MECH_ENG 469: ML & AI In Robotics

Report 1: Search and Navigation
A* Implementation

Michael Jenz

10-27-2025

Table of Contents

1 Part A

1.1 Introduction

This assignment focused on the implementation of the A* algorithm to generate 2-D paths that navigate between start and goal locations while avoiding obstacles. Additionally, we were asked to implement a simple controller that enables a wheeled mobile robot to drive the A* generated paths. A* is the most common type of informed search strategy. These strategies use functions known as heuristic functions that provide insight on the location of goals [1]. This robot is based on a differential wheeled robot, its configuration can be fully described by its position in 2-D space and its orientation $[x, y, \theta]$. This problem setup can be seen below in Figure 1. Furthermore, the robot is commanded to move through the linear and rotational velocities $[v, \omega]$ shown in Figure 1. Also shown in Figure 1 is the setup to find the distance and heading from the robot to a given point. This will be referenced later in Part B.

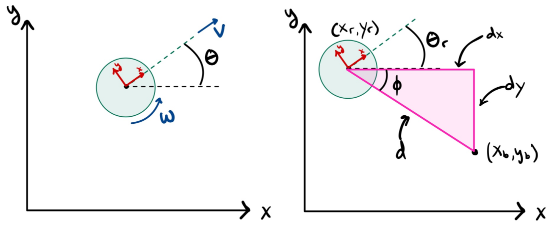

Figure 1: Configuration of mobile robot in 2-D space (left) and problem setup to find distance from robot to given point (right).

1.1.1 Occupancy Grid (Q1)

The A* algorithm operates within a discrete world finding paths between nodes n to the goal node. Since robots must operate in the real world, which is continuous, we need to discretize space in some way to use the A* algorithm. One popular method is to use an occupancy grid, first introduced in 1985 by Moravec and Elfes who used this method to represent the probability that an area was occupied as determined by sonar measurements and their algorithm [2]. In this case, I use the occupancy grid to represent the world discretely, as did Moravec and Elfes, but there is no distribution for the occupancy probability. Rather, I know exactly which cells are occupied and which are not. I used the landmark locations of ds1 in homework 0. The occupancy grid, as seen in Figure 2, shows the world cut into 1 meter squares. An entire square is occupied if a landmark is located inside. Thus, a continuous world represented by coordinates $[x, y]$ is also represented by cells with indexes denoted by $[i, j]$. In order to allow for free conversion between coordinates and cell indices, I chose to represent the cell by its bottom left corner or minimum allowable values.

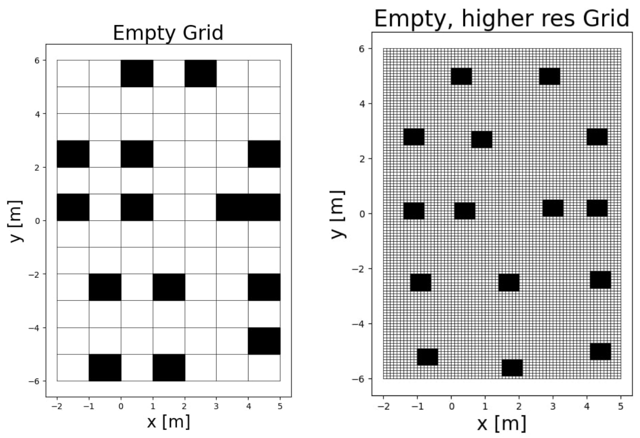

Figure 2: Visualization for occupancy grid with scale of 1 meter (left) and 0.1 meters (right).

1.2 Offline A* Algorithm

1.2.1 Implementation (Q2)

The A* algorithm operates starting from the start node and moves towards the goal node. It estimates the cost of all nodes in the open set (initialized as nodes neighboring the start node) and chooses the lowest cost node. Then, it adds all neighboring cells of the chosen cell to the open set and then repeats this process until it reaches the goal node. Finally, following the expansion of the nodes in reverse from end to start, the algorithm provides the optimal path to the goal.

