Reinforcement Learning for Wall Following Robotic Behavior

1st Ian Sinclair
Ritchie School of Engineering and Computer Science
University of Denver
ian.sinclair@du.edu

***Abstract*—Reinforcement learning algorithms are used to teach an autonomous mobile robot to follow walls and avoid obstacles in a fixed simulated environment. This involves learning an optimal mapping from *states* to control *actions* that allows the robot to maintain a fixed distance to walls without crashing. An optimal policy can be formally described by the statement, if at time t the robot is at a desired distance from a wall, d_w, then at time $t+1$, this distance is maintained while continuously moving forward and without crashing. Generally, a policy is preferred if it induces the robot to cover more distance along the wall in less time. Two strategies are employed, being off-policy temporal difference learning (Q-learning) and on-policy SARSA. The effectiveness of either strategy is determined by the learning convergence rate and accuracy (episodes vs. correct actions).**

***Index Terms*—Reinforcement Learning, Q-Learning, SARSA**

I. Introduction

A. Problem Formalism

Considered is a navigation problem that allows an autonomous agent to target its control strategy by its local surroundings. Wall following constitutes having the robot move along the circumference of its environment without crashing. In addition, a robot that covers greater portions of the wall faster is considered better. And correspondingly, there is a balance between moving quickly vs moving safely. If the robot is too close to the wall it will likely crash, vs too far away it will lose the topology of the wall and get lost. To that regard, take d_w to be some desired distance (*a priori* parameter) the robot should be from the wall that maintains its safety without getting lost. To formalize the problem, let $d(s_t)$ be the distance of the robot from a wall for state s at time t. Then an optimal policy should stabilize the displacement variable,

$$\Delta D_t + \delta(t) \to 0, \qquad \Delta D_t = |d(s_t) - d_w| \tag{1}$$

Where s_t is the state of the robot at time t, and $\delta(t)$ is a small perturbation function. Or rather the optimal policy should drive the robot to the desired distance from the wall and maintain that distance even if the robot is minorly thrown off course. This desired distance can be chosen somewhat loosely, relative to the system specifications. We chose,

$$d_w = 0.5m \tag{2}$$

with the expectation that a *'good'* distance is between $0.45m$ and $0.55m$ or $\pm 10\%$ of d_w. In Addition for the purpose of our experiment we consider only **right wall following** as a means of simplifying the state space and control behavior.

B. Environment

To simplify the learning problem, a single static environment is used during training. This is in contrast to a stochastic environment commonly used during generalizable reinforcement learning models. Furthermore, we are only interested in the robots ability to learn four behaviors well. This being, front wall following, $90°$ turns, I turns, and $180°$ U-turns.

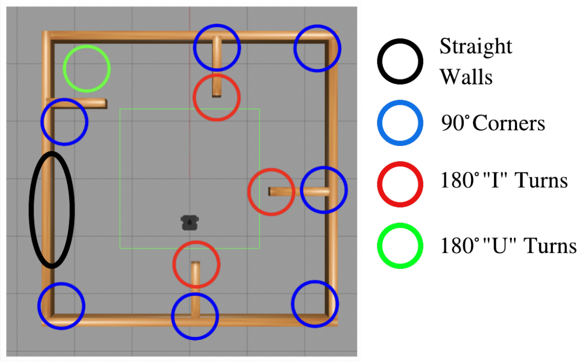

Fig. 1. Concept Art: Displaying the simulated environment and type of behaviors the robot should be able to preform. A grid tile is $1m^2$.

And so a successful policy should be able to navigate the environment while completing each of the four behaviors. In addition, by constructing a small time-invariant state space, the behaviors are also likely generalizable to other *similar* environments. However, testing this is out of the scope of the experiment.

C. Robot Specifications

The robot used during the simulation is a Turtlebot3 Waffle Pi. [Robotis()] Under the following specifications.

Maximum translational velocity	$0.26m/s$
Maximum rotational velocity	$1.82rad/s$
Size (L x W x H)	$281mm \times 306mm \times 141mm$
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02

Importantly, the robot is equipped with a $360°$ LiDAR scanner that reveals the distance from the robot to its immediate surroundings. This data is returned as a list of 360 distance measurements rotating around the robot counter clockwise starting at 0° being directly forward in the robots frame. The LiDAR scanners have an accurate operating range of approximately $1.5m$ to $3.5m$, and so all instrument readings are clipped to this range. LiDAR scans are emitted at $10Hz$. The robots state at any point in time is entirely informed by some encoding of the LiDAR scans, and so are discretized to steps of 0.1 seconds.

