RoboticsLab3

ras70

October 2017

This document explores the performance of different probabilistic planners implemented in Klamp't. More specifically, the performance of a probabilistic road map planner (PRM) and rapidly expanding random trees (RRT) for the setup given in *Lab3b* is analyzed.

1 Part A

Planner Variability

To experiment with planner variability I ran 20 trials of the PRM planner (0.36 obstacle radius and knn=10). The results are detailed below.

Trial	Samples*	Edges connected	Time	Milestones used in path
1	30	56	0.0627	9
2	32	62	0.0704	9
3	34	66	0.0739	7
3	28	52	0.0626	6
5	24	46	0.0739	6
6	26	50	0.0584	10
7	18	34	0.0520	8
8	28	54	0.0727	4
9	33	64	0.0643	9
10	30	58	0.0650	8
11	35	68	0.0786	8
12	35	64	0.0690	7
13	36	70	0.0663	7
14	28	54	0.0603	9
15	57	112	0.0987	8
16	34	66	0.0694	8
17	29	56	0.0714	5
18	27	52	0.0650	8
19	24	46	0.0600	8
20	30	58	0.0585	9

* Feasible Milestones sampled

From the above data it is clear the variance for each performance metric is relatively low. Each of the given values fall within a predictable range. This being said, Trial 15 was particularly different than all of the others. This trial took multiple iterations to find a solution, which explains why more samples were taken and why it took a longer time to find a path. For completeness, the sample variance calculations are given below[1]:

Measurement	Mean	Sample Variance	Standard Deviation
Feasible Milestones Sampled	30.9	57.78	7.6
Edges Connected	59.5	229	15.1
Time	0.0677	9.5e-9	9.75e-9
Connected milestones in path	7.6	2.24	1.49

Time and connected path milestones were particularly invariant over the 20 trials. The number of milestones used in the path did not vary from 7 or 8 often. This makes intuitive sense as there is certainly a threshold necessary to get around the circular obstacle, which is why the value is never lower than 4 (the goal and start milestones plus two additional milestones).

Path variance

I expected the path to go *over* the obstacle approximately half the time and *under* the other half. To my surprise the path went below the circular obstacle 18 out of the 20 trials. Trial 4 and Trial 7 were the only two trials in which the path went above the obstacle. I think this strange behavior occurred because the start location is more than half way up the obstacle (in the y direction). Therefore, there is slightly more clearance to build a path in the negative y direction, below the obstacle.

For the most part the paths were fairly similar. Most of the paths first went downward (negative y), then right (positive x), then up (positive y) towards the goal. Almost all of these paths proceeded towards the goal in a manner that did not have a backward step. In other words, at each step positive progress was made around the obstacle *or* towards the goal directly. There was one exception to this, in Trial 20, where the ball is under the obstacle and travels in the negative x direction (Fig. 1).

As expected the curvature between paths varied. This was related to the number of milestones in the path, as more milestones related to a more curved path. Additionally, although the paths followed the same general shape, the location of the milestones in the path were fairly variant. Sometimes the milestones migrated to the corners and sometimes they were near the boundary of the obstacle. Additionally, the distance between the milestones in the path varied quite significantly. This behavior is expected of a probabilistic planner, in which the milestones are samples at random.