This summary is reinforced by Equation 1- 3 which define the evaluation function $f(n)$ in terms of the true cost $g(n)$ (provided by the instructor) and the heuristic $h(n)$. The evaluation function is the function that estimates the cost of the cells in the open set. The A* algorithm is cost-optimal due to the characteristics of the heuristic, namely its admissibility. A heuristic is admissible if it never overestimates the cost to reach a goal. This makes A* cost-optimal, since it will always find the lowest cost path [1].

My heuristic is the euclidean distance between the center coordinates of a cell and the center coordinates of the goal cell. This heuristic is admissible because the euclidean distance between the center of two cells will never be greater than 1. That is, as defined by $g(n)$, the true cost of the transition between two cells that are not occupied. This proof is shown in Figure 3. This comes with one major caveat and modification: if the scale of the grid is larger than 1 meter then this heuristic is no longer admissible since the euclidean distance between two cells will be greater than 1. This condition for admissibility is given below in Equation 4. The solution would be to scale the heuristic by a factor of $\frac{1}{s}$ where s is the scale of the grid. Since this case is not considered in this homework, I did not implement such a heuristic.

This A* implementation is offline because it is able to trace back from the goal to the start point, and eliminate any extra nodes visited. This type of pruning would not be possible in an online scenario and is the main difference between this and the online implementation.

$$f(n) = g(n) + f(n) \tag{1}$$

$$g(n) = \begin{cases} 1, & \text{if neighbor cell } n \text{ is unoccupied} \\ 1000, & \text{if neighbor cell } n \text{ is occupied} \end{cases} \tag{2}$$

$$h(n) = \sqrt{(x_{goal}^{center} - x_n^{center})^2 + (y_{goal}^{center} - y_n^{center})^2} \tag{3}$$

$$h(n_1) = d = s \leq g(n_1) \iff s \leq 1 \tag{4}$$

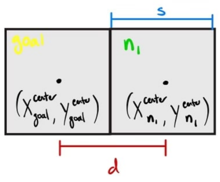
s
goal
n_1
$(X_{goal}^{center}, Y_{goal}^{center})$
$(X_{n_1}^{center}, Y_{n_1}^{center})$
d

Figure 3: Problem set up for admissibility proof of my heuristic.

1.2.2 Testing (Q3)

To test the offline A* implementation, I passed in the following sets of points in Table 1, the results of which can be seen in Figure 4.

Set ID	Start $[x, y]$	End $[x, y]$
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]

Table 1: Point sets A, B, and C passed into A* algorithm.

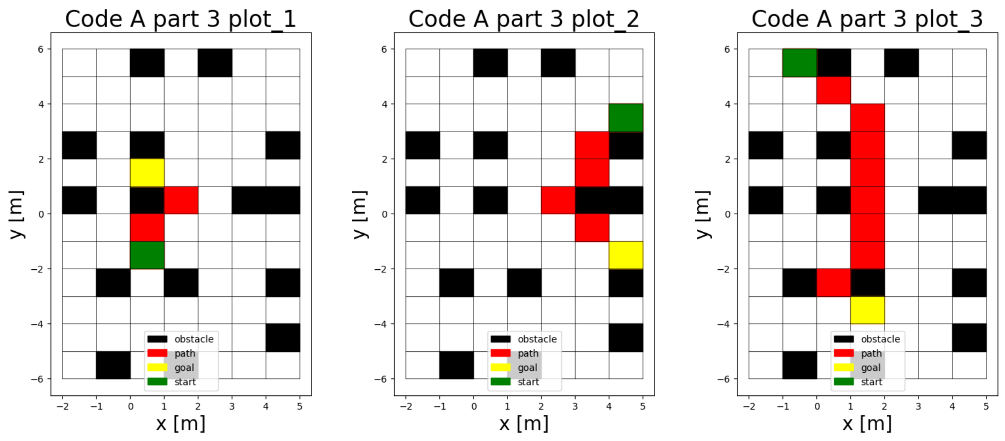

Figure 4: Results from testing Offline A* algorithm from specified start to goal points. From right to left, results for A, B, and then C.

