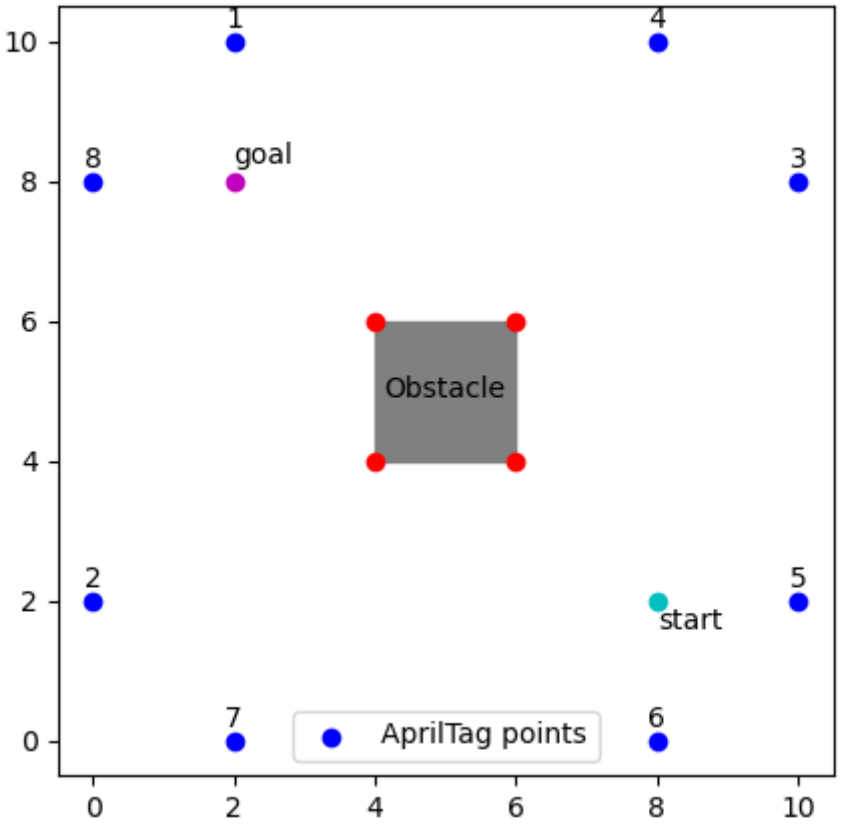
CSE 276A
Homework 4 - Path Planning

Rami Altai, Raghul Shreeram Sivakumaran

Problem setup:



Start at (8, 2) feet, Goal at (2, 8) feet

We treat the environment bounding box of 10ft x 10ft as an enclosed space, with each side as a 10ft long wall/obstacle. All graphs use units of ft unless otherwise specified.

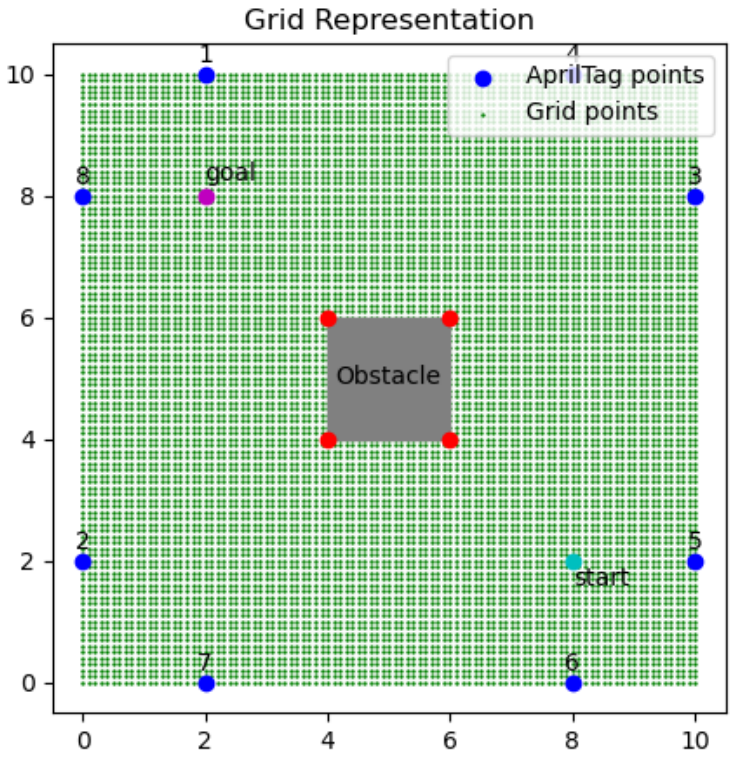
Minimum distance algorithm

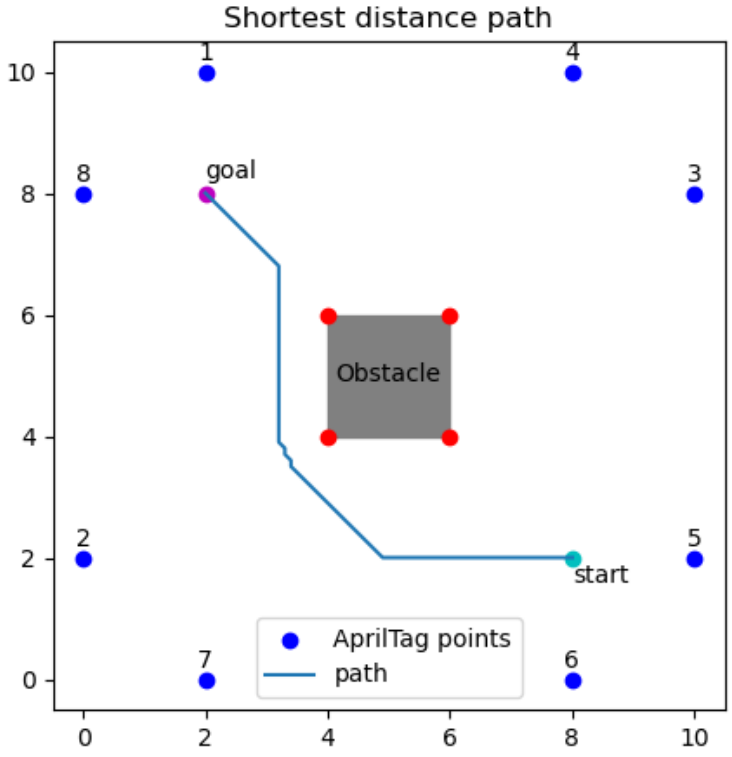
To find the shortest path from the start point to goal point, we represent the space as a grid for simplicity. In our grid, we allow the robot to move in the 4 cardinal directions, as well as on the diagonal ordinal directions.

To find the shortest path on this grid, we implement the A* algorithm, using Euclidean distance from the goal point as the heuristic. We choose the A* algorithm over Dijkstra's due to its faster convergence on average, in case we may use this for real-time path planning. The cost of an edge is 1 for cardinal directions, and sqrt(2) for ordinal directions.

To avoid running into obstacles, we use infinity as the edge costs of points that are within the obstacle space or are closer than the robot's radius away from the obstacle. This makes them prohibitive to traverse in the A* algorithm, so those nodes would not be chosen in the final path. This does make the simplification that the robot is circular, but this does not greatly affect the algorithm's outcome.

Representation and Path visualization:





The “steps” in the shortest path graph are small diagonal motions

Safety algorithm

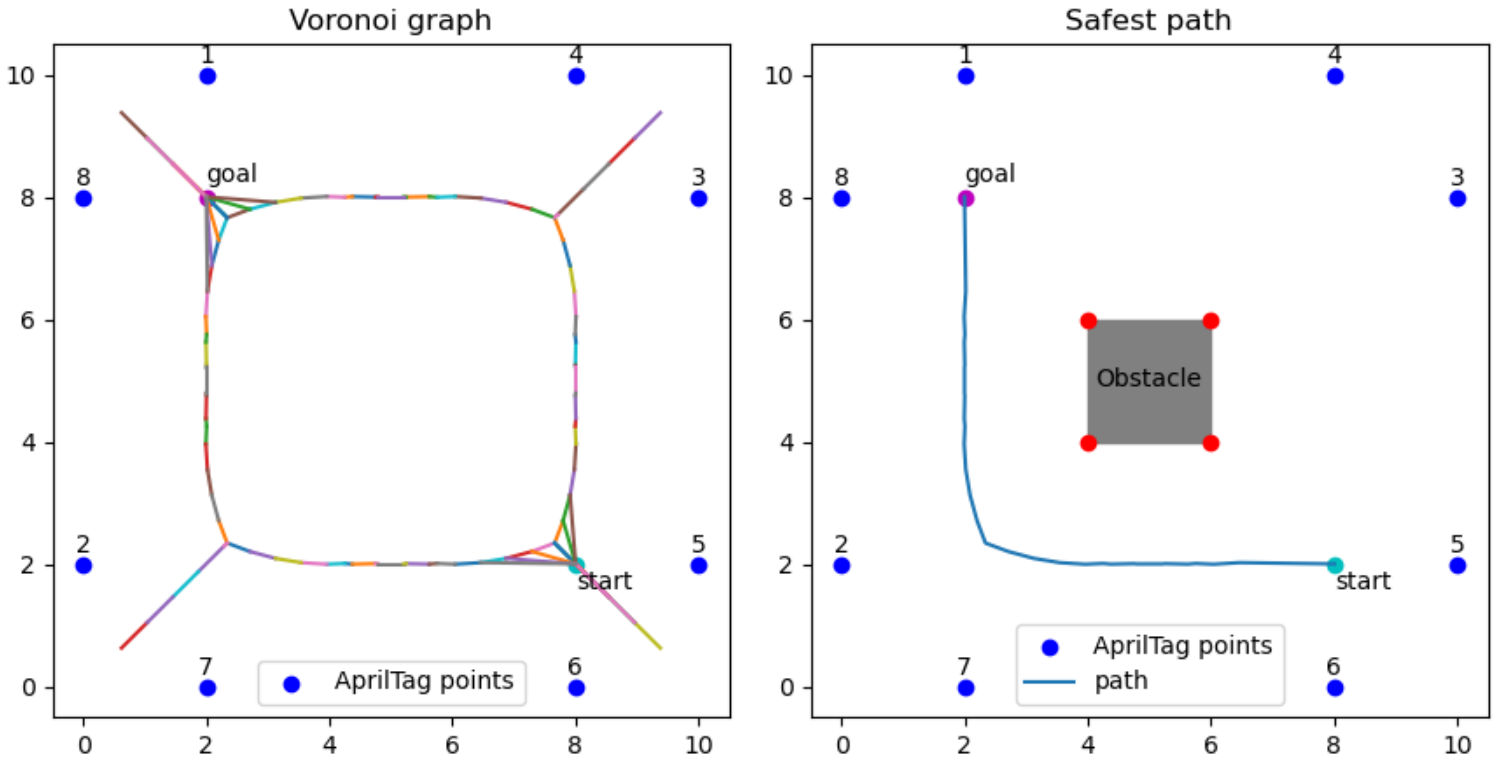
We define the safest path as being the path that is at all points as far away as possible from all obstacles in the scene, to minimize chances of collision.

We represent the space with a roadmap constructed by a Voronoi graph. By using the points of obstacles to construct the Voronoi graph, the edges of the produced cells represent the paths in the space which maximize distance from all obstacles. We then find the shortest safe path along these edges using the A* algorithm (our heuristic is again Euclidean distance to the goal). We also use Euclidean distance to define the cost of each edge.

As the start and goal points may not necessarily be on the edges of the Voronoi graph, we add them to the graph after construction, connecting both to the 10 nearest vertices in the graph (using Euclidean distance).

To make doubly sure that the robot stays away from obstacles, we remove all vertices and edges from the Voronoi graph that are in an obstacle or whose distance from an obstacle is less than the robot’s radius This again makes the simplification that the robot is circular, with minor effect on algorithm outcome.

Representation and Path visualization:

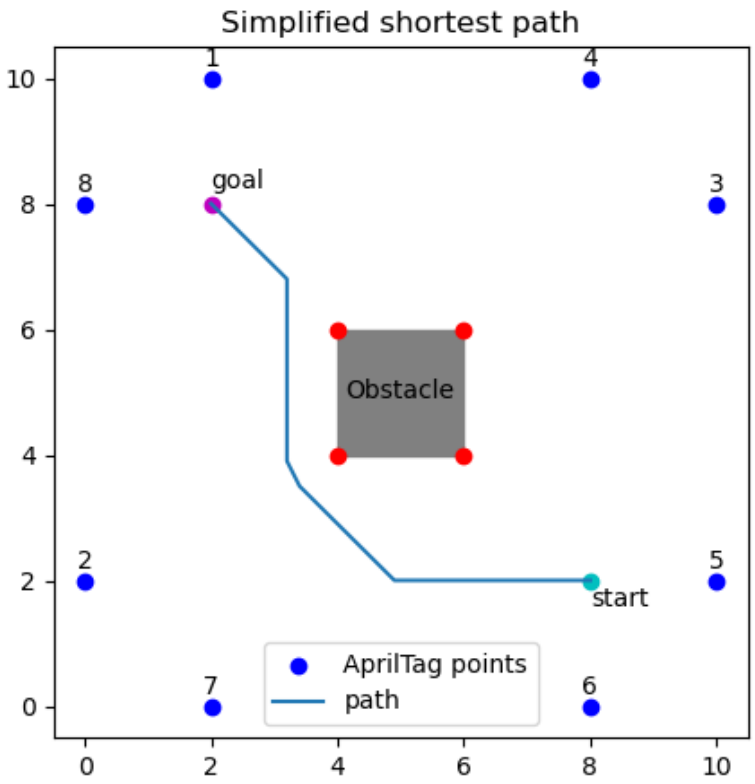


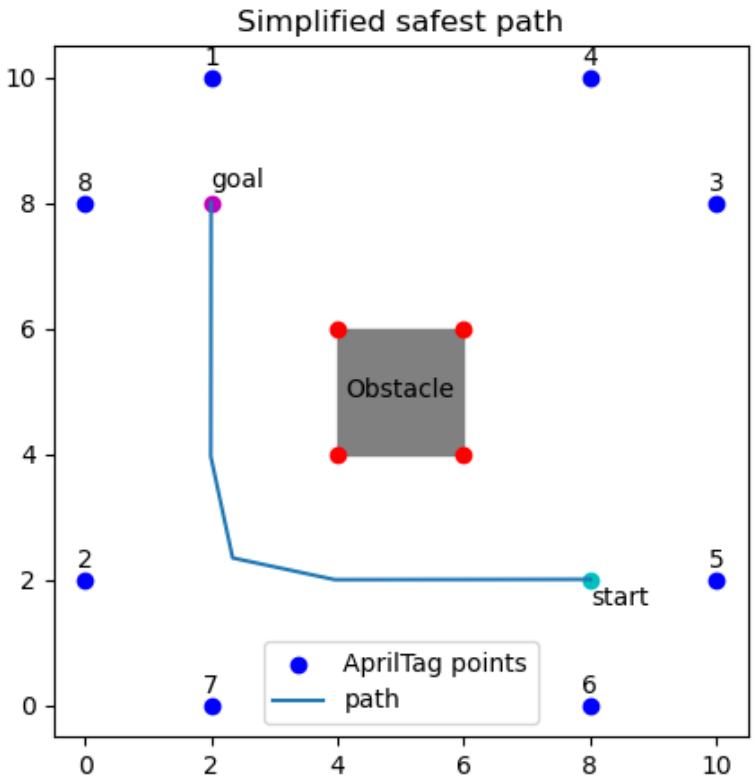
Each color on the Voronoi graph represents an edge.

