Search and Navigation

Created by: Shangzhou Ye
Last updated: Oct. 28, 2019

PART A

Question 1

In the code and the following discussion, three coordinates was defined for different purposes. They are named as cartesian coordinate, cell coordinate and plot coordinate. The definition of each coordinate will be discussed in detail. Helper functions including *cart2cell, cart2plot, cell2plot* were written in the code for transformation between these coordinated.

1. Cartesian coordinate: The coordinate used by the world map. For example, the world is limited within (-2, 5) in the x-direction and (-6, 6) in the y-direction. These limitations are represented in the cartesian coordinate.

2. Cell coordinate: Cell coordinate has an origin at the left-bottom corner. The unit length in the cell coordinate is the same as the grid size. In other words, the unit length in the cell coordinate is equivalent to 0.1 m in the cartesian coordinate when the grid size is 0.1 m.

3. Plot coordinate: The origin of the plot coordinate is at the center of the left-top cell in the plot. This coordinate is specifically used by the scatter plot function when plotting the robot trajectory on the grid. Its y-axis has a positive direction pointing downwards.

Figure 1 has an illustration of the three coordinated defined above. These coordinates will be used in the code and the discussion below.

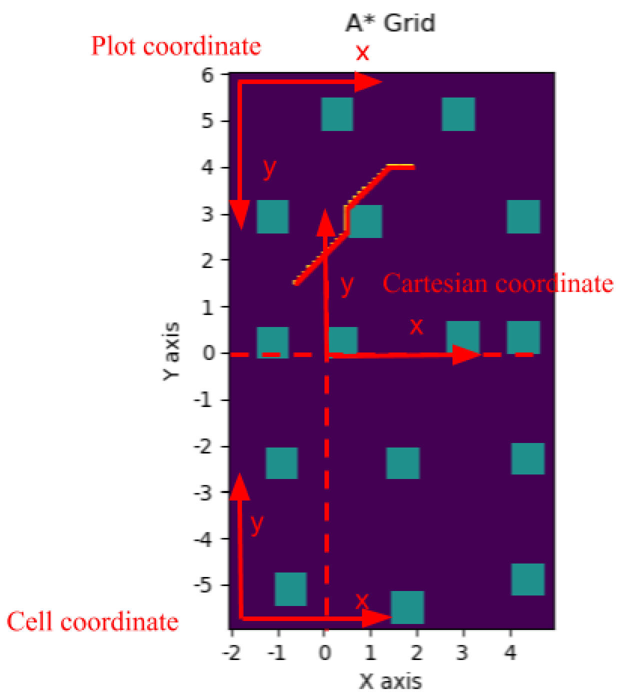

Figure 1. Illustration of three coordinates used in the code.

Question 2

The heuristic function used is the straight-line distance between the current position and the goal position. A heuristic function should never overestimate the cost of reaching the goal in order to be admissible [1]. In other words, the cost calculated by the heuristic function should be to obey less strict conditions compared to the true cost. The given true cost requires +1 to move to a neighbor, which only allows eight directions of transition for a robot at one step. The straight-line distance did not have this constraint on the transition directions. Meanwhile, the true cost would +1000 to move into an occupied cell while the straight-line cost does not take that into account.

The straight-line distance is calculated by:

$$h = scale * \sqrt{(x - x_G)^2 + (y - y_g)^2}$$

, where x, y represents the current position and x_G, y_G represents the goal position. Because the true cost of the robot moving diagonally is 1 and the straight-line distance would be $\sqrt{2}$, the scale parameter is needed. Otherwise, the heuristic function may overestimate the true cost in this case.

Therefore, the path expansion is based on:

$$f(n) = g(n) + h(n)$$

, where g denotes the true cost and h denoted the heuristic function.

Question 3

The visual display of the planned path is shown in Figure 2. Blue cells are the occupied cells and yellow cells are the planned path. In all three tests below, the A* search algorithms successfully find the path to the goal.

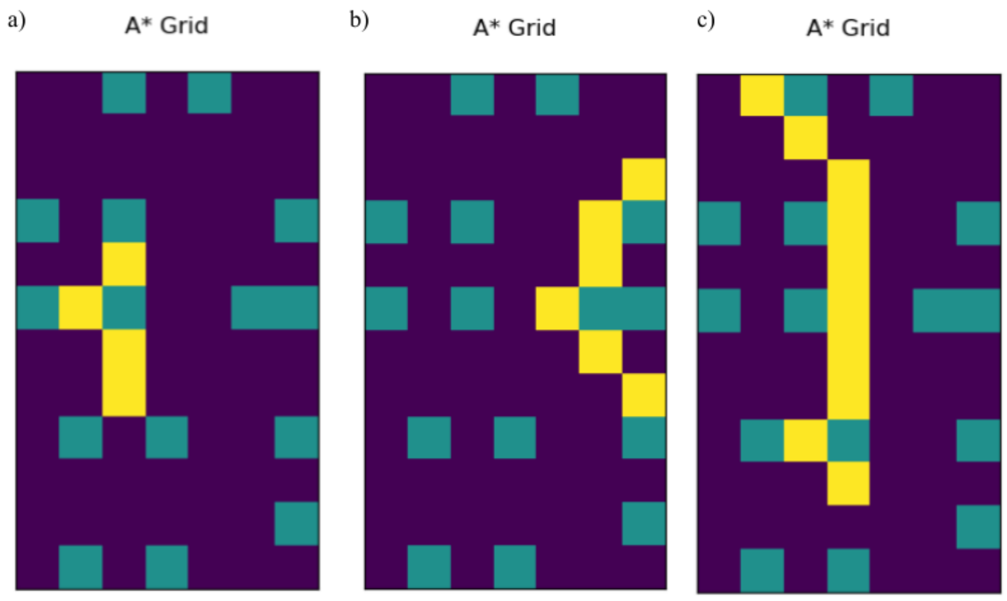

Figure 2. Searching results of a) test 1: from [0.5, -1.5] to [0.5, 1.5]. b) test 2: from [4.5, 3,5] to [4.5, -1.5]. c) test 3: from [-0.5, 5.5] to [1.5, -3.5]

Question 4

The A* searching algorithm was then modified to be 'online'. The difference between online and offline is that, when searching offline, the robot begins the movement after the expansion is completed. As the robot has knowledge of the obstacles in 'offline' searching, the algorithms can expand completely before it starts to move the robot. The open list has cells that are not just their immediate neighbors. When the robot begins the movement, it only chooses the optimal path find by the algorithms. In the ‘online’ search, however, the robot may need to travel back and forth to find the best path when a new obstacle was detected.

I gave my ‘online’ searching robot the capability to memorize the position of the obstacles it has found. In other words, the robot is building a map while exploring the world, it may increase the computational load but would possibly find the path quicker when traveling to some area it has visited before.

No difference was noticed between the implementation with map building and without map building since in all three tests, the robot is not traveling back to the cells it has visited before.

The implementation of the ‘online’ searching version is based on the ‘offline’ version. The steps are summarized below:

- At the very beginning, the robot’s memory of the obstacles’ positions is an empty array. In other words, it does not have any *prior* knowledge of the world.
- The robot then looks at its eight immediate neighbors and add the obstacles to its memory (map) if there is any.
- The robot completes an A* star search based on this current memory (with knowledge on the current immediate neighbors and an empty array elsewhere) and finds a path. Since the robot does not have knowledge of the obstacles that are not in its immediate neighbors, the planned path would just be a direct path to the goal.
- After that, the robot moves to the position next to its current position on the planned path, then looks at its new neighbors, add them to the memory, and re-plans with A* searching algorithms.
- The robot would repeat this ‘search, look, plan, move’ cycle until it moves to the goal.

Question 5

The visual display of the planned path in those three tests is shown in Figure 3.

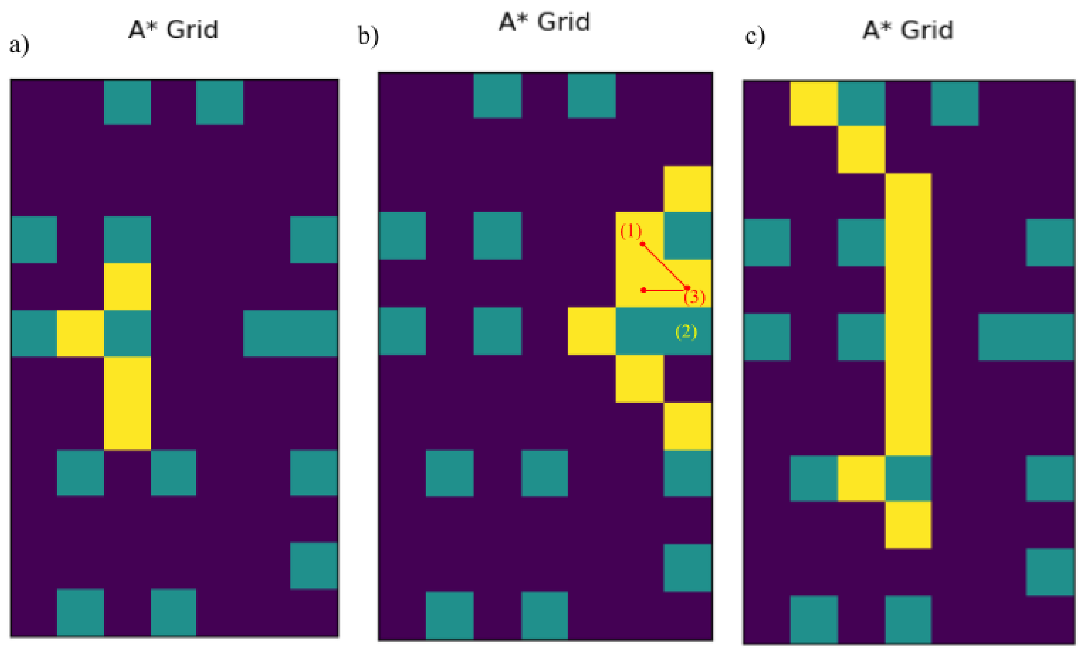

Figure 3. ‘Online’ searching results of a) test 1: from [0.5, -1.5] to [0.5, 1.5]. b) test 2: from [4.5, 3,5] to [4.5,-1.5]. c) test 3: from [-0.5, 5.5] to [1.5, -3.5]

Interestingly, the ‘online’ searching result is different from that of ‘offline’ searching in the second test. As highlighted in red lines in Figure 3b, the robot goes diagonally to the cell at the right side first, then goes to the cell on the left. This behavior costs the robot an extra step to get to the goal.

