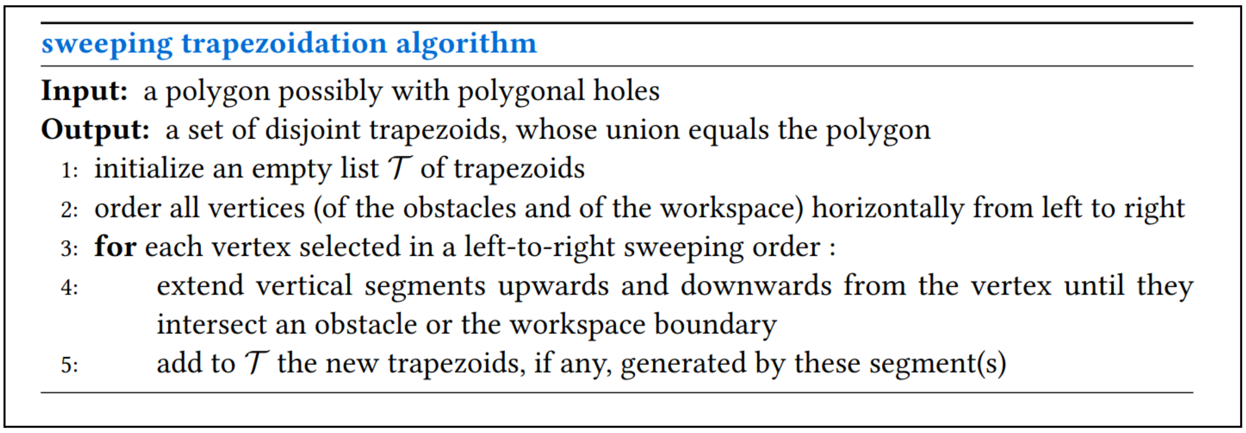
Name: Chinmay Makarand Pimpalkhare
Roll Number: 200100115
Group 3
SC627 - ROS Assignment 2
Date of Submission: 23 March 2023

The simulations mentioned in this report were done by the following people
APF : Harsh Raj Chaudhari
Voronoi: Harsh Raj Chaudhari
Trapezoidal Cell Decomposition: Chinmay Pimpalkhare

Trapezoidal Cell Decomposition Algorithm

In our code we are making use of an offline algorithm, which first constructs the cell decomposition using the sweeping trapezoidal algorithm.



sweeping trapezoidation algorithm

Input: a polygon possibly with polygonal holes
Output: a set of disjoint trapezoids, whose union equals the polygon

```
1: initialize an empty list $\mathcal{T}$ of trapezoids
2: order all vertices (of the obstacles and of the workspace) horizontally from left to right
3: for each vertex selected in a left-to-right sweeping order :
4:     extend vertical segments upwards and downwards from the vertex until they
       intersect an obstacle or the workspace boundary
5:     add to $\mathcal{T}$ the new trapezoids, if any, generated by these segment(s)
```

The given pseudocode has been taken from [1]

