RSS HW3

Harin Kumar Nallaguntla (NUID: 002751978)
Programming Partners: Aditya Bondada, Mani Chandan Chakinala

M0. The function check_collision analyzes a 4-DOF robot, which is defined as a SerialLink object, with a given configuration q, a spherical obstacles set specified by sphere_centers and sphere_radii, and the cylindrical body radius of each link, link_radius. The linear interpolation is used to calculate a sequence of points along the robot's links, which are subsequently checked to see if they are inside any of the spherical obstacles. If any point falls within one of the spheres, the function returns true to indicate that a collision has happened, and false otherwise.

Similarly, the function check_edge, which also takes in the same parameters as check_collision, as well as two configurations q_start and q_end, is used to determine whether a collision has occurred between the robot and the spherical obstacles when it moves along a straight-line path defined by q_start and q_end. To do so, it creates a sequence of configurations by performing linear interpolation on the start and end configurations. After that, the function checks every configuration in the sequence for collisions using check_collision. If any configuration is in collision, the function will return true, otherwise false.

Two issues in the collision-checking algorithm are:

1. The accuracy of collision detection can be affected by the resolution of the discretization used to sample points along links and configurations. In case the resolution is too low, it may miss potential collisions, whereas higher resolution can increase computational time and still not guarantee full accuracy
2. Collision detection solely considers the cylindrical body of the robot's links, which may not reflect the actual shape of the links. This can cause false negatives if an obstacle contacts a part of the link that is not included in the cylindrical body or false positives if the cylindrical body extends beyond the physical limits of the link

M1.

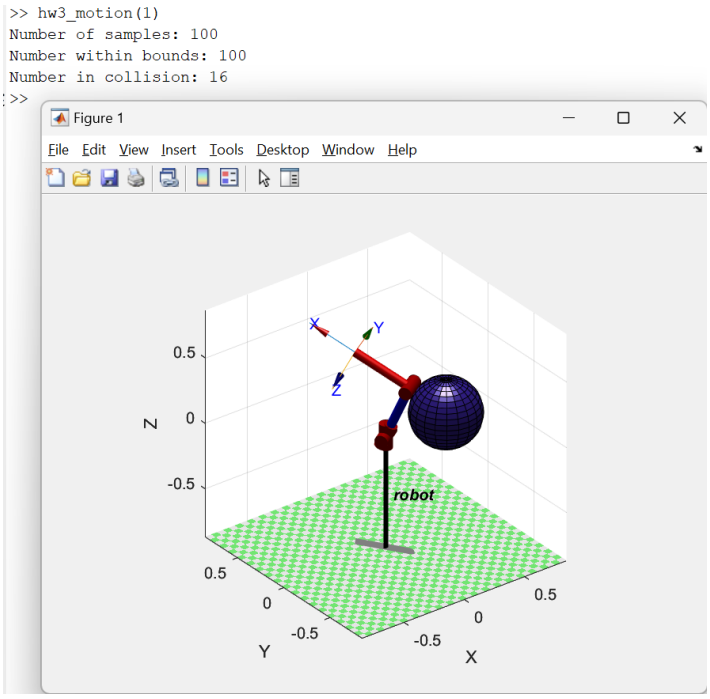

Fig 1. Output of M1.m

The above image shows the result of M1.m. You can see the number of samples in collisions and you can confirm that all the samples are within the bounds

M2.

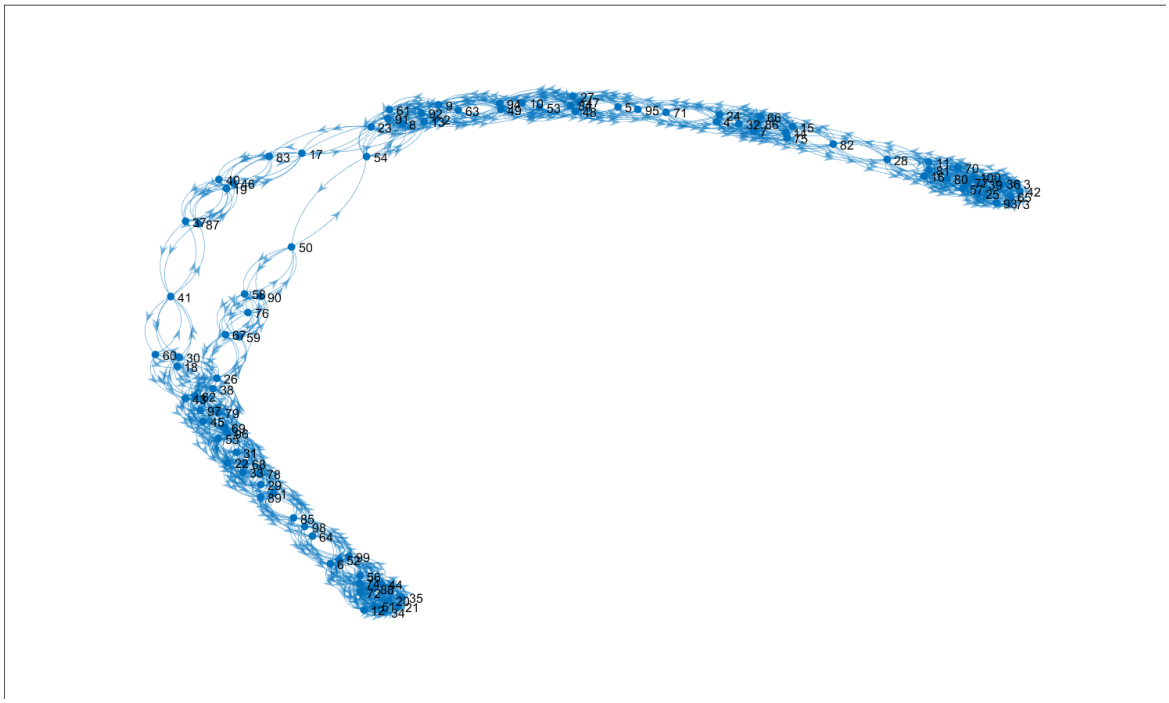

Fig 2. Digraph Plot of Adjacency Matrix

Fig 3. Plot of Adjacency Matrix

M3.

```
>> hw3_motion(3, samples, adjacency)
Path found with 7 intermediate waypoints:
         0   -0.7854         0   -0.7854
   -0.2698   -0.5418         0   -0.7248
   -0.0095   -0.0246         0   -2.4865
    0.1556   -0.1074         0   -2.1620
    0.4310   -1.0702         0   -1.1414
    0.6154   -0.6495         0   -0.4487
    0.6125   -2.0543         0   -1.1279
    0.3579   -2.9322         0   -2.6698
         0   -3.0000         0   -3.0000
```