This behavior can be explained by the difference between the ‘online’ and ‘offline’ searching. When the robot is at cell (1), it can not see the obstacle at cell (2), after calculating the heuristic (straight-line distance), the robot found it would cost less to move diagonally to cell (3) then move one cell downwards. Only after the robot moves to cell (3), it realized there are two obstacles below itself, and the robot has to go around it.

Question 6

The inflated obstacles are approximated by squares. The inflation method is shown in Figure 4. Each obstacle was inflated by 0.3m in each direction. The corresponding cells were calculated. After that, all the 49 cells as shown in the figure below are labels as occupied.

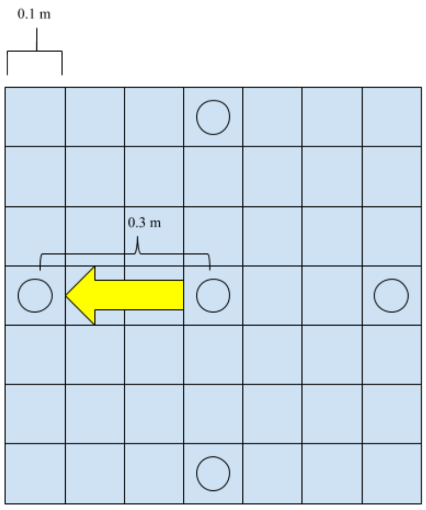

Figure 4. Illustration of the inflated obstacle.

Question 7

Figure 5 is the visual display of the results (‘online’ searching) with 0.1 m grid size and inflated obstacles. As pointed out by the red arrow in the figure, the ‘online’ algorithm commands the robot to move left with an intention to move directly to the goal. Only when the obstacles are within the immediate neighbors of the robot, it decides to move down to go around the occupied cells.

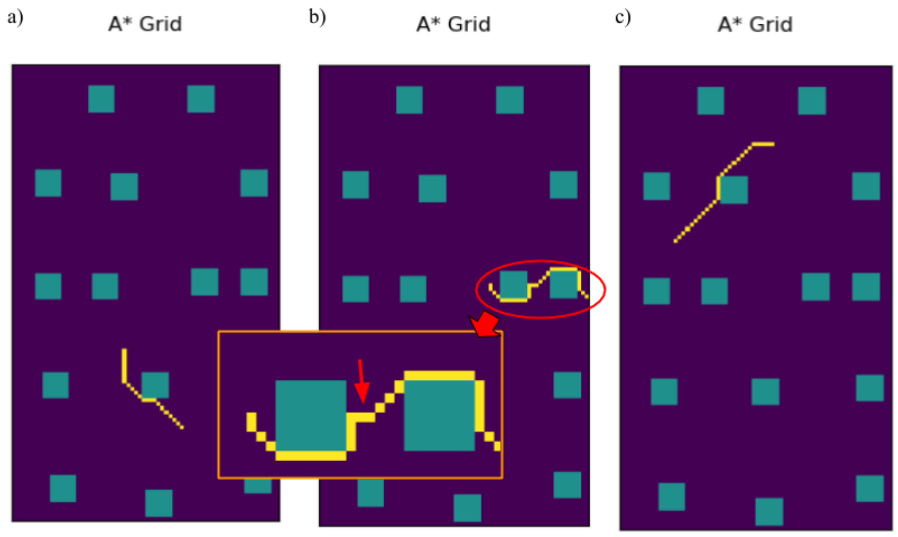

Figure 5. Searching results with 0.1m grid size.

Two other true cost functions are tested. Figure 6a shows the results when the diagonal movement cost is $\sqrt{2}$. The cost of $\sqrt{2}$ was chosen so that the diagonal movement costs more than a single left/ right movement, but less than a combination of moving left/right and up/ down. Comparing the three test situations between Figure

6a and Figure 5, it can be concluded that the robot is avoiding diagonal movement in the later cost function when a single left/ right or up/ down transition is possible. It is more obvious in the second test, where the robot chose an entirely different path to go around the obstacles.

Figure 6b shows the results when the diagonal movement cost is 2. In this case, the movement cost of moving diagonally is the same as the combination of a left/ right movement and an up/ down movement. It is shown in the result that the robot tends to execute more horizontal or vertical transitions.

