Joe Miceli

13Nov2020

ASEN 5519 – Algorithmic Motion Planning

Fall 2020

Homework 4

Exercise 1

a) Path: Start → C → K → Goal
Path length: 4
Iterations: 14

b) My A* algorithm can be turned into Dijkstra's by modifying the calculation of priority for each node. An "if" statement can be added to change how priority is computed depending on if Dijkstra is being used or not. If it is, then the heuristic will not be included in the priority.

```
…
if (USING_DIJKSTRA)
        then f = g
else
        f = h+g
…
```

c) Using Dijkstra, the resulting path did not change.
Path: Start → C → K → Goal
Path length: 4
Iterations: 14

d) It appears that Dijkstra's algorithm did not perform any better than A* on this particular graph. This is not surprising considering all nodes were processed in the A* algorithm. Typically, I would expect Dijkstra to explore more of the graph than A* but in this case, there are no more nodes to explore than what is explored by A*.

   i) To search a large graph for the shortest path between v_{start} and v_{goal}, I would use A*. This is because A* uses a heuristic (and if the heuristic is admissible), so it can intelligently expand nodes. It only expands nodes that improve the path to goal. It can be more efficient at finding a single path than Dijkstra.

   ii) To search a large graph for the shortest path between v_{start} and every $v \in V$, I would use Dijkstra's algorithm. This is because Dijkstra expands the node closes to the current node regardless of its distance from the goal. So naturally, this algorithm explores more of the graph. This behavior would be ideal for finding paths to all nodes in the graph.

e) To make this implementation bi-directional, I would essentially duplicate assignment of neighbors. If node B is added as a neighbor of node A, then node A would have to be added as a neighbor to node B. This would increase the number of nodes to evaluate during expansion. However, this implementation would ideally not increase computation time dramatically. These additional nodes would still only get processed one time. If they are on the processed list (as they would have been in the directed graph) they will be ignored during the expansion phase when their neighbor is selected from the open list.

Exercise 2

a) Planning problem from Homework 3 Exercise 2.a is defined as:
2D C-space ($C = \mathbb{R}^2$) with q_{start} = (0, 0), q_{goal} = (10, 0) and two square obstacles with side length of 1 centered at (4, 1) and (7, −1)

The bounds of the workspace are defined as:
x ∈ [−1, 11] and y ∈ [−3, 3]

i) Figure 1 shows the nodes generated during the graph generation phase of the PRM planner. While the obstacles are not explicitly shown in the figure, their outline can roughly be seen from the node placement in the graph. Figure 2 shows the path identified by the A* search algorithm on this graph. It is apparent from the number of nodes in the final path that some back pointers were incorrectly assigned during the A* search. Unfortunately, I was not able to identify this bug. However, the behavior and shape of the path is what would be expected from the given planning problem.

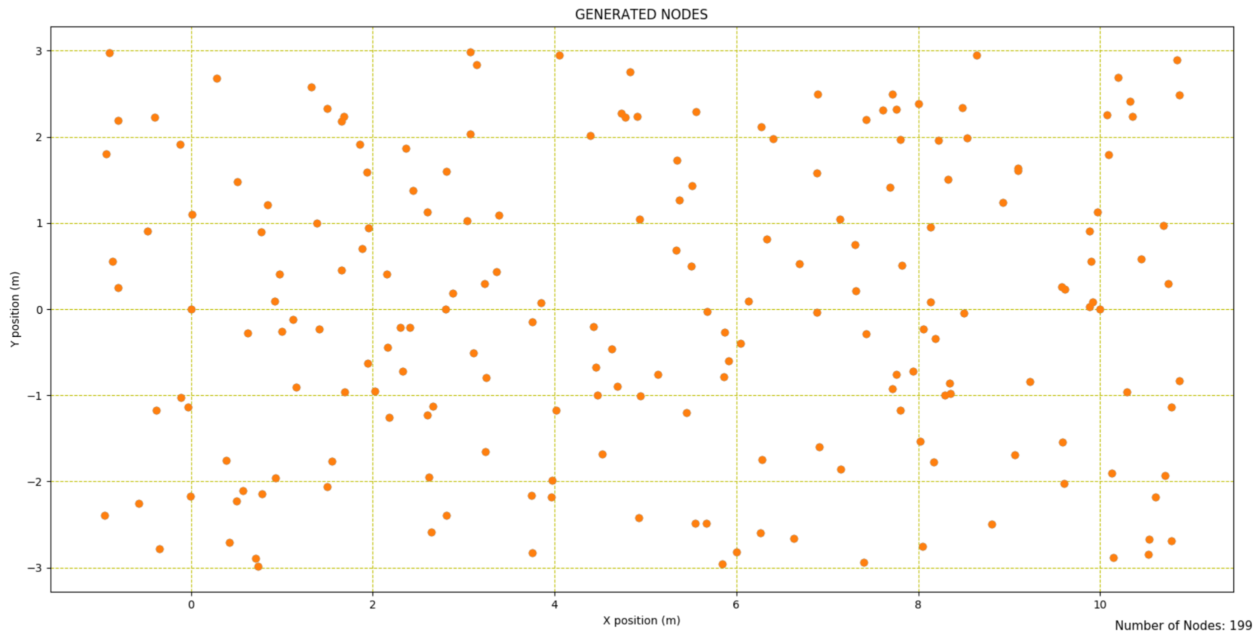

Figure 1 Nodes generated by PRM planner with n = 200 and r = 1

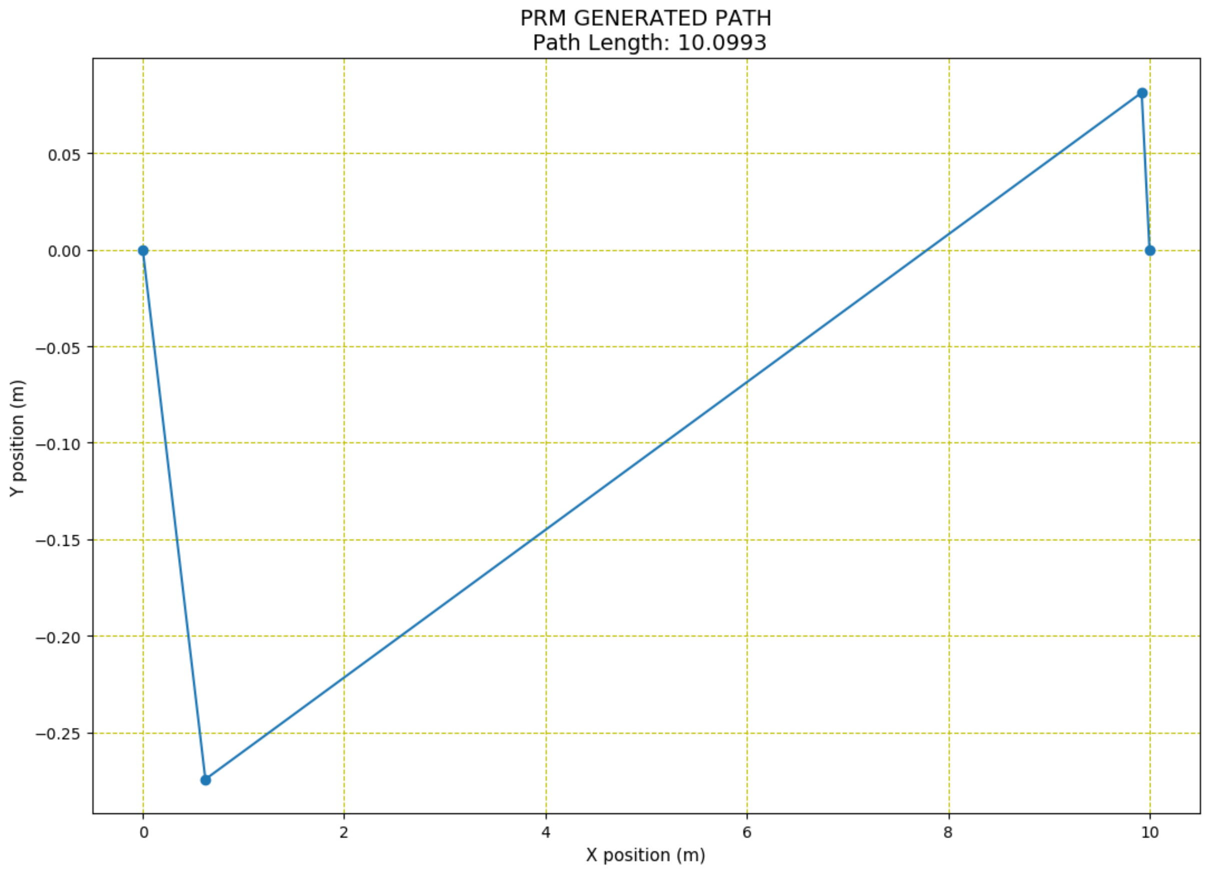

Figure 2 Path generated by PRM planner with an apparent bug in the back-pointer assignment during A* search

ii) In general, increasing the number of nodes increased the execution time more significantly than increasing the radius of the local neighborhood. This is not what I expected. Increasing the radius of the local neighborhood requires more local path planning (and more discretizations, a computationally expensive task). It also increases the number of nodes to evaluate during the expansion portion of A* search. Increasing the number of nodes in the graph can also have these effects but that change is not guaranteed to affect every node in the graph in the same way of increasing the radius of the local neighborhood.

Strangely, my algorithm performed very slowly. Execution times, path lengths, and valid solutions are shown in the following tables for various PRM configurations (n, r). It was not feasible for me to run 100 time due to the speed of my algorithm and therefore did not have enough data to generate box plots. To add, the path length did not change at all between runs of the same configuration.

It should be noted that for n = 200, r = 0.5 and n = 500, r = 200, no solution was ever found. In both cases, the program exited during the query phase. This is likely because the algorithm was unable to attach n_{init} and n_{goal} (the