[1] http://www.alcula.com/calculators/statistics/variance/

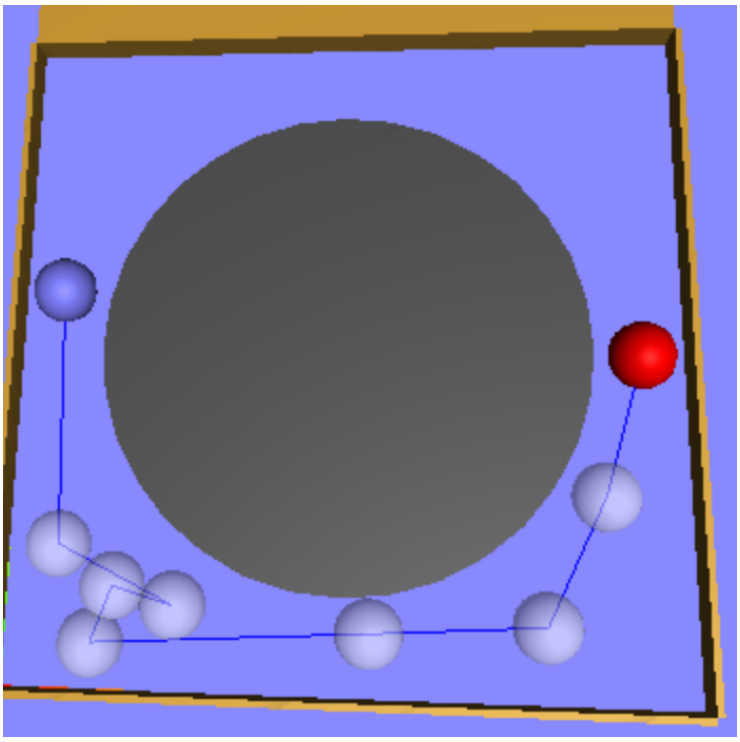

Figure 1: Trial 20 - Backward progress

Altering Parameters

Before the performance of PRM and RRT could be compared the performance of each was determined for different parameters. The performance of the PRM for the original obstacle size (0.36), max plan time of 0.5, and max iterations of 100 is given below.

PRM

The PRM planner samples the space at each iteration. It checks each of these samples for collisions. If these samples are in free space they are kept and called milestones, if not they are discarded. Then the PRM algorithm connects these milestones (including the start and end as milestones) to a collection of nearest neighbors, as long as the path between them is collision free (completely in free space). Once this graph is built the PRM uses a graph searching algorithm to find a path from start to goal.

knn

The *knn* parameter denotes how many nearest neighbors are connected to

a given vertex after sampling. The results of running the simulation are given below:

knn	Samples*	Edges connected	Total time	Milestones used in path
2	32	62	0.0662	8
5	29	56	0.0721	5
10	31	60	0.0708	5
40	37	72	0.0741	7
60	28	54	0.0621	8
100	25	48	0.0626	6

* Feasible Milestones sampled

A value of *knn*=1 was tried. However, the simulation took too long to get any data. To explore this case further I turned on the draw road map feature. After this I was able to see that the road map was growing extremely slow, which explains why the simulation was taking so long. This makes sense since for a *knn* value of one each miletsone is only collected to its nearest neighbor.

For *knn* values from 2 to 100 the feasible milestones sampled, edges connected, and milestones used in path have no significant observable patterns. Additionally, there doesn't appear to be much variation in the total time of simulation for the different values of *knn*.

At first it might be surprising that the performance of the PRM planner is similar for such a wide range of *knn* values. However, the configuration space has relatively low visibility. In other words, given the large circular obstacle it is difficult to move large distances in the configuration space without colliding with the obstacle. Connecting a milestone to a large number of its nearest neighbors would add larger paths into the PRM planner. However, because of the low visibility it is likely a lot of these longer paths are in collision and thus do not help increase performance. This phenomena is observed in the edges connected. This value does not increase as expected with increasing *knn* do to the low visibility of the space.

From Figure 2 it is clear that there is relatively no observable pattern in the efficiency parameters for *knn* with an obstacle size of 0.36. By looking at the table for different *knn* values it is clear that total time follows a similar patter, rising slightly at *knn* equals 40 before falling again. It is not surprising that the number of milestone samples and edges are correlated to the total time of the simulation run. More sampled milestones and edges correspond to more iterations of the algorithm, hence a longer running time.

Changing obstacle radius

I scaled down the obstacle radius to 0.2 and repeated the process above. Below are my findings.

