1 Question 1

Done. See code.

2 Question 2

Done. See code.

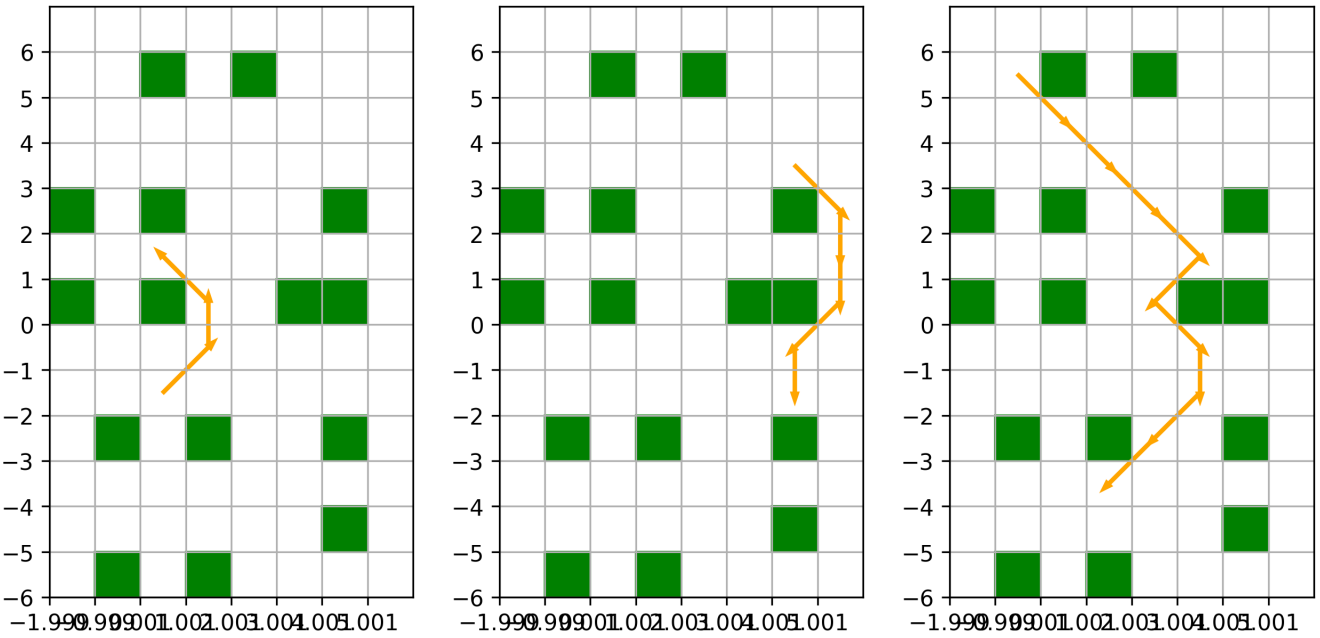
The definition of admissible is that it's always less than or equal to the true cost. In other words, it must be optimistic. Because in the problem statement the cost for diagonal or grid-aligned movements are both 1.0, that means our heuristic can't just be the euclidean distance (since that would be an overestimation of the diagonal cost).

Instead we use the exact cost it would be if there were no obstacles between ourselves and the goal which can be computed in a closed-form manner. That is, moving diagonally until we are axis-aligned with the goal, and then moving straight toward the goal. Because this uses the actual movement cost, but with no obstacles, we guarantee that the heuristic will always be less than or equal to the real cost.

$$h = min(abs(g.x - s.x), abs(g.y - s.y)) + abs(abs(g.x - s.x) - abs(g.y - s.y)) \quad (1)$$

The first term is the diagonal move, and the second term is the remainder of the distance to the goal as a straight-line move

3 Question 3



(a) Q3 Plan A. (b) Q3 Plan B. (c) Q3 Plan C.

4 Question 4

Done. See code.

The way I solved this problem was by creating an online-planner class which takes a robot, a map, and a planner as input. I also added *unexplored* variables to each *Node* class.

For this question I created a *GridRobot* which moves instantaneously from cell to cell

The cost for a node is then 1.0 if it's either free or unexplored (we assume the unexplored map is free), or 1000.0 if it's occupied.

Lastly, the online planner will iteratively plan, move the robot, observe the neighbors, and then update the map before repeating. So no change had to be made to the set of expanded nodes (besides adding the aforementioned *unexplored* variable)

When using an online A* with a sensing model, it means there's a chance our robot falls into a "trap", meaning that it could pursue a very promising direct route, only to find out there's walls in the way. The time it would take to sense the walls and plan around them would reduce the robot's time-to-goal compared to if it had complete environmental knowledge.

5 Question 5

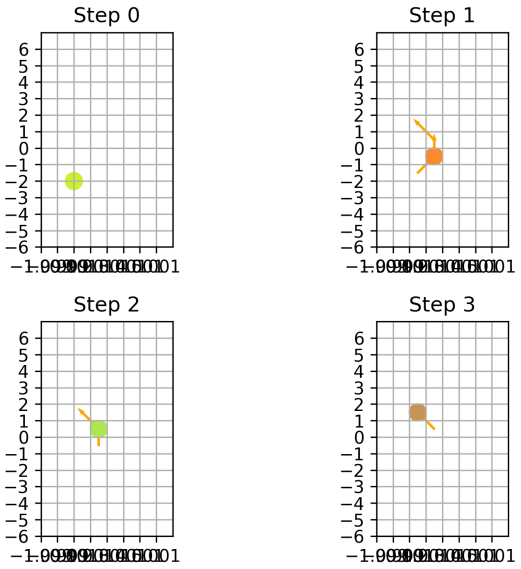


Figure 2: Q5 Plan A.

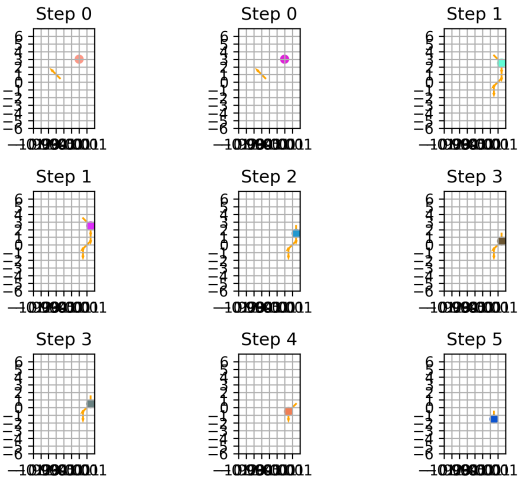


Figure 3: Q5 Plan B.

