Northwestern University

ME 469: A* PLANNING

MICHAEL RENCHECK

Question 1: Building a Grid

A robotics study was conducted at the Autonomous Space Robotics Lab at the University of Toronto Institute for Aerospace Studies (UTIAS) to use multiple robots for Cooperative Localization and Mapping. In conjunction with a paper, the full data set for the robots used in the experiment was also published.[1] For this report, the landmark locations for dataset 1 were used to create a similar course, but without the interior walls. The work space was divided into 1x1m squares and cells with a landmark were classified as occupied. The environment was also approximated to a rectangle instead of using the outside boarder walls. Figure 1 shows an image of the real course and the software generated grid space. For the generated grid space, the obstacles are denoted by the cyan triangles and the purple infill shows that the cell is classified as occupied.

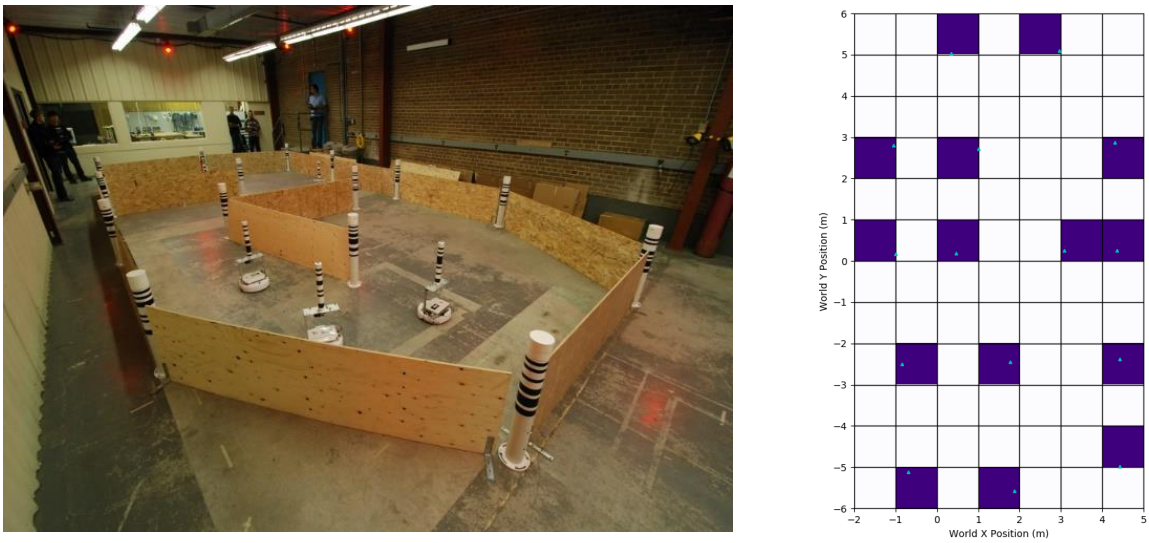

Figure 1. Picture of the area for dataset 1[2] and the generated grid space

Question 2: Designing the A* Algorithm[3]

The A* algorithm was first implemented on the Shakey robot developed at Stanford in the early 1970s, and since then it has been the inspiration of the continued development of AI in robotic path planning. While newer algorithms, such as D*, have been design to help account for the physical world challenges associated with path planning for robotics, A* is still suitable and also currently being used by different robots today. When applied to path planning, the A* algorithm determines a complete and optimal path to a specified goal location from a known starting location. It accomplishes this by discretizing the state space into a grid and identifying the "lowest cost" path of adjacent cells between then given points. The cost is a parameter that the algorithm computes to define how appealing that respective grid cell, or node, is to include on the final path. In general, the cost is determined based on the scenario or application, but for path planning the main consideration for cost is usually distance. So, by determining the "cheapest" path from the start to the goal, the algorithm should identify a path optimized for distance.

Before making any design decisions about the algorithm, assumptions about the scenario must be made to base the design decisions on. For this implementation, the robot was allowed to move to any adjacent node (forward, sideways, or diagonal) that was one space away. If a node also contained a landmark, it was flagged as occupied and the algorithm should avoid moving through these cells. The main functions that the A* algorithm uses to identify the cheapest path are the true cost, the heuristic, and the evaluation cost functions. The true cost and the

[1] (Halpern, Lui, Barfoot, & Leung, July 2011)
[2] Image provided by UITAS
[3] (Russel & Norvig, 2010) & (Patel, 2019)

heuristic functions are considered design decisions and as such will change based on the application. For this implementation, the cost functions that yielded the best results across all test scenarios are shown below.

$$h(n') = \max\left(|x_g - x_{n'}|, |y_g - y_{n'}|\right) \qquad (1)$$

Where:
h(n): heuristic function, estimated cost to get to the goal from node *n'*
x_g , y_g*:* the x, y position of the goal node
$x_{n'}$, $y_{n'}$*:* the x, y position of thee successor node

$$c(n, a, n') = \begin{cases} 1000, & if\ n' is\ occupied \\ \sqrt{2}, & if\ n'\ is\ diagonal \\ 1, & else \end{cases} \qquad (2)$$

