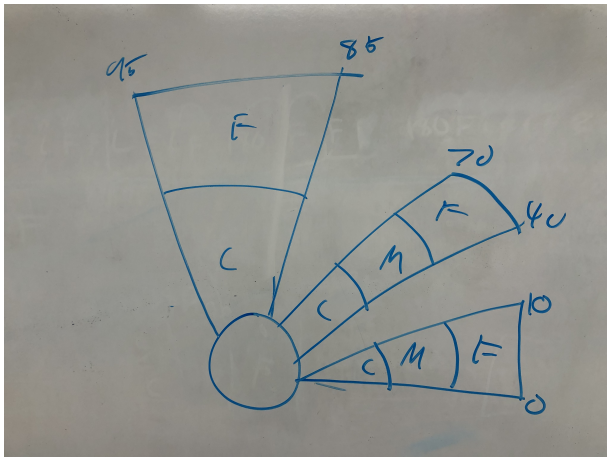
Project 2 part 2

Gordon Dina

I. Introduction

For this part, we take what was learned from part 2 and expand on it through deep learning. Like before the bot will take in an input state based on the Lidar Data collected and use a Q-table to determine its action for following the wall. The Q-table that is used will be obtained through training the bot by deep Q-learning.

II. States and Actions



States: Close, Far, and Medium. I had 3 visions for my bot: one in front of the bot (85-95 degrees, close and far), one at a left angle (40-70 degrees, close, medium, and far), and one to the right of the bot (0-10 degrees, close, medium, and far). The bot had different degrees of vision, the new degrees improve data collection. Vision 85-95 was only given 2 states as that vision was mainly used for checking for corners. While the other 2 visions were given 3 states as they were the primary source for checking if the bot was following the wall.

Actions: Forward, Right Forward, Left Forward. Forward is simple if not close to the wall go to the wall, Right and Left forward to similar things they both go forward in the y direction but the difference between them is the turn of the bot, Left rotates the bot counterclockwise while Right rotates clockwise, they help keep close contact with the wall allowing the bot to follow the wall.

III. Rewards

I focused more on punishment than rewarding. If the bot got too close to a wall it got punished because I didn't want it to hug the wall, If it was too far away from a wall I would punish it a little bit due to not want to be too far from the wall but not too far that nothing happens. It got rewarded for getting in that sweet spot of hugging the wall, not to close but not too far. I also rewarded it for keeping its 85-95 vision close to the wall for more clarity.

IV. Chosen Actions

Actions were chosen through an Epsilon Greedy Algorithm. Where we have an Epsilon variable that is between 0 and 1 (for this it was 0.9) as well as generate a random number between 0 and 1, next would be to see if the random number is greater than Epsilon: if so then it will choose a greedy action from the Q-table, but if not it will choose a random action. After the action has been chosen and run it will then undergo a decaying process, decreasing in size as the bot goes through more episodes focusing more on exploitation and less on exploring.

V. Update Q-table

Updating the Q-table was broken up into two parts: adding new states and modifying old states. The algorithm used was a double 'for' loop, one to go through 200 episodes, and the inner loop iterates through 1000 times. The Q-table is set up where the dictionary is initially empty, and as the bot discovers new states it will add that state to the Q-table with empty actions. The bot will first choose an action based on the Greedy algorithm that was described before, execute that auction and observe the new state through the Lidar, calculate its reward at its state, then it will update the Q-table through the formula shown in the image below which is the formula for Q-Learning, then it will decay the epsilon for the next episode, and then will check for certain conditions like if it gets stuck or starts flying and will terminate that episode and move onto the next one.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

VI. Performance

The bot performed better than last time due to a change in its Lidar vision.

References

[1] Project2 D3 slides
[2] Reinforcement learning activity slides
[3] https://answers.ros.org/question/9783/programmatically-get-modelstate-from-gazebo/ (Set state)