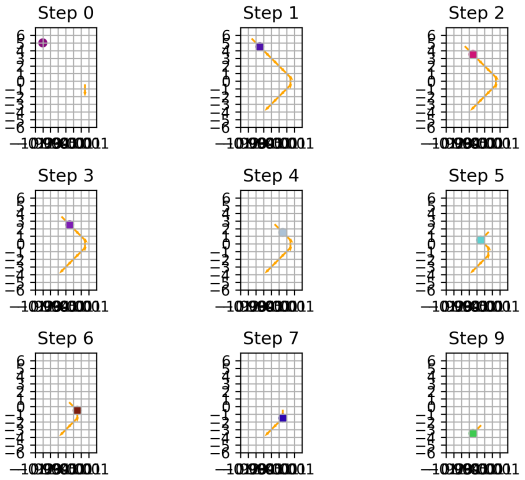


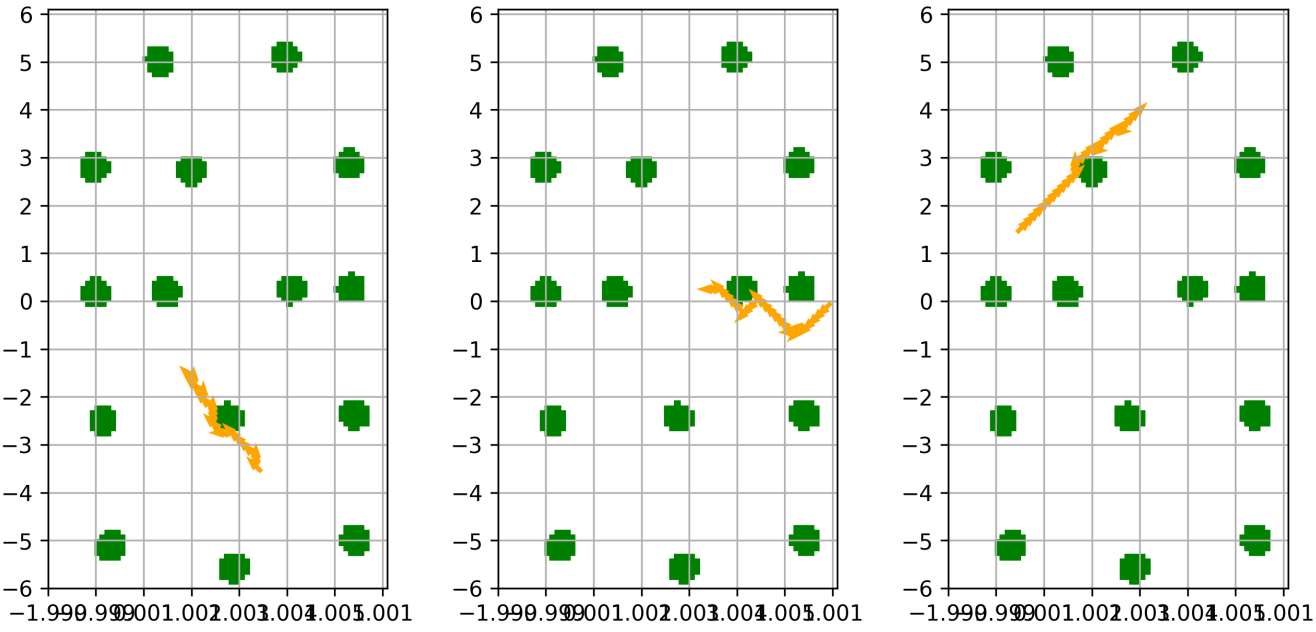
Figure 4: Q5 Plan C.

6 Question 6

Done. See code.

7 Question 7

Because the cost was 1.0 whether A* explored diagonally or straight, it means that the plan will choose diagonal moves that bring it closer to the goal, even if it creates jagged final paths. A simple fix would be to multiply diagonal move's costs by $\sqrt{2}$ to account for the extra distance covered.



(a) Q7 Plan A.

(b) Q7 Plan B.

(c) Q7 Plan C.

8 Question 8

Done. See code.

I used the state as (x, y, θ) and the controls as (v, ω). I use an value of $0.25 * cell_width$ to determine if the robot was near a goal. The value must be less than half the cell-width to ensure the robot only proceeds to the next waypoint cell if it actually entered the previous waypoint cell.

I use my motion model from assignment 0 so refer to that submission for more details on my motion model. I also use my range and bearing calculating equation from assignment 0.

For the controller I use a simple proportional controller based on the error in heading and range. The gains were tuned empirically to 0.5, and 0.5

$$v_{cmd} = K_v * e_v \tag{2}$$

$$\theta_{cmd} = K_\theta * e_\theta \tag{3}$$

The acceleration limits were enforced by numerically integrating them to velocity limits given the dt value of 0.1.

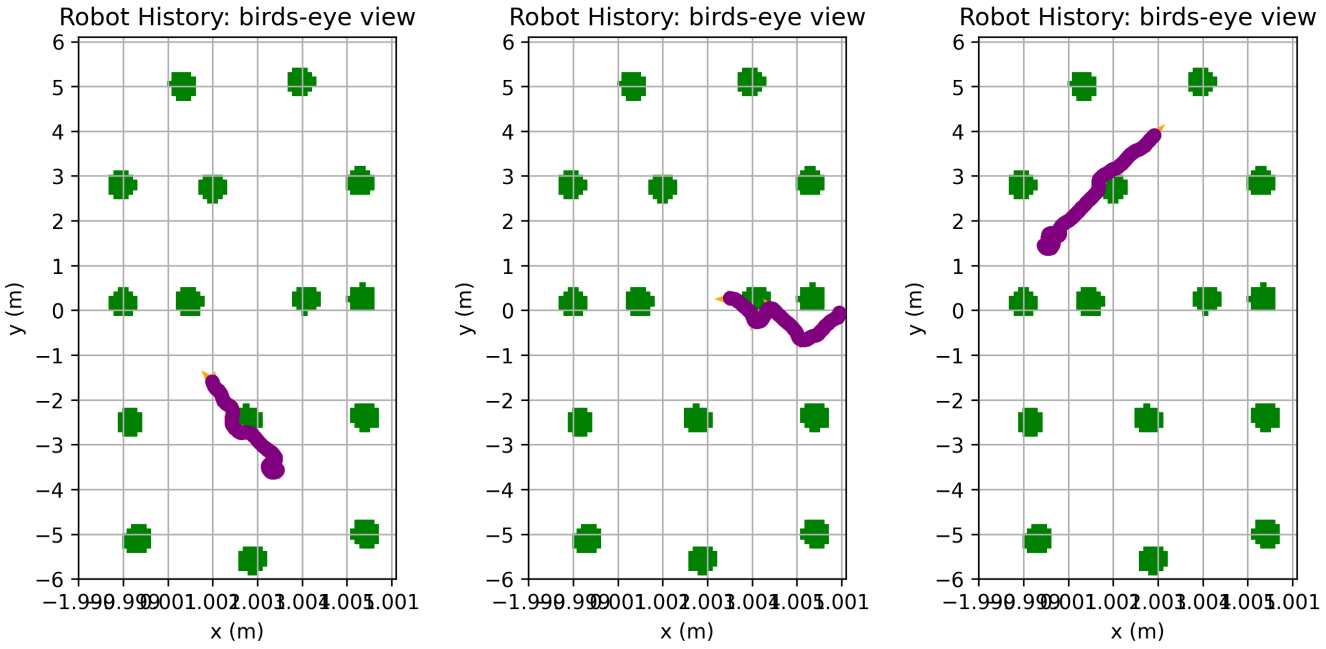
$$v'_{cmd} = sign(v_{cmd}) * min(v_{cmd}, v$$