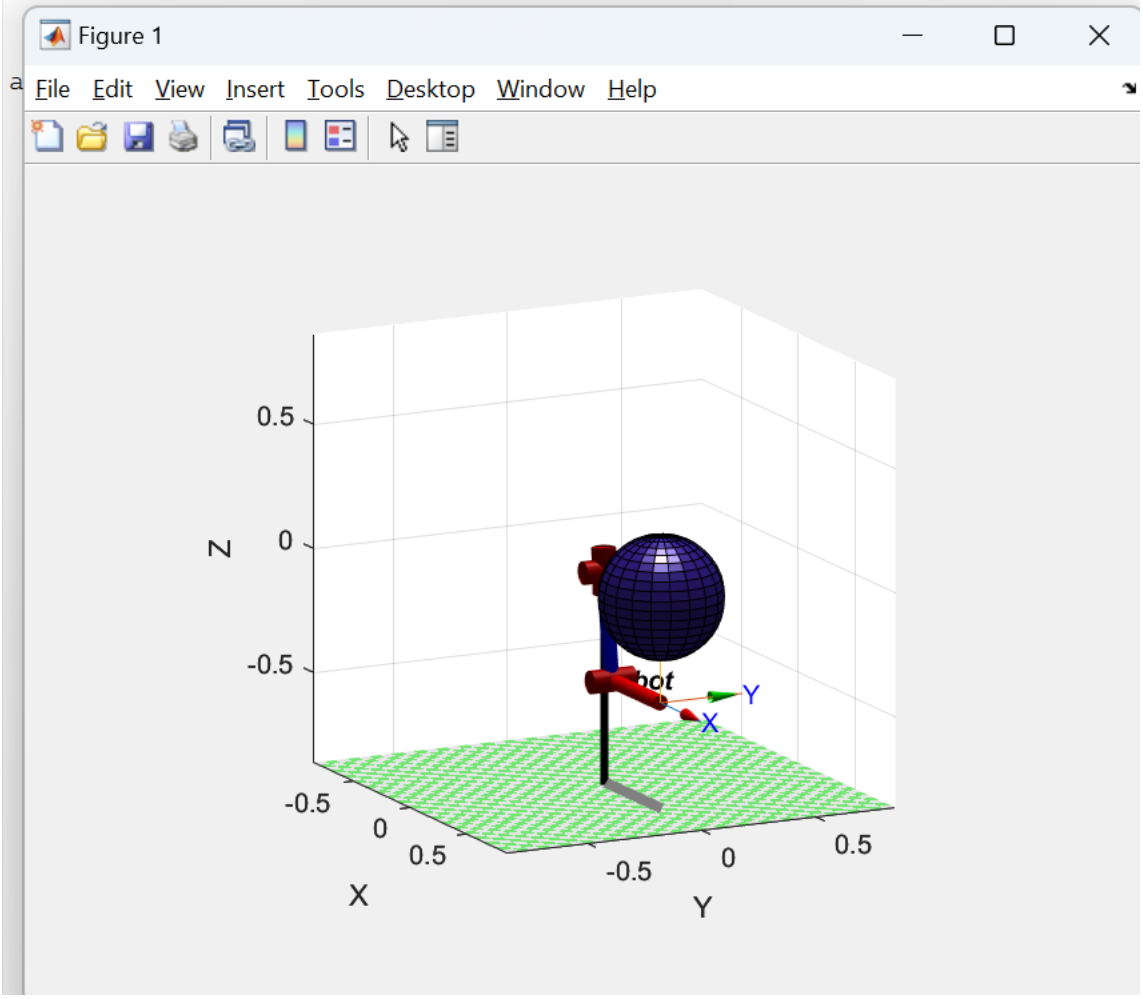

Fig 4. Output of M3.m

I used KNN search to find the nearest neighbors to q_start and q_goal in the samples and I used shortestpath function to find the path

M4.

```
>> hw3_motion(4)
Path found with 5 intermediate waypoints:
         0   -0.7854         0   -0.7854
    1.3352   -0.3020         0   -0.3020
    1.2227   -1.3998         0   -1.3998
    1.4239   -1.1190         0   -1.1190
    0.5096   -2.3330         0   -2.3330
   -0.2034   -3.2662         0   -3.2662
         0   -3.0000         0   -3.0000
```

