Project 2 Deliverable 3

Zach Mills

***Abstract*— This is the project report/write-up for D3 of Project 2 for CSCI473 Human Centered Robotics, Spring 2020. It describes my approach and my successes and failures.**

I. Sensing Field-Of-View

As was described in the assignment, the robot uses LIDAR to observe its environment. My robot divides its LIDAR field of view into 3 directions that it needs to use to make decisions: left, front, and right. In order to distinguish between these directions, only a portion of the LIDAR data is used for each one. Since the LIDAR data is divided into 360 vertical slices, which are enumerated from 0 to 359, 3 specific slices are used to establish the robot's perception. The 0 slice represents the right of the robot, so that slice is used for sensing to the right. Following this, 90 degrees counter-clockwise, or slice 59, is used for sensing to the front. Finally, 180 degrees, or slice 179 is used to sense the left. These 3 points of perception were sufficient for the robot to successfully navigate the maze in D1, so this is where I began for D2.

II. Discretization Definitions

The following subsections describe the way I defined the three parts of the descrete learning space for the Q-learning algorithm: state, action, and reward.

A. State

In order to define a finite number of states that could be used by the algorithm to learn the maze, I defined a series of substates for each sensing direction, which would allow the continuous LIDAR data to be discritized for use in Q-learning. For the left, front, and right sensing directions, I defined substates representing close, medium, and far distances. The substate of each sensing direction was determined by the LIDAR reading for that direction. Substates were enumerated 0 (close), 1 (medium), and 2 (far), and they were returned by the callback function used by the LIDAR subscriber. States were then stored as combinations of substates, in the order (left, front, right). These tuples were used wherever state was needed. For example, if the robot was really close to a wall on the right, the state would be (2, 2, 0), representing a substate of far for the left, far for the front, and close for the right. These tuple states will later be used in the construction and accessing of the Q-table.

B. Actions

There were many possible ways to define actions, but, in order to keep it simple, I chose to only implement 3 possible actions for the robot. The first action is driving straight forward, which consists of giving the robot a positive y velocity and zero velocity in the x direction. The second and third actions are driving to the left and the right. Because I wanted the robot to always be moving, I chose to make these actions turn while driving, and not turn in place. I did this in hopes that it would prevent the robot from simply spinning in place forever. To implement the turn left action, I provide the robot a positive velocity in the y direction as well as a positive angular velocity. The turn right action similarly consists of a positive velocity in the y direction, but has a negative angular velocity. These actions are enumerated as 0 (turn left), 1 (drive straight), and 2 (turn right). They are enumerated this way for consistency with the enumeration of the states and substates. These actions will later be used in the Q-table with the states.

C. Rewards

The rewards of my Q-learning algorithm are defined in terms of the sensor readings of the robot, just like the states. By default, all combinations of substates give a reward of 0, but each time the robot checks to see if it gets a reward, certain states are rewarded or punished based on the status of one or more of the substates that make them up. For example, any state where the left substate is 0, meaning the robot is close to a wall on its left, or the front substate is 0, meaning it is close to a wall on the front, is given a punishment with a reward of -1. Next, any state where the right substate is 1, representing a medium distance, and the front is not 0, representing a close distance, is given a reward value of 1. This encourages the robot to stay close to the right wall, but not get close to other walls. The front wall is especially important to avoid, because the robot is always moving forward, so colliding forwards is the worst situation it can get into. These rewards are also used in the Q-table

III. Q-table

A. Definition

My Q-table is defined as a dictionary where the value is the Q-value for a particular state-action pair. The key to this dictionary is a tuple, where the first value is the state (also represented as a tuple), and the second value is the action. This means the dictionary keys follow a format similar to ((0, 1, 2), 2), which in this case would represent turning right when you are close to a wall on the left, a medium distance from a wall in front, and far from a wall on the right.

B. Initialization

Initially, a looping structure is used to set the Q-value for all state-action pairs to 0. Then, the prior knowledge is

added. This prior knowledge is equivalent to the Q-table that I designed in D1, which allowed the robot to make correct decisions for at least one example of each type of situation in the maze. The prior knowledge provides incentives for the robot to approach the right wall when it is too far, avoid the right wall when it is too close, avoid the left wall when it is too close, and to turn left when the front wall is too close. It also incentivizes going straight when the right wall is a medium distance and neither of the other readings are too close. I decided that using the full Q-table from my D1 as prior knowledge for the robot would give it the best chance of successfully learning the maze.

IV. Model Parameters

A. Epsilon, ϵ

In my algorithm, ϵ represents the probability that any given decision will be made according to the Q-table, which is the inverse of the probability that the decision will be made randomly. ϵ starts out very low in my algorithm, very close to 0 in fact, but it quickly approaches 0.9 with an inverse exponential function each episode. Eventually, the robot will be making random decisions only ten percent of the time in training.

B. Learning Rate, α

The learning rate, α, in my algorithm is equal to 0.2

C. Discount Factor, γ

The discount factor, γ, in my algorithm is equal to 0.8

V. Training

A. Episodes

My algorithm generally required far more episodes than I had hoped, around 500 episodes, before it started to perform adequately. Even with significantly more than that, around 2000, it never became perfect.

B. Time

The time required by each episode tended to increase as the training process went on, so 500 episodes could take quite a while. Generally, my algorithm was able to get to 500 episodes in a round an hour.

VI. Performance

My robot never became very good at the maze, and it generally crashed before encountering all of the 5 situations described in the project assignment. However, it was very good at the 180 degree turns, and almost always got out of them. The hardest part to get right was turning around corners, because I had chosen not to implement a sensor that sensed right-forward. That said, my final implementation was successfully able to navigate the 180 degree turns around I-shaped corners. The biggest problem with my robot was that it had a tendency to turn toward a wall it is following randomly. I am not sure what its sensors were seeing that caused it to do this, and this part of its training was definitly given a punishment by the reward system, so I am not sure why it learned to do this. This is what causes my robot to not be able to complete the maze.