$$_{max} * dt) \tag{4}$$

9 Question 9

The robot was able to successfully drive the paths generated in Step 7. Because my controller was proportional only (and not optimally tuned), the robot sometimes overshot the waypoint and had to circle back.

Continuous robot path in purple.



(a) Q9 Plan 7.A.

(b) Q9 Plan 7.B.

(c) Q9 Plan 7.C.

10 Question 10

Because I had already designed my code to be modular and allow both online planners and offline planners, as well as grid robots or continuous robots, it means that programming this step was relatively simple.

Continuous robot path in purple.

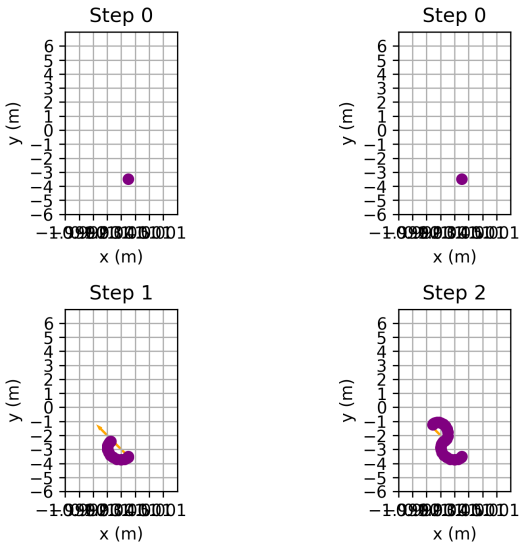


Figure 7: Q10 Online Plan 7.A.

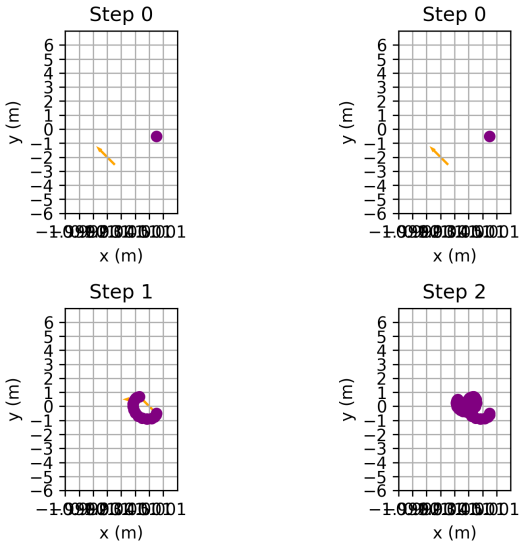


Figure 8: Q10 Online Plan 7.B.

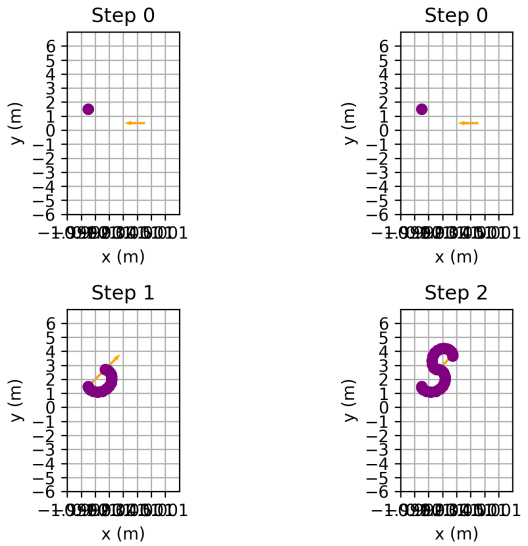


Figure 9: Q10 Online Plan 7.C.

