Autonomous Robotic Wall Following Utilizing Q and SARSA Reinforcement Learning Algorithms

Anthony Olvera[1]

***Abstract*— In this paper we discuss and implement multiple reinforcement learning algorithms for the purpose of teaching a robot to learn how to autonomously follow along a wall with no need for human intervention. Using ROS a Triton robot is utilized inside the gazebo simulator and trained using both q and SARSA reinforcement learning policies. The robot is equipped with a 3D laser scanning Li-DAR sensor which provides 360 degree range data that allows us to construct a state model in terms of the distances to the surrounding walls. This state model is used for both algorithms. We define a set or actions the robot can take and a set of rewards given for taking the appropriate action. The robot was able to learn how to follow along the wall using both algorithms. The robot was also able to turn corners of both 90 and 180 degrees while moving along the wall. We found that the SARSA method converges faster, needing fewer episodes for training than the traditional off-policy q-learning method. We also found that the SARSA method outperforms the q-learning method in terms of robot performance while following along a wall, finding the wall faster and moving along a straighter path. All code can be found at `https://github.com/arolvera/RobotWallFollowingUsingQLearning.git`**

I. INTRODUCTION

In this work a mobile robotic platform was programmed in simulation to learn how to follow along a wall using q-learning and SARSA (state action reward state action) policies [1]. Following training, the performance and outcomes of the two algorithms were evaluated and compared. We utilize a Triton robotic platform. Triton is an omi-directional robotic platform developed by the Colorado School of Mines for research purposes. The robot is equipped with a Li-DAR 3d laser scanner which gives 360 degree range data. The Triton was programmed using the python programming language managed by ROS (Robot operating system) and all experiments were ran in the gazebo simulator.

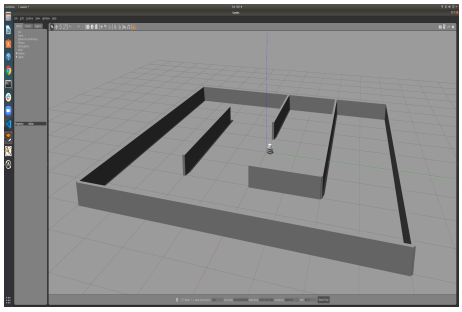

Fig. 1. Triton robot and simulated maze environment

[1]Anthony Olvera is a student in the Computer Science Department, Colorado School of Mines, 1500 Illinois St, Golden CO, 80401, USA `aolvera@mines.edu`

ROS is an open source framework for development of robotic software and testing [2]. Gazebo is a simulator that integrates well with ROS and provides many experimental environments useful for testing. The robot navigated in a maze environment containing multiple types of walls ranging from inside corners, outside corners, 180 degree I shaped turns and U shaped corridors depicted in figure 1. Using the laser scan data we developed a model to depict robot state. We then define a set of actions the robot can take at any given time step. Lastly a set of rewards are defined and given to the robot dependant on the given action it takes when in a particular state. The rewards are stored in something called a q-table which the robot will update after every iteration with the new reward value. There are multiple methods possible for how the robot selects its actions. In this approach we utilize an epsilon greedy policy for action selection [3]. We define a parameter epsilon which decreases over time allowing the robot to leverage exploration of new state-action pairs which may provide higher reward and exploitation on the actions the robot has already learned return high rewards. The result is, after many iterations, the robot will learn the behavior in which we intended. Both algorithms were implemented and the simulation was allowed to train for 30000 one-half second time steps for a total of 4 hours and 10 minutes. For both methods the robot was able to learn how to follow along a wall on the right-hand side and also was able to handle all the corner conditions mentioned earlier. The SARSA method was shown to converge faster than q-learning alone, following the wall more efficiently and not needing to complete as many episodes in the same amount of time as the q-learning method. We cover the model, design and results in greater detail in the following sections.

II. APPROACH

Reinforcement learning is branch of machine learning in which an agent is given the freedom to take an action in its environment which alters its state. The agent then receives some reward for taking that particular action. Over some time the agent can learn to take the appropriate actions to maximize reward and learn some intended behavior. Q-learning is a popular off-policy reinforcement learning algorithm. Off-policy meaning it learns the value of the optimal policy independently of the agent's actions. On the other hand. SARSA is a on-policy variation to Q learning, meaning it learns the value of the policy being carried out by the agent, including the exploration steps to find a policy that is optimal, taking into account the exploration inherent

in the policy [4]. Both methods were implemented in this work.

