ITCS 6152

ROBOT MOTION PLANNING

NON-HOLONOMIC MOTION PLANNING IN A COMPLEX ENVIRONMENT

Veerendranath.M 800902656
vmannepa@uncc.edu

Objective:

Implementation of Randomly-Exploring Random Tree (RRT) algorithm for a complex environment in presence of obstacles with collision detection in which a 2D Robot having 3 Degrees of Freedom (DOF) and having Non-Holonomic can traverse to goal position.

Motion planning:

Type of robot: Single 2D square robot of dimensions 20x20 having 3 DOF such as R^2 x S^1 configuration space .The robot can move in X-Y plane and rotate about Z-axis.

Type of Motion: Robot has Non-Holonomic constraints which meant by it can perform only restricted motion they are subject to nonintegrable equality involving velocity the following characteristics.

- It can move forward or backward with a constant velocity
- It can steer around with 45 degrees such that it can just take left or right turn and it can't move side-wards as in holonomic case.

Equation that explain this kind of motion similar to a simple car:

Equation of motion based on Rear wheels of the car:

-dx*sin(theta)+ dy*cos(theta) = 0

dx = s*cos(theta)

dy =s*sin(theta)

dtheta = s/L * tan(steer)

Equation of motion based on Front wheels of the car:

-dx*sin(theta)+ dy*cos(theta) = 0

dx = s*cos(steer)*cos(theta)

dy =s* cos(steer)*sin(theta)

dtheta = s/L * sin(steer)

Here S is the speed of the car and L is the distance between the front and rear wheels of the car and is it orientation with respect to z axis and it a positional value as q(x,y,theta). In this project I am considering speed as 1 s/L ratio as 0.06 and varied the ratio to get better results.

Runge- Kutta Method:

In this RRT computation to get smoother curves and to execute non-holonomic constraints we use 4th order Runge-Kutta method. This method is approximate to solve above ODE - (Ordinary Differential Equations). This method can be explained as follows: (source Wikipedia)

Let an ODE be specified as follows.

$$\dot{y} = f(t, y), \quad y(t_0) = y_0.$$

Here, *y* is an unknown function (scalar or vector) of time *t* which we would like to approximate; we are told that $\dot{y}$, the rate at which *y* changes, is a function of *t* and of *y* itself. At the initial time t_0 the corresponding *y*-value is y_0. The function *f* and the data t_0, y_0 are given.

Now pick a step-size *h*>0 and define

$$y_{n+1} = y_n + \tfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$
$$t_{n+1} = t_n + h$$

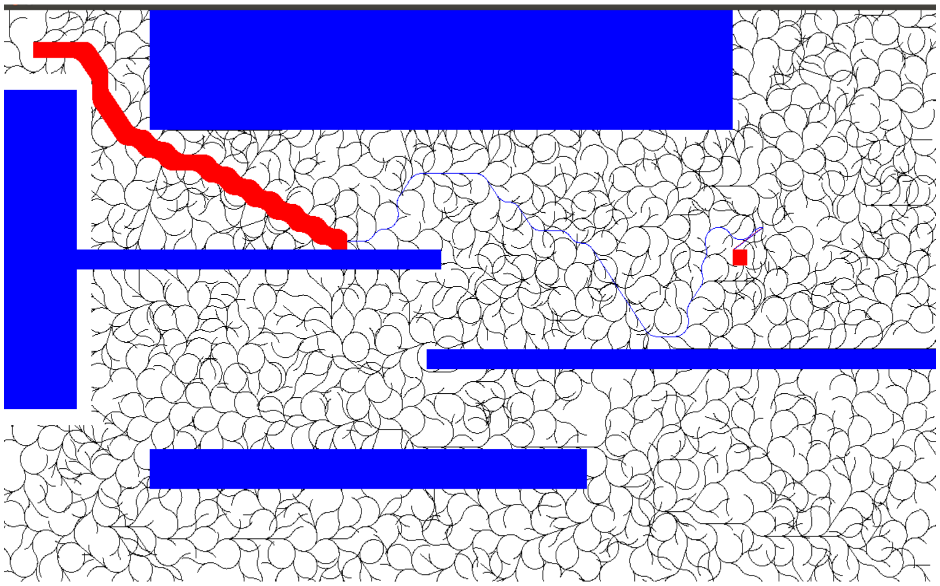
for *n* = 0, 1, 2, 3, . . . , using

$$k_1 = f(t_n, y_n),$$
$$k_2 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1),$$
$$k_3 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2),$$
$$k_4 = f(t_n + h, y_n + hk_3).$$

In our environment RRT uses the above method to draw curves to the random point that helps the robot to reach the goal with added constraints.

