MEEG 591V Final Project

Reinforcement Learning for Autonomous Mobile Robot Navigation

Mishek Musa

December 16, 2021

Abstract

In this project a reinforcement learning approach, DQN, was used to train a robot for navigation in an unknown environment by using data available from sensors on the robot (LiDAR, encoders, etc.). Currently a robot prototype does not exist, however a simulated robot and environment was created using the open-source physics engine Gazebo in ROS. The robot was trained for 110 episodes and it can be seen that the robot is able to navigate through the environment and reach its goals while avoiding collisions with the boundaries. This work lays a good foundation for the further development of the system to be used for real-time mobile robot navigation.

1 Introduction

In the field of mobile robotics, path planning is the basis of mobile robot navigation and control [1]. The basic idea of path planning revolves around giving the robot a start and end goal, and finding a path between these two goals. Typically the path should be optimal with regards to minimizing the travel distance, travel time, or robot energy, it should be collision-free, and should be smooth enough that the robot can actually traverse it without violating the system dynamics [2]. Many algorithms have been developed to solve the path planning challenge, however, they suffer from several pitfalls particularly with respect to navigation in an unknown environment. In an unknown environment, full information about obstacles is not available and so the robot must rely on sensor perception to gain partial knowledge about the environment [3]. Even with this knowledge, designing controllers/algorithms to make good decisions is a daunting task. Additionally, the traditional path planning algorithms typically rely heavily on static, global maps of the environment, which is not feasible in real-world scenarios where unknown static and dynamic obstacles may exist [4]. Reinforcement learning is an approach that can be used to eliminate the reliance on the environment information, and enable the robot to quickly adapt to changes in its surroundings.

In general, the fundamental path planning algorithms can be divided into two main categories based on their functionality. These two categories are: 1) grid-based search algorithms, and 2) sampling-based algorithms. Two of the most common grid-based search approaches are the Dijkstra and A* algorithms [5]. In these approaches, the algorithm overlays the map with a grid of nodes and attempts to find the shortest path connecting the nodes from the start to the end goal. The challenge with using an approach like this however is the high computation cost, especially if dealing with high-dimensional problems. Additionally, this approach relies heavily on a map of the environment. The second type of approach, sampling-based algorithms, generate a roadmap

by sampling the configuration space. In the case of the rapidly exploring random trees (RRT) algorithm, a root is established at the initial start configuration and branches are created by sampling the environment randomly [6]. If the branch is feasible (no collisions) then it is saved and more branches are created by sampling around the new branch. This is repeated until the target goal is acquired. This approach works well for high-dimensional problems, but again, relies heavily on a known map of the environment.

At the most fundamental level, reinforcement learning is a subset of machine learning in which agents attempt to take actions or make decisions with the goal of receiving some form of a reward at the end [7]. Reinforcement algorithms are typically comprised of 3 basic elements: 1) the agent, 2) the environment, and 3) the reward. The agent is responsible for taking actions in its current state, the environment is what responds to the actions the agent takes and gives new inputs to the agent, and the reward is an incentive that is returned by the environment. The primary goal of reinforcement algorithms is to maximize rewards based on trial-and-error. However, it is important to maintain a balance between exploration (trying new things) and exploitation (relying on previously captured data).

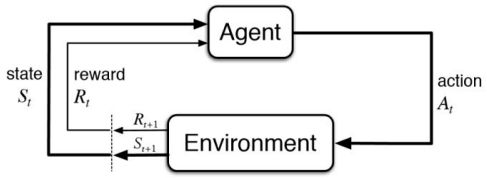


Figure 1: Structure of a typical reinforcement learning algorithm [7]

In navigation with reinforcement learning, the robot learns the best strategy to navigate through the environment by directly interacting with the environment through a trial-and-reward based approach. In the literature, several reinforcement learning approaches have been proposed for autonomous robot navigation. Without going into too much details on the specifics of each algorithms, one of the first implementations was seen in Smart et al. whereby a Q-learning algorithm was used to give sparse rewards to the robot upon successful navigation of the environment [8]. More recently, Deep Q-Networks (DQN) have successfully been implemented to achieve optimal end-to-end mobile robot path planning [9, 10, 11]. Another reinforcement learning algorithm that has been implemented for robot motion planning with increased performance and fast convergence is Deep Deterministic Policy Gradient (DDPG) [12, 13]. In each case, the mobile robot is able to navigate through the environment with collision-free paths without any prior knowledge of the map, is computationally efficient (after training), and has fast convergence.

2 Methods

2.1 Mobile Robot Kinematics

For this work, only the kinematic formulation of the differential drive mobile robot is considered (ie. no forces affecting motion are considered). The kinematic formulation allows for the robot velocity to be represented as a function of the driving wheel velocities subject to geometric parameters of the robot.

Robot Constraints

1. The robot motion is in the X-Y plane and so has 3 DoF.

2. No lateral slip occurs between the robot and the surface.

3. The robot is undergoing pure rolling and so there is no slipping of the wheel along the longitudinal axis and no orthogonal skidding.

The linear and angular velocities of the robot with respect to the local frame is then given by

$$v = \begin{bmatrix} v_x & v_y & 0 \end{bmatrix}^T \tag{1}$$

$$\omega = \begin{bmatrix} 0 & 0 & \omega \end{bmatrix}^T \tag{2}$$

The robot state vector (position and orientation) in the inertial frame is given by

$$q = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \tag{3}$$

The relationship between the robot frame velocities and the inertial frame velocities can be written as follows:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{4}$$

For this robot, the goal of the kinematic formulation is to represent the robot's velocities as a function of the driving wheel velocities. The linear velocity of the robot in the robot frame can be written as the average of the linear velocities of the two wheels

$$v = \frac{v_R + v_L}{2} \tag{5}$$

Similarly, the angular velocity of the robot can be written as the difference between the linear velocities of the two wheels

$$\omega = \frac{v_R - v_L}{2R} \tag{6}$$

These can further be written in terms of the angular velocities of the wheels given by

$$v = \frac{v_R + v_L}{2} = R\frac{(\dot{\gamma}_R + \dot{\gamma}_L)}{2} \tag{7}$$

$$\omega = \frac{v_R - v_L}{2R} = R\frac{(\dot{\gamma}_R - \dot{\gamma}_L)}{2L} \tag{8}$$

Rewriting in matrix form, the relationship between the robot velocities in the robot frame and the wheel angular velocities is given by

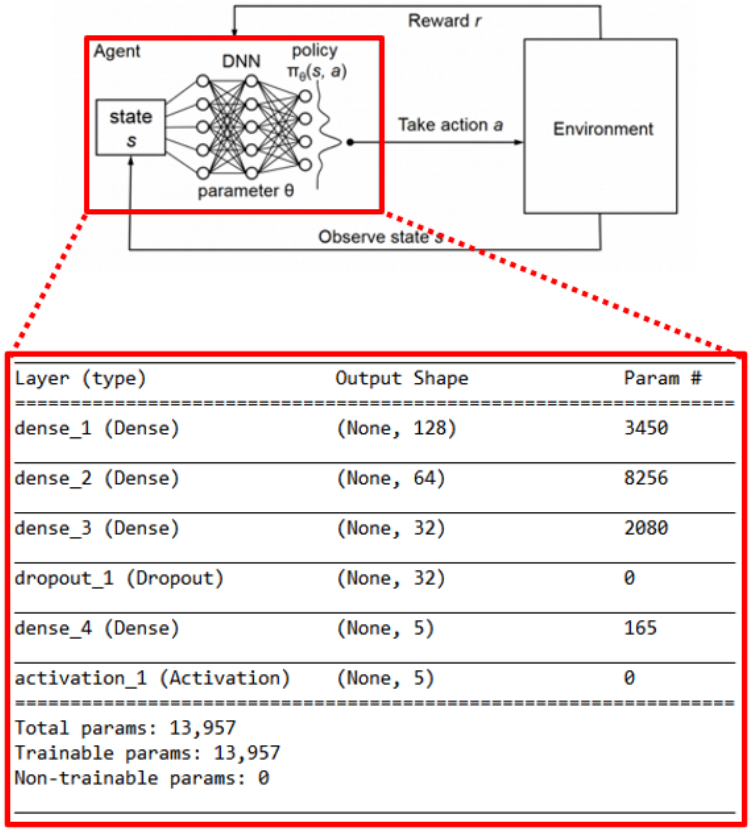
$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\gamma}_R \\ \dot{\gamma}_L \end{bmatrix} \tag{9}$$

