Undergraduate Summer Internship Project Report

LEARNING-BASED AUTONOMOUS NAVIGATION IN CROWDED ENVIRONMENTS

Swetha Rajagopalan

Supervisor: Dr. You Yangwei

1. **Introduction**

The field of robotics has witnessed many major advancements over the years and has transformed into an area of great interest to many. An important application of robotics is autonomous navigation, or self-driving cars. With appropriate hardware and robust learning algorithms, it is possible to train agents to navigate through a path safely and intelligently. One such learning method is reinforcement learning. Reinforcement Learning is a branch of Machine Learning. It is a form of supervised learning method which can be used to train an agent to perform actions which would maximize its reward in a given situation. Deep Reinforcement Learning (DRL) is an improvised form of Reinforcement Learning and A policy maps an agent’s perceived states of the environment to what actions the agents must perform, and hence determines which actions would maximize the agent’s reward. The goal of this project was to develop environments to train robots and evaluate the robots’ learning behavior using specific learning algorithms.

1. **Objective of the Project**

The main objective was to develop environments for training. These environments would be tested using specific control algorithms which utilize reinforcement learning. The environments were of three types: obstacle-free, containing static obstacles, and containing dynamic obstacles. To minimize randomness while training real robots, we decided to test our environments and learning algorithms on a simulator, Gazebo, and determine whether the training data could be transferred to a real robot. We decided to use specific deep reinforcement learning algorithms to train the virtual robot because DRL enables the learning process to be faster, more robust, and utilize large datasets, which is potentially required in case of training self-driving agents. To do this, we made use of the Gym-Gazebo[[1]](#footnote-1) repository on Github created by Erle Robotics.

1. **Project process:**
2. ***Install the necessary packages***

The first stage of the project involved cloning the Gym-Gazebo repository to the system and making the packages required. The Gym-Gazebo repository contains custom-made, non-catkin packages and hence, performing catkin\_make or catkin build proved to be initially difficult and required some debugging. Once this was done, the testing codes were executed to check whether the installation had completed successfully.

1. ***Creating the environments***

To create the environments, we utilized the world file named “circuit2” from the Gym-Gazebo repository and added custom objects to it as obstacles when required.

* 1. *Obstacle-free environments:*

The circuit2 world file contained a maze with no obstacles and a clear path. This was the first environment we used. The below figure illustrates the environment along with the robot:

A screen shot of a computer

Description automatically generated

*Figure 1: Robot in the environment containing no obstacles*

While running the simulation in Gazebo, we also ran RViz to see evaluate the behavior of the robot’s lidar scanner. Below is a figure illustrating the RViz graph:

A screenshot of a computer

Description automatically generated

*Figure 2: RViz graph of robot while navigating in obstacle-free environment*

The robot’s learning behavior was stable and predictable. By the end of 400 episodes, the robot had learnt to navigate through the maze once without colliding into any wall. By the end of 1000 episodes, the robot was able to navigate through the maze successfully at least thrice. The below graph illustrates the robot’s behavior and the rewards for each episode at the end of 100 episodes:

A close up of a logo

Description automatically generated

*Figure 3: Graph illustrating the rewards for each episode as a measure of the robot’s behavior at the end of the first 100 episodes*

* 1. *Environments containing static obstacles:*

To create an environment containing static obstacles, we added a ball inside the maze as a first step. Once this environment was able to run successfully, we added multiple objects of varying sizes to create the environment. The below figure illustrates an example of such an environment:

A screenshot of a computer

Description automatically generated



*Figure 4: Example of an environment containing static obstacles*

The below RViz graph illustrates how the robot’s lidar scanner behave differently this time because of the static obstacles present in its path. The scan lines seem to be denser or closer because the robot is able to detect the box (marked with a red dot) almost in front of it, as can be seen from the image (on the left bottom of the below figure).

