Steven Hyland

RBE 550 – Motion Planning

02/23/2023

PRM Algorithm Implementation

Each of the four sampling methods have different pros and cons.

Uniform Sampling

This method takes evenly distributed samples across the C-space, only avoiding configurations that are obstructed, due to an obstacle, for instance. The benefits of uniform sampling is that it is the most likely to produce a feasible of the four sampling methods. Since it samples the entire C-space, it will always find a path, if a feasible path is possible. The downsides are that many unnecessary nodes are created in the graph that are far from the goal and start point. Additionally, the final path may not be optimal, since obstacles interfere with the distribution of samples. This means that uniform sampling is robust, yet slow and not always optimal. For example, narrow passages may never be captured by uniform sampling.

Random Sampling

This method takes random samples across the C-space. There may or may not be fewer nodes in the graph than uniform sampling, due to the random nature of this method. This means that the computational complexity may not be much more efficient than uniform sampling, if at all. However, since the C-space is randomly sampled, the possibility of capturing narrow passages is increased. This method is naturally hit-or-miss due to the random nature. The following two methods employ random sampling, with modifications that vastly improve the performance.

Gaussian Sampling

This method is similar to random sampling, in that a point is chosen at random from the C-space. However, a second random point is sampled using a Gaussian distribution about the initial point. Then, only point-pairs that have a collision between them are kept and added to the samples. I.e. only samples that are around obstacles are added. This reduces the cost of keeping all nodes, and more importantly, all edges in the memory, and is far more likely to capture narrow passages. The downside is that this method requires more initial samples to **try**. Since many ‘attempted’ samples will be discarded due to not meeting the conditions of surrounding obstacles, the initial sampling phase will run for longer than for the previous two methods. Luckily, since fewer initial edges will be connected, the searching phase will be shorter.

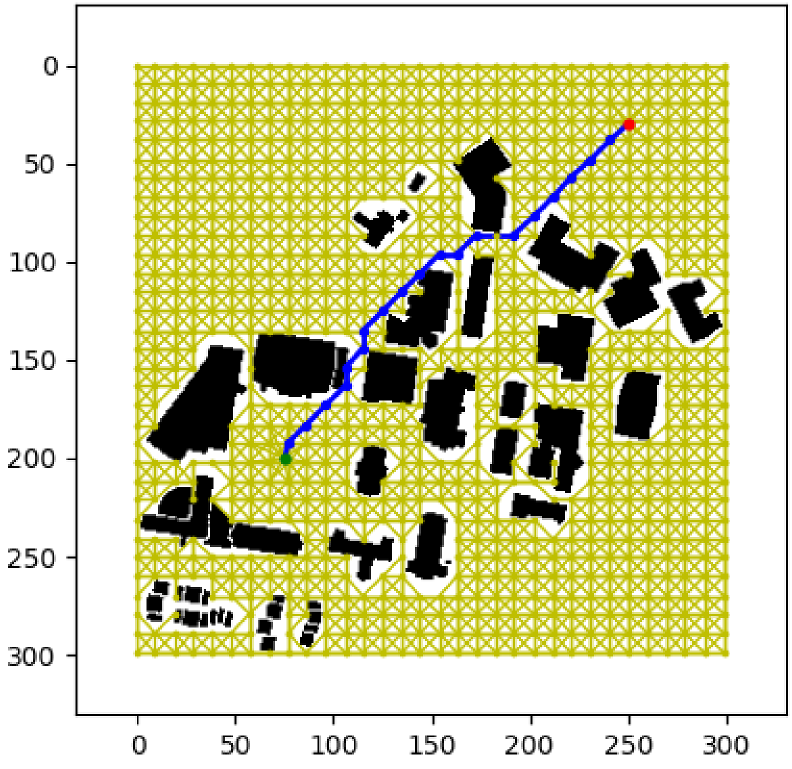
Bridge Sampling

This method builds upon Gaussian Sampling by eliminating the first randomly selected point IF it is not in collision. The second point can be sampled using some probability distribution, such as Gaussian Sampling, and only kept if it is also in collision. This way, only two samples that are in collision are tested, with their midpoint being added to the sample space. This results in great capturing of narrow features, and important corner features (local minima). This method benefits even more from fewer nodes AND from fewer edges. This reduces the cost of the search phase greatly. It also means that a more optimal solution can be found than any of the previous methods. The downside is that if the random sampling isn't completed wholistically (e.g. the gaussian sampling has too narrow search bounds), an optimal solution could be worse than uniform sampling. It also suffers from the same downside of Gaussian sampling, that the 'learning phase' will take longer due to the sheer number of candidate points that are discarded. In this homework, this method provided the best results.