2.2 Deep Q-Network (DQN)

Deep Q-networks are an extension of the traditional Q-learning algorithm used in reinforcement learning. The Q-learning algorithm was first developed by Watkins, and is described as "provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains" [14]. In a Markovian domain, a Q function calculates the expected utility for a given finite state s and possible finite action a. The agent then selects an optimal action a based on its current state s in the environment by evaluating the maximum value of $Q(s, a)$. Initially, $Q(s, a)$ function values are assumed to be zero, however, after each training episode, the values are updated according to the function

$$Q(s, a_t) \leftarrow Q(s, a_t) + \alpha[r + \gamma \max Q(s_{t+1}, a)] \tag{10}$$

where α is the learning rate, and γ is an attenuation factor. The function values of $Q(s, a)$ are typically stored in a lookup table commonly referred to as a Q-table. This can become very cumbersome and so to overcome this, DQN uses a neural network to create the mapping between input states to the action and Q value pairs. Due to its complexity of implementation, particularly when trying to interface with ROS, a DQN architecture was adapted from the open-source libraries created by Robotis [15]. The hyper parameters for the RL model used are outlined in the table below. Additionally, the architecture of the neural network used in the DQN agent can be seen in Fig. 2.



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	3450
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 5)	165
activation_1 (Activation)	(None, 5)	0

Total params: 13,957
Trainable params: 13,957
Non-trainable params: 0

Figure 2: Structure of a typical reinforcement learning algorithm with DQN agent

Hyper parameter	Value	Description
Batch Size	64	size of training sample group
Target Update Rate	2000	update rate of the target network
Discount Factor	0.95	represents how much future events lose their value according to how far away
Learning Rate	0.005	
Epsilon	1.0	probability of choosing a random action
Epsilon Decay	0.99	After each episode, the epsilon reduces at this rate

2.3 Simulation Environment

The robot and its environment were simulated in the Robot Operating System (ROS) (version: ROS Kinetic Kame) using the built-in physics simulator Gazebo on a Lenovo Yoga-710 machine (CPU: Intel i7) running Ubuntu 16.04. The environment used is based on the open-source testbed provided by Robotis. The environment consists of 4 walls that are considered boundaries/obstacles for the robot to interact with. The differential drive kinematics outlined in the previous section were implemented using open-source ROS nodes for the Turtlebot3. The Turtlebot3 is an open-source 2 wheeled robot commonly used by research groups for mobile robot algorithm development and testing. The robot is equipped with a 360° LiDAR (light detection and ranging) sensor which is capable of measuring distances and relative angles of the obstacles in the environment. Additionally, the initial position and the goal point of the robot is given in absolute coordinates, and the current position of the robot is calculated according to its odometry.