1.3 Online A* Algorithm

1.3.1 Implementation (Q4)

As mentioned previously, the major difference between Online and Offline A* is that the algorithm is no longer able to prune the path for unnecessary node visitation. This is because in the Online case, the robot is physically moving to each node as the A* algorithm plans a path. This is more realistic because often a robot may not be able to observe if a cell is occupied until the cell is the robots physical neighbor. This is another way of saying that the robot does not have *a priori* knowledge of the obstacles. So, the only modification I made to my A* algorithm from step 2 was to remove the pruning step that occurs at the end of the algorithm run.

1.3.2 Testing (Q5)

The results for the Online implementation of A* are in Figure 5. The only key difference between the results of the Online and Offline implementation is seen in the results for point set B where the Online algorithm takes an extra step. This is consistent with our expectations for the Online implementation.

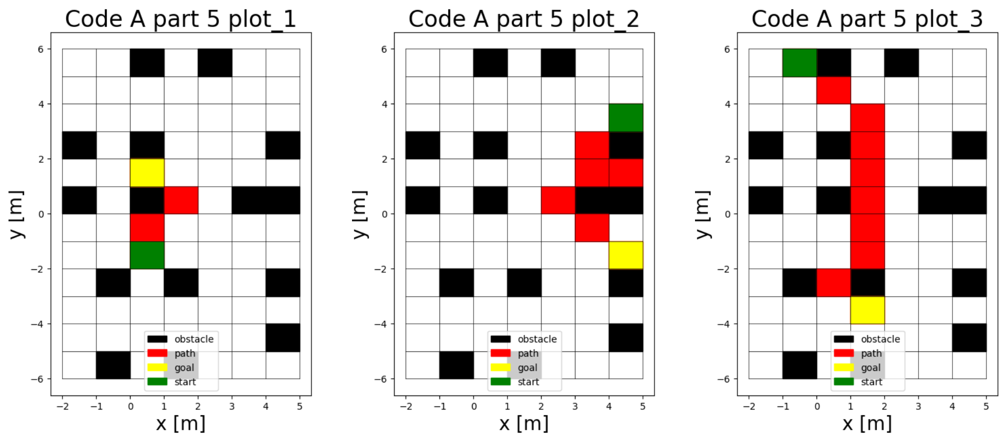

Figure 5: Results from testing Online A* algorithm from specified start to goal points. From right to left, results for A, B, and then C.

1.4 Higher Resolution Grid

1.4.1 Implementation (Q6)

The higher resolution grid, with cell side length of 0.1 meters, provides a more realistic representation of the world. Since the landmarks likely take up space larger than their 0.1 by 0.1 meter grid cell, I inflated their size to be 0.6 by 0.6 meter squares. Otherwise, my implementation for the grid was exactly analagous to the 1 meter scaling.

1.4.2 Testing (Q7)

To test the Online A* implementation on the higher resolution grid, I passed in the following sets of points in Table 2, the results of which can be seen in Figure 6.

Set ID	Start $[x, y]$	End $[x, y]$
D	[2.45, -3.55]	[0.95. -1.55]
E	[4.95, -0.05]	[2.45, 0.25]
F	[-0.55, 1.45]	[1.95. 3.95]

Table 2: Point sets D, E, and F passed into A* algorithm.