II. Design Parameters

A. State Space

The state space of the system is a description of the robots surroundings that can be used to inform behavioral decisions. Here, we assume all the information the robot needs to navigate can be encoded into a finite number states, where each state can be mapped to a certain type of action (controlled behavior). And so again, the learning problem is to uncover a mapping from the state space to a finite set of actions that enable to robot to solve each of the four behaviors. Here a state is defined by the tuple,

$$\begin{cases} \text{State} = & (\text{ right status, front status,} \\ & \text{ right diagonal status, left status}) \end{cases} \tag{3}$$

The status of each direction is informed by the minimum distance along the range of LiDAR scans. In ccw degrees, where 0 degrees is directly in front of the robot, the scan ranges are taken as follows

$$\begin{cases} \text{right} & \leftarrow [245^\circ, 335^\circ] \\ \text{front} & \leftarrow [-30^\circ, 30^\circ] \\ \text{right diagonal} & \leftarrow [-90^\circ, 0] \\ \text{left} & \leftarrow [45^\circ, 135^\circ] \end{cases} \tag{4}$$

The minimum distance of each is discretized to inform the directions status. Let $d_{direction}$ be the minimum distance provided by the laser scan over the corresponding directional ranges above. Then the discretization of each direction is found by the key,

$$\text{Right Status} \leftarrow \begin{cases} \text{close} & d_{\text{right}} \in [0, 0.35]m \\ \text{tilted close} & d_{\text{right}} \in (0.35, 0.45]m \\ \text{good} & d_{\text{right}} \in (0.45, 0.55]m \\ \text{tilted far} & d_{\text{right}} \in (0.55, 0.75]m, \\ \text{far} & d_{\text{right}} \in (0.75, \infty]m \end{cases} \tag{5}$$

$$\text{Front Status} \leftarrow \begin{cases} \text{close} & d_{\text{front}} \in [0, 0.5]m \\ \text{medium} & d_{\text{front}} \in (0.5, 0.75]m \\ \text{far} & d_{\text{front}} \in (0.75, \infty]m \end{cases} \tag{6}$$

$$\text{Right Diagonal Status} \leftarrow \begin{cases} \text{close} & d_{\text{right diagonal}} \in [0, 0.85]m \\ \text{far} & d_{\text{right diagonal}} \in (0.85, \infty]m \end{cases} \tag{7}$$

$$\text{Left Status} \leftarrow \begin{cases} \text{close} & d_{\text{front}} \in [0, 0.75]m \\ \text{far} & d_{\text{front}} \in (0.75, \infty]m \end{cases} \tag{8}$$

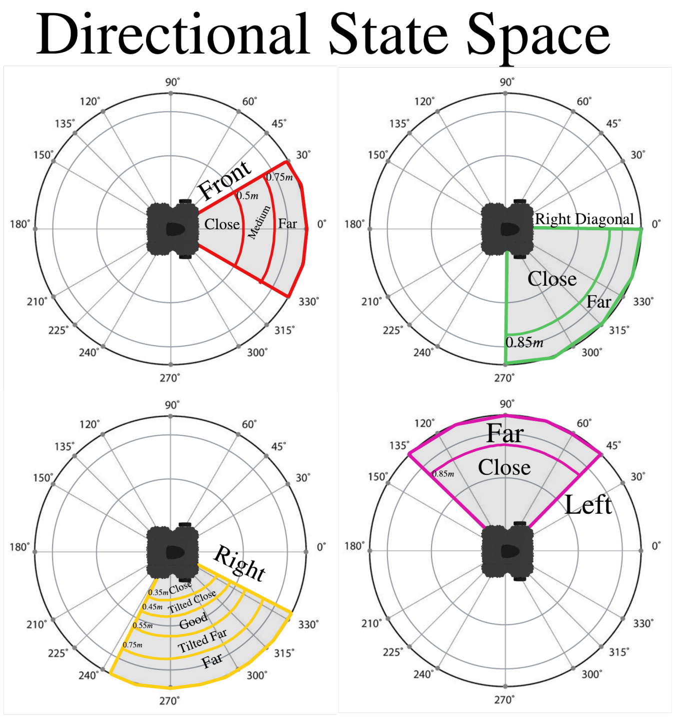

Fig. 2. Concept Art: Displaying the disretization of states in the state space.

The size of the state space can be determined by the Cartesian product over the codomain of each directional status. Resulting in,

$$|S| = 5 \times 3 \times 2 \times 2 = 60 \tag{9}$$

This discretization is chosen somewhat loosely, but with the goal of providing enough information so that robot positions/ surroundings that are resolved by different actions belong to different states.