Implementation

One disadvantage of both of our planning algorithms is the large number of points along the paths that are returned. Since we use a PID controller with distance to the waypoint as its error, the large number of points would result in small distances, meaning the robot would move very slowly at each timestep. We reduce the number of points in a path by removing unnecessary points that are in the same direction of motion (Ramer–Douglas–Peucker algorithm). A visualization of the reduced points for each algorithm is shown below.

Simplified waypoints visualization:



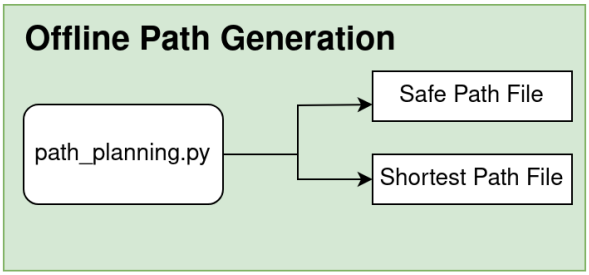


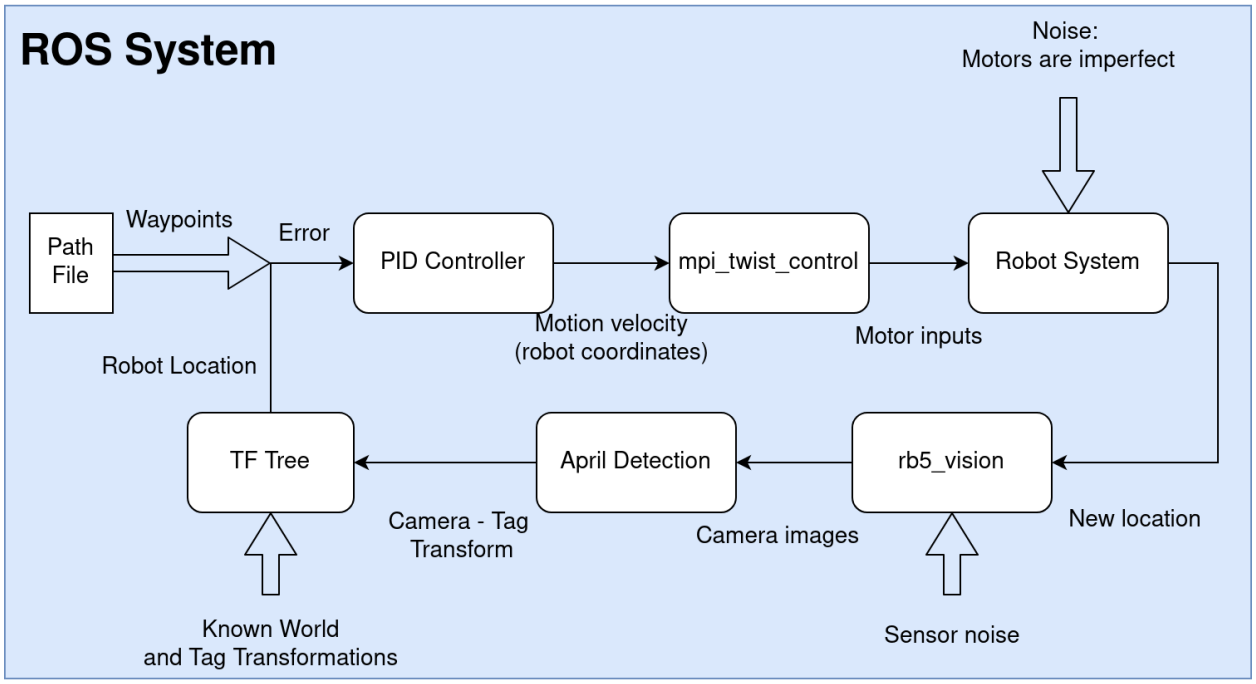
Grid points (left) were reduced from 92 to 7 waypoints. Voronoi points (right) were reduced from 31 to 5 waypoints.

As the start point, goal point, and map of the environment is known beforehand, we precompute the path for each algorithm, filter points as above, and then save the paths to disk to later be read in at the beginning of robot movement.