Where:
c(n,a,n'): the cost of some action *a*, to move from node *n* to node *n'*

$$g(n') = g(n) + c(n, a, n') \qquad (3)$$

Where:
g(n'): true cost function, the true cost of moving to successor node *n'* from the start location
g(n): the true cost of moving to current node *n* from the start location

$$f(n') = g(n') + h(n') \qquad (4)$$

Where:
f(n'): evaluation cost function, the evaluated cost of node *n'*

The corner stone feature of the A* algorithm is that it is both complete and optimal. Complete meaning that if a path from the start to the goal exists, the algorithm will always find it. And optimal meaning the final path identified is the lowest cost path of all possible solutions. To achieve optimality, the heuristic function *h(n')*, must be both admissible and consistent. To be admissible, the heuristic must never overestimate the cost to reach the goal from the node *n'*. To be consistent, the heuristic must abide by the following:

$$h(n) \leq h(n') + c(n, a, n') \qquad (5)$$

For this implementation, the heuristic function being used is a function under the diagonal distance family called the Chebyshev Distance. The estimated cost to the goal is set as the maximum difference in position from the node to the goal along one of the axes. This function is admissible and consistent in the context of this application because it will always return an estimated cost less than or equal to the true cost to reach the goal from a given node. For example, given a node at (2, 3) and a goal at (2, 5), the heuristic will determine an estimated cost of 2. Given that the minimum value that *c(n, a, n')* can yield is 1, the heuristic can never be greater than the true cost and is admissible. Also, assuming an unobstructed path to the goal from the node, the position of closest successor node would be (2, 4), yielding an estimated cost of 1. Again, since 1 is the smallest value that *c(n, a, n')* can yield, the heuristic is guaranteed to not violate Equation 5 and is consistent.

Question 3: Implement the A* Algorithm

An A* algorithm was developed using the design considerations and functions discussed above and was applied to the data points show in Table 1. The results for each test are shown in Figure 2. The final path determined by the A* algorithm is denoted in red and all of the nodes that the algorithm evaluated are identified by the black dots. The results from Test B demonstrate the advantage of using A* over a similar search method such as Dijkstra's Algorithm (DA) or Greedy Best First Search (GBFS) [4]. DA uses a heuristic function of zero so the evaluation cost is only based on the true cost. The results for the final path would have been similar to those

[4] (Russel & Norvig, 2010)

shown below, but the number of nodes evaluated would have increased as DA expands every node starting from the initial position and radiates outward. For a small grid and short distance, this compute difference is negligible, but as the number of cells increase, the compute time of DA will be much greater than A*. Using GBFS, the true cost is not considered and only relies on a heuristic to identify the lowest cost path. This results in a behavior that prioritizes nodes with the shortest distance to the goal. Depending on the heuristic function used, the GBFS has the potential to get stuck in a corner at node (4.5, 1.5). The methodology of how GBFS tries to find an alternative path can lead the algorithm to getting stuck in an infinite loop resulting in never finding the goal. A* has a more robust method to go back and consider other evaluated nodes, so if the heuristic led the path into that same corner, the algorithm could have gone back to a previous node when no more options were available.

Table 1. List of start and goal positions to test for coarse grid

Test	Start Position	Goal Position
A	(0.5, -1.5)	(0.5, 1.5)
B	(4.5, 3.5)	(4.5, -1.5)
C	(-0.5, 5.5)	(1.5, -3.5)

