**PROJECT REPORT**

**ROBOT MOTION PLANNING**

By Aditi Mishra

800917773

**Project Proposed** : Originally proposed to implement a Voronoi Roadmap based motion planner for a non-holonomic 2D robot in an environment with 2D polygonal obstacles but changed it to an RRT based motion planner for a holonomic path traced by a 2D robot which is capable of translation in an environment with 2D polygonal obstacles.

**Algorithm Implemented for Motion Planning** :

**Algorithm** BuildRRT

Input: Initial configuration *qinit*, number of vertices in RRT *K*, incremental distance *Δq*)

Output: RRT graph *G*

*G*.init(*qinit*)

**for** *k* = 1 **to** *K*

*qrand* ← RAND\_CONF()

*qnear* ← NEAREST\_VERTEX(*qrand*, *G*)

*qnew* ← NEW\_CONF(*qnear*, *qrand*, *Δq*)

*G*.add\_vertex(*qnew*)

*G*.add\_edge(*qnear*, *qnew*)

**return** *G*

This is the Rapidly-Exploring random tree algorithm proposed by Steven M. LaVelle. The data structure was of a tree to store the nodes implemented via dictionary in Python.

**Algorithm for Collision Detection / Avoidance:**

The motion planning algorithm allows for the robot configuration to lie anywhere in the workspace however due to the presence of obstacles, there are certain configurations it needs to avoid. I used two packages in python to help create the function required for that purpose –Pygame and Shapely.

1. Pygame

I used Pygame to create polygonal obstacles using pygame.draw.rect and pygame.draw.polygon and then to check if a configuration of the 2D robot lies inside these obstacles, I checked if the new node to be added to the tree (corresponding to the top left vertex of the robot) lies inside these polygons. This was done using pygame using polygon.collidepoint(newnode) .

But this condition gave nodes which were being generated on the border of the obstacle hence would result in a collision eventually. So I needed another check to avoid collisions better.

1. Shapely

Shapely.geometry provides us shapely.geometry.point and shapely.geometry.polygon modules which define Point and Polygon as classes. Using these modules and special functions such as finding exterior coordinates of the Polygon or passing the vertices of the new node generated using Point to do basic distance calculations easier.

The second condition for collision detection was to check if the new node (or topleft vertex of robot configuration) is at least at a clearance of 20 units from the entire polygonal obstacle. This is done by the following algorithm,

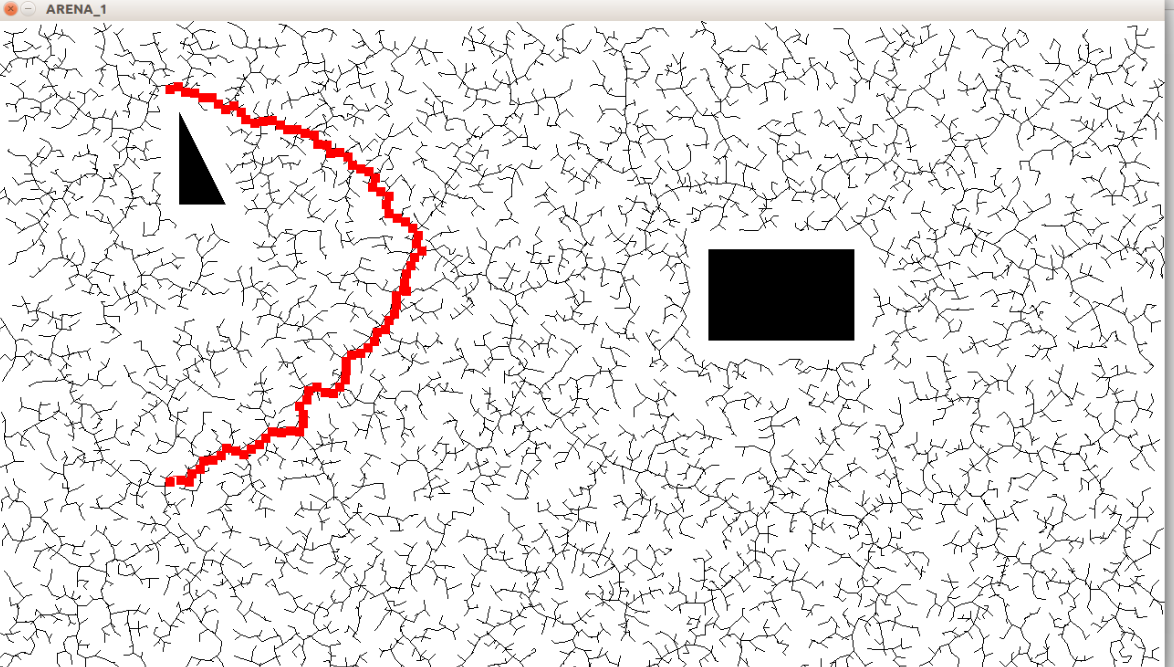
* Pairs from vertices of the obstacles are generated
* Using these paired vertices, the edge length is found and then intersection point is computed using slopes of the line segments formed .
* The base of the triangle formed by the point and ttwo vertices of the edge (distance between intersection point and actual point) is found which is checked for the clearance from obstacle property .

Condition : if distt < 20

1. Once these two conditions are true then the newnode will be appended and collision detection for that configuration has been successful.

Few corner cases were discussed in the demonstration. I simulated those problems to show how the 2D robot configuration avoids obstacles.

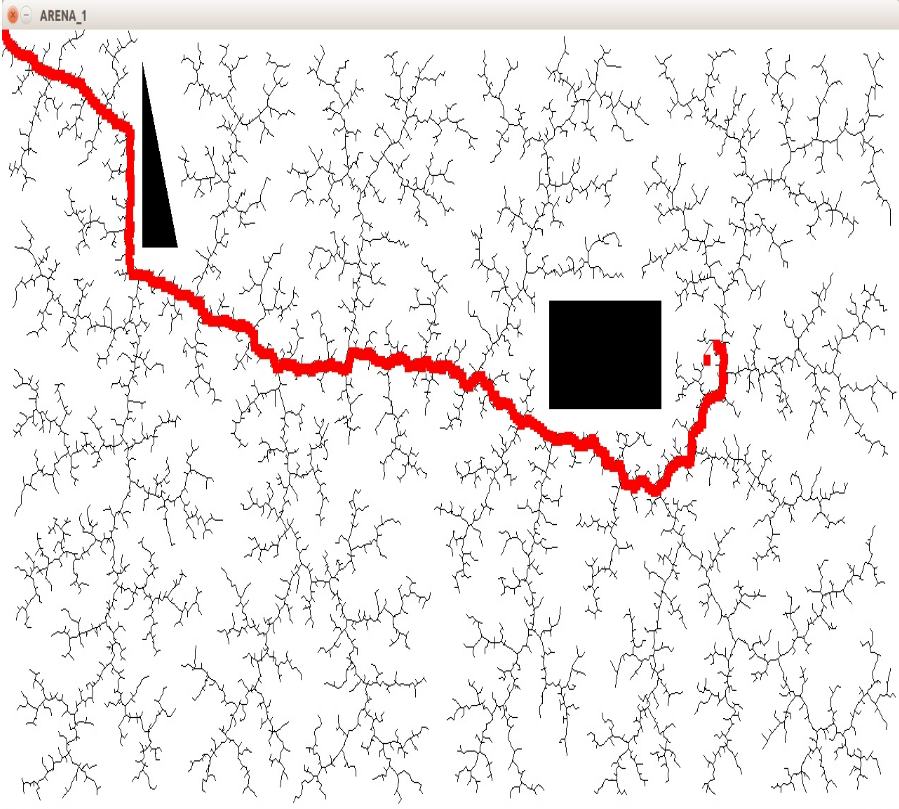
* One case was when the bot was half inside the obstacle with its newnode (or reference point ) being outside the obstacle. The simulation ran and exited . This was because the reference point was still inside the 20 units clearance hence it did not take that configuration as a correct one and exited the program.
* Another case was how this reference point calculates distance between itself and the polygon when its in the clearance limit by the X axis but actually outside it due to the Y coordinate .(this was due to the question raised about projecting distance from the point to the polygon ) . I found it doesn’t generate any nodes in the clearance region but takes a path around the obstacle. Also the projection of the vertex of the triangle upwards doesn’t come into picture when it tries to calculate the clearance since there is no intersection point. A snapshot is shown below.