ROS Architecture Diagram

We build off of our HW2 solution, adding an offline path generation component.



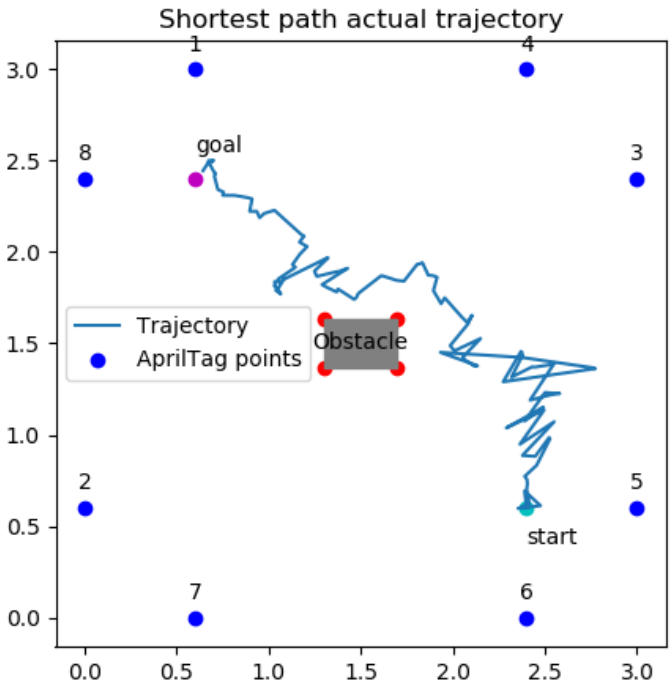


Results

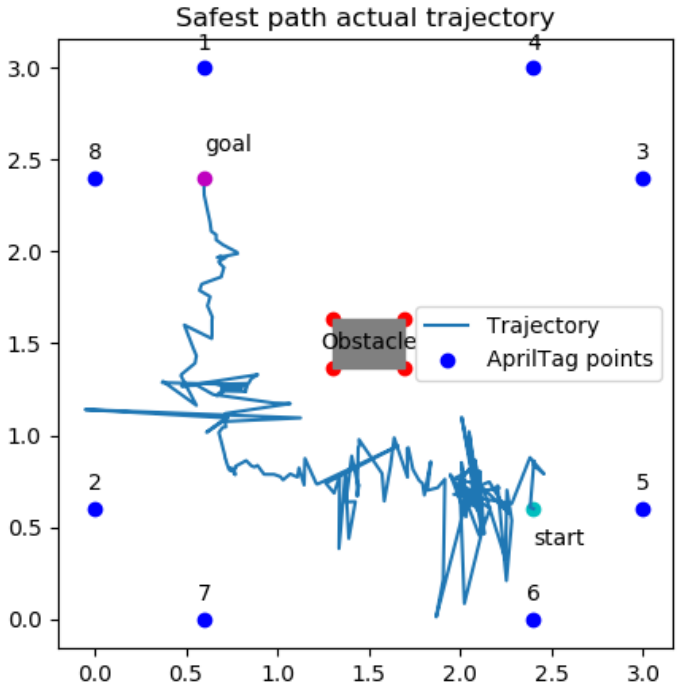
We scale down our map slightly to 3m x 3m as our prior code primarily deals in meters. Following graphs use units of meters on the x and y axes. Our obstacle was also changed to a rectangle that is 39cm x 26cm as this is what we had available.

Shortest distance path video: https://youtu.be/H3IRXTwfHCk

Safest path video: https://youtu.be/Z7JbDYoLoEI



Actual traversed path (in 41 seconds)



Actual traversed path (in 63 seconds)

It seems that some of these positions are inaccuracies in the camera estimation. For example, at no point in the video does the robot go to (0, 1.0) as shown in the graph.

However, there is still a lot of noise that affects the robot's performance. We believe this is primarily due to the lag from the camera sensor in providing measurements, causing the robot to receive position estimates that are up to a second in the past. This noise heavily affects the robot despite our efforts to slow the robot down as much as possible.