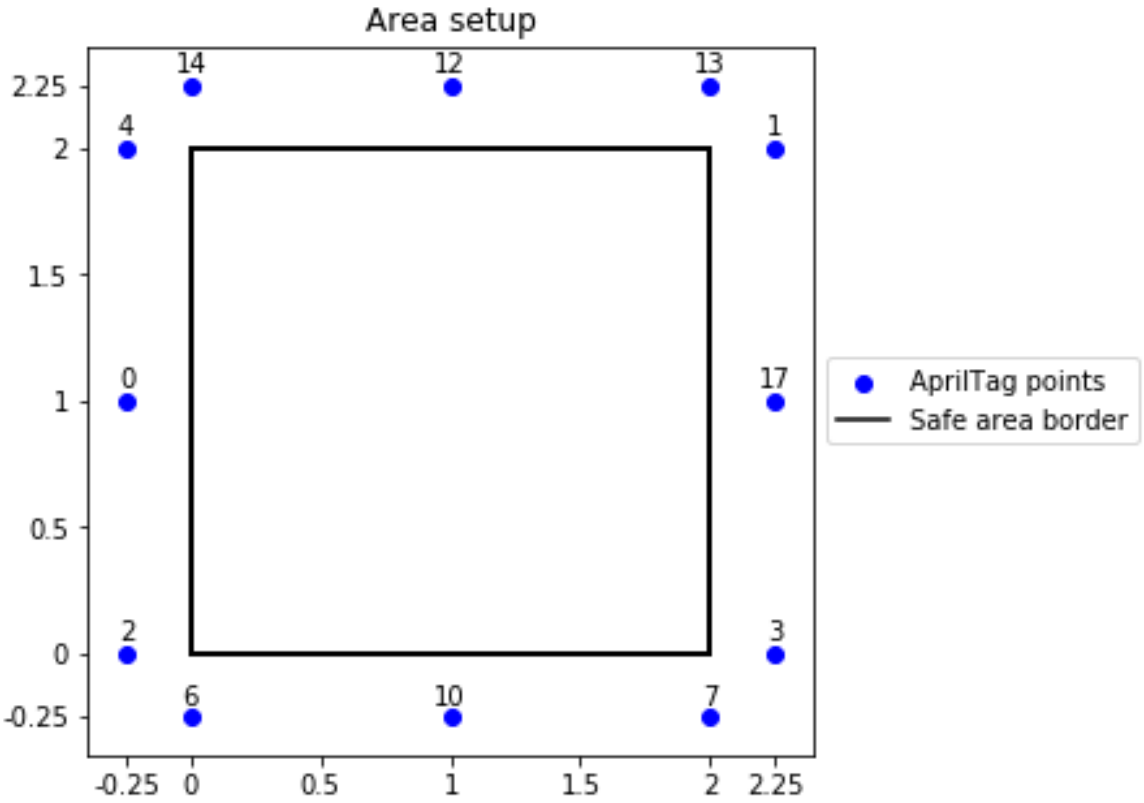
CSE 276A
Homework 5 - Robot Vacuum Using SLAM

Rami Altai, Raghul Shreeram Sivakumaran

Problem setup:



We treat all four sides of the landmarks above as solid walls. In the case of obstacles, we assume that on each face of the obstacle is a visible landmark.
We downscale to a 2.5m x 2.5m space due to space constraints. All graphs use units of meters on the x and y axes.

Approach

We chose an offline planning approach to guide our “Roomba” robot. Given the positions of the walls and obstacles, we create a path to take us through the entirety of the environment (providing coverage), whilst staying a safe distance away from obstacles and walls (avoidance). This gives us the advantage of being able to create a precise path plan, but also requires us to have knowledge of the locations of the walls and obstacles in the environment beforehand.

Once we have a path plan, our robot is able to read in the list of waypoints and follow it point by point, using SLAM and AprilTags placed on the wall and obstacle faces to localize.

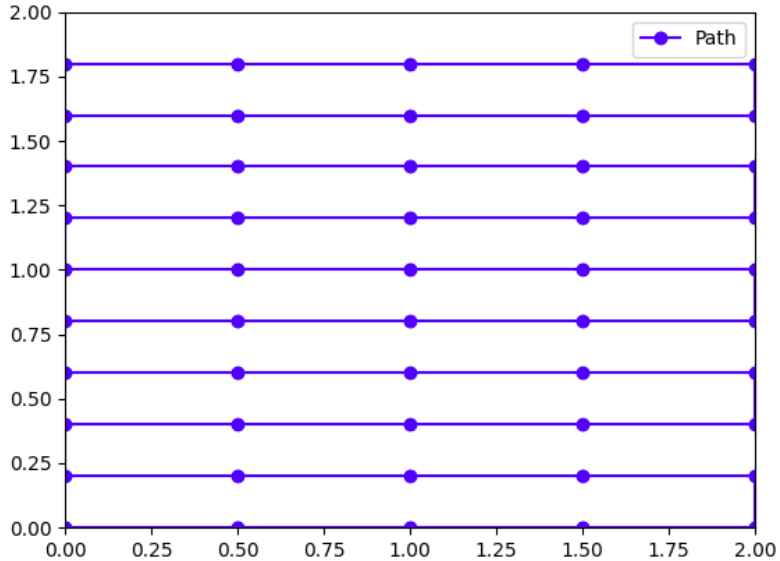
Path Planning

To ensure that we avoid hitting obstacles and/or walls, we limit our paths such that they must stay within a “safe distance” of any obstructions. To ease distance computation, we treat our robot as a circle, and find its radius to be ~12.5 cm. As the robot may drift and its localization may not be precise, we add 5 cm as an additional buffer to define a “safe distance”. Due to our grid representation and pathfinding algorithm explained in further detail below, this distance must additionally be multiplied by sqrt(2), which gives us a final definition of “safe distance” as being at least ~25 cm away. We use this definition “safe distance” for the rest of this report.

Path planned for the robot without any obstacles:

We used a sweeping back-and-forth path to cover the whole grid and generate the waypoints for this path. This is an offline path planning i.e, we generated the waypoints and plugged it to the robot to move.
Plot of the sweeping path:

This moves left -> right, up, right -> left, up, left -> right, etc.



Path Planning for the robot when there's an obstacle:

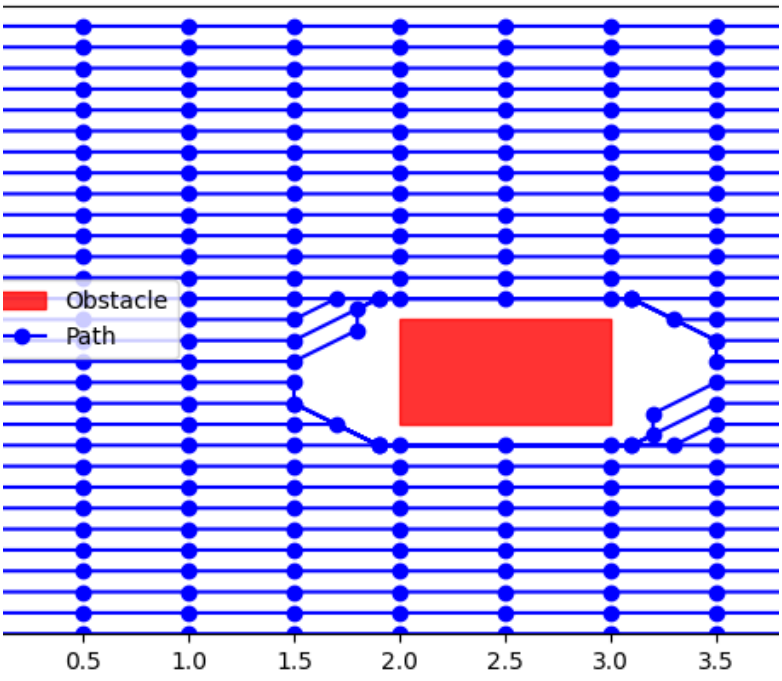
Motive: The robot should cover all the area except the area covered by or very close to the obstacle.

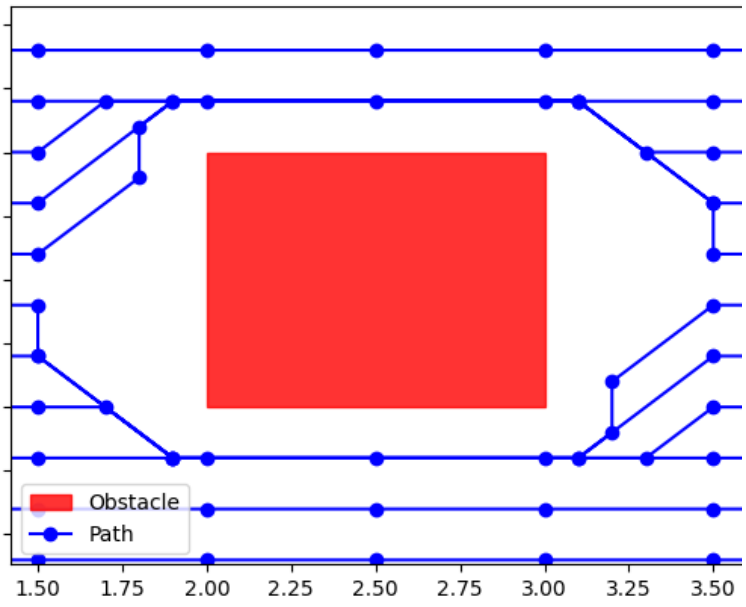
Approach: For the robot to cover the whole area except the obstacle's area, the robot should change direction when it is near the obstacle and go around it.

To generate our path, we first create a basic path assuming no obstacles like the one above, and then modify it. We remove waypoints that are not within a "safe distance" of the obstacle, and then reconnect the graph using the A* algorithm.

In detail: When the robot approaches the obstacle, we check if the next waypoint of the robot is inside the obstacle beyond the obstacle. If this condition is true, we consider the current coordinate or waypoint of the obstacle as the start point in A* star algorithm. Goal point is decided by two conditions:
Condition 1: If the obstacle is between the current robot waypoint and the next robot waypoint, the goal point is considered as the goal point in A* algorithm.

Condition 2 : If the next robot waypoint is inside the obstacle or on the edges of the obstacle or is very near to the obstacle, we move on to the subsequent point and check if this subsequent in inside the obstacle again until we get the next waypoint that is actually not in the obstacle and consider that as goal point in A* algorithm.





Left: Plot of the full path when there's an obstacle in the environment
Right: Enlarged region of the area around obstacle

We can see some intersections in the plot because the robot moves to the next waypoint using A* algorithm and retracing its own path to move to generated waypoints again to sweep across the whole area.

Space representation and A*:

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. In order to make sure the robot's "safe distance" radius is not broken when traveling on a diagonal, we must additionally multiply the "safe distance" of the robot by sqrt(2), as previously mentioned in the "Approach" section.
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.

Path simplification

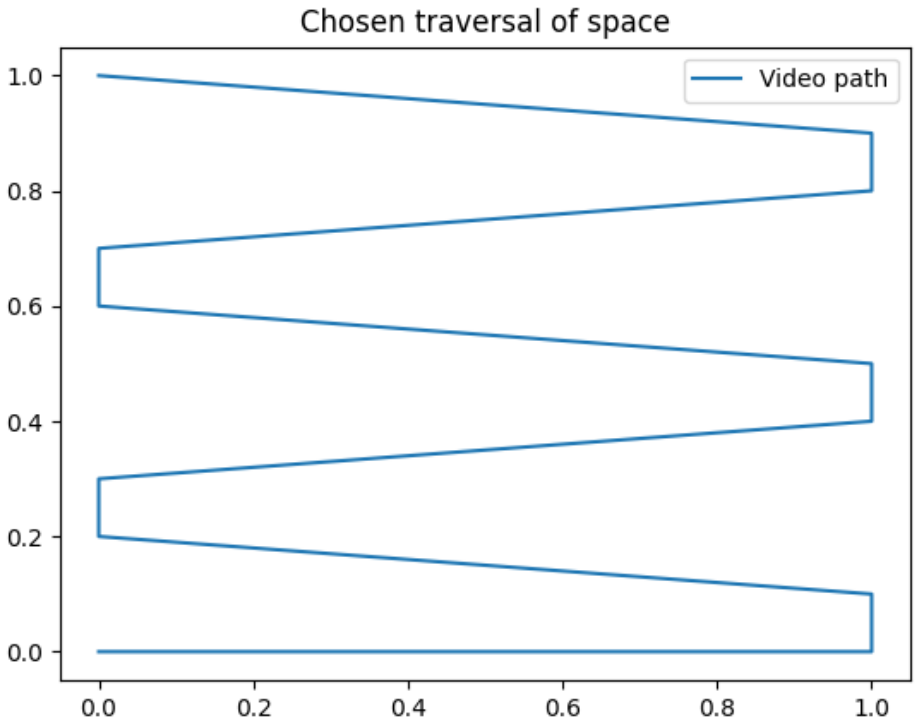
One disadvantage of the output of our planning algorithm is the large number of points that A* pathfinding provides, due to the small grid discretization needed. Since we use a PID controller with distance to the waypoint as its error, the large number of points would result in small distances, meaning 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). The above visualizations include this reduction of points.

Traversal: Row spacing issue

When designing our path algorithm, we wanted the robot to cover some of the same space on consecutive rows (or "sweeps"). This would allow us to have greater confidence in our total coverage should the robot veer off of our path due to poor localization. We originally decided that per row, 37.5% of the robot's width would re-traverse space from the previous row, with 37.5% chosen because it was a middle between 25%, which seemed too small in case of large veers, and 50%, which would take too long to run.

Our robot is 16cm wide, so this meant each row should be spaced 10cm apart in our path planning algorithm. However, we were unable to get our robot to consistently perform small motions without the motor stalling. If we tried to calibrate our robot for small motions, larger movements would then be imprecise.

We ended up making a compromise. In the path planning algorithm, we set the spacing between rows to be 20cm. Empirically, we found this actually achieved the 10cm distance we desired. However this meant that while traversing a row, the robot's localization would slowly move it to the actual 20cm position, causing our horizontal rows to become slanted. **Example below:**

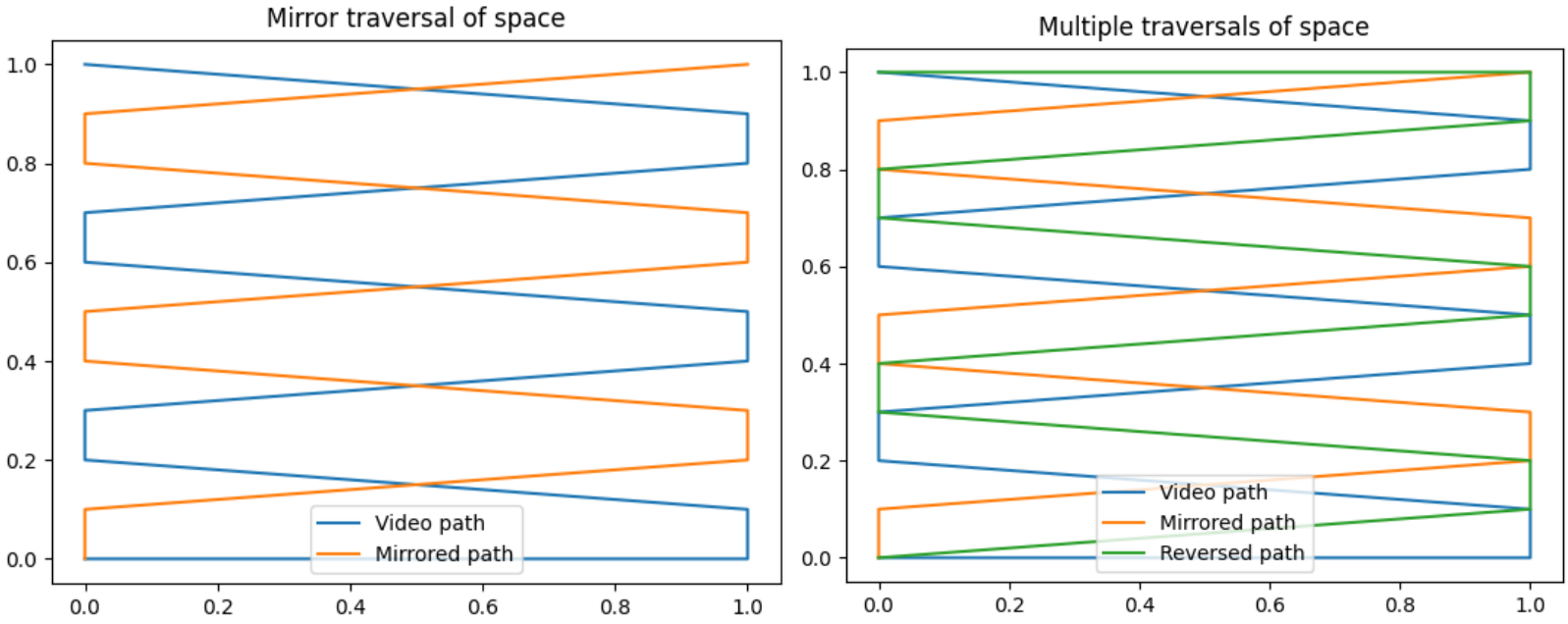


This method means that if the robot follows our path exactly, there are spaces that are missed (gaps >10cm). In practice, these uncovered spaces are very small, but this is not ideal.
We considered two ways to fix this issue.

- (Option 1): We create two calibration settings in our PID controller, one for small motions, and one for large motions. Based on the desired distance, we alternate which calibration we use. However, this would require a whole new re-calibration, and after having calibrated 3 different robots this quarter (we encountered many faulty motors), we weren't keen on this idea.

- (Option 2): Without any modification to our existing motion controller we traverse the space again, but this time in opposite directions. This would mean our slanted rows would have the opposite slope as before, and the missing spaces are covered.

We opted for option 2 and accepted our compromise. Although we do not show this behavior in the final video due to time constraints, the only changes necessary would be to append the same original list of waypoints, but in mirrored and reverse orders.
Example of option 2 trajectories:



Left: Example of actual path traversed + mirror path
Right: Example of full space traversal with mirrored and reverse paths
(No gaps in space >10 cm)

Implementation

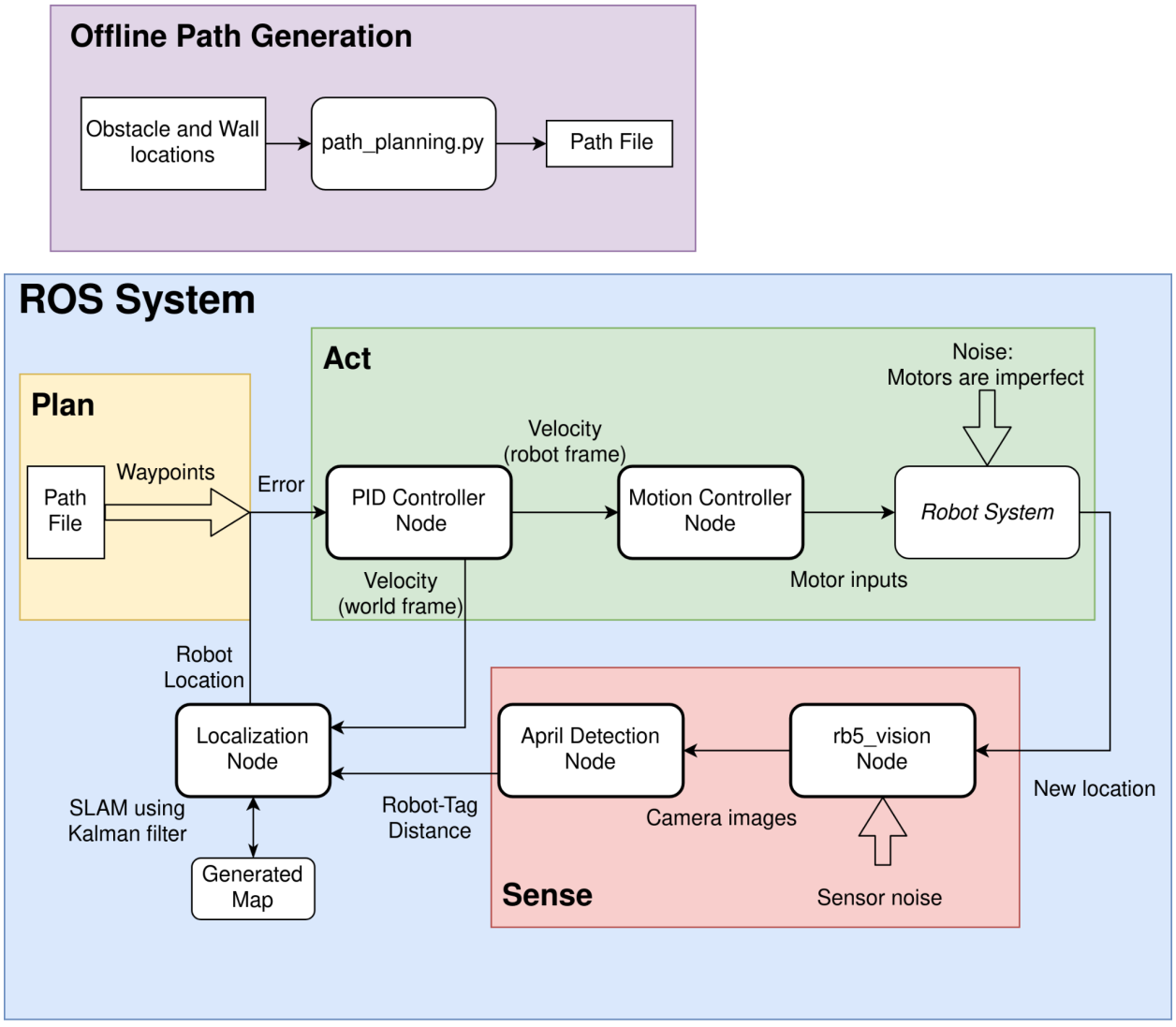
Modifications to our SLAM algorithm:

The details of our Kalman filter implementation has stayed the same since HW3, but we had to make some modifications to our algorithm as our original mapped its surroundings poorly. In summary, we: (1) Fixed bugs in Kalman filter creation, (2) Took into account AprilTag detection delays, and (3) Moved the localization processing to its own node (because independent components deserve to be independent).

Sensor delay in particular heavily impacted our robot's performance. We saw an average delay of 0.9 seconds when receiving AprilTag detections. This heavily misled localization during rotations, as the robot would localize off of non-rotating AprilTag detections during rotation and constantly overshoot the desired angle. To solve this, we took inspiration from the HW3 solution and only updated our Kalman filter with measurements that were at least 0.9 seconds old. This gave a localization that was 0.9 seconds old, which we updated to the present moment using our motion model (previous velocities * timestep = change in position).

ROS Control Flow Diagram

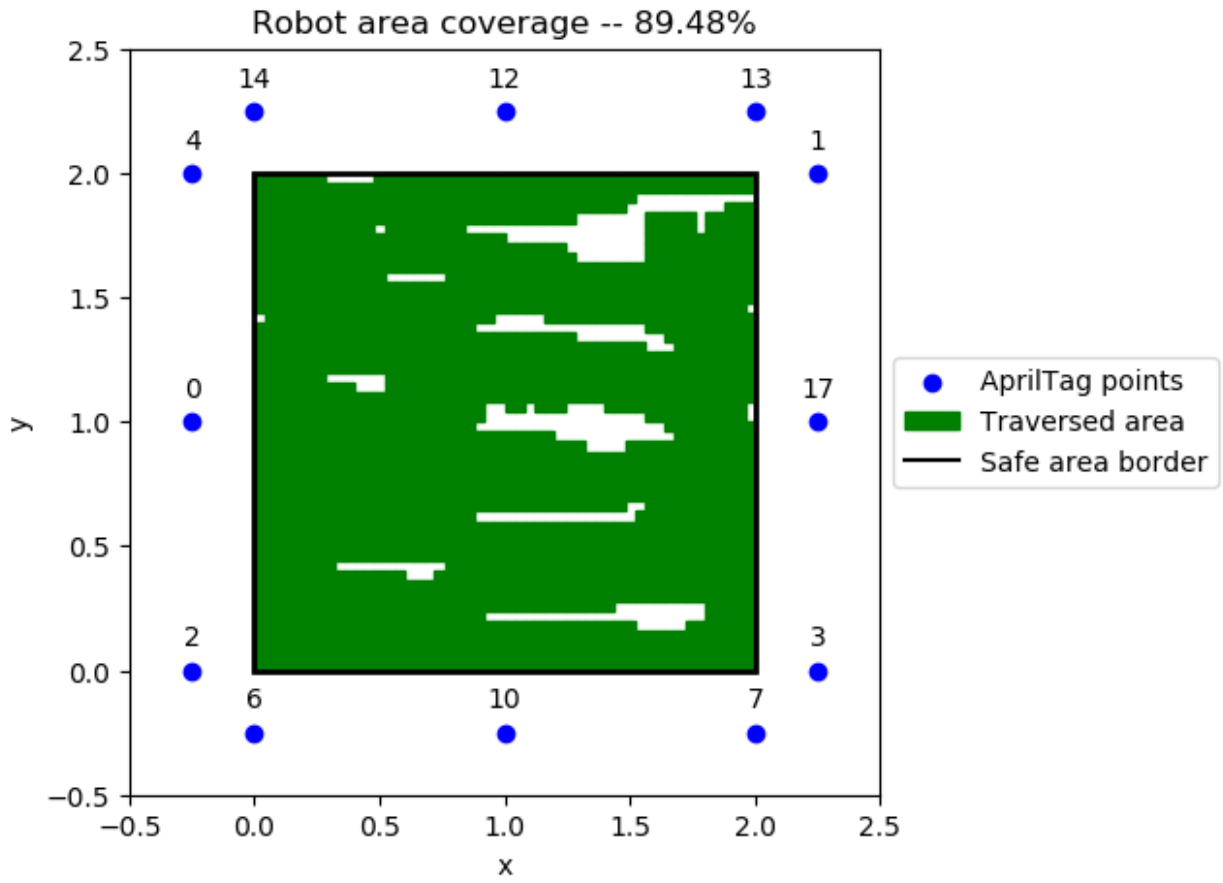
We build off of our HW3 solution, adding an offline path generation component, and separating our localization to its own ROS node.