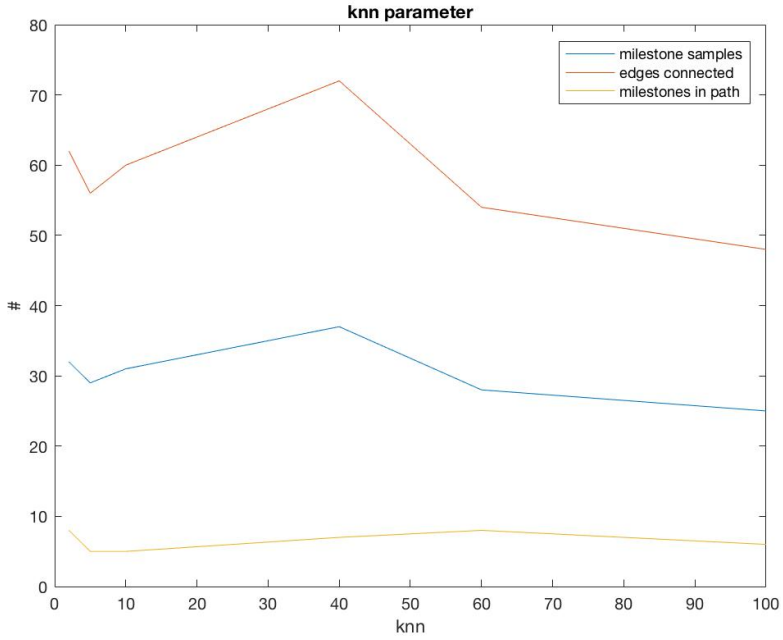

Figure 2: Efficiency Parameters for knn (obstacle radius = 0.36) - No observable pattern

knn	Samples*	Edges connected	Total time	Milestones used in path
2	722	140	0.125	6
5	66	130	0.119	4
10	61	120	0.124	4
40	66	130	0.106	8
60	70	138	0.116	6
100	72	142	0.127	6

* Feasible Milestones sampled

Like when the radius was 0.36 there is no observable pattern in the recorded parameters. However, there are interesting insights that can be gained from comparing the results *between* the trials when the radius was 0.36 and when it was 0.20. With a smaller obstacle the number of feasible milestones, edges connected, and total time increased, while milestones used in paths decreased slightly. This is a result of there being more free space with a smaller obstacle. With more free space the likelihood that a sample is in collision drops and more samples are kept. This leads to more milestones, which leads to more edges, which leads to a larger graph, which leads to a longer running time (to traverse a larger graph).

It is also important to note that as the size of the obstacle decreased the rigidness of the path became more noticeable. The path did not curve around the obstacle as desired, but rather took longer, rigid steps in the free space (refernece Fig. 3.

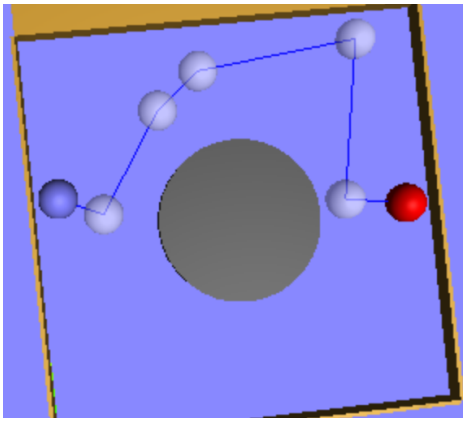

Figure 3: PRM path for small obstacle

Connection Threshold

The *connection threshold* parameter denotes the maximum distance at which neighbors are connected (if *knn* is set to zero). The results of running the simulation are given below:

Conn. Threshold	Samples*	Edges connected	Time	Path Milestones
0.1	228	454	0.2685	21
0.2	103	204	0.1997	14
0.3	33	62	0.0911	8
0.5	30	58	0.1120	6
0.7	28	54	0.1328	4
0.9	26	50	0.1137	6

* Feasible Milestones sampled

From the table above a couple of patterns can be detected. First, it appears the number of milestones in the path decrease as the connection threshold increases. This is intuitive, as more milestones are needed in the path if the distance between milestones is more likely to be smaller (with a smaller connection threshold). The first two simulations (connection threshold = 0.1, 0.2) took multiple iterations. As expected the time per simulation decreased as the connection threshold increased. However, it appears that as the connection threshold is approximately 0.5 and higher the time of simulation begins to

plateau. This is linked to the dimensions of the free space, which is 1x1. With connection thresholds greater than 0.5 it only takes around 6 milestones to get around the obstacle. Therefore, increasing the threshold past this point doesn't particularly increase the time performance.

