Vincentius Aditya Sundjaja
COMP3702
45610099

COMP3702/7702 ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2 REPORT

Canadarm – Continuous Motion Planning

I. Configuration Space

The Configuration Space (C-space) generally consists of forbidden region which represents the obstacles given in the problem. Then, it has a free space which is an area in the C-space that can be populated with n number of configurations. A configuration q that is placed in the free space means that q is collision-free and does not intersect any obstacles in the C-space.

In the canadarm problem, as it is simplified, it operates in a 2D workspace (a plane represented as [0, 1] x [0, 1] $\subset$ R2, and is populated by rectangular obstacles). The robotic arm is composed of x links and x joints, where x is a non-negative integer, with two end effectors EE1 and EE2, which can attach onto grapple points.

The configuration space for this problem is the set of all points. For example, a robotic arm with 3 links and 3 joints will have a C-space of (ee1_x, ee1_y, angle_1, angle_2, angle_3, length_1, length_2, length_3).

For searching in continuous space, the problem is converted to a search problem which need a state graph. To generate the state graph, PRM (Probabilistic Roadmap) was used. After, the state graph is fully generated, a simple BFS (Breadth First Search) algorithm is used to search through the roadmap and find the shortest path between the initial and goal node. More detail on the next section.

II. Sampling Based Method

1. Sampling Strategy

By using PRM, then nodes will be randomly generated until a certain number of nodes is reached or a path to the goal node is constructed or time limit is reached.

```
PRM Pseudo Code


G(V,E) = Null //Initialize a graph as empty
limit = n //number of nodes to make graph out of
Rad = r //radius of neighborhoods
For itr in 0...limit:
    Xnew = RandomPosition()
    Xnearest = Near(G(V,E),Xnew,Rad) //find all nodes within a Rad
    Xnearest = sort(Xnearest) //sort by increasing distance
    For node in Xnearest:
        if not ConnectedComp(Xnew,node) and not Obstacle(Xnew,node):
            G(V,E) += {Xnew,node} //add edge and node to graph
            Xnew.comp += node.comp//add Xnew to connected component
Return G(V,E)
```

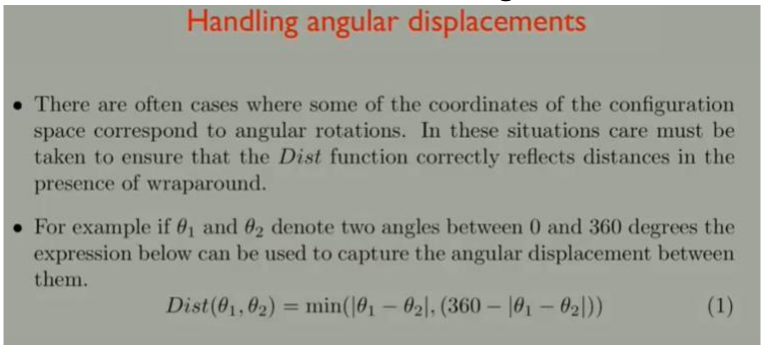
source: https://medium.com/@theclassytim/robotic-path-planning-prm-prm-b4c64b1f5acb

Following the above pseudocode (with slight changes), the PRM was constructed. Until "n" valid nodes are generated, then a random node will be produced and checked. Then, after all nodes are generated, my algorithm will loop through all of them to find the nearest nodes in "r" radius. All nodes in the "r" radius of node "p" will be the neighbor of node "p".

After trial and error, a 'good' value for "n" and "r" were found. For "n" total nodes, the calculation is *100 * num_segments + 100 (num_grapple_points – 1)*. For "r" radius, the calculation is *0.25 * num_segments + 0.05 * (num_grapple_points - 1)*. The reason the numbers are increasing when number of segments and grapple points are increased is that the problem will be harder to solve, and more nodes are needed to be able to find a valid path. Another factor that can be considered is the number of obstacles and how narrow they are.