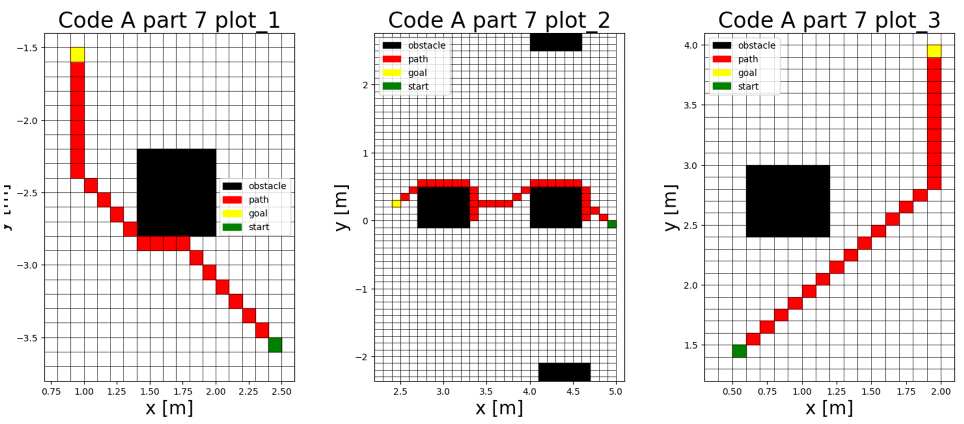

Figure 6: Results from testing Online A* algorithm on the higher resolution grid from specified start to goal points. From right to left, the results for point sets D, E, and F.

2 Part B

2.1 Offline Controller

2.1.1 Implementation (Q8)

The offline controller emulates a differential wheeled robot driving on the Online A* generated path. Thus, the A* algorithm acts as a high level controller providing a list of way-points to follow, while the Offline Controller acts as a low level controller that generates velocity commands to move the robot along this path. So, the high level controller operates by commanding the robot to a specified way-point along the A* generated path until it reaches the point within a tolerance (calculated in Equation 5), and then commands the robot to the next point and so on. I made two specific modifications to this high level controller setup. First, the high level controller operates with a high tolerance of 0.25 times the scale size until the last (goal) point at which point it switches to a tight tolerance of 0.01 meters. This modification helps to keep the robot moving smoothly between cells and avoids getting stuck searching for a specific point in space while still achieving the goal coordinates with high precision. Next, the high level controller augments the A* commanded cell coordinates if the cell is neighboring an occupied cell. This is because if the robot is traveling near an obstacle, I wanted to make sure it gave a wide berth to the obstacle. Thus, the high level controller will search for obstacles neighboring the next point and shift the goal position away from the obstacle by 0.75 times the scale of the grid.

For the low level controller operation, the robot operates entirely on the heading error, seen in Equation 6, which is the difference in radians between the current robot heading and the direction of the goal point. I designed this controller uses a state machine for the robot. The robot has a state for its rotational and linear motion. The rotational state is determined by the sign of the heading error, positive corresponding to LEFT and negative corresponding to RIGHT. The rotational state of the robot is never stopped, it is always turning towards the goal point. The robot will turn towards the goal point until it is within 0.1 radians of tolerance before moving forward towards the goal. Up until this point, the robot linear state has been STOPPED. Once it is pointed towards the goal point, its linear state changes to FORWARD. This state machine for linear and rotational motion is essentially bang-bang control; however, it is highly effective in this setting.

Finally, the controller must convert the robot velocity states to actual velocities. The robot acceleration is limited to $\dot{v} = 0.288 \frac{m}{s^2}$ and $\dot{\omega} = 5.579 \frac{rad}{s^2}$. Since I am using bang-bang control, the controller applies the maximum acceleration depending on the state. If it is FORWARD or LEFT the controller applies maximum positive acceleration, and in the reciprocal case it applies maximum negative acceleration. I do not want the robot to move backwards in the STOPPED state. Therefore, I instituted speed clamping on the controller. The angular velocity ω is limited between $[1.0, -1.0] \frac{rad}{s}$ and the linear velocity v is limited between $[0.0, 10.0] \frac{m}{s}$. This completes the low level controller. Also, to make this more realistic I added noise to the system. This represents how issues such as slip may result in outcomes that are different than those expected by the controller. I applied this noise during the state transition step as a randomly selected number between $[-0.01, 0.01]$ for each of the components of the robot configuration $[x, y, \theta]$. This turned the robot driving paths from straight lines to squiggly paths as the controller compensated for the unpredictable outcomes of its commands.

$$\text{distance } = \sqrt{(x_{goal} - x_{current})^2 + (y_{goal} - y_{current})^2} \tag{5}$$

$$\text{heading error} = arctan(\frac{y_{goal} - y_{current}}{x_{goal} - x_{current}}) - \theta_r \tag{6}$$

2.1.2 Testing (Q9)

I tested the Offline controller on the paths generated by the Online A* implementation. The results are in Figure 7. Note how the controller gives a wide berth to the obstacles while still following the A* generated path. On trials where there are no neighboring obstacles, the path is not altered.