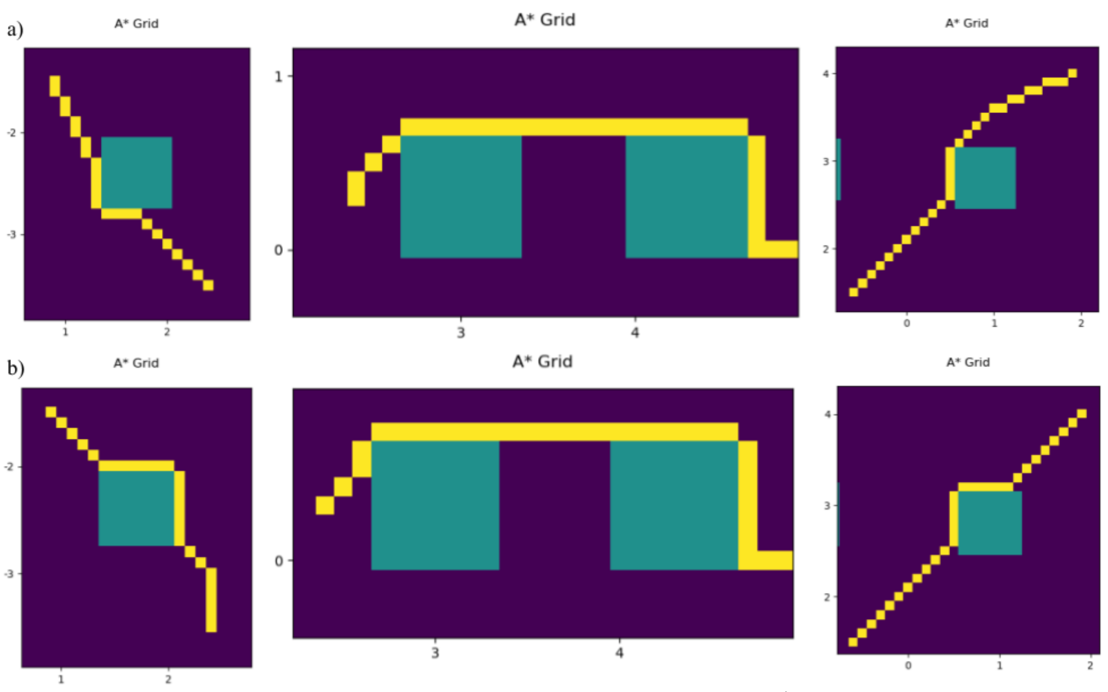

Figure 6. ‘Online’ searching results when a) diagonal movement cost is $\sqrt{2}$. b) diagonal movement cost is 2.

Question 8

The controller is achieved by a proportional steering control and proportional speed control. For speed control, the velocity of the robot is proportional to the distance between the current position of the robot and the goal. The Euclidean distance between the robot and the goal is calculated by the equation below:

$$Distance = \sqrt{(x^* - x)^2 + (y^* - y)^2}$$

, where (x^*, y^*) is the goal point in the plane. The velocity command is the calculated distance multiplied by a proportional gain for linear velocity.

$$v^* = K_v \cdot Distance$$

To calculate the desired steering angle of the robot, the arctangent function was used. The steering angle in the world frame was calculated by the equation below:

$$\theta^* = tan^{-1}\frac{y^* - y}{x^* - x}$$

In the code, the arctangent function was achieved by using *atan*2 function in the *numpy* library to ensure the angle is within $-\pi$ and π.

With a proportional controller, the angular velocity command was calculated by [2]:

$$\gamma = K_h(\theta^* \ominus \theta),\ K_h > 0$$

, where the $\ominus$ operator forces the angle difference to stay within $-\pi$ and π. Otherwise, when multiplying the angle difference with the proportional gain, the same error with a different expression would get a different angle velocity command.

To make the controller more realistic as instructed, the linear and angular acceleration was restricted within $0.288\ m/s^2$ and $5.579\ rad/s^2$, respectively. The tolerance was set to be 0.01 m in the code. (If the distance between the current position of the robot and the goal is smaller than 0.01 m, it is regarded as having reached the goal.)

The proportional gain for the linear and angular velocity was chosen to be 0.5 and 4 respectively. The final results are shown in Figure 7a. The robot is making a 180° turn and moving from (0, 0) to (0, 0.1) in the figure. The robot had an acceptable overshoot as its position on the x-axis went to -0.02. Figure 7b shows the results without the acceleration restrictions. It can be concluded that without the acceleration restrictions, the robot can is able to make the turn more stable as shown in the figure. However, it may not be realistic in the real world.

Figures 7c and 7d show the movement of the robot with a smaller proportional gain and a larger proportional gain respectively. When the gain is too small, the robot travels around with a large offset, which may cause the robot to move to its neighbor cell during the movement. A larger gain causes oscillation during the movement as shown in Figure 7d.