2. Connection Strategy

As mentioned before, for connecting the nodes, a radius was used. All nodes within "r" radius will be considered as a neighbor. The element that is checked by the radius is the difference between each angle of two configuration. Example:
C1_angles = a, b, c; C2_angles = 1, 2, 3; Difference = a-1, b-2, c-3
If each diff in difference is <= radius, then it is considered to be a neighbor
To calculate the difference between angle, I used this formula:

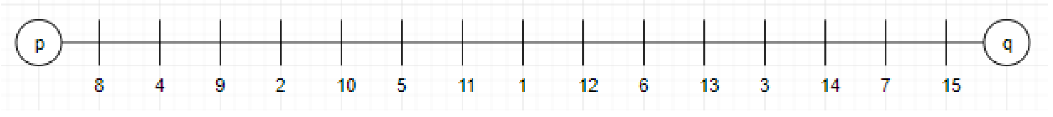

Handling angular displacements

- There are often cases where some of the coordinates of the configuration space correspond to angular rotations. In these situations care must be taken to ensure that the *Dist* function correctly reflects distances in the presence of wraparound.
- For example if θ_1 and θ_2 denote two angles between 0 and 360 degrees the expression below can be used to capture the angular displacement between them.

$$Dist(\theta_1, \theta_2) = \min(|\theta_1 - \theta_2|, (360 - |\theta_1 - \theta_2|)) \quad (1)$$

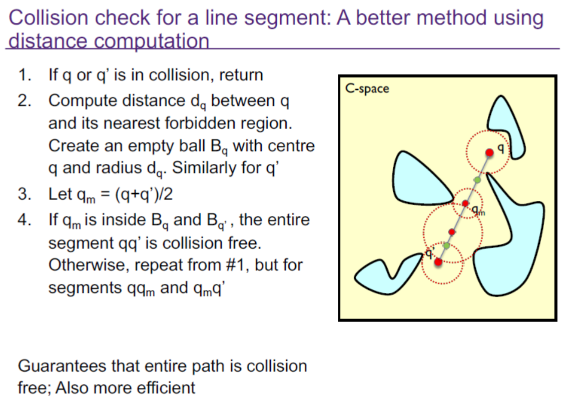
Source: https://www.youtube.com/watch?v=hFGhaSRV1zY (3:35)

Then, for every edge between 2 nodes, it will be split to several parts such as:



Neighbor of p will then be 8, neighbor of 8 will be p and 4, and so on. The reason to split this is to have more accurate nodes so that when primitive steps are generated, it won't collide to an obstacle. The result of the search will be smoother even though it's still not less than the primitive step required. It still needs to be processed with the generate primitive step algorithm.

3. Configuration Checks
   For checking the configurations, several methods from the provided *tester.py* are used. They are *test_obstacle_collision(), test_self_collision(), and test_environment_bounds().*
   For checking if a line segment in C-space (edge between 2 nodes), it can be done by the following



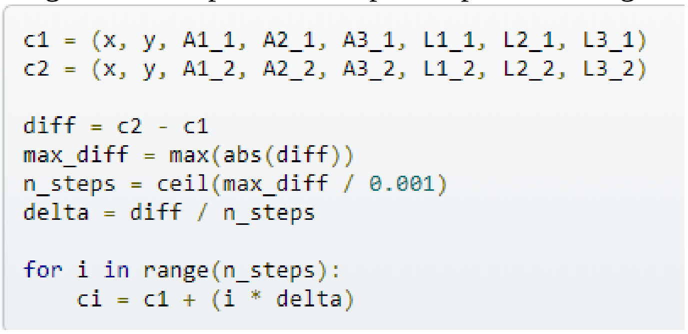
Collision check for a line segment: A better method using distance computation

1. If q or q' is in collision, return
2. Compute distance d_q between q and its nearest forbidden region. Create an empty ball B_q with centre q and radius d_q. Similarly for q'
3. Let $q_m = (q+q')/2$
4. If q_m is inside B_q and $B_{q'}$, the entire segment qq' is collision free. Otherwise, repeat from #1, but for segments qq_m and q_mq'

Guarantees that entire path is collision free; Also more efficient

   Source: Lecture 4 slides
   However, in my case, the results of the configs are not optimal (most likely because of a fault in the implementation). They were not always passed the testcases. Thus, I used the split method mentioned earlier. By discretizing smaller segments and connected those segments to other edges, the search path will be more flexible and can generate more optimal solution.

4. Generating primitive steps
   To generate the primitive steps, the pseudocode given in Piazza was used

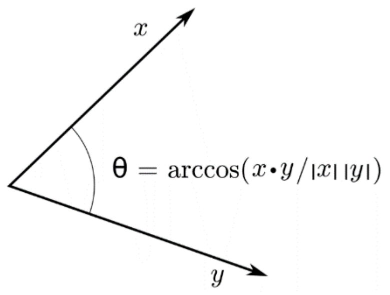


```
c1 = (x, y, A1_1, A2_1, A3_1, L1_1, L2_1, L3_1)
c2 = (x, y, A1_2, A2_2, A3_2, L1_2, L2_2, L3_2)

diff = c2 - c1
max_diff = max(abs(diff))
n_steps = ceil(max_diff / 0.001)
delta = diff / n_steps

for i in range(n_steps):
    ci = c1 + (i * delta)
```

   First, the difference between all angles (in radians) and all lengths are calculated. Then, taking the maximum absolute difference, we can compute the number of steps needed so that each primitive step will be less or equal to 0.001. Delta is calculated so that each transition between angles and the lengths will be smoother.

5. Generating bridging configurations
   This part in my program is still not working properly. It is working for the testcases with 2 grappler points, but not for 3 and more. My strategy for this is

first, to generate n-1 (n number of segments) angles and lengths. Then calculate the distance and angle of the last point to the next grappler point.



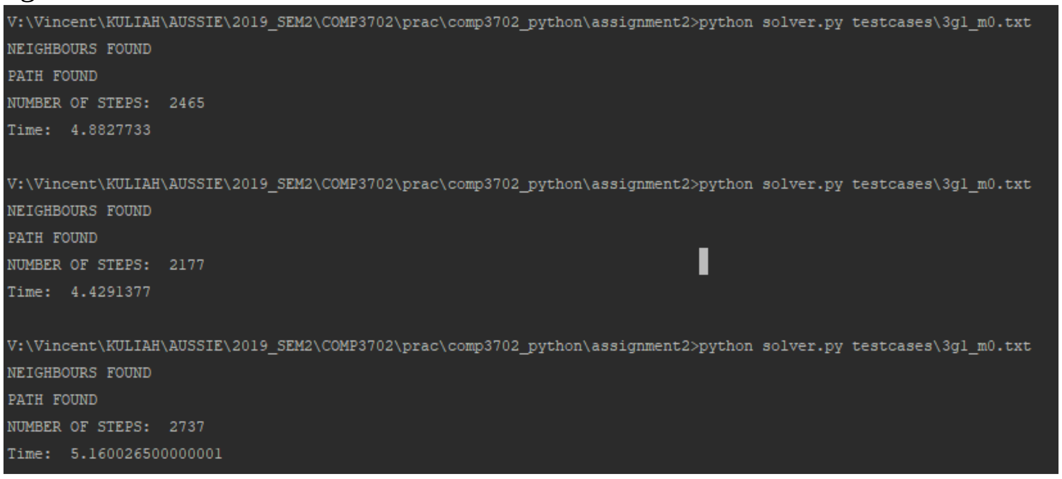
Source: https://medium.com/@manivannan_data/find-the-angle-between-three-points-from-2d-using-python-348c513e2cd

By using trigonometry (I use arccos), we can calculate the distance and the angle needed for the n-th segment to reach the next grappler point. If the angle and length is valid (within the given range), then check if the new configuration is valid (not collide to an obstacle or itself or out of bound). After a valid bridge configuration is found, we can split the search in two parts, from the initial node to the bridge node, and from the bridge node (change the grapple) to the goal node. Then concatenate the two results to be one.

III. Class Scenarios

From the testcases, my solver was able to solve all level 1 (3g1_m0, 3g1_m1), all level 2 (3g1_m2, 4g1_m1, 4g1_m2), and 2 level 3 (3g2_m1, 3g2_m2).
Here are the stats from each test cases:

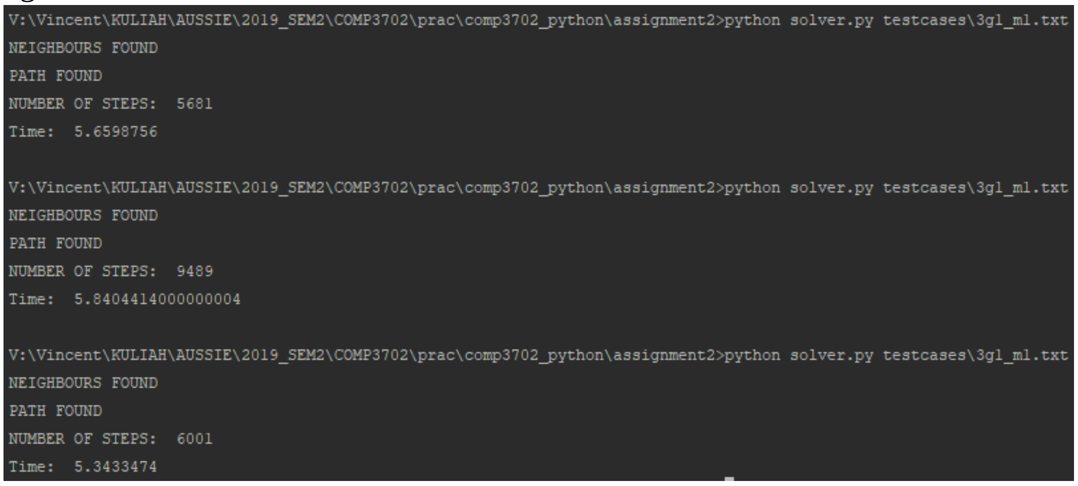
1. 3g1_m0



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m0.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  2465
Time:  4.8827733

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m0.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  2177
Time:  4.4291377

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m0.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  2737
Time:  5.160026500000001
```

Solved 3/3

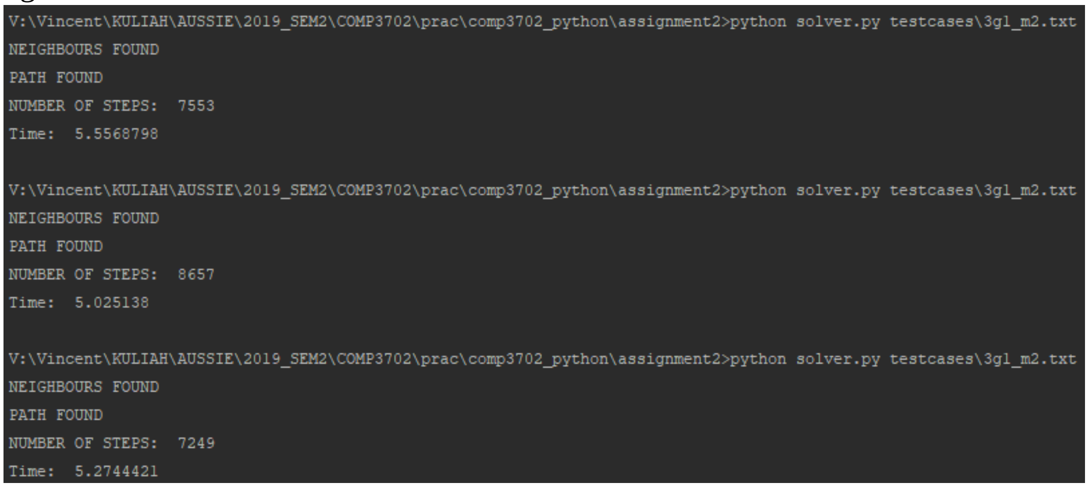
2. 3g1_m1



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  5681
Time:  5.6598756

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  9489
Time:  5.8404414000000004

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  6001
Time:  5.3433474
```

Solved 2/3 (1 solution has 12 collisions)