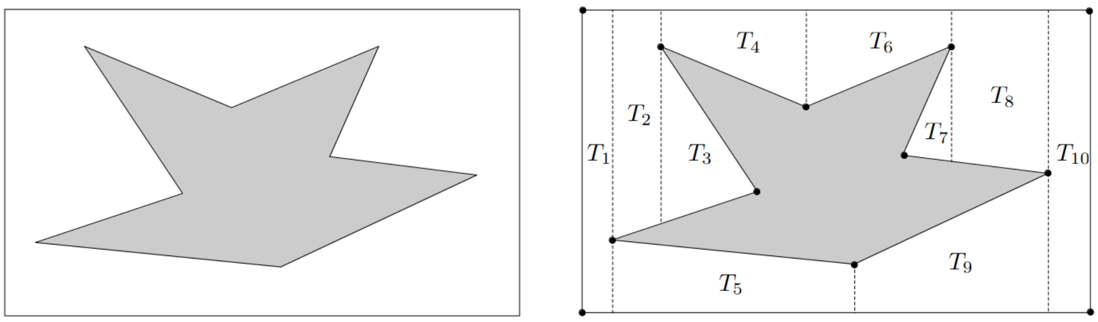


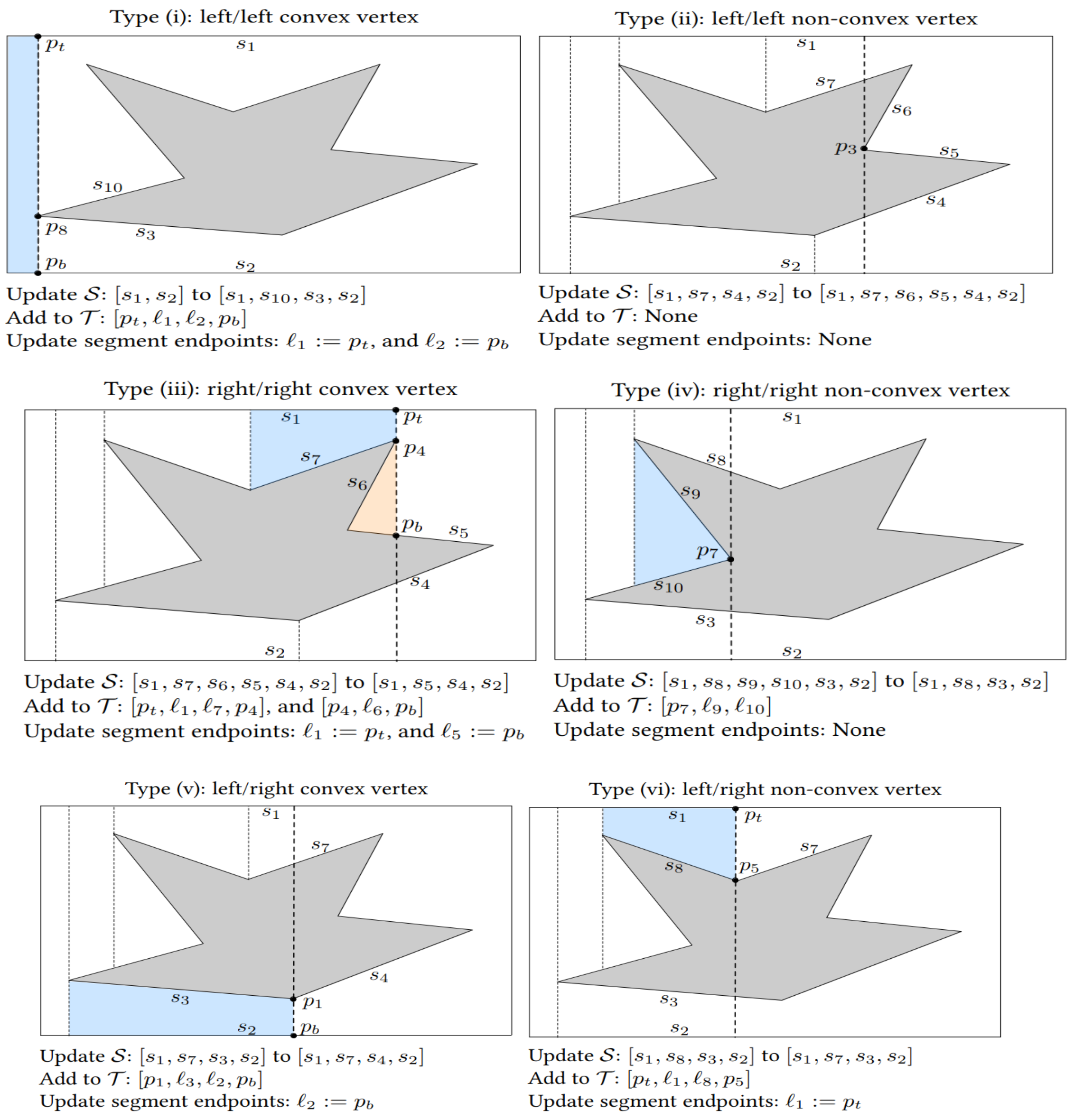
Figure 2.5: A trapezoidation of a workspace into trapezoids $T_1, \ldots, T_{10}$, computed by the sweeping trapezoidation algorithm

This is how a given workspace can be divided into disjoint trapezoids using the algorithm.

Vertex Type	Vertex as Endpoint of Two Segments	Vertex as Convex or Non-Convex	Example
(i)	left/left	convex	p_6 and p_8
(ii)	left/left	non-convex	p_3
(iii)	right/right	convex	p_2 and p_4
(iv)	right/right	non-convex	p_7
(v)	left/right	convex	p_1
(vi)	left/right	non-convex	p_5

Table 2.1: The six vertex types encountered during the trapezoidal decomposition algorithm

Different kinds of points need to be treated differently. It depends on the nature of the convexity or concavity at those points.



Update $\mathcal{S}$: $[s_1, s_2]$ to $[s_1, s_{10}, s_3, s_2]$
Add to $\mathcal{T}$: $[p_t, \ell_1, \ell_2, p_b]$
Update segment endpoints: $\ell_1 := p_t$, and $\ell_2 := p_b$

Update $\mathcal{S}$: $[s_1, s_7, s_4, s_2]$ to $[s_1, s_7, s_6, s_5, s_4, s_2]$
Add to $\mathcal{T}$: None
Update segment endpoints: None

Update $\mathcal{S}$: $[s_1, s_7, s_6, s_5, s_4, s_2]$ to $[s_1, s_5, s_4, s_2]$
Add to $\mathcal{T}$: $[p_t, \ell_1, \ell_7, p_4]$, and $[p_4, \ell_6, p_b]$
Update segment endpoints: $\ell_1 := p_t$, and $\ell_5 := p_b$

Update $\mathcal{S}$: $[s_1, s_8, s_9, s_{10}, s_3, s_2]$ to $[s_1, s_8, s_3, s_2]$
Add to $\mathcal{T}$: $[p_7, \ell_9, \ell_{10}]$
Update segment endpoints: None

Update $\mathcal{S}$: $[s_1, s_7, s_3, s_2]$ to $[s_1, s_7, s_4, s_2]$
Add to $\mathcal{T}$: $[p_1, \ell_3, \ell_2, p_b]$
Update segment endpoints: $\ell_2 := p_b$

Update $\mathcal{S}$: $[s_1, s_8, s_3, s_2]$ to $[s_1, s_7, s_3, s_2]$
Add to $\mathcal{T}$: $[p_t, \ell_1, \ell_8, p_5]$
Update segment endpoints: $\ell_1 := p_t$

Figure 2.7: Classification of the six types of vertices. For each vertex type, the figure illustrates the actions performed in Step 4: and Step 5: to update list of trapezoids trapezoids $\mathcal{T}$, the segment list $\mathcal{S}$, and the segment endpoints.

This is how the algorithm proceeds[1].
Once this is done, we need a roadmap algorithm to connect the various points along which the robot will traverse.

---

roadmap-from-decomposition algorithm

Input: the trapezoidation of a polygon (possibly with holes)
Output: a roadmap

```
1: label the center of each trapezoid with the symbol ◇
2: label the midpoint of each vertical separating segment with the symbol ●
3: for each trapezoid :
4:         connect the center to all the midpoints in the trapezoid
5: return the roadmap consisting of centers and connections between them through
   midpoints
```

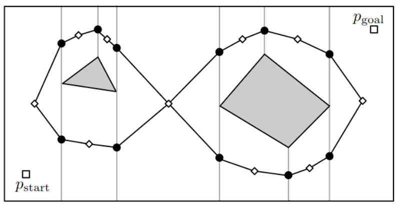


Figure 2.8: An example roadmap for a free workspace

This is the pseudocode for the roadmap creation algorithm. This will create a graph and now we must traverse on this graph.

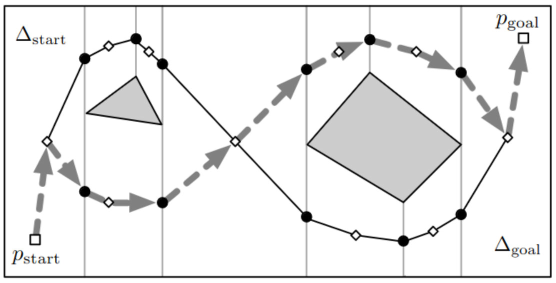


Figure 2.9: Illustration of the planning-via-decomposition+search algorithm

planning-via-decomposition+search algorithm

Input: free workspace W_{free}, start point p_{start} and goal point p_{goal}
Output: a path from p_{start} to p_{goal} if it exists, otherwise a failure notice. Either outcome is obtained in finite time.

```
1: compute a decomposition of $W_{\text{free}}$ and the corresponding roadmap
2: in the decomposition, find the start trapezoid $\Delta_{\text{start}}$ containing $p_{\text{start}}$ and the goal trape-
   zoid $\Delta_{\text{goal}}$ containing $p_{\text{goal}}$
3: in the roadmap, search for a path from $\Delta_{\text{start}}$ to $\Delta_{\text{goal}}$
4: if no path exists from $\Delta_{\text{start}}$ to $\Delta_{\text{goal}}$ :
5:         return failure notice
6: else
7:         return path by concatenating:
                 the segment from $p_{\text{start}}$ to the center of $\Delta_{\text{start}}$,
                 the path from the $\Delta_{\text{start}}$ to $\Delta_{\text{goal}}$, and
                 the segment from the center of $\Delta_{\text{goal}}$ to $p_{\text{goal}}$.
```

This is the pseudocode of the planning-via-decomposition+search algorithm. This is the final part where we traverse through the graph.

```
1:  begin with the start node and mark as visited    // The start node forms Layer 0
2:  for each unvisited neighbor $u$ of the start node :
3:      mark $u$ as visited, and set the start node as the parent of $u$.    // The nodes
        $u$ form Layer 1
4:  for each unvisited neighbor $v$ of the nodes in Layer 1 :
5:      mark $v$ as visited and record the neighbor from Layer 1 as the parent of $v$
6:  repeat the process until you reach a layer that has no unvisited neighbors
7:  if the goal node has been visited :
8:      follow the parent values back to the start node, and return this sequence of
        vertices as the shortest path from start to goal
9:  else
10:     return a failure notice (i.e., the start and goal node are not path-connected)
```

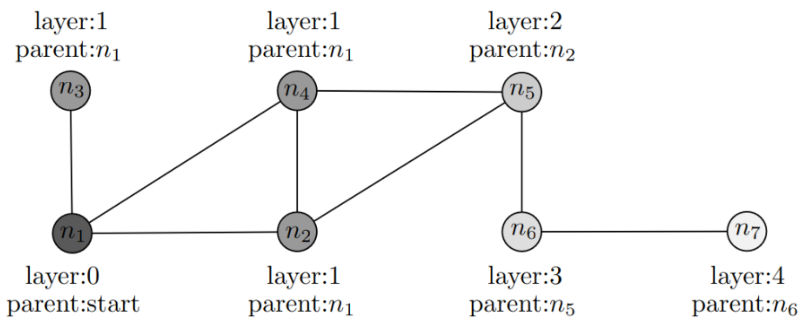
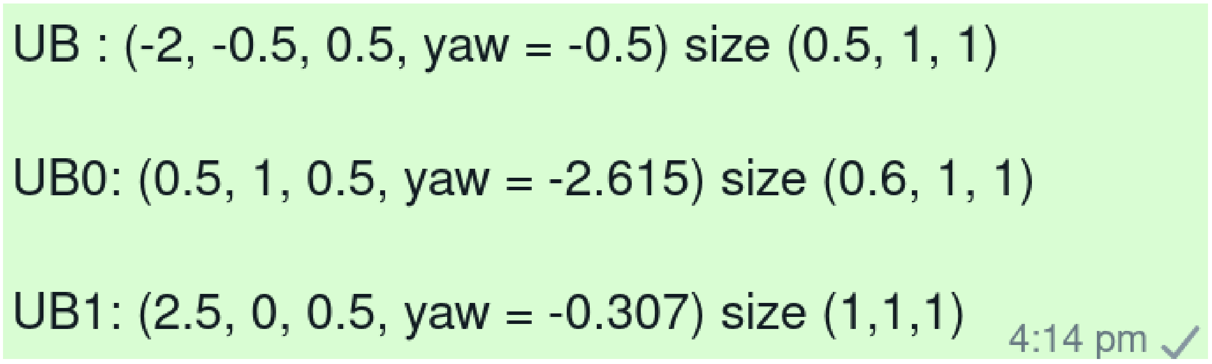