A. Design

For any q-learning or SARSA policy a discrete set of states need to be defined for which the robot can occupy at any given time. Typically in robotics we define robot kinematics in terms of pose (position and orientation). Pose is defined by a rigid bodies translation and rotation with respect to some reference frame. More specifically we are interested in where the robot is in the Cartesian coordinate frame and what is its angle with respect to that reference. Because Triton is a wheeled platform, this problem is simplified to translation only along the xy-plane and rotation only along the z axis. Using the laser scan data from the 360 degree Li-DAR on the Triton we can define the robots pose with respect to the surrounding walls. Figure 2 depicts the model that was utilized for this work and credit is given to [5] for defining this model.

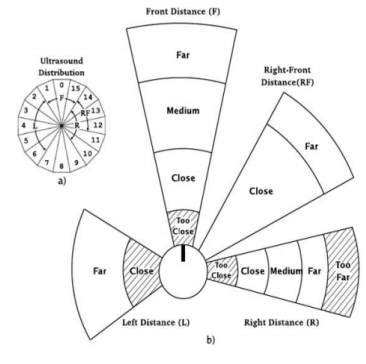

Fig. 2. The state model defined for this work

The distances were defined as follows.

Too Close	d $\leq$ 0.5m
Close	0.5m $<$ d $\leq$ 0.6m
Medium	0.6m $<$ d $\leq$ 0.8m
Far	0.8m $<$ d $\leq$ 1.2m
Too Far	1.2m $<$ d

The left and front sectors span sixty degrees while the right and right-front sectors span thirty degrees. The distance is defined as the minimum distance to the nearest wall for the sector which intersects a given region. In addition, we define state in terms of the robots orientation in terms of the angle θ between its direct right-hand side and the point of minimum distance to a wall.

Approaching	$5^\circ < \theta \leq 180^\circ$
Parallel	$355^\circ \leq \theta \leq 5^\circ$
Leaving	$180^\circ < \theta < 355^\circ$

The set of possible states the robot can occupy is the number of unique combinations of each of the above regions for each of the four sectors and three orientations listed above. This gives a total of $2 \times 4 \times 2 \times 5 \times 3 = 240$ states. In addition we define three possible actions the robot can take after observing its state. These actions are some combination of linear and angular velocities the robot can execute. These are summarized below.

	Linear m/s	Angular rad/sec
Forward	0.3	0
Left	0.3	$\pi/4$
Right	0.3	$-\pi/4$

Given each of these three actions our q-table is defined in python as a nested dictionary with each of the 240 state combinations and the above three possible nested actions as the keys, giving $240 \times 3 = 720$ total state-action pairs. The dictionary is initialized with all reward values set to zero. Lastly the reward policy was defined as follows.

$$R = \begin{cases} -1 & \text{If right = too close} \vee \text{right = too far} \\ & \vee \text{ left = close} \vee \text{front = too close} \\ 0 & \text{any other situation} \end{cases}$$

B. Learning Approaches

As mentioned, the q-learning algorithm is an off policy reinforcement learning method. A q-table is defined for each discrete state-action pair with all rewards in the table initialized to zero. During training The robot uses an epsilon greedy approach to determine whether to explore or exploit on actions. We decrease epsilon during training using the following formula $\epsilon = \epsilon_0 d^n$, where ϵ_0 is set to 0.99 d is also 0.99 and n is the episode number. We generate a random number r between 0 and 1 and compare it with ϵ. If $r > \epsilon$ we select the action from the q-table with the highest reward, otherwise we select a random action, execute it, receive an immediate reward and update the q-table for the given observed state transition. The q-table is updated using the bellman equation.

$$\underbrace{Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \alpha[\underbrace{R(s,a)}_{\text{Reward}} + \gamma \overbrace{\max Q'(s',a)}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

Learning rate

Discount rate

In contrast to q-learning which chooses actions in a greedy fashion, SARSA chooses actions based on a certain policy. For this, an epsilon greedy policy was also utilized. However a fixed ϵ of 0.9 was used. The equation for table update is as follows.

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma Q'(s',a') - Q(s,a)]$$

For both algorithms a learning rate of $\alpha = 0.2$ and discount rate of $\gamma = 0.8$ were used.

