RBE550– Motion Planning
Homework- 2
Standard Algorithm Implementation

By
Ajith Kumar Jayamoorthy

Contents:

1. PROBABILISTIC ROADMAP (PRM)

The basic idea behind PRM is to take random samples from the configuration space of the robot, testing them for whether they are in the free space and use a local planner to attempt to connect these configurations to other nearby configurations, if there no collision in path. PRM is a multi-query algorithm. The probabilistic roadmap planner has two levels:

1. Construction phase
2. Query phase

In the construction phase, a roadmap (graph) is built, approximating the motions that can be made in the environment.

1. A random configuration is created.
2. Then, it is connected to some neighbours, typically either the k nearest neighbours or all neighbours less than some predetermined distance.
3. Configurations and connections are added to the graph until the roadmap is dense enough.

In the query phase,

1. The start and goal configurations are connected to the graph
2. The path is obtained by a Dijkstra's shortest path query.

Pseudo code for PRM:

Let: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
1: **loop**
2: $c \leftarrow$ a (useful) configuration in $\mathcal{C}_{\text{free}}$
3: $V \leftarrow V \cup \{c\}$
4: $N_c \leftarrow$ a set of (useful) nodes chosen from V
5: **for all** $c' \in N_c$, in order of increasing distance from c **do**
6: **if** c' and c are not connected in G **then**
7: **if** the local planner finds a path between c' and c **then**
8: add the edge $c'c$ to E

For PRM 4 different sampling methods have been implemented:

a) Uniform sampling
b) Random sampling
c) Gaussian sampling
d) Bridge sampling

a) Uniform Sampling:

EXPLANATION:

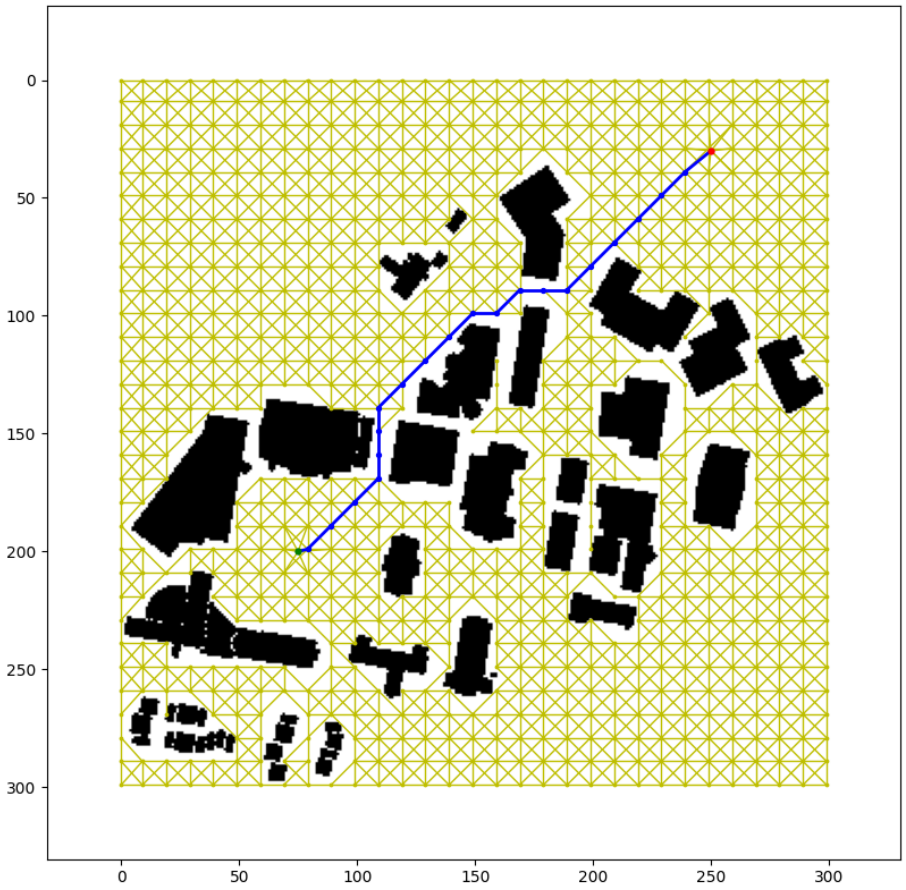
In uniform sampling the probability of picking any point from the configuration space is **same or uniform**. In this regard, we have sampled points in a grid form, where the corners of the grid represent the sampled points/nodes. As the resolution of the image is square, therefore the following formula is used:

Number of points to be samples on each side = $\sqrt{npoints}$, where 'npoints' is the number of points to be sampled.

```
def uniform_sample(self, n_pts):
    '''Use uniform sampling and store valid points
    arguments:
        n_pts - number of points try to sample,
                not the number of final sampled points

    check collision and append valide points to self.samples
    as [(row1, col1), (row2, col2), (row3, col3) ...]
    '''
    ## Initialize graph
    self.graph.clear()
    sample_row = np.linspace(0,self.size_row-1,int(np.sqrt(n_pts))).astype(int)
    sample_col = np.linspace(0,self.size_col-1,int(np.sqrt(n_pts))).astype(int)
    for i in sample_row:
        for j in sample_col:
            if self.map_array[i,j] == 1 :
                self.samples.append((i,j))
```

OUTPUT and OBSERVATIONS:



We can observe that the nodes are equally spaced and each node has same probability of getting sampled. Thus, the major part of the C-space if split into sample configurations and the shortest path in this samples is evaluated through Dijkstra search algorithm. The following image gives the information of number of nodes and edges as well as path length:

```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
The constructed graph has 809 nodes and 2664 edges
The path length is 262.18
```

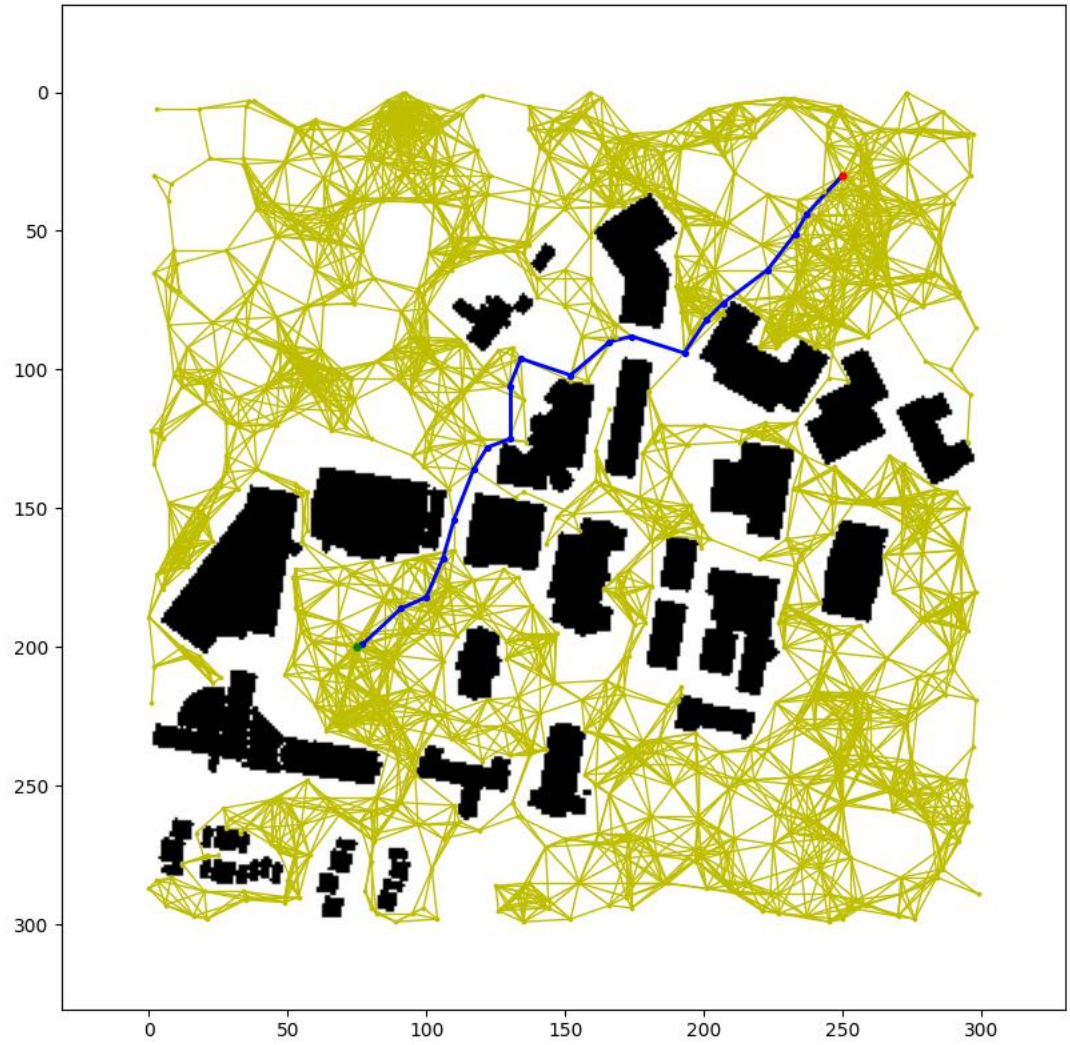
b) Random Sampling

EXPLANATION:

In this sampling method, the points are sampled at random. After that they are checked if they lie in region of obstacle, if not it is appended to the sample list, else discarded. We choose the each axis values at random using the python numpy's random integer generator. The following the code snipped corresponding to sample generation:

```
# Initialize graph
self.graph.clear()
x = np.random.randint(0,self.size_row,n_pts)
y = np.random.randint(0,self.size_col,n_pts)
```

OUTPUT and OBSERVATIONS:



Like uniform sampling, each node has the same probability of getting sampled in random sampling, but the main difference is that the nodes are randomly picked during sampling and are connected with each other. Thus, here the majority of the C-space may not be evenly sampled like uniform sampling and the shortest path in this samples is also evaluated through Dijkstra search algorithm. The following image gives the information of number of nodes and edges as well as path length:

```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
The constructed graph has 841 nodes and 4731 edges
The path length is 280.10
```

