Solving the Mobile Robot Wall-Following Problem by Q-Learning*

Xinlei Zhang[1]

***Abstract*— This reports presents a Q-learning approach to solve the autonomous wall-following problem for a mobile robot, Triton, with only a 2D 360° LiDAR sensor. With the learned Q-table, the robot is able to maintain an optimal distance from walls while navigating diverse scenarios, including straight wall following and executing 90° and 180° turns in simulation testings.**

I. Introduction and Problem Description

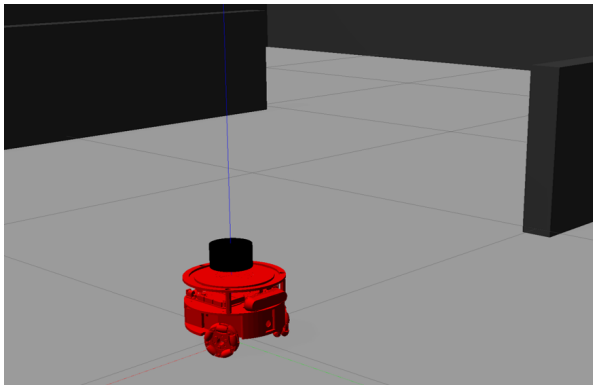
Mobile robot navigation in unstructured environments remains a significant challenge, particularly when safe and efficient maneuvering around obstacles is required. One well-established strategy is wall-following, where a robot maintains a constant, optimal distance from walls to ensure both safety and effective mapping of its surroundings. In this project, our objective is to train a mobile robot, Triton (as shown in Fig. 1a), with only a 2D 360° LiDAR sensor, to autonomously follow walls and negotiating corners (as depicted in Fig. 1b) of different shapes using the Q-learning approach.

Equipped with a 2D LiDAR sensor, Triton captures a full 360° scan of its environment. The sensor data are processed to discretize the surroundings into a finite set of 81 states, defined by three regions (front, front-right, and right) and the relative wall orientation. The robot's action space consists of nine discrete pairs of linear and angular velocities, enabling controlled maneuvers for wall following and negotiating corners with different shapes. A ϵ-greedy policy and temporal difference update based Q-learning algorithm is employed to update the Q-table, thereby learning an effective wall-following policy (i.e., Q-table) that generalizes across different scenarios. This report describes the detailed methodology, simulation setup, learning results and performance evaluation.

II. Methods

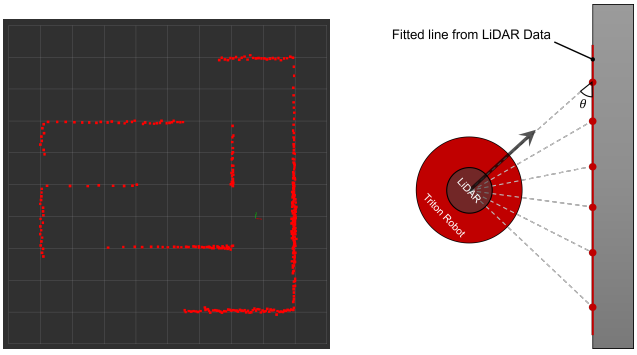
A. Sensing from 2D LiDAR and Preprocessing

The Triton robot uses a 2D LiDAR sensor to obtain a comprehensive 360° scan of its surroundings (see Fig. 2a). For the wall-following task, the sensing field is segmented into three regions: *front*, *front-right*, and *right*. The minimum distance values within these regions are extracted and discretized based on predefined thresholds to effectively capture proximity information. Furthermore, the robot estimates its relative orientation with respect to the wall on its right-hand side by applying a linear regression method to the LiDAR data (illustrated in Fig. 2b). In the orientation estimation process, only distance measurements below 3 m are considered.



(a) (b)

Fig. 1: (a) Visualization of the Triton robot in Gazebo; (b) Overview of wall-following and corner-turning tasks.



(a) (b)

Fig. 2: (a) LiDAR measurement visualization in RViz; (b) Illustration of relative orientation estimation using linear regression.

B. State, Action, Reward Definition

State: The robot's state s is determined by four elements: the discretized distance measurements from the front,

*This work was a course project for CSC 791: Advanced Robotics, 2025 Spring, North Carolina State University.

[1]Xinlei Zhang is a Ph.D. student with the Department of Mechanical and Aerospace Engineering, North Carolina State University, USA xzhan245@ncsu.edu

TABLE I: States and Actions Values

State (s)	Front Distance	Front-right Distance	Right Distance	Relative Orientation
Values	`Close`	`Close`	`Close`	`left_biased`
	`Good`	`Good`	`Good`	`aligned`
	`Far`	`Far`	`Far`	`right_biased`

Action (a)	Linear Velocity	Angular Velocity
Values	0.2	$-\pi/4$
	0.5	0
	0.8	$\pi/4$

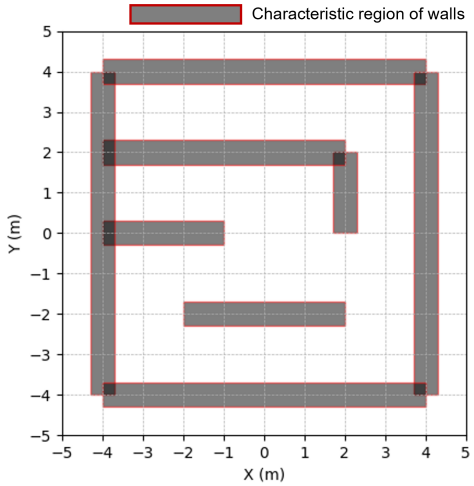


Fig. 3: Inflated wall regions used for collision detection.

front-right, and right regions (each categorized as `Close`, `Good`, or `Far`), and the relative wall orientation (categorized as `left_biased`, `aligned`, or `right_biased`). This yields a total of $3 \times 3 \times 3 \times 3 = 81$ possible states in the state space $\mathcal{S}$. The thresholds for distance are defined as follows: `Close` for distances in $(0, 0.5)$ m, `Good` for $[0.5, 1.0]$ m, and `Far` for distances greater than 1.0 m. The relative orientation categories are defined over the intervals $(-\frac{\pi}{2}, -\frac{\pi}{8})$ for `left_biased`, $[-\frac{\pi}{8}, \frac{\pi}{8}]$ for `aligned`, and $(\frac{\pi}{8}, \frac{\pi}{2})$ for `right_biased`.

Action: The action space $\mathcal{A}$ consists of 9 discrete pairs of linear and angular velocities. The linear velocity v is selected from $\{0.2, 0.5, 0.8\}$ m/s, and the angular velocity ω from $\{-\pi/4, 0, \pi/4\}$ rad/s. Zero linear velocity is excluded to prevent the robot from stopping, thereby ensuring continuous movement during learning.

Reward: The total reward R_{total} for a given state is computed as the sum of rewards from proximity, wall-following consistency, and collision penalties:

$$R_{\text{total}} = R_{\text{proximity}} + R_{\text{wall\_following}} + R_{\text{collision}}, \qquad (1)$$

where $R_{\text{proximity}}$ rewards optimal distance and alignment (e.g., +10 for `Good` readings with `aligned` orientation, and -5 for `Close` readings), $R_{\text{wall\_following}} = 10\times$ (number of consecutive wall-following steps), and $R_{\text{collision}}$ applies a penalty of -10000 if a collision is detected. For collision detection, walls are inflated by 0.3 m on both sides to account for the robot's footprint (see Fig. 3).

C. Q-Learning Algorithm

Q-table Definition: The Q-table is defined as an 81-by-9 matrix, where each entry $Q(s, a)$ represents the expected cumulative reward for taking action a in state s. Initially, the Q-table is set to zero or can be initialized with expert knowledge.

Q-Learning Update Rule: The Q-learning algorithm updates the Q-table using the temporal difference rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right],$$

where s and s' are the current and next states, a is the selected action, r is the reward received, α is the learning rate, and γ is the discount factor. An ϵ-greedy policy is employed to balance exploration and exploitation, with the exploration rate ϵ decaying exponentially after each episode.

