ITCS 6152 - Robot Motion Planning Final Project

Prasanth Reddy Pallu
ppallu@uncc.edu
800886181

Problem Statement :

To guide a car like robot from initial configuration to goal configuration in a complex geometrical world.

Introduction:

The main goal of this report is to through light on work I have done for my final project in Robot Motion Planning class. The problem I am tackling in this project is to guide a guide a car like robot from given initial configuration to goal configuration. The environment I considered is a a 2-D environment with 2-D obstacles. I have done all this in Python language. I have used pygame library for visual representation of the computations.

Where I started:

Initially I started with a point like robot. Using Rapidly Exploring Random Tree (RRT) algorithm I am able to find the shortest path between a given goal point from initial point. I stored the predecessors of each new node and the distance between the two in a dictionary. In this way by finding the predecessor of goal node and then predecessor of that node and so on, I am able to go to the start node. If the end node is not present in the node list, it is appended to it.

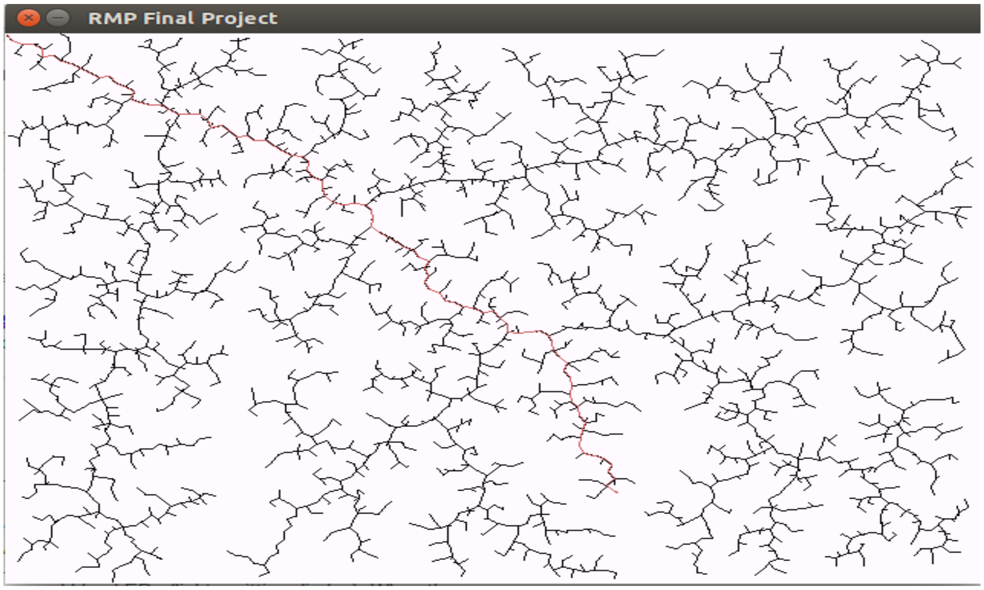
Basic RRT Algorithm used :

```
Input : Initial point(q_init), Number of nodes(n), incremental distance(e)
Output : RRT graph G

G. add(initial_point)
for k =1 to k= n
    q_rand ← random_config();
    EXTEND(G,q_rand);
return G

EXTEND(G,q)
    q_near ← NEAREST_NEIGHBOUR(q,T);
    if NEW.CONFIG(q,q_near,q_new,u_new) then
        G.add_vertex(q_new);
        G.add_edge(q_near,q_new,u_new)
        if q_new = q then
            return Reached;
        else
            return Advanced;
    return trapped;
```

The screen-shot shows the initial output:



Steps I have gone through:

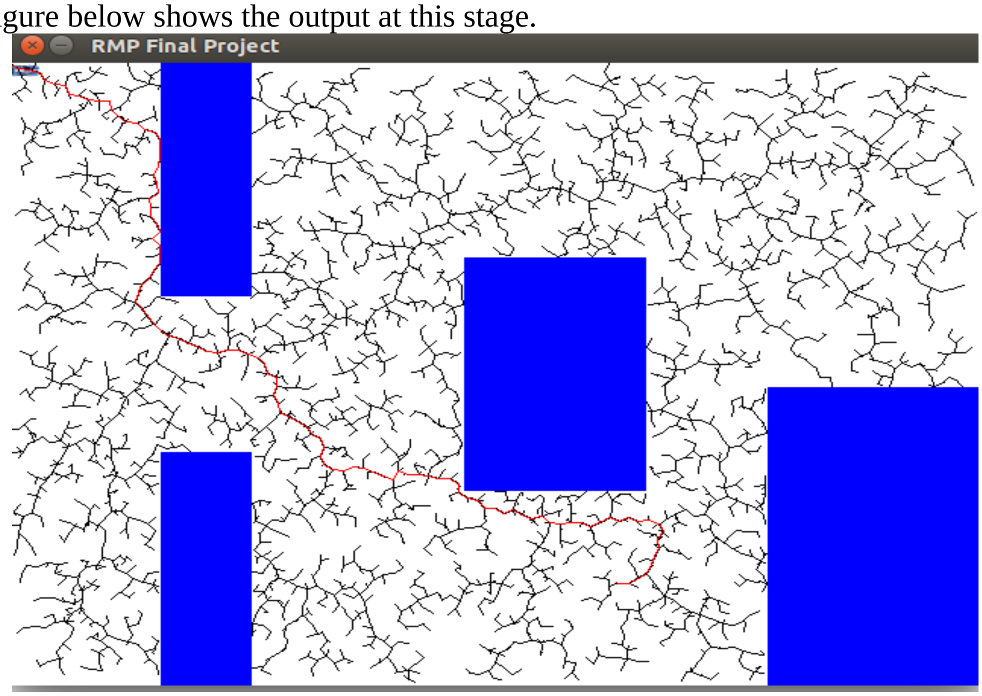
once I have the initial RRT working I inserted obstacles using sprites in pygame. Obstacles are a bunch of rectangles which are inserted in desired positions. Here I have done ray- casting method to check whether generated new node falls inside the obstacle region. If the point is inside the obstacle region I am not adding it to the node list.

Ray casting method :

Input : Point, Polygon

Output : True or False that the point is inside.

The figure below shows the output at this stage.



Non-holonomic part:

Now I have to model this for car-like robot. Car like robot are considered as non-holonomic because they are subject to non-integrable constraints like velocity. For implementing non-holonomic motion I have choosed dubin's car which can move only in forward direction and has a constant steering angle of 45 degrees. Car like robots cannot rotate at a single point, they will take only curved paths because steering angle is restricted between -45 degrees to + 45 degrees. For this part I have used Euler integration method to obtain curve-linear paths. For this step I have to insert theta dimension to each node to specify the configuration of the robot.

For a given point (x,y,theta), euler integration is done using :

dx = x + h*cos(theta)

dy = y + h*sin(theta)

dtheta = theta + (v/L)*tan(steering angle)

h = step size

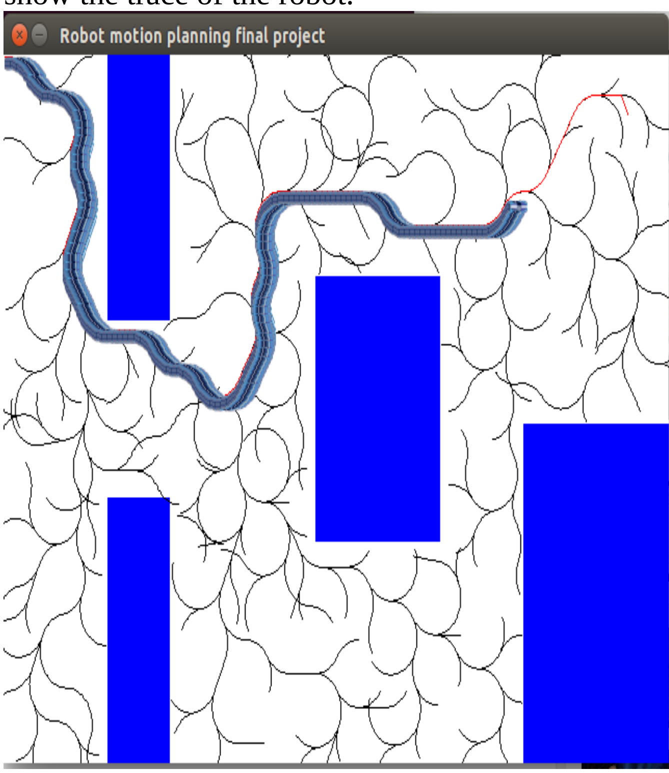
v/L = 0.05 (Which I considered)

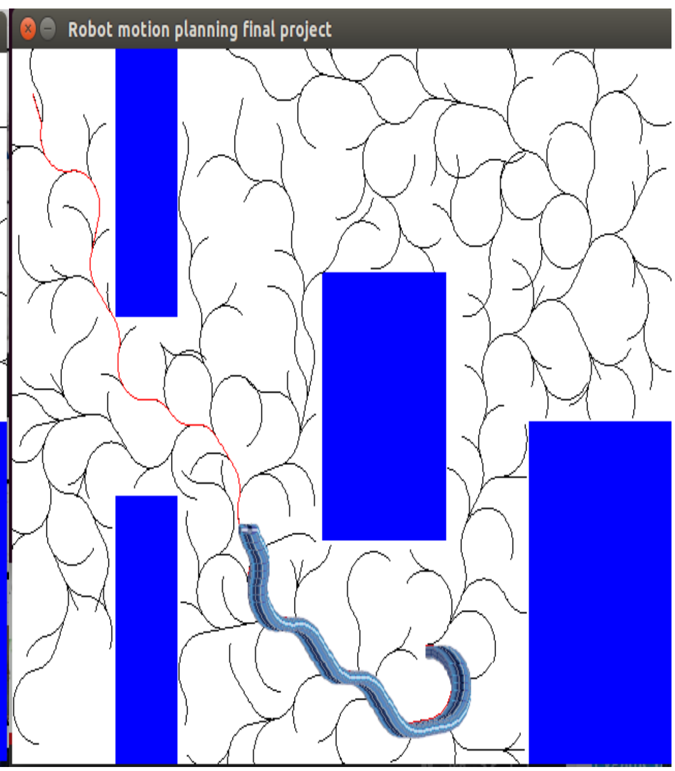
Steering angle = 45 or 0 or -45

Once the curved path is obtained they are inter-linked by the predecessors in a dictionary.

What I accomplished :

I am able to move a dubin's car lke robot is a non-holonomic path. The screen-shots below show the trace of the robot.





Challenges and what I have not done:

1. I am able to implement non-holonomic motion of a dubin's car .But I am not able to graphically represent a dubin's car.
2. When the curved path comes closer to obstacles, a 2-D robot may collide, given more time I may be able to do that.
3. Currently I am not checking if the given start point and end point are inside the obstacle region, given more time I can do this.
4. I have not used any collision detection package, instead of which I am using ray-casting method, which gives correct results some times and some times my path goes closer to obstacles.

List of References:

1. https://en.wikipedia.org/wiki/Rapidly_exploring_random_tree
2. http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu/
3. http://pygame.org
4. https://en.wikipedia.org/wiki/Dijkstra's_algorithm
5. https://www.youtube.com/watch?v=vDFJouaOqNs
6. Resources in course website