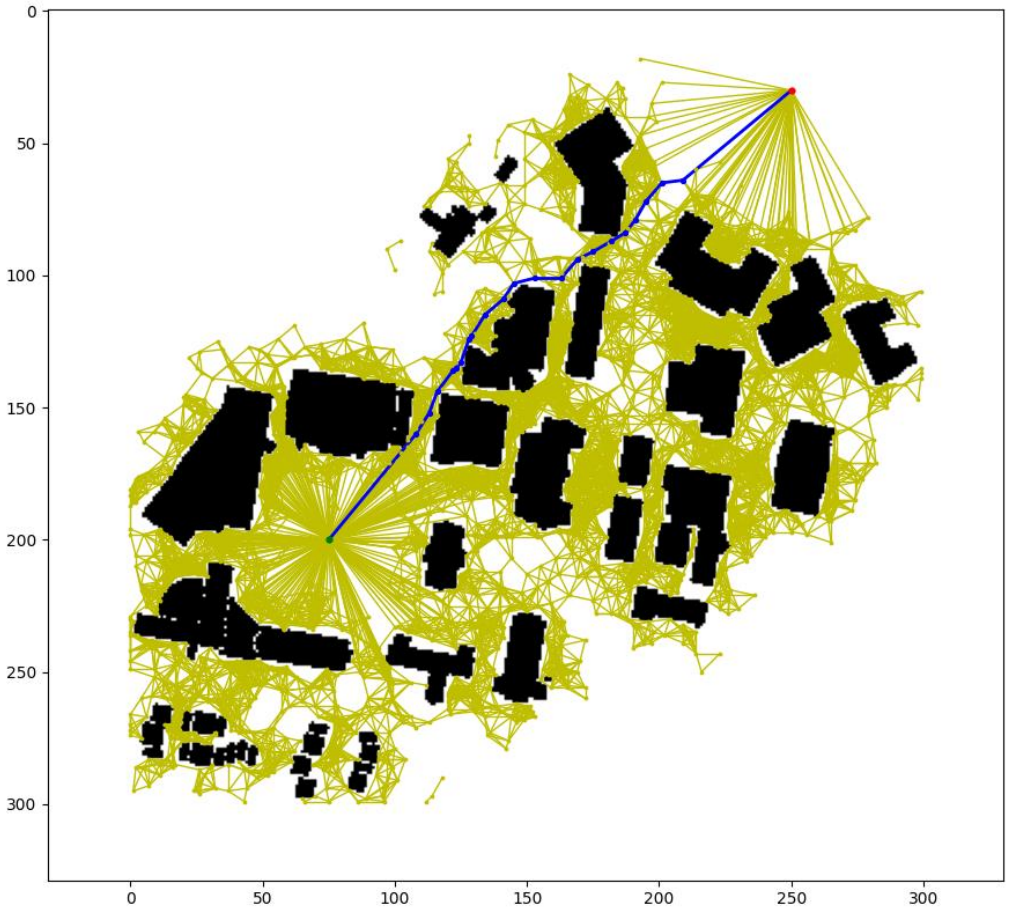
c) Gaussian Sampling

EXPLANATION:

Gaussian sampling is a method used to sample new points near obstacles. The idea is to take a sample point in the obstacle region. A new point is sampled from the from a Gaussian distribution with mean as the first point. If the new point is in free space, then this point is added to the samples list. The algorithm for gaussian sampling is as follows:

1. **loop**
2. $c_1 \leftarrow$ a random configuration
3. $d \leftarrow$ a distance chosen according to a normal distribution
4. $c_2 \leftarrow$ a random conf. at distance d from c_1
5. **if** $c_1 \in C_{\text{free}}$ **and** $c_2 \notin C_{\text{free}}$ **then**
6. add c_1 to the graph
7. **else if** $c_2 \in C_{\text{free}}$ **and** $c_1 \notin C_{\text{free}}$ **then**
8. add c_2 to the graph
9. **else**
10. discard both

OUTPUT and OBSERVATIONS:



As mentioned above the points are sampled close to obstacle and thus all the boundary of the obstacles are know to us to some extent. This helps to create a path around the boundaries for the robot. The sampling probability of points near the boundaries are very high compared to those are in the larger free space. The following image gives the information of number of nodes and edges as well as path length:

```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
The constructed graph has 1924 nodes and 12884 edges
The path length is 253.92
```