A screenshot of a computer

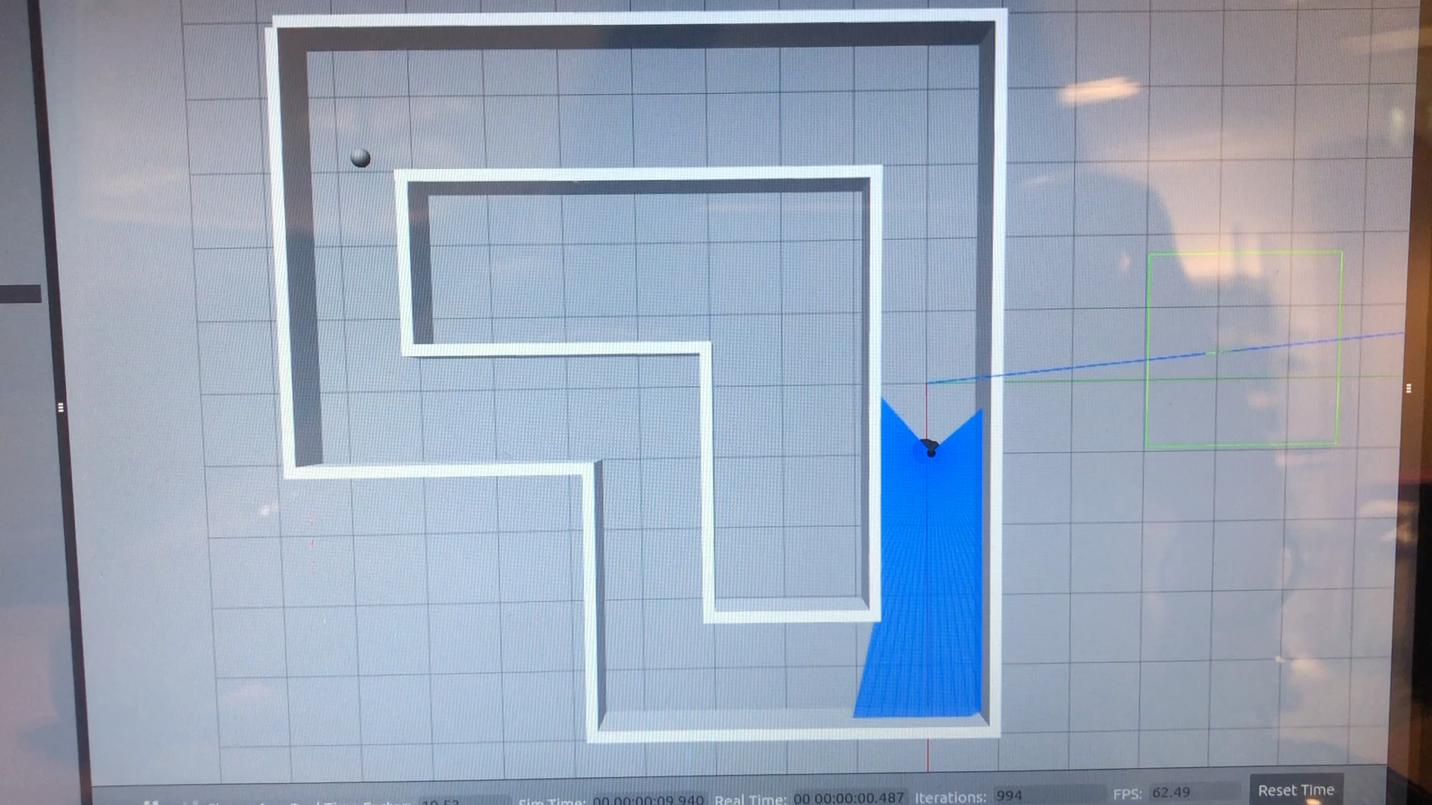
Description automatically generated

*Figure 5: RViz graph for robot navigating in an environment containing static obstacles. The Image graph on the bottom left shows that the robot is able to detect the static box in its path.*

The robot was able to learn to navigate successfully around the maze once in approximately 400 episodes. By the end of 1000 episodes, the robot was able to navigate successfully around the maze a little more than two times.