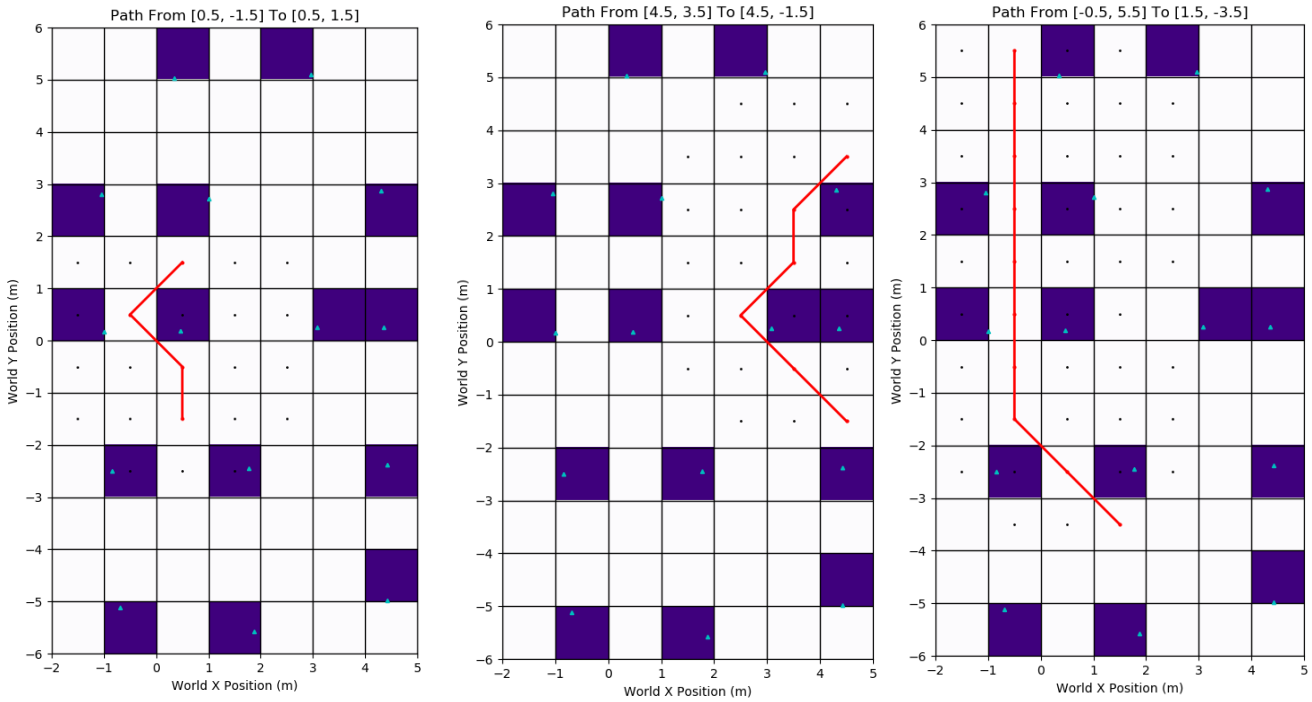

Figure 2. Question 3 test results for test A (left), B (center), and C (right)

Question 4: Designing the "Online" A* Algorithm

Another key feature of A* is how it maintains a running history of explored nodes. It manages this information by storing it in two separate lists called the Open List and the Closed List. The Closed List is a compilation of all expanded nodes, meaning that a node was identified as potentially being on the lowest cost path and the algorithm evaluated the eight potential neighbor nodes. Once a node is added to the Closed List it can never be reevaluated or re-expanded. The Open List is a compilation of all nodes that the algorithm has evaluated due to expanding another node. Every A* algorithm begins with an Open List containing just the start location and an empty Closed List. A* then selects the node on the Open List with the lowest evaluation cost $f(n)$, moves it into the Closed List, and expands the eight potential neighbor nodes. These eight nodes are added to the Open List and the process is repeated until the goal is found or there are no longer any nodes on the Open List. For this implementation, if multiple nodes have the same evaluation cost, the selection is then based on the lowest heuristic.

The difference between “online” A* and the A* algorithm used for Question 3 is how the Open List is managed. In “online” A*, the Open List is only allowed to contain nodes neighboring the most recently expanded node. This means that the maximum size the Open List can ever reach is eight nodes. Then once the next node is selected from the Open List, the remaining evaluated nodes are deleted from memory. In practice, the change to the algorithm is very small only requiring one extra if statement, but it has a large impact on the behavior of the algorithm for different scenarios.

The advantage of the “online” A* approach is that it models how a real time search would function when applied to a physical robot navigating in an environment. This modification allows the user to simulate the robot’s behavior in a virtual environment in order to improve the efficiency of the cost functions without risking harm to the physical robot or the environment. The drawback to “online” A* is that it is now possible for A* to not be a “complete” algorithm. Rather, there is now a nonzero chance that the algorithm will not find a path when a solution does exist. In Question 3, it was noted that if A* expanded a node that resulted in no valid neighbors, it has the ability to jump back to a previous evaluated node stored on the Open List and continue the search. However, if a robot is planning in real time, it does not have this luxury of teleporting multiple squares away to continue searching and thus has the potential to get stuck. This is one example that led to the development of more algorithms specialized for robotics and real time navigation in the physical world.

Question 5: Implement the “Online” A* Algorithm

The “online” A* algorithm developed in Question 4 was applied to the same test sets shown in Table 1 and the results are shown below in Figure 3.

For the heuristic and true cost functions used in this implementation, the resulting lines ended up actually being identical between the two A* algorithms. The difference in the visualizations comes down to the lack of black dots in the surrounding nodes. This is due to the change made to make the A* search “online”. The Closed List is only comprised of the nodes include on the final path and the Open List is erased after a neighbor node is selected to be expanded. It was noted that during trial runs of Test B, the “online” search method lead the final path through the occupied cell at (0.5, 4.5) depending on the heuristic being used. This is due to the “online" modification forcing the search to select the next node from only the immediate neighbors. Since our cost function says that an occupied cell has a cost of 1000, the algorithm saw it as traversable, but with a high cost. However, since the search is restricted from going to any node on the Closed list, the occupied node was the best available option. This scenario demonstrates the benefit of using simulation before physical testing, because it revealed a potential issue and allowed for experimenting with different true cost and heuristic functions to try and avoid that scenario.

Question 6 & 7: “Online” A* with a Finer Grid

In order to make this simulation more realistic, the grid cell size was reduced to 0.1x0.1m but maintaining the same overall boundary dimensions. As seen in Figure 1, the landmarks do not actually take up a full square meter as they are just small tubes and a cell size of 0.1x0.1 more accurately represents the footprint. In order to also account for the size of the robot, the footprint of the landmark was expanded by ±0.3m in the x and y directions. This creates a buffer region around the known landmark position that will inform the algorithm to set more than just the cell containing the landmark location as occupied. The “online” A* was then applied to a new set of tests to analyze the search behavior for the finer grid. The data points for each test are shown in Table 2 and the results for each test are shown in Figure 4.

Observing the layout of the occupied regions shows that using a finer grid cell size makes the scenario more realistic compared to the actual landmark location as it more closely resembles the photo in Figure 1. During experimentation, Test B yielded drastically different results depending on the heuristic and true cost functions used. For example, using the Euclidean Distance formula, biases the search to select the cell with the shortest straight-line distance to the goal. This behavior caused the “online” A* to move up in between the two obstacles between

the start and the goal. This test did end up completing, but the final path was not the shortest path. The heuristic was then changed to the method shown in Equation 1, which determine the estimate cost by distance along only one axis. This biased the algorithm to prioritize node that reduced the distance along that axis. This allowed the "online" A* to plan past the gap before heading up toward the goal.