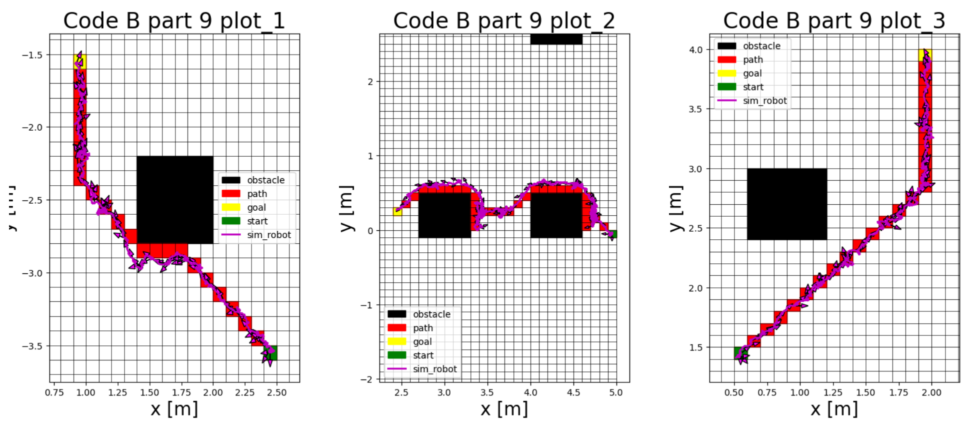

Figure 7: Results from testing Offline Controller on Online A* algorithm from specified start to goal points. From right to left, the results for point sets D, E, and F.

2.2 Online Controller

2.2.1 Implementation and Testing (Q10)

Putting it all together, my Online Controller can drive the paths as they are explored by the Online A* algorithm. This required a large change to my high level controller. Previously, I adjusted the path to have a wide berth around obstacles. This required seeing into the 'future' and knowing if the goal cell had a neighboring occupied cell. This time I used the information of the cells neighboring the currently occupied cell. The high level controller only modified the path if the next goal cell or way-point was located diagonally from the current cell. This is because moving between adjacent cells directly does not present a risk of hitting obstacles. But often when moving between diagonal cells directly with a neighboring obstacle, the robot will clip the edge of the obstacle. Thus, in the case that there is a diagonal command and a neighboring obstacle, my high level controller adds an additional goal cell that is adjacent to the current cell. In essence, this connects the A* path with entirely adjacent cells. This method avoids collisions with obstacles while also using only information available to the robot. See the results in Figure 8

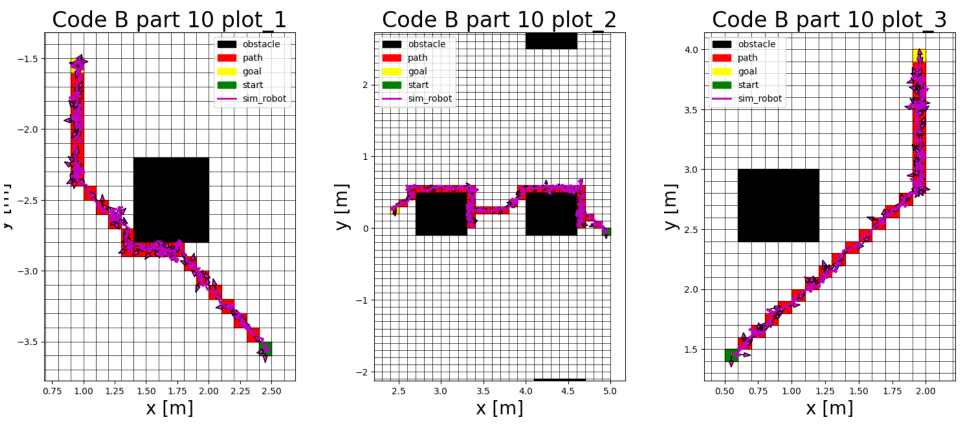

Figure 8: Results from testing Online Controller on Online A* algorithm from specified start to goal points. From right to left, the results for point sets D, E, and F. Notice the augmented target cells.

