Solving a Maze with a Wall Following Robot Using SARSA

Chance Cardona[1]

***Abstract*— The purpose of this project was to implement a reinforcement learning algorithm that would enable a robot to be able to traverse a maze. Q Learning can be utilized to make a robot learn how to perform on an arbitrary maze if the appropriate parameters can be provided. For this project a Triton robot with a 360° LIDAR equipped. The experiments and training were performed in a simulated maze with Gazebo using the stingray-sim package. The LIDAR was divided into multiple domains which were then assigned different states depending on their minimum value detected in the domain. The specific implementation was a SARSA algorithm with a greedy decayed epsilon policy. The best results were found with parameter values as follow: Learning Rate of 0.2, Discount Factor of 0.8, and a decayed epsilon with an initial value of 0.9 with a decay rate of 0.985/episode. Finally, an Orientation parameter in the model's state was found to be necessary for proper training.**

I. Introduction

The objective of this project was to make a reinforcement learning algorithm that would teach an autonomous robot how to follow a wall and avoid obstacles. This would be used to have it complete a maze to test its ability and the algorithm's performance. This means that the ideally trained robot would first seek out a wall and then maintain a set distance d from the wall, performing the correct action when it comes across different corner types. The RL algorithm used was SARSA, short for "State, Action, Reward, (future) State, (future) Action". SARSA is an on-policy learning algorithm used to optimize a Markov decision process. Each time an action is made SARSA updates the Q table (a table of rewards for each each possible action that can be taken for each state) according to the equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

Where s_t, a_t, r_t is the state, action, and reward at time t. Here state is defined as the tuple of discretized LIDAR sensor values for the domains: Right, Front-Right, Front, and Left. This is listed in more detail in the Approach section. To create a finite Q table the minimum value from each of these distances were further discretized into the vales "Too-Close", "Close", "Medium", "Far", and "Too-Far" according to the slides as recommended by TA Qingzhao Zhu. The action space used was Go Forward, Turn Right, and Turn Left, with all of these having $v_x = 0.3$. The actions are chosen at each iteration of SARSA using a "decayed greedy ϵ" policy, which means that the policy begins by making mostly random actions and as the number of training episodes increases it slowly changes that to make the action with the maximum Q value for that state.

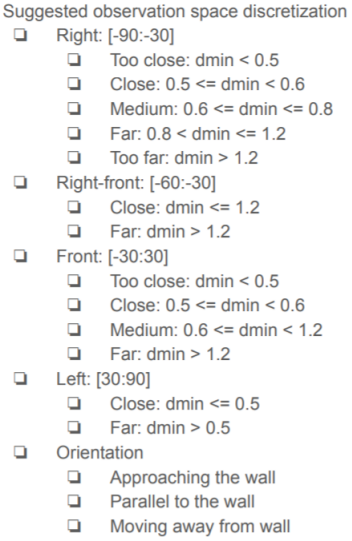

Fig. 1. LIDAR Scan sweep values used to generated R, FR, F, and L domains.

II. Approach

The algorithm was implemented in Python, with the states being described as a tuple of LIDAR ranges with discretized distances as seen in figure 1, obtained from Qingzhao's slides. To create the Q table a dictionary using these tuples as "keys" was used. The "value" was a numpy array with a length of the action space. The program could be ran in 2 modes: "train" and "test". Train would first initialize the Q table so the robot could follow a straight wall. This was accomplished by initializing the Q table with -0.5 for all values except for the elements corresponding to the Forward action, which received a value of -0.4 (encouraging the robot to move forwards initially). It was also initialized with the value -0.1 to Left if the R domain was close, and -0.1 to Right if the R domain was far. This was done to increase the results of the training and decrease training time.

Next, the SARSA algorithm was called in a series of episodes. For each episode the position of the robot was randomized to the center of any square within the maze. It would then use the SARSA algorithm and go through the maze. If the robot became stuck or tipped over (from running into a wall), the episode would end. The maximum step size

*This work is part of Colorado School of Mines CSCI 473 Human Centered Robotics

[1]Chance Cardona is an undergraduate student with the Faculty of Engineering Physics and Computer Science at Colorado School of Mines, 1500 Illinois St, Golden, Colorado. ccardona@mines.edu

Hyperparameter	Value
α	0.2
γ	0.8
ϵ	$0.9d^n$
d	0.985

TABLE I

HYPERPARAMETER VALUES FOR SARSA, WITH N BEING THE NUMBER OF EPISODES AND D IN THIS CONTEXT BEING THE DECAY CONSTANT.

of the episodes was 4000 steps, and a "good policy" was considered to be found if the robot wasn't too close or too far from the right hand wall for over 1000 steps. This was done at a loop rate of 10Hz, so about 10 actions were theoretically called per second (theoretically due to unknown constraints with the functions used to get the robots current position and time taken to run the loop each time). The 'next' action and state (t+1) was actually the current action (measured after the policy was called), and the 'current' action and state (t) were the previous ones, measured the loop before. After each Q table assignment these values were then updated.
For the Hyperperameters for SARSA, these values were assigned as listed in Table I.