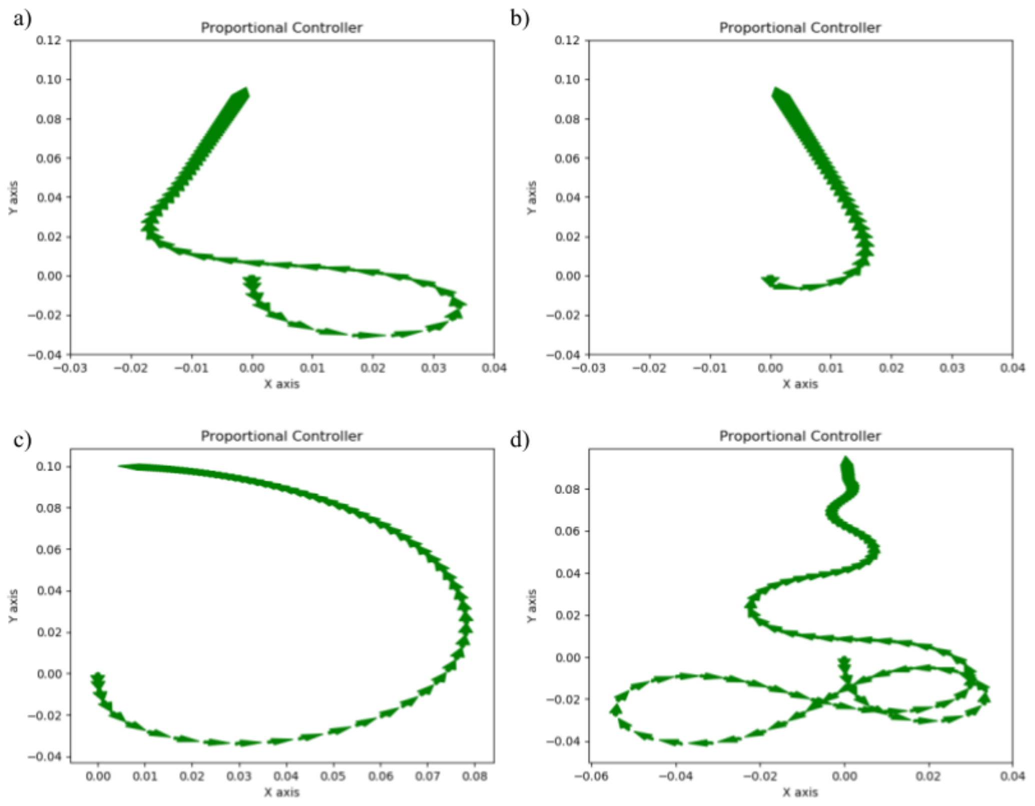

Figure 7. Proportional controller a) final results ($K_v = 0.5,\ K_w = 4$). b) results without acceleration restrictions. c) results with a small proportional gain for angular velocity. d) results with a large proportional gain for angular velocity

Question 9

Figure 8 shows the robot tracking the waypoints (path) generated in step 7. Yellow cells are the planned path and red arrows show the actual robot trajectory. The robot is able to move smoothly along the path.

