ASSIGNMENT 2: Canadarm – Continuous Motion Planning

Nick Morris
The University of Queensland

1.1 Configuration space

The configuration space for the problem is the set of all possible robot configurations.

- A configuration is the parameters that uniquely define the position of every point on the robot: q = (q1, q2, … qn)
- [Eex, Eey; A1, A2 ... An; L1, L2 … Ln] makes a configuration
- The forbidden region is made up of the obstacle list
- Free space is the C- space without the forbidden region
- = {x | x is a pose of the robot}

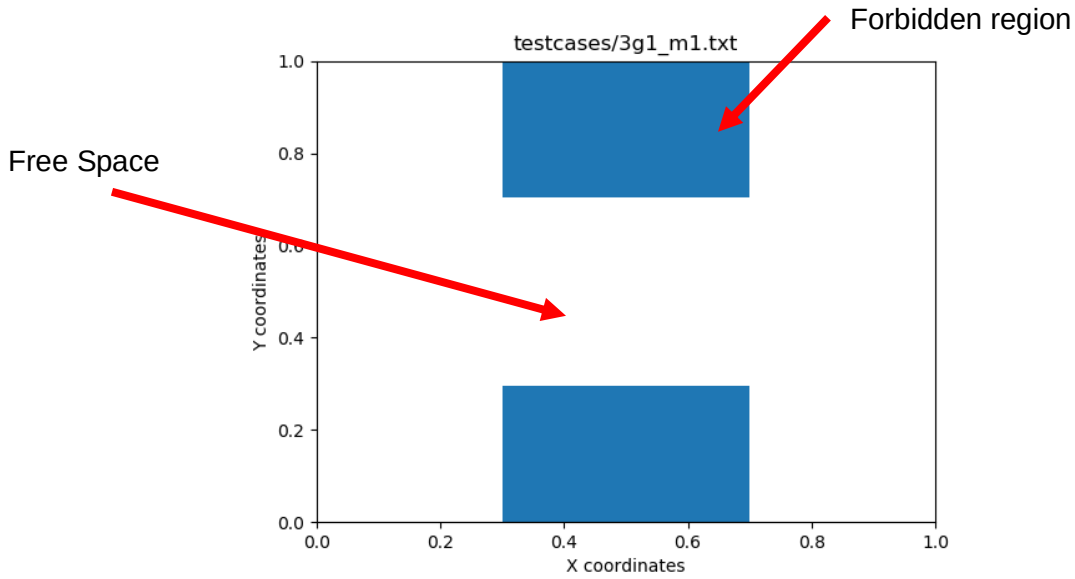


Figure 1: Configuration space

The method used for searching in the continuous search was Probabilistic Road Map (PRM). From the empty graph G, a configuration q is sampled by picking a random arm pose. The random arm pose is checked whether they are inside the min and max conditions, are not inside an obstacle, and do not form any collisions with obstacles or itself (if q -> Q_Free, then add to G). This is repeated until N samples have been created. Each of these configurations become a node to be added to the state graph. Using matplot in python, the EE2 of the robot arm can be plotted for a visual representation.

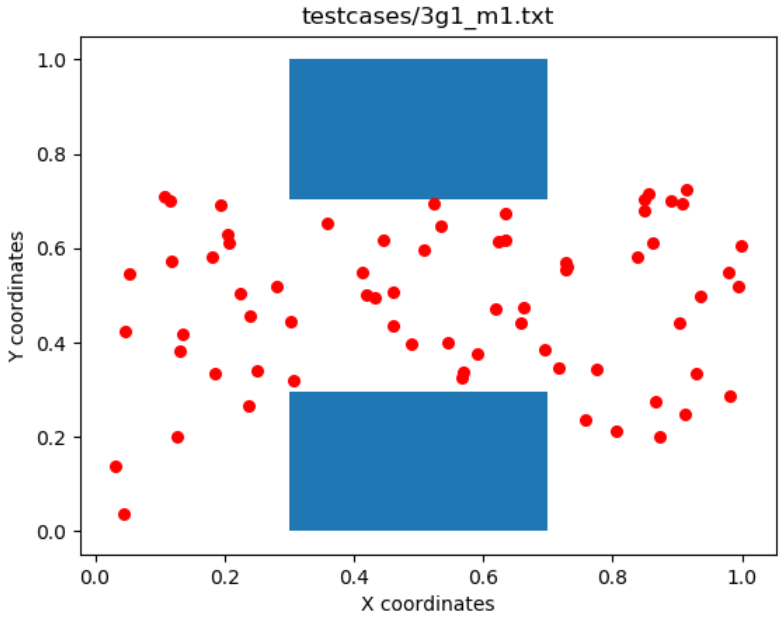


Figure 2: PRM random sampling of points

Additionally, the goal (q_goal) and start (q_start) are added as nodes. The grapple points are also plotted; however, they are only included as nodes if there is more than one grapple point, in which it will be considered an initial goal point.

