depicts the SAC's network structure. The actor-input network's is the status of the mobile robot at the moment in its surroundings. Three fully connected neural network layers with 512 nodes are added after this input before the output layer. The structural network created in [10] was carefully followed in determining the number of layers and nodes. The angular and linear velocities transmitted to the agent are generated by the output layer. However, due to the hyperbolic tangent function (tanh) being utilized as the activation function, the value of the actions is between (1, 1) instead. Thus, the angular velocity and the linear velocity are restricted to intervals of 2 to 2 rad/s and 0 to 0.22 m/s before the Turtlebot3.

The critic network provides the Q-value for the agent's present condition and activity. Furthermore, the value of the present state is anticipated in the value network. The state inputs were processed by the two networks using just three fully connected neural network layers. A linear activation function is used to activate the Q-value and the current state value:

$$y = kx + b \tag{4}$$

where x is the last layer's input, y is the Q-value, which is the value predicted from the current state, k are the training weights, and b is the last layer's bias.

In order to compare the two networks utilized in this study, we adopted the identical structure for the DDPG network as in [23] with a few minor alterations. The network topology of the DDPG is displayed in Fig. 5. The actor-network and

the critic network adhere to the SAC network's organizational framework.

B. Reward Function

It is possible to simulate a controlled mobile robot carrying out a navigation job when the environment is previously set. The Deep-RL network's reward and penalty mechanism must first be established. The agent is provided rewards and punishments, which are just numerical representations of a function model built on actual data and developed during the problem-solving process. The network will thus do a feedforward and backpropagation process to learn the hyperparameters.

In this paper, we have 2 conditions for the reward function. The reward function used for this paper was the following:

$$r(s_t, a_t) = \begin{cases} r_{arrive} & \text{if } d_t < c_o \\ r_{distance} & 100 * distance + 3 * angle \\ r_{collide} & \text{if } min_x < c_o \end{cases} \quad (5)$$

Reward	Value
Arrive at Goal	100
Collision	-10

TABLE I
REWARD FUNCTION 1

Reward	Value
Arrive at Goal	100
Collision	-500

TABLE II
REWARD FUNCTION 2

A positive reward (r_{arrive}) of 100.0 is given if the robot reaches the goal while checking a threshold c_d (10 cm), while a negative reward $(r_{collide})$ of -10 is given if the robot runs into an obstruction for the reward function 1. A better reward function is created where the positive reward of (r_{arrive}) of the same value as 100.0 is given but the robot is penalized heavily by a negative reward $(r_{collide})$ of -500.0 as well as to improve upon the distance and the maneuverability a distance reward and an angle reward is added in the reward function 2. The distance reward and angle rewards are computed using the following equations:

$$\text{Distance} = \text{Past Distance} - \text{Current Distance} \quad (6)$$

$$\text{Angle reward} = 180 - \text{Heading} \quad (7)$$

The past distance and the current distance are the distances between the goal and the position of the turtlebot at time step $t-1$ and t respectively. Here heading is the current orientation of the turtlebot.

The training episode can finish under any circumstance. We chose to only offer incentives when the robot collides with and reaches the targets. This choice was made after a careful study revealed that using other sorts of incentives would lead the agent to perceive the map incorrectly and would lead us to maximize the agent's benefits while simultaneously achieving our objectives.

V. Environment Setup

The equipment, programs, and settings used in this work are detailed here. The simulations were done using Gazebo as the simulator and ROS as the simulation engine. The algorithms employed the PyTorch package to build the neural networks and Python as the programming language. The Turtlebot 3 Burger robotic platform has been chosen.

A. PyTorch

PyTorch is an open-source machine learning library applied here to the creation of deep neural networks. It provides two high-level features: tensor computation with acceleration via a graphical processing unit and a tape-based automatic differentiation system. It is well appreciated for its ease of use and simplicity, it incorporates concepts from Python, such as classes, structures and conditional loops. The commercial applications of Pytorch are growing rapidly, with companies such as Tesla, Facebook, Uber etc. The academic field is already widely employed in research, in the fields such as natural language processing, image processing, object recognition etc.