11 Question 11

Operating on a coarser grid offers some benefits and detriments. First, the planner will complete sooner, meaning that it could be run at a higher frequency during online navigation. But due to the coarseness, it means the final plan won't follow the curvature of obstacles as closely, and therefore will likely take the robot longer to reach its goal.

Finer grids have the opposite properties. They run slower, in fact their computational complexity grows exponentially as the number of state variables increases $O(n^d)$, where n is the number of cells in a dimension, and d is the number of state variables. So increasing the grid density can be extremely costly in higher dimensions.

Finer grids will also yield quicker routes as the plan can skirt more closely around obstacles.

Some coarser grids also yielded curvier continuous robot paths. This may due to the fact that our search space doesn't include heading, which means there's no constraints on what heading the robot has as it drives through waypoints. With my proportional controller, if a goal is far away then it will begin driving more quickly, meaning the radius of curvature will be larger, resulting in the seen curvy paths. This can be corrected by either reducing the K_v value, or by including bearing in the search space, or by using a more complex controller which takes advantage of the robot's holonomic dynamics (ex: zero forward velocity until bearing is perfectly aligned).

Online paths with a grid-width of 1.0

Continuous robot path in purple.

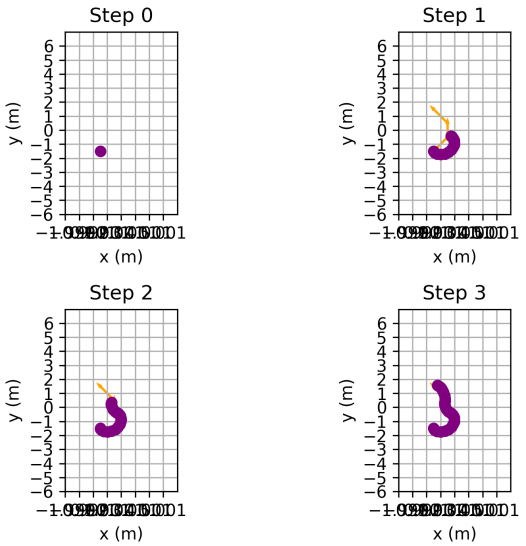


Figure 10: Q11 Online Plan 3.A, cell-width 1.0.

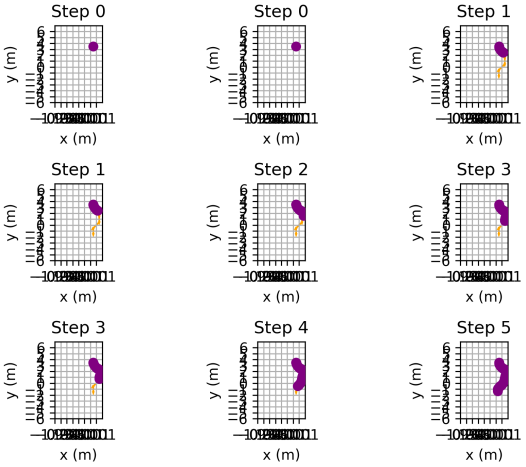


Figure 11: Q11 Online Plan 3.B, cell-width 1.0.

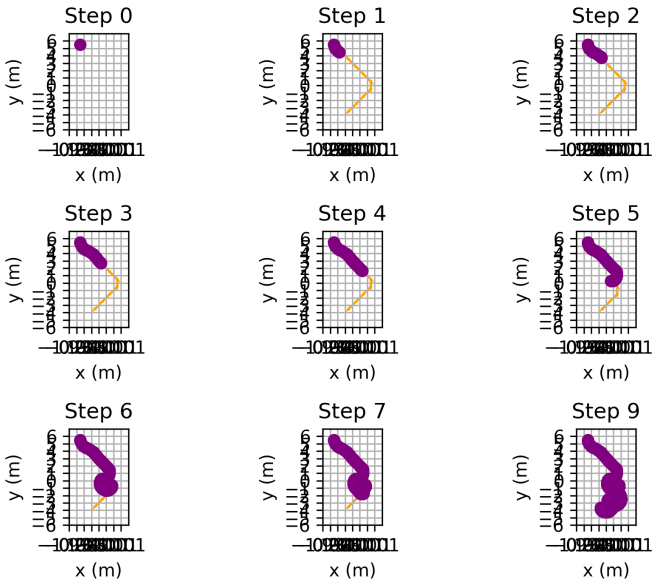


Figure 12: Q11 Online Plan 3.C, cell-width 1.0.