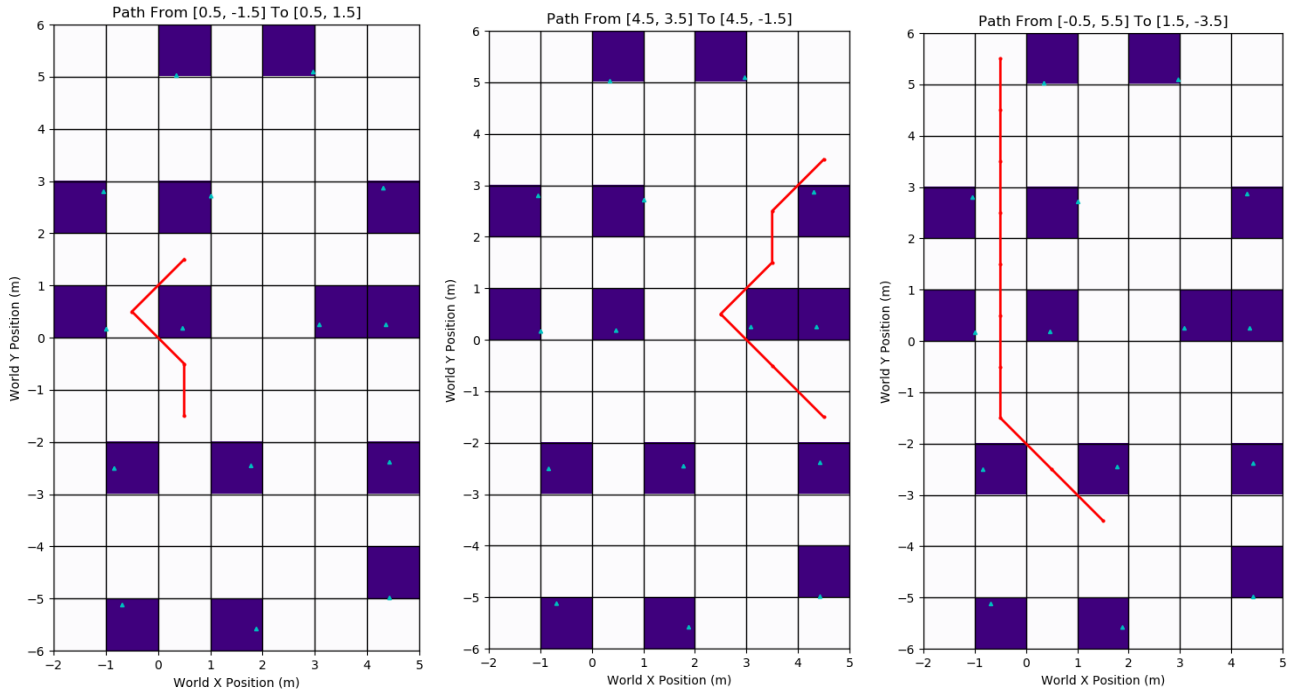

Figure 3. Question 5 test results for test A (left), B (center), and C (right)

Table 2. List of start and goal positions to test for fine grid

Test	Start Position	Goal Position
A	(2.45, -3.55)	(0.95, -1.55)
B	(4.95, -0.05)	(2.45, -0.25)
C	(-0.55, 1.45)	(1.95, 3.95)

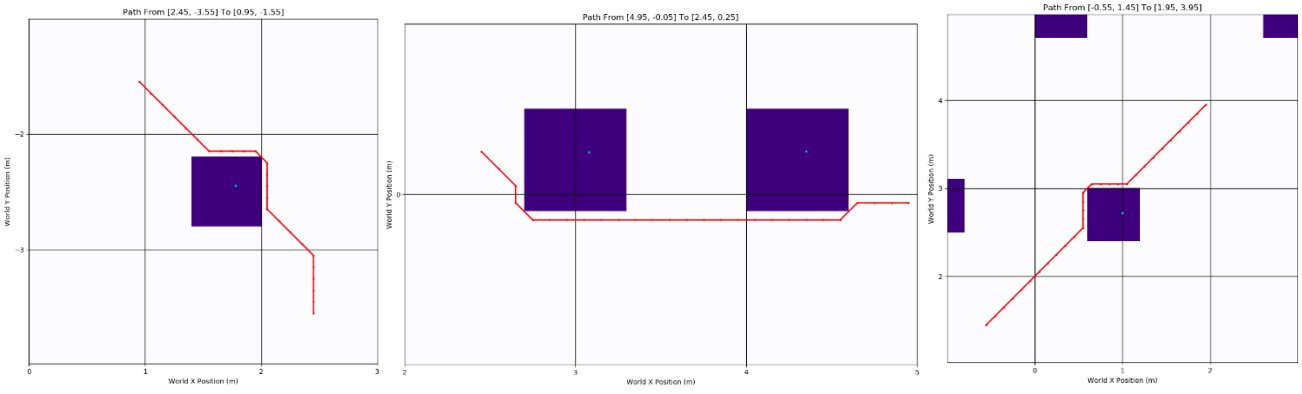

Figure 4. Question 7 test results for test A (left), B (center), and C (right)