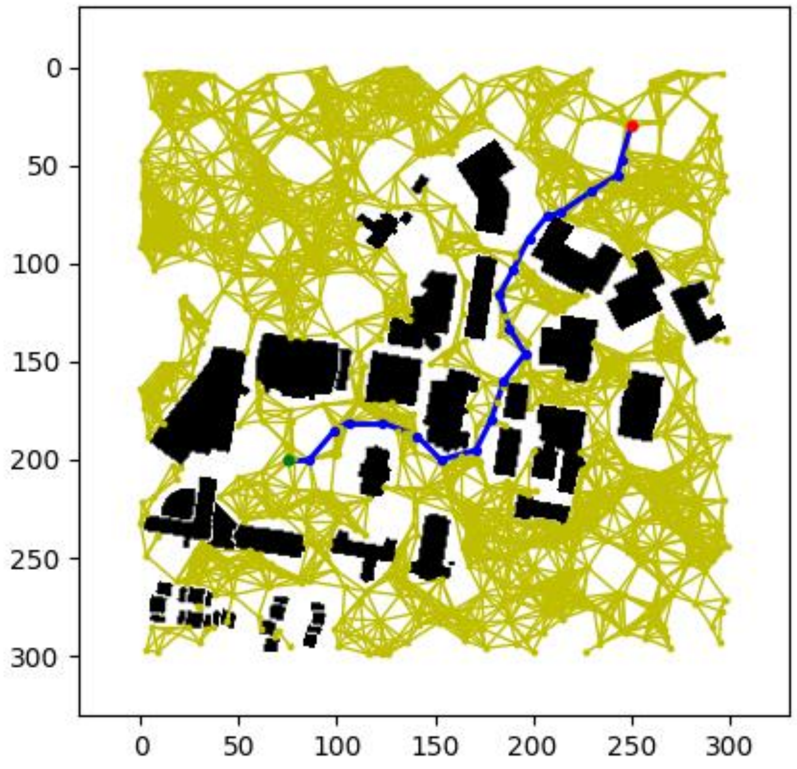
<u>Algorithm Results and Discussion</u>

The algorithm performed very well, and as expected, improved with the more sophisticated sampling methods. Each of the methods used different parameters for calculating the k-d tree and search stage. The main difference between each of the methods is the **k-nearest** neighbors metric searched in the k-d tree. The optimal **k-value** for each method was found using trial-and-error, with the best value minimizing *k* while constructing a complete graph and finding an optimal solution. To optimize performance, only a single k-d tree was generated for each of the separate methods, instead of updating the tree every time step.

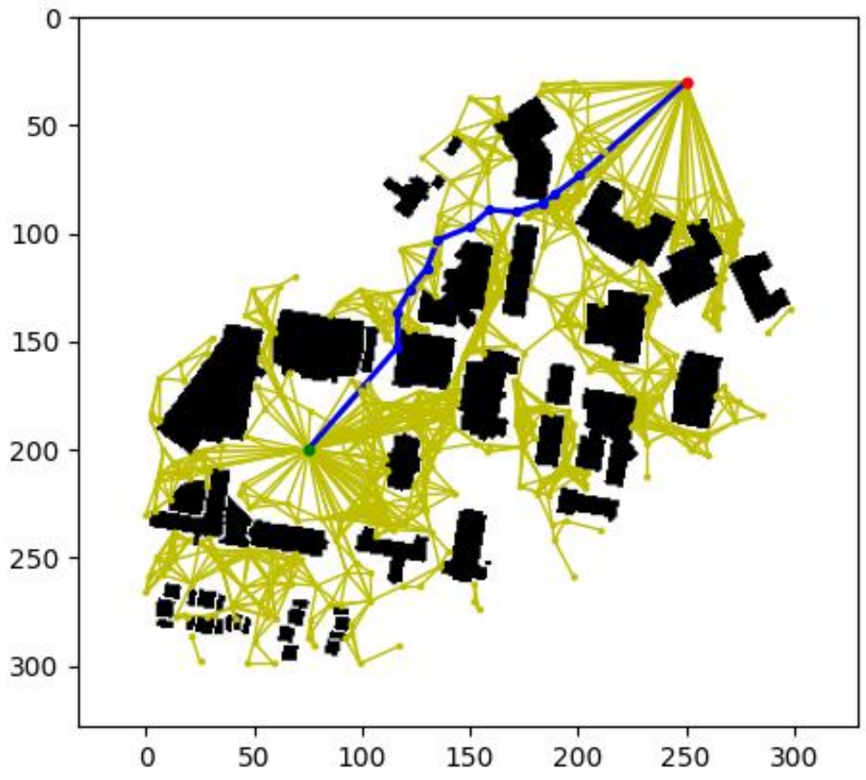
The uniform sampling sampled the most nodes, with 873 nodes and 4181 edges. The path was length **258.70. The k was set to 20** for both the sampling/learning stage and the search stage. Here is the constructed graph and path:



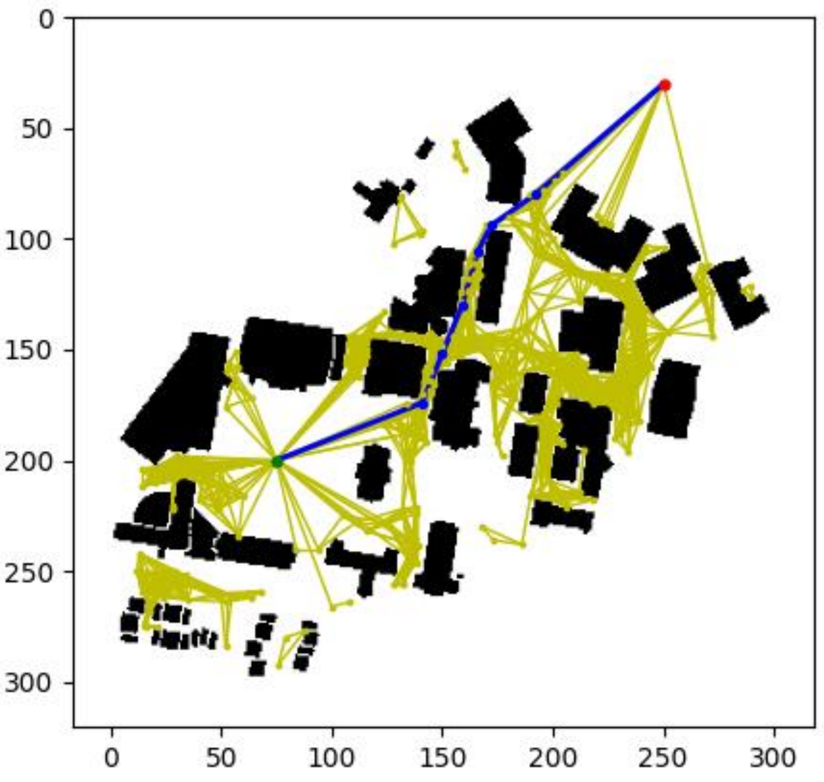
As stated previously, the random sampling can perform much better, or much worse than the uniform sampling, depending on the random samples selected. The random sampling consisted of 851 nodes, with 4796 edges. The path was length **314.27**, considerably worse than uniform sampling. The k-value that worked best actually was the same as for the uniform distribution, ***k*=20** for both the sampling and search phase. I decided to include this iteration of random sampling to show the inconsistency with this method.



The Gaussian sampling performed very well, although only slightly surpassing the uniform sampling in terms of path length. It resulted in 390 nodes, with 1430 edges, and a path length of **258.07**. However, the number of nodes and edges is far, far fewer than either previous two methods. The optimal **k=20** for sampling and **k=80** for the search phase. Furthermore, the Gaussian process requires a **scale,** which was found to work best at **12**. This determines the scope of gaussian distribution about each randomly selected initial point.



The final method, bridge sampling, performed the best. It resulted in 313 nodes and 2245 edges, with a path length of **257.88**. This shows how it had the optimal path compared to the other methods. However, it had fewer nodes, yet interestingly, **more** edges. This is likely due to the unnecessary edges added in-between the high-obstacle area in the east. Finally, the best k-value for this method was **k=25** for sampling and **k=100** for searching. The Gaussian distribution was set to find points at **scale = 23**. With a different map, these values would likely need to be adjusted. Furthermore, these values could be learned or iterated automatically to find the optimal value of **k and scale**.


0
50
100
150
200
250
300
0
50
100
150
200
250
300