initial and goal nodes) to the graph. Also, strangely, my algorithm was so inefficient that I was never able to find solutions for n = 500, r = 1.5 or n = 500, r = 2. In both cases, I let the algorithm run for greater than 10 min and it never exited.

Table 1 PRM Metrics for n = 200, r = 1			
Run	Computation Time (us)	Path Length	Valid Solutions
1	74000153	10.0993	1
2	66772540	10.0993	1
3	73092853	10.0993	1
4	73760834	10.0993	1
5	71698313	10.0993	1
AVG:	71864938.6	10.0993	1

Table 2 PRM Metrics for n = 200, r =1.5			
Run	Computation Time (us)	Path Length	Valid Solutions
1	81221407	10.0993	1
2	77101402	10.0993	1
3	77642867	10.0993	1
4	79073730	10.0993	1
5	82879317	10.0993	1
AVG:	79583744.6	10.0993	1

Table 3 PRM Metrics for n = 200, r = 2			
Run	Computation Time (us)	Path Length	Valid Solutions
1	91131781	10.0993	1
2	90035395	10.0993	1
3	78168086	10.0993	1
4	74859090	10.0993	1
5	78462911	10.0993	1
AVG:	82531452.6	10.0993	1

Table 4 PRM Metrics for n = 500, r = 1			
Run	Computation Time (us)	Path Length	Valid Solutions
1	213908510	10.5493	1
2	195687604	10.5493	1
3	196366926	10.5493	1
4	193832686	10.5493	1
5	188946926	10.5493	1
AVG	197748530.4	10.5493	1

iii) Based on the tables shown above, n=200, r=1 performed best in the metrics of interest when compared to the other configurations. All (with exception of n=200, r=0.5) were able to find paths to the goal that were extremely similar in length. Therefore, I would use computation time as the driving factor to select the inputs for the PRM planner.