Results

Video run: https://youtu.be/ohH0A4W_Ij4?si=LK3VOiyGeWFWbgQv

Coverage

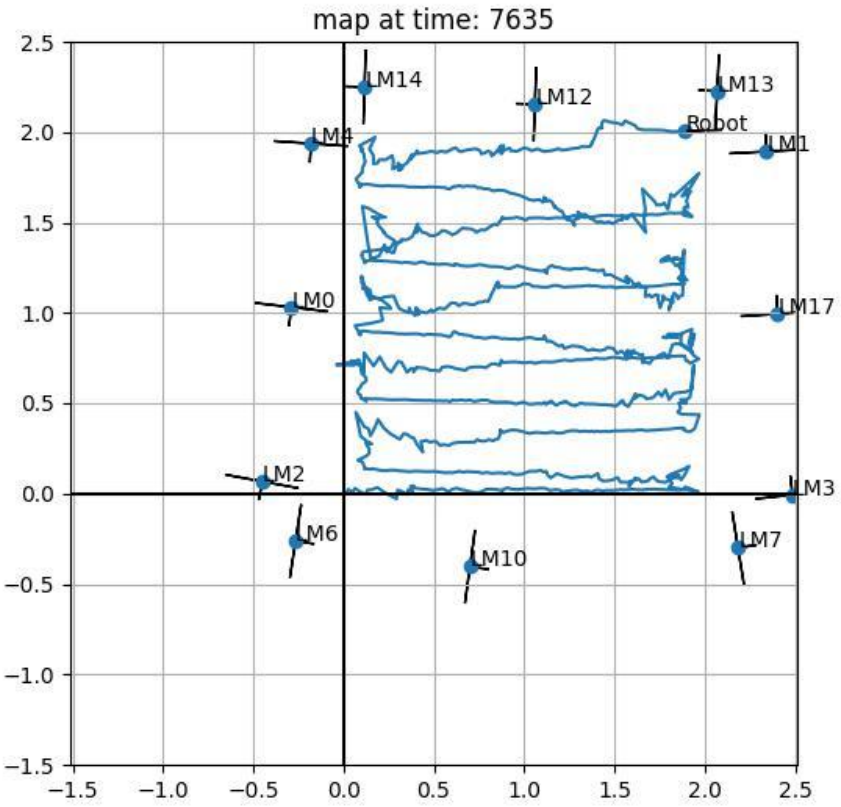


This coverage map was created by discretizing the 2x2m area as a grid, and coloring each grid square green if the robot passed over the center of that square. The robot is treated as a rectangle, with a measured width of 0.16m and length of 0.19m. Rotations are taken into account. We cover almost 90% of all grid squares in our video trajectory.

SLAM Map

This plot was created using a modified version of the given hw3 submission code for plotting trajectories.

Video of the localization plot over time: https://youtu.be/2er3U5Xvdgs?feature=shared



Interestingly, the bottom landmarks become more poorly estimated over time. We believe this is due to the robot never facing the -y direction, and continually moving in the positive y direction. Thus the bottom landmarks are only detected at the edge of the robot's field of view, which gives poor pose estimates. As the robot goes farther away, these estimates become worse and disrupt the estimated location of the bottom landmarks.