Lastly, it appears that the *knn* parameters has very little impact of the space of the path.

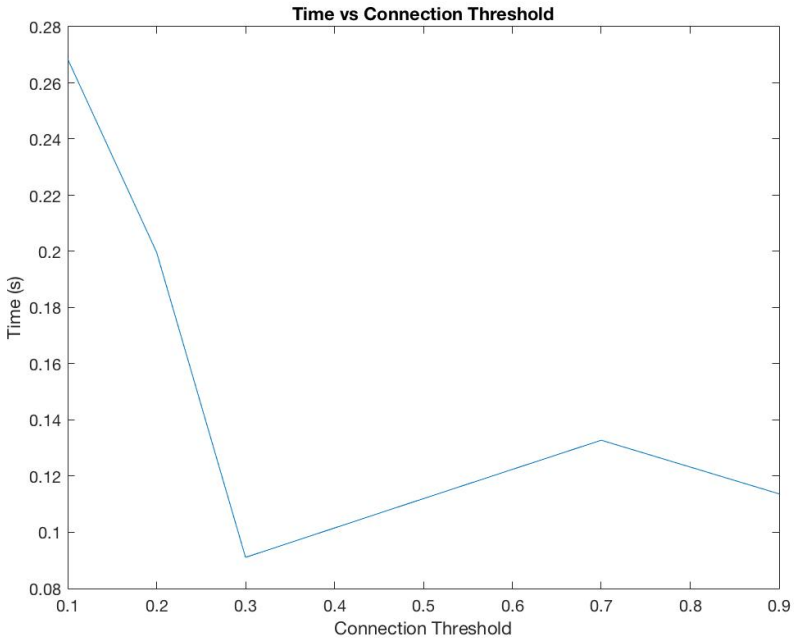

Figure 4: Time vs Connection Threshold

Additionally, it appears that runs with smaller connection thresholds find a slightly more optimal path (in terms of overall distance). This is likely because paths generated with lower connection thresholds have a higher likelihood of curvature. A curved path better fits the free space, given the circular obstacle. This is best pictured in the two figures below.

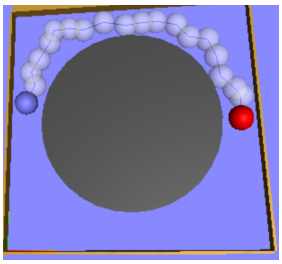

Figure 5: Connection threshold = 0.1 (r=0.36)

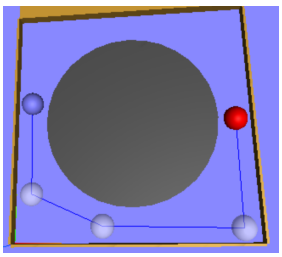

Figure 6: Connection threshold = 0.9 (r=0.36)

Changing obstacle radius

Conn. Threshold	Samples*	Edges connected	Time	Path Milestones
0.1	176	350	0.219	25
0.2	58	114	0.125	10
0.3	60	118	0.185	8
0.5	64	126	0.321	4
0.7	70	138	0.402	4
0.9	56	110	0.369	4

* Feasible Milestones sampled

The pattern observed was similar with a smaller obstacle radius. Path feasible sampled milestones, connected edges, and path milestones all decreased as the connection threshold was increased, until the values began to plateau. For large connection threshold values the path milestones did not deviate above 4 (goal, start, and two other milestones).

A similar phenomenon occurred as when the radius of the obstacle was decreased when analyzing *knn*. The decreasing of the obstacle radius revealed the crudeness of the path chosen by PRM, especially for large connection threshold values.