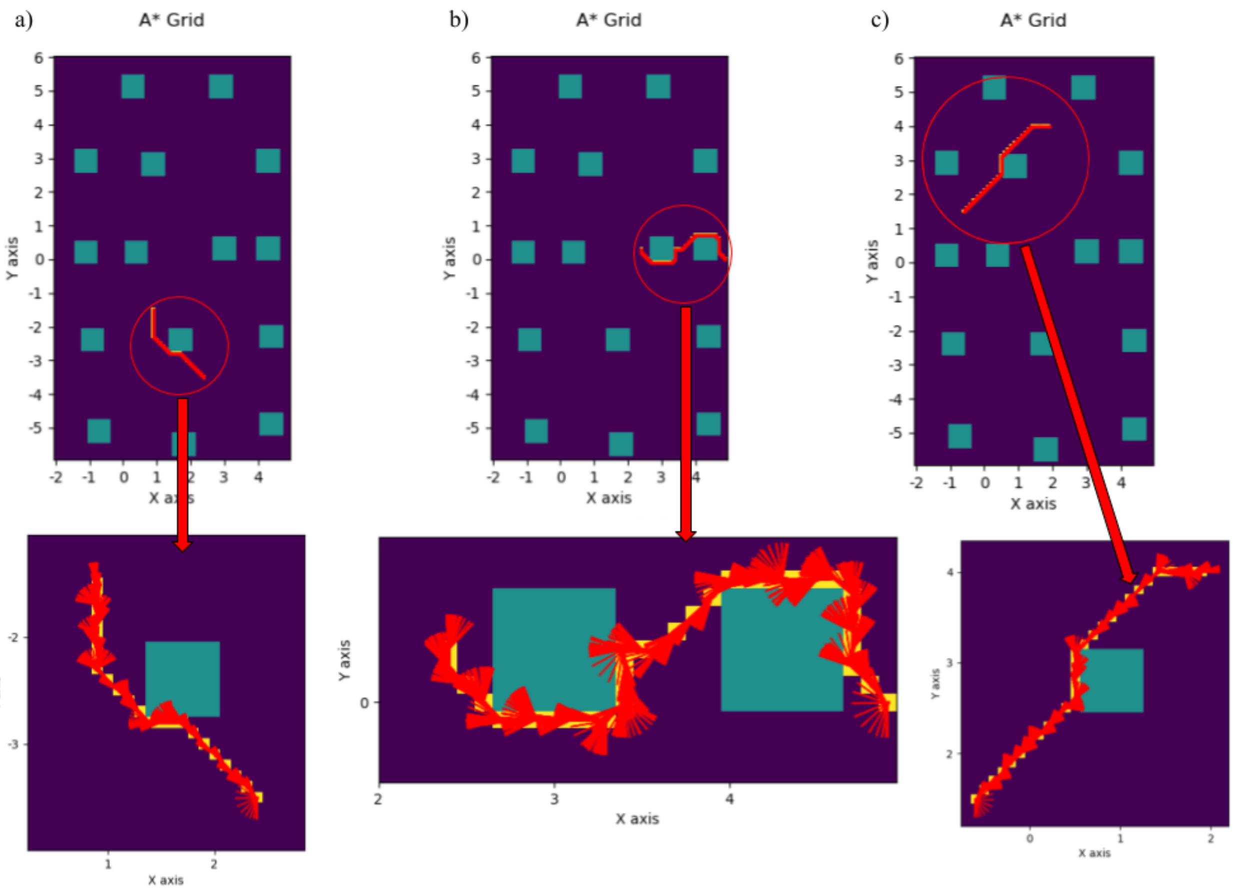

Figure 8. Navigation trajectory of the robot a) test 1. b) test 2. c) test 3.

To make the waypoint tracking more realistic, the noise was added to the motion model. Normally distributed noise with a standard deviation of 0.002 m was added to the x and y position of the robot as well as the heading. Figure 9 shows the comparison between the robot trajectory with noise and without noise.

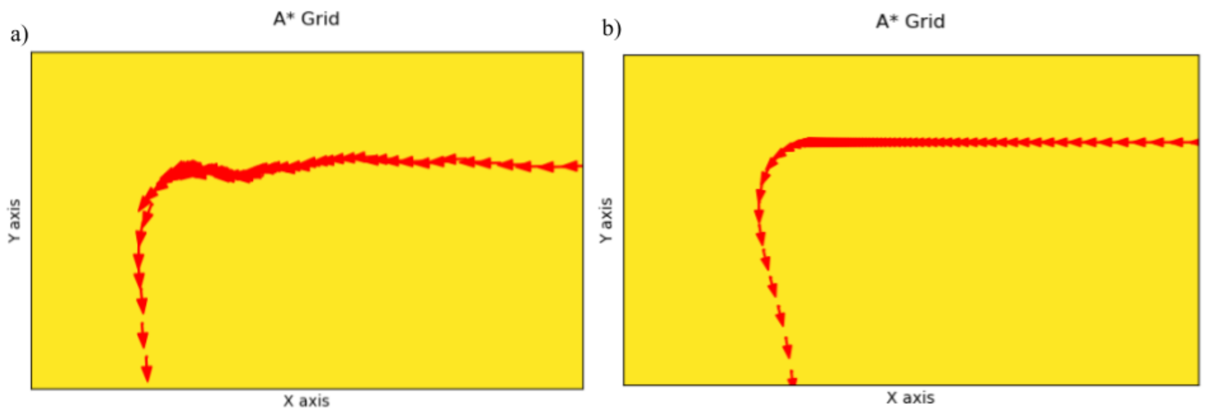

Figure 9. Navigation trajectory of the robot a) with noise. b) without noise.

Interestingly, it is noticed that, with noise, the robot can move into its neighbor cell when moving diagonally. As shown in Figure 10, when the robot is trying to travel from the cell (1) to cell (2), it moves across the cell (3) when doing the transition. It is not the case when there is no noise as shown in Figure 10b. To explore this situation more, the time interval (dt) was set to 0.05 instead of 0.1. The results are illustrated in Figure 10c, where, in many diagonal transitions, the robot goes into its neighbor cell because of the noise added on the motion model. It can be inferred that when doing online planning and executing the robot motion simultaneously in question 10, this behavior will cause the robot to move into an unexpected neighbor cell, and have to re-plan.