Online paths with a grid-width of 0.1:
Continuous robot path in purple.

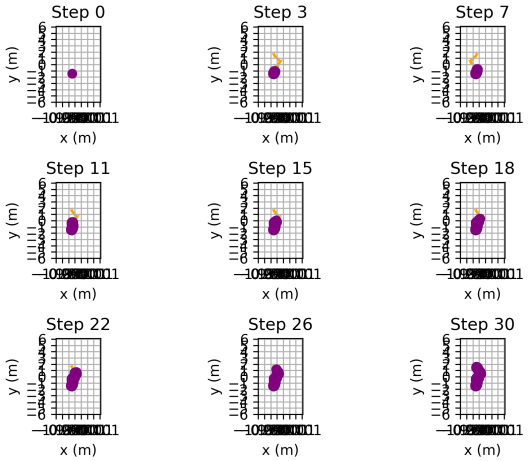


Figure 13: Q11 Online Plan 3.A, cell-width 0.1

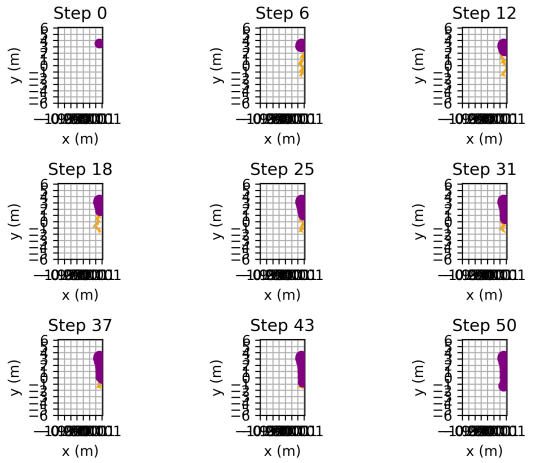


Figure 14: Q11 Online Plan 3.B, cell-width 0.1.

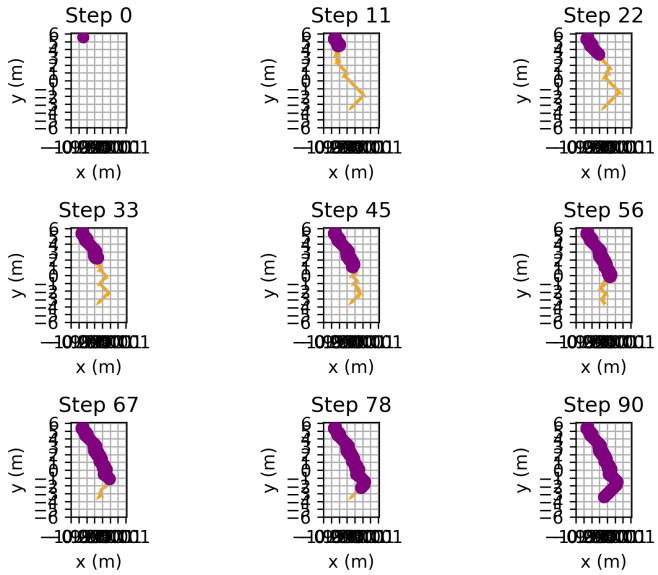


Figure 15: Q11 Online Plan 3.C, cell-width 0.1.

12 Question 12

In the real world robot sensors may not give accurate readings for obstacle locations. Real robots also may not have a great estimation of their own state. For both of these issues, sensor fusion methods and SLAM methods may be integrated.

Maps limits may not be known beforehand.

Obstacles may be dynamic, thereby requiring more frequent re-plans.

Robots doing anything interesting may also need to receive goals from some mechanism, be it another classical algorithm, a human, or a machine learning agent. This may limit the amount of compute available to the path planner.

Real robots may have many more dimensions (ex: multi-fingered robot hands) for which grid-based search algorithms like A* don't scale.