Lab2 - Path Planning - Radhika Nayar - 805-226-085
0(a): Github repo link:
0(b,c): split ⅓ - all of us worked on the code together and then we spit the report

Problem Statement:
The objective of this lab was to build autonomy into a simple two-wheeled non-holonomic robot. The robot occupies a space of 90mm X100mm and has two wheels of given radius separated by a given distance. We implemented sampling based planning algorithm such as RRT and RRT* to get our robot to a desired state.

Tasks: We set up operational space and configurational space of our robot first.
We made objects for Obstacle (two diagonal coordinates) and States (x, y, angle, time, parent) and set our space of dimension as 1500 x 1500 mm. We implemented the functions below:
get_path function returns the distance between two points and gives the slope.
closest_point function returns a state from a list of statelist that is closest to the next state.
one_second_trajectory returns the next state of our robot after it has taken one second to move towards target state
is_collision_free function returns a boolean that says whether our state collides with the obstacle or not
Using these above functions, we are able to implement the RRT and RRT* algorithm. The reference below explains RRT and RRT* in pseudocode very well: https://medium.com/@theclassytim/robotic-path-planning-rrt-and-rrt-212319121378

RRT

```
Qgoal //region that identifies success
Counter = 0 //keeps track of iterations
lim = n //number of iterations algorithm should run for
G(V,E) //Graph containing edges and vertices, initialized as empty
While counter < lim:
    Xnew  = RandomPosition()
    if IsInObstacle(Xnew) == True:
        continue
    Xnearest = Nearest(G(V,E),Xnew) //find nearest vertex
    Link = Chain(Xnew,Xnearest)
    G.append(Link)
    if Xnew in Qgoal:
        Return G
Return G
```

RRT*

```
Rad = r
G(V,E) //Graph containing edges and vertices
For itr in range(0…n)
    Xnew = RandomPosition()
    If Obstacle(Xnew) == True, try again
    Xnearest = Nearest(G(V,E),Xnew)
    Cost(Xnew) = Distance(Xnew,Xnearest)
    Xbest,Xneighbors = findNeighbors(G(V,E),Xnew,Rad)
    Link = Chain(Xnew,Xbest)
    For x’ in Xneighbors
        If Cost(Xnew) + Distance(Xnew,x’) < Cost(x’)
            Cost(x’) = Cost(Xnew)+Distance(Xnew,x’)
            Parent(x’) = Xnew
            G += {Xnew,x’}
    G += Link
Return G
```

The sampling based planning (SBP) algorithms such as RRT and RRT* are used to provide quick solutions for high dimensional problems using randomized sampling in the search space. RRT constructs a tree using random sampling in the search space. The tree starts from the initial state and expands to find path towards goal state. During each iteration, a random node is selected from the configuration space. If the random sample lies in the obstacle free region, then the nearest node to the random node is searched in the tree according to euclidean distance metric. If random node is accessible to the nearest node according to the predefined step size (distance from one-second trajectory), then the tree is expanded by connecting random node to the tree. Also a boolean collision checking process is performed to ensure collision free connection between tree new node and nearest node. This process continues until an initial path is found or until fixed number of iterations incase the algorithm does not converge. The RRT* improves the path quality by introducing major features of tree rewinding and best neighbour search. RRT* due to increased efficiency is less jagged and shorter path. However, RRT* obtains the asymptotic optimality at the cost of execution time and slower path convergence rate. These operations have an efficiency trade-off. The rewire function checks if the cost to the nodes in the Xnear is less through Xnew as compared to their older costs and then its parent is changed to Xnew. Next the choose parent function selects the best parent Xnew from the nearby nodes.

Results:
To evaluate our RRT algorithm we have randomly selected different number and dimension of obstacles along with different goal states and initial states. In each trajectory, we listed out the number of nodes in the tree, the number of nodes in the path and the time it took to run RRT algorithm. We made two trajectories for head-in parking and two trajectories for parallel-parking that avoided collision. The runtime for two trajectories of parallel parking was: 0.105s and 1.06s. The runtime for two trajectories for head-in parking was: 0.156s and 2.404s. As the number of nodes in the tree increased, our RRT algorithm became more inefficient and took a longer time to run. RRT* algorithm improved the path quality by being less jagged and shorter however it increased the computational time. The complexity of both algorithms is the same. Note in stochastic condition our planner will take even longer to converge. A few potential tasks our robot could find applications in are autonomous valet and parking systems, construction vehicles carrying materials through a cluttered environment with obstacles.