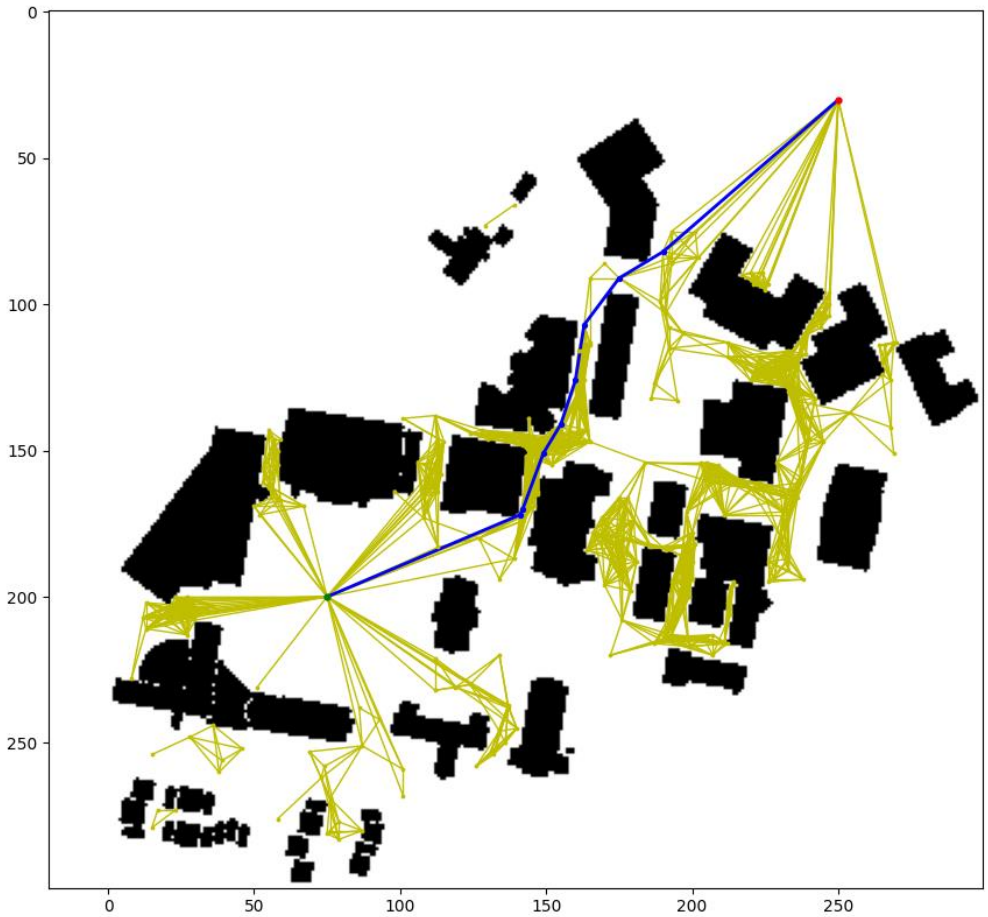
d) Bridge Sampling

EXPLANATION:

The bridge sampling method is used to cover narrow passages in the configuration space which might be an optimum path. The idea behind this method is to sample a random point and check if it is an obstacle. If the random point is an obstacle then second point is sampled from a Gaussian distribution with mean as the first point. The second point is check if it is an obstacle. Then if both the points are obstacle, then the middle point between them is computer and if it is a free space, then it is appended to the samples list. The algorithm for the bridge sampling is as follows:

1. **repeat**
2. Pick a point x from $\mathcal{C}$ uniformly at random.
3. **if** CLEARANCE(x) returns FALSE **then**
4. Pick a point x' in the neighborhood of x according to a suitable probability density λ_x.
5. **if** CLEARANCE(x') returns FALSE **then**
6. Set p to be the midpoint of line segment $\overline{xx'}$.
7. **if** CLEARANCE(p) returns TRUE **then**
8. Insert p into G as a new milestone.

OUTPUT and OBSERVATIONS:



As mentioned above the points are sampled in between the obstacle and thus most of the narrow passage between obstacles are know to us to some extent. This helps to create a path through the narrow passage between boundaries for the robot, thus helping to obtain a shortest optimum path. The sampling probability of points between boundaries are very high compared to those are in the larger free space. The following image gives the information of number of nodes and edges as well as path length:

```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
The constructed graph has 295 nodes and 1720 edges
The path length is 257.78
```