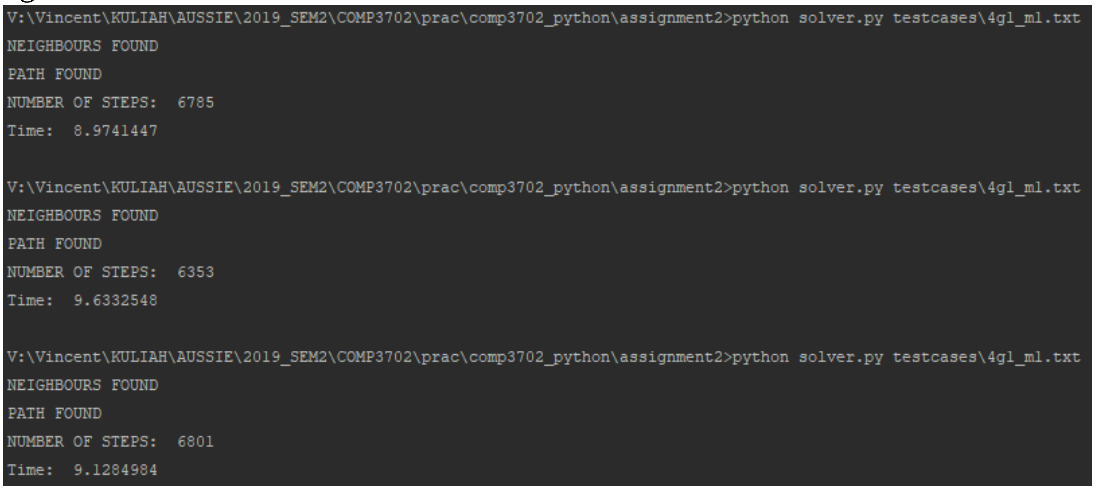
3. 3g1_m2



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  7553
Time:  5.5568798

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  8657
Time:  5.025138

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g1_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  7249
Time:  5.2744421
```

Solved 2/3 (1 solution has 11 collisions)

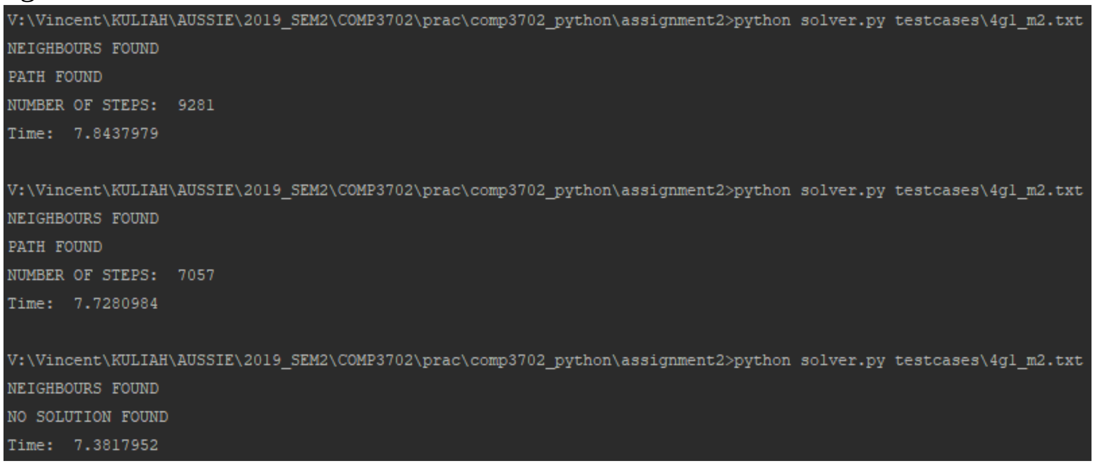
4. 4g1_m1



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  6785
Time:  8.9741447

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  6353
Time:  9.6332548

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  6801
Time:  9.1284984
```

Solved 3/3

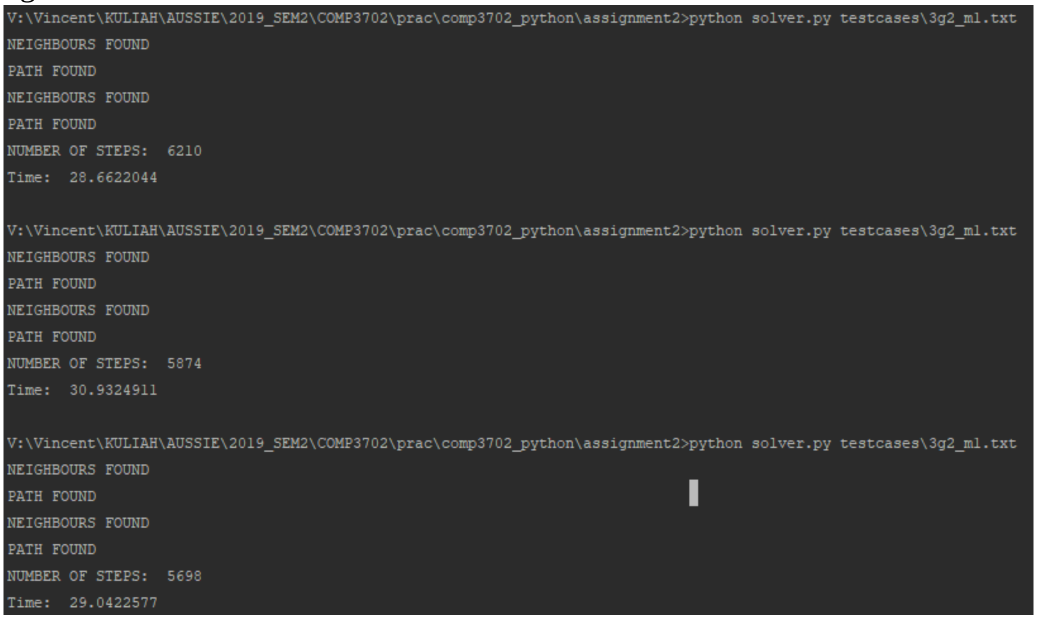
5. 4g1_m2



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  9281
Time:  7.8437979

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  7057
Time:  7.7280984

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\4g1_m2.txt
NEIGHBOURS FOUND
NO SOLUTION FOUND
Time:  7.3817952
```

Solved 3/3

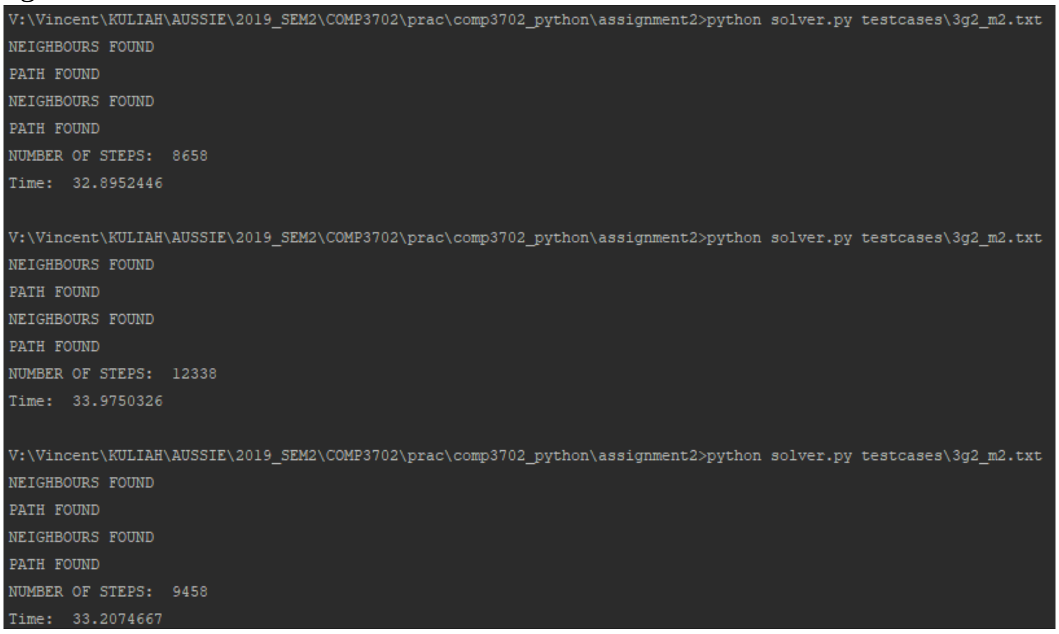
6. 3g2_m1



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  6210
Time:  28.6622044

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  5874
Time:  30.9324911

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m1.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  5698
Time:  29.0422577
```

Solved 3/3

7. 3g2_m2



```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  8658
Time:  32.8952446

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  12338
Time:  33.9750326

V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment2>python solver.py testcases\3g2_m2.txt
NEIGHBOURS FOUND
PATH FOUND
NEIGHBOURS FOUND
PATH FOUND
NUMBER OF STEPS:  9458
Time:  33.2074667
```

Solved 3/3

Problems with more than 2 grappler points are still not solvable. Methods for generating bridging nodes is not working.

Problems with very narrow gaps are not likely to be solved (my programs not checking how narrow the gaps are).

Problems with more joints and angles will be harder to solve but still possible (might have less successful trials).