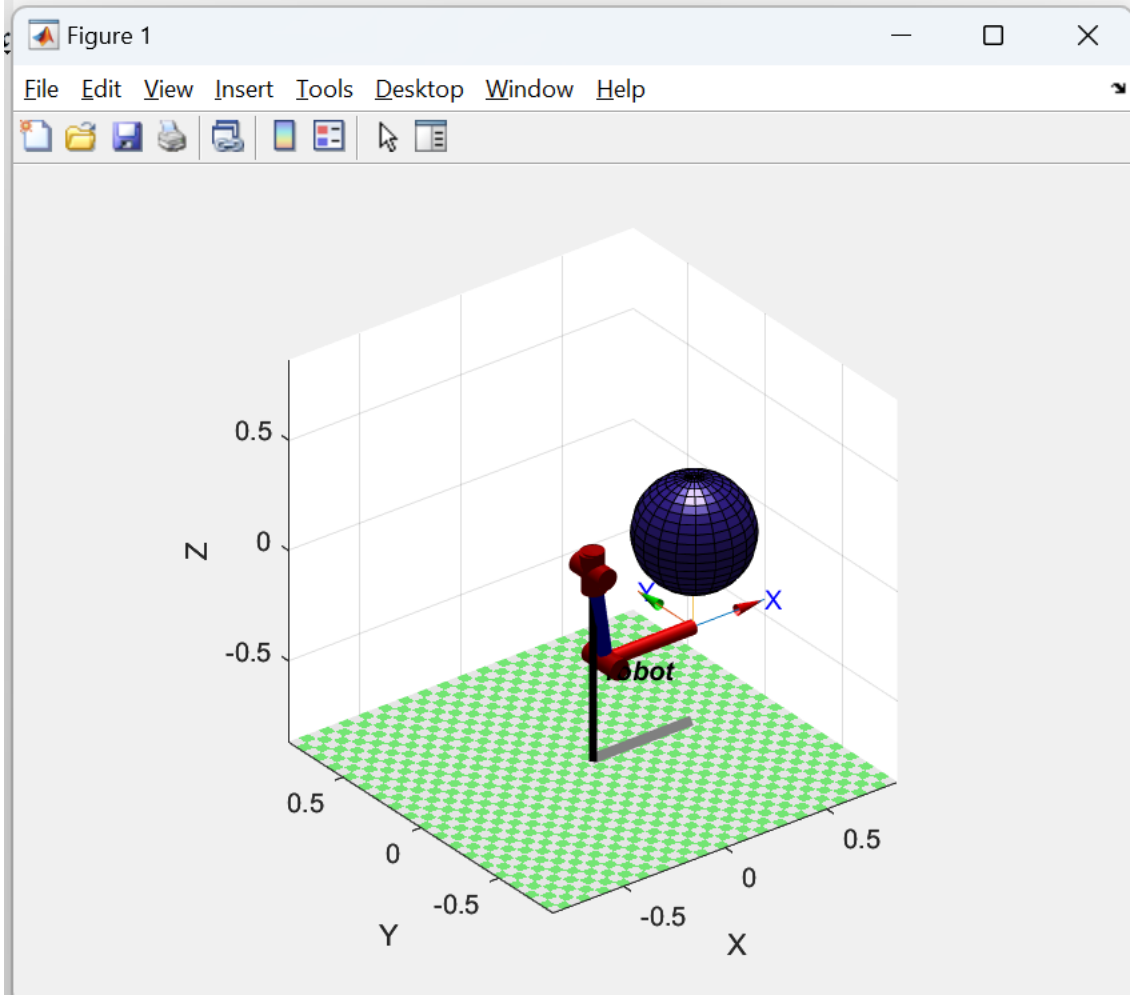

Fig 5. Output of M4.m

My hyperparameters are: sampling frequency of q_goal = 0.7, step_size = 1.5, max_nodes = 10000 and tolerance = 0.8

M5.

```
>> hw3_motion(5)
Path found with 19 intermediate waypoints:
         0   -0.7854         0   -0.7854
    1.3903   -0.3872         0   -0.3872
    0.0172   -0.8142         0   -0.8142
    1.3933    0.6735         0    0.6735
    1.8228    0.6284         0    0.6284
    1.4785   -1.0534         0   -1.0534
    1.7701    0.6389         0    0.6389
   -0.0184   -0.7517         0   -0.7517
    1.8385    0.6250         0    0.6250
    1.7698    0.6390         0    0.6390
    0.4630    0.4465         0    0.4465
    1.6117    0.6619         0    0.6619
    0.2003   -0.4984         0   -0.4984
    1.5129   -0.8943         0   -0.8943
    1.8316    0.6265         0    0.6265
    0.8223    0.5945         0    0.5945
    1.4499    0.6726         0    0.6726
    1.1788   -1.4853         0   -1.4853
    0.4556   -2.4145         0   -2.4145
   -0.2675   -3.3438         0   -3.3438
         0   -3.0000         0   -3.0000

Smoothed path found with 1 intermediate waypoints:
         0   -0.7854         0   -0.7854
    1.1788   -1.4853         0   -1.4853
         0   -3.0000         0   -3.0000
```

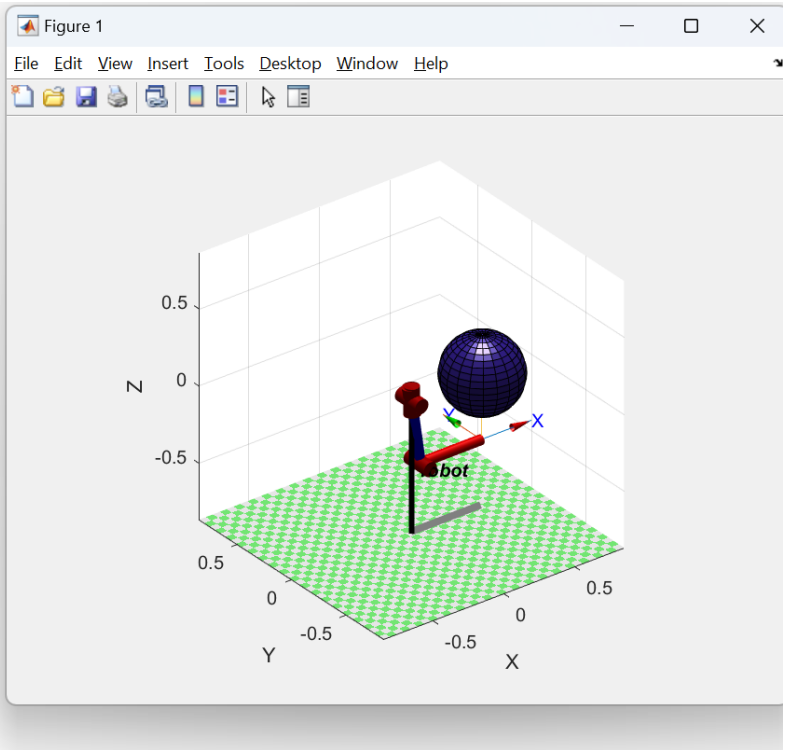

Fig 6. Output of M5.m