Question 8: Designing a Motion Model

In order to simulate a physical robot driving the path determined by the "online" A* search, an inverse kinematic controller was developed to compute a linear and angular velocity command. This command was then used to propagate the simulated robot along the path moving from node to node, until reaching the goal. In order to maintain a realistic approach, the simulation had constraints on the acceleration of the robot based on the data provided by UITAS. It was also assumed that the robot perfectly knew its position after each movement. The equations used to control the robot simulation are the following:

$$\begin{bmatrix} v_t \\ w_t \end{bmatrix} = \begin{bmatrix} k_v * (\sqrt{(x_t - x_n)^2 + (y_t - y_n)^2} + \varepsilon_v \\ k_w * (\Delta\theta) + \varepsilon_w \end{bmatrix}; \ \Delta\theta \mapsto (-\pi, \pi) \qquad (6)$$

Where:
v_t: linear velocity at time *t*
w_t: angular velocity at time *t*
k_v, k_w: proportional control gains
x_t, y_t: the coordinates of the robot
x_n, y_n: the coordinates of the target node
$\Delta\theta$: the difference between the robot's heading and the straight-line path to the target node
$\varepsilon_v, \varepsilon_w$: Noise component sampled from a Gaussian distribution

$$\begin{bmatrix} \dot{v}_t \\ \dot{w}_t \end{bmatrix} = \begin{bmatrix} \frac{v_t - v_{t-1}}{dt} \\ \frac{w_t - w_{t-1}}{dt} \end{bmatrix} \qquad (7)$$

$$\begin{bmatrix} v_t \\ w_t \end{bmatrix} = \begin{bmatrix} \dot{v}_t * dt + v_{t-1} \\ \dot{w}_t * dt + w_{t-1} \end{bmatrix} \qquad (8)$$

Where:
$\dot{v}_t$: linear acceleration at time *t*
$\dot{w}_t$: angular acceleration at time *t*
dt: time step

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_t \cos\theta_t * dt \\ v_t \sin\theta_t * dt \\ w_t * dt \end{bmatrix} \qquad (9)$$

Equation 6 models a proportional controller, meaning that as the robot gets closer to the node that smaller the velocity will be. The gain values are set as a design decision and were selected based on experimentation in the simulation. In order to ensure the simulated robot always makes the shortest turn, the difference in the heading, $\Delta\theta$, should be mapped between $-\pi$ and π before calculating the angular velocity. Also, to make the model more realistic, noise was added to each calculation to model variations due to the components of the system.

In order to take into account acceleration limits, Equation 7 was used to determine the instantaneous acceleration given the change in velocity at time *t*. If the calculated acceleration exceeded the limits set for the simulation, then the velocity command for the respective component was recalculated based on Equation 8. For this simulation, the acceleration limits were ($0.288 m/s^2$, $5.579 rad/s^2$) and the time step was set to 0.1sec. Once the velocity command was determined to be valid, Equation 9 was used to calculate the next position and heading of the robot.

The experimentation used to tune this control scheme is discussed in the following sections.

Question 9: Simulate Paths from Part 7

Using the motion model described above and the paths generated by the "online" A* search in Question 7, a simulation of a robot driving along the path was performed. The initial linear and angular velocity was assumed to be zero and the initial heading was always set to $-\pi/2$. In order to also display the heading, the position of the robot was plotted using an arrow where the location of the robot is located at the center of the base of the arrow.

Due to the amount of calculations, the arrows have blurred together, but all changes in heading are observable along with some small deviations due to noise and overshoot due to acceleration limits. Figure 5 contains the results of the simulating the paths from Questions 7.

The biggest challenge with implementing the simulation was tuning all the gain parameters to be consistent across all tests. Since nonzero linear and angular velocity commands were being for sent simultaneously, the proportional gain values for each velocity equation had to be set such that the robot could turn fast enough before driving too far forward. In early tests, the robot would translate too far forward, missing the first target, and cause the simulation to become unstable. The gain values were tuned such that the angular velocity was allowed to accelerate much faster than the linear velocity.

The other challenge was setting a distance threshold such that the simulation could advance to the next target at the proper time. Because the velocity is based on proportional control, as the robot gets closer to the target the velocity calculation approaches zero. If the simulation waited for the distance to drop to zero, the robot would slow down such that the velocity commands would be unrealistically small and the noise in the system would be the driving force of the velocity instead of the actual calculation. In order to overcome this, it is common to set a distance threshold such that once the robot enters a certain radius, the target would be updated to the next node in the path. However, at locations where the simulation had to complete a sharp turn, the robot did not complete the turn quick enough before the target would change. This sudden change would cause errors in the heading calculation and lead to instability. Part of this behavior was corrected by adjusting the gains, but also balancing the distance threshold assisted in solving this behavior as well.