2. RAPIDLY-EXPLORING RANDOM TREE (RRT)

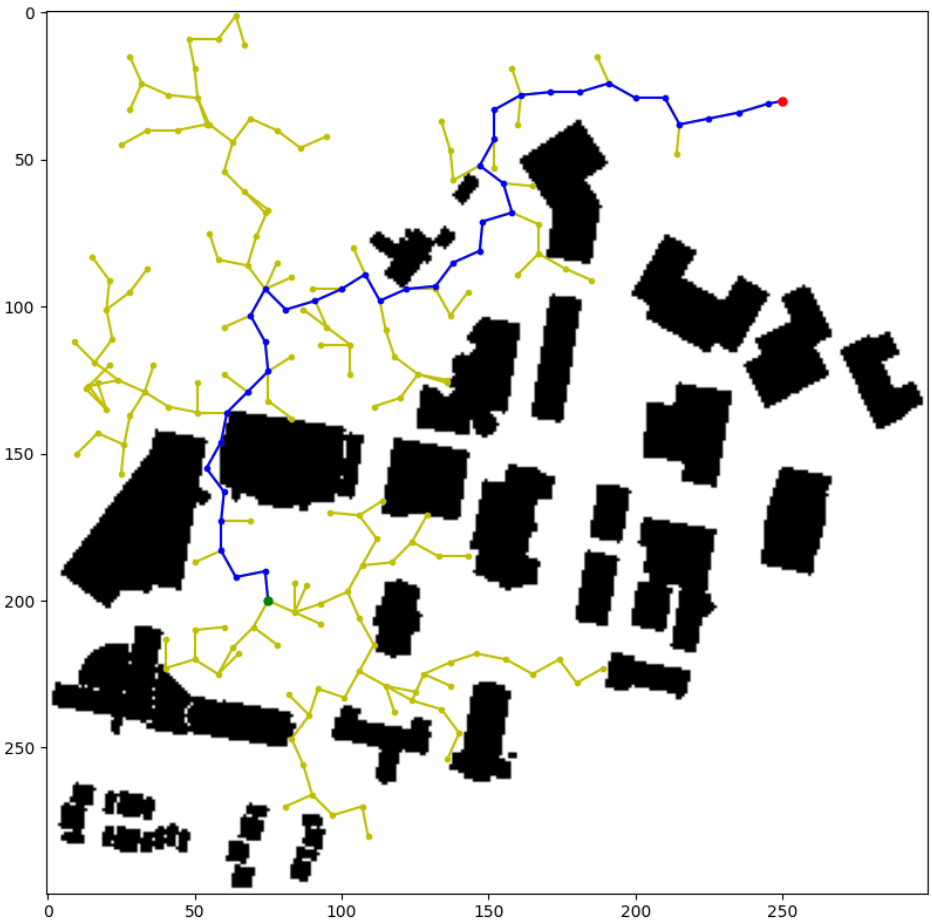
EXPLANATION:

The Rapidly-exploring Random Tree (RRT) is an algorithm designed to efficiently searching an non-convex high-dimensional spaces by randomly building a space-filling tree. The tree is built in steps from a random point sampled in space and there is bias given to the sampling of the random point. In our case, the bias given to the goal to be sampled. The root of the tree is considered to be the start node and the tree grows from the start position until it reaches the neighbourhood of the goal region. After which the goal is connect and appended to the nodes list. The RRT can be represent using the following algorithm:

```
Algorithm BuildRRT
    Input: Initial configuration q_init, number of vertices in RRT K, incremental distance Δq
    Output: RRT graph G

    G.init(q_init)
    for k = 1 to K do
        q_rand ← RAND_CONF()
        q_near ← NEAREST_VERTEX(q_rand, G)
        q_new ← NEW_CONF(q_near, q_rand, Δq)
        G.add_vertex(q_new)
        G.add_edge(q_near, q_new)
    return G
```

OUTPUT and OBSERVATIONS:



The tree initiate from the start position, which forms the root, from there the tree rapidly samples random points and then extents toward those point with by a step (delta). The path formed by the tree is sub-optimal as the loop terminates as soon as the goal is reached. The following image gives the information of number of nodes and edges as well as path length:

```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
It took 186 nodes to find the current path
The path length is 389.73
```