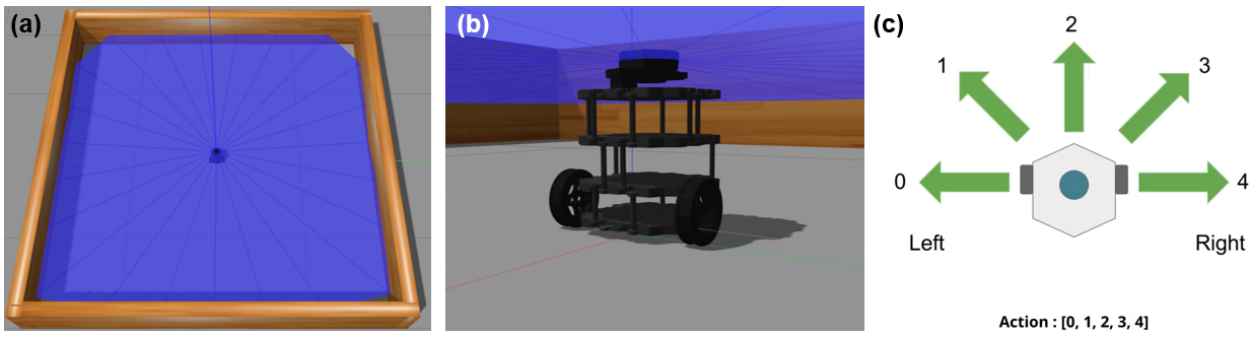


Figure 3: (a) Gazebo environment with 4 walls as the boundaries, (b) simulated Turtlebot3 with LiDAR Scanner, (c) discrete actions possible for the robot to make

The robot has a constant linear velocity of 0.15 m/s and is capable of taking 5 discrete actions given in the form of the following angular velocity commands:

Action 0: -1.5 rad/s

Action 1: -0.75 rad/s

Action 2: 0 rad/s

Action 3: 0.75 rad/s

Action 4: 1.5 rad/s

The environment is also responsible for returning rewards to the agent during training. The closer the robot gets to the goal the more positive the reward is, while the further away, the more negative the reward becomes. When the robot reaches the goal, it receives a large positive reward and a new goal point is randomly defined. An episode ends if the robot collides with the boundary or if the simulation time goes beyond a threshold. The reward of the robot is defined by the following equations:

$$\theta = \frac{\pi}{2} + action \cdot \frac{\pi}{8} + \phi \tag{11}$$

$$R_\theta = 5 \cdot (1 - \theta) \tag{12}$$

$$R_d = 2^{\frac{D_c}{D_g}} \tag{13}$$

$$R_{total} = R_\theta \cdot R_d \tag{14}$$

where θ is the heading angle of the robot in the global frame, ϕ is the yaw angle of the robot in the robot frame, R_{total} is the total reward of the robot, R_θ is the angular reward, and R_d is the distance reward. The simulation runs in near real time and so is quite time consuming. Additionally, the simulation is very large and unfortunately, I ran into multiple memory issues while training the RL agent. The results presented in the following section are for 110 episodes.

3 Results and Analysis

The DQN agent was trained for 110 episodes. This was the maximum number of episodes achievable before the computer crashed due to memory issues. Figure 4(a) shows the total cumulative reward per episode during training. It can be seen that the reward increases with episode number which is expected as the robot is increasingly reaching the goal and not colliding with the walls as frequently. Figure 4(b) shows the average max Q-value determined by the neural network. We can see that the value is increasing indicating that the robot is still learning. One would expect the max Q-value to plateau after a sufficient number of episodes. Qualitatively the performance of the DQN can be seen in the attached videos where the *EarlyRuns* video shows the robot in its first few episodes of training. The robot is taking random actions and frequently collides with the boundary. In the *EndRuns* video, the robot actions are still a bt random and sporadic, however, collisions with the walls are less frequent and the robot reaches the goal consistently.