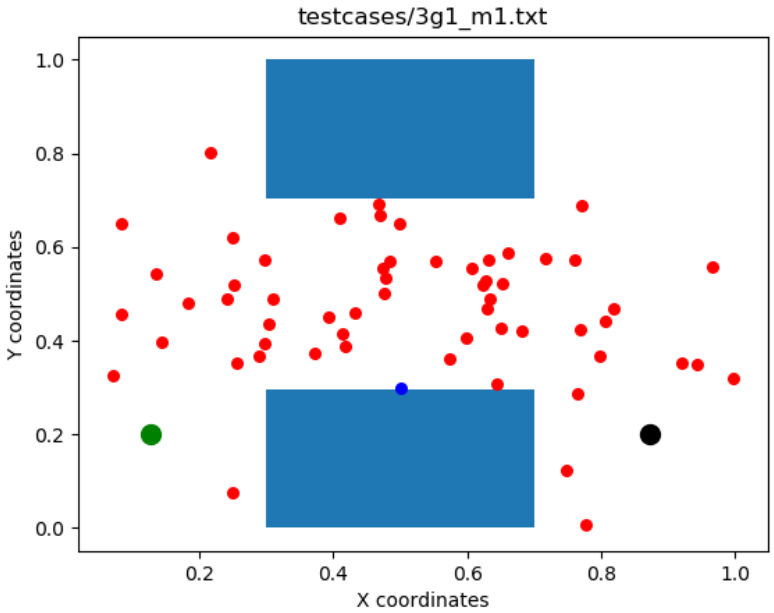


Figure 3: PRM start (green) and goal (black) EE2 positions

To create paths, each node q is linked to its nearest neighbour q'. If there is a free local path, such that there are no collisions, an edge added between q and q'. This is repeated until each node q is connected to its nearest k neighbours.

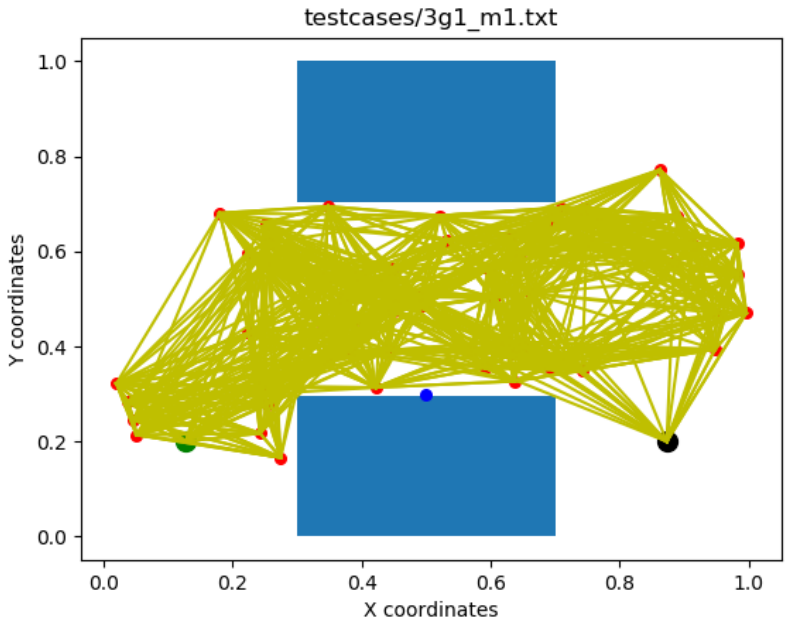


Figure 4: PRM neighbours and possible paths

Using a BFS search algorithm, the path from start to goal can be found:

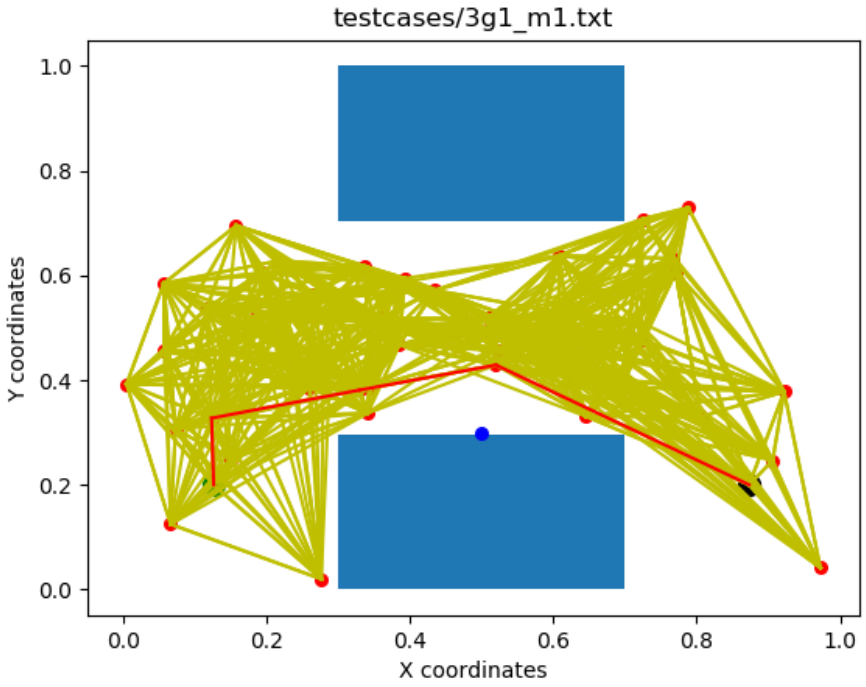


Figure 5: Path found by BFS (red line)

A BFS was used over alternatives such as GBFS or A* search, simply because it found an optimal solution within a small-time frame (e.g. 7.569s for 500 nodes on high level test case) to run the program, which is within the requirements of 1-2 minutes.

2.1 Strategy

2.1 Sampling strategy:

In order to generate random configurations:

- Random angles are generated for number of arm links
- Random lengths are generated for number of arm links between such that $\text{min\_length} \leq L_i \leq \text{max\_length}$
- The position of the grappled end effector.

The configuration is checked whether it is valid, and if it is, it is appended to a list of nodes (More on valid checking in 2.2.3).

From these input values, the position of the un-grappled end effector can be calculated and plotted for a visual representation of the end point.

2.2.2 Connection strategy

The connection strategy involves comparing every valid node q to every other valid node q'. Iterating through the nodes, if the node is not being compared with itself, and there is no collision between the two nodes in a straight line, then the lengths and angles between the two nodes are interpolated. For every ten primitive steps (0.56 degrees or 0.01 unit length) a collision check is performed on the interpolated collision. If there are no collisions between the interpolated points, then the Euclidian distance between the two nodes is calculated and appended to a list of distances. This list is then sorted from smallest to largest distances, and for the first k distances in the list, their corresponding node q' is added as a neighbour for q.

To see the connection strategy used to connect between different grapple points, see section 3.3 (Page 11)

2.2.3 Checking if a configuration is valid

To check whether a configuration is valid, the following checks are applied:

- Check the node is inside the specified min and max
- Check the un-grappled end effector is not inside an obstacle
- Check whether the configuration (arm links in addition to EE) are in collision with obstacle
- The check for collision with itself

To check whether a node is inside the specified min and max, the x and y coordinate points are compared such that:

```
x_min <= node.x <= x_max and y_min <= node.y <= y_max
```

To check the un-grappled end effector is not inside an obstacle, the EE's coordinates are compared with each obstacle coordinate points such that:

$$x_1 \leq EE_x \leq x_2 \text{ and } y_1 \leq EE_y \leq y_2$$

Where x1 and x2 are the x coordinate min and max, and y1 and y2 are the y coordinate min and max for an obstacle.

To check for a configuration collision, a bounding box test is done for each of the point of a configuration against each obstacle, in addition to edge checks for all obstacle edges with robot arm links.

Two bounding boxes for three points are in collision if:

$$a_x < c_x < b_x \text{ and } a_y < c_y < b_y$$

To test for an edge collision for sampling, self-collision and the connection strategy, the following check is completed as follows:

Each obstacle is a rectangle with four corners, which when paired make up the four edges of a rectangle. Each obstacle edge containing two corners (A and B) is compared with two points (B and C), be they two node points or two points of a robot arm segment. The following cross product is used to find the orientation of three points:

$$(ax,ay), (bx,by), (cx,cy) = (bx - ax) * (cy - ay) - (cx - ax) * (by - ay)$$

Python implementation:

```
area_abc = (b[0] - a[0]) * (c[1] - a[1]) - (c[0] - a[0]) * (b[1] - a[1])
area_abd = (b[0] - a[0]) * (d[1] - a[1]) - (d[0] - a[0]) * (b[1] - a[1])
area_cda = (d[0] - c[0]) * (a[1] - c[1]) - (a[0] - c[0]) * (d[1] - c[1])
area_cdb = (d[0] - c[0]) * (b[1] - c[1]) - (b[0] - c[0]) * (d[1] - c[1])
```

If abc and abd have different orientations and cda and cbd have different orientations, there is a line collision between ab and cd.

```
if (np.sign(area_abc) != np.sign(area_abd)) and (np.sign(area_cda) !=
np.sign(area_cdb)):
    return True
```

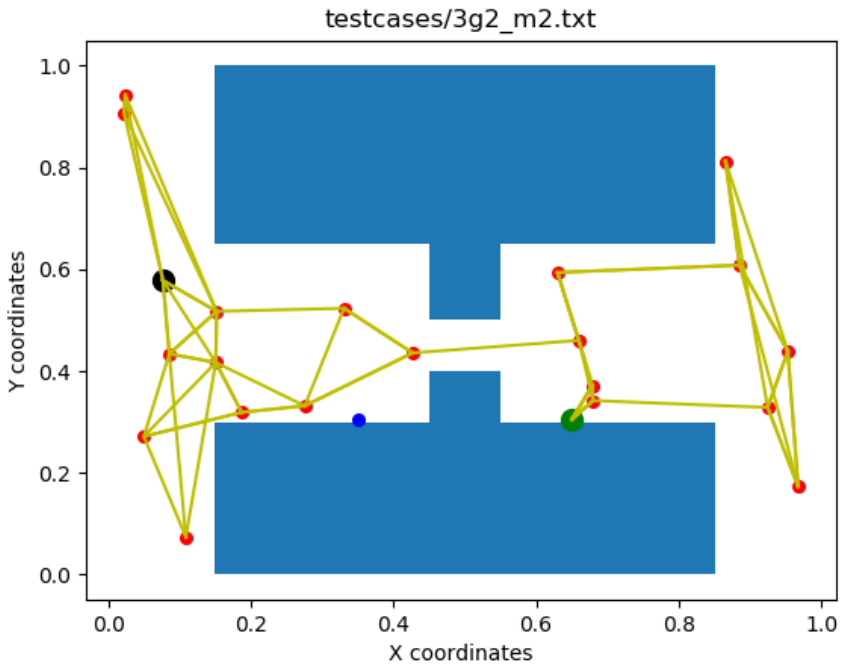
Otherwise, there is no collision (return False)

3.0 Scenarios

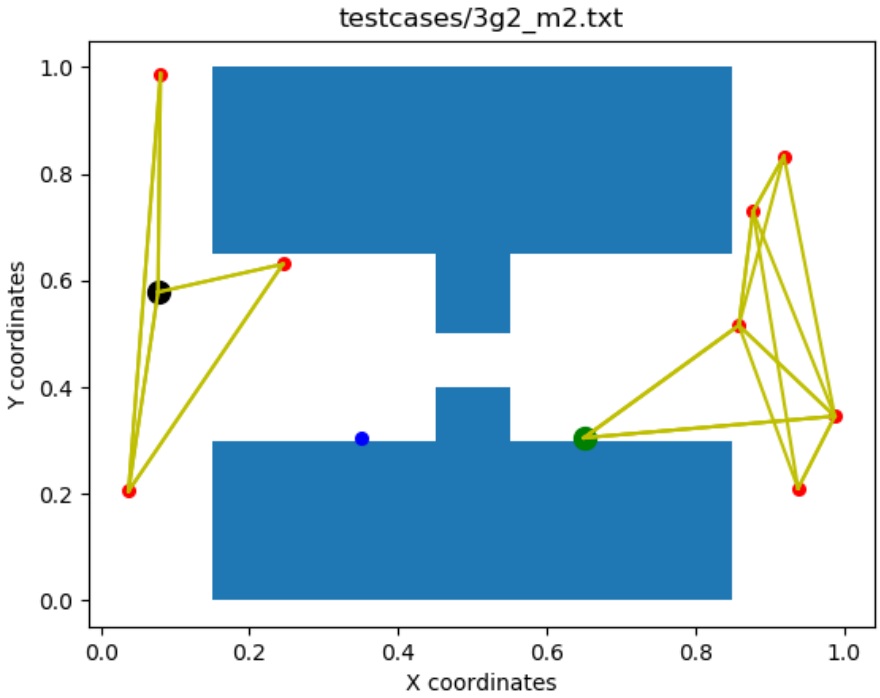
The method used of PRM is well suited for repeated planning (multiple queries) between different pairs of start and goal nodes in the same environment. This is useful in the case of multiple grapple points, which can be broken down into multiple search problems between start node to required grapple node, then to grapple node to goal node.

3.1 Different gaps and obstacles

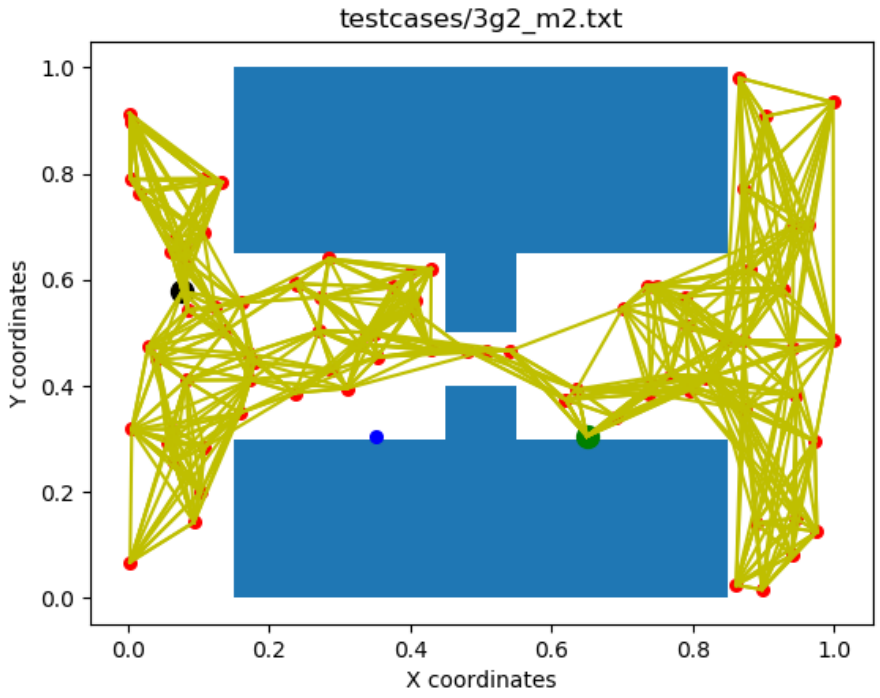
Tight obstacles can be difficult, as enough samples must be taken so that a collision free path between a tight space can be found:



When the number of samples is too low, a path may not be possible:



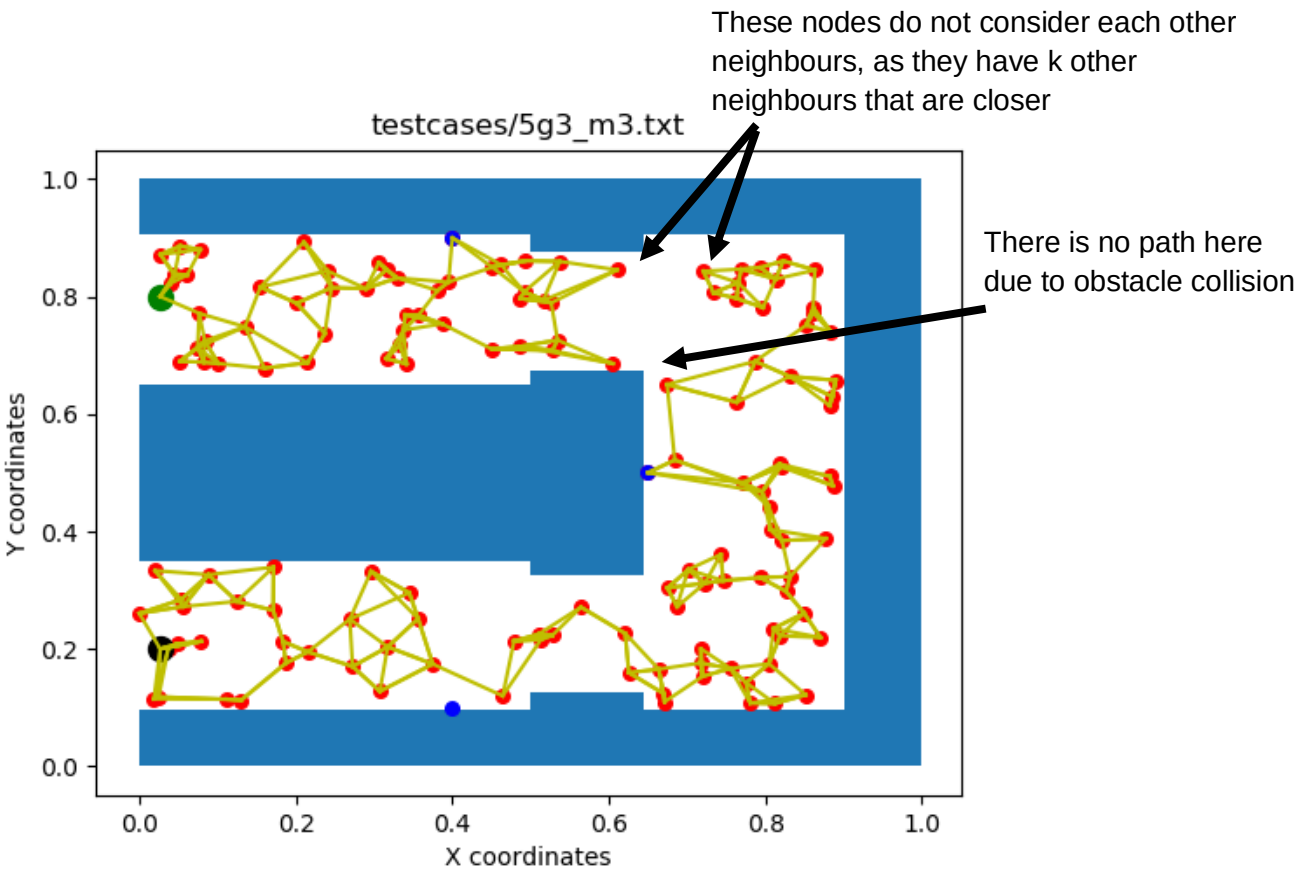
Increasing the number of nodes/samples allows a path to be found:



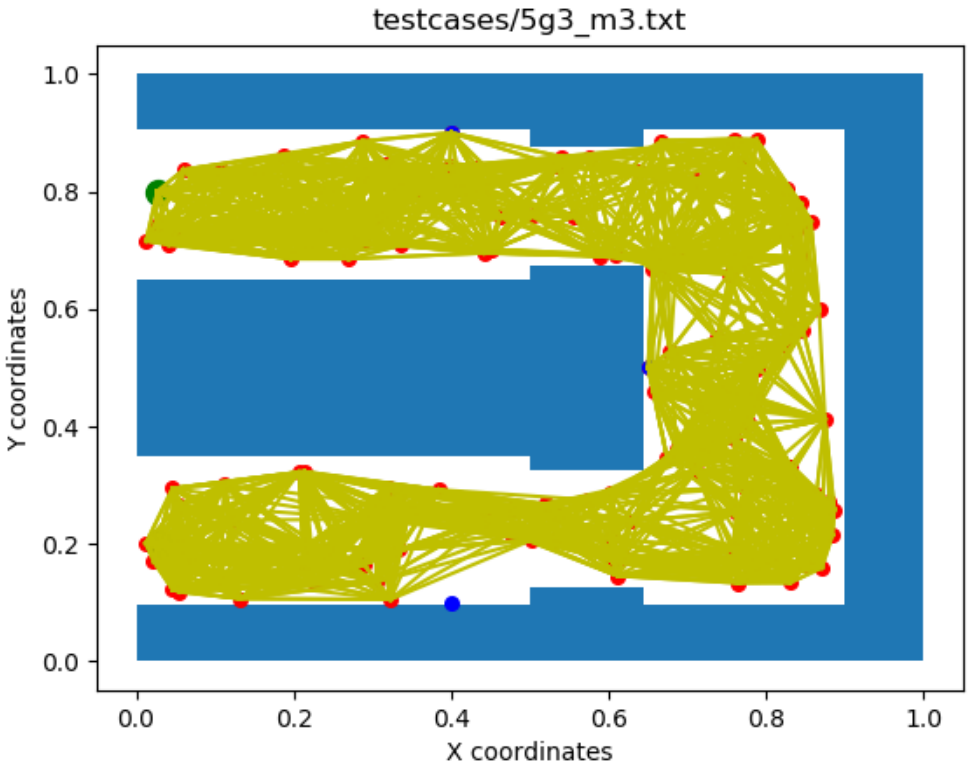
Increasing the number of nodes increases the time to create the points and find paths between the nodes.

3.2 K nearest neighbours:

Another influencing factor in PRM is the k value for the number of neighbours added for each node. When the k value is too low, it is possible a path from start to goal will not exist:



A solution is to increase the k value. A high k value creates many paths between the nodes, allowing for a more optimal solution from start to goal to be found.

