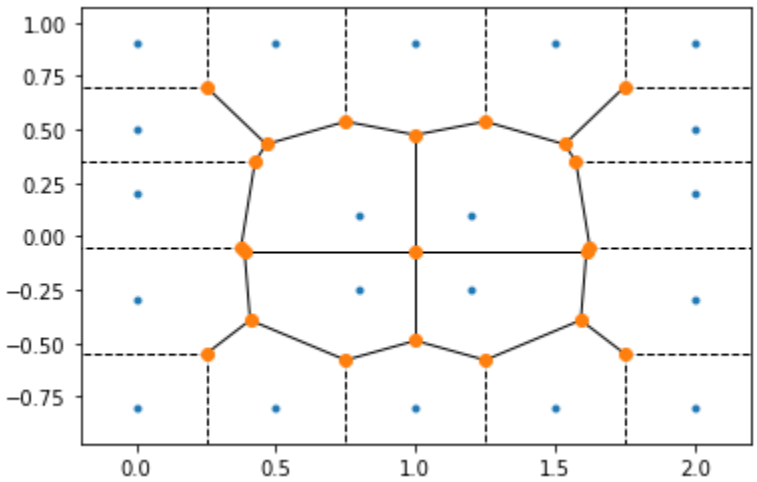
Path Planning Algorithm:

1. Choose points along the boundaries that restrict the robot navigation space. We choose points on the boundary of our map, formed by our april tags as the robot cannot go outside this region. We chose points along the boundary of our obstacle(s) as the robot cannot go through the obstacle(s).
2. Create a voronoi diagram for the selection of points.
3. In the resulting voronoi graph, eliminate the edges that cross the obstacle region and the edges that go out of the map.
4. Now we have a graph that gives us a path from 1 region to another without crossing the obstacle or going out of the map.
5. Given a starting point and an end point: Choose the vertices on the graph closest to the actual starting and end points. For simplicity, we chose our starting and end points as one of the graph vertices.
6. Using these starting and ending vertices, use Breadth First search on the voronoi graph to get the shortest path between the vertices.
7. Use the vertices in the voronoi graph that lie on this shortest path as the Waypoints for the jetbot to travel to, one after another.
    a. **Since our motion model is still point and move, the translation is always in a straight line which is made sure by using small translational steps and correcting the course on the straight lines via localization using the april tags. So our jetbot never deviates from the edges of the graph.** We could have sampled more points on the edge for a different motion model, but in our case we don't need to do so because of the above mentioned reason.
8. For the second endpoint, we make our 1st endpoint as the new starting point and get the new path via BFS and continue moving.

Our algorithm is inspired from this project: Robot Path Planning Using Generalized Voronoi Diagrams. (columbia.edu)

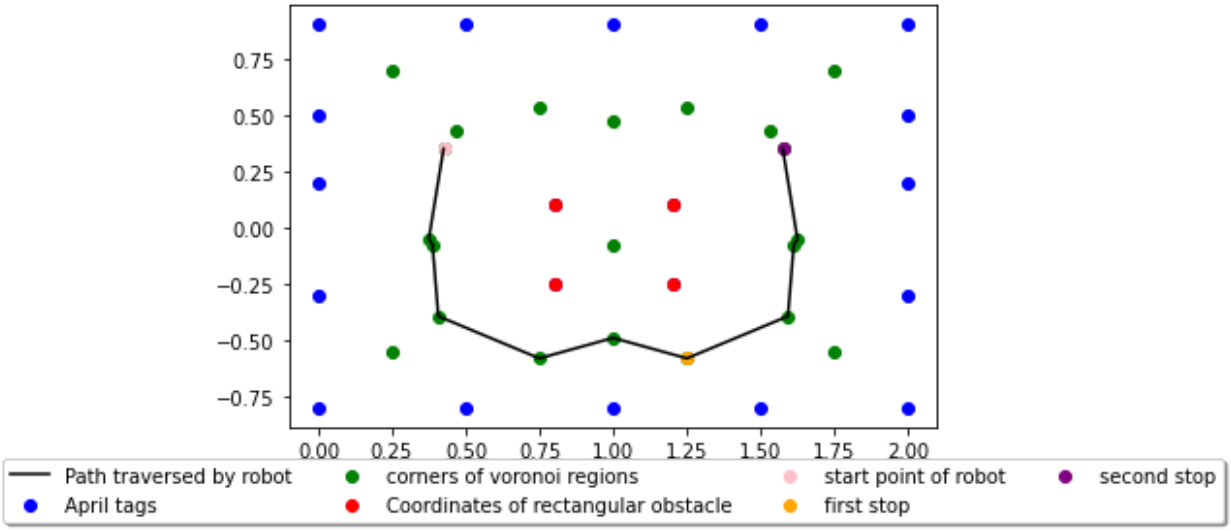
The code for the above algorithm (to get the waypoints) is given in a jupyter notebook, submitted along with the report. The video of our robot moving from 1 waypoint to another is here: https://youtu.be/VGR4au_6fKs

Figure 1: Voronoi Graph obtained:
X axis is the vertical axis, Y axis is on the bottom.



The blue points on the outside represent the boundary along the April tags and blue points in the middle represent the boundary of the obstacles. We then divided the entire movement plane of the robot into voronoi regions around the blue points. The orange dots denote the corners of the voronoi regions, which give us the voronoi graph. We remove the edges that go outside the map, i.e. the dotted edges. We remove the edges that go to the vertex in the middle, i.e. that pass through the obstacle.

Our final path can be found in **Figure 2 (x-axis is the vertical, y-axis is on the bottom): Distribution of graph vertices into start, stops, path points as given by BFS:**



Main Limitations:

1. Even though this algorithm guarantees 0 collisions, it may not give us the shortest path. For example: The shortest path between 2 points might be just the straight line connecting them if it does not pass through the obstacle. But in this algorithm, the robot will always follow the edges of the graph which can be a much longer path. We can check if the straight line passes through the obstacle or not and add the condition to only use the voronoi graph if the straight line pases via the obstacle. But even if it does, the voronoi graph path may not be the shortest path.
2. Also, this algorithm only works if we have the entire map information beforehand, including the obstacle boundaries. For a dynamically changing map, we will have to collect map information and rerun the algorithm everytime the map changes which will be very inefficient. A randomized path selection algorithm might be better in that sense that optimizes on the immediate next waypoint towards the goal without collisions such as RRT.