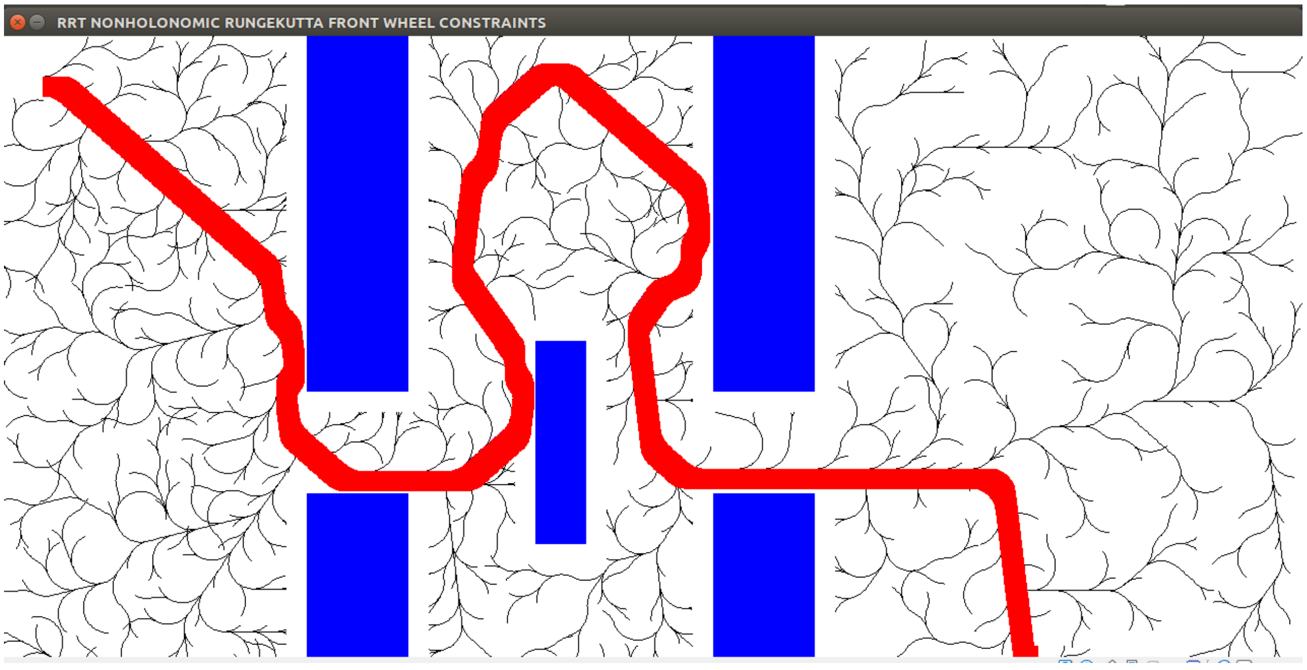
Type of Environment:

The map is a complex environment that has 2D polygonal obstacles of different sizes which is as follows:



The Environment above mainly consists of the Rectangular objects having various width and height. The main interesting part in this environment is while robot is placed at top left of the environment it has a very less space which is of the size of the robot to move between the two obstacles and it to develop a tree to reach the goal configuration.

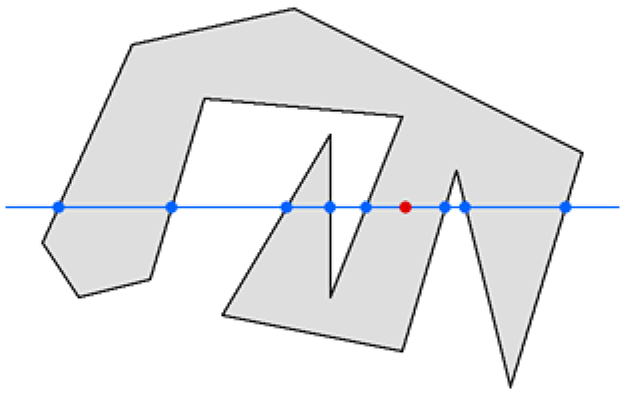
The below one is another interesting example in which the robot have to pass through narrow passes and reach its goal position.