The most inconsistent behavior that occurred was with the angular velocity control and the heading calculation. During some of the simulations, the robot would be following a horizontal line in the negative x direction and seemingly at random would become unstable in its heading calculation. This would propagate to the velocity controls and the whole simulation would spiral out of control. The cause of this behavior was determined to be in the mapping logic for the $\Delta\theta$ value. A heading of along the negative x direction is either π or $-\pi$ depending on how the robot last turned. The robot was observed to slightly oscillate back and forth across the path of motion which on the horizontal is continually crossing over the mapping threshold. It is believed that these rapid changes caused a numerical error in the motion model and it was subsequently unable to recover. The solution to this behavior was to increase the limits on the mapping threshold to slightly larger than $\pm\pi$ to accommodate the small oscillations and keep the heading calculation more stable. After this modification, the simulation remained stable on horizontal paths for all tests.

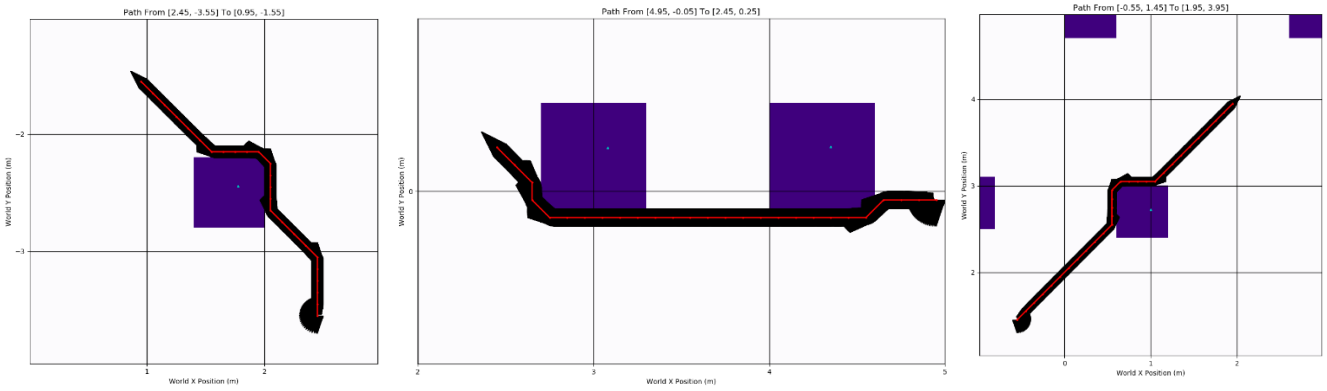

Figure 5. Question 9 simulation results for Question 7 test A (left), B (center), and C (right)

Question 10: Simulation with Real Time Planning

The purpose of using the "online" A* algorithm is to model what is physically possible when implementing real time planning on a physical robot. Using the same test scenarios listed in Table 2, the simulation was completed using real time planning. The results of this test are shown in Figure 6. The simulation was provided with a start location and would then determine the first target node using the "online" A* search algorithm. Once a node was identified, the robot would begin to move to that target. Since the robot can only observe its immediate neighbor nodes, the next target could not be determined until the previous target was reached. Each target found by the search algorithm is marked with a red dot.

The same code for the "online" A* search completed in Question 7 and the driving simulation completed in Question 9 was used. These functions were proved to be functional before completing this task so the behavior of these simulations was expected to match the previous results. Comparing the images in Figure 6 with Figure 5, the location of all selected nodes is identical and the robot follows a nearly identical trajectory.

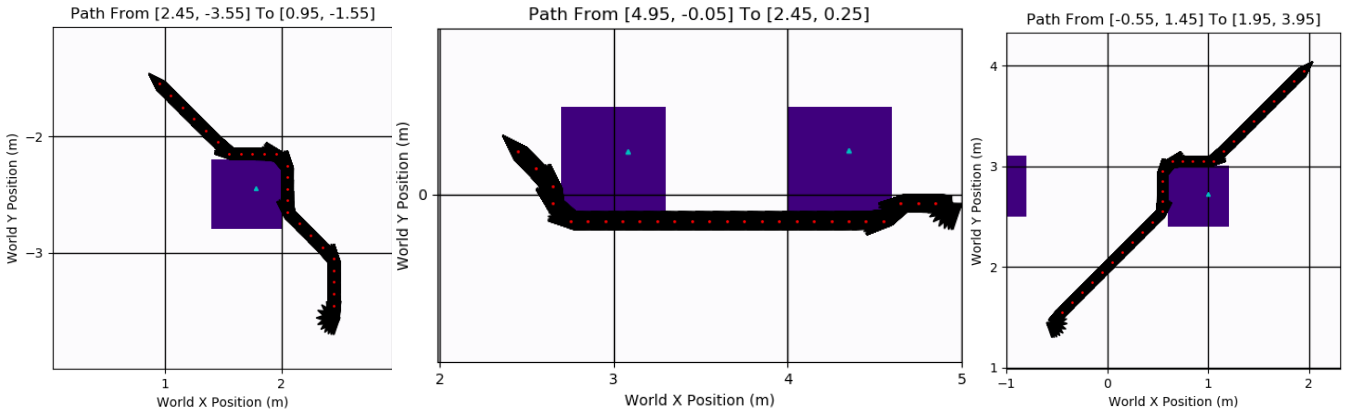

Figure 6. Question 10 simulation results

Question 11: More Simulation with Real Time Planning

The same real time planning simulation was applied to the test scenarios listed in Table 1 for both the coarse and fine grid layout. The results of the coarse grid tests are shown in Figure 7 and the fine grid test results are in Figure 8. All tests for both grid layouts were able to successfully complete, but some of the solutions varied due to the different occupied cell layout.

