RBE 550: Standard Search Algorithms Implementation Report

Joel John
jjohn2@wpi.edu

Worcester Polytechnic Institute — March 8, 2022

1 Sampling Method Comparison

There are four different sampling methods implemented. These are :-

- Uniform sampling
- Random sampling
- Gaussian sampling
- Bridge sampling

1.1 Uniform Sampling

Uniform sampling is a sampling approach in which points are selected at a uniform distance from one another and then connected. This method has the drawback that if not enough points are sampled, narrow spaces cannot be identified, which may result in the failure to find an optimum path, or even a path at all. Resampling the map will not solve the problem because the nodes and edges created will always be the same for the same number of sample points.

1.2 Random Sampling

The points are chosen at random from the map. Because the points are assigned at random, they may or may not be linked depending on how close the points are to each other and the obstacle positions. As a result, at times you won't be able to locate a path at all. Paths discovered using random sampling can be very close to the ideal path or very long. Because the nodes and edges will always be different when resampled, it is completely luck based.

1.3 Gaussian Sampling

Gaussian sampling is a method for capturing the shape of an obstacle. As a result, if there are a lot of obstacles close together, it can be used. Points are randomly sampled and taken into account if they fall on an obstacle. A point in the free space near the obstacle is then identified using the point on the obstacle. This is accomplished by utilizing a Gaussian (normal) distribution to draw the points. To gather enough points in the map's free space, this method requires more points to be sampled than uniform or random sampling. As a result, this method takes longer to compute. If the obstacles are far apart, the points will not be linked since they are too far apart. This will lead to a path not found.

1.4 Bridge Sampling

Bridge sampling is the most effective method for finding a path through narrow regions. Other sample methods do not have enough points in the narrow regions, resulting in insufficient connections and the inability to consistently find a path. The first points are randomly picked and taken into account if they fall on an obstacle. The first points sampled are then used to find a second point on a nearby obstacle (gaussian distribution). Finally, a sample is taken if the midpoint of these locations is in the open space. The majority of these samples fall along narrow regions. This method is slow since it requires a substantially large number of points to be sampled compared to the other methods in order to find enough points on the map.

2 RRT vs RRT*

RRT stands for Rapidly Exploring Random Trees. In this approach, a point is randomly generated in the map. Then it is connected to the nearest node in the tree. Connections are made only if there is no obstacle on the line connecting the points. This is done till the goal is found or the specified number of nodes have been generated. This algorithm is fairly quick but the path is not optimized.

RRT* is an improvement over RRT as it optimizes the path as much as possible by rewiring the tree. When a new point is generated, first it finds the neighbours of the point using a specified radius. It then connects the point to the neighbour by which the point will have the minimum cost to come. Then all the neighbours are checked to see if they will have a lower cost to come when connected to the new point rather than their cuurent parent. If this check is true, the neighbours new parent will be the new point, in this way the tree is 'rewired'. In this way the path is optimized as the tree is rewired to have the shortest cost to come to any point. The more neighbors it considers, the better rewiring and better the path. This algorithm does not find the true optimal path, it just optimizes the path based on the points generated. Points are still added even after the goal is found to consider if there is a more optimal path as the tree will keep getting rewired. This algorithm is much slower than RRT due to the rewiring.

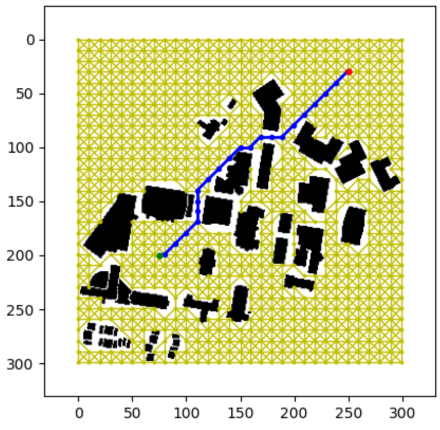
3 PRM vs RRT

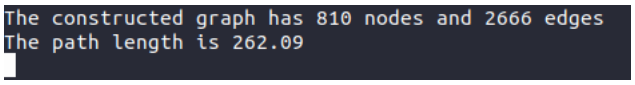
If a robot is required to find a path that passes through various waypoints. To find a path, RRT will have to construct a new graph at each waypoint. It is inefficient to repeatedly generate a graph in order to locate a single path. If the area being explored is large, it becomes even more so. PRM, on the other hand, will create a roadmap that covers the majority of the open space and navigates it using a shortest path algorithm. As a result, the graph does not need to be produced again. PRM is incapable of dealing with map changes, whereas RRT is.

4 Algorithm Implementation and Results

4.1 PRM - Probabilistic RoadMap

It has two phases, the learning phase and the query phase. In the learning phase the points are sampled using a specific method, then a KD-tree is built and the pairs to be connected are found and a graph is built.
In the query phase, the start point and goal point are connected to the samples using a threshold. Then a shortest path algorithm finds a path from the start to the goal.