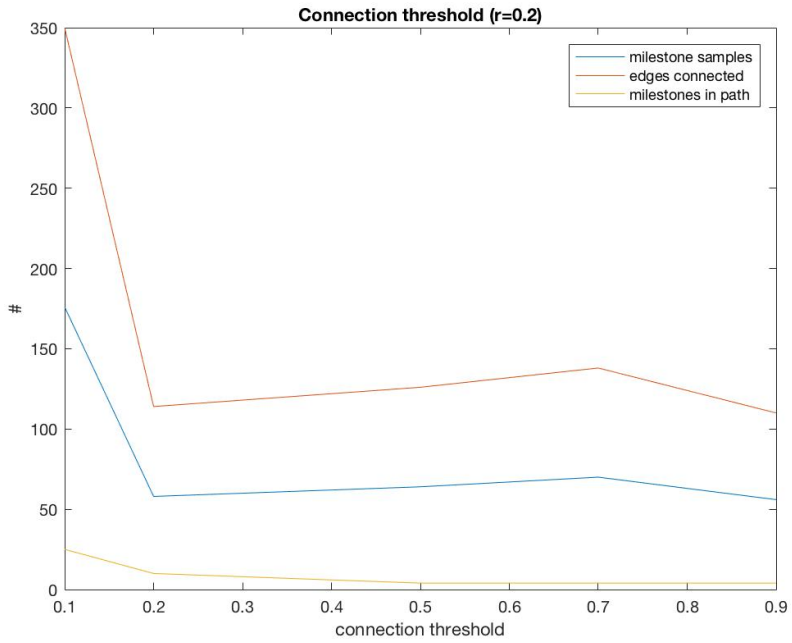

Figure 7: Connection threshold values (r=0.2)

RRT

The rapidly expanding random tree was set to be bidirectional, meaning it built a tree structure from the start and goal simultaneously. The perturbation constant of the planner was altered, which changed the distance at which the next random point in the tree is selected.

Perturbation Threshold

The perturbation constant was varied from 0.01 to 1.0 (with an obstacle size of 0.36 and connection threshold equal to 0.1).

Pert. Threshold	Samples*	Edges connected	Time	Path Milestones
0.01	1114	2226	0.267	174
0.05	181	360	0.104	35
0.1	147	292	0.108	19
0.4	117	232	0.117	12
0.9	221	440	0.174	17

* Feasible Milestones sampled

From the above data it is clear that generally the number of samples, edges

connect, and milestones in the path decline as the perturbation threshold is increased. This is particularly true as the perturbation threshold is increased from 0.01 up to 1. After this point the performance of the RRT seems to level off (Fig. 8). Performance time, which is likely linked to the above performance variables, follows a similar trend.

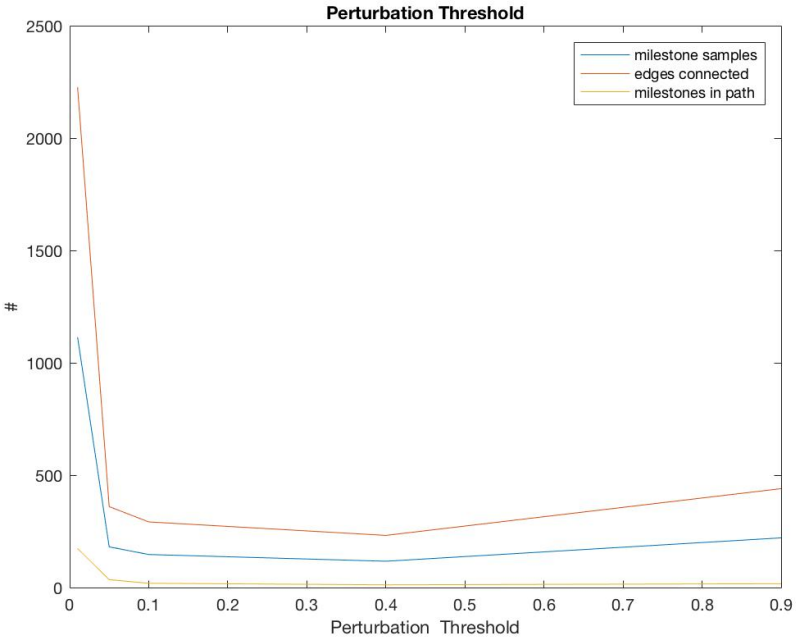

Figure 8: Perturbation performance

As expected a more refined perturbation constant gives the path a likelihood of having more curvature and following the boundary of the obstacle more closely, which in this case is a more optimal (distance minimizing) curve. This is shown below through two images, one with a perturbation threshold of 0.9 and one with a threshold of 0.01.