The main advantage of using a grid with a larger cell size is speed. Even for these tests in a relatively small area, the coarse grid simulation completed noticeable quicker than the fine grid. It was determined that the bulk of this difference in speed is due to the robot's sensing ability. On both grids, the robot is only able to use "online" A* for the surrounding 8 neighbor node, however the true distance the robot is observing on the coarse grid is 10x larger than the fine grid. This implies less computational time because the coarse grid requires less targets to reach the goal. Also, since the coarse grid requires less targets, this also results in a faster overall travel time to reach the goal. The simulation directs the robot toward the target node and since the motion model uses a proportional control, the robot comes to a near stop as it approaches each target. Since the coarse grid requires fewer nodes to reach the goal this inherently meant the robot had to slow down less which decreased the overall simulation time.

Although the coarse grid excels at quickly determining a solution, it fails at determining the best path with respect to distance efficiency. The most drastic example of this is shown by comparing the results of Test B. The coarse grid test, was forced to take a large turn around the two occupied cells, before being able to turn back towards the goal. In contrast, the finer grid has two advantages, the first being that it is able to represent the landmark size with

better accuracy and reveals that there is actually passable space between the two obstacles. The second is that it can make tighter turns around an obstacle reducing the overall travel distance.

The grid cell size is an important design decision and must be selected based on the priorities of the task at hand and what option best fits the environment. Also in practice the most efficient model might be to use a combination of both, where the area surrounding an obstacle has a smaller cell size and the open space has larger cells.

Also it was observed that the results for Test B in Figure 8 did not find the best overall path. By observation, the shortest path to the goal is to turn left (from the perspective of the robot) at the first obstacle and then proceed towards the goal as opposed to turning right as the simulation did. Digging into the sorting behavior and cost functions reveals the cause of this behavior. When the simulation reached the node directly in front of the obstacle and ran the search algorithm, the two results that would have been identified as the lowest evaluated cost would have been the nodes directly the right and left. Both nodes would have and equal true cost $g(n)$ since the cost is one to move into either node. Also the estimated cost $h(n)$ will be equal since the heuristic uses the maximum distance along a single axis and both are at the same y coordinate. These calculations result in two nodes with identical values for $f(n)$ and $h(n)$. Based on the node selection methodology, the typical tie breaker is to use the estimated cost, but in the event that a tie still remains the algorithm will select the node that was evaluated first. For this implementation the methodology of evaluating the neighbors always starts with the nodes with smaller x position (for this orientation, the nodes to the robot's right) and finishes with the nodes with a larger x position. Because the node to the robot's right was evaluated first, it was selected over the left node even though it leads to a more efficient path. This scenario shows the limitation only using the "online" A* search. As an improvement the algorithm could implement a local planning routine to conduct a local search to verify if the robot is in fact on the most optimal.

Question 12: Additional Considerations

While this simulation does demonstrate the power of the A* search algorithm, some assumptions were made to simplify the driving simulation that do not reflect how a real robot moves. Assuming that the robot knows its exact position with complete certainty is an unrealistic assumption unless the environment is using high quality motion tracking cameras (even then there is still some small variance). In real navigation, basing the estimated location solely on odometry controls will yield an inaccurate and unusable belief of the robots true state. This inaccuracy is due in large part to slipping as the robot move about the environment, creating an error that will slowly build over time. This phenomenon can be observed by plotting the odometry data against the ground truth data provided by UITAS. One option would be to equip the robot with some form of sensor and employ a filtering algorithm to establish a more accurate believed state.

Another assumption made by the search algorithm, is that the robot can only sense up to one node away. In practice this parameter should change based on the sensor suite equipped to the robot and the grid size being used for the search. For example, if the senor allowed the robot to detect any object in a 3m radius but a grid used a cell size of 0.1m, the algorithm could make a plan for the whole visible space and before making a movement. This change would create a more direct path through the open space and most likely assist with making the movement more fluid.

References

Halpern, Y., Lui, H., Barfoot, T., & Leung, K. (July 2011). The UTIAS Multi-Robot Cooperative Localization and Mapping Dataset. *International Journal of Robotics Research*, 30(8):969–974.

Patel, A. (2019). Retrieved from Red Blob Games: http://theory.stanford.edu/~amitp/GameProgramming/

Russel, S., & Norvig, P. (2010). *Artificial Intelligence, A Modern Approach.* Pearson.

