Reinforcement Learning for Robot Obstacle Avoidance and Wall Following

Chi Zhang
Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, TN 37996, USA
Email: czhang24@utk.edu

Abstract—**Reinforcement learning approach is learning how to map environment situations to actions, to maximize a reward signal. In this project, the SARSA(λ) algorithm is applied to teach a robot to follow a wall and avoid running into obstacles. The robot is equipped with a laser scanner and odometry to perform the learning. The robot senses environment situation from the laser readings and decides which action to take by the learned policy. Then a reward is given to the robot according to the new state, and the policy is updated.**

In this project, the SARSA(λ) reinforcement learning method successfully taught the robot to learn the task after around 10 minutes training. Firstly, the resolution of states effected learning speed. If there are too many states, the robot would take a long learning time. Actions effected the quality of learning due to that more choice of actions resulted in more degrees of freedom for the robot to move sophisticatedly. Secondly, the reward function impacted the Q table in a complicated way. If the reward definition was too complex, there might be some conflicting gains and it would be difficult to get a stable Q table. A concise reward function benefited to shorter learning time. Thirdly, exploration and exploitation was investigated by different greedy thresholds. Decaying the greedy value was necessary for the robot to explore at early stage and drive on correct path later. To verify the effectiveness of my design, I tested the robot in another map, and found that the robot took longer time to learn the more complex map, but can still successfully learn the skills. I also explore the impact of level of momentum, which is different level of Markov Decision Process, by tuning the value of λ.

I. Introduction

Reinforcement learning is a computational approach to learn form interaction, which is a foundational idea in almost all the learning methods. Different from supervised learning, the reinforcement learning actually learns from the interaction with environment. In the real world, a knowledgable external supervisor might not be alway available. So it is often impossible to obtain examples of perfectly correct guide for all the situations. In this case, the agent must learn from its own experience.

One of the challenges in reinforcement learning is the trade-off between exploration and exploitation. On one hand, the agent must try to take actions with better rewards to achieve the the best accumulative rewards. But on the other hand, to find such good action, it need to try actions that it has not selected before. The underlying idea is that the agents has no idea what the reward would be unless it takes the action. Consequently, the agent has to exploit the already known best action to obtain rewards, while also explore the untaken actions to make better action selections in the future. Another key difference separate reinforcement learning from other learning methods is that reinforcement learning considers the whole problem during in the learning process while many supervised learning approaches only consider subproblems and do not explicitly specify how such learning ability come from and how it would be useful. Taking genetic programing in the last project for example, the fitting equation only solves the problem in a given range, but would not be applied to points outside of the range[1].

To formulate the reinforcement learning problem, several main objects are defined: state, reward and action. The agent sense its state of the environment and decides which action to take in the current state, then a reward is given to the agent by it's next state. The Q table, which is also known as the policy, is updated according to the rewards. Figure ?shows the structure of such a system.

II. Problem Formulation

A. Reinforcement learning approach design

1) State definition: The robot senses the environment by its laser sensors, which covers a rang of the semicircle in front of the robot. States of the robot are defined according to these 361 readings. In this project, a good state definition [2].was found to be consisted by four sectors and one slope variable, as Figure 1 shows.

The semicircle sensing area was equally divided to 3 sectors, each of which was 60 degree. And a left front sector was defined as the upper left with 30 degree sector inside of the left sector. So that a state contains 4 sector variables. To indicate the angle between the robot and the wall, a slope was calculated by points in the left front sector and left sector, using linear regression. In the training process, the robot was learning to follow by its left side, so that we define a left-front sector to monitor the following actions. In the front sector, four segments was defined by distance: too close, close, medium and far. In the right sector, because the robot don't care about if it is following the wall by right, the sector was divided into two segments: far and close. The left sector played an important role in following the wall, so that it was divided with a fine grain: too close, close, medium, far and too far. In the left-front sector, the close segment is larger than the "close area" defined in the left sector because generally it should has a longer distance from the wall than the left sector when

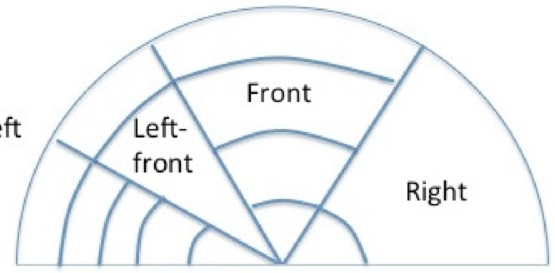

Figure 1. Definition of state

following the wall with a good gesture. If the left-front is the too close to the wall, the robot was at high risk of running into obstacle.

During the experiments, I found that the four sectors cannot fully represented the state of robot, and the direction of robot was very important for the following and avoiding behaviors. Therefore, the slope state was roughly defined by robot's directions: approaching, moving away, parallel, and nothing. For the approaching state, the robot moves towards the wall. In the moving away state, the robot moves away from the wall. In the parallel state, the robot moves parallel with the wall. If the wall is too far or too close to the robot, the slope state is nothing. Note that the slope was defined as the angle between the robot moving direction and one wall, assuming that the robot only follow one wall at a time. If the readings is too large, it indicates that the laser scans a large area, so that the sensor data might come from several different walls, in this case the slope value might not be correct. In the too close case, the accuracy of slope result cannot be guaranteed so it is also set to be zero.

According to the definition, the total number of states was 320, but in the implementation, each sector was assigned values ranging from 0 to 4, to simplify encoding the states. Note that only 320 states had meanings in the Q table. Further more, not all of the 320 states would be used due to the characteristic of environment. Therefore, the actual states was controlled in a small number to constrain the search space so that improve the learning efficiency.

2) Action definition: Because each Q values in the Q table was defined by a state-action pair, more actions would increase the Q table exponentially, but also leads to more degrees of freedom of the motion capability. In this project, since the number of states was quite small, the number of actions was designed flexibly.

To simplify the definition, the speed of each action was fixed to be 0.3 meter per second, while the turn rate was from -180, -90, -45, 0, 45, 90, 180. The backward action was defined as 0 turn rate with -0.3 speed.

According the state and action definition, the Q table has 320 rows and each row has 8 columns, so that the total size is $320 * 8$.

3) Reward function: Design of the reward function is critical in the reinforcement learning algorithm. In the experiments, I tried lots of different reward functions, and some of them were quite complicated. I found that a complicated reward function was not necessary to accomplish quick and good learning. Conversely, if the rewards were over-defined, it would take more time for the Q table to learn a stable policy. The possible reason might be contradict contradict reward gained in different state-action pairs.

4) λ selection : The most important feature of SARSA(λ) is the eligibility trace used in learning. The λ controls the influence coming from the former states when the robot is thinking which action to take under the current state. If the λ is small, e.g., 0, the SARSA(λ) becomes dynamic programming, which only takes one step ahead into account. If the λ is large, the SARSA(λ) algorithm holds more characteristic of Monte Carlo, which considers all the steps from now to the terminal step. In this project, the best λ value was found to be 0.8.

5) Termination of an episode: In this project, a learning episode contained infinite steps until the robot hit the wall or was trapped in a small area. It was implemented by checking the location of robot for a certain period of time. Then the program would reset the robot to the start point and began a new episode. When the robot moved for certain number of steps without failure: running into obstacle, trapped in a small area, or lose the wall, the learning was considered to get converged. The number of successful steps was decided by different maps. For the given map, since the robot was trained in a small room, 250 steps were sufficient to cover every wall and corner in the room. However, there were still lots of different situations in the given map, so that to train the robot move correctly for the whole map, longer training steps were required. Experiments with 1000 successfully learned steps showed that the robot could follow most part of the map except the complex doorway area in the upper left corner of map.

B. SARSA(λ) algorithm

After outlining the structure of the SARSA(λ), the algorithm [3] was displayed as follows.