III. EXPERIMENTAL RESULTS

Training for each algorithm was implemented and ran in simulation for two different methods. First an open loop training was ran for 30000 one-half second time-steps for a total of 4 hours and 10 minutes and the accumulated reward for each episode was collected. A comparison of accumulated rewards for q-learning and SARSA can be seen below. In the second training method was ran indefinitely until the maximum number of time-steps in a single episode reached 1000 with no termination. An episode is defined as the sequence of time-steps for which the robot does not get stuck and the training is not complete. The robot was determined to be stuck when its motion in the xy plane has not exceeded a threshold of 0.05 meters within the past three time-steps. In this case the episode would terminate. For the first training method the q-learning algorithm reached a maximum number of 1837 time-steps in a single episode whereas the SARSA algorithm reached a maximum number of 2984 time-steps in a single episode. This is an indicator that SARSA method likely learned a more effective policy than the q-learning method given the same amount of time. In the second training method the q-learning approach completed an episode of at least 1000 time-steps at time-step 19478 or 2 hours and 42 minutes into training. In comparison the SARSA method only took until time-step 12655 to reach at least 1000 time-steps in a single episode, approximately 1 hours and 45 minutes into training. This indicates that the on-policy SARSA method converges faster and the results can be seen below in figure 3. The reason for the uniformity is because as the training progresses the episodes become longer in terms of time-steps, thus the total accumulated negative reward may accumulate less frequently however over longer intervals of time. In total the q-learning method staged a total of 525 episodes for training whereas SARSA staged only 326 episodes within the 30000 one-half second time-step training interval.

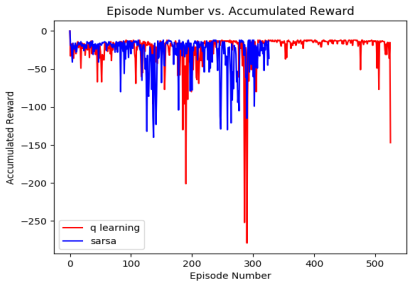

Fig. 3. Accumulated reward, q-learning vs. SARSA

Although SARSA performs slightly better, after training, both methods were able to achieve proficient performance. They were able to navigate the maze maneuvering around 90 degree, inside and outside L shape corners and 180 degree I shaped and U shaped corners and also strait walls. See figure 4 below.

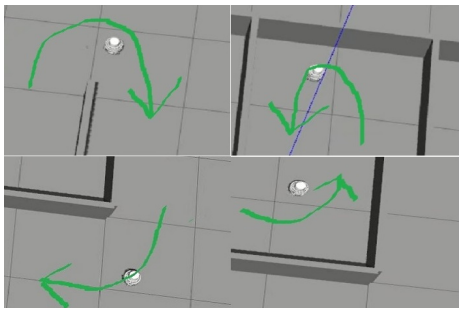

Fig. 4. Robot maneuverability after training

IV. CONCLUSION

In this work we examined two reinforcement learning algorithms an their effectiveness on a robotic platform for autonomous navigation by means of following along a wall. The Triton robotic platform was utilized in the gazebo simulation provided through ROS for experimentation. A 360 degree Li-DAR sensor providing range data about the robots surroundings allowed us to construct an intuitive design of a state model given distances to the walls in certain regions surrounding the robot. After careful consideration in the design of state, action and reward we implemented both the off-policy q-learning and on-policy SARSA reinforcement learning algorithms and examined their performance both during and after training the models. We discovered both algorithms to be effective in accomplishing the wall following task after the training was complete. However during training we discovered the on-policy SARSA learning method to be more efficient, converging faster than the off-policy q-learning method. In future work we will consider using an on-policy method to eliminate training time. This could be incredibly useful in a broad range of applications with much larger state spaces, such as robots that navigate in more complicated and dynamic environments such as crowded spaces or outdoor environments. We also discovered that the SARSA method learned a more effective policy after training than the q-learning method. The robot was able to find the wall more quickly and follow along it in a more direct path. During this work we discovered the importance of defining state, and how a good and bad design can make a huge difference in the effectiveness of the algorithm. Having a greater number of states can enhance the performance but will have a negative impact on the training time of the model. Likewise defining more actions can also enhance the robots performance at the expense of more time or computational cost for model training. Lastly careful consideration needs to be taken when defining reward. We discovered appropriate and broad reward definition will have a positive impact on learning time, in addition and if practical pre-defining

as much of the q-table as possible will also decrease the required time too learn an effective policy.

REFERENCES

[1] Corazza, Marco and Sangalli, Andrea, Q-Learning and SARSA: A Comparison between Two Intelligent Stochastic Control Approaches for Financial Trading (June 10, 2015). University Ca' Foscari of Venice, Dept. of Economics Research Paper Series No. No. 15/WP/2015, Available at SSRN: https://ssrn.com/abstract=2617630 or http://dx.doi.org/10.2139/ssrn.2617630

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

[3] Dabney, Will et al. "Temporally-Extended epsilon-Greedy Exploration." arXiv: Learning (2020): n. pag.

[4] https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c

[5] 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.