B. ROS and Gazebo

An intermediate that controls the connections between software and hardware in robots is the open-source robot operating system (ROS). Although ROS is not an operational system, it does offer several of the basic services found in operational systems, such as package management, device low-level control, hardware abstraction, and messaging between processes. The processes that ROS runs are represented using a graph architecture, where the processes are nodes and the communications they exchange are termed topics. The primary client libraries are available for C++ and Python.

Without simulation, it is extremely difficult to develop a robotics application, hence navigation tasks simulation is a crucial element in the roboticist's toolkit. A good simulator should have the following qualities: the ability to develop new robots and environments, test algorithms, gather data and apply artificial intelligence models easily and simultaneously with realistic configurations. All of the aforementioned features are feasible on Gazebo and have the benefit of a vibrant community. It comes installed with fully functional associations alongside ROS.

C. Turtlebot

Robot Turtlebot belongs to the ROS standard platform. Fig. 6 shows the Turtlebot, which is a programmable mobile robot that may be used in research, teaching, pastime, and product prototyping. In this case, the robot is simply used for

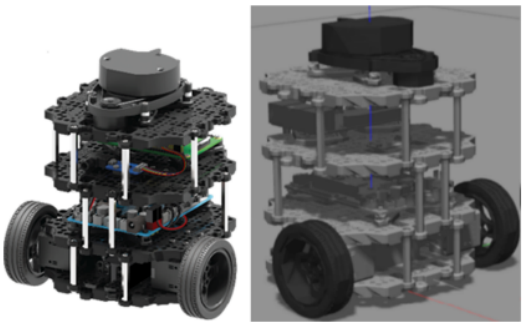

Fig. 6. Real and Simulated Turtlebot3 version Burger

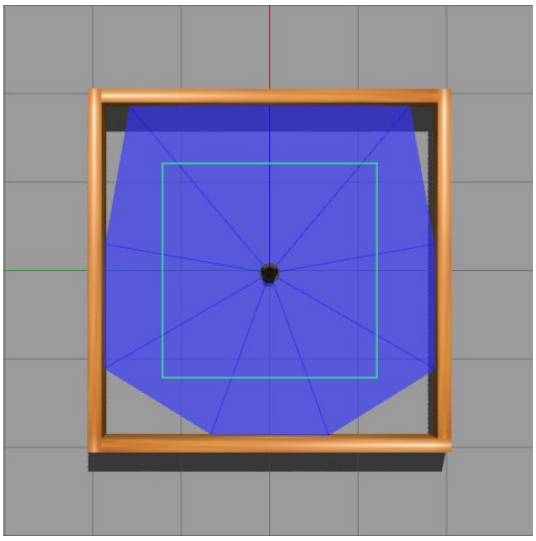

Fig. 7. Training environments used on Gazebo simulation: First environment

simulation, and the only modifications made to the packages were made to the LiDAR sensor, where the output was lowered to 10 and distributed evenly across 180 degrees with a range of 12 centimetres to 3.5 meters. This alteration was made to lessen the number of inputs into the networks.

D. Environment

We choose to run the simulations in two different settings. Fig. 7 depicts the initial environment, which is a region with no obstacles. With a symmetric map that offers the agent two alternative routes with the same cost, still there is a challenge for the agent to not collide with the walls around. In environment 1, we get a 360 degree laser map and the agent is free to explore mutliple pathways for reaching the same goal.

The second environment, represented by the asymmetric situation in Fig. 8, enables the algorithms to be tested under new circumstance with cylindrical obstacles introduced. It implies that this environment is more complicated, necessitating the development of a better collision avoidance approach by the intelligent agent. The objectives are dispersed throughout the settings and set up such that a continuous path is possible. These positions were selected experimentally based on the size of the environment's area; the first contains more positions

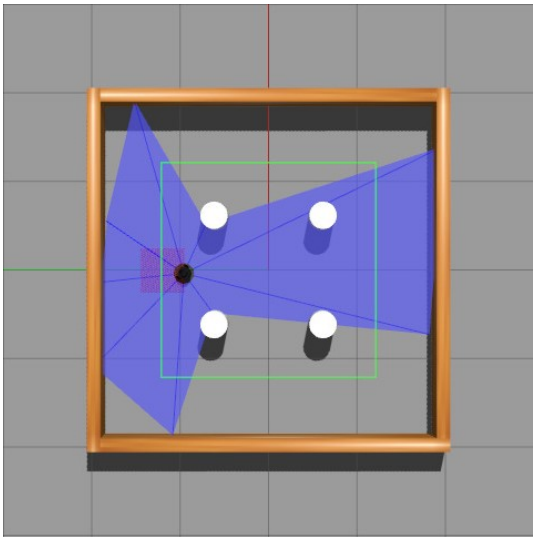

Fig. 8. Training environments used on Gazebo simulation: Second environment

than the second since it has a bigger area, with the goal of making the robot traverse the whole environment, whereas in the second environment, the agent is constrained to choose the path around the obstacles. The agent has to avoid collisions with both the obstacles as well as the walls.

E. Hyperparameters

The hyper parameters in any neural network structure are the top parameters that control the learning process of the weights that the model ends up learning. The hyperparameters utilised in these two network structures are as follows:

- Batch Size : 256
- Learning Rate : 0.001
- Discount Factor : 0.99
- Target Update Factor : 0.001
- Replay Buffer Size : 200000
- Exploration Decay Rate : 0.001

VI. Experiments

The motive of this project was to make a turtlebot robot learn to reach a goal without any prior knowledge of the map it was going to explore. Keeping in mind, the non-holoniomic nature of the turtlebot and the continuous state space we implemented the DDPG and SAC algorithms to get linear and angular velocities as the action space output. The following steps were followed during the entire experimental analysis of this project:

1) **Define the environments**: Certain set of environments were generated to train the robot about the specific setting of the environments. The primary objective was to make the robot learn how to reach the goal. So the initial environment which is the first environment as shown in Fig 7 was created just for the robot to learn to reach the goal and avoid collisions with the wall. After that the second environment was created to let the robot

learn not to collide with the obstacles present in the environment other than the walls of the environment.

2) **Define the Reward Function**: The learning in any Reinforcement Learning algorithm happens when the reward is associated with an action such that the robot learns either to perform that action by receiving a positive reward or to avoid that action for the penality that the robot received. Therefore, the reward function for this project is divided into two parts as shown in tables I and II. For the first reward function we have a static value that is 0 for the distance and heading components of the odometry output of the robot while the distance and the heading components were used and employed in the reward function. The reason for the two reward functions is that in the first environment we didn't want to optimize the distance as we just wanted for the robot to learn to reach the goal. While the second environment consisted of cylindrical obstacles and we wanted to make the robot learn to maneuver around the obstacles while keeping the distance minimum and therefore we introduced the distance and the angle reward.
3) **DDPG Algorithm (First Environment)**: Since first we wanted the robot to learn to reach the goal quickly and also avoid collisions with the boundary walls of the environment we implemented the DDPG algorithm on the first environment where the maximum number of the episodes were 10000. Due to bandwidth on the project completion time we implemented the DDPG algorithm upto the point where we started to see improvements in the performance of the turtlebot which was around 100 episodes where the robot stopped colliding with the boundary walls and starting going towards the goal.
4) **DDPG Algorithm (Second Environment)**: The second environment was a level up for the robot to learn as it consisted of the obstacles therefore we used the trained model for first environment as the starting point for the implementation of the second environment. But since the second environment had obstacles we trained it without and with the distance and angle rewards. The reason for the distance and angle rewards was to make the robot learn maneuverability around the obstacles. The robot was trained on the first static reward function on environment 2 for around 1800 episodes. For the better dynamic reward function the turtlebot was trained for 3000 episodes.
5) **SAC Algorithm (First Environment)**: Following the tryouts with DDPG, we wanted to test the performance of another algorithm, SAC. Similar to the DDPG case, we first wanted the robot to learn to reach the goal quickly. We trained the model for maximum of 10000 episodes using the reward model1, i.e. static model. Due to bandwidth on the project completion time we implemented the SAC algorithm up to the point where we started to see improvements in the performance of the turtlebot which was around 100 episodes where the robot stopped colliding with the boundary walls and starting going towards the goal.
6) **SAC Algorithm (Second Environment)**: Following the first trail of SAC without collisions, we wanted to extend the testing of SAC for Motion planning with collisions. For the second environment, we tried different combinations of reward model, with and without the distance and angle rewards. The reason for the distance and angle rewards was to make the robot learn maneuverability around the obstacles. The robot was trained on the first static reward function on environment 2 for around 1500 episodes. For the better dynamic reward function the turtlebot was trained for 1000 episodes.