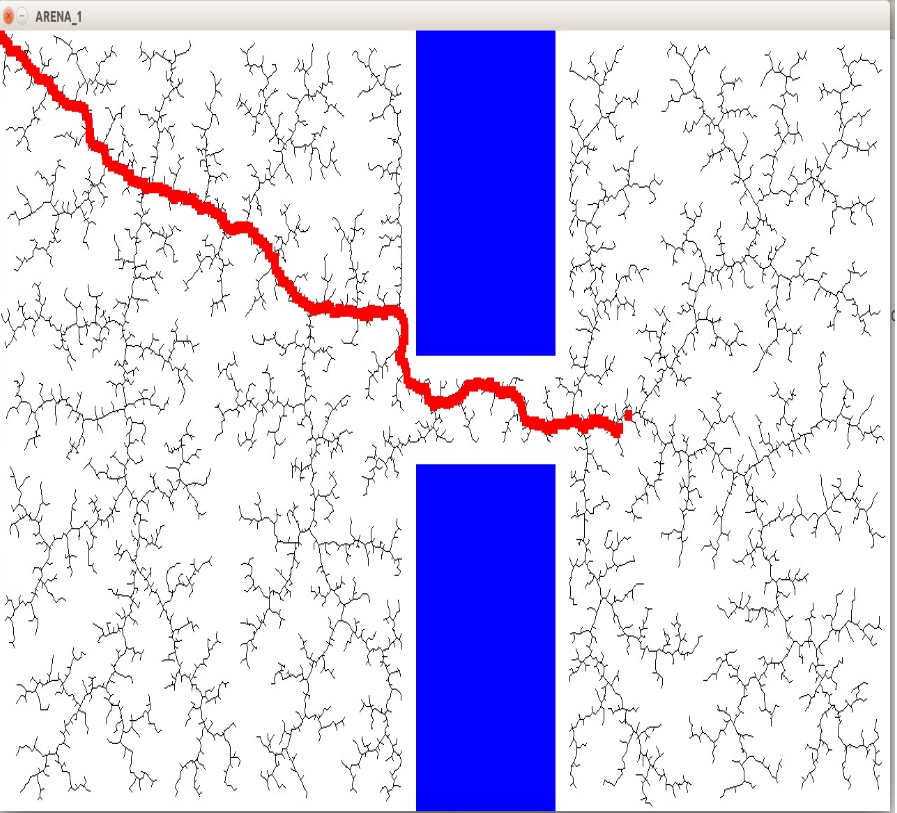
**Results :**

Three arenas were designed to test the motion planner. The following outputs were observed.

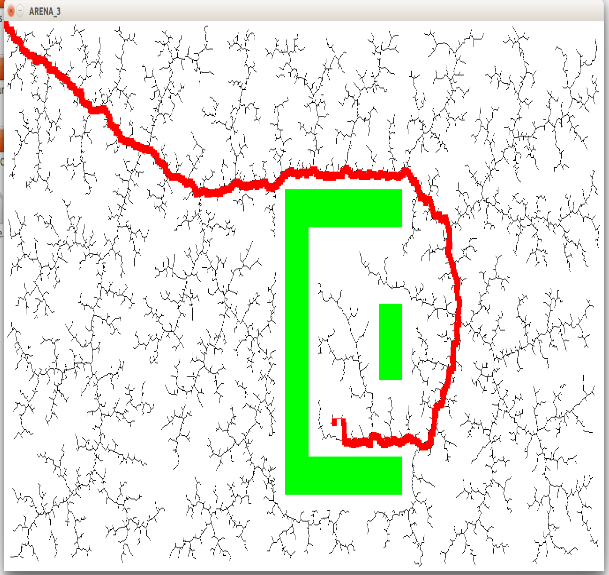
1.Basic Polygonal Obstacles



2. River Crossing Scenario



3.Parking Lot Structure



**Expectations, Accomplishments and Challenges:**

* The main expectation of the project was that a collision free path is found in the number of nodes specified and the robot successfully moves along it. This was accomplished.
* Collision with obstacles were avoided due to two features in the code without using any external package :

1. A clearance distance of 20 units was maintained between the node and the obstacle such that any node which is less than 20 units away from the obstacle is not appended to the tree and the tree doesn’t extend in that direction.
2. Instead of checking the clearance distance condition for all random nodes sampled and generated for then current node, we only check the distance condition for the node selected out of all the random nodes generated for that step.

* Due to the random nature of the tree expanding, we can see that the path found is not the shortest possible path in the workspace.
* I was short on time hence I was not able to implement non holonomic motion or path coordination as I had planned.
* Major Hurdle and Further Ideas: I tried to do collision detection using a mix of RRT and Artificial Potential field ideas such that there will be high repulsive force around the obstacle and the tree will avoid the obstacle that way. Later on, I realized it was based on the distance property itself and comparison of distance with a clearance (assuming some kind of a boundary around the obstacle-idea from Minkowski Sum) and thus used distance for collision avoidance.

This may be used for an algorithm based on both existing algorithms since it will avoid the local minima (no attractive force to have net force as zero) but again this force should be calculated for only the chosen random node and not all random nodes generated.

References :

1. Shapely user manual
2. Pygame user manual
3. Rapidly Exploring Random Trees :A New Tool for Path Planning , Steven M. LaVelle
4. Wikipedia.com