The reward function utilized was simply
{-1 if: R, or F were too close, or if L was close, and 0 else}.
To check if the robot was stuck or to randomize its position, Gazebo's getModelState and setModelState were used.
For the Test mode, the Q table was loaded from memory from the training and then ran using a fully greedy policy. If no Q table could be found it would use the pre-initialized Q table instead.
Due to many unsuccessful runs, a Linear Regression was implemented on the LIDARs values for the right hand side of the robot to add the "orientation" to the state tuple. Initially the robot was moving along its y-axis, so this range being regressed was -30:30°, however this caused many errors because of a discontinuity between the last and first index of the LIDAR ([-1] and [0]). Due to this, the robot was eventually changed to move along its x-axis and the final domains (Figure 1) were used. This fixed the discontinuity in the right domain for the Linear Regression. After this was fixed it could then be observed that the data needed to be converted from polar into cartesian for the regression to work. See Figures 2 and 3 for before and after this transformation. Next, the Linear Regression was translated into an Orientation by this criteria: if R^2 was $<$ 0.5 Orientation was considered undefined. Else, if r was $>$ 0.5: if slope was greater than 0.003 the robot was considered approaching the wall, if slope was less than -0.003 the robot was leaving the wall, and in between these 2 values it was considered parallel to the wall.

Please see the "Attemped Solutions" portion of the Appendix for other failed attempts and more of a progression from these failures to the current working model.

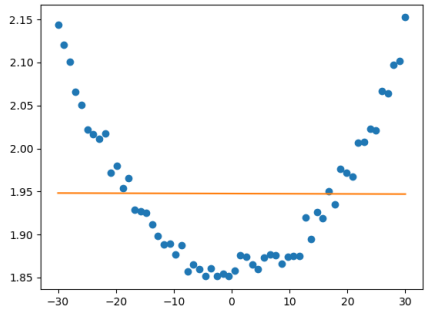

Fig. 2. Raw data from LIDAR scan along with its Linear Regression. Very low R^2 confidence.

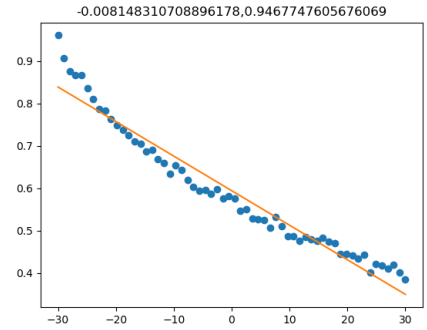

Fig. 3. Transformed (from polar into cartesian) data from LIDAR scan along with its Linear Regression. High R^2 confidence.

III. RESULTS

With the Orientation successfully implemented, the robot was trained with the further restraint that the computer never went to sleep and Gazebo's getModelState never hung up for more than 1 second after training began. After about 60 runs or 1.5 hours the robot displayed clear improvement and was able to navigate any corners it came across, only hitting corners occasionally. It was then allowed to train for 3 more hours to complete the 150 episodes. These results can be seen on youtube at: .
Figures 4, 5, 6, and 7 provide screenshots showing the robot navigating through all 4 types of turns present in the maze, and serve as experimental evidence of the SARSA training's success.

IV. CONCLUSION

I believe that the safe conclusion one can develop from this experiment is that although Q-learning can be a powerful algorithm, it can also be very finicky and one must first understand every aspect of the infrastructure under it before utilizing it. This is mainly in regards to the LIDAR sensor values (discontinuities and where the lidar is "active"), ROS rates (how long does it take to update, what speed is the subscriber getting information at, etcetera), and service delays (if Gazebo stopped running and didn't present an error the training would be thrown off). With the appropriate problem

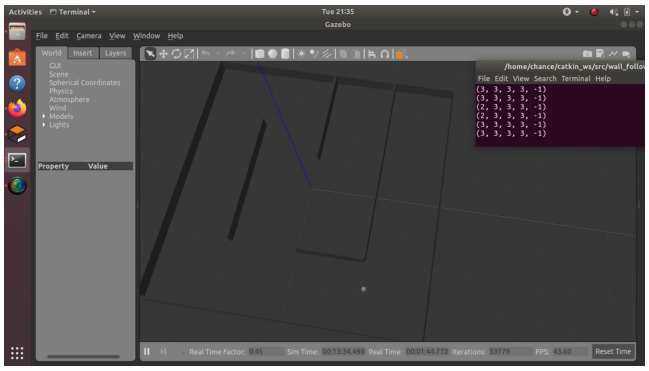

Fig. 4. Robot as seen in Gazebo turning Right 90 degrees at an L shaped corner.