Collision Detection:

Collision detection is one of the interesting part of the project such that the tree shouldn't traverse inside the obstacle and should determine a path to the goal position. The Environment uses **Ray Casting method** to detect the obstacles in the map. It works as follows:

```
count ← 0
for each side in polygon:
        if ray_intersects_segment(P,side) then
                count ← count + 1
if is_odd(count) then
        return inside
else
        return outside
```



In this Algorithm, it takes the point coordinates that we have to check and polygon vertices, it draws an imaginary line in a single direction from the point and stop drawing it when the line leaves the polygon. It will count the number of times the line crosses the polygon's boundary. If the count is an odd number the point must be inside. If it's an even number the point is outside the polygon. Finally the algorithm returns a bool value as True or False.

Algorithm:

Randomly-Explored Random tree (RRT) is the motion planning technique implemented in which it involves generation of tree in the configuration space (map) and finally returning the path from initial position to goal position where robot has to move in presence of obstacles.

The Algorithm works as follows:

- It takes the initial position of robot and append it to list of nodes that stores all the points that tree is going to traverse to goal position.
- Now it generates a random point in the map and finds which node of the tree is closer to the random point.
- After finding nearest node it generates three points left, right and straight from the nearest node in the tree and checks which point out of these three is nearest to the random point.

- The tree grows to the next node based on the three points using non-holonomic constraints using Runge-Kutta Integration method and appends all the nodes to the tree.
- In this way the tree grows using the previous three steps until all the random points generated are traversed.
- Now it appends the goal position to the tree from its nearest node in the tree.
- Finally, the path from initial position to goal position gets established.

```
Algorithm BuildRRT
  Input: Initial configuration $q_{init}$, number of vertices in RRT $K$, incremental distance $h$)
  Output: RRT Tree $T$

  $T$.init($q_{init}$)
  for $k = 1$ to $K$
    $q_{rand} \leftarrow$ RAND_CONF()
    $q_{near} \leftarrow$ NEAREST_VERTEX($q_{rand}$, $T$)
    $q_{new} \leftarrow$ NEW_CONF($q_{near}$, $q_{rand}$, $h$)
    $T$.add_vertex($q_{new}$)
    $T$.add_edge($q_{near}$, $q_{new}$)
  return $T$
```

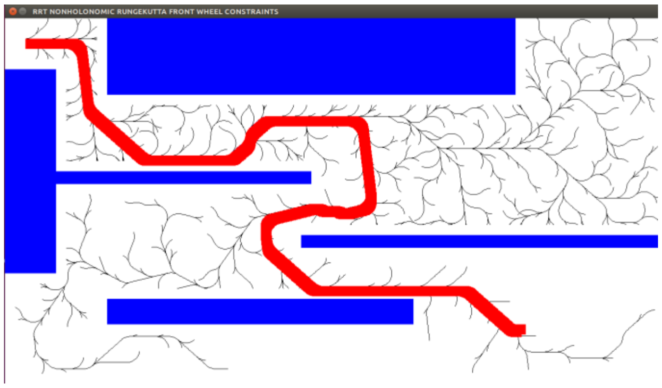
Note:

1. The RRT mainly depends on the number of random points used for developing the tree.

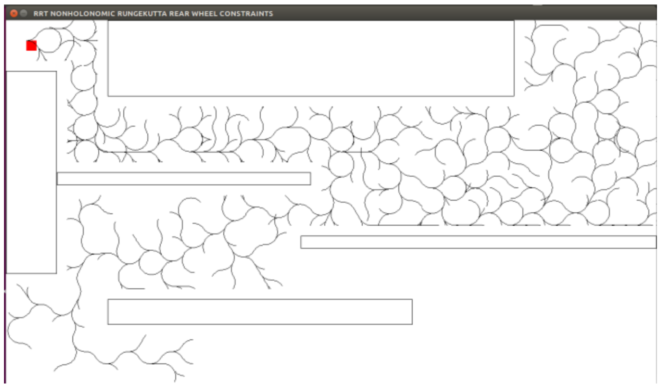
Result accomplished:

Results of the RRT growth in the given environment mainly depends on the number of random points generated and appended in the tree. Shown below are many different cases that changes the number of random nodes in the tree growth and its successful and failed cases. When the random points are increased the tree has reached all the parts of the environment increasing success rate but it had also increased the running time of the Algorithm.