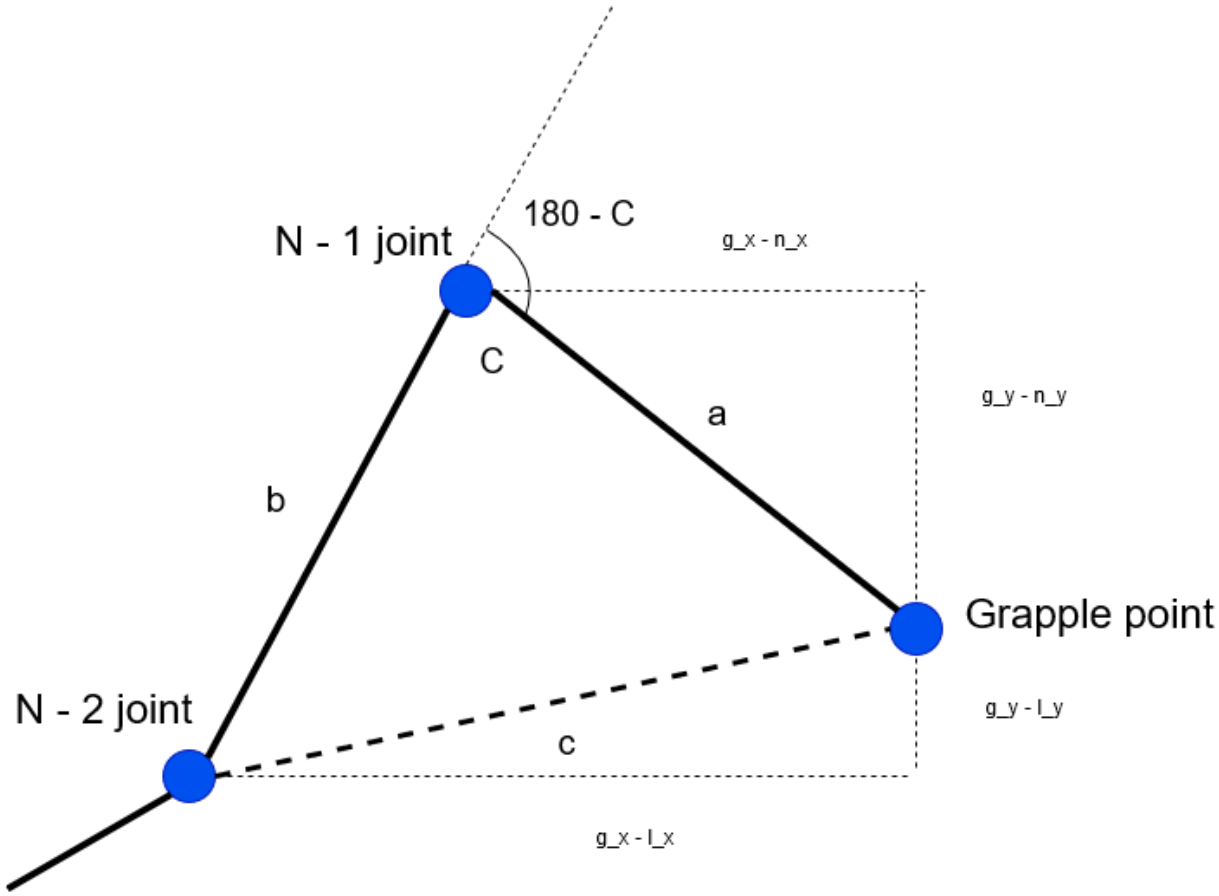


Increasing the K value increases the run time for the search algorithm, as there are more successors to each node to search.

3.3 Multiple grapple points

The advantage of using a PRM is it allows for multiple queries, as all the sample/node points have already been generated for each grapple point. However, for each additional grapple point, samples must be generated a for a robot arm configuration at that point, which exponentially increases run time. With a high number of grapple points, it is possible the program will not be able to complete within the given time limit. Additionally, a bridge configuration must be found for the arm to move between the grapple nodes, since the configuration of a grapple point the arm wishes to move to is initially unknown. The method to find a bridge configuration is as follows:

For N – 1 links (where N is number of arm links) of the robot arm, create a random configuration of random angles and random arm lengths. For the Nth link, calculate the arm link length and angle from the end effector to the desired grapple point. The length can be found using SSS triangle trigonometry:



From the diagram above depicting a possible bridge configuration, the lengths for a, b, and c are required.

b = length of last arm link

$$a = \sqrt{(g_x - n_x)^2 + \left(g_y - n_y\right)^2}$$

$$c = \sqrt{(g_x - l_x)^2 + \left(g_y - l_y\right)^2}$$

Using the values found above to find the angle C:

$$C = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

The angle for the Nth – 1 link is a_n = -(180 – C), and the length for the Nth link is the value a. A bridge configuration has be found if the following conditions are satisfied:

$$\text{min\_length} \leq a \leq \text{max\_length and} -165 \leq a_n \leq 165$$

If it does not meet the conditions, try again with a different configuration for Nth – 1 angles and lengths.

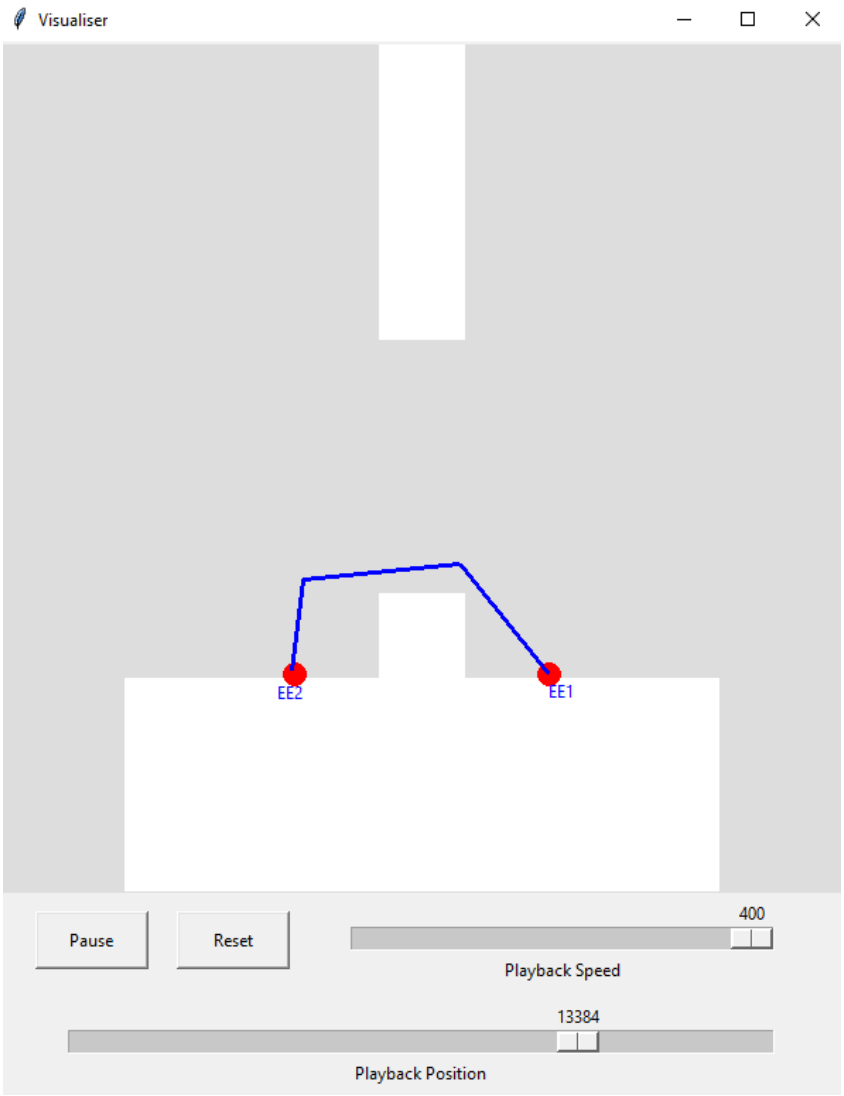


Figure 6: Example of bridge configuration

3.4 Robot arms with telescoping lengths and number of links

Since the sample/node points of the PRM are created using random angles and lengths within bounds, having the robot arm have the ability to change its length makes the problem easier, as there are more configurations for the robot arm to get to tricky points by varying its length.

Increasing the number of links of the robot arm increases the run time, particularly during collision checking during interpolation of configurations, as every robot arm link must be checked for obstacle or self-collisions.

3.4.1 Explanation of results

Testcase	Grapple pts	Number links	Time required (s)
3g1_m0	1	3	>17.960721969604492
3g1_m1	1	3	>14.96879529953003
3g1_m2	1	3	>11.359419822692871
3g2_m1	2	3	> 63.381054639816284
4g1_m1	1	4	>19.049856901168823

Unsurprisingly, the test case with a four-link arm required the most time for a single grapple point, as more interpolation and collisions checked were required for the additional degree of freedom. While the simplest, test case 3g1_m0 took longer than the other three link arm testcases, as due to the absence of obstacles, more valid nodes were able to be created, thus increasing the configurations the search algorithm must search.

In comparison of the number of nodes, test case 3g2_m1 shows a major increase in run time with the addition of an extra grapple point, while the same number of nodes are used, the program must solve two search problems from start to grapple point, and grapple point to goal.

3.5 “Bad luck” – collision detection between node points failing to detect obstacle

Bad luck might not be a “correct” heading, however - The interpolation between node points takes a primitive step of 0.01 radians for angles and 0.01 units for lengths (0.01 radians corresponds to 0.5729 degrees). For the testcase 4g1_m1, there was a test run that failed once out of up to 20 trials with the same settings. During interpolation, a collision check is completed for every 0.5729 degrees for each link of the robot arm. In the trial run that failed, the robot arm end effector can be seen barely touching an obstacle as it passes by, as a collision was not detected.