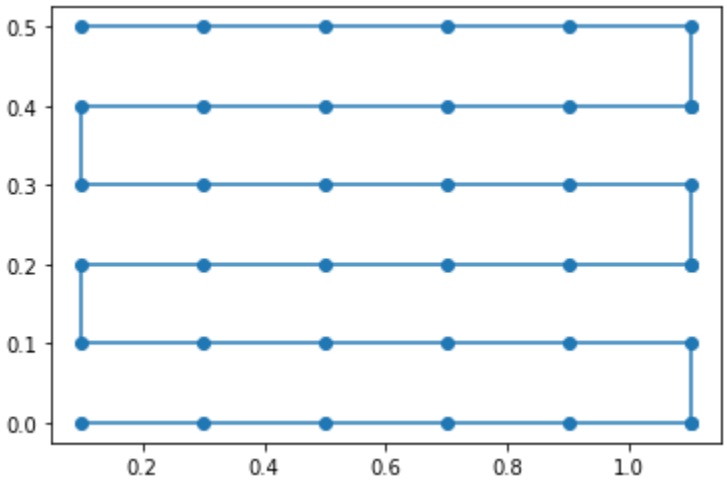
Algorithm for Coverage for a rectangular map like the one given:

Note: We write our algorithm assuming *No obstacles* since for the practical implementation for this homework there are no obstacles. We extended the algorithm to include avoidance later in the report, but it was not implemented.

1. Note the jetbot dimensions : (l,b). At every position, the jetbot covers some area. We have to calculate waypoints such that the union of the area covered by the jetbot at every waypoint is the total area that we want to cover.
2. Map Corner points: (0,0), (w,0), (w,h), (0,h) in order. They are enough to define a rectangular area.
3. Start at (0,0). Choose waypoints by incrementing x coordinate by l (length of the jetbot), as this is the direction in which the jetbot moves. Increment x till x >= w.
4. This way, we have covered the area = b (i.e. width of jetbot) * (w + l) > b*w
5. Choose the next sample point by incrementing y coordinate (from 0 in this case) by b (width of jetbot).
6. Now decrease x coordinate by l for each waypoint, and do this till x = 0.
7. Now, by moving up and down the x axis, we have covered an area of (2*b) * w.
8. We keep repeating this (moving up and down), and everytime we reach the end of one direction, increase the *y* coordinate by b.
9. We do this till we have covered both the coordinates (w,h) and (h,0). This will happen when we have y coordinate (ceiling of int(h/b)) times. For the last increment, if h is not a multiple of b, then we keep y = h instead of crossing the map. After this last increment, we move towards (0,h) if we are at (w,h) or the other way.

This way, we should ideally cover all the required area. Below we show the trajectory selected by this algorithm.



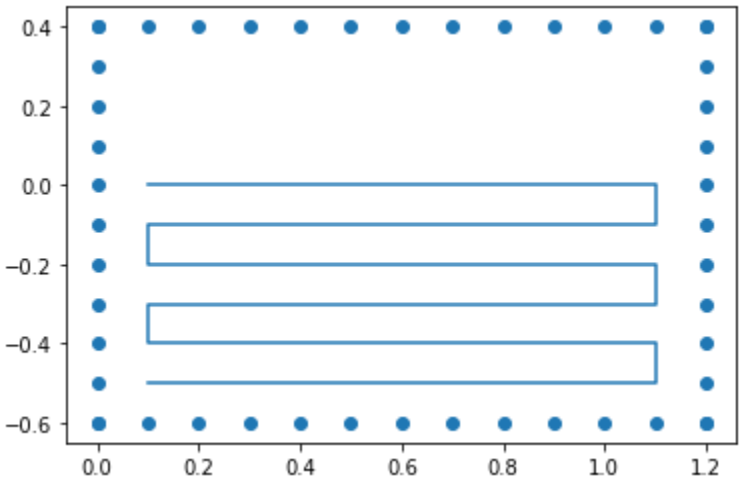
In this figure, the map that needs to be covered is (0,0) to (1.1, 0.5)
The jetbot will start from (0,0.1), goes to (1.1,0), increases y by 10 cm and goes to (0,0.1). This continues till it reaches both (1.1, 0.5) and (0, 0.5).

The above waypoints cover the entire area, as at each waypoint it covers a rectangle of area (l*b). Each path along the x axis is separated by a distance of b.

Implementation details: (Video : https://youtu.be/_ZCEcQk53TU)

Non-Unique Landmarks problem:
We solve the issue of not knowing the april tag ids as follows:
We calculate the global pose using every april tag's pose matrix for the id we get (1 or 2). We compare each global pose with the estimated pose (using open_loop estimation just for the current movement). The global pose closest to the estimated pose is chosen as the correct jetbot pose.

Also, our area to be covered less than the map covered by the april tags. We keep our actual map boundaries 25 cm away from the april tag boundaries so that the jetbot has enough distance to view the april tags properly.



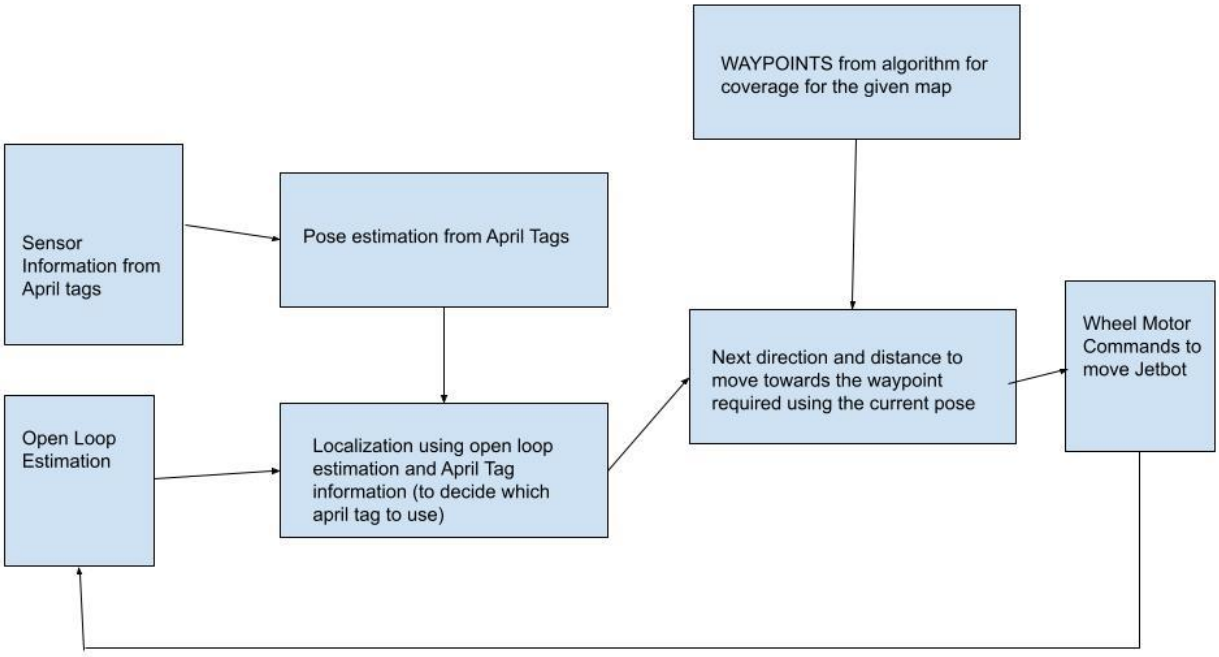
For the video purpose we only cover a part of the entire area due to time and battery life constraints. We have assumed that if the robot travels half of the area in an almost flawless manner, it will be able to travel the entire area flawlessly. The path travelled by the robot in our video is shown in the graph below.
Thus, for practical purpose, we only give our "map to be covered" as partial area, and the jetbot is able to cover the required area by almost 100%

Obstacle Avoidance:

The voronoi based path planner that was designed in homework 4 made sure that our robot never runs into obstacles. For homework 5, we did not have any obstacles. But in case we are to have obstacles:
We keep our original waypoints obtained from the above algorithm. If an edge passes through an obstacle, we remove that edge. In order to go from these nodes, we use a voronoi path as obtained in hw4 (with the dijkstra algorithm with edge distance as weights instead of BFS).

Control Architecture:

We follow a simple architecture as follows :



Performance Guarantee:

1. Our localization threshold of error is set to 4cm in our code, as in keep moving to the current waypoint as long as the jetbot is within 3 cm radius of the waypoint.
2. With unique april tags (i.e. we have id information), the localization information given by the april tag is perfect, as this is what defines our map. We test for multiple points for each april tag, and verify that the coordinate in the x-y plane is the same as the ones given by every april tag.
3. Thus, for this case, the only deviation from the waypoints calculated by the algorithm is from the 3cm threshold set by us. Thus, in theory some area can be uncovered if the jetbot deviates on the y axis by 3 cm for every straight path along the x axis. So, We can guarantee that the jetbot will cover at least [(b-3)*w]*(h/b) area, (as there are (h/b) straight paths along the x axis). This is = (b-3)/b fraction of the total area (w*h). Thus, for our case, b = 10 cm. So we can give a performance guarantee of 70% area coverage.
4. For non-unique landmarks, sometimes an error might occur when we select which april tag to use using our open loop estimation. But, if the april tags are far away from each other, that is the position given by each tag is very different from each other, and if we move our jetbot in small steps (both translation and rotation), then our open-loop estimation can be very close to the actual global pose, and the poses given by the april tags are very different so the correct april tag is always easily identifiable. Thus the performance guarantee does not go below 70% by a significant amount.

CSE 276A Video : https://youtu.be/_ZCEcQk53TU Rohin Garg|Shreya Saha
HW 5 A59002346 |A53299278

Notes:

1. Our current algorithm assumes that the robot knows the map of its surroundings beforehand.
2. Sometimes there is slight deviation in the measurement of the global orientation of the robot as seen from two or more landmarks. Due to this, the robot at times spends some time in fixing its pose for a longer period of time.
3. At times if the robot cannot see any April tag, it assumes that its estimated position is equal to its global position.