CS5478 LAB 1

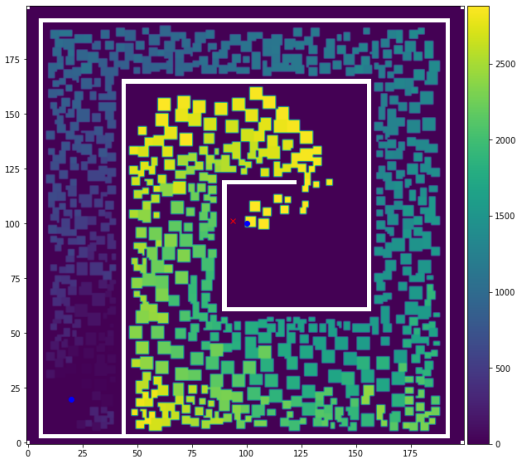
Prepared by: Chong Yihui (A0119331N)

1. DSDA

For the DSDA model, I have used the A* Search algorithm. After discretization of the state space, the configuration spaces are not large and can be easily handled using DP or Forward Search. Since we only require the shortest path from a single starting point, Forward Search has better computational efficiency than DP. For the A* algorithm, I have used L1 distance as the heuristic. Since the robot can only move along the horizontal and vertical axis and not diagonally, the minimum distance travelled between any 2 points will be their L1 distance rather than their Euclidean distance. There may be many paths with the same length to reach the goal. This might result in a longer search since the A* might explore all of the paths with the same values. As such to further reduce computation, I have used the insertion order of any 2 nodes with the same value as a tie-breaker. Under this regime, the A* algorithm will first explore the node that is inserted later.

2. CSDA

For the CSDA model, the state space is discretized adaptively with variable size based on the velocity of the controls as shown in the figure on the right. For controls with smaller velocity, we will draw a smaller box around the destination point. When the box of a new point overlaps with a previous box, we will only retain the configuration with the smaller value.



For controls, I have used a set of 3 fixed and 2 sampled movements. The fixed controls are:

- Full speed forward: (1, 0)
- 45-degree anti-clockwise rotation follow by a forward: (0, -pi/2), (v, 0)
- 45-degree clockwise rotation follow by a forward: (0, pi/2), (v, 0)

where the speed of the forward motion after the turn is adaptively determined proportionate to the L2 distance to the nearest obstacle. At every turn, we sampled 1 control for each rotation direction, consisting of a less than 45-degree turn followed with a forward motion based on the adaptive velocity. As shown in the figure above, the sampled controls add stochasticity into the exploration. This helps to avoid locally optimal path of structure exploration with only fixed controls.

3. DSPA MDP

In the MDP setting, we formulate the reward function as such:

- For any actions that resulted in collision, we penalized the agent with a -25 reward.
- For any other actions, the reward given is -1.

I have solved the MDP using value iteration without decay. We translated the given probability table into a transition matrix for each state and action. The probabilities of any next state that will result in a collision is set to be 0. The above formulation of the reward function depends on the current state, the action taken and the next state. I have

reformulated it to depend on only the current state and action by giving the agent the expected reward based on the transition probabilities (probability of collision next state are not zeroed here). Lastly, to optimize the computation time, I have leveraged on numpy fast “vectorized operations” and implemented a vectorized version of the value iteration algorithm.

4. LESSONS & INSIGHTS

The hybrid A* in CSDA is hard to tune. With too many controls, the algorithm takes a long time to complete due to the higher branching factor associated with more controls. With only a few fixed controls, the paths obtained are sub-optimal due to the structured exploration. If we tune the velocity lower to explore the map more meticulously, the algorithm again takes a longer computation time. Adding stochastic controls might result in slow exploration where hybrid A* get stuck in optimizing its path within a small neighboring area. Particularly, I had to tune the velocity to ensure that the controls for the same starting point will have minimal overlaps. I have discovered that using a set of fixed and stochastic controls with adaptive velocity helps the algorithm to find the goal more efficiently.

For the MDP, it took me a long time to figure out that the transition probabilities for a collided next state needs to be 0 for the value iteration to work. Otherwise, the values will converge to a non-optimal solution with one or more “looped” path (e.g., node1’s “optimal” action goes to node 2 and node 2 “optimal” action goes to node 1).