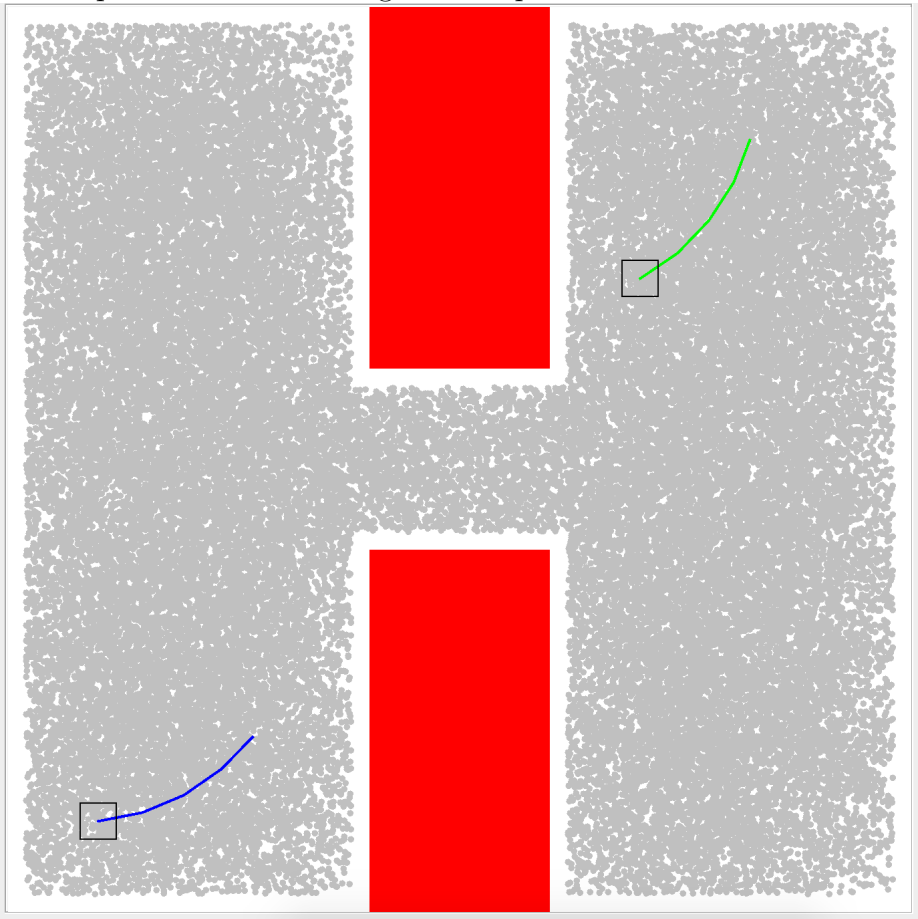
COMP3702 - Assignment 2

Roy Portas - 43560846

September 21, 2016

1 Chosen Configuration Space

The chosen configuration space will be the area containing all valid chair movements. Below is an example of what the configuration space will look like.



In the image above, the grey dots are the chair basepoints generated by the sampling strategy. It can be seen that the border of the grey area will be the configuration space, as the area outside of it is invalid for the chair to move.

2 Sampling Strategy

There are various sampling strategies that could work for the given problem, most notably:

- Sampling Near Obstacles

  This method is promising, since we are given all the obstacles at the start. Thus this sampling only needs to be done once. Furthermore, sampling near obