Robotics Assignment 2 Report

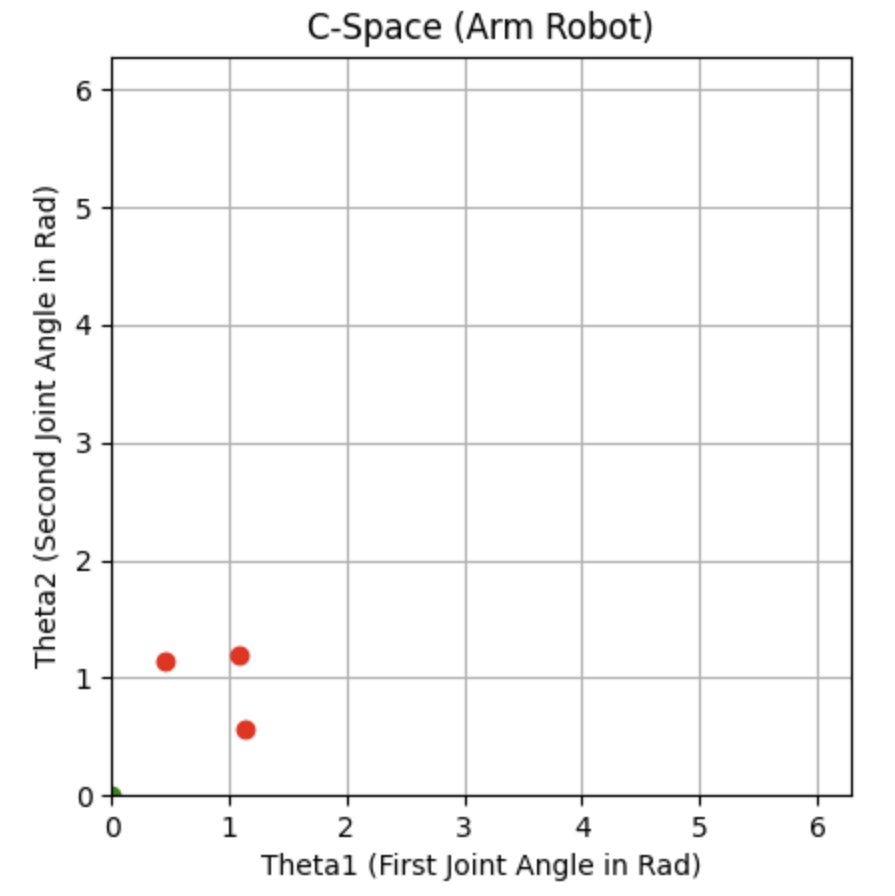
**1) Generating Environments**

In this part, we implement the functions asked of us to help us generate 5 environments that we will use in our program. Each environment is generated with a different number of obstacles, and they are sorted from least to most obstacles. We also implement functions to convert our environments to txt files as well as to convert txt files back to an environment. Lastly we have a function that allows us to visualize the environment with its obstacles in it

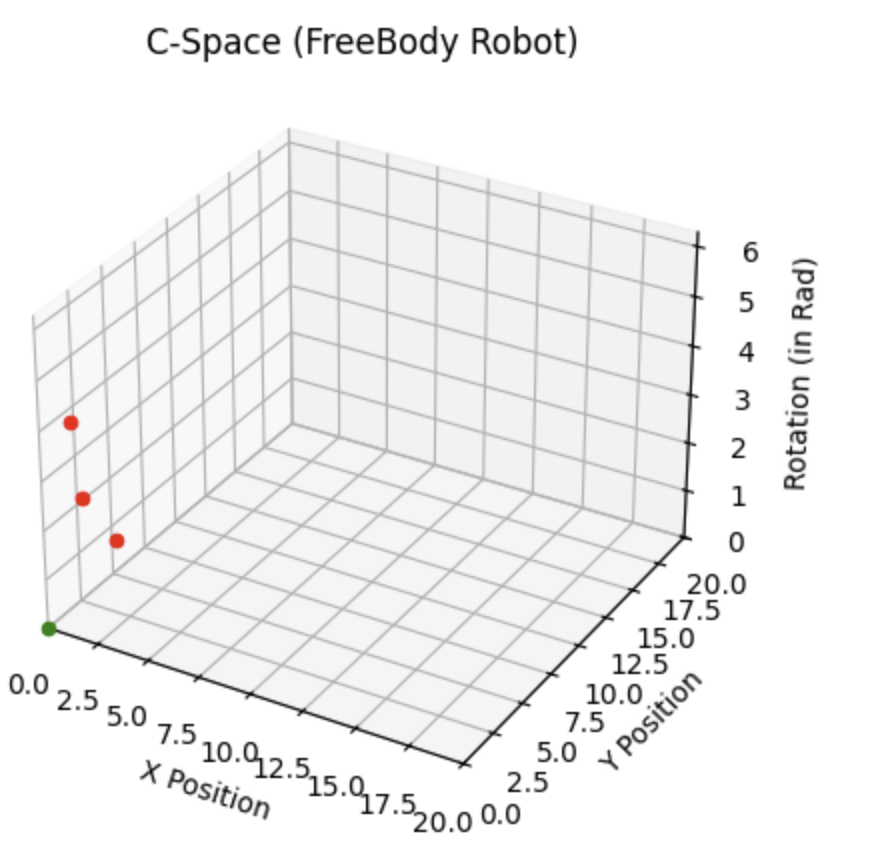
Environments can be found in the environments folder:

**2) Nearest neighbors with linear search approach**

Given a target, we have to output the three nearest neighbors to the target from a list of points. We save these points in config files, separate for each type of robot (“configs\_arm.txt” and “configs\_freeBody.txt” respectively).

Nearest Neighbors C-Space for ARM: For target 0 0, k=3, configs = “configs\_arm.txt”  


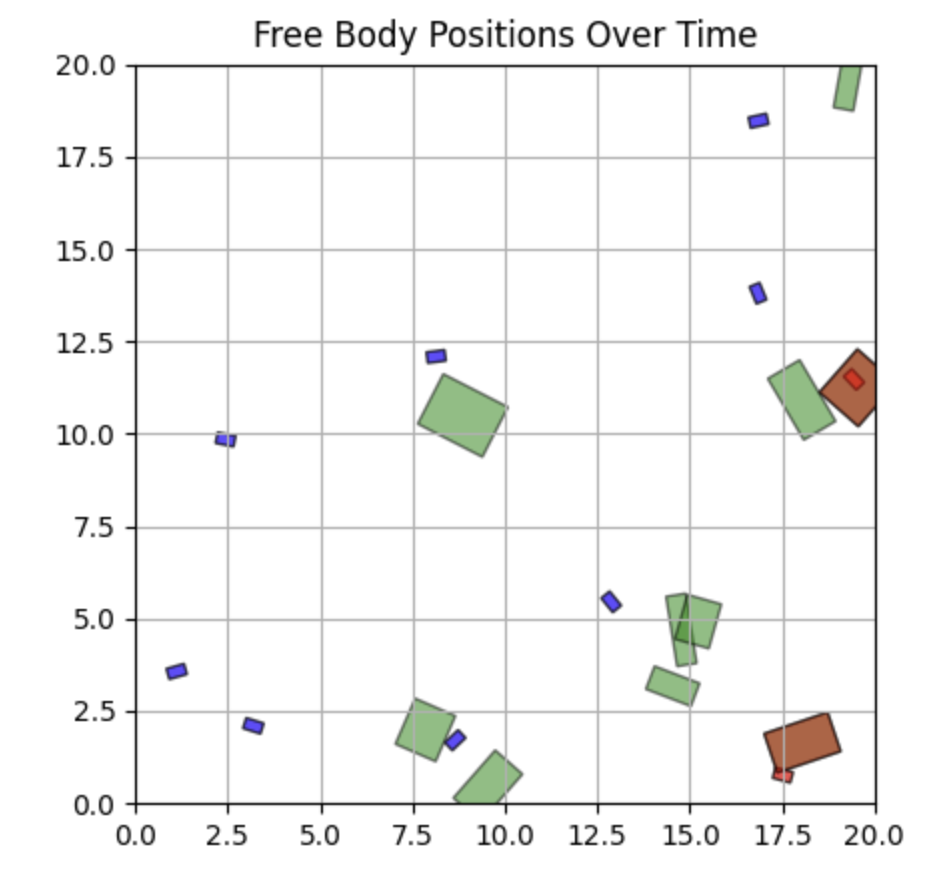
Nearest Neighbors C-Space for freeBody: Target: 0 0 0, k = 3 , configs = “configs\_freeBody.txt”



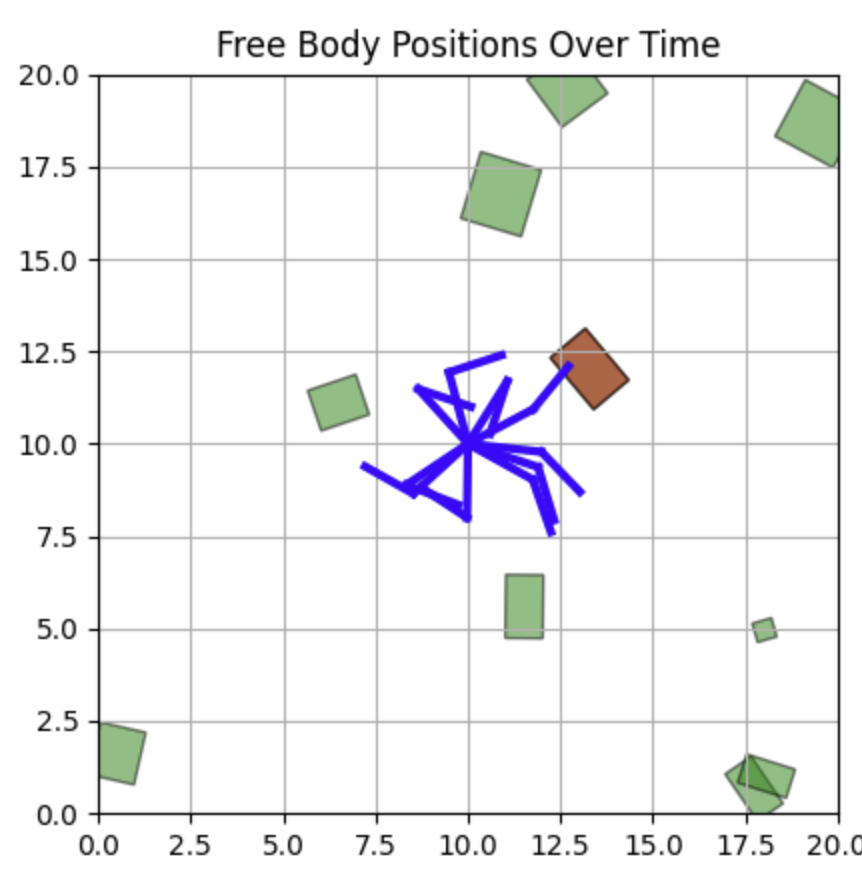
**3)** **Collision checking**

We use various functions to help us determine if an object is colliding with an obstacle in the environment. After loading our environment into an array, we use the separating axis theorem to determine whether or not a collision has occurred. The separating axis theorem states that if there is an axis where two objects don’t overlap, the objects don’t collide.

For environment 5 FreeBody:



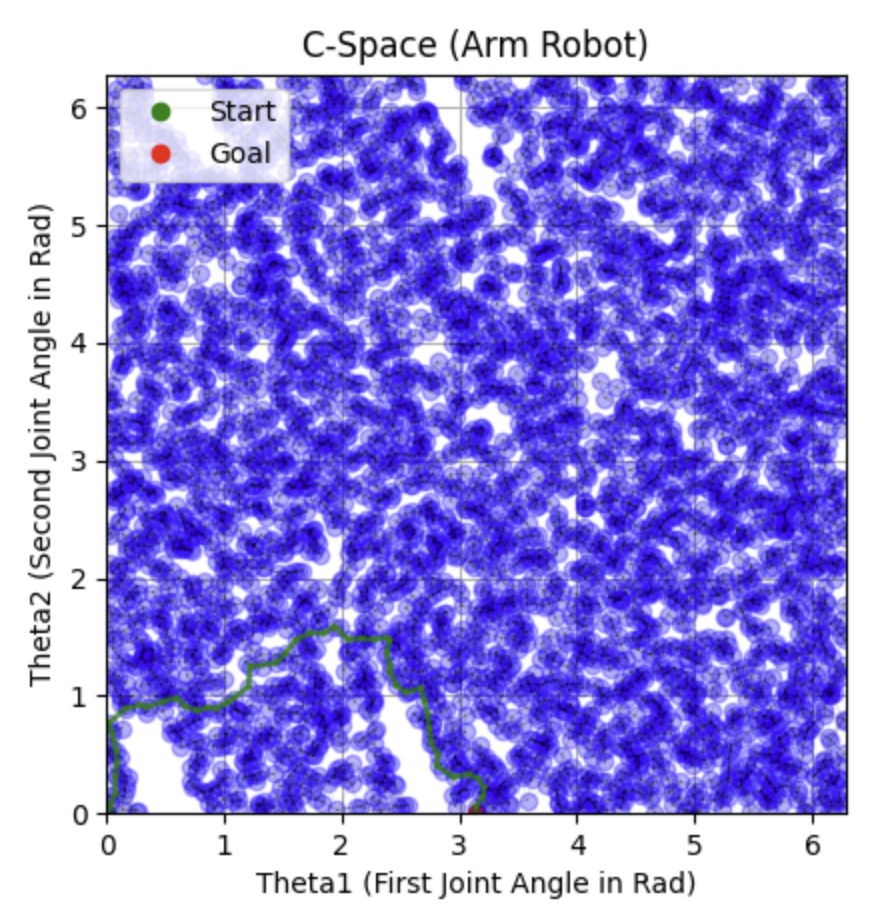
For Environment 5 arm:



**4) PRM**

We connect the 6 nearest nodes to our current node, up to a maximum of 5000 nodes. To find our path, we connect the start node to the goal node, connecting every nearest neighbor on the path that isn’t a collision.

Nodes and Links of arm in environment 5 start (0,0) to (3.14, 0):

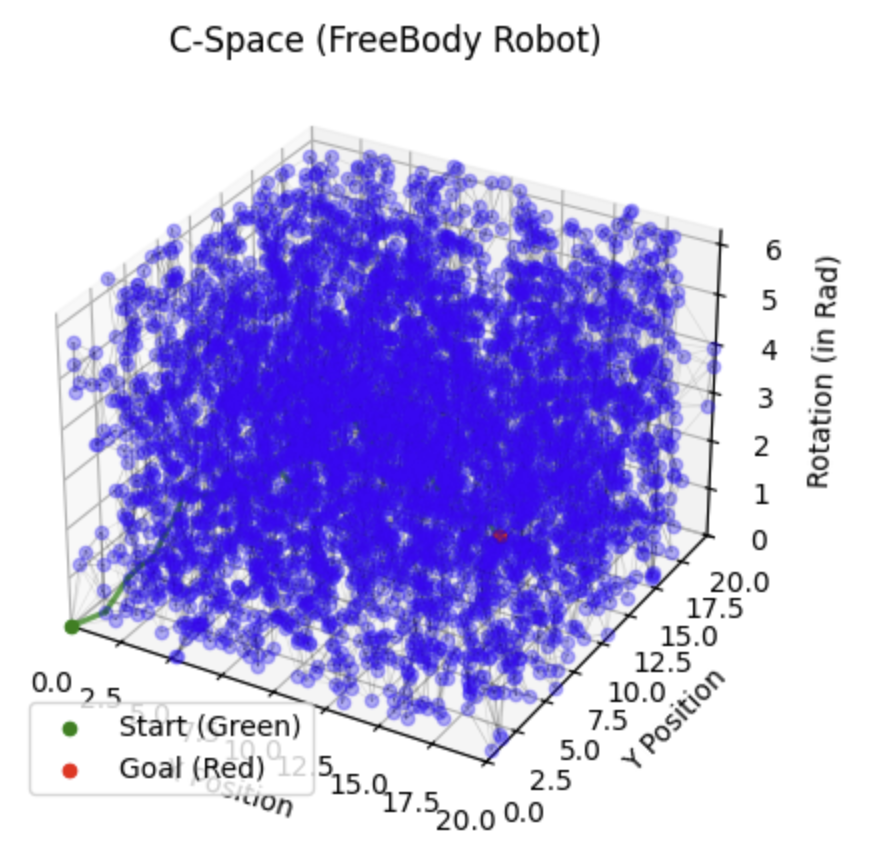


Samples:

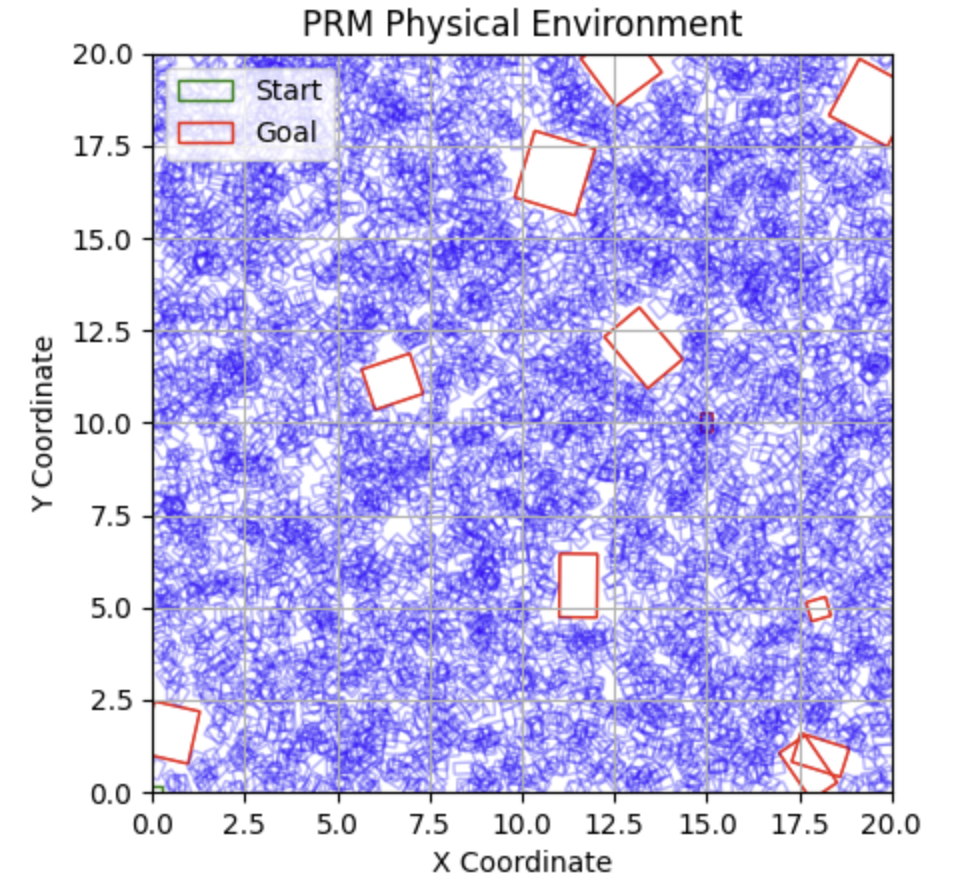


Gif of animation can be viewed in folder aswell.

Nodes and Links of freeBody in environment 5 start (0,0,0) to (15, 10, 1.57):



Samples:



**4.1) RRT**

We implement RRT, which can be used for both freeBody and arm robots. It uses some functions in collision\_checking (to check for collisions of course). We include visualizations for c-space, samples, and a physical environment animation for both freeBody and arm.

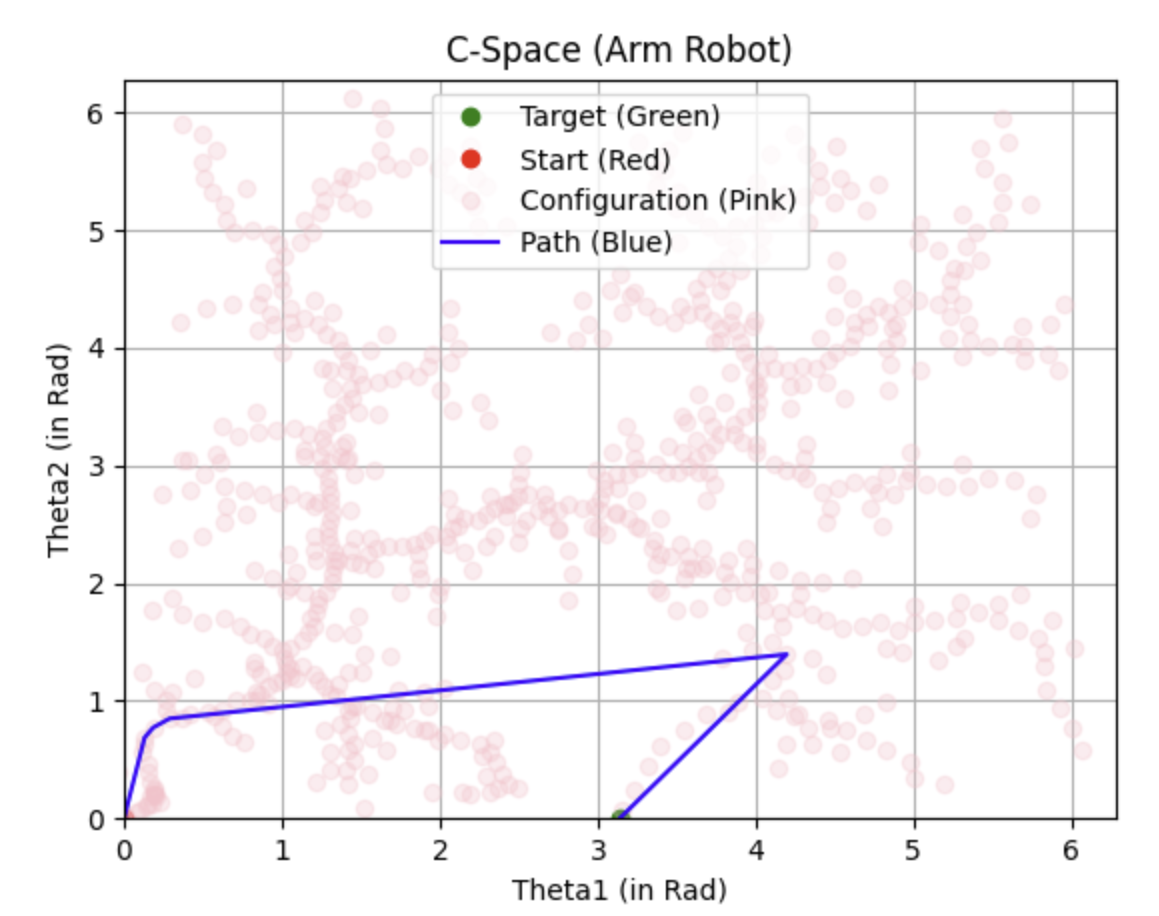
Algorithm implementation:

* Create an empty search tree starting at the starting position
* Loop the algorithm until we reach the goal position
* Select a random point around our current position, look at its neighbor cells, and connect the current to any neighbor cell where no collision occurred

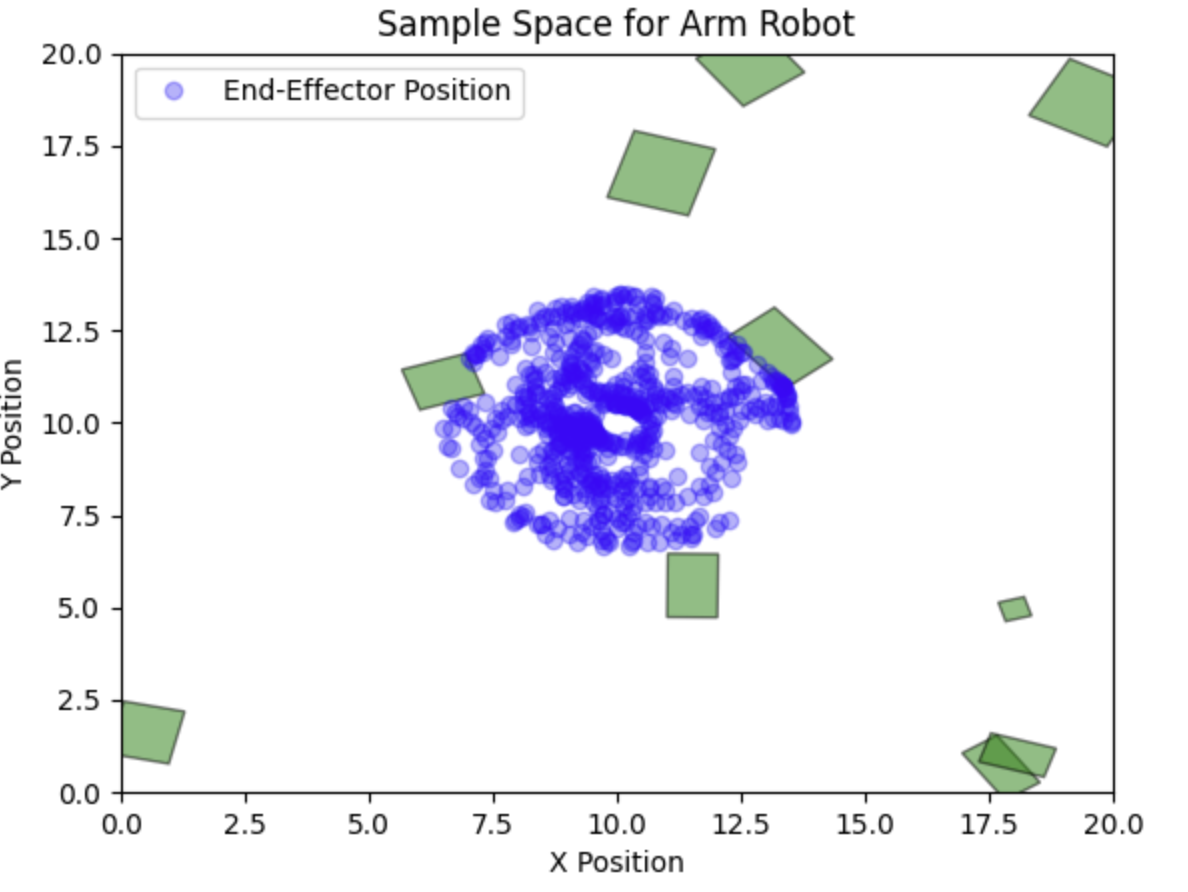
Some things to note about our visualizations here:

* Our physical environment animation for the freeBody robot seems to end about a frame early, since it looks like it stops just short of the goal position. However, you can see in c-space that the freeBody robot does in fact reach the goal
* For the physical environment animation for the arm robot, in the bottom left corner you can see the RRT tree expansion in terms of theta1 and theta2 (which takes quite some time of the animation, later in the animation you can see the robot move to the goal). This is better visualized in c-space for the robot arm

RRT Nodes and Connections for environment5 (0,0) to (3.14, 0):

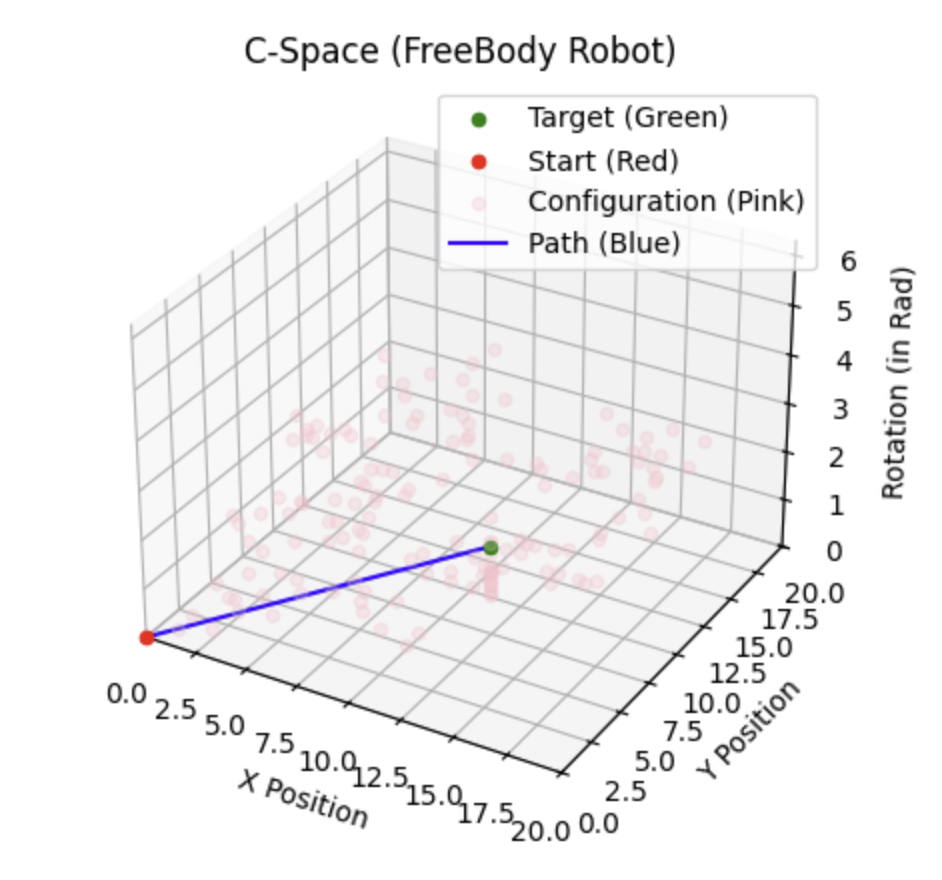


Samples:

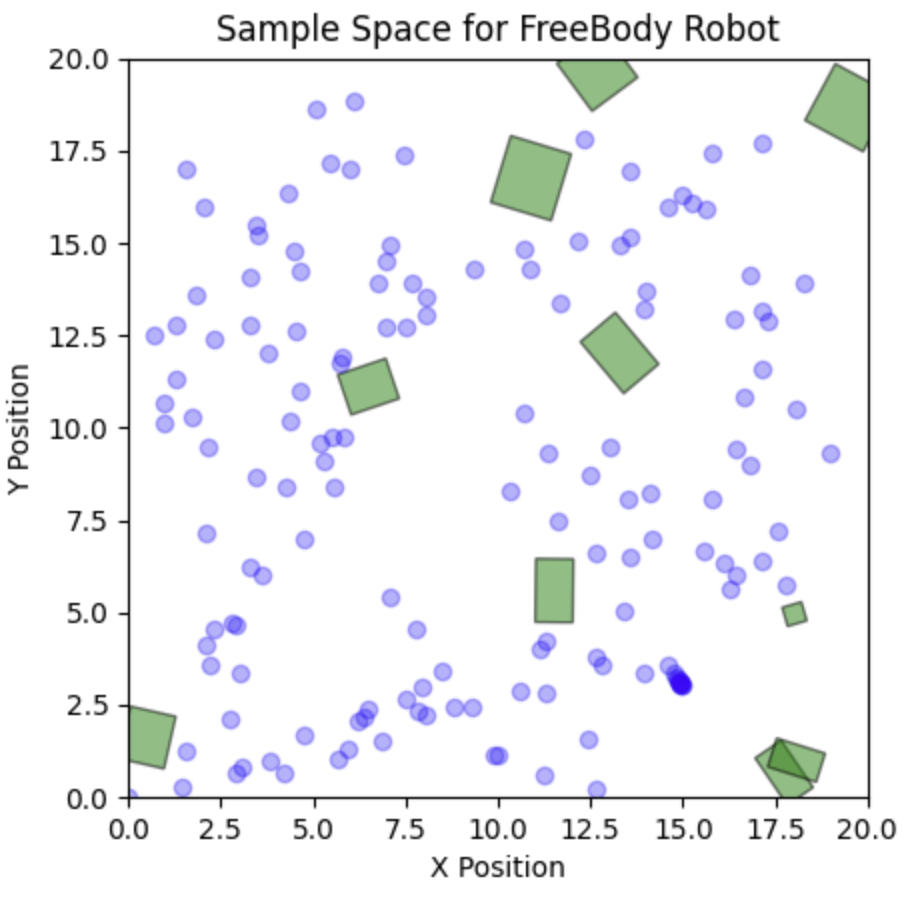


Animation can be seen in folder;

RRT NOdes and connection for freebody( 0,0,0) to (15, 10 ,1.57):



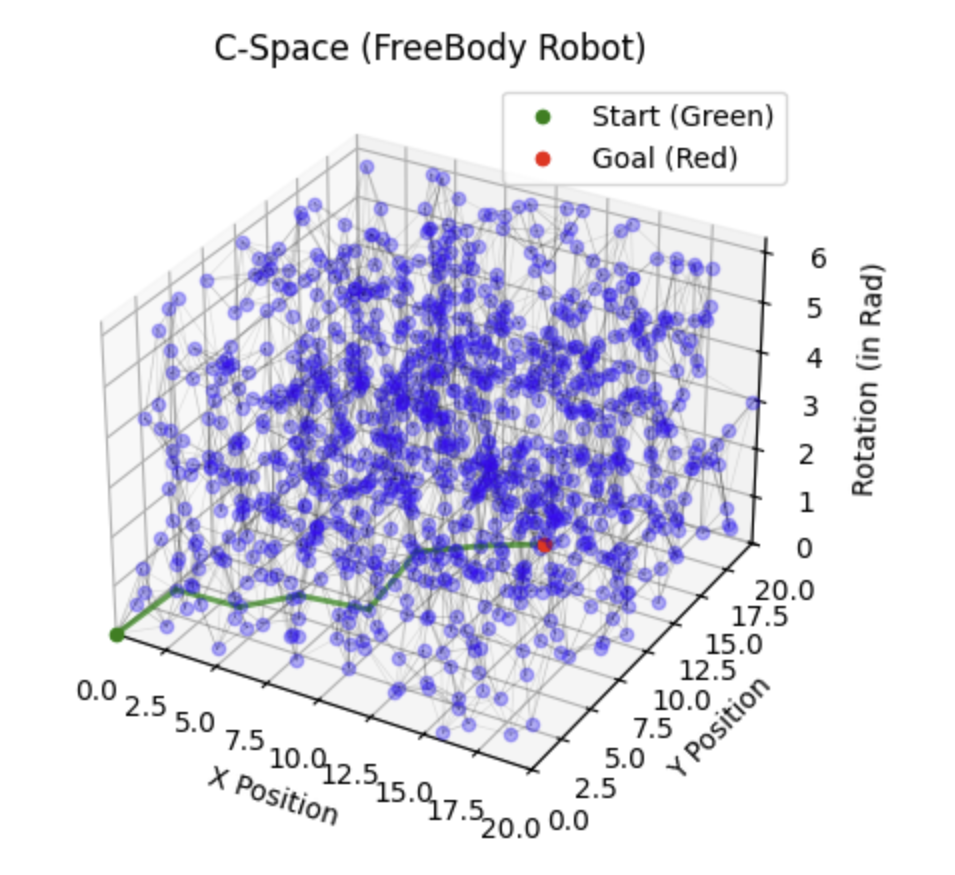
Sample space for freeBody:



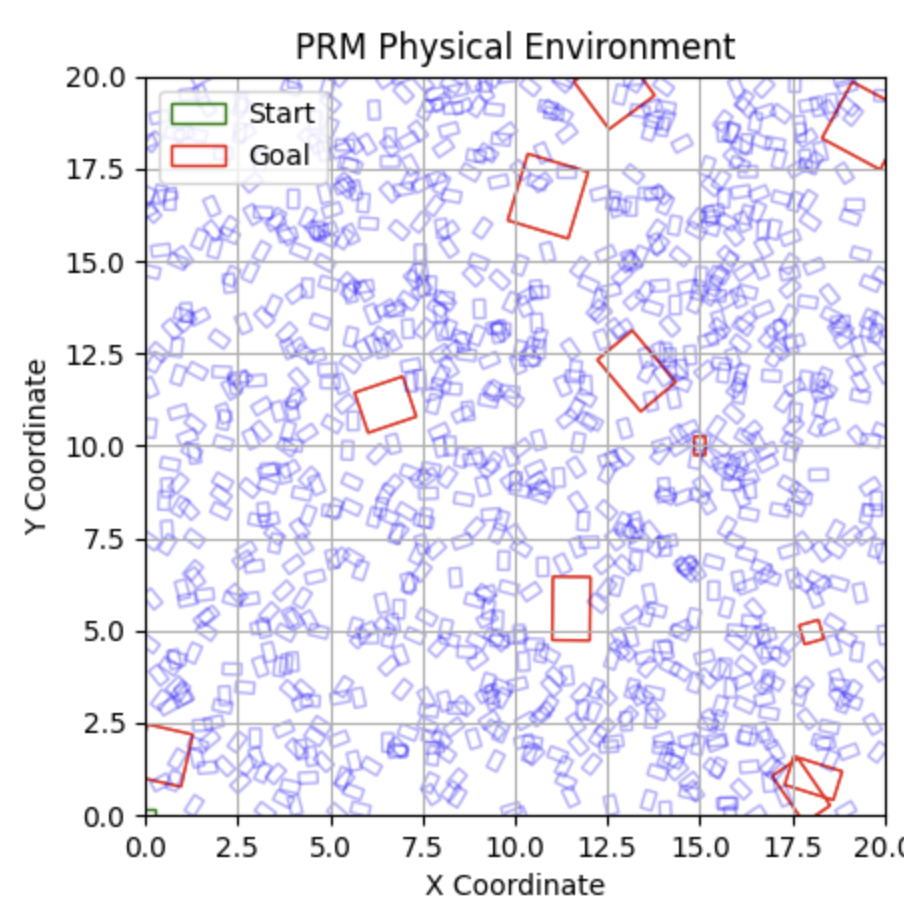
**5) PRM\***

PRM\* is an iteration of PRM where we incrementally improve connections in our roadmap. This ensures that as we approach an infinite number of nodes generated, we get closer to the optimal path.

C-Space of nodes and connections for freeBody from (0,0,0) to (15,10,1.57):



Samples:



Animation can be seen in folder aswell.

**6) Comparison of Planners**

We pretty much do exactly as requested in the writeup, we do 10 runs with a max of 500 iterations on 5 environments with 5 start/goal pairs for PRM and RRT. We do this through an evaluation function and print out a data frame containing the average results from the testing.