RoboNav: Robot Navigation in Dynamic Environments using Deep Reinforcement Learning

Abhishek Jain
ajain4@wpi.edu

Kavit Nilesh Shah
kshah@wpi.edu

Kenechukwu C. Mbanisi
kcmbanisi@wpi.edu

Sanjeev Kannan
skannan@wpi.edu

***Abstract*—Navigation is a fundamental behavior required by mobile robots to operate in human environments. Classical navigation methods exist which comprise a pipeline of environment perception, localization, path planning and control. In recent years, deep reinforcement learning approaches to navigation have proved to be promising in addressing the limitations of the classical methods (e.g. reliance on highly accurate maps and localization). In this project, we explored an end-to-end learning approach to train a navigation agent from raw perception information (i.e. laser scans) to velocity commands. Specifically, we consider two off-policy learning algorithms, *Deep Q Network* and *Deep Deterministic Policy Gradient* and train agents in different simulated training environments to perform point-to-point (P2P) navigation without colliding with obstacles. We evaluate our models against a baseline, *Move-Base*, which is a well-known classical navigation implementation in *ROS*. This report discusses our implementation, simulation results, findings and lessons learned.**

***Index Terms*—Robot Navigation, Deep Reinforcement Learning**

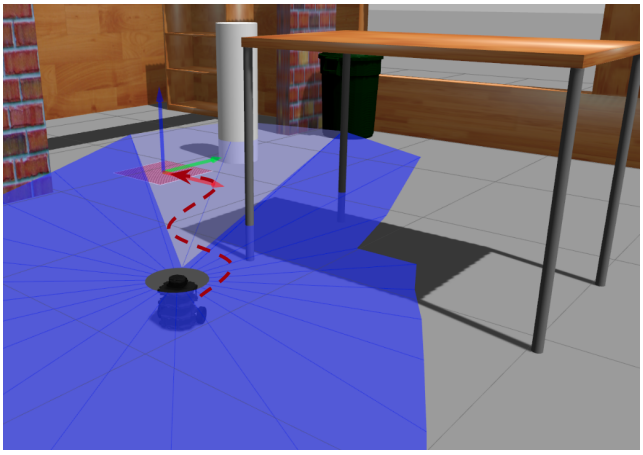


Fig. 1: Our end-to-end navigation agent maps raw laser scan data directly to velocity commands to perform point-to-point (P2P) navigation while avoiding obstacles in the environment.

I. Introduction

Autonomous navigation of mobile robots has remained an active research topic in the robotics community for decades. This is driven by the appeal to have mobile robots operate around humans in a safe and effective way with little or no human supervision. For a mobile robot to navigate autonomously, it must be able to perceive its environment, generate collision-free and dynamically feasible trajectories from its position to the goal, and then generate control commands to its actuators to track the trajectory. This sequence of capabilities describes the classical navigation pipeline for mobile robots.

Advances in perception, planning and control have lead to the application of mobile robot navigation in various fields such as space and underwater exploration, transportation (with self-driving vehicles, and more recently, delivery robots), military operations such as surveillance, etc.

Although the existing classical navigation methods have provided remarkable results, they suffer from significant limitations and drawbacks. For instance, these methods rely heavily on the availability of high-quality geometric maps to enable effective path planning. These maps are usually not readily available because of (1) in-feasibility of generating them, (2) high cost of sensing equipment required. Also, most classical navigation methods may require significant tuning and engineering for every new robot and environment they are to be deployed to. This limits the scalability of these systems in real-world scenarios.

With the burgeoning of artificial intelligence and machine learning, interest in learning-based approaches to the robot navigation problem has continue to grow. In the literature, the navigation problem can either be formulated as an end-to-end learning problem [1] [17] (where the RL agent learns the complete pipeline from perception to control) or on a component-level (i.e. either learning global planning or obstacle avoidance and relying on classical methods for the others). One of the advantages of the end-to-end learning approach is that the navigation pipeline is learned in a single model, therefore, eliminating the chance of cascading performance errors across components (which often occurs in classical methods). However, one major drawback is with classic problem of deep learning - lack of explainability.