Termination and Convergence Conditions: An episode terminates if the robot collides with a wall, the number of steps exceeds a preset maximum, or when the cumulative reward converges beyond a defined threshold. Convergence is determined when the cumulative rewards of the last five episodes exceed the threshold.

The pseudo-code of the Q-learning algorithm for this work is provided in the Algorithm 1 below.

Algorithm 1: Q-Learning with Epsilon-Greedy Policy and Temporal Difference Update

Input: Initialized Q-table $Q(s, a)$, learning rate α, discount factor γ, decay rate β, and initial exploration rate ϵ
Output: Trained Q-table $Q(s, a)$

for *each episode* **do**
  Initialize state s and reward history `r_history`
  while *termination conditions are not met* **do**
    $\epsilon \leftarrow \epsilon \times \beta^{\texttt{episode\_count}}$
    Draw a random number r from [0,1]
    if $r > \epsilon$ **then**
      $a \leftarrow \arg\max_{a'} Q(s, a')$ `// exploit`
    else
      $a \leftarrow$ a random action from $\mathcal{A}$ `// explore`
    end
    Execute action a and observe reward r and next state s'
    Update: $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
    $s \leftarrow s'$
    Append r to `r_history`
    Check termination conditions with s and r
  end
  Check convergence using `r_history`
  if *convergence not achieved* **then**
    continue
  else
    break
  end
end

III. Simulation Study

This section details the simulation environment setup, the learning of the Q-learning algorithm, and the evaluation of the learned policy across various tasks.

A. Simulation Setup

The simulation is conducted using ROS Noetic and the Gazebo physics simulator with the Triton robot. The robot receives LiDAR data via the ROS topic `/scan` and velocity commands through the ROS topic `/cmd_vel`. The robot's position is monitored using the ROS service `/gazebo/get_model_state` for collision detection, while services such as `/gazebo/reset_simulation`, `/gazebo/pause_physics`, and `/gazebo/unpause_physics` are used to reset the simulation between episodes. The initial position of Triton is set via the service `/gazebo/set_model_state`.

B. Q-Learning Setting and Results

The Q-table, with dimensions 81 (states) by 9 (actions), is initially set to zero in the learning process. The hyperparameters used in the Q-learning algorithm are summarized in Table II.

TABLE II: Hyperparameter Settings for Q-Learning

Hyperparameter	Value
Learning rate (α)	0.2
Discount factor (γ)	0.9
Initial exploration rate (ϵ)	0.9
Decay rate (β)	0.985
Cumulative reward threshold	50000
Maximum steps per episode	50000

Cumulative Reward vs. Episode: Fig. 4 shows the cumulative reward per episode. The rewards of the first 300 episodes are close to -10000, since all of these episodes end with collision and get the collision penalty (-10000) at the last step.

Q-Table Evolution: Fig. 5 illustrates the evolution of the Q-table during learning, highlighting the action with the highest Q-values for each state.

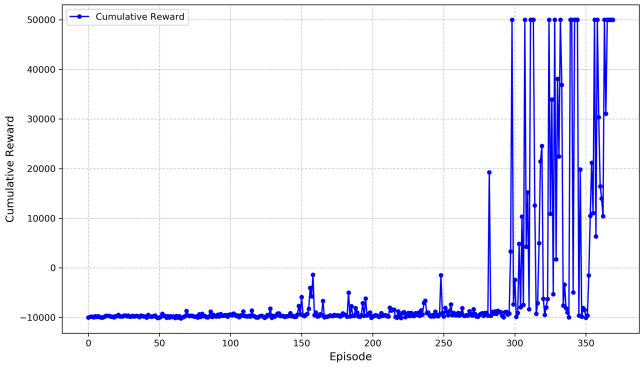


Fig. 4: Cumulative reward versus episode.

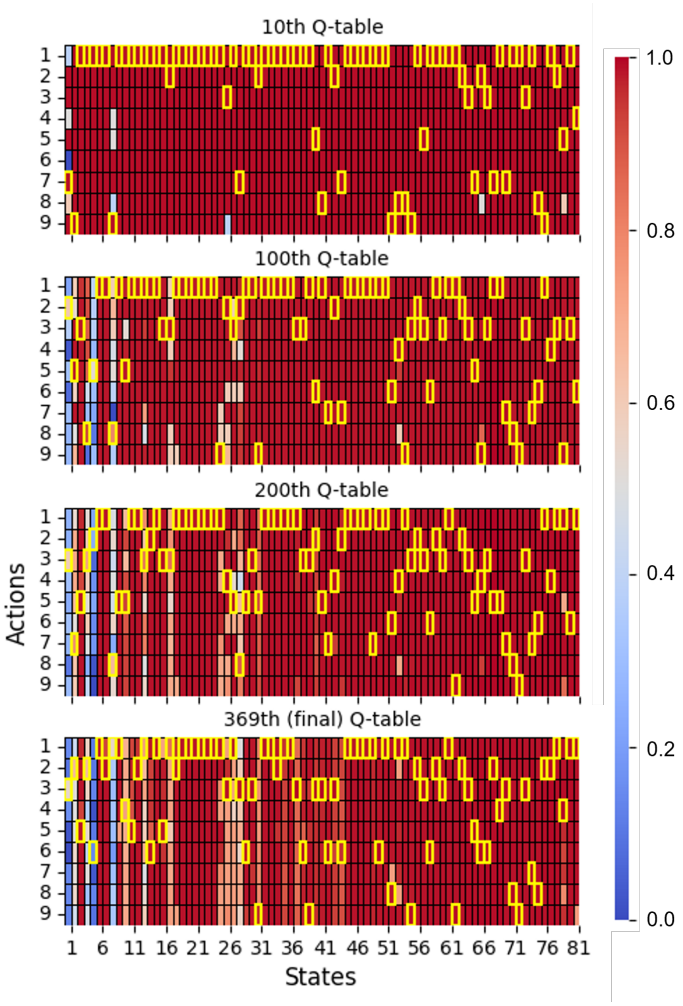


Fig. 5: Evolution of the Q-table during learning; the highest Q-value for each state is highlighted.

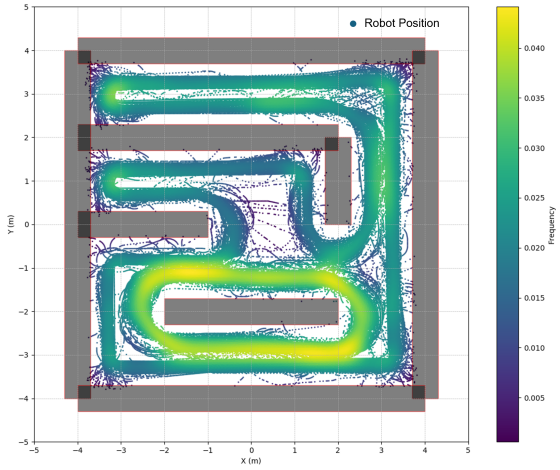


Fig. 6: Robot trajectory during learning; light colors indicate high-frequency regions, while dark colors represent low-frequency areas.