Varying obstacle size

As expected decreasing the size of the obstacle increased the efficiency of the RRT planner. The RRT planner was able to find a path quicker because it was less likely that the tree would hit the obstacle. However, I would not say the tree was any more optimal after decreasing the size of the obstacle. It still did not follow the shape of the obstacle well.

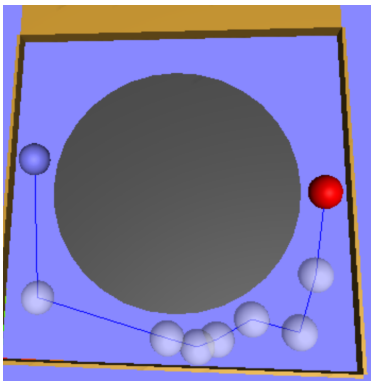

Figure 9: Perturbation threshold = 0.9 - a more rigid path

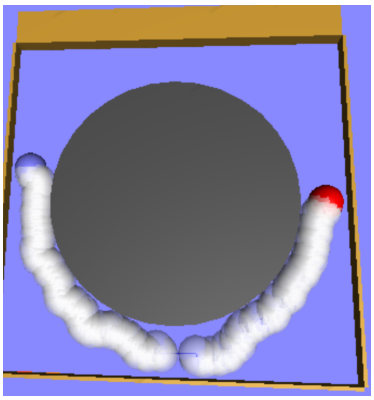

Figure 10: Perturbation threshold = 0.01 - a smoother path

PRM vs RRT

To compare the performance of PRM and RRT I first scaled the size of the obstacle down to 0.2. I set the PRM planner to have a *knn* value of 10 and the RRT planner to have a perturbation threshold of 0.4. With this parameters it appeared the PRM had better performance. The two had similar time performance, around 0.1 seconds each. However, the PRM planner used far less milestones in its path, usually 3-6 milestones. The RRT planner used 10-20 milestones in its path.

Using less milestones in the chosen path doesn't necessarily lead to a shorter, more optimal path. Actually, the result is usually the opposite. Sometimes more milestones can lead to more complex paths that map the space better. However, in this case the random tree often used milestones that backtracked, while the PRM went straight towards the goal. Here, I use the term *straight* relatively. By this I mean the PRM planner path rarely backtracked and made positive

progress at each step.

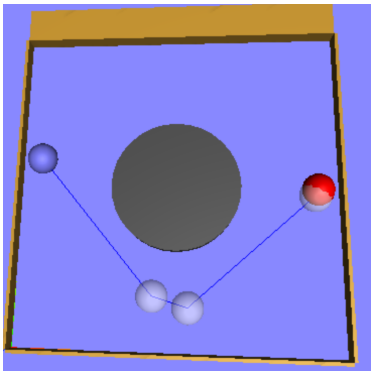

Figure 11: PRM performance with obstacle radius = 0.2

The worst case of the PRM planner took the ball to 2 of the 4 corners as it progressed towards the goal. However, the ball rarely backtracked.

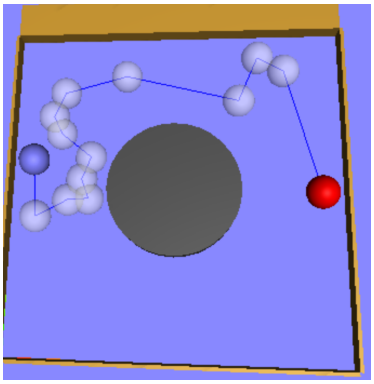

Figure 12: RRT performance with obstacle radius = 0.2

In Figure 12 the path takes a convoluted s shape when a simple c shape is needed. This leads to a longer, less optimal path. Although the PRM path is rigid it more closely follows the c shape need (Fig. 11.