iv) I was not able to implement path smoothing.

b) Workspace 1 of Homework 1 Exercise 7 was defined as:
qstart = (0, 0), qgoal = (10, 10), and obstacles
$WO = \cup_{i=1}^{5} \mathrm{WO_i}$, where each $\mathrm{WO_i}$ is a rectangle with vertices:
$\mathrm{WO_1}$: {(1, 1), (2, 1), (2, 5), (1, 5)}
$\mathrm{WO_2}$: {(3, 4), (4, 4), (4, 12), (3, 12)}
$\mathrm{WO_3}$: {(3, 12), (12, 12), (12, 13), (3, 13)}
$\mathrm{WO_4}$: {(12, 5), (13, 5), (13, 13), (12, 13)}
$\mathrm{WO_5}$: {(6, 5), (12, 5), (12, 6), (6, 6)}

The bounds of the workspace 1 are defined as:
x ∈ [−1, 13] and y ∈ [−1, 13]

Workspace 2 of Homework 1 Exercise 7 was defined as:
qstart = (0, 0), qgoal = (35, 0), and obstacles
$WO = \cup_{i=1}^{9} \mathrm{WO_i}$, where each $\mathrm{WO_i}$ is a rectangle with vertices:
$\mathrm{WO_1}$: {(−6, −6), (25, −6), (25, −5), (−6, −5)}
$\mathrm{WO_2}$: {(−6, 5), (30, 5), (30, 6), (−6, 6)}
$\mathrm{WO_3}$: {(−6, −5), (−5, −5), (−5, 5), (−6, 5)}
$\mathrm{WO_4}$: {(4, −5), (5, −5), (5, 1), (4, 1)}
$\mathrm{WO_5}$: {(9, 0), (10, 0), (10, 5), (9, 5)}
$\mathrm{WO_6}$: {(14, −5), (15, −5), (15, 1), (14, 1)}
$\mathrm{WO_7}$: {(19, 0), (20, 0), (20, 5), (19, 5)}
$\mathrm{WO_8}$: {(24, −5), (25, −5), (25, 1), (24, 1)}
$\mathrm{WO_9}$: {(29, 0), (30, 0), (30, 5), (29, 5)}

The bounds of the workspace 1 are defined as:
x ∈ [−1, 13] and y ∈ [−1, 13]

i) The nodes generated for n=200, r=2 are shown in Figure 3. The path found by the PRM planner on workspace1 is shown by figure 4. Again, from the number of nodes in the final path, it is apparent there was a bug during the back-pointer assignment of the A* search algorithm.

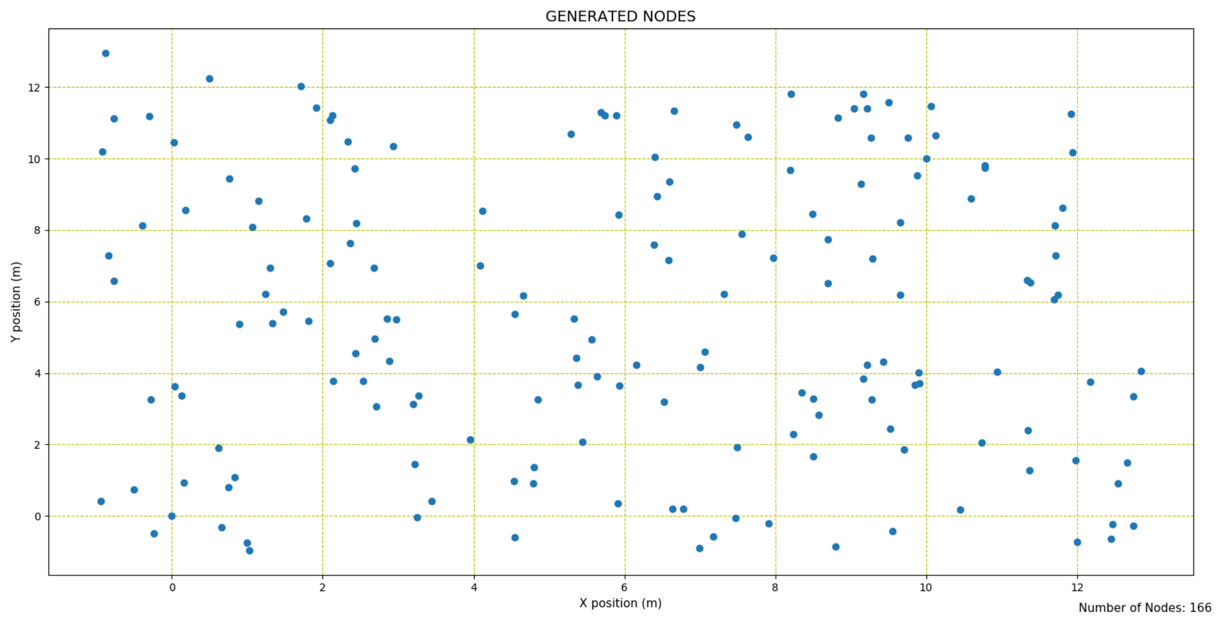

Figure 3 Nodes generated by PRM planner for workspace 1 with n = 200 and r = 2

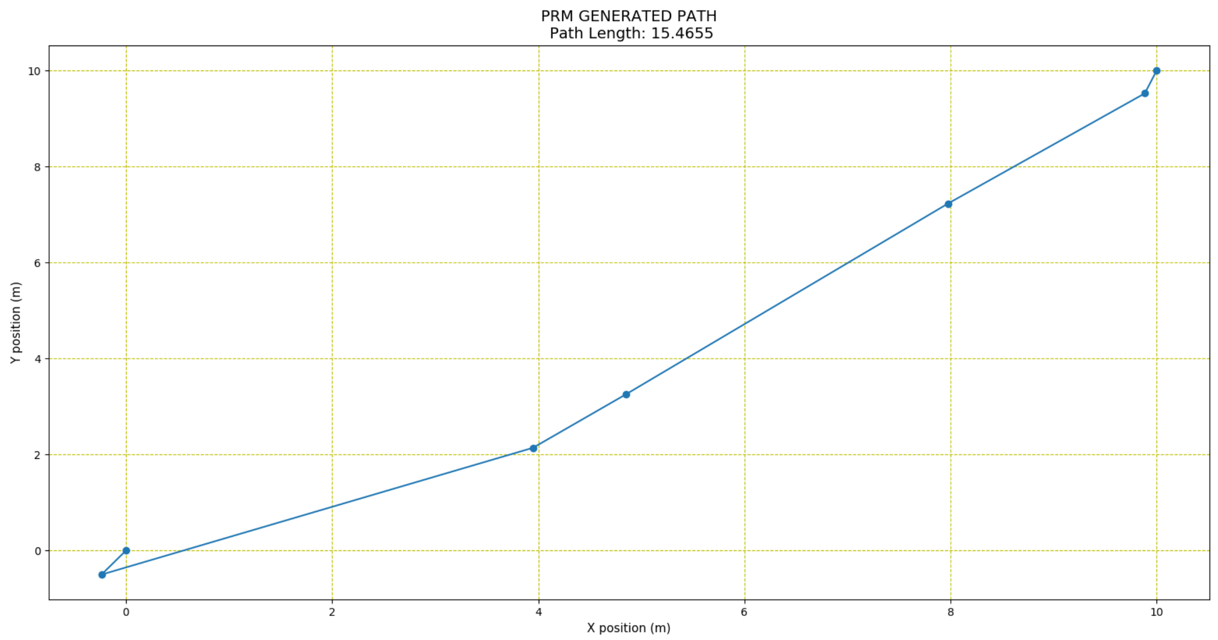

Figure 4 Path generated by PRM planner for workspace 1 with n = 200 and r = 2 with an apparent bug in the back-pointer assignment during A* search

The nodes generated by the PRM planner for workspace 2 are shown in figure 5. The path generated for workspace 2 is shown in figure 6. The path still shows an issue with the assignment of the back-pointer of each node during the A* search portion of the PRM planner. However, this is a reasonable route given the layout of workspace 2.