B. Action Space

The action space $\mathcal{A}$ is a set of control inputs that affect the velocity of the robot. The RL model is tasked with selecting a single action from the action space for each state. And correspondingly, the controller can only adjust the velocity of the robot if the state changes. The action space is designed to give the learning algorithm enough angular control to adjust around turns without hitting the wall, while not overshooting from the 'good' distance from the wall. The linear velocity for all actions is fixed to,

$$\text{Linear Velocity} = 0.2m/s \tag{10}$$

This is to prevent the learning algorithm from reaching a good wall distance and then stopping. Practically, it would be useful to add stopping and/or reversing linear velocity actions as part of a different 'un-stucking' subsystem. Unique actions in the action space can be defined by their angular velocity,

$$\mathcal{A} = \{-\pi/2, -\pi/4, 0, \pi/4, \pi/2\} rad/s \tag{11}$$

Which correspond to the literals,

$$\begin{cases} -\pi/2 & \text{Hard Left Turn} \\ -\pi/4 & \text{Slight Left Turn} \\ 0 & \text{Straight} \\ \pi/4 & \text{Slight Right Turn} \\ \pi/2 & \text{Hard Right Turn} \end{cases} \tag{12}$$

Note, $\pm\pi/2rad/s$ approaches the angular velocity limit of the robot, and so preforming rotation at this speed or oscillating between $\pm\pi/2rad/s$ rotations quickly may cause challenges in lower level control systems. (Make it difficult to accurately control the robot as a result of for example wheel slippage.) This is taken into account in the reward function for the learning algorithms, described later.

III. Learning Algorithms

A. Algorithm

The algorithm can be highlighted by,

Algorithm 1 Reinforcement Learning Algorithm

```
Initialize $Q$-table with all 0 q values.
repeat
    Initialize Environment
    repeat
        Pick $a$ for $s$ from $\epsilon-$greedy strategy
        Take action $a$ observe new state $s'$ and reward $r$
        Update $Q-$table with learning strategy
        $s \leftarrow s'$
    until Terminating condition
until (For all episodes)
```

The remaining sections explain each component in more detail.

B. Reinforcement Learning

Two Reinforcement Learning algorithms are used, either off-policy temporal difference learning or on-policy SARSA(λ). Both are online learning strategies that through many successive trial and error attempts find a mapping between the state and action spaces. This is done by iteratively constructing a table ($Q-$table) describing the expected *quality* of every action for each state. And so for action $a \in \mathcal{A}$ and state $s \in S$, the q value, $Q(s,a)$ gives the expected utility or quality of preforming action a.

C. Policy Generation

Both learning algorithms find an optimal policy $\pi^* : S \rightarrow \mathcal{A}$, such that,

$$a^* = \pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

Q values for a particular state $s \in S$, are propagated by the updating strategy related to either Temporal Difference or SARSA.

D. Off-Policy Temporal Difference, Q-learning

Q-learning assumes the robot will act in an ideal sense after reaching a new state. And so after taking action a in state s and observing a new state s' with a corresponding reward r, the Q table is updated by,

$$\begin{cases} sample & \leftarrow R(s,a,s') + \gamma \max_{a'} Q(s',a') \\ Q(s,a) & \leftarrow (1-\alpha)Q(s,a) + \alpha[sample] \end{cases} \tag{13}$$

For learning rate α, and update rate γ. For both Q learning and SARSA,

$$\alpha = 0.1, \qquad \gamma = 0.8 \tag{14}$$

Because Q-learning is off-policy and assumes the agent will act optimally at the new state s' it can be more *risk prone* or rather if a new state has a high punishment adjacent to a high reward, only the high reward is observed.

E. On-Policy SARSA

SARSA on the other hand is on-policy meaning it assumes the robot will act in accordance to what its has learned so far. As a result, from the previous example, it is likely the high punishment will factor into the decision to move to s'. This learning strategy also employs the new action, a' leaving s'.

$$\begin{cases} sample & \leftarrow R(s,a,s') + \gamma Q(s',a') \\ Q(s,a) & \leftarrow (1-\alpha)Q(s,a) + \alpha[sample] \end{cases} \tag{15}$$

The difference is subtle, however, here the next action a' directly informs the q value from the previous state. This can often result in a more *risk-adverse* behavior.