Algorithm 1 Pseudo Code of SARSA(λ) algorithm

```
Initialize $Q(s,a)$ arbitrarily and $e(s,a)=0$, for all $s$, $a$
Repeat (for each episode)
    Initialize $s$, $a$
    Repeat (for each step of episode)
        Take action $a'$, observe $r, s'$
      Choose $a'$ from $s'$using policy derived from $Q$ (ε greedy)
        $\delta = r + \gamma Q(s',a') - Q(s,a)$
        $e(s,a) = e(s,a) + 1$
        For all $s$, $a$:
            $Q(s,a) = Q(s,a) + \alpha\delta e(s,a)$
            $e(s,a) = \gamma\lambda e(s,a)$
        $s = s'$
        $a = a'$
    Until $s$ is terminal
```

III. Experiments

A. Learning from the interaction with environment

Before finding a proper learning policy, I did an experiment with a fairly long training time to have a general sense about how the robot learns from interaction with the environment. Experiments showed that different start point may had different learning time. If the robot started from a wide open area, it had to explore randomly to find a wall that was close enough to follow. Considering most rooms in the given map had the same shape: straight walls and 90 degree corners, I trained the robot in a small room to make it learn more quickly, and then tested it with other rooms as well as the wide open space.

Figure 2 was the snapshots showing that the robot was learning as it had more interactions with the environment. It performed better and better and finally can follow every wall and handle every corner in the room. At the very beginning, e.g., the first snapshot of the 36th second, the robot moved randomly and the footprint was disorder. The second snapshot taken at the 57th second indicated that the robot was trying to approach the wall for a short period of time but then hit the wall and was reset to the start point again. An interesting phenomenon happened during the time from 1 minute 35 seconds to 4 minutes 16 seconds (snapshot 3, 4, 5). During this period of time, the robot was partially following the wall but also lost the wall easily, so that the following footprints were inconsistent. After learning for 5 minutes, the robot finally learned how to follow the straight wall, but hit the corner when the wall ended. This was reasonable because the robot never saw a corner situation before so that it didn't know how to avoid running into it. After trying for several times, the robot successfully handled the corner following. At last, both the wall following and obstacle avoidance were learnt by the robot after learning for around 7 minutes in the room.

The greedy parameter ε was decaying as the robot was learning better and better. It was designed as a piecewise function of steps, so that when the robot has learned the skill very well, the best action would always be selected.

Even though the environment in the selected room had most features of the given map, one kind of convex corner was not included. So I extended the training from the room to the open space with convex corners. Figure 3 shows that the robot learned quickly how to follow such convex corner without collision and followed most part of the given map. Note that after training for 30 minutes, the robot ran pretty good except for the complex area in the upper left. However, after training for 1 hour, the robot can successfully handle the complex area, as Figure 4 shows. Please note that for learning the whole map, I slightly modified the reward function: if any of the sector was too close, the reward was -1; if the left sector was medium, which is the perfect situation, the reward was 1; if the left sector was close or far, the reward was 0.5; in any other case, giving a zero reward.

B. Rate of convergence of learning

Figure 5 shows the rate of convergence of learning. The total number of episode was set to be 20000, which is large enough