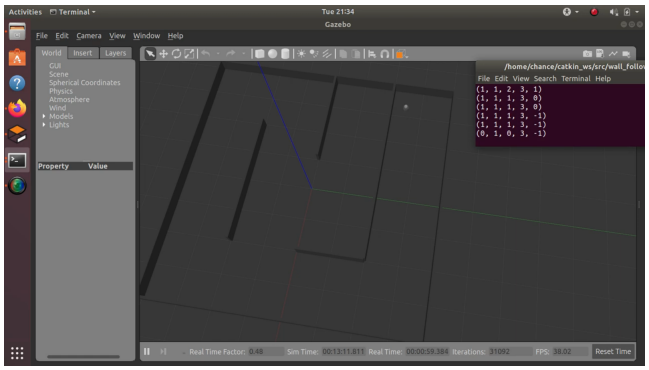

Fig. 5. Robot as seen in Gazebo turning Left 180 degrees at a U shaped corner.

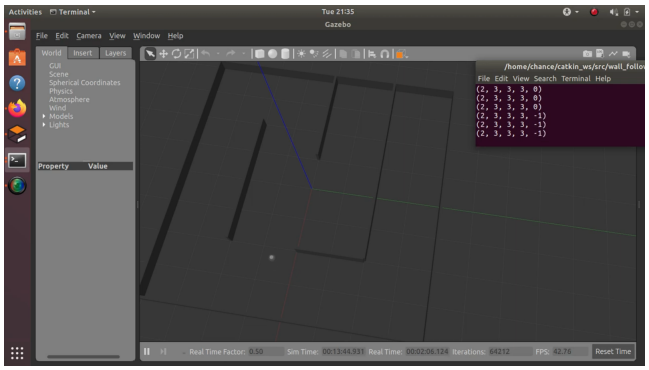

Fig. 6. Robot as seen in Gazebo turning Right 180 degrees at an I shaped corner.

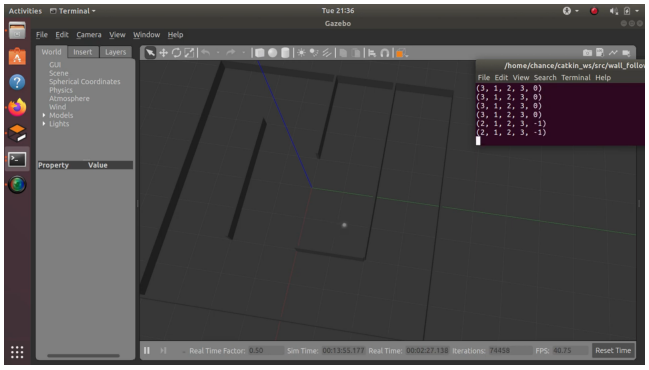

Fig. 7. Robot as seen in Gazebo turning Left 90 degrees at an L shaped corner.

spaces and approaches, Q learning can be a very powerful and somewhat efficient approach to solving a Markov decision policy. For future work hierarchical methods of Q learning may be utilized to create more complex tasks and work on improving the time taken for the robot to train and navigate the maze could be improved.

APPENDIX

Attempted Solutions

This section lists solutions that were attempted prior to the final working model with the working linear regression.

The initial training (before an orientation state was added) did not yield positive results. The robots performance decreased with more training. Many techniques and modifications were attempted (unsuccessfully) to implement this:

The domains used to divide up the LIDAR values were initially as follows: right -15:15°, front-right 30:60°, front 75:105°, and left 120:180°. These values were played with a lot initially trying to get the robot to work. Left was expanded and decreased between a sweep of 15° total to 60°. The front domain was changed to -10:10° to be tighter as this would make front more focused directly ahead of it, reducing errors from say a wall off to the front side of the robot. These values were also rotated 90° during the implementation of the linear regression. The final values listed in the Approach section provided the best overall results.

In addition to domain changes, multiple action spaces were tried, from the one listed in the main body to the one referenced by [1]. None of these helped; however the latter did enable the robot to learn further, yet it instead began to simply go around in circles instead of following a wall. It is believed that this is due to some local minimum caused by this giving the reward value a “good” outcome some of the time. The current action space seemed to train more reliably and with less episodes needed, and is the final version.

Multiple Q table initializations were tried. First they were all initialized to 0, yet this did not achieve any results when trained. Next it was tried with the wall following procedure as listed above, which gave better results and looked promising yet still yielded no good results. On the contrary, Training (without the orienation state) appeared to make the robot perform worse after 150 episodes. A Q table that would allow the robot to follow a wall was finally chosen in the working version of the code.

Initially, simple Q-learning was used to train the code, however this was changed to SARSA since it yielded better results in the long run.

ACKNOWLEDGEMENTS

Special thanks to Qingzhao Zhu for the bulk of suggestions and guidance, and to Luke Drong for a few recommendations for where to go next, in addition to ROS launch tips.

REFERENCES

[1] Moreno, D.L., Regueiro, C.V., Iglesias, R. and Barro, S., 2004. Using prior knowledge to improve reinforcement learning in mobile robotics. Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK.