

A\* Write Up

2. The A\* algorithm assigns a total cost to each node.

T = h + g

T: total cost of node  
h: heuristic cost – estimated cost from current node to goal node  
g: cost to reach the current node from the initial starting goal

The algorithm relies on assessing the total costs of each node to determine the most optimal next step in the path. When choosing a heuristic for the model, it must be admissible which means the heuristic cost must always be less than or equal to the actual cost of getting to the goal from the current node. Common heuristics are the euclidean distance or the manhattan distance.

Euclidean Distance:

$$h = \sqrt{(x_{node} - x_{goal})^2 + (y_{node} - y_{goal})^2}$$

Manhattan Distance:

$$h = |(x_{node} - x_{goal})| + |(y_{node} - y_{goal})|$$

In this problem, the cost of traversing a node is either 1 if the space is open in all 8 directions, or 1000 if the space contains an obstacle. Because the cost of moving to a node diagonally is only 1, the 2 heuristics mentioned above are not admissible.

Euclidean Distance =  $\sqrt{2} > 1$   
Manhattan Distance =  $2 > 1$

Therefore, one option for a heuristic is to just use the Euclidean or Manhattan Distances divided by an integer.

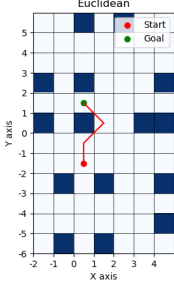
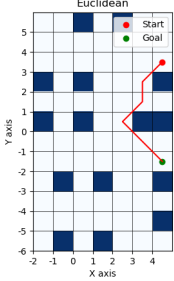
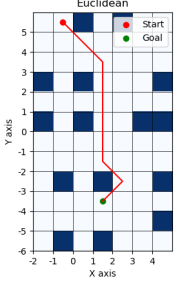
Euclidean :  $\sqrt{(x_{node} - x_{goal})^2 + (y_{node} - y_{goal})^2} / 2 = \sqrt{2} / 2 < 1$   
Manhattan :  $|(x_{node} - x_{goal})| + |(y_{node} - y_{goal})| / 3 = 2 / 3 < 1$

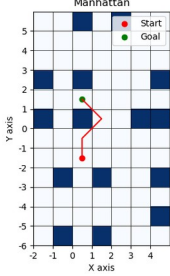
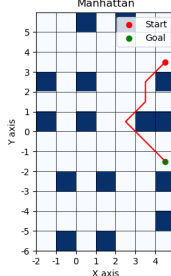
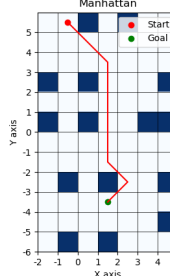
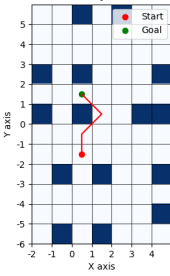
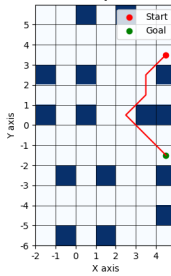
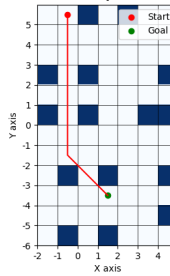
Both of these heuristics are now admissible. The Chebyshev Distance is also another commonly used heuristic that is admissible.

Chebyshev Distance :

$$h = \max(|x_{node} - x_{goal}|, |y_{node} - y_{goal}|) = 1 @ \text{ a diagonal}$$

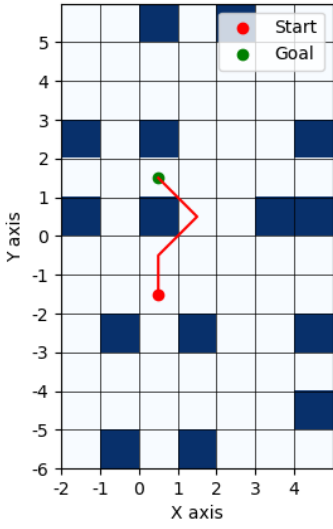
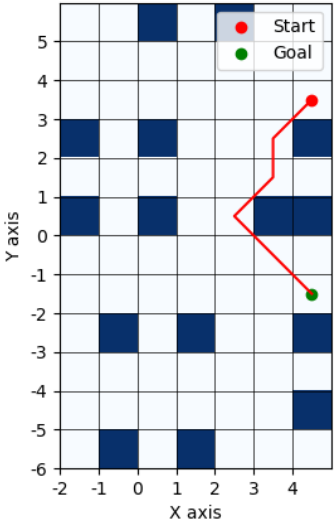
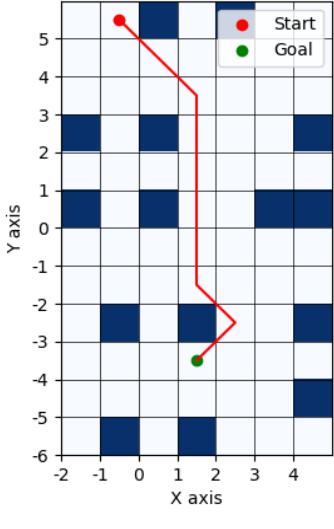
To determine the best heuristic for this problem, I compared the three heuristics above.

	Case A	Case B	Case C
Euclidean			

Manhattan			
Chebyshev			

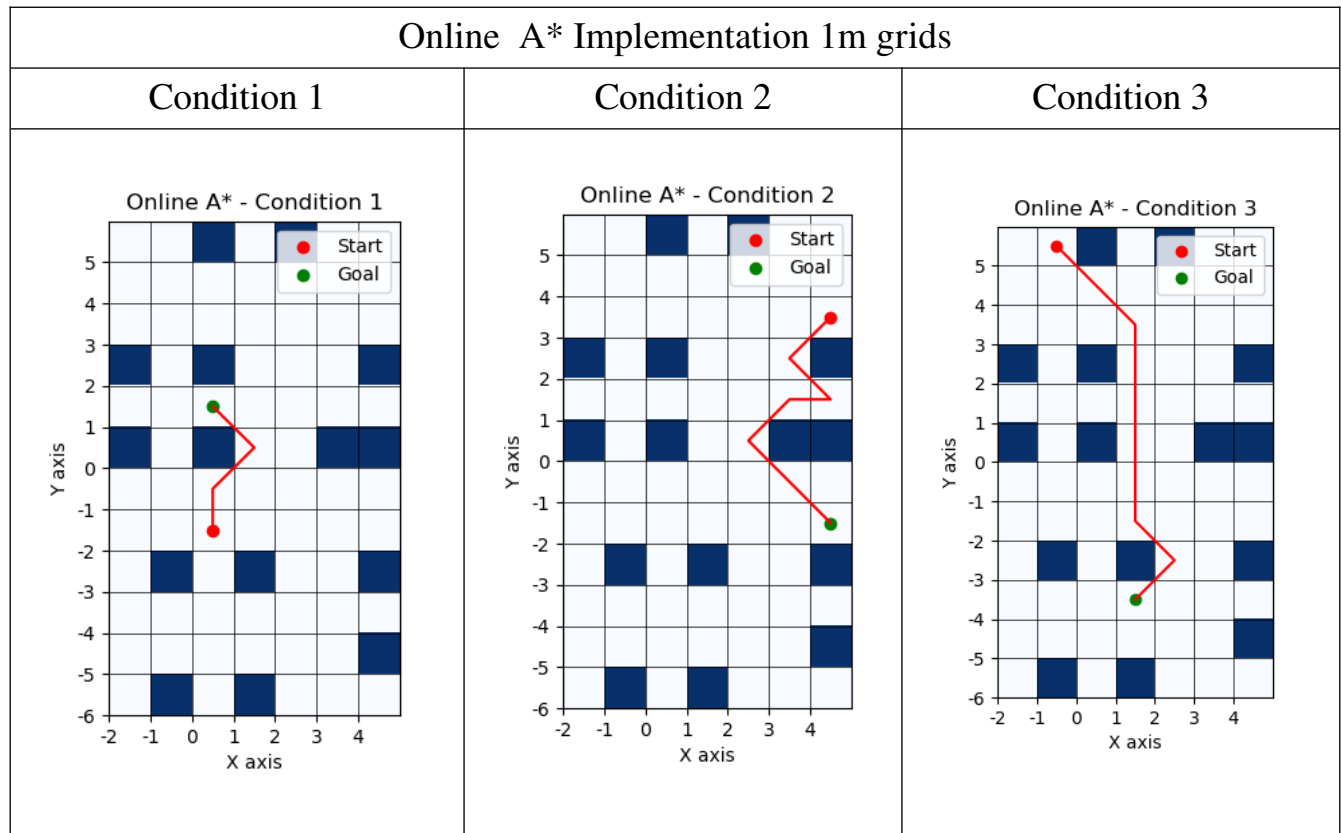
The results of the comparison showed that the heuristics perform the same in every case except the 3<sup>rd</sup> where the Chebyshev heuristic produces a different path than the Euclidean and Manhattan heuristics. Because there is no stark benefit between the three heuristics, I continued this homework with the Euclidean Distance heuristic.

### 3. Offline A\* implementation with a course grid.

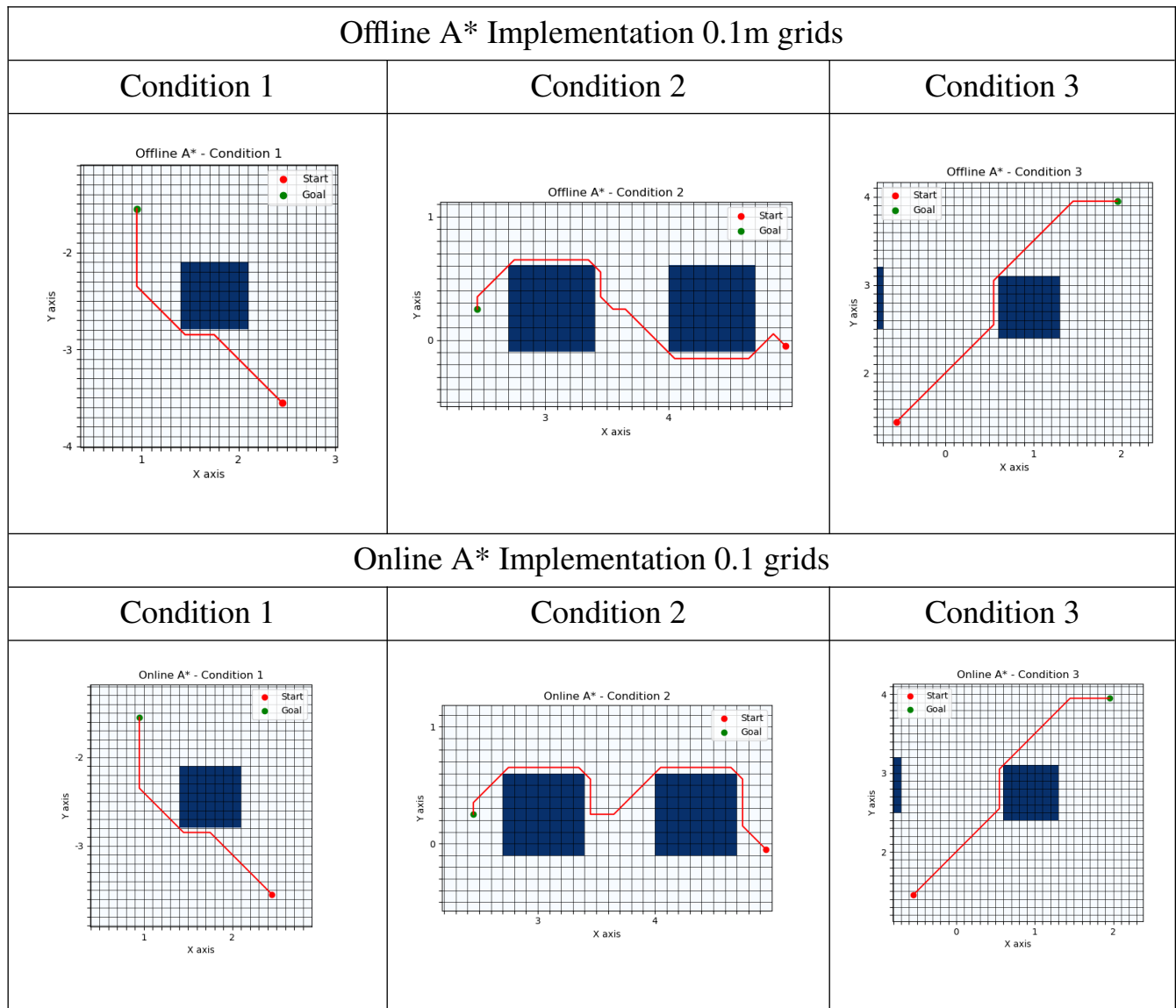
Offline A* Implementation 1m grids		
Condition 1	Condition 2	Condition 3
<p>Offline A* - Condition 1</p> 	<p>Offline A* - Condition 2</p> 	<p>Offline A* - Condition 3</p> 

4. When moving from an offline A\* algorithm to an online one, the algorithm can no longer search all previous neighbor nodes in the open set. It can only look at the cost of the neighbors of the current node to make a decision for the optimal next step. This means the set of expanded nodes is not cumulative but is reset at each node. The maximum number of nodes in the openset is now limited to 8 possible neighbors of the current node. The major edit in code from the offline to online version lied in the resetting of the openset after the next optimal step was chosen. Additionally, the final path was formed by each node that was visited during the online version whereas the path in the offline version was determined recursively at the conclusion of the algorithm.

#### 5. Online A\* implementation with a course grid



#### 7. Offline and Online A\* implementation with a less course grid



8. The controller I designed was a PI controller.

Linear Velocity:

$$v = K_p_v * (\text{distance\_error}) + K_i_v * (\text{cumulative\_linear\_error})$$

$$\text{where } \text{distanceError} = \sqrt{(x_{curr} - x_{goal})^2 + (y_{curr} - y_{goal})^2}$$

$$\text{cumulativeLinearError} = \sum_i \text{distanceError}_i$$

and  $K_p_v$  and  $K_i_v$  are constants

Angular Velocity:

$$w = K_p_w * (\text{angular\_error}) + K_i_w * (\text{cumulative\_angular\_error})$$

$$\text{where } \text{angularError} = \arctan((y_{goal} - y_{curr}) / (x_{goal} - x_{curr})) - \Theta_{curr}$$

$$\text{cumulativeAngularDistance} = \sum_i \text{angularError}_i$$

and  $K_p_w$  and  $K_i_w$  are constants

The Proportional and Integral control constants were tuned over the course of development to fit the system the best. During tuning, I discovered that a very small  $K_i$  was important to counteract integral windup without

needing to implement a windup correction while simultaneously containing the effect of noise on the system. The Proportional control aided in balancing the ratio of linear and angular velocity growth.

Finally, the model checks that the calculated  $v$  and  $w$  do not exceed the motor's acceleration limits. Once the final  $v$  and  $w$  are determined, they are fed to the motion model to calculate the new current  $x$ ,  $y$ , and heading of the robot on the path. The motion model is as follows :

$$x_{new} = x_{curr} + v * \cos(\Theta_{curr} + w * dt) * dt + \epsilon$$

$$y_{new} = y_{curr} + v * \sin(\Theta_{curr} + w * dt) * dt + \epsilon$$

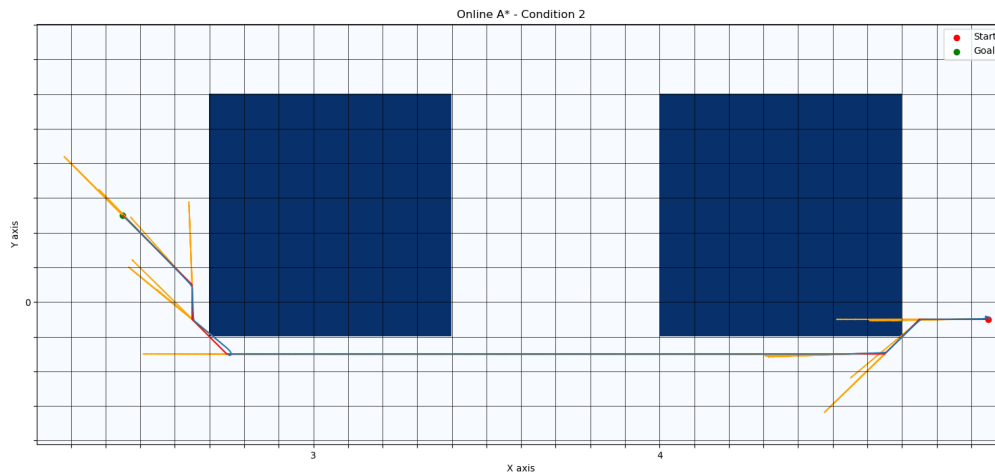
$$\Theta_{new} = \Theta_{curr} + w * dt + \epsilon$$

Where  $\epsilon$  is noise

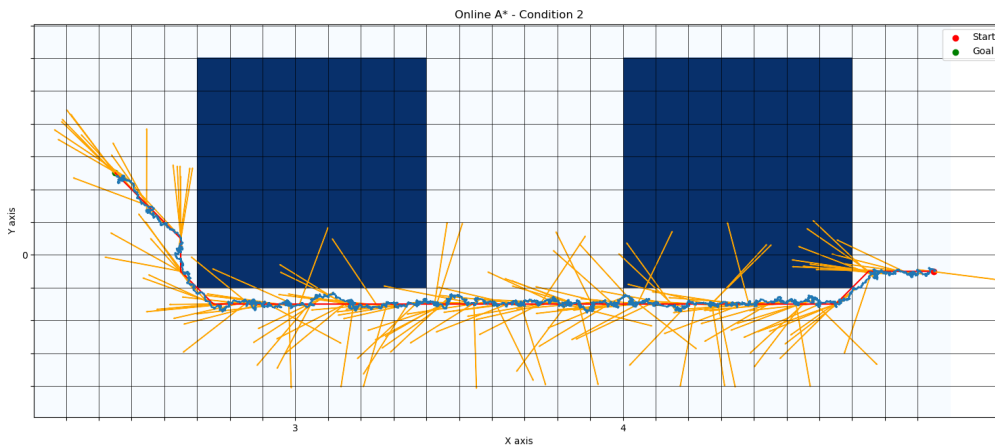
9. Originally, I was only using a proportional controller when moving the robot before adding noise to the system. However, after the addition of noise to the motion model, I had to include an integral term to counteract the noise. I was surprised by the robot's ability to move close to the generated path, but the robot is also moving very slowly. Because the increments of velocity and positions are very small, it can maintain a small error, but if I adjust it to move faster, the model gets significantly worse.

## Planning the entire path, then moving the robot

Proportional, no noise



Proportional, with noise



# Proportional and Integral, with noise

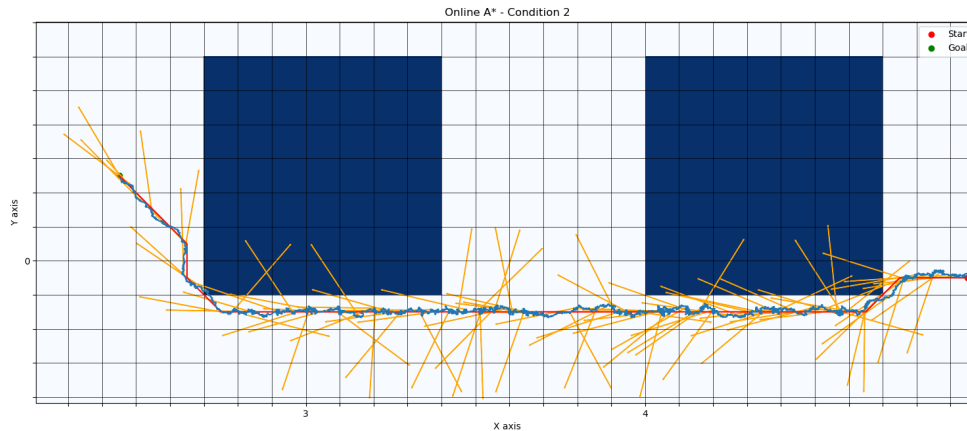
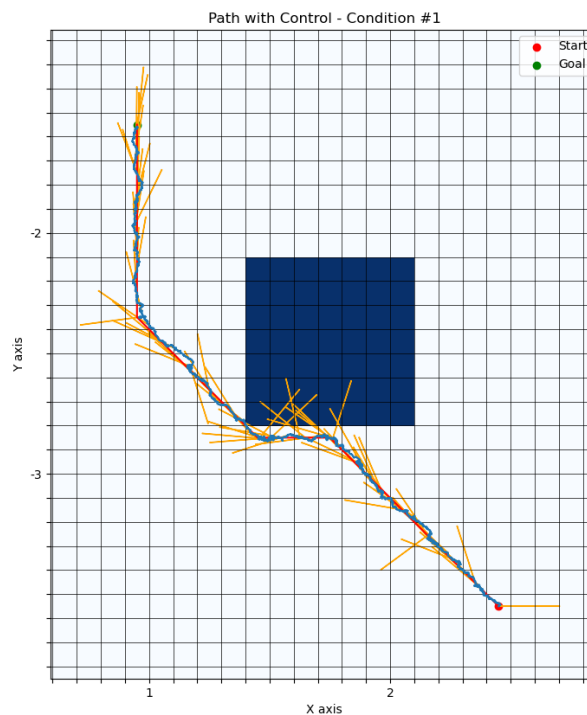


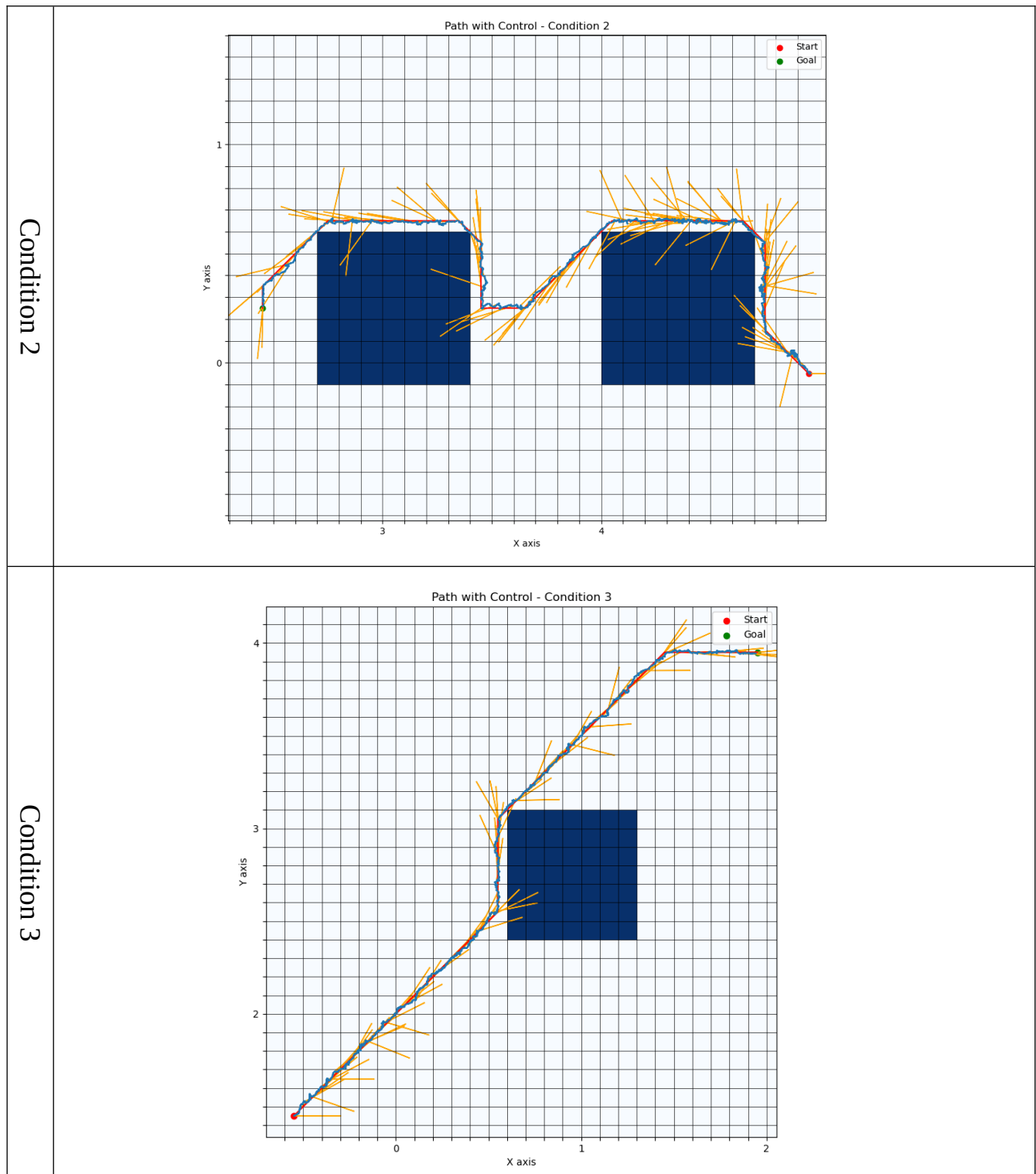
Table 1: Without noise, the Proportional control performs well by itself. When noise is introduced, there is an increase in angle variety and the signal is significantly noisier. With the addition of Integral control, there are fewer angles and the overall magnitude of signal noise is less.

10. When merging the planning and controlling together, I expected there to be more miscommunication between the robot and the algorithm, however the result is almost identical to question 9. I believe this is due to my controllers need to execute until the desired way point or goal node has been reached within a set tolerance before it will allow A\* to resume path planning. I think it would be interesting to explore how changes in that tolerance would result in the robot potentially arriving in a cell that was not the intended next step. I believe I coded a robust enough controller, so if the robot does get off track, the A\* algorithm will continue to function properly with the new robot position/new current node. Below are the plots generated for this question.

## Planning the path and moving the robot simultaneously

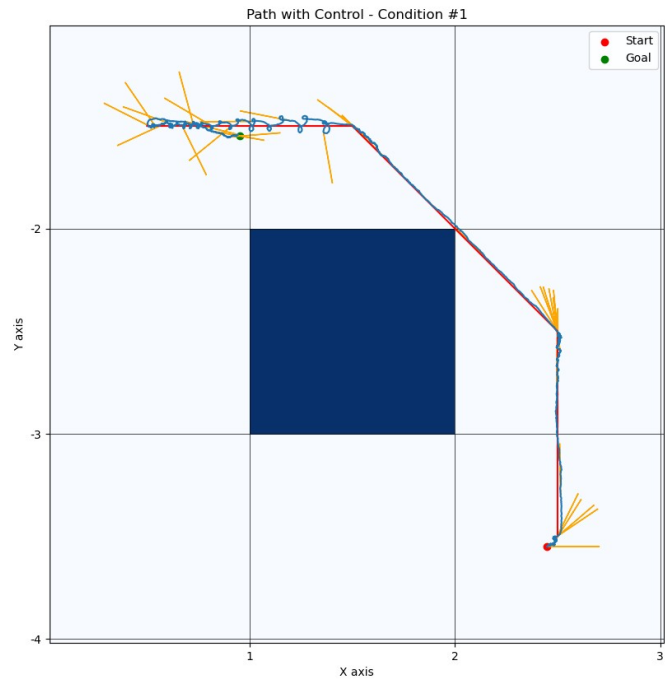
### Condition 1



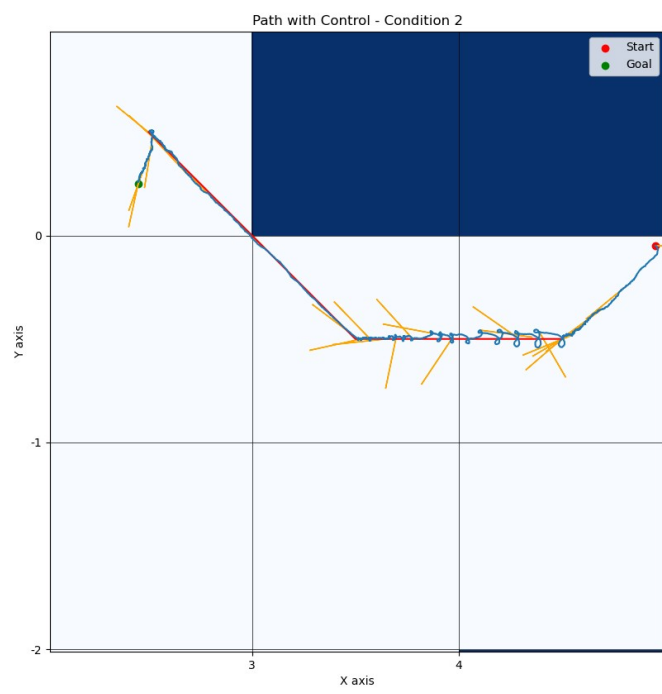


11. Changing the the grid size while keeping the start and goal positions the same, lead to some challenges because the locations of those points do not fall in the middle of the large grid size, but rather on the grid line or non-centric in the cell. To ensure the path completed at the actual point instead of just the center of the goal cell, I had to add a final movement step once it reached the final grid position to then go to the goal. While this approach works, you can see in the figures below that the final path is choppy and some paths even pass the goal position to get to the center of the grid before it begins moving towards the actual goal (See conditions 1 and 2).

Condition 1

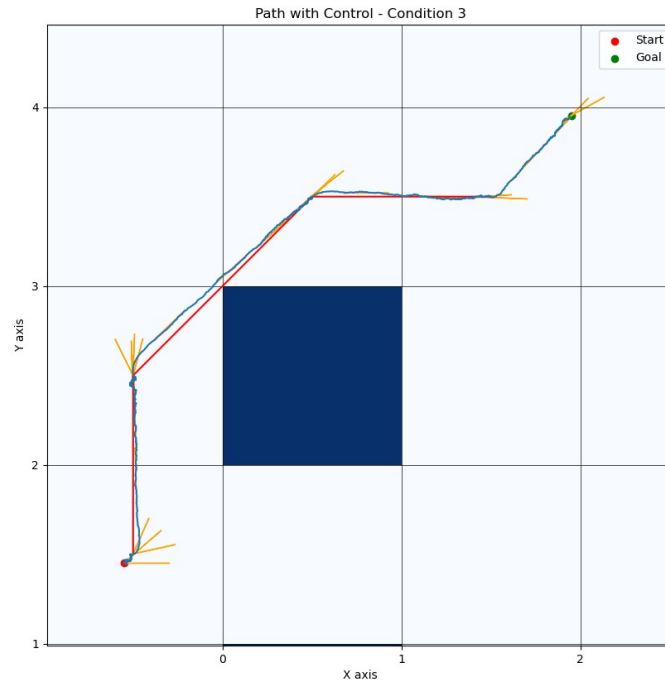


Condition 2





### Condition 3



In the future, to mitigate the backtracking seen in conditions 1 and 2, I could apply a strategy that would allow the robot to sense when it has entered the goal grid. The robot could then change its current goal node (the center of the grid) to the final goal position to minimize the backtracking error. However, this approach is still flawed and contains some unnecessary traversing because there could be a better angle of approach to the grid based on the location of the goal within the grid.

While I have discussed some disadvantages of this approach, I think it's important to note the decrease in headings plotted on the graphs compared to the graphs previously shown in question 10. In a coarser grid, the robot can move quicker and take larger steps because you don't have to fine tune the error as much as the finer grid. Using a larger grid can also decrease runtime for the those same reasons.

12. When creating this model, I did not make it so the environment can change and the robot still perform well. The offline version of A\* is not compatible with a changing environment, however online A\* should be able to manage because it only looks at the neighbors of the current node. However, I currently do not allow the robot to return to a grid cell that it has already been to, and because of that it cannot be robust enough to a changing environment or it might get stuck with no new node to explore. Additionally, the noise added to the motion model caused the resulting path to be noisy as well which can be difficult for an actual robot to perform those controls and move accordingly. Utilizing a grid sweep for the parameters in the PI controller could result in a smoother traversed path. Another option would be to implement a real-time filter on the motion data. Another simplification lies in our motion and control models. If these models do not realistically represent the actual behaviors of the robot and world, the error can propagate throughout the entire system. Finally, in my model, I do not have a cap on the linear or angular velocity of the robot. In the real world, motors have both a maximum acceleration and also velocity, and while I accounted for acceleration, I did not account for velocity.