Figure 2.11: A sample execution of the breadth-first search strategy (in an unweighted graph). Each vertex is labeled with (and shaded according to) its layer, which turns out to be equal to the distance between that node and the start node, and with the identity of its parent node.

This is an example pseudocode for the BFS algorithm which we have implemented.

Now, we shall come to the running of our code.

First of all, we have been given information about the centres of the obstacles in Gazebo. We also know the orientations. In the case of hardware, this data is obtained from vicon



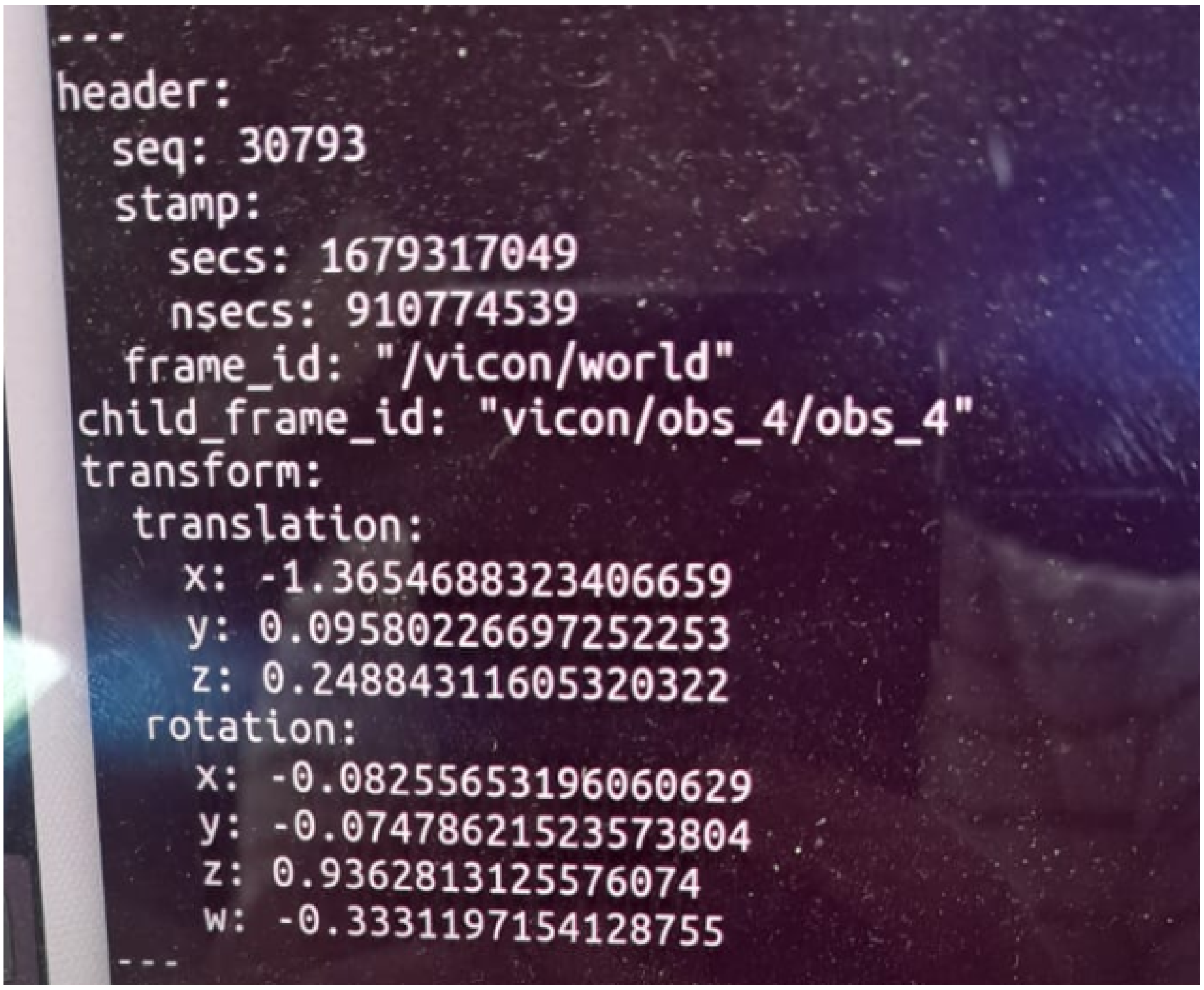
UB : (-2, -0.5, 0.5, yaw = -0.5) size (0.5, 1, 1)

UB0: (0.5, 1, 0.5, yaw = -2.615) size (0.6, 1, 1)

UB1: (2.5, 0, 0.5, yaw = -0.307) size (1,1,1)

4:14 pm ✓

This is the data about the obstacles for the gazebo sims. In case of vicon, an example is something like this.



```
---
header:
  seq: 30793
  stamp:
    secs: 1679317049
    nsecs: 910774539
  frame_id: "/vicon/world"
child_frame_id: "vicon/obs_4/obs_4"
transform:
  translation:
    x: -1.3654688323406659
    y: 0.0958022669725225З
    z: 0.2488431160532032
  rotation:
    x: -0.0825565319606062
    y: -0.0747862152357380
    z: 0.936281312557607
    w: -0.333119715412875
---
```

We can use the translation and the rotation data for our work.
Now next we have to use this data to obtain the actual vertices.
I have written a script in Excel for that, which computes the vertices once the following parameters are known.