Given Figures 11 and 12 I concluded that the PRM planner was slightly better on average for a smaller obstacle radius.

Next, I scaled the obstacle size up to 0.38. This left very little free space. When I did this I observed that PRM and RRT created similar paths. This is to be expected, since there is little room in the free space for the paths to deviate. Thus, judging performance in the case of a larger obstacle is more closely tied to the efficiency of the scheme, not the efficiency of the path, because the paths showed little variation. Figures 13 and 14 show the similarity in paths selected by the two planner algorithms.

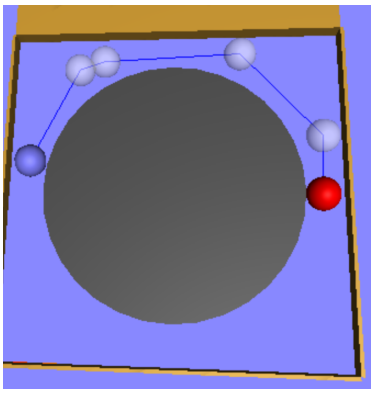

Figure 13: PRM path with larger obstacle (radius = 0.38)

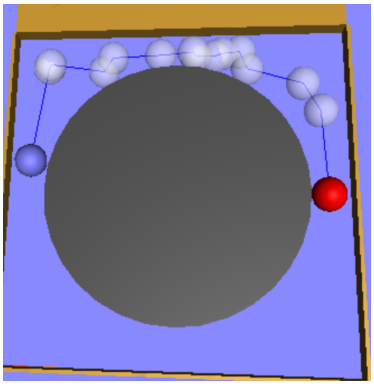

Figure 14: RRT path with larger obstacle (radius = 0.38)

The PRM planner was more efficient in finding the path. On average the PRM planner took around 0.05-0.08 seconds to find a path. Conversely the RRT planner took around 0.17 to 0.20 seconds. I think the PRM planners is able to find a path in shorter time because it only needs to sample a small amount of points or below the obstacle to find a feasible path.

Given the efficiency differences I would conclude that the PRM was more optimal in this situation.

High quality values

Given the discussion above I would chose the PRM planner. I would chose a *knn* value of 10. This value properly balances the trade off between efficiency and finding an optimally short path.

If I had to use the RRT planner I would suggest a perturbation threshold of 0.05 (for a connection threshold of 0.1). Increasing the perturbation threshold past this point doesn't reap much of a performance increase. The low value of the perturbation threshold (0.05) produces a relatively curved path, which produces a path with a short distance.

Of course the above explanations hold for the general case, when efficiency (time) and length of the path are both a concern. In the length of the path takes key priority I would chose the RRT planner with a small perturbation threshold, such as 0.01. This scheme had the worst efficiency. However, it had the highest likelihood of creating a curved path that followed the boundary of the obstacle.

2 Part B

I altered the start, goal, and obstacles as follows:

Obstacle: Circle(0.6,0.5,0.1), Circle(0.2,0.5,0.15)

Start: (0.5,0.9)

Target: (0.5,0.1)

This configuration has the start at the top-center of the box and the goal in the bottom-center of the box. There are two circular obstacles of varying size in the center of the square (reference Fig. 15).

With this configuration the optimized PRM shortcut planner is not guaranteed to converge to the shortest path. This is shown in Figure 15. In this example the probabilistic sampler choses a path to the right of both obstacles. The yellow lines in Figure 15 indicate the road map chosen. Once the algorithm then attempts to shortcut creating the optimal path out of the road map is impossible.