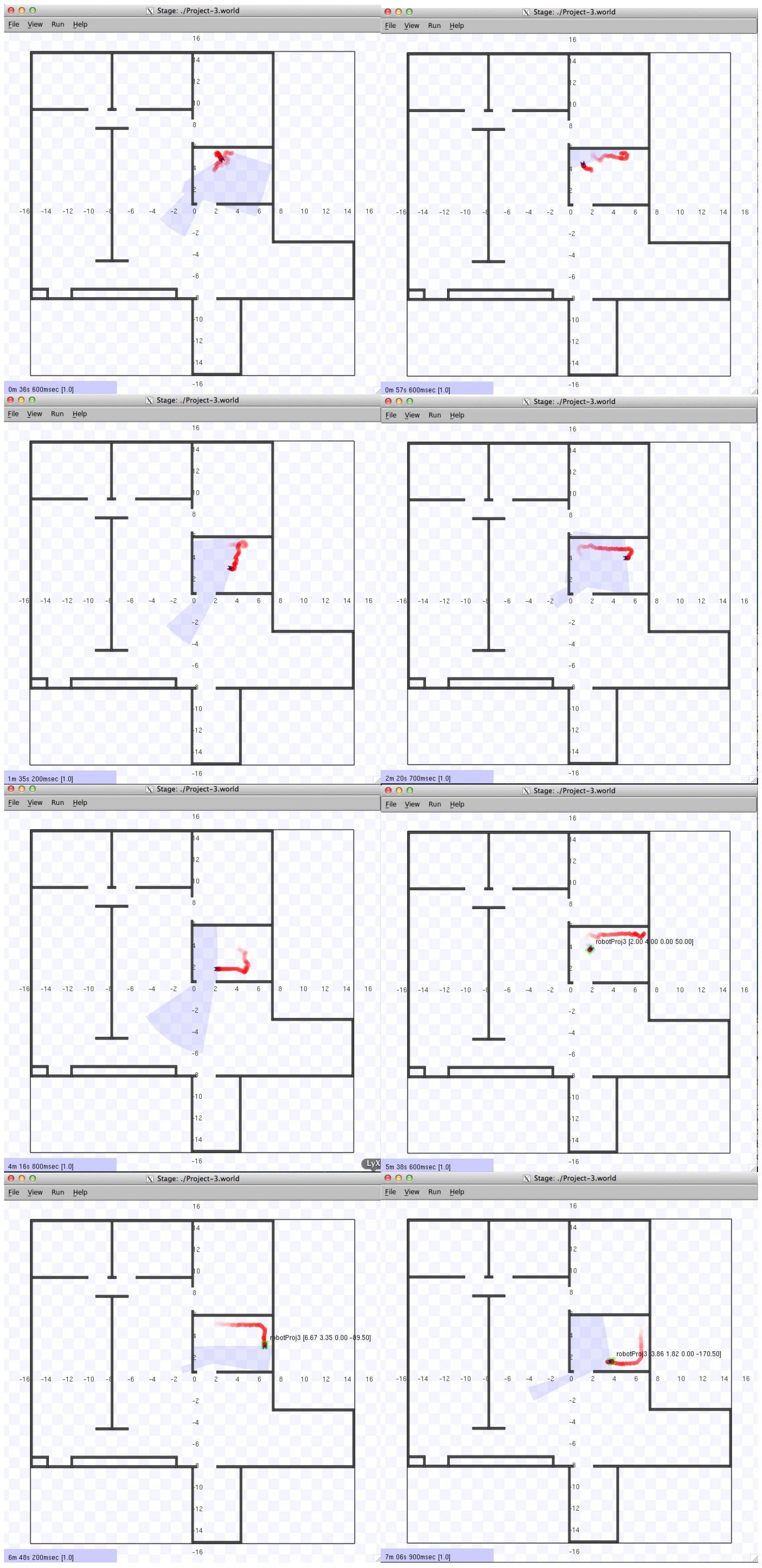

Figure 2. Robot behavior in different learning episodes

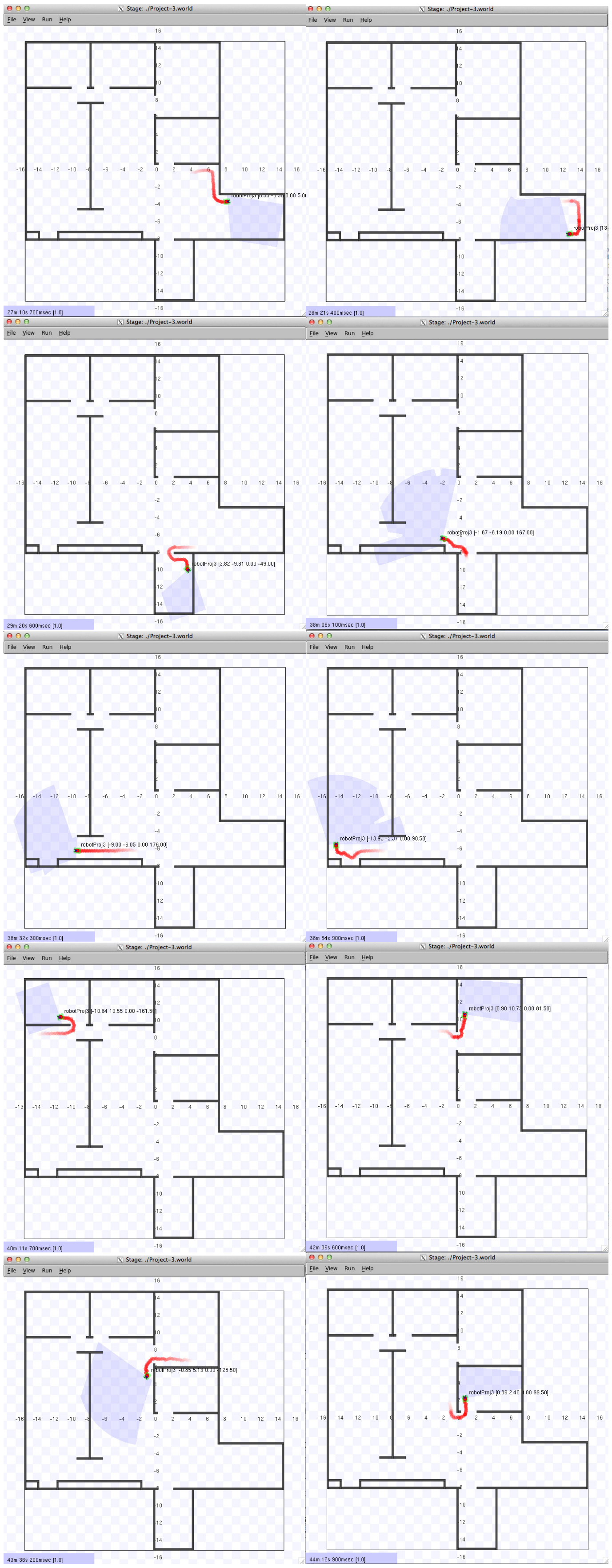

Figure 3. Exploring the other part of the map (after learning for 25 minutes)

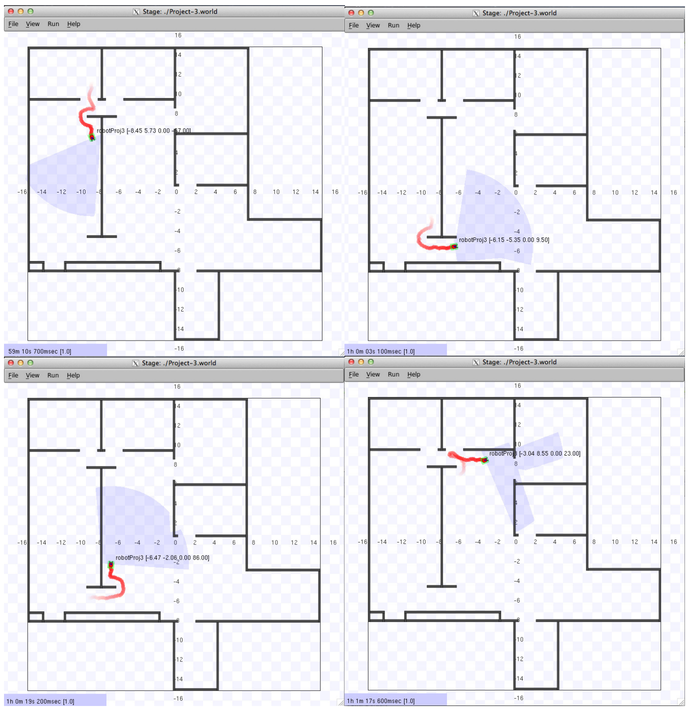

Figure 4. Handle the complex upper left area(after learning about 1 hour)

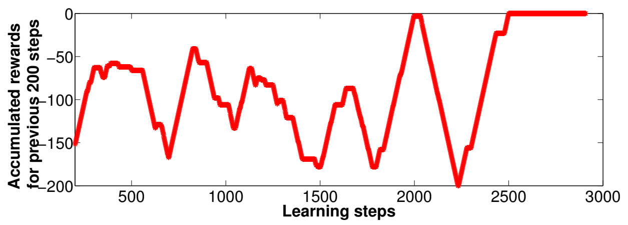

Figure 5. Rate of convergence of learning

to meet the learning requirement. The termination condition was described in Section 2.1.5. The rewards were increasing as the robot took more learning steps. For the small room case, the robot learned how to follow and avoid after about 2600 learning steps.