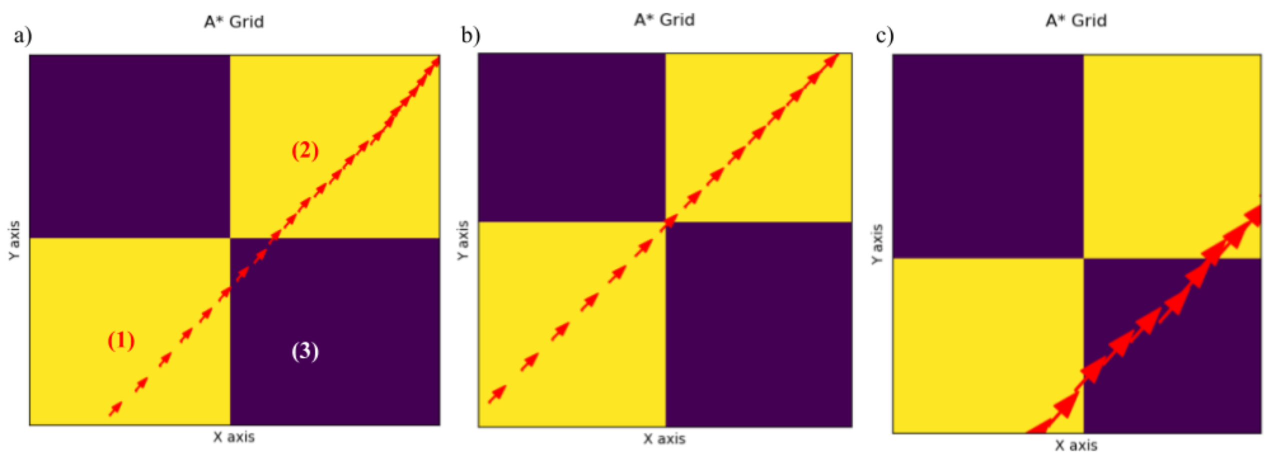

Figure 10. Navigation trajectory of the robot a) with noise. b) without noise. c) with time intervals of 0.05 s (zoomed in).

Question 10

Figure 11 illustrates the 'driving while planning' results. The algorithm was implemented with several modifications compared to question 9.

- After each step of planning, the robot starts to execute the motion with the proportional controller from its current position to the next target node.
- If the robot successfully reached the goal, the searching algorithm begins to plan the next step as question 9.
- If the robot moves to an unintentional node instead of the target node, the robot would re-plan using its current position and the goal position.
- The current node, which the robot was unintentionally moved into, was added to the path. The targeted node, which the robot fails to reach, was deleted from the path.
- After re-planning, the robot controller moves the robot towards the next node.

As discussed in question 9, when the robot moving diagonally with noise, the robot can unintentionally move to its neighbor. This behavior is not uncommon in the results as shown in Figure 11. In Figure 11d, it is shown that the robot moved into cell 3 while its initial goal is moving from cell 1 to cell 2 directly.

The situation would become worse if the controller gain is too small or too large as the robot would frequently move to an unintentional cell and re-plan. In the current setup, the proportional gain is suitable and the robot has good performance. The robot is able to smoothly reach the goal with only a few deviations from the initially planned path (as in question 9)

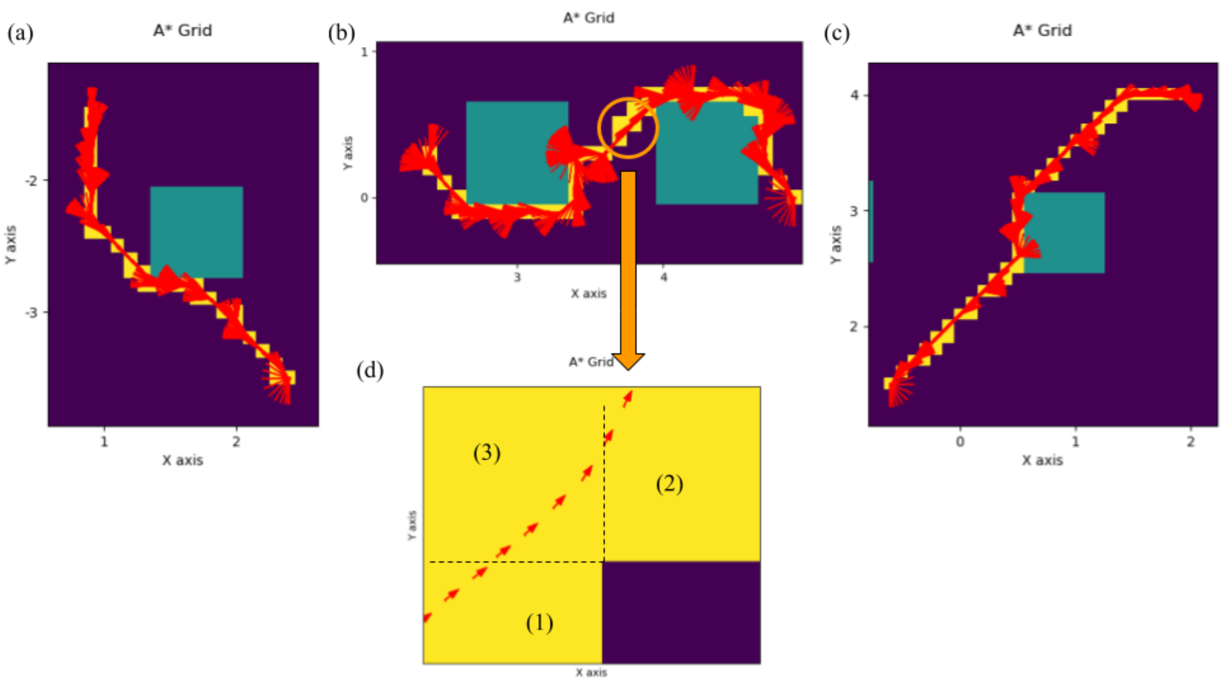

Figure 11. Navigation trajectory of the robot a) test1. b) test2. c) test3. d) when the robot moves to unintentional node

Question 11

'Driving while planning' results with the start/ goal position listed in question 3 are shown in Figure 12. Compared to the fine grid, the coarse grid would be less computationally expensive. However, if the actual position of the obstacle is at the margin of the cell, using a coarse grid would be less accurate then inflating the obstacle with the fine grid. There would be the possibility that the robot collides with the obstacle because of the noise in the execution of motion control when using a coarse grid.