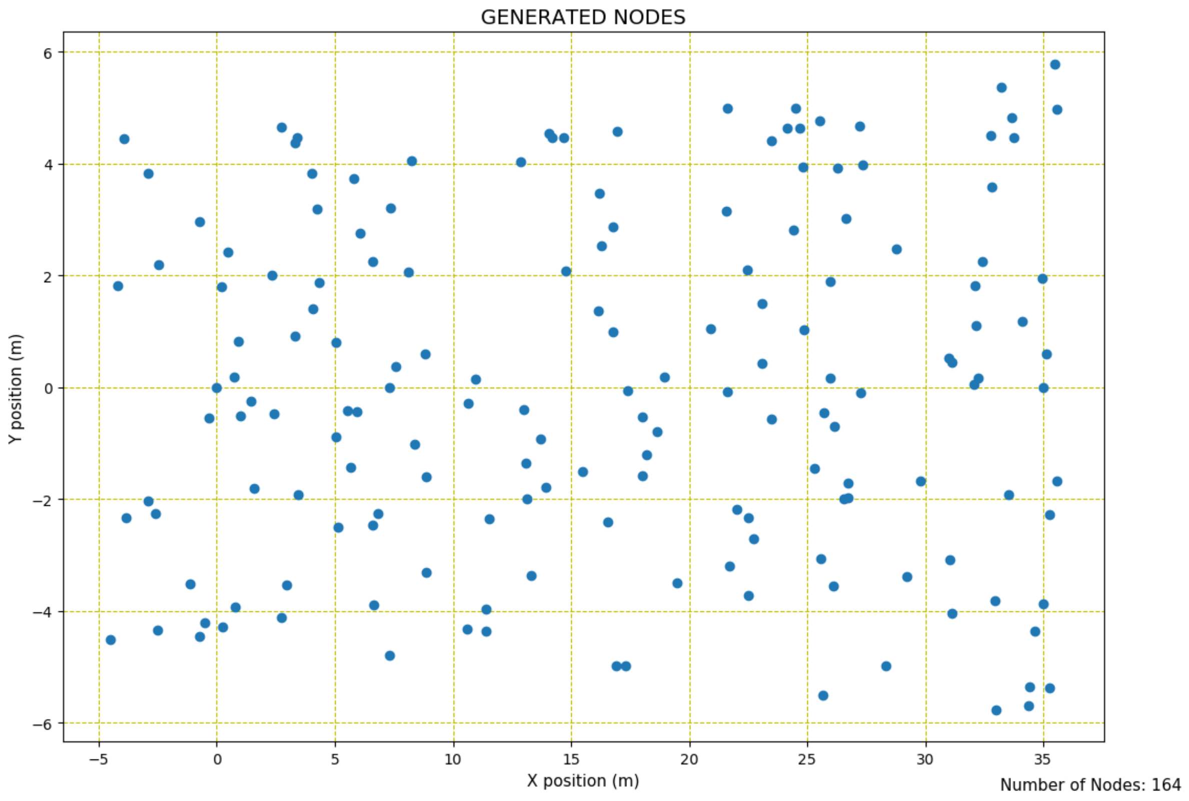

Figure 5 Nodes generated by PRM planner for workspace 2 with n = 200 and r = 2

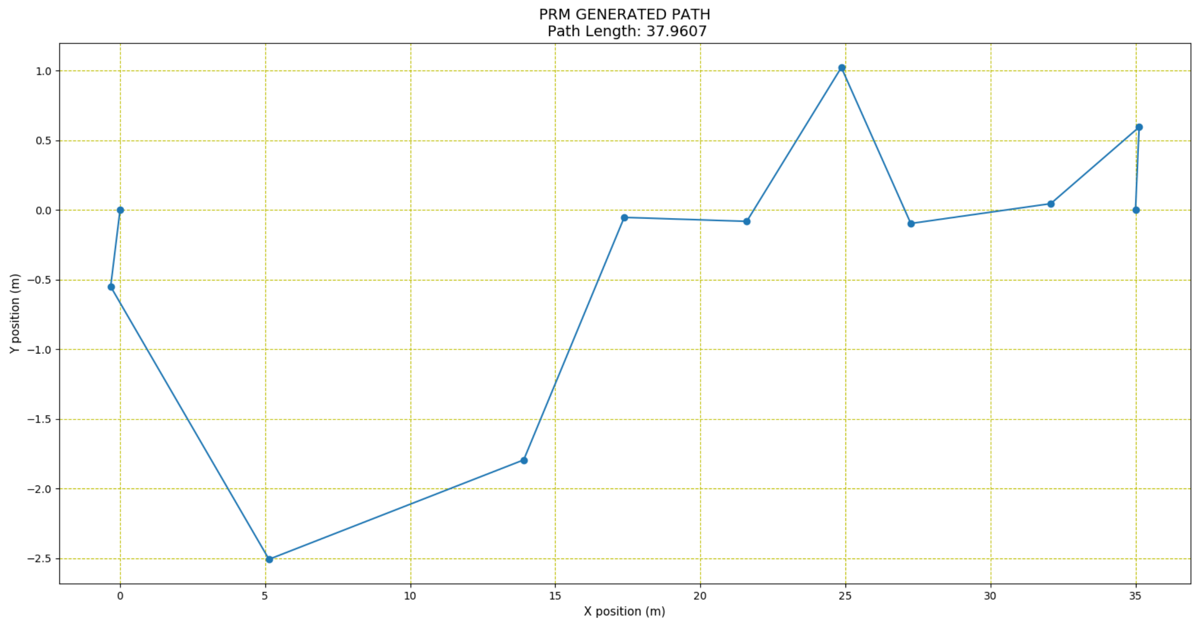

Figure 6 Path generated by PRM planner for workspace 2 with n = 200 and r = 2 with an apparent bug in the back-pointer assignment during A* search

ii) Computational inefficiencies prevented me from properly benchmarking this algorithm. Increasing n dramatically increased computation time, more so than increasing the local neighborhood radius, r. Similar to part a, this result is surprising and likely related to a bug somewhere in the query of the graph (this was by far the longest portion of the algorithm). I was able to gather metrics for some values of n and r but at n = 500, execution time was over 15 minutes. It was not feasible for me to run 100 time due to the speed of my algorithm and therefore did not have enough data to generate box plots. The results I was able to gather are shown in the following tables.
As in part a, the path length does not vary between computations. This may be due to unidentified bug in the random sampling portion of the algorithm. If the sampling was actually *not* random, the path would not vary between runs. In an attempt to avoid this issue, the algorithm was cleaned and rebuilt before each run.

Table 5 PRM Metrics for Workspace 1 with n = 200, r = 1

Run	Computation Time (us)	Path Length	Valid Solutions
1	169981372	15.4655	1
2	140912226	15.4655	1
3	149659863	15.4655	1
4	144097593	15.4655	1
5	146839454	15.4655	1
6	122032228	15.4655	1
7	122122991	15.4655	1
8	122979088	15.4655	1
9	122504901	15.4655	1
AVG:	137903301.8	15.4655	1

Table 6 PRM Metrics for Workspace 1 with n = 200, r = 2

Run	Computation Time (us)	Path Length	Valid Solutions
1	124681210	15.4655	1
2	128360492	15.4655	1
3	144396591	15.4655	1
4	141286881	15.4655	1
5	141564178	15.4655	1
6	143069446	15.4655	1
7	141829012	15.4655	1
8	142623723	15.4655	1
9	142797974	15.4655	1
AVG:	138956611.9	15.4655	1