2.2.2 Performance Comparison (Q11)

To compare the performance of the controller in the two grid resolutions, I ran the Online A* and the controller on the set of points A, B, and C. Results in Figure 9 appear as if the robot performs better in the lower resolution grid, since the paths are mostly straight and there is a wide berth given to the obstacles. I would actually agree with this assessment because if we consider the robot paths in the finer grid, the robot path is extremely close to the obstacles. While we have inflated the size of them to account for this, the small grid size forces the robot to be close to the obstacle. For this reason, I would argue that there is an advantage to using the larger grid size in the real world. Beyond this effect, however, I do not see a discernible difference in the performance of the controller.

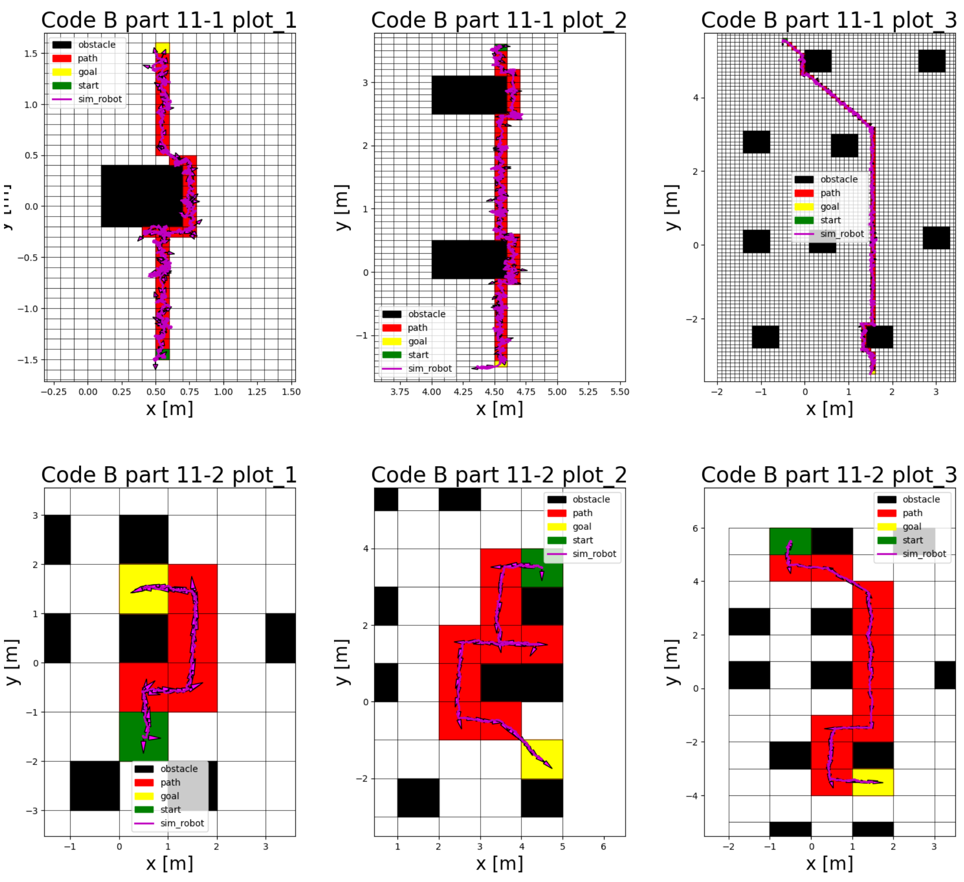

Figure 9: Results from testing Online Controller on Online A* algorithm from specified start to goal points. From right to left, the results for point sets A, B, and C. Same for top and bottom rows, using different grid resolutions.