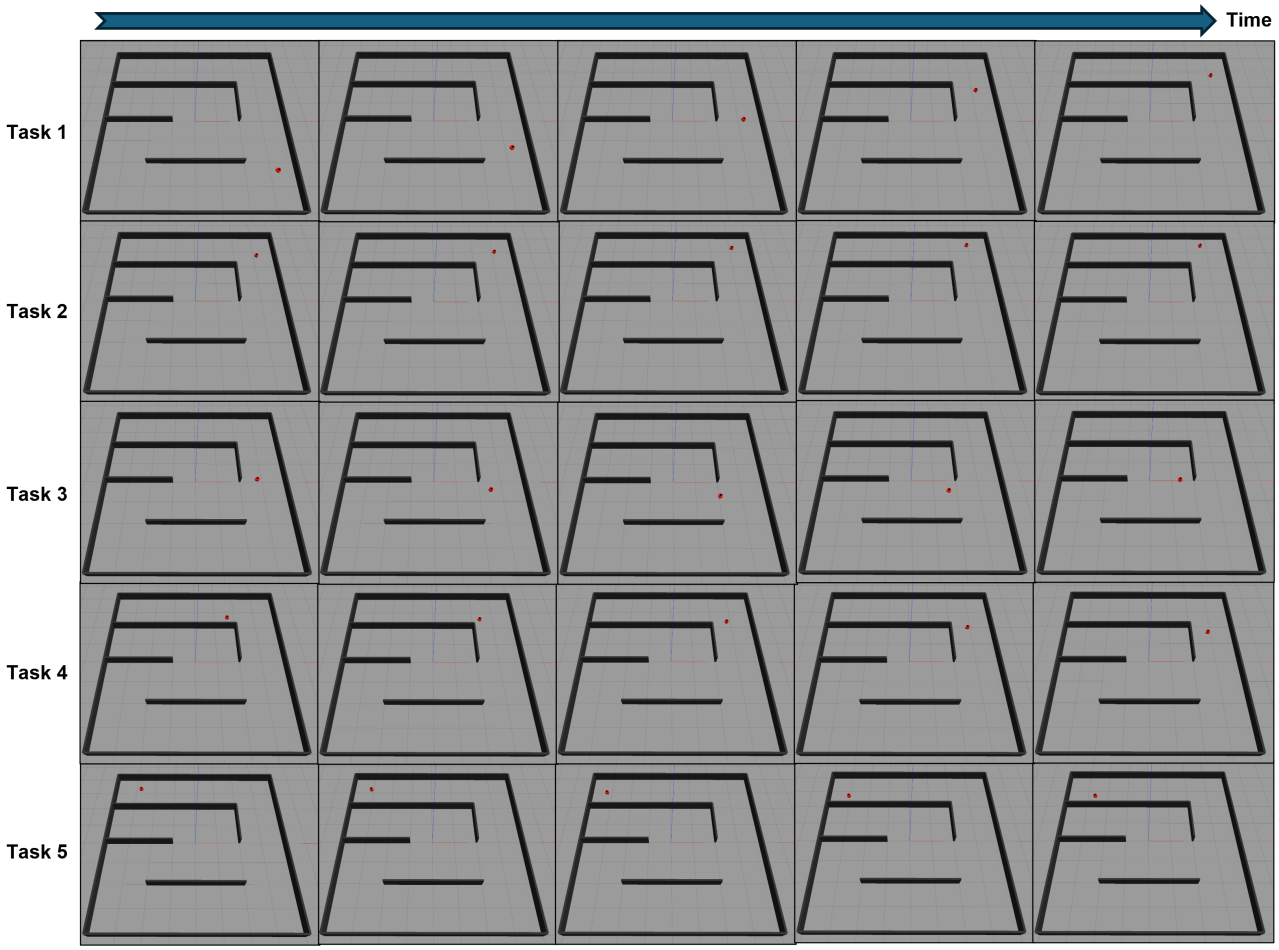


Fig. 7: Performance of the learned Q-table on various wall-following tasks.

Robot Trajectory During learning: Fig. 6 displays the robot's trajectory during learning, while the position with light color indicates high-frequency regions, and dark color represents low-frequency areas. The convergence of the robot's trajectory implies that the robot is capable of completing all tasks.

C. Performance Evaluation

Tasks Evaluation in Different Scenarios: Fig. 7 shows the robot's performance in negotiating various corner scenarios, including 90° and 180° turns. These results confirm that the learned policy generalizes well across different wall configurations, and the robot is able to achieve all tasks.

IV. Conclusion

This work demonstrates that Q-learning enables effective autonomous wall-following for mobile robots using only 2D LiDAR data. The learned policy allows the robot to maintain optimal wall distances and navigate diverse corner configurations, including 90° and 180° turns. Simulation results validate the approach generalizes well across different scenarios.