The optimal path is through the middle of the two obstacles, to the left of the rightmost obstacle. From the figure it is clear that the robot can fit in this gap. However, to prove this the math is given below:

$$\text{gapSize} = (x_2 - r_2) - (x_1 + r_1)$$

where x indicates the x-location of the obstacle's centers and r gives their radii

$$\text{gapSize} = (.6 - .1) - (.2 + .15) = .15 > 0.1 = \text{two times robot radius}$$

The *rrt** planner was chosen to show the optimal path. This planner did converge to the optimal path, shown below:

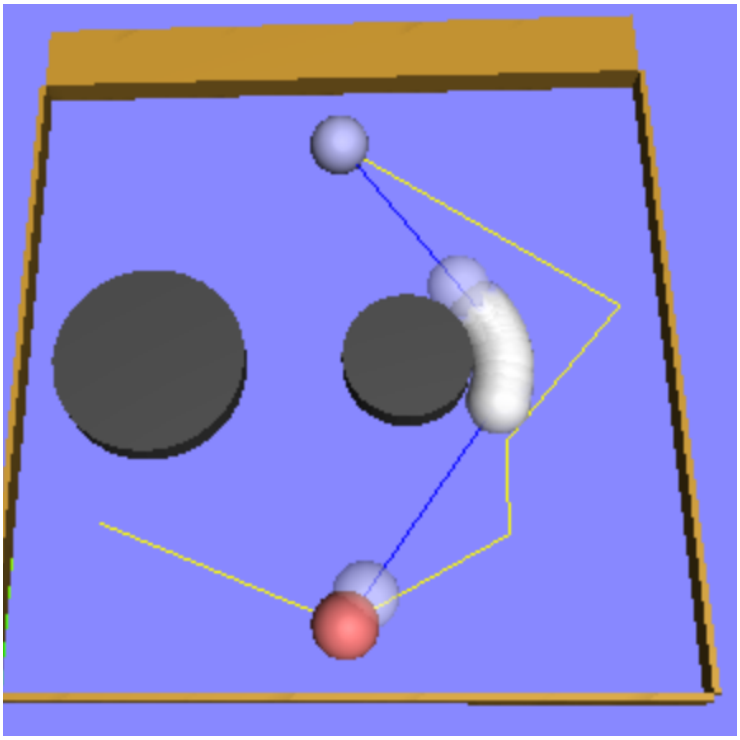

Figure 15: PRM optimized planner with shortcuts fails to converge to optimal path

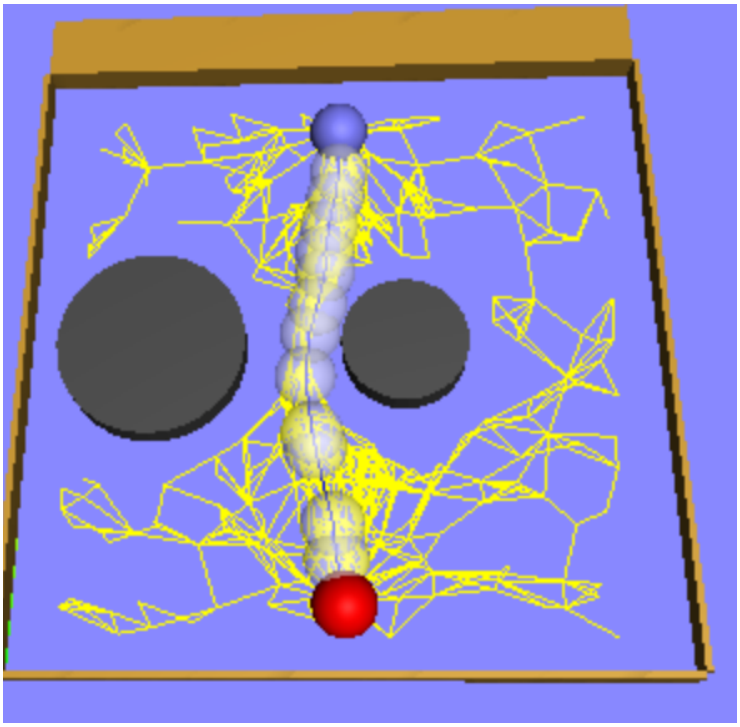

Figure 16: rrt* optimized planner converges to optimal path