Center x	0.43
Center y	-0.47
Yaw	0.7853981634
Beta	0.7853981634
Dimension x	0.2
Dimension y	0.2
Dimension z	0.2828427125
Scale	1.5

After this data is known, we can use geometry and find out the four vertices. Here beta is the tan inverse of y/x and returns the angle made by the diagonal of the rectangle with the edge of the rectangle. **Further, since we have a robot which is not a point but occupies finite nonzero space, we need to take this size into account while we consider the free configuration space. Hence we have introduced a factor called scale which**

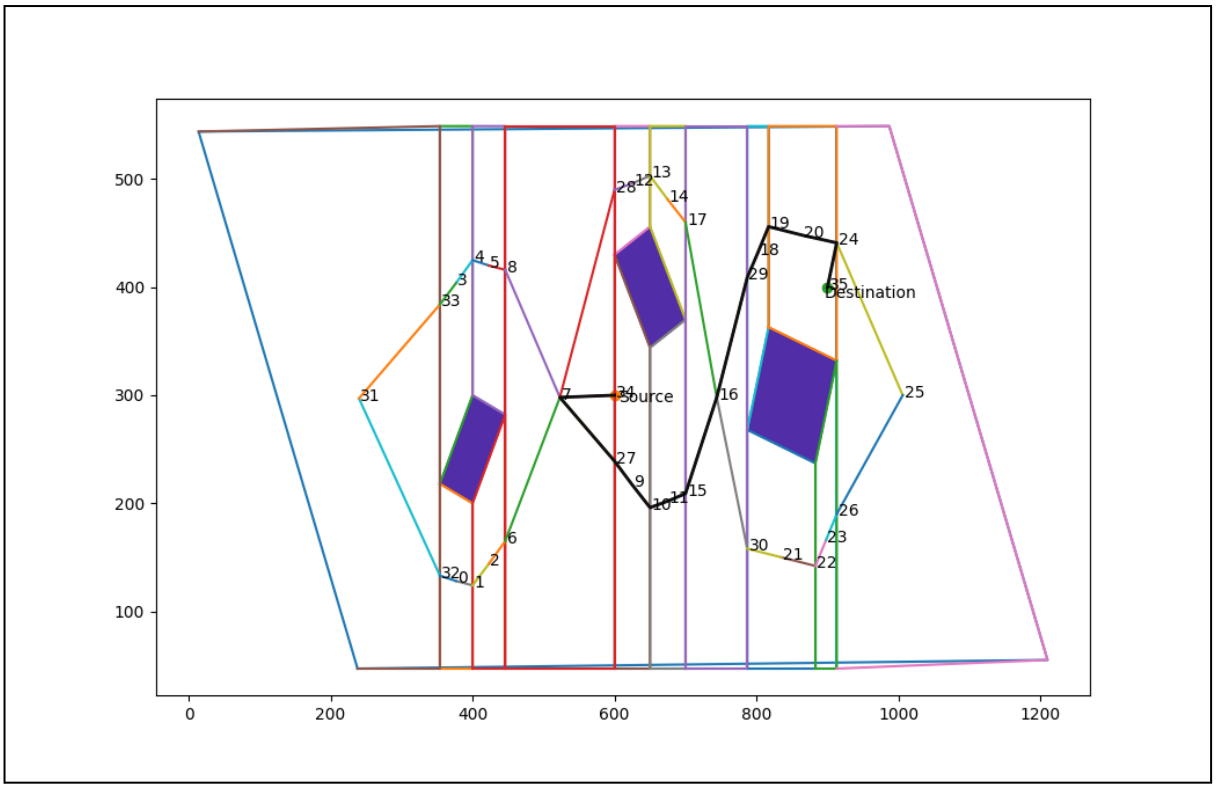
overestimates the size of the obstacle, which stops the robot from going too close to the obstacle.
The output of the vertex locations will be something like this:

Left Bottom x	0.43	**Left Bottom y**	-0.6821320344
Right Bottom x	0.6421320344	**Right Bottom y**	-0.47
Right Top x	0.43	**Right Top y**	-0.2578679656
Left Top x	0.2178679656	**Left Top y**	-0.47