F. Reward Function

The reward function is used to inform the quality of preforming action a and state s. Here, both an intrinsic and extrinsic reward is used to reward the robot for maintaining a distance d_w on its right from the wall, and punish it for preforming actions that are risky or may make the robot difficult to control in a real world setting. The extrinsic reward uses the literal distance, d, from the robots right side to the wall as input to the graduated reward function,

$$r_{ext} = \begin{cases} 5 & |d - d_w| < 0.1 \\ -5\sin(\frac{\pi(x-0.5)}{6}) + d_{front} & |d - d_w| > 0.1 \end{cases} \tag{16}$$

And the intrinsic reward, slightly punishes the use of risky actions.

$$r_{int} = \begin{cases} -\frac{1}{2}|a| & |a| = \pi/2 \\ 0 & \text{Otherwise} \end{cases} \tag{17}$$

Then the final reward function is the sum of the two,

$$r(a,s') = r_{ext}(s') + r_{int}(a) \tag{18}$$

This is designed to promote the use of manageable turning speeds in cases where they are acceptable, while still allowing the algorithm to pick greater angular speeds if necessary to prevent crashing.

G. Learning Algorithm

Aside from the Q table update strategy, Temporal Difference learning and SARSA use the same overarching algorithm that is run over a series of episodes.

H. Episodes

An episode is a single learning session where the robot is allowed to try to freely explore its environment and learn as much as possible before reaching a terminating condition, in which case the episode ends, the world gets reset, and the next episode starts. Here, three terminating conditions are employed, first, the robot collides with a wall. Second, the state of the robot does not change for a significant amount of time (It is likely the robot is spinning or otherwise not exploring). Third, to much time has passed without reaching another terminating condition, this is either because the robot is lost in empty space, or has mastered not hitting walls.

I. Steps

Within each episode are steps, where the robot is given the opportunity to select a new action to test for which ever state its in. Because the LiDAR scans are taken at $10Hz$, a new step is decided to be taken every 0.1 seconds.

$$\text{Step Size} \longrightarrow 0.1 \text{ Seconds} \tag{19}$$

Note, that it is not guaranteed that the robot will have entered a new state on each step, and so it is likely that the Q value for each state action pair will be updated multiple times before transitioning to a new state. This allows the algorithm to try out multiple actions for every state in each episode, and may quicken learning speed. In general, at every step, the algorithm evaluates its state and then selects a new action to try, however, some choices are still better than others and so we use an epsilon-greedy strategy to select actions based on how far into the learning process the episode is.

J. Epsilon Greedy Strategy

Early in the training process it may be preferred for the robot to move randomly so to explore its environment more and look for better styles of behavior. However, as the learning continues and the number of completed episodes grows it may be better for the robot to start exploiting some of its accumulated knowledge to progress further into its environment. To accommodate this, an epsilon greedy strategy is used where at every step, the algorithm randomly decides to either take a random action or exploit its $Q-$table. And so take r to be a random number; then, for some parameter $\epsilon \in [0.1, 0.9]$ if $r < \epsilon$ the robot moves randomly, and otherwise exploits its Q table. And so at state $s \in S$ the associated action is,

$$a = \begin{cases} \text{random} & r < \epsilon \\ \arg\max_{a\in\mathcal{A}} Q(s,a) & \text{otherwise} \end{cases} \tag{20}$$

The parameter ϵ decays linearly as the number of episodes increases. Starting at $\epsilon = 0.9$ at episode 0, and decaying to $\epsilon = 0.1$ on the final episode. Next, consistent with [Moreno et al.(2004)Moreno, Regueiro, Iglesias, and Barro] after deciding to move randomly, the random action is selected using softmax. Where the probability of taking action a_i in state s is given by,

$$\mathbf{Pr}(s, a_i) = \frac{e^{Q(s,a_i)/T}}{\sum_{j=1}^{n} e^{Q(s,a_j)/T}}, \qquad i = 1, 2, \cdots, n \tag{21}$$

Where $T = 10$ is a temperature parameter to prevent that denominator from becoming intractable.

IV. Evaluation of Success

Success is evaluated by two metrics, being the learning convergence rate and accuracy. The convergence rate and accuracy are informed by selecting a few states with known 'best' actions and comparing the ratio of steps where the algorithm encounters a known state and also greedily selects the correct action against the number of epocs. Faster convergence speed indicates that the learning strategy is capable of learning an 'acceptable' behavior with fewer training cycles. While, a greater accuracy informs the quality of the policy each learning strategy is capable of finding.

V. Results

To evaluate the success of the algorithm during the learning process three states for each action are selected and watched during each epoc. These states have known correct actions, and so if the robot encounters one of the watched states and greedily preforms the correct action then it is a sign that it is learning the correct policy. Correspondingly, we plot the ratio of correctly selected actions over the total number of occurrences for all known states against the total number of episodes.

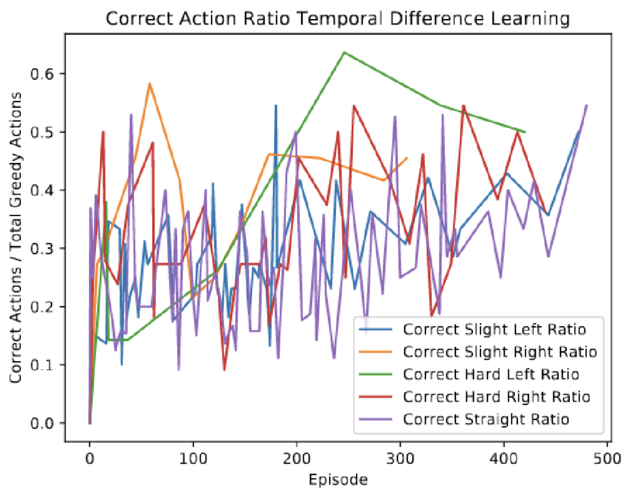

Fig. 3. Chart displaying the ratio of correct actions over total number of observed known states against the total number of episodes for Temporal Difference learning.

Notice above, there is a slight upward trend on the ratio of correct actions as the number of episodes increases. However, none of the action ratio's converges to a fixed number. This could be caused by poor learning performance of the algorithm, or poor states were selected as known state candidates. Note, that $|\mathcal{A}| = 5$ and so random selection should result in a correct action ratio of approximately 0.2. Most of the action lines on the Temporal Difference plot oscillates around 0.35 which slightly improves on random choice.

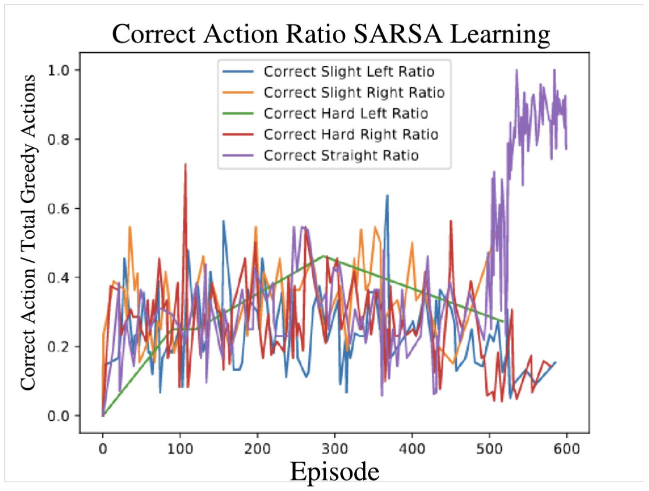

Fig. 4. Chart displaying the ratio of correct actions over the total number of observed known states against the total number of episodes for SARSA learning.

Above is the correct action ratio of SARSA learning strategy. Notice, there is a consistent pattern with Temporal Difference for 4 out of 5 actions. (Staying unconverged around 0.35). However, the 'go straight' action converged to around 0.9 at around episode 500. This is a fairly strong result for that direction and is a good indicator that the algorithm is learning. Additionally, actions 'Hard Right' and 'Slight Left' converge below to around 0.1 which is less than random chance, (0.2). Which indicates that the algorithm learned a policy around these states, however, learned a action than what is expected. Overall the algorithm may not have learned the correct policy for every direction, however, it improved somewhat as the number of episodes increases.

VI. Conclusion

Two learning algorithms, Temporal Difference and SARSA, are used to teach a modile robot the wall following behavior. Both algorithms used a type of online reinforcement learning based on constructing a table that serves as a map between state and action spaces by ranking the utility of all action at every state. An epsilon greedy strategy using softmax is employed to allow the robot to balance exploring its environment vs exploiting the knowledge its accumulated. The learning strategy have decent results, preforming slightly better than random chance, and are able to complete three out of four of the desired behaviors. Excluding the 180° U-turn.

References

[Moreno et al.(2004)Moreno, Regueiro, Iglesias, and Barro] D. L. Moreno, C. V. Regueiro, R. Iglesias, and S. Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*, page 33, 2004.

[Robotis()] Robotis. Robotis e-manual turtlebot3 waffle pi specifications. URL `https://emanual.robotis.com/docs/en/platform/turtlebot3/feat`