* 1. *Environments containing dynamic obstacles:*

To create environments containing dynamic obstacles, we had to add the object and enable it to move. While adding the object used the same procedure from above (2.2), making the object move was challenging. We tried various methods, such as using the actor method in Gazebo and coding standalone C++ plugins. However, these methods did not reflect good learning behavior when the control algorithm was run. Another issue was the size of the ball: when we tried resizing the ball to make it smaller, the ball’s gravity somehow got affected and the ball fell through the ground, which caused it to disappear from the environment at random intervals.

The method which finally worked made use of an independently written python code, which manually updates the object’s state every second and publishes this state to the rostopic /gazebo/set\_model\_states. The below video illustrates this environment with the robot running (click to play the video):

*Video illustrating the environment containing dynamic obstacles and the robot’s behavior during training*

Although this ensured that the environment was robust, the control algorithm from the Gym-Gazebo repository still could not produce good learning behavior. The robot seems to learn good navigation behavior until about 100 episodes. However, its learning pattern becomes random and does not build on the learning and experience it has gained in the previous episodes. The below graph illustrates this behavior:

A screenshot of a computer

Description automatically generated

*Figure 6: Graph illustrating the robot’s random learning behavior. The robot earns higher rewards in the first 100 episodes but displays random behavior afterwards, not utilizing any information from the previous 100 learning episodes.*

While we have not been able to identify the exact cause of this problem, we hypothesize that this issue might be occurring either due to a bug in the environment on in the control algorithm. With regards to the environment, as it can be seen in the video, the moving ball has a jerked motion and its position resets every time a new episode begins. This could somehow be affecting the robot’s position or tendency to move further in the maze. With regards to the control algorithm, there might be some parameters which do not encourage the robot to explore further or retain information it has learnt. This might have something to do with experience replay. The robot’s failure to learn effectively could also be attributed to the fact that most control algorithms are fundamentally modelled using human behavior. Humans sometimes tend to make minimal movements when they find themselves unable to cross an obstacle in front of them and this is probably how the robot is behaving as well, which is why it is not able to learn how to move past the ball successfully.

To resolve this issue, we looked into better, more robust control algorithms and found the algorithm used for training the Atari games. Parameters in this algorithm currently do not support very good learning but we believe that altering them, researching more about how the algorithm affects the robot’s behavior, and choosing a more effective policy might improve the robot’s learning behavior.

1. **Extension to the project**

An extension to this project was the transfer of the trained model on the real robot. Theoretically, exporting the ROS\_MASTER\_URI of the system running the simulation onto the system controlling the robot must allow both systems to communicate. An independent python code connecting the input and output topics via a custom node can make the simulation topics and real robot’s rostopics talk to each other. However, while doing this, two rosmasters get initiated.

Therefore, another method we thought of was to import the model architecture and weights into an independent python code, extract the inputs from the real robot’s /scan topic, run them through the trained model, and transfer the twist commands produced by the network to the real robot’s /RosAria/cmd\_vel topic. This code is still in progress and to be tried on the real robot.

1. **Conclusion**

The goal of this project was to develop environments with no, static, and dynamic obstacles, which was successfully completed. Training the robots in these environments was the purpose of creating these environments. While training was successful in environments with no or static obstacles, training the robot in a dynamic obstacle environment was not very effective due to the control algorithm utilized and the fact that a robot’s behavior is modelled largely on human behavior. To be able to establish effective control algorithms would be the next and most important stage of this project and will provide a good foundation for training autonomous agents for dynamic obstacle avoidance.

1. <https://github.com/erlerobot/gym-gazebo> [↑](#footnote-ref-1)