In our project, we consider the end-to-end learning approach to train a navigation agent from raw perception information (i.e. laser scans) to velocity commands. Our work focuses on learning a policy for short-range point-to-point (P2P) navigation, thereby addressing the local planner problem. In real robotics systems, a high-level route planner would be required to provide waypoints for the local planner to follow. We consider two scenarios in formulating our problem, (1) a discrete action space, (2) a continuous action space. For the former, we implement a *Double Deep Q Network (DQN)* and for the latter, *Deep Deterministic Policy Gradient (DDPG)*.

We train and evaluate our policies in 4 different environ-

ments with varying geometric complexities and compare our performance with a standard, well-known classical navigation method implemented in ROS (Robot Operating System).

In the next section, we present related work and ground our approach in existing methods. Then in section III, we present our problem and methodology. Section IV covers the experiments performed and simulation results. Section V presents a comprehensive discussion on our findings and we conclude this report with a conclusion in section VI.

II. Related Work

Motion planning for mobile robots has been a popular research topic. Motion Planning is a critical component in robotic systems like self-driving vehicles and household helper robots. Classical methods to solve motion planning problems - range from traditional graph search methods to more recent techniques such as Structure from Motion (SfM) [14], Simultaneous Localization And Mapping (SLAM) [15], and Visual Odometry (VO) [16] are used.

Using Reinforcement Learning for mobile robot navigation has been a relatively new research topic. A recent paper by Liu et.al. [1] discusses a mobile robot operating in public environments that can avoid pedestrians in a safe and successful manner. Their approach combines deep reinforcement learning with imitation learning, and is capable of operation with limited field of view. [4] addresses a couple of major issues of Reinforcement Learning - large training data and lack of generalization to new goals. The paper uses a policy based on both goal and current state to tackle the latter, and a high quality 3D Map with a physics engine to address the former issue. Both the papers above use an approach based on existence of a map. However, there are several approaches that work well without one. A good example is the work by L. Tai et. al. [5] , where sensor range data and current position are used as inputs to the model and then output the required steering actions to navigate a mobile robot. A huge advantage of such a map-less approach is the ability to work in unseen virtual and real environments.

Common reinforcement algorithms used for mobile robot navigation include DDPG (Deep Deterministic Policy Gradient) and DQN (Deep Q-Networks). [11] discusses and compares the two approaches for mobile robot path planning. The paper also discusses an implementation of DDPG - its advantages in a continuous control setting and its disadvantages when it comes to training efficiency and convergence rates. [12] is another good paper that talks about introducing an influence value to shorten convergence time by 91% in DDPG and 78% in DDQN(Duelling DQN) for a mobile robot navigation problem.

For our project, we considered both map-based and map-less aproaches. Eventually, we went ahead with a map-less approach to teach our agent - a differential drive mobile robot to navigate an indoor closed room with moving obstacles using both DQN and DDPG. We also intend to compare the performance of both approaches in different environments.

III. Methodology

A. Problem Formulation

We can consider the mapless motion planning problem as a decision making process. At time-step t $\in$ [0, T], the robot chooses an action at according to the state s_t, it then executes the action, reaching a new state s_{t+1} and receiving a reward r. Our goal is to reach assigned goal locations in indoor environments with obstacles by maximizing the total discounted reward from time-step t_0 onward.

Any physical environment has two basic components: static and dynamic components. While static components are relatively easy to model as they have fixed properties, the dynamic components pose a great challenge. We aim to counter these challenges by using different approaches for these components. Whilst the static components can be modelled using occupancy grids, the dynamic components will be modelled using the sensor data to detect the real-time parameters for these components. This data is fed to the reinforcement agent along with the robot parameters as input whereas the output will be fed to the robot controller as input. The controller will output the modified robot state back to the environment.

The type of agent being used will be driven by the kind of action space being used. In case of continuous space, policy based techniques will be used while in the discrete case, value based techniques will be used. A key component of this method is the kind of reward function being used. We aim to experiment with both sparse and shaped reward functions.