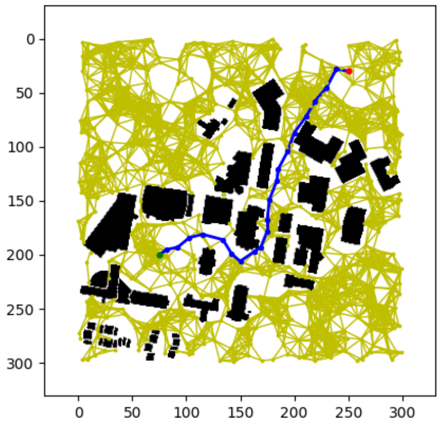


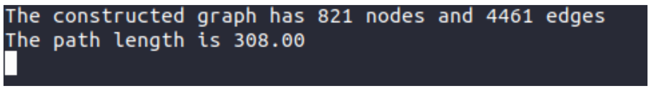


```
The constructed graph has 810 nodes and 2666 edges
The path length is 262.09
```

Figure 1: Uniform Sampling Output

Fig.1 uses Uniform sampling to sample the points. Uniform sampling samples points having a uniform distance between points. It will always have the same path each time it is resampled as it samples the same amount of points.

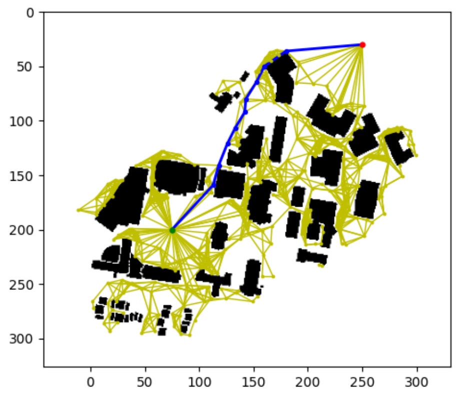


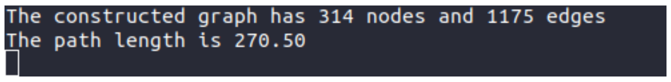


```
The constructed graph has 821 nodes and 4461 edges
The path length is 308.00
```

Figure 2: Random Sampling Output

Fig.2 uses Random sampling to sample the points. Random sampling randomly samples points from a uniform distribution. A new path will be found each time the map is resampled as the points are random. A path may not be found always as it depends on the amount of samples and where the samples are randomly located.

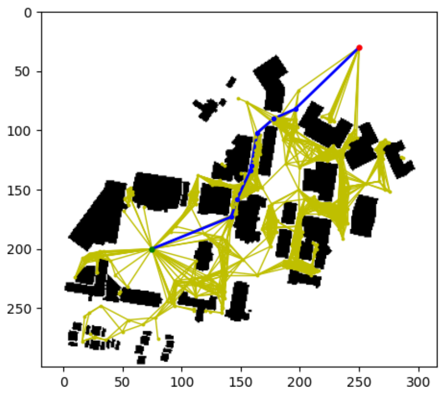


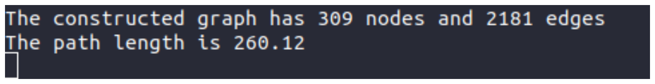


```
The constructed graph has 314 nodes and 1175 edges
The path length is 270.50
```

Figure 3: Gaussian Sampling Output

Fig.3 uses Gaussian sampling to sample the points. Gaussian sampling samples points around an obstacle. The full method is explained in Section 1. We notice since only the points around the obstacle are sampled the number of nodes and edges are lower compared to the previous methods.





```
The constructed graph has 309 nodes and 2181 edges
The path length is 260.12
```

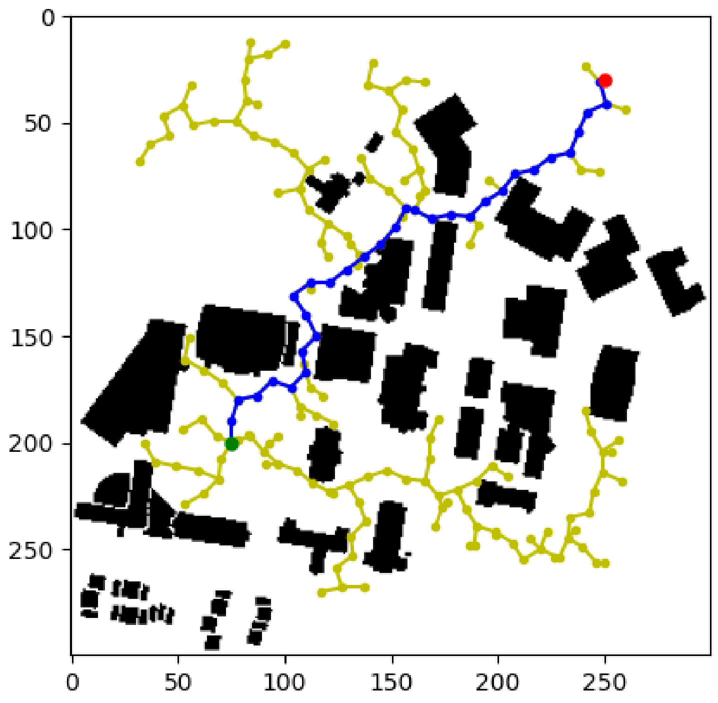
Figure 4: Bridge Sampling Output

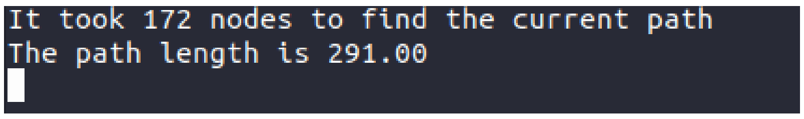
Fig.4 uses Bridge sampling to sample the points. Bridge sampling samples points in narrow regions in the map. The full method is explained in Section 1. The total samples are significantly larger than the other methods. This implementation casts 20000 samples for bridge sampling compared to the 2000 for gaussian and 1000 for uniform and random. The final nodes and edges are very low compared to the total samples cast.

Working of the functions used is explained as comments in the code

4.2 RRT and RRT*

RRT is implemented in the following way. First a random node is generated. A collision check is made to see if the point is in collision. If its not, the nearest node in the node list to the random node is found. The tree is then extended towards the random node in steps from the nearest node. If the line connecting the extended node and the nearest node has no obstacles then the costs and parent information is assigned to the extended node and it is added to the node list. A goal bias is introduced such that occasionally the goal node is used as the random node generated. This process keeps happening in a loop till the goal is found or the specified number of nodes to be generated has been generated.



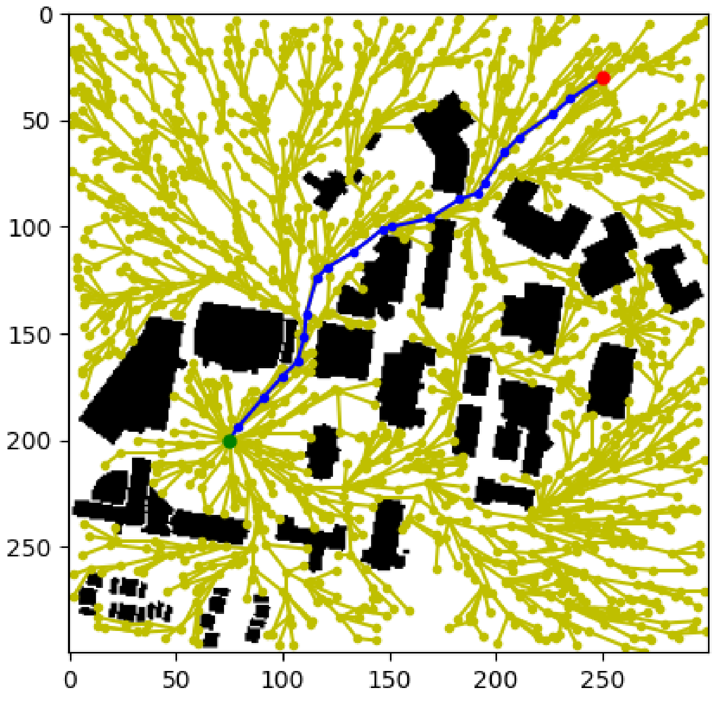


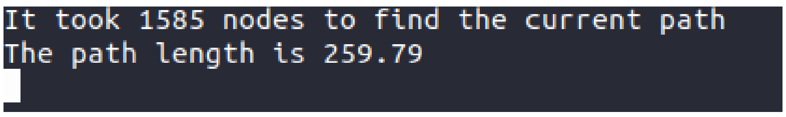
```
It took 172 nodes to find the current path
The path length is 291.00
```

Figure 5: RRT Implementation Output

Fig.5 shows the RRT implementation for a goal bias of 5% and a step length of 10. The tree grows and finds the shortest path, the growth of the tree terminates when the goal is found.

RRT* is implemented similar to RRT. If there are no obstacles on the line connecting extended node and the nearest node then the neighbours of extended node are found using a neighbour size (radius around the extended node). The extended node is connected to the neighbour which would give it the minimum cost to come. Then all the neighbours are rewired. (See Section 2 for more on rewiring). Costs and parent information is assigned and the extended node is added to the node list. This process keeps happening in a loop till the specified number of nodes to be generated has been generated. The loop doesnt terminate when the goal is found as when the tree keeps growing, the path gets more optimized due to the rewiring.





```
It took 1585 nodes to find the current path
The path length is 259.79
```

Figure 6: RRT* Implementation Output

Fig.6 shoes the RRT* Implementation for a goal bias of 10% and a step length of 10. We notice that RRT* path length is much shorter than that of RRT, this is due to the rewiring that happens in RRT*. The rewiring can be noticed in the shape of the tree. We also notice that the amount of nodes is larger than RRT, this is due to the loop not terminating when the goal is found. RRT* has significantly longer computational time than RRT.

Working of the functions used is explained as comments in the code