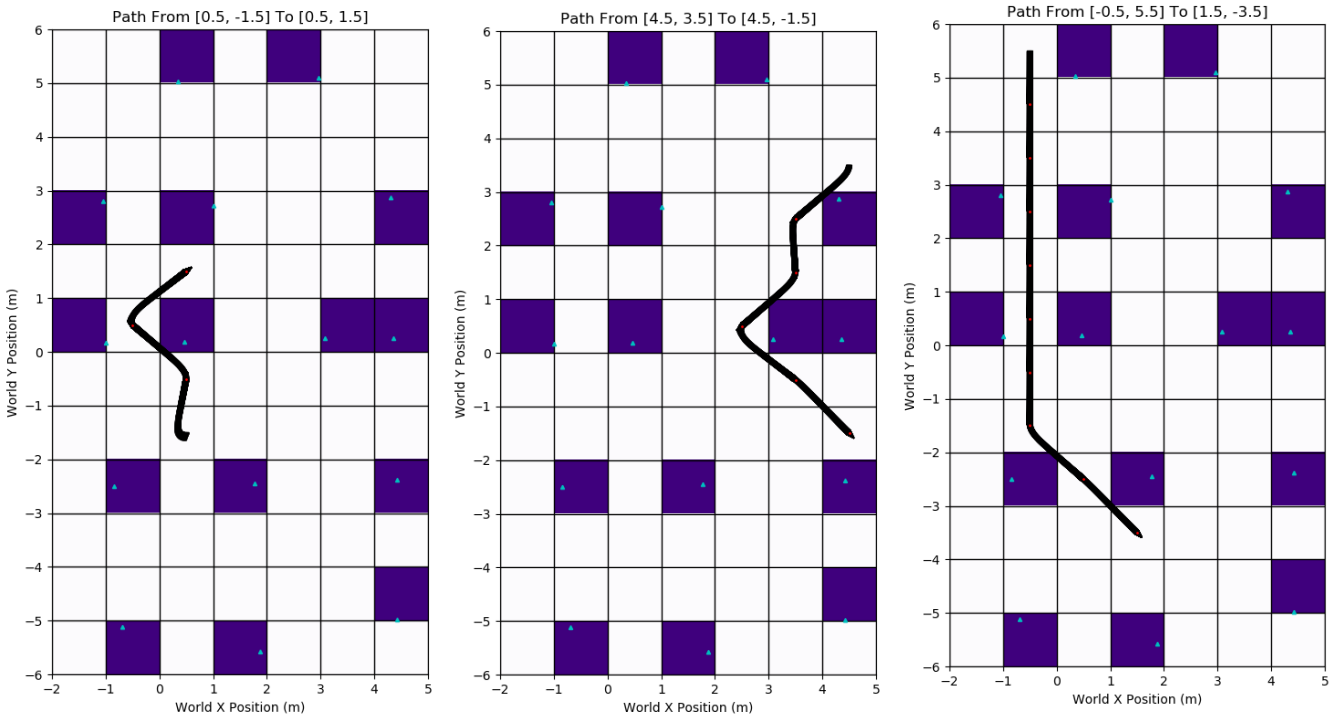

Figure 7. Question 11 simulation results for course grid

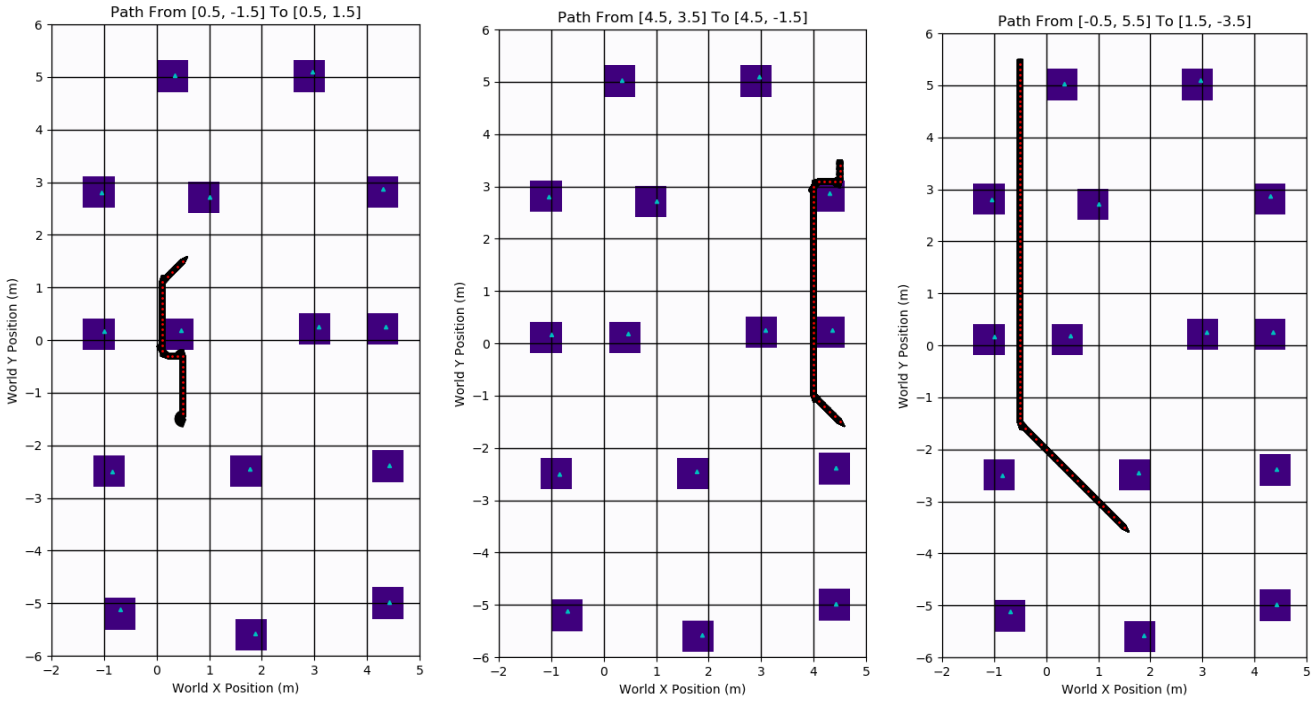

Figure 8. Question 11 simulation results for fine grid