IV. Discussion

A. *Effect of different learning layouts*

To test if the effectiveness of the learned skills, I did experiments with a different layout cave.png. The cave map was more complicated than the autolab map, given that it had more complex corners which were not 90 degree and the surface of wall was not smooth. Therefore, the robot had to handle more difficult environment. The robot learned the longest continues wall in the right side of the map, because this wall contained the most different surface than the autolab walls. And the robot was set to start from location [14, 4] to be closer to the desired learning area, other than randomly exploring in the wide open space. Figure 6 shows that the robot successfully followed walls and handled concave corners and convex corners in the cave map.

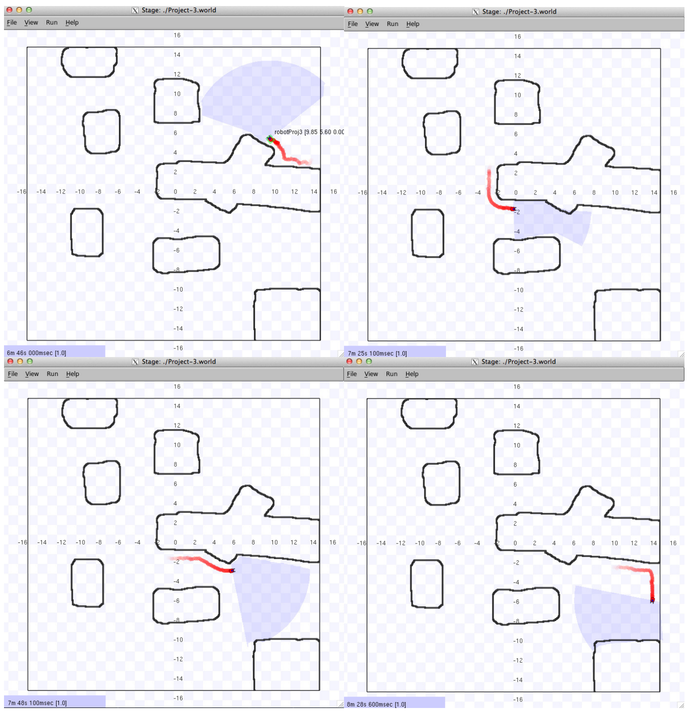
Figure 6. Learning the cave map

Layout	auto_lab	cave
Quality	good	good
Learning Time (s)	7"06	9"13

Table 1
COMPARE AUTOLAB MAP WITH CAVE MAP

It was easy to see from table 1 that the robot took longer time to learn the cave map than the autolab map because it had to experience more states to learn. The learning quality was as good as in the autolab map.

B. *Effect of different state and action resolutions*

During this project, I tried lots of different resolutions of states and actions, and found that the number of states effected the learning quality significantly. If the state space was too large, the robot took a long time to learn well, but the learning quality were generally the same. Table 2 shows the learning time for two different state spaces which were both success in the experiments. The 4096 state space divided the semicircle readings into six sectors, and each sector has four possible situations: too close, good distance, far and too far. The criteria for finish learning is that when the robot follows the wall without running into obstacle for 250 steps, which was approximate 20 meters.

When change the action space to 6, the learning speed was slower than the default 8 actions. In the new action space, turn rate of +180 and -180 degrees were removed. Figure 7 shows that the new action space can handle doorways easily, but it took more time to learn turning in corners, as Table 3 shows. The default 8 action space took 2616 steps to converge, while

State Space	320 (default)	4096
Quality	good	good
Learning Time (s)	7"06	6"30

Table 2
EFFECT OF DIFFERENT STATE SPACE

Action Space	8	6
Quality	better at corner	better at doorway
Learning Time (s)	7"6	14"10

Table 3
EFFECT OF DIFFERENT ACTION SPACE

the 6 action space took 5674 steps to achieve learning success condition.

C. *Effect of different reward functions*

Reward function played an important role in the learning task. At the early stage of doing this project, I defined a complicated reward functions which was a linear function of distance from wall / obstacles, but it took a long time to get converged, and sometimes even failed in learning. To quantitatively investigate the effect of reward functions, I added a positive 1 reward to the default reward function, when the left sector of robot is in the medium distance from the wall. It was interesting to find that it took triple time for the learning algorithm with the complex reward function to converge. To take more advantage of the distance, the third reward function defined the reward as: if the left sector was medium, a positive 1 reward would be given; if the left sector was either close or far, a 0.5 reward would be assigned; if any sector detected too close, the reward would be -1; in any other case, giving 0 reward.

The underlying reason might be that a complicated reward might introduce contradict behavior gain at the early learning. For example, image a robot moving in the good distance from the wall, despite of the moving direction, the robot will

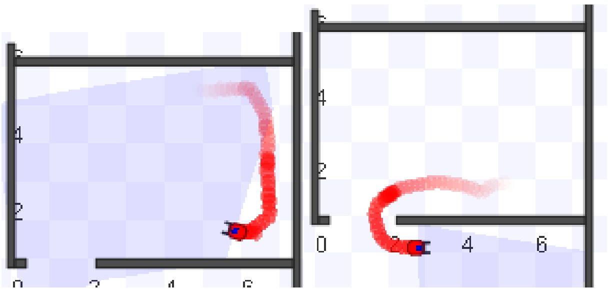
Figure 7. Robot behavior at corner and doorway with 6 actions

Reward function	{-1, 0} (default)	{-1, 0, +1}	{-1, 0, 0.5, 1}
Quality	good	good	best, can handle complex area
Learning Time (s)	7"06	23'03	9"13

Table 4
EFFECT OF DIFFERENT REWARD FUNCTIONS

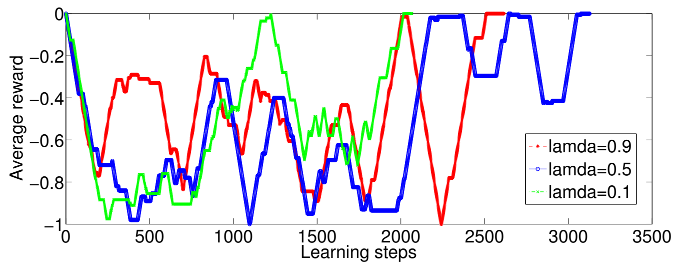

Figure 8. Effect of λ on the average reward

always get a positive reward +1 in the second reward function. However, these kinds of states will lead the robot get out of the best distance area. The third reward function overcame the contradiction of rewarding by considering more cases. It was proved to be the best to handle some complex area, like doorway.

D. *Effect of learning parameters* λ

The λ value was used to shift from one-step TD methods to Monte Carlo methods. It characterizes how far into the future the λ-return algorithm looks in determine its backup. In this project I turned the λ from 0.1 to 0.9, and observed the influence in the average reward. Because the accumulated reward was computed from the previous 200 learning steps, all the accumulated rewards were normalized to 200 so that all the values located between -1 and 0. As Figure 8 shows, when λ is 0.5, the learning speed the slowest and when λ was 0.1, the robot learned using shortest time. Consequently, the value of λ need to be tuned properly in this problem.

V. Conclusion

In this project, a SARSA(λ) reinforcement learning algorithm was applied to solve the problem of learning wall following and obstacle avoidance for a robot. Several observations were made through the experiments. Firstly, the definition of state effected the search space significantly. Too large state space resulted in a very slow learning due to the fact that the robot was learning in real time. Secondly, the action space effected quality of learning in different environments. Large range of action selection benefited flexibility when the robot was adjusting the turn rate in following corners, but introduced more uncertainty in doorway situations. The reward function need to be carefully designed. A proper reward function can handle more complicated environment quickly.

During the experiments, I found that a decaying greedy was necessary. It would make the robot learn faster at the early learning period, and also let the robot focus on the best action as it learned more. The design was proved to be effective with other map like cave.png.

References

[1] Tom Mitchell, McGraw Hill, *Machine Learning*. Mcgraw-Hill International Edit, 1997

[2] David L. Moreno, Carlos V. Regueiro†, Roberto Iglesias and Sen´en Barro, *Using prior knowledge to improve reinforcement learning in mobile robotics*, Proc. Towards Autonomous Robotics Systems

[3] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html