Because the grid size of the coarse grid is larger, there would be less possibility that the robot would unintentionally move into its neighbor cell when turning its heading. On the other hand, however, since the robot is always targeting the center of each cell, the traveling distance of the robot would be larger with the coarse grid.

As pointed out by the blue arrow in Figure 12c, that cell is actually an occupied cell. However, as the robot steps into the cell because of the noise in motion control when trying to move diagonally, it has already received the penalty. According to the implemented algorithm, the robot would directly be targeting the re-planed next cell. This behavior would increase the possibility that the robot colliding with the obstacle.

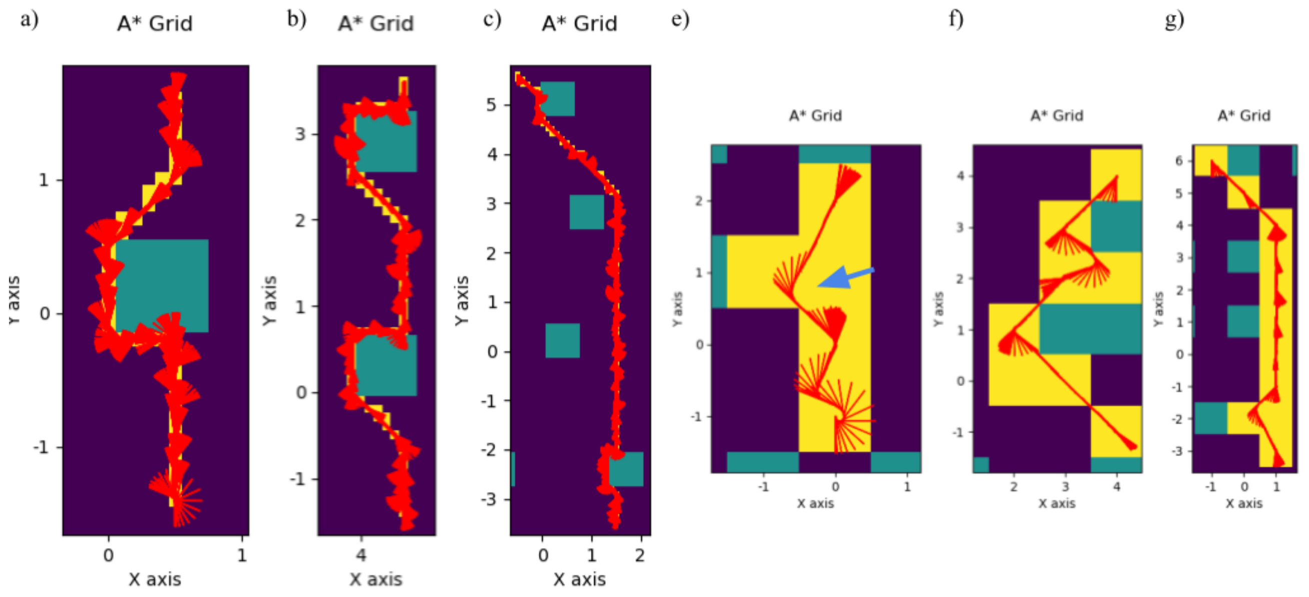

Figure 12. Navigation trajectory of the robot with the fine grid: a) test1. b) test2. c) test3. with the coarse grid: d) test1. e) test2. f) test3.

Question 12

There are a few simplifications this simulated world made.

- The robot position in the world is uncertain. It is impossible for the robot to have the information on its accurate position without any uncertainty. As covered by previous materials in this course, the robot can only have a 'belief' on its position, which is usually a probability distribution.
- The simulated robot does not have constraints on its linear and angular velocity, which may not be the case in real-world applications.
- The simulated robot does not require smooth acceleration. In other words, the robot does not require the acceleration to be continuous, which may cause the robot motion to be jerky.
- The stability of the proportional controller is not guaranteed. It may cause issues when the robot is required to make more challenging behavior.
- In the cases with high speed, acceleration or load, the kinematic model may not be valid [3].

Reference

[1] S. J. Russell, *Artificial intelligence : a modern approach*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2010.
[2] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, vol. 73. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
[3] F. D. Boyden and S. A. Velinsky, "Limitations of Kinematic Models for Wheeled Mobile Robots," in *Advances in Robot Kinematics and Computational Geometry*, J. Lenarčič and B. Ravani, Eds. Dordrecht: Springer Netherlands, 1994, pp. 151–160.