|  |
| --- |
| RRT for Holonomic and Dubin’s car |
| Robot Motion Planning Course project |
|  |
| This project implements the Rapidly exploring Random Tree (RRT) in python. This algorithm will deal with the systems that have differential constraints. In this case, the system is Dubin’s car. |
|  |

RRT for Holonomic and Dubin’s car

Robot Motion Planning Course project

# RRT Path Planning

• • •

RRTs were developed by Steven M. LaValle and James J. Kuffner Jr. They easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and have been widely used in autonomous robotic path planning. A simple and efficient randomized algorithm is presented for solving single-query path planning problems in high-dimensional configuration spaces. The method works by incrementally building two Rapidly-exploring Random Trees (RRTs) rooted at the start and the goal configurations. The trees each explore space around them and also advance towards each other through the use of a simple greedy heuristic.

# Objective:

In our day to day life we face problem of path planning. This problem falls in domain of Robotics and is of critical importance. Here, aim of the project is to implement path planning algorithm in 2D environment. First we solve the problem for holonomic motions and then we modify the planner for non-holonomic systems. For non-holonomic system, a Dubin’s car model is considered. The environment used is without obstacles with a single body robot. Ultimate aim is to make a system for UAV like devices to fly form one point to other cover particular points.

The reason to choose RRT algorithm among all the planning algorithms is it is basic building block for advance algorithms like RRT\* and can be modified easily to mutate in another version of RRT. Also, it can handle non-holonomic systems.

### Design:

The basic idea is to use control theoretic representation, and incrementally grow a search tree from an initial state by applying control input over a short time interval to reach new state. Each vertex in the tree represents a state and each directed edge represents the child vertex and parent vertex pair. The algorithm is referred from the book Planning Algorithm by LaValle and related papers[1].

Input: Initial configuration of a robot (qinit).

Output: RRT Graph

RRT algorithm:

*G*.init(*qinit*)

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

*qrand* ← RAND\_CONF()

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

*qnew* ← steerNgo(*qnear*, *qrand*, *Δq*)

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

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

**return** *G*

Except from some additional constraints, algorithms remains same for holonomic and non-holonomic systems.

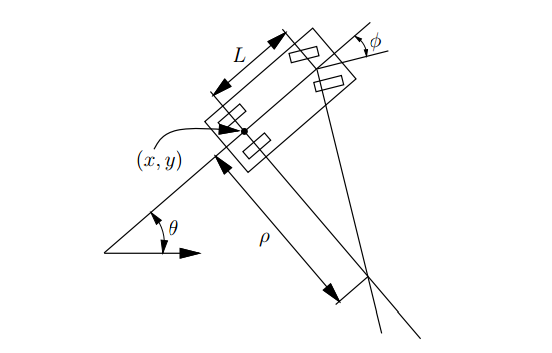
## Implementation:

This algorithm is implemented in python. The creation of Graph is done with class objects as below:

C:\Users\Sachin\Downloads\RMP.jpg

This is very simple to understand and self-explanatory diagram. In this diagram dist12 represents the distance from source to destination i.e. from (x1,y1,theta1) to (x2,y2,theta2) this is useful for further phase of the project RRT\*. This implementation minimizes the use of external libraries. The libraries used are numpy and pygame for display purpose.

Here the systems is 2D Dubin’s car which cannot move backwards and has a minimum turning radius of 45 degrees.



Above figure is taken from chapter 13 of Planning Algorithm book[2].