2.2.3 Real World Application (Q12)

As I mentioned in the section above, the transition from this idealized grid world to the real world requires extra consideration. The issue of the grid size may be addressed by adjusting the grid size to appropriately match that of the robot and obstacles. Beyond this, there are several assumptions that we have made in our analysis that will haunt us in the real world. Primarily, we have disregard any part of sensing in this assignment. We assume that somehow the robot detects that its neighbors are occupied, or that cells across the map are occupied. While this makes the problem much easier, it skips a vital step in bringing a simulation to reality. We would likely need Lidar or sonar sensor that can detect objects. Even the best sensors still have some uncertainty, which would require further processing or stochastic methods to account for. Furthermore, we have assumed that the robot has a precise localization system. Although we did introduce noise, we assumed that the robot would be able to figure out where it was despite that. In reality, localization is a huge challenge that would require more sensing and perhaps using filtering methods such as Kalman Filters to localize the position of the robot as it drove the paths.

2.2.4 Working in Continuous Space

The A* algorithm requires the use of a occupancy grid representation of the world. I find this to be annoying, so I implemented a version of potential fields to help control the robot in continuous space. Instead of assigning specific cells as occupied, potential fields methods use a global equation to describe the arena. The equation is representative of a force that is pushing the robot towards the goal state (and away from obstacles). So, I represented the goal state as a 3-D Gaussian distribution with positive amplitude, and obstacles as those with negative amplitudes. This formulation created a potential field as seen in Figure 10. Then, using gradient ascent techniques (as high level control) and the low level controller I developed above, the robot was able to find the goal position on 4 out of 6 of the point sets. In the 2 that it failed, the robot got stuck on local maximums or was not able to reach the goal within tolerance because the goal was close to an obstacle. Implementing potential fields shows how continuous methods are another great alternative when mapping for mobile robotics.

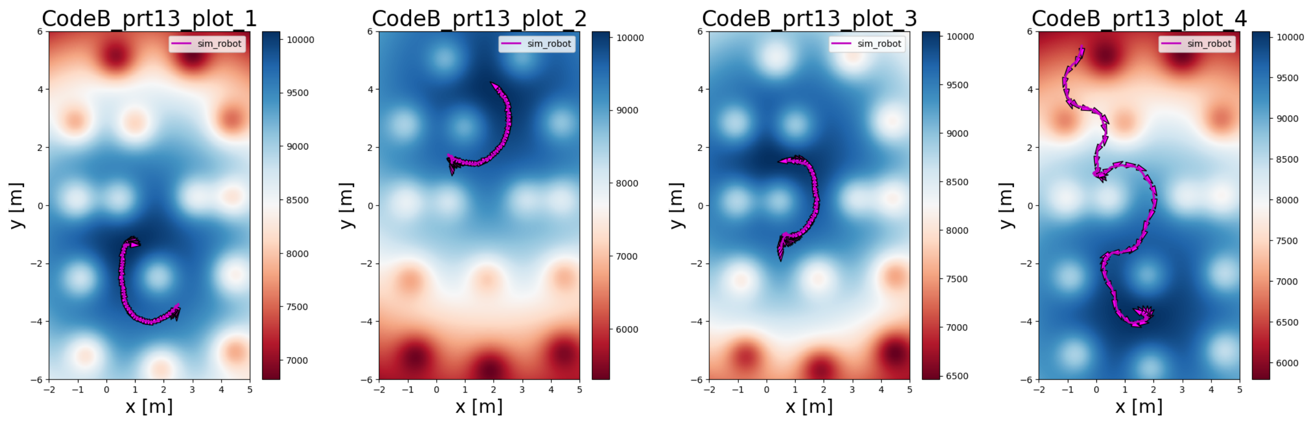

Figure 10: Results from testing potential fields from specified start to end points. From right to left, the results for point sets A, C, D, and E.

References

[1] S. Russel and P. Norvig, *Artificial intelligence: A Modern approach*, 4th ed. Prentice Hall, 2021.

[2] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, 1985. DOI: 10.1109/robot.1985.1087316.