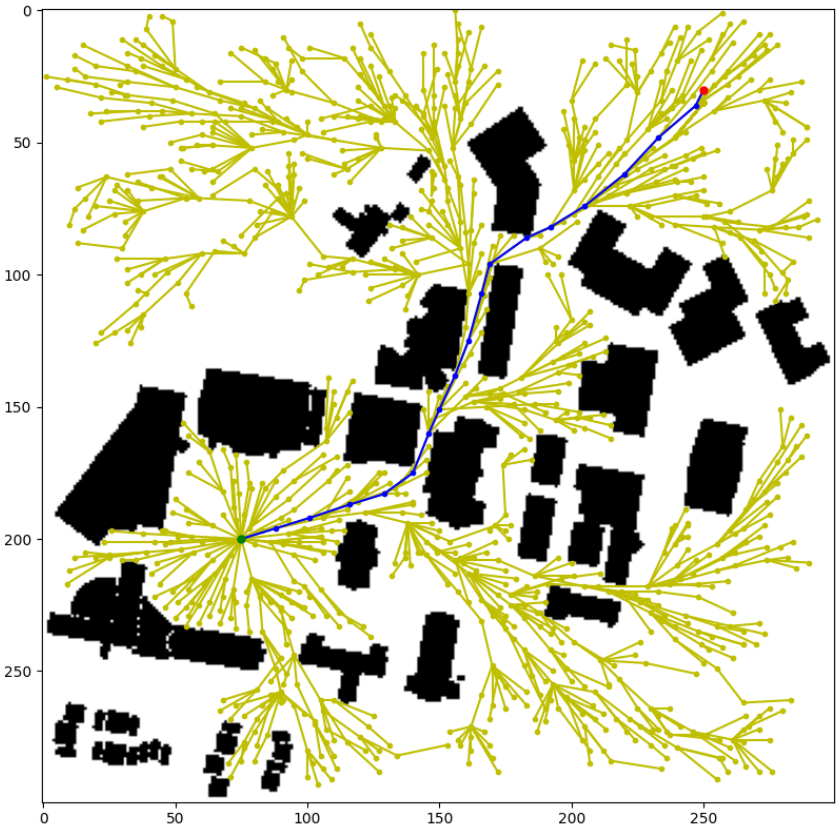
3. RAPIDLY-EXPLORING RANDOM TREE* (RRT*)

EXPLANATION:

RRT* is a variant of the RRT where additional constraints are considered for connection. Like RRT, the sampling of the new node remains the same, but instead of connecting the new node to the nearest node, the neighbours of the new node are considered and the cost-to-come from start for the new node is checked through all the neighbours. The neighbour that provides the least cost-to-come to the new node is chosen as the parent for the new node. Then the connections are rewired such that the cost-to-come for the neighbours are less than the cost-to-come through the new node, if not the new node is assigned as the parent for the neighbour node. The RRT algorithm can be explained as follows:

$T \leftarrow$ InitializeTree();
$T \leftarrow$ InsertNode($\emptyset$, z_{init}, T);
for i=0 to i=N **do**
  $z_{rand} \leftarrow$ Sample(i);
  $z_{nearest} \leftarrow$ Nearest(T, z_{rand});
  (x_{new}, u_{new}, T_{new}) $\leftarrow$ Steer ($z_{nearest}$, z_{rand});
  if Obstaclefree(x_{new}) **then**
    $Z_{near} \leftarrow$ Near(T, z_{new}, |V|);
    $z_{min} \leftarrow$ Chooseparent (Z_{near}, $z_{nearest}$, z_{new}, x_{new});
    $T \leftarrow$ InsertNode(z_{min}, z_{new}, T);
    $T \leftarrow$ Rewire (T, Z_{near}, z_{min}, z_{new});
return T

OUTPUT and OBSERVATIONS:




```
[Running] python -u "/home/ajith/Documents/Standard Search Algorithms/main.py"
It took 1207 nodes to find the current path
The path length is 261.51
```

We can observe that the RRT and RRT* algorithm has produced different trees. This is mainly due to the variation in the implementation of the connection between the new node and tree, as well as due to the implementation of rewire function.

4. ANSWERS:

1. For PRM, what are the advantages and dis-advantages if the four sampling methods in comparison to each other?

   a) **Uniform sampling:**
   Advantages:
   - Low complexity
   - Always ensures to find a path as sample is through the C-space

   Disadvantages:
   - Computationally expensive as huge number of nodes needs to be explored, which might not be a requirement to find the path.

   b) **Random sampling:**
   Advantages:
   - Low complexity

   Disadvantages:
   - Fails to find a path sometime if the sampled nodes are not spread or if not close to the goal.

   c) **Gaussian sampling:**
   Advantages:
   - In scenarios where the robot is stuck in between obstacles, it helps to find the path around the obstacles efficiently

   Disadvantages:
   - It depends on the variance of the Gaussian distribution to find the nearest point and if the variance is set low, then it samples points very close to each other.
   - As the points sampled might be close to each other they might not be able to samples points close to goal, thus not being able to find a feasible path.

   d) **Bridge sampling:**
   Advantages:
   - Helps find paths between obstacles in the narrow passages, thus reducing the path length and time to travel.
   - Less number of nodes/sample points to deal with and thus computationally less expensive.

   Disadvantages:
   - If proper radius near the goal is not set, then the goal might not be able to connect with any of the sampled points and thus no path may be found.

2. For RRT, what is the main difference between RRT and RRT*? What changes does it make in terms of efficiency of the algorithms and optimality of the search result?

   - The main difference between RRT and RRT* is that, in RRT the new sampled point connects with the nearest node available from the tree irrespective of the cost-to-come, where in case of the RRT*, the neighbour nodes are searched to find the optimum cost-to-come and the new node then connects to that neighbour node.
     Another major difference is the rewiring of the neighbouring nodes in RRT* with respect to the new node, where as it RRT this doesn't happen.

   - **Efficiency:** If we evaluate efficient with respect to time, then RRT gives an path quickest, but at the same time the path is sub-optimal. On the other hand RRT* algorithm takes a longer time to converge to an optimal path as well as takes more nodes to rewire and connect and thus not efficient with respect to time.

   - **Optimality:** On evaluating optimality of the algorithm with respect to path, RRT* converges to a optimal path with time when compared to RRT which most of the time returns an sub-optimal path.

3. Comparing between PRM and RRT, what are the advantages and disadvantages?

PRM:

Advantage:

- For static environments, PRM can be used to re-evaluate path is the goal is changed in between transversing. The major advantage is that the new path need not be evaluated from beginning through sampling and a feasible path can be obtained from already created connections.

Disadvantage:

- PRM need an optimum range to be set to each for nearest neighbouring node (Local planner), if no connections can be formed then no path can be formed.
- PRM coverage of the entire C-Space entirely depends on the type of sampling we perform in the algorithm and if the sampling method is not suitable for the environment, then maybe no path can be found.
- PRM cannot be used for Dynamics obstacle environments as re-evaluation of path to goal becomes computationally very expensive.

RRT:

Advantage:

- RRT can be used to find path in dynamic obstacle environment. If an obstacle is encounter, a new point can be sampled and the tree can be built around the obstacle thus providing a path.
- RRT can be bias to explore points near the goal or near free space and thus able to cover wider area of the C-space
- RRT is very efficient in terms of time and thus helpful in real-world scenarios for re-evaluating path when encountered with dynamic obstacle.

Disadvantage:

- RRT give an sub-optimal path most of the time.

5. REFERENCES

1. Geraerts, R., & Overmars, M. H. (2006). Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, *54*(2), 165–173. https://doi.org/10.1016/j.robot.2005.09.026

2. V. Boor, M. H. Overmars and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), 1999, pp. 1018-1023 vol.2, doi: 10.1109/ROBOT.1999.772447.

3. D. Hsu, Tingting Jiang, J. Reif and Zheng Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), 2003, pp. 4420-4426 vol.3, doi: 10.1109/ROBOT.2003.1242285.

4. https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

5. F. Islam, J. Nasir, U. Malik, Y. Ayaz and O. Hasan, "RRT∗-Smart: Rapid convergence implementation of RRT∗ towards optimal solution," 2012 IEEE International Conference on Mechatronics and Automation, 2012, pp. 1651-1656, doi: 10.1109/ICMA.2012.6284384.