VII. Results

The experiments performed while training the turtlebot to learn to navigate the two environments took some time and gave the outputs as represented by the following graphs.

1) **DDPG Algorithm (First Environment)**: The DDPG algorithm took around 100 episodes to learn to reach goal and avoid colliding with the wall boundaries of the environment and the graph for the mean reward vs episodes can be seen in the following Fig 9.

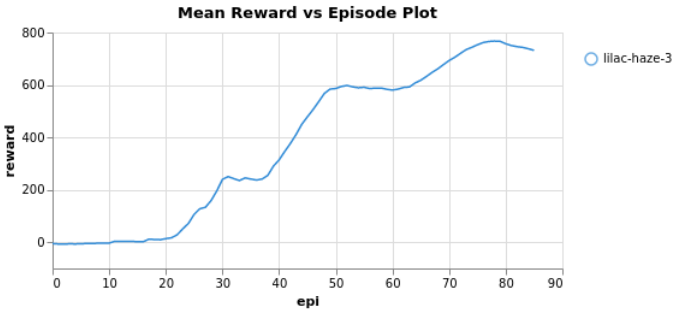

Fig. 9. Mean Reward vs Episodes DDPG Environment 1

2) **DDPG Algorithm (Second Environment)**: For the second environment as mentioned in the experiments section the DDPG algorithm was tested for the static reward function where the distance and angle reward had nothing to contribute and also for the dynamic reward function where the distance and angle reward contribute to the training of the turtlebot. The results for both of those can be seen through the plot of mean reward vs episode for the different reward functions in figs. 10 and 11
3) **SAC Algorithm (First Environment)**: The SAC algorithm was quick to learn to reach goal, within 25 episodes and avoid colliding with the wall boundaries of the environment and the graph for the mean reward vs episodes can be seen in the following Fig 12.
4) **SAC Algorithm (Second Environment)**: Following the first trial of SAC in environment 1, as mentioned in experiments section, we tested the SAC algorithm for motion planning with cylindrical obstacles. We used two sets of reward models, with and without considering

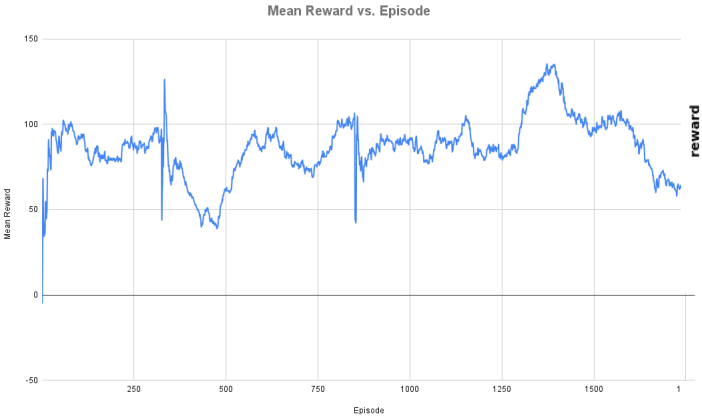

Fig. 10. Mean Reward vs Episodes DDPG Environment 2 Static Reward

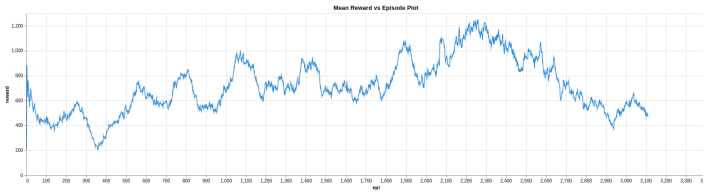

Fig. 11. Mean Reward vs Episodes DDPG Environment 2 Dynamic Reward

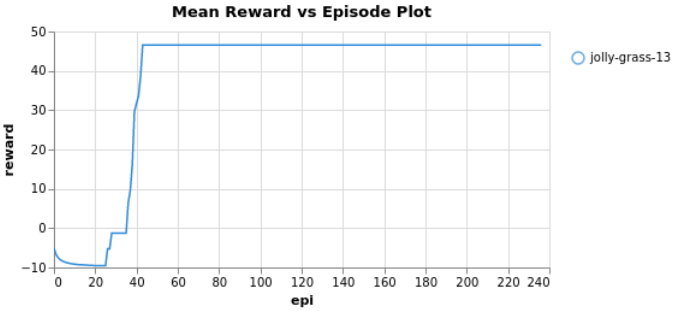

Fig. 12. Mean Reward vs Episodes SAC Environment 1

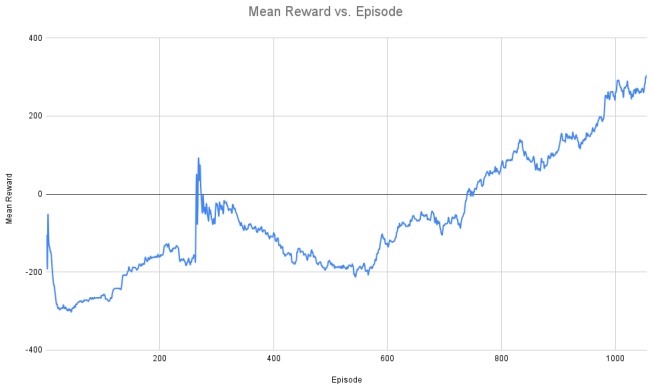

Fig. 13. Mean Reward vs Episodes SAC Environment 2 Static Reward

distance and angular reward. We found that the SAC algorithm has faster convergence in environment 2, within lesser number of episodes 500 in the following Fig 13 and 14.

VIII. Conclusion

In this study, we used a SAC network and a DDPG network for continuous control navigation of a mobile robot in simulated surroundings. In two separate simulated environments—one without obstacles and one with obstacles—the robot had to reach a goal point. It was suggested to use a reward mechanism to achieve this goal. The robot's linear and rotational velocities were produced using the SAC network and DDPG network, respectively.

For both suggested settings, the two algorithms could handle the problem of robot navigation. A study was done utilizing the performance of the intelligent agent algorithm in the job of avoiding obstacles and reaching the end objective using the training results acquired in the simulated environments. The SAC network was found to be more effective than the DDPG network since it demonstrated a very clean trajectory with minimal deviations and more precision while approaching the target and SAC gave faster convergence than DDPG over lesser number of episodes as it can be seen from the graphs. Also, we found that both the algorithms performed better with dynamic reward which took into account the distance and angular orientation of the robot with respect to goal position and orientation and had a more negative penalised reward for collision.

IX. Future Work

It is feasible to draw the conclusion that DDPG and SAC networks are appropriate for the creation of applications that require continuous robotics control. If the reward structure is appropriate for the issue that Deep-RL networks are trying to solve, they can deliver great results. These methods may be used to manipulate robotic arms, pendulums, and games, among other control systems.

In a later project, we want to apply similar techniques to the management of many agents in mobile robots as well as to the navigation of autonomous vehicles, where ill-defined regulations might result in hazardous circumstances.

Acknowledgment

We would like to express our gratitude to Prof. Yanhua Li for facilitating an intellectual dialogue on robotics and artificial intelligence.

References

[1] Auat Cheein, F.A., Lopez, N., Soria, C.M. et al. SLAM algorithm applied to robotics assistance for navigation in unknown environments. J NeuroEngineering Rehabil 7, 10 (2010). https://doi.org/10.1186/1743-0003-7-10.

[2] Macario Barros, Andréa, et al. "A comprehensive survey of visual slam algorithms." Robotics 11.1 (2022): 24.

[3] Duo, Nanxun, et al. "A deep reinforcement learning based mapless navigation algorithm using continuous actions." 2019 International Conference on Robots & Intelligent System (ICRIS). IEEE, 2019.

Fig. 14. Mean Reward vs Episodes SAC Environment 2 Dynamic Reward

[4] Zhu, Kai, and Tao Zhang. "Deep reinforcement learning based mobile robot navigation: A review." Tsinghua Science and Technology 26.5 (2021): 674-691.

[5] Takleh, Talha Takleh Omar, et al. "A brief survey on SLAM methods in autonomous vehicle." International Journal of Engineering & Technology 7.4 (2018): 38-43.

[6] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, A. Jagadeesh, A review: On path planning strategies for navigation of mobile robot, Defence Technology, Volume 15, Issue 4, 2019, Pages 582-606, ISSN 2214-9147, https://doi.org/10.1016/j.dt.2019.04.011.

[7] Xiao, X., Liu, B., Warnell, G. et al. Motion planning and control for mobile robot navigation using machine learning: a survey. Auton Robot 46, 569–597 (2022). https://doi.org/10.1007/s10514-022-10039-8.

[8] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[9] Schaul, T., Quan, J., Antonoglou, I. and Silver, D., 2015. Prioritized experience replay. arXiv preprint arXiv:1511.05952.

[10] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[11] Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.

[12] Gu, S., Holly, E., Lillicrap, T. and Levine, S., 2017, May. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In 2017 IEEE international conference on robotics and automation (ICRA) (pp. 3389-3396). IEEE.

[13] Mahmood, A.R., Korenkevych, D., Vasan, G., Ma, W. and Bergstra, J., 2018, October. Benchmarking reinforcement learning algorithms on real-world robots. In Conference on robot learning (pp. 561-591). PMLR.

[14] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Gupta, A., Fei-Fei, L. and Farhadi, A., 2017, May. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In 2017 IEEE international conference on robotics and automation (ICRA) (pp. 3357-3364). IEEE.

[15] Gu, S., Lillicrap, T., Sutskever, I. and Levine, S., 2016, June. Continuous deep q-learning with model-based acceleration. In International conference on machine learning (pp. 2829-2838). PMLR.

[16] Tai, L. and Liu, M., 2016. Towards cognitive exploration through deep reinforcement learning for mobile robots. arXiv preprint arXiv:1610.01733.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. nature, 518(7540), pp.529-533.

[18] Uhlenbeck, G.E. and Ornstein, L.S., 1930. On the theory of the Brownian motion. Physical review, 36(5), p.823.

[19] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P. and Andrychowicz, M., 2017. Parameter space noise for exploration. arXiv preprint arXiv:1706.01905.

[20] Achiam, J., 2018. Spinning up in deep reinforcement learning. GitHub repository.

[21] Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S., 2018, July. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). PMLR.

[22] Polyak, B.T. and Juditsky, A.B., 1992. Acceleration of stochastic approximation by averaging. SIAM journal on control and optimization, 30(4), pp.838-855.

[23] Jesus, J.C., Bottega, J.A., Cuadros, M.A. and Gamarra, D.F., 2019, December. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In 2019 19th International Conference on Advanced Robotics (ICAR) (pp. 362-367). IEEE.

[24] Zhelo, O., Zhang, J., Tai, L., Liu, M. and Burgard, W., 2018. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. arXiv preprint arXiv:1804.00456.

[25] Tai, L., Paolo, G. and Liu, M., 2017, September. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 31-36). IEEE.

[26] Chen, Y.F., Everett, M., Liu, M. and How, J.P., 2017, September. Socially aware motion planning with deep reinforcement learning. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1343-1350). IEEE.

[27] Kober, J., Bagnell, J.A. and Peters, J., 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), pp.1238-1274.

[28] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. and Levine, S., 2018. Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.

[29] url = https://www.irobot.com/enUS/roomba-vacuuming-robot-vacuum-irobot-roomba-69-4020/R694020.html last accessed Dec 5, 2022

[30] url = https://www.controleng.com/articles/robots-taught-to-hike-treacherous-uneven-terrain/ last accessed Dec 5, 2022