Random points: 3000 Front wheels:

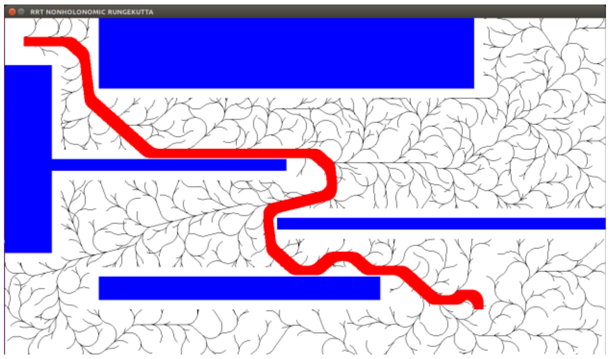


Random points: 3000 Rear wheels:

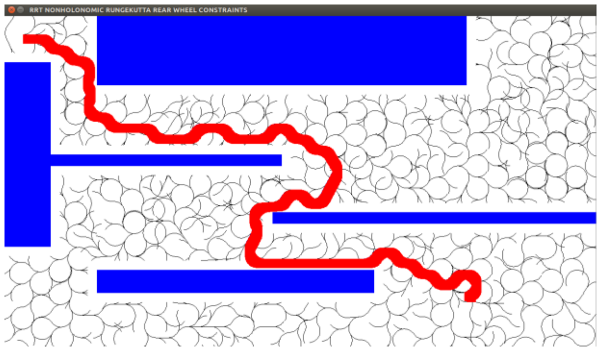


When the number of random points are taken as 3000 and tested for two Non-holonomic cases of simple car. The front wheels constraints case is successful traversing the path with higher probability than rear wheel constraints. These cases indicate that below 3000 random points tree growth is not so successful in this complex environment.

Random points: 5000 Front wheels:

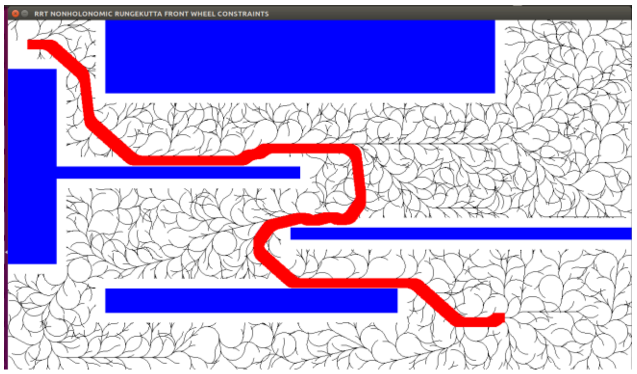


Random points: 5000 Rear wheels:

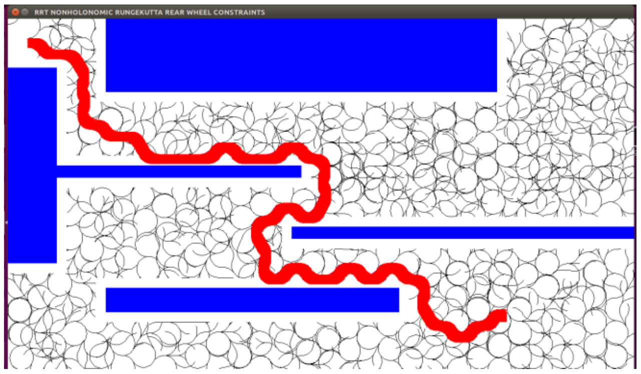


In case of the 5000 random points the robot has successfully traversed from the start to goal point. In this case front wheel constraints have taken smoother curves and turns when required than the rear wheel constraints.

Random points: 10000 Front wheels:



Random points: 10000 Rear wheels:



:

When the random points were increased to 10000, the tree had densely traversed the environment such that having 100% success rate. As observed from above pictures the cost of robot moving from initial to goal position had decreased.

Challenges:

Succeeded:

1. Used a data structure called dictionary in python to successfully store the parent nodes of the tree and generate the path.
2. Successfully detected obstacles in environment and restricted tree to grow inside.
3. Extended the Robot from point to 2D nature with Non-Holonomic constraints.
4. Implemented curve smoothing using 4th Runge-Kutta Method.
5. Designed a complex environment of obstacles and successfully traversed 2D robot to goal position.

Yet to do:

1. Appending end node to the tree with the required orientation of robot is still under implementation. Right now the robot reaches it goal coordinates but without orientation.
2. Developing some more environment to test the working of the planner.

REFERENCES:

1. http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf
2. http://www.math-cs.gordon.edu/courses/ma342/python/diffeq.py
3. https://rosettacode.org/wiki/Ray-casting_algorithm
4. https://www.cs.ucsb.edu/~pconrad/cs5nm/topics/pygame/drawing/