Table 7 PRM Metrics for Workspace 1 with n = 500, r = 1			
Run	Computation Time (us)	Path Length	Valid Solutions
1	833021327	15.0905	1
2	836566209	15.0905	1
3	910409768	15.0905	1
AVG:	859999101.3	15.0905	1

Table 8 PRM Metrics for Workspace 1 with n = 500, r = 2			
Run	Computation Time (us)	Path Length	Valid Solutions
1	884758229	15.0905	1
2	935073680	15.0905	1
3	915535905	15.0905	1
AVG:	911789271.3	15.0905	1

Beyond n = 500, the computation time became too large to gather metrics for workspace 1. The program ran for greater than 20 minutes and never exited.

The computation time for workspace 2 is much larger than that of workspace 1 for all PRM inputs n and r. This is likely due to the added obstacles in workspace 2. Each iteration of the PRM algorithm requires several collision checks. One of these collision checks is a "path" collision check that involves discretizing a path between local nodes and checking it for collisions with all obstacles. Increasing the number of obstacles increases the amount of check that must be performed on each point of the discretized path.

Table 9 PRM Metrics for Workspace 2 with n = 200, r = 1			
Run	Computation Time (us)	Path Length	Valid Solutions
1	431601836	37.9607	1
2	450175402	37.9607	1
3	447117930	37.9607	1
AVG:	442965056	37.9607	1

Table 10 PRM Metrics for Workspace 2 with n = 200, r = 2			
Run	Computation Time (us)	Path Length	Valid Solutions
1	476166970	37.9607	1
2	459437653	37.9607	1
3	441951395	37.9607	1
AVG:	459185339.3	37.9607	1

Beyond n = 200, the computation time became too large to gather metrics for workspace 2. The program ran for greater than 25 minutes and never exited.

iii) In both workspaces, the PRM inputs n = 200 and r = 1 were optimal. The path produced for all variations of n and r were nearly equal in length but took significantly longer to compute as n and r increased. The added computation time added no benefit in the resulting path length therefore, the inputs that produced the shortest computation time (n = 200 and r = 1) should be used for both workspaces.
iv) I was not able to implement path smoothing on this problem

c) This implementation of a PRM planner needs to change only slightly to answer the planning problem in Exercise 4 of Homework 3. First, the obstacle representation would have to change. This problem has triangular obstacles as well as rectangular. The obstacle class currently implemented only allows for rectangular obstacles. To add, the current implementation assumes configuration space and workspace are equivalent. To properly plan this problem, obstacles would have to be mapped to and represented in the configuration space.
Next, the sampling portion of the planner would have to be modified. Current implementation bounds the sampling based on the bounds of the workspace however, in this problem, the PRM planner would have to sample *configurations* of the robotic manipulator. The configuration space of the manipulator is defined by angles so the bounds of the random sampler should account for this.

Exercise 3

I was not able to accomplish this problem.

Exercise 4

While I was not able to implement the GoalBiasRRT planner, I can still comment on the performance of gradient descent with potential function, wavefront planner, and PRM planners. Of the three, (and in my opinion) the gradient descent planner was the simplest to implement. Unfortunately, it was the most difficult to configure. Gradient descent planners offer several configurable parameters that can affect the performance (and even success) of the planner. I found the planner to be extremely sensitive to changes in those parameters and the workspace in which it was planning. I also found the planner often got stuck in local minima. In fact, the gradient descent planner was so sensitive to configuration changes and the workspace layout that it was unable to find a collision free path in the problem.

The wavefront planner took longer to implement but performed much better than the gradient descent planner in the 2D workspaces. It found paths consistently though required longer execution time. This planner would not scale well with increases to the size or dimension of the workspace but would be able to successfully plan in the presence of additional obstacles in 2D. It would even perform well in the presence of narrow passages.

PRM was the most challenging to implement. Due to bugs in my implementation it also performed much slower than the other two planners. However, it did find valid (and optimal) paths to the goal so, if corrected, I anticipate this planner to perform much better than the other two. It is able to operate in higher dimensions and does not suffer from local minima problems.