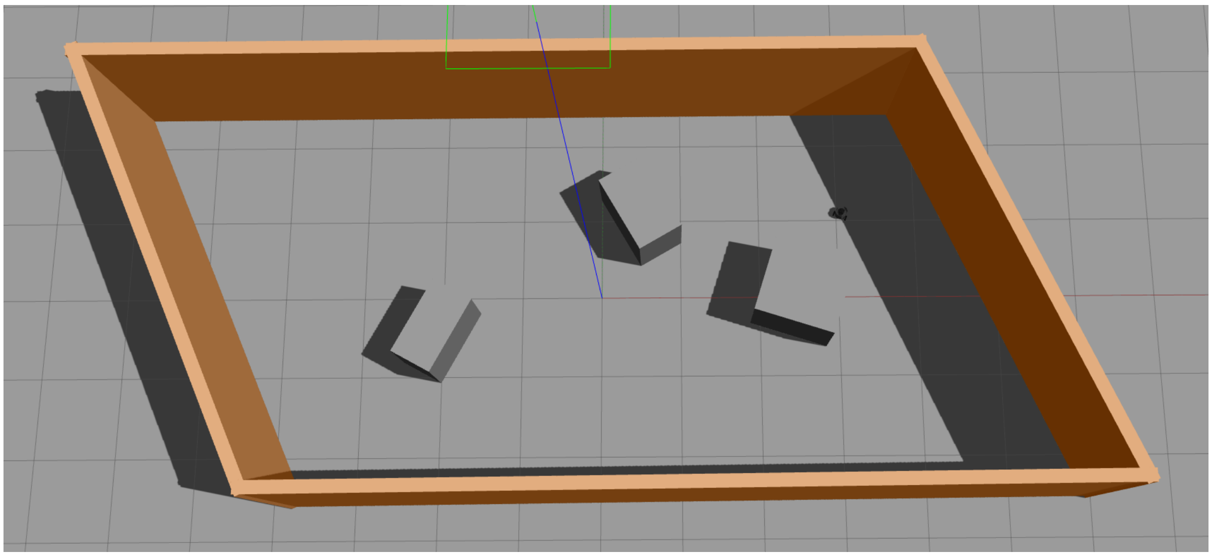
These coordinates can then be input into the cell decomposition algorithm, which computes the sweeping trapezoidal decomposition.
Now, after processing the values a bit, we pass them into the function for cell decomposition.

ROS Coordinates / Vicon Coordinates			**Converted** **Apply the transform X = 100x + 600, Y = 100y + 300**	
Wall				
-3.619	-2.524		238.1	47.6
6.107	-2.448		1210.7	55.2
3.873	2.486		987.3	548.6
-5.853	2.448		14.7	544.8
Unit Box				
-2.46	-0.82		354	218
-2	-1		400	200
-1.54	-0.18		446	282
-2	0		400	300
Unit Box0				
0.5	0.44		650	344
1	0.7		700	370
0.5	1.56		650	456
0	1.3		600	430
Unit Box1				
1.87	-0.32		787	268
2.83	-0.63		883	237
3.13	0.32		913	332
2.17	0.63		817	363

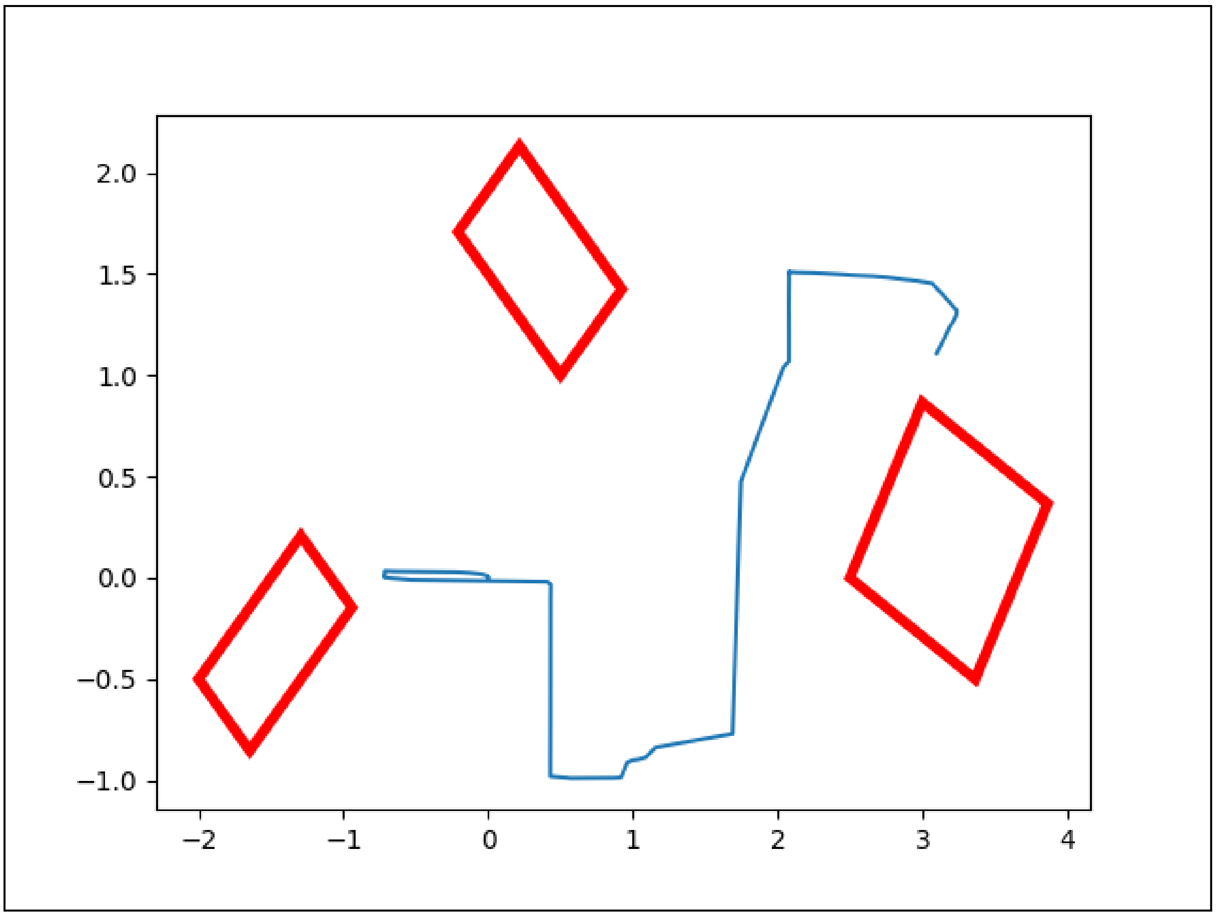
Now, we can input these coordinates into the code for vcd. This gives us the following output.



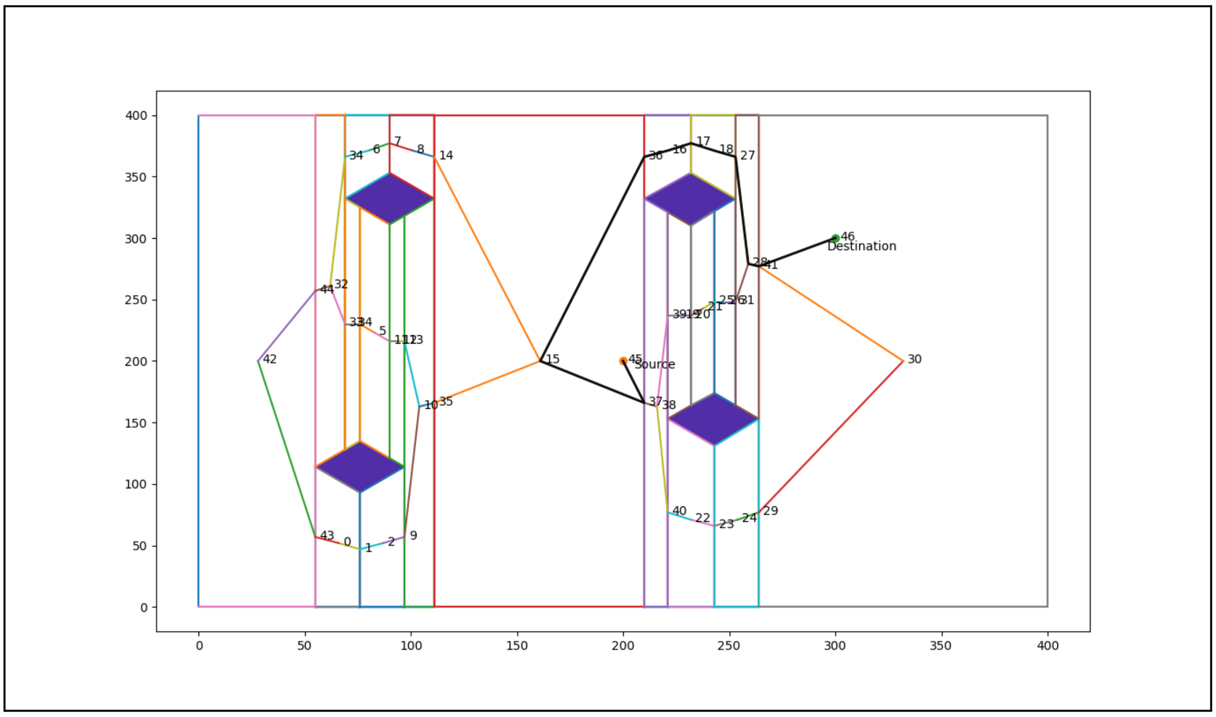
The obstacles can be seen. The black path is the path from the start to the goal locations.



If we then implement the code, we see that the robot has successfully reached its final location. We have a made a plot of the odometry data and it looks like this



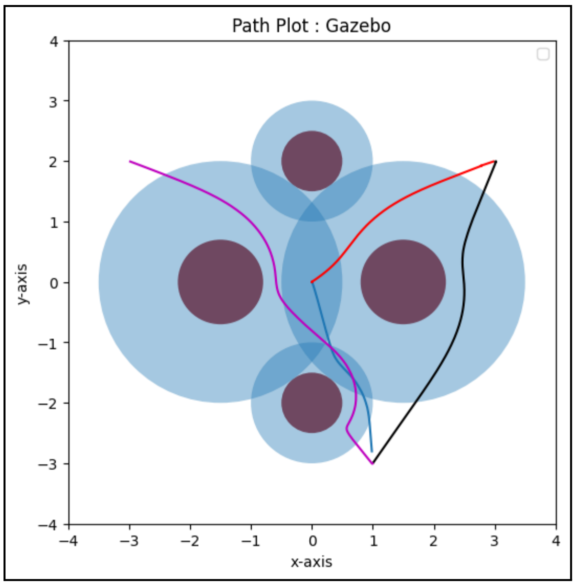
Thus the robot is able to move without colliding with obstacles.
Similarly, we had generated the path for the hardware implementation using the offline roadmap generation. Unfortunately, the odometry data for the hardware demo is unavailable.