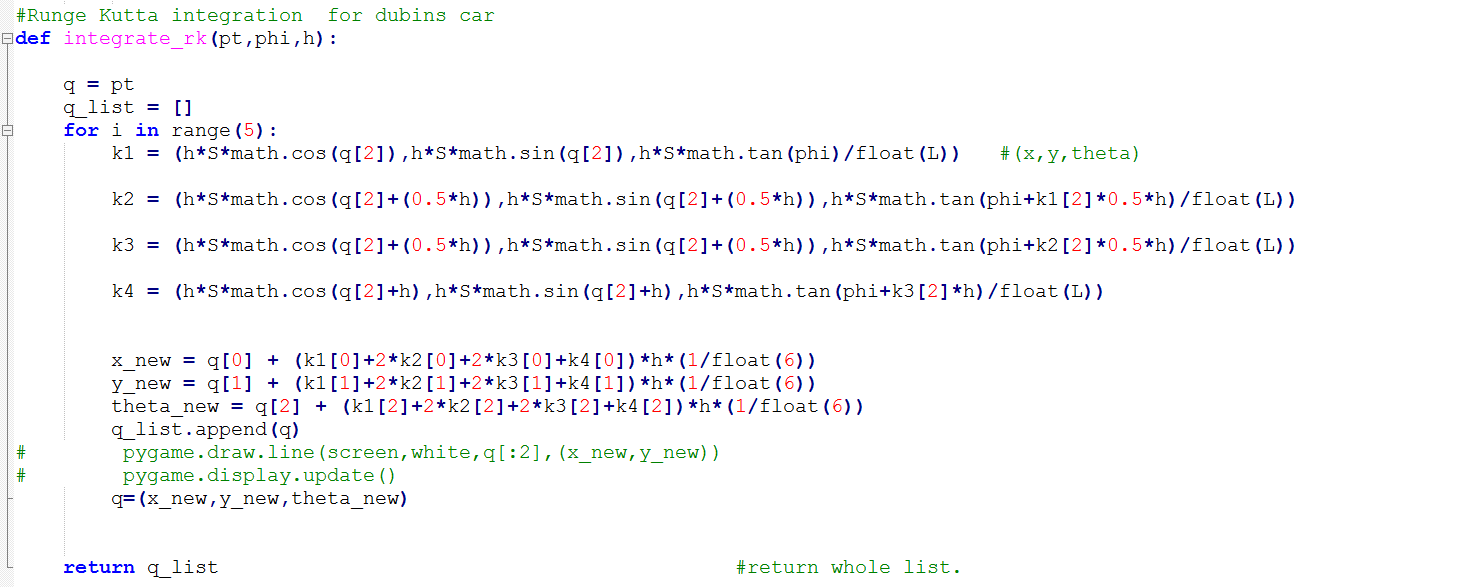
Car has 3 degrees of freedom- Cspace = R2 × S1. The equations for Dubins car is as below:

X’ = us cos θ …….. us is a control input +1 to move forward, we don’t move back in this case

Y’ = us sin θ

θ = us/L tan uΘ ….. uΘ control input for left and right turn

In order to get a next state/Xnew we need to integrate the above equation. This us done using Runge Kutta method. Integration will give you 5 intermediate points. This will be done for all three control input 1. Left turn and move forward 2. No turn and move forward 3. Right turn and move forward. Hence, we will be getting total 15 points. We will check the all 15 points against the random point. If random point is matches with one of the points, we return that point. Otherwise, we will return the nearest point.

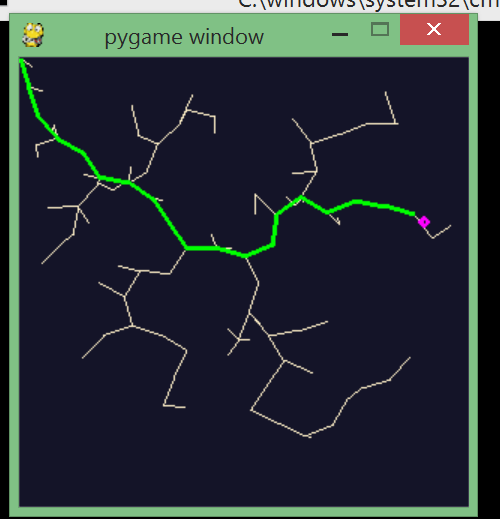


Runge Kutta implementation for Dubin’s car.

## Examples:

Below are illustration for different parameters on RRT-Holonomic system.

Epsi = 20 #iterations = 100



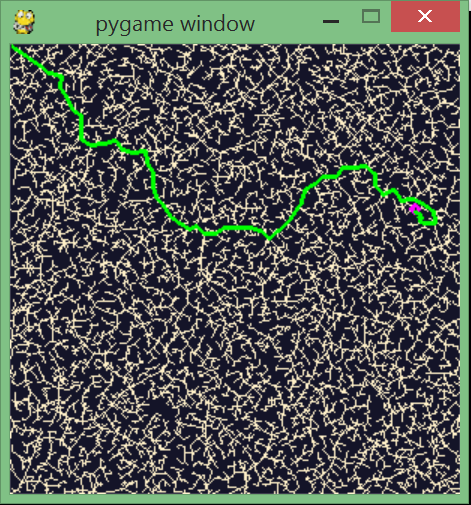
2. Epsi = 25 #iteration = 500



3. Epsi = 8 #iteration 2000



4. Epsi = 8 #iteration = 10000



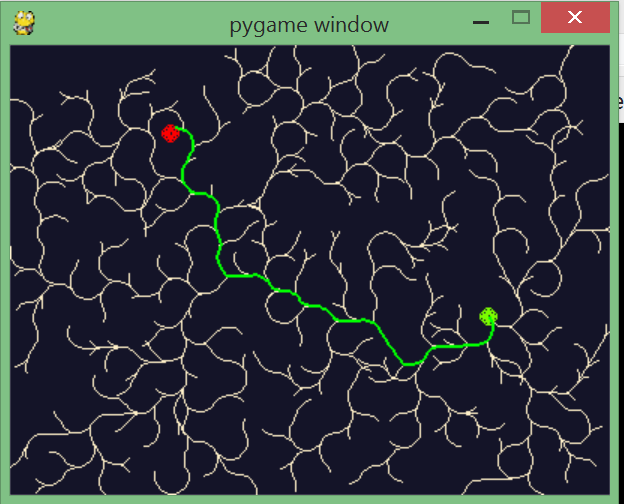
RRT for Non Holonomic System:

1. S=12

h=0.5

L=15

K= 1000

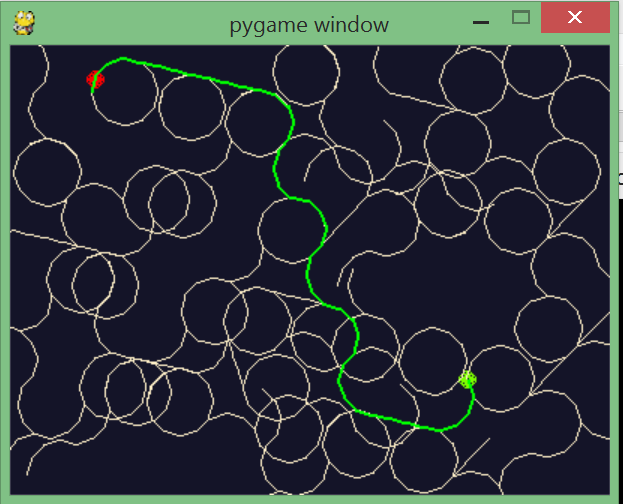


2. S=12

h=1

L=15

K=500

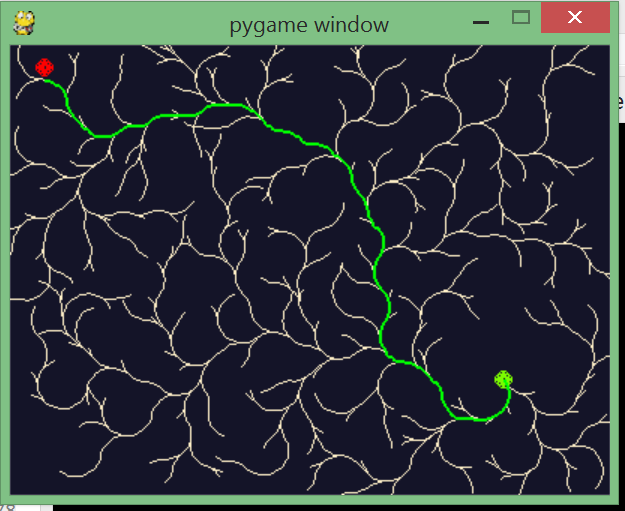


3. S=12

h=0.5

L=20

K=1000



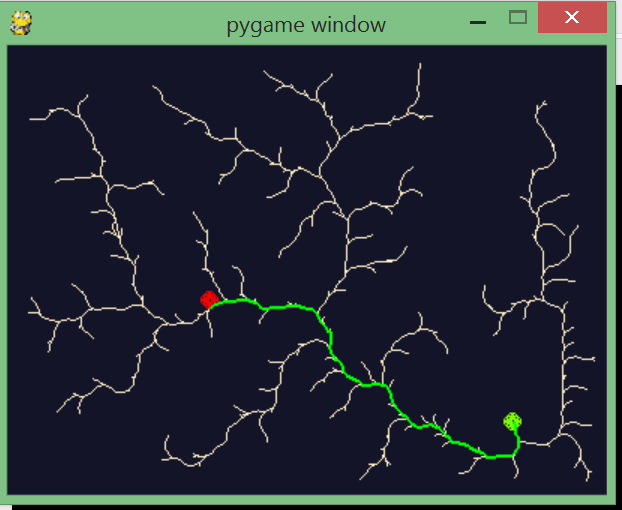
4.

S=5

h=0.5

L=10

K=1000



## Special cases:

As a Dubins car cannot move backwards and has a minimum turning radius, when your qinit is in corner such that there is no place to move or turn, tree will not explore. Hence, robot cannot move. You may need more iteration when the h is small/minimum turning angle is more. In such condition you will need to sample the space rigorously to reach to qgoal.

# Assumptions:

If qgoal does not belong to the graph then the nearest point is found and path is found till the nearest point. Also, robot is a point robot. It is visually difficult to show the car like robot on the pygame screen.

## Accomplishment:

Definitely goal was to implement RRT planner for holonomic robots. Furthermore, take this implementation to implement Dubin’s car so that we can apply this to a UAV or similar objects easily. Ultimately, the system will have RRT\* algorithm for UAV like systems. The steps towards the RRT\* exist in the code such as shortest path algorithm ‘Dijkstra’s algorithm’.

## References:

[1],[2] Planning Algorithm by LaValle - http://planning.cs.uiuc.edu/ch13.pdf

[3] <http://www.pygame.org/hifi.html>