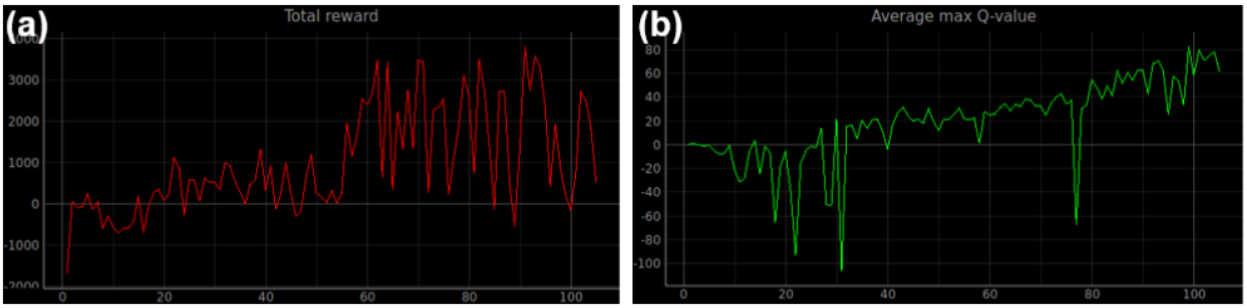


Figure 4: (a) cumulative reward vs. number of episodes, (b) average max Q-value vs. number of episodes

4 Conclusion

In this work, reinforcement learning was investigated as a means of achieving autonomous mobile robot navigation. The system was tested in a simulated environment implemented using open-source modules for ROS. The DQN algorithm was able to learn policies that allowed the robot to navigate through the environment to the goal while avoiding collisions with the boundaries, however, the motion of the robot still requires more training to become optimal. Some of the main challenges faced included the interfacing of the DQN algorithm with the Gazebo simulation. Additionally, significant memory issues occurred resulting in limited training episodes. The work conducted in this study lays a good foundation for the development of more complex scenarios, with custom designed robots that match the work being done for my research. In the future, assuming the hardware limitations are fixed, the trained DQN could be tested in other environments to demonstrate its ability to adapt to unknown environments.

References

[1] S. Ghosh, P. K. Panigrahi, and D. R. Parhi, "Analysis of fpa and ba meta-heuristic controllers for optimal path planning of mobile robot in cluttered environment," *IET Science, Measurement & Technology*, vol. 11, no. 7, pp. 817–828, 2017.

[2] J. Han and Y. Seo, "Mobile robot path planning with surrounding point set and path improvement," *Applied Soft Computing*, vol. 57, pp. 35–47, 2017.

[3] Y. Wang, Y. Fang, P. Lou, J. Yan, and N. Liu, "Deep reinforcement learning based path planning for mobile robot in unknown environment," *Journal of Physics*, vol. 1576, p. 012009, jun 2020.

[4] L. Chang, L. Shan, C. Jiang, and Y. Dai, "Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment," *Autonomous Robots*, vol. 45, no. 1, p. 51–76, 2020.

[5] Z. He, J. Wang, and C. Song, "A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures," 2021.

[6] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[7] S. Sharma, "The ultimate beginner's guide to reinforcement learning," Jun 2020. Available at `https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec`.

[8] W. Smart and L. Pack Kaelbling, "Effective reinforcement learning for mobile robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, pp. 3404–3410 vol.4, 2002.

[9] J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path planning," in *2017 Chinese Automation Congress (CAC)*, pp. 7112–7116, 2017.

[10] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *CoRR*, vol. abs/1610.01733, 2016.

[11] X. Xue, Z. Li, D. Zhang, and Y. Yan, "A deep reinforcement learning method for mobile robot collision avoidance based on double dqn," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, pp. 2131–2136, 2019.

[12] Y. Dong and X. Zou, "Mobile robot path planning based on improved ddpg reinforcement learning algorithm," in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 52–56, 2020.

[13] X. Gao, L. Yan, G. Wang, T. Wang, N. Du, and C. Gerada, "Toward obstacle avoidance for mobile robots using deep reinforcement learning algorithm," in *2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 2136–2139, 2021.

[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, p. 279–292, 1992.

[15] "Robotis." Available at `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`.