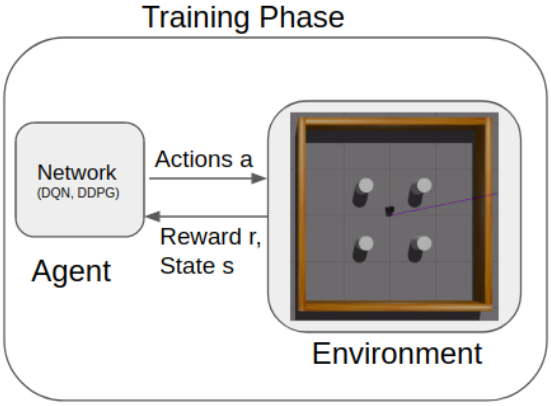
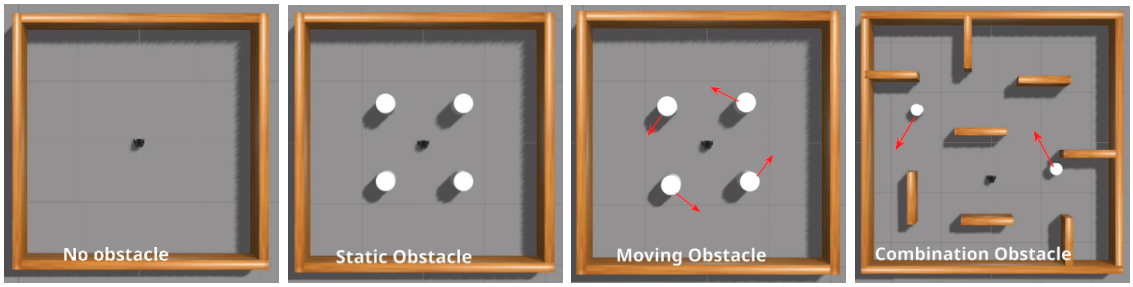


Fig. 2: Basic Architecture

B. Observation Space

The observation is local knowledge of the world acquired by the robot. The observation space comprises of almost all of the following:

1) Laser-scan data: An nx1 dimensional vector which contains laser range for the 'n' laser scans around the robot.
2) Position error: It is the distance between current position of the robot and the goal location inside the world.



(a) Empty world **(b)** With 4 static obstacles **(c)** With 4 moving obstacles **(d)** Maze

Fig. 3: Four training environments with incremental levels of complexity

3) Orientation error: It is the angle difference between current robot heading and direction of robot goal.
4) Minimum distance to obstacle : It is the minimum distance to any obstacle around the robot measured at every time-step using the laser-scans
5) Angle to obstacle : It is the angle of the closest obstacle measured with respect to the robot heading direction.

This approach is known as the map-less approach where the agent only has access to the laser range values from the robot environment as opposed to the other approach where one provides an entire map of the local environment as an image to the agent.

C. Action Space

The possible set of action inputs needed to drive the turtlebot are v_x (linear velocity along the robot heading) and w_z (angular velocity about vertical axis of the robot). We employ a discrete action space for DQN whereas a continuous action space for DDPG.

1) Discrete-action space :
The robot has a constant linear velocity while a discrete angular velocity space

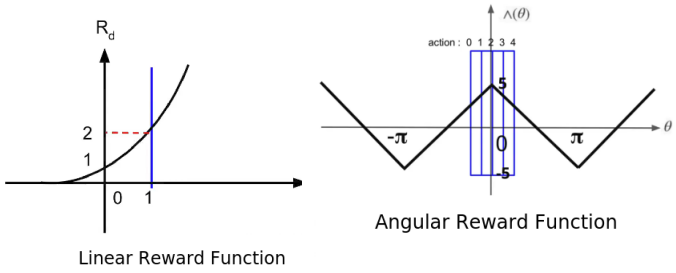
$$v_x = 0.15\ m/s \qquad (1)$$
$$w_z = \{\ 1.5\ 0.75\ 0\ -0.75\ -1.5\ \}\ rad/s$$

2) Continuous-action space : The robot has a constant linear velocity while the angular velocities of the robot are continuous.

$$v_x = 0.15\ m/s \qquad (2)$$
$$w_z \in [\ -2\ ,\ 2\ ]\ rad/s$$

D. Reward Function

Accurate training necessitates error-free reward function. The robot should receive a high positive reward for reaching the goal, and be penalized for its every collision with the world. Both - collision and reaching the goal, will terminate an episode. Also, the robot should be penalized at every step for the error in its orientation from the goal as well as its distance from the goal. This would not only ensure it always heads towards the goal and minimizes the distance, but would also imply it trying the follow "shortest-path" to the goal, essentially acting as a heuristic. The reward function in fig4 was used in the ROBOTIS' open-source implementation of DQN on turtlebot3 [9].



(a) Linear reward function **(b)** Angular reward function

Fig. 4: Reward function for discrete-action space

$$\theta = \begin{cases} \frac{\pi}{2} + \text{action} * \frac{\pi}{8} + \phi & \text{discrete action-space} \\ \frac{\pi}{2} + (1 - w_z) * \frac{\pi}{4} + \phi & \text{continuous-action space} \end{cases}$$

$$\mathrm{R}_\theta = 5 * (1 - \Lambda(\theta))$$

$$\mathrm{R_d} = 2^{\frac{D_c}{D_g}}$$

$$\mathrm{R_{net}} = R_\theta * R_\mathrm{d}$$

where,

action : Action index
ϕ : Yaw of turtlebot3
θ : Angle from goal
$\Lambda(\theta)$: Oscillating reward function
R_θ : Angular reward
D_c : Current distance from goal
D_g : Absolute distance from goal
R_d : Distance reward
R_{net} : Net reward
w_z : Angular speed in continuous space

The only change in both the reward functions is the discrete-action space is linearly mapped into a continuous action space, in the latter.

The reward function to penalize the robot at each step is the product of linear reward and angular reward which can be seen in Fig. 4. The oscillating reward function is a W-shaped discontinuous linear function which safeguards that the robot gets positive rewards when facing the goal and is penalized when facing away from the goal.

E. Deep Q Network (DQN)

DQN is an off-policy, model-free reinforcement learning algorithm which applies a deep neural network (as a non-linear function approximator) to estimate the "value", Q_π, over state-action pairs, (s_t, a_t), of taking a particular action a_t given a given state s_t, under a given policy π [6], [7].

This algorithm was introduced as an extension to the classical Q-learning approach. By leveraging deep neural networks, DQN is able to deal with high dimensional state/observation spaces (such as with image data).

For a given policy π, the action-value function Q_π is defined as follows:

$$Q_\pi(s_t, a_t; \theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | S = s_t, A = a_t \right] \quad (2)$$

where $Q_\pi(s_t, a_t)$ is the expectation of the total return (i.e. discounted accumulated reward) of taking action a_t at state s_t following policy π to the end.

DQN leverages the Bellman equation to iteratively learn and update the optimal action-value function based on:

$$Q^\star(s_t, a_t; \theta) = \mathbb{E}_\pi \left[R(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) | s_t, a_t \right] \quad (2)$$

To address limitations of the baseline DQN algorithm, techniques such as experience replay (to address interdependence and correlation in subsequent transition samples) and fixed target network (to improve stability of learning) have been proposed in the literature [6].

In this project, the DQN with experience replay and fixed target network are applied in training the value-based agent for the navigation task. The simple network structure, shown in Fig. 5, includes an input layer with 26 units (or 26 units for Stages 2-4), two fully connected layers with 512 units and an output layer with 5 units (for the 5 discrete actions defined above).

F. Deep Deterministic Policy Gradient (DDPG)

DDPG is an off-policy, model-free reinforcement learning algorithm for learning actions in continuous space. It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). It involves 2 neural networks: 'Actor and Critic' working together. The Actor network is a policy network which estimates the best action in a given state while

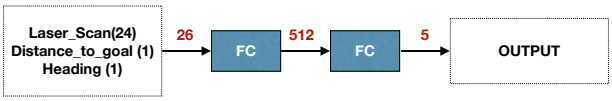


Fig. 5: DQN network architecture with input layer of 26 units (28 units for Stages 2-4) and output layer of 5 units (for Q values of all discrete actions)

the Critic network is a value based network which estimates the Q value of a given state-action pair [8].

DDPG leverages the advantages of both the networks to give a policy which is stable and works in continuous action space. It works in the same way as DQN works by using experience replay and fixed target actor and critic networks. The Critic network is exactly the same as the DQN network which estimates the Q values according to the modified Bellman Equation:

$$Q^\star(s_t, a_t; \theta) = \mathbb{E}_\pi \left[R(s_t, a_t) + \gamma Q(s_{t+1}, A(s_{t+1})) | s_t, A^\star(s_t) \right] \quad (2)$$

where $Q_\pi(s_t, a_t)$ is the expectation of the total return (i.e. discounted accumulated reward) of taking action a_t at state s_t following policy π to the end, a_t is the action predicted by the Actor network given by the Equation:

$$A^\star(s_t) = a_t \quad (2)$$

The actor network work towards selecting the action that maximizes the value of a state using the same Bellman Equation. In contrast to DQN, the training in DDPG is relatively more stable due to slower 'soft' updates of the target networks. Instead of total updates of the target network, the networks are updated continuously in small proportions which provides stability to the training using the Equation:

$$Target(w) = (1 - \tau) Current + (\tau) Target \quad (2)$$

where τ is another hyperparameter to be tuned.

In this project, a simple standard DDPG architecture is used. The actor is a neural network with an input layer with 26 units, a batch normalisation layer, three fully connected layers with 512 units, 256 units and an output layer with 2 units (for the 2 continuous actions defined above). The critic is a neural network with an input layer with 28 units, two batch normalisation layers, three fully connected layers with 512 units, 256 units and an output layer with 2 units.

IV. Experiments

A. Training Stages and Parameters

Training of the navigation agent was performed on a simulated differential drive robot (*Turtlebot3 Burger*) in a series of four simulated environments with incremental levels of complexity (see Fig. 3) [10].

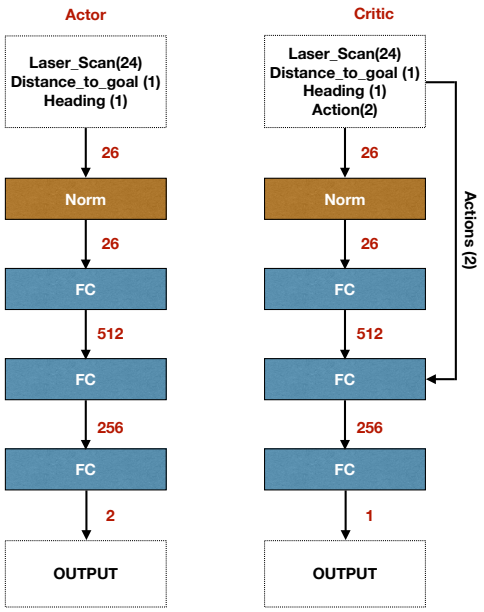


Fig. 6: DDPG Architecture

1) **Stage 1** (Empty Box): 2m x 2m square world with robot spawned at the center.
2) **Stage 2** (With static obstacles): Augments Stage 1 with 4 static white cylinders.
3) **Stage 3** (With dynamic obstacles): Animates static obstacles in Stage 2 in a constrained circular motion.
4) **Stage 4** (Maze with dynamic obstacles): Maze-like world with dynamic obstacles random motion.

The training and evaluation simulations are implemented in Gazebo simulator via ROS (Robot Operating System) [9]. ROS provides the framework for enabling communication between the simulator (Gazebo) and our training algorithm implemented in Python using PyTorch.

For environment perception, a 360-degree LiDAR mounted on the robot provided laser scans in every simulation step. The resolution of the laser scan was reduced from 360 points to 24 points to reduce the neural network input dimensionality. We trained the various models using an Adam optimizer on GPU-enabled clusters provided by GCP (Google Cloud Platform). Table I shows the parameters applied in the agent training process. Individual agents were trained for each of the environments. Through experimentation, we observed that stages with more complexity (Stages 3-4) required a slower exploration rate (i.e. rate at which ϵ decays) to enable the agent encounter more state-action pairs in the environment. Therefore, for Stages 1-2, we applied decay to 0.1 in 100,000 steps for DQN and then for Stages 3-4, we applied decay to 0.1 in 300,000 steps.

B. Evaluation Metrics

After the models are trained, we test the models using a comprehensive framework. Each trained model is run on a set of 10 trials, where each trail consists 5 randomly generated goal locations. For each trial, parameters including x,y coordinates, distance to obstacles and velocities are logged. After the trial, the logged data is used to compute :

1) Success Rate: A boolean that stores true if all goal locations have been reached, else false.
2) Path Length: This alludes to the total length of the distance travelled by the robot to cover all goal locations.
3) Clearance: This is the minimum distance of the robot to any obstacle in its vicinity.

Additionally, we also compute Mean path length, Mean clearance across several trials to compare performance of DQN, DDPG with a baseline method (Move-base).

Parameter	DQN	DDPG
Minibatch size	64	64
Replay buffer size	1.0e6	1.0e6
Replay start size	5.0e3	5.0e2
Learning rate	2.5e-4	2.5e-4
Model update freq.	5.0e3	1
Tau	-	1.0e-3
Initial exploration	1.0	1.0
Intermediate exploration	-	0.1 (in 1.0e5)
Final exploration	0.1 (in 1.0e5)*	0.01 (in 0.9e6)
Discount factor	0.99	0.99

TABLE I: Training Parameters for DQN and DDPG

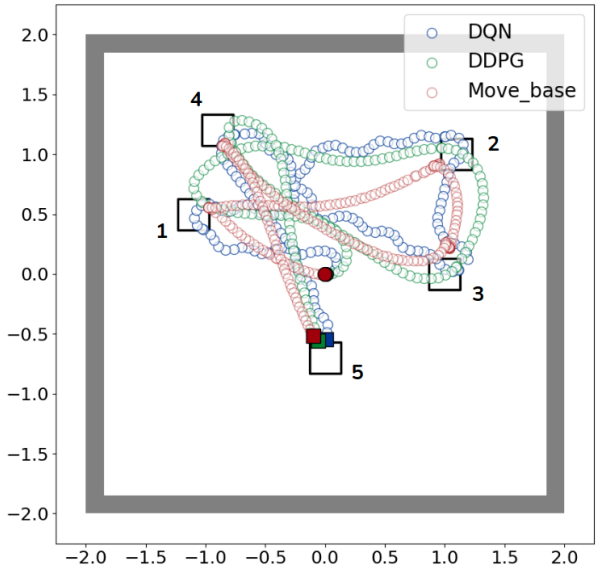


Fig. 7: Example plot of an evaluation trial in Stage 1 with three methods (DQN, DDPG and Move-base) reaching 5 goal positions.

C. Baseline: Move-Base

The move_base package, an integral component of the ROS Navigation Stack is a library for autonomous robot navigation in indoor environments. To compute a path, the package makes the use of one of the many motion planning algorithms such as A*, D*, Dijkstra, Elastic Band,etc. For this it requires a map of the indoor environment which is obtained by performing

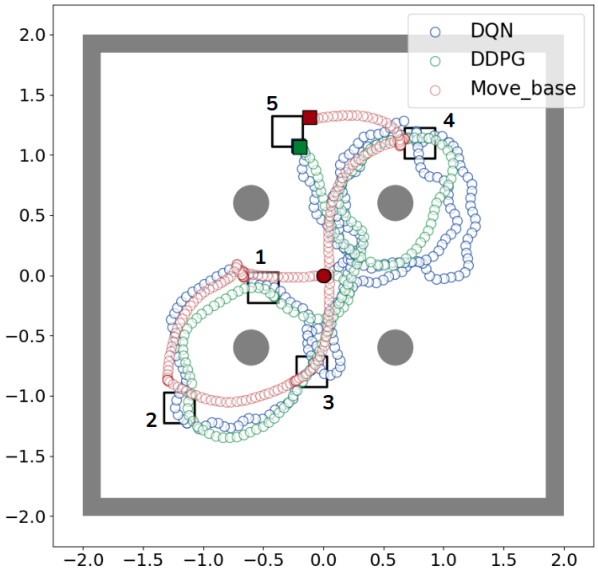


Fig. 8: Example plot of an evaluation trial in Stage 2 with three methods (DQN, DDPG and Move-base) reaching 5 goal positions

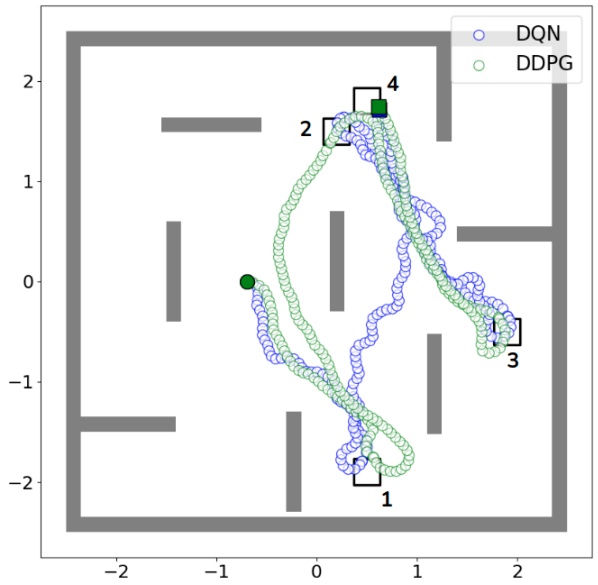
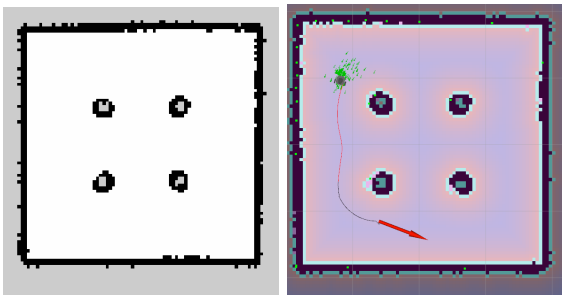


Fig. 9: Example plot of an evaluation trial in Stage 4 showing a compariosn of our two trained RL methods (DQN and DDPG) reaching 4 goal positions

Simultaneous Localization And Mapping (SLAM) on the world that generates the world's map like seen in the Fig.10a.

Utilizing this global map of the world, the move_base computes a global path with its global_planner, and while the robot executes this path, its local_planner iteratively fine-tunes the local path for path optimization, obstacle avoidance and path clearance as seen in Fig.10b. This is called hierarchical motion planning and this is makes move_base robust and an ideal choice to benchmark our model.



(a) SLAM Map for Stage2 **(b)** RVIZ window

Fig. 10: Move_base in action

Metric	DQN	DDPG	Move_Base
Success rate (%)	100	100	100
Mean Path Length (meters)	9.25	9.44	8.47
Mean Clearance (meters)	1.09	1.01	1.12

TABLE II: Performance metric comparison for Stage 1

Metric	DQN	DDPG	Move_Base
Success rate (%)	70	100	100
Mean Path Length (meters)	12.14	10.35	9.60
Mean Clearance (meters)	0.47	0.39	0.45

TABLE III: Performance metric comparison for Stage 2

Metric	DQN	DDPG	Move_Base
Success rate (%)	40	90	20
Mean Path Length (meters)	14.70	14.69	7.96
Mean Clearance (meters)	0.43	0.42	0.46

TABLE IV: Performance metric comparison for Stage 3

Metric	DQN	DDPG	Move_Base
Success rate (%)	20	60	60
Mean Path Length (meters)	11.59	11.97	7.73
Mean Clearance (meters)	0.52	0.44	0.47

TABLE V: Performance metric comparison for Stage 4

D. Simulation Results

The results from the experiments performed in the 4 stages give some very useful insights:

Table II shows that:

1) All the three agents successfully completed all the trial for the stage 1.
2) The Move-Base agent gave the shortest path followed by the DQN agent and then DDPG agent.
3) The DQN and Move-base agents had a much better obstacle clearance as compared to the DDPG agent.

Table III shows that:

1) DDPG and Move-Base agents successfully completed all the trials for the stage 2 while the DQN agent was able to complete 70% of the trials.
2) The Move-Base agent gave the shortest path followed by the DDPG agent and then DQN agent.

3) The DQN and Move-base agents had a much better obstacle clearance as compared to the DDPG agent.

Table IV shows that:

1) The DDPG agent gave the best performance in Stage 3 while the DQN agent performed poorly and the Move-Base agent failed miserably.
2) The Move-Base agent gave the shortest path while the DDPG and DQN agents gave comparable paths.
3) The DQN and Move-base agents had a much better obstacle clearance as compared to the DDPG agent.

Table V shows that:

1) The DDPG and Move-base agents gave comparable performance while the DQN agent performed poorly.
2) The Move-Base agent gave the shortest path while the DDPG and DQN agents gave comparable paths.
3) The DQN agent had a much better obstacle clearance as compared to the other agents.

V. Discussion and Future Work

We performed robot navigation using 3 different techniques: DQN, DDPG and Move-Base. DQN and DDPG are based on deep reinforcement learning where the agent learns to navigate in the environment whereas Move-Base is based on motion planning where the agent calculates the path based on the given instantaneous data. As seen in Figures 7, 8 and 9, the DQN agent follows a jittery path whereas the DDPG and Move-base agents follow a much smoother path. This owes to the discrete action space that the DQN agent operates in while the DDPG and Move-Base agents have been operating in a continuous action space.

As we moved from Stage 1 to Stage 4 environments, the complexity of the environments gradually increased with respect to the number of obstacles and nature of their motion. Stage 1 being just an open space without any obstacles had all the agents successfully completing all the trials where as Stage 2 had some static obstacles in the space, which led to some agents failing some trials. Stage 3 had dynamically moving obstacles but in a fixed motion. Therefore, the results in stage 3 gave us some great insights: Since, the DQN agent had a limited action space, the agent did perform well in some trials but failed in other trails. The DDPG agent performed very well in most of the trials owing to its large available action space whereas the Move-Base agent failed miserably due to the fact that it just performed according to the available data and did not had the capability of estimating the behaviour of the moving obstacles. Stage 4 again showed similar results to that of Stage 3 where the DQN agent nearly performed the same whereas the perfomance of the DDPG agent dropped a bit but the Move-base agent performed significantly better probably due to Stage 4 environment having random obstacle motion.

There were some interesting findings in the project. The Move-base agent gave the best trajectory out of all the 3 agents but gave a poor performance in case of dynamic environments. The DDPG agent gave the best performance in all the 4 stages and a significantly smoother path whereas the DQN agent was superior in terms of learning quickly from the environment but gave average results in terms of successful navigation and path smoothness owing to its limited action-space. This leaves us with multiple avenues for experimentation and research where we could improve the performance of the Deep Reinforcement Learning agents. One possibility could be to use a stack of sensor data and not just instantaneous data so as to learn the dynamic aspects of the environment. We could also try to increase the number of layers used in our neural networks so as to see if the agents are able to extract more information from the available data. We could also increase the number of LiDAR scan samples from 24 to some other larger value so as to get more precise information from the environment.

VI. Conclusion

The aim of the project was to use Deep Reinforcement Techniques (DRL) for navigating a mobile robot in different indoor environments - with both static and dynamic obstacles. There were 2 different DRL agents: DQN and DDPG that were used in this project. Their performance - in terms of mean path length, clearance, and path smoothness is compared to a traditional motion planner (based on Move-Base framework). From our results, we were able to gather a lot of insights into pros and cons of both methods. The DQN agent was quick to learn but gave average performance. The DDPG agent gave better performance but had huge computation requirements. The Move-base agent gave superior performance but performed poorly in the case of dynamic environments. Through this project, we were able to understand and successfully apply Reinforcement Learning techniques to mobile robot navigation.

References

[1] Liu, L., Dugas, D., et. al. Robot Navigation in Crowded Environments Using Deep Reinforcement Learning. Presented at IROS 2020.

[2] Chen, G., Pan, L., et. al (2020). Robot Navigation with Map-Based Deep Reinforcement Learning. http://arxiv.org/abs/2002.04349

[3] L. Tai, G. Paolo and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," 2017 (IROS), Vancouver, BC, 2017, pp. 31-36.

[4] Y. Zhu et al., "Target-driven visual navigation in indoor scenes using deep reinforcement learning," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 3357-3364, doi: 10.1109/ICRA.2017.7989381.

[5] L. Tai, G. Paolo and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, 2017, pp. 31-36, doi: 10.1109/IROS.2017.8202134.

[6] 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.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[8] Lillicrap, Timothy & Hunt, Jonathan & Pritzel, Alexander & Heess, Nicolas & Erez, Tom & Tassa, Yuval & Silver, David & Wierstra, Daan. (2015). Continuous control with deep reinforcement learning. CoRR.

[9] https://www.ros.org/about-ros/

[10] https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/

[11] Dong, Yuansheng, and Xingjie Zou. "Mobile Robot Path Planning Based on Improved DDPG Reinforcement Learning Algorithm." 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2020.

[12] Yu, Jinglun, Yuancheng Su, and Yifan Liao. "The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning." Frontiers in Neurorobotics 14 (2020).

[13] Marchesini, Enrico, and Alessandro Farinelli. "Discrete Deep Reinforcement Learning for Mapless Navigation." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.

[14] Ullman, Shimon. "The interpretation of structure from motion." Proceedings of the Royal Society of London. Series B. Biological Sciences 203.1153 (1979): 405-426.

[15] Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous localization and mapping: part I." IEEE robotics & automation magazine 13.2 (2006): 99-110.

[16] Nistér, David, Oleg Naroditsky, and James Bergen. "Visual odometry." Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.. Vol. 1. Ieee, 2004.

[17] Bansal, S., Tolani, V., Gupta, S., Malik, J. and Tomlin, C., 2020, May. Combining optimal control and learning for visual navigation in novel environments. In Conference on Robot Learning (pp. 420-429). PMLR.