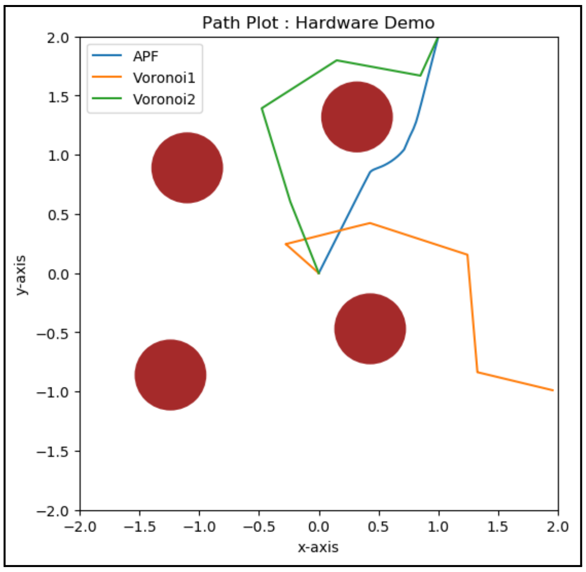


In this case, four obstacles were placed. Hurray, our code has worked!

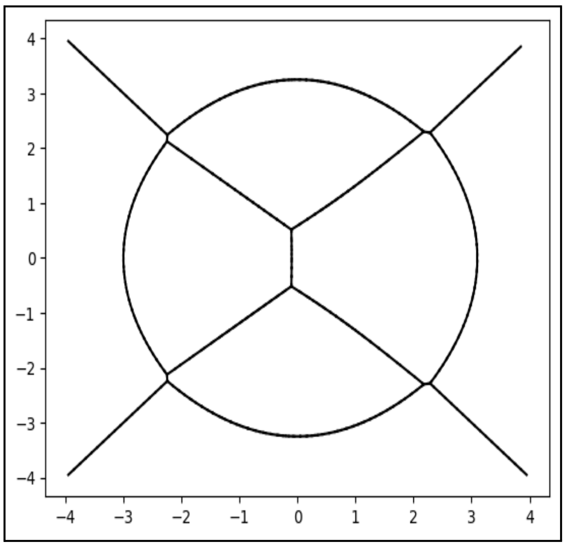
APF and Voronoi Algorithms

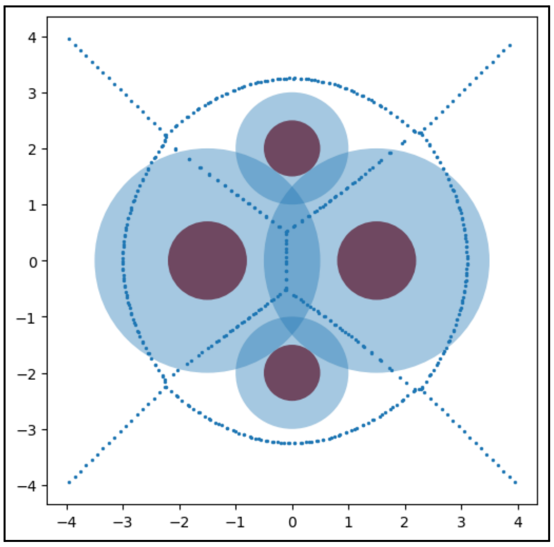
(*Since this part was mainly done by Harsh, I have focused more on the Trapezoidal algorithm, which was done by me)*





Here, we can see the simulation plots for the Gazebo and the Hardware simulations. A comparison of the three methods can be seen. In Voronoi, the path is being decided first and then the robot is traversing through that path, while for the APF algorithm, only the potential function is being defined and then the robot moves along to the gradients of that potential function. As we can see in the image above the APF path goes close to the top obstacle.





These are the images that show the Voronoi Graph Decomposition

---

How to run the code (README for the code)

1. First we have to find the coordinates of the obstacle, which can be done using the google sheet or the excel sheet provided. After that the transformations x = x*100 + 600 and y = y*100 + 300 are done to bring all values to positive and integers. The origin shift is 300, 600 is tuned as per the gazebo wall definition.

2. After that is done, we enter these into the file ‘if.txt’. The first line contains the vertices of the bounding box (walls) in an anticlockwise order. The last line contains the start location and the goal location. Since whenever we bringup the robot, its location position becomes (0,0), **please set the start position to (600, 300).** This is as per the transformation given. Now run the vertical_cell_decomposition.py file
3. After that we will get the output plot of the path. We can get the waypoints by moving the cursor over these points. Again we have to compute the inverse transformation to get the points, which can be done in the given excel sheet.
4. Now we go to the file trapezoid_chi1.py. Here there are some point definitions at the start of the code. We enter the coordinates for these points and set the variable graph_len to (1 + the number of points entered). Also make sure to uncomment the graph.append parts.

Comparison of the algorithms

- APF can be used in case of dynamic obstacles while the other two cannot be. This can be done by specifying the movement of the obstacles as a function of time. This is not possible in Voronoi because the update of the cell boundaries and the roadmap will be computationally very expensive.
- APF suffers from the issue of local minima. This issue is not faced in case of Voronoi or Trapezoidal decomposition.
- For APF, we need the objects to be circular, otherwise the definition of the boxes is approximate, and not completely accurate.
- If the number of obstacles is very high, cell decomposition will break the domain into a very large number of parts. It is difficult to comment on whether that is good or not.

References

[1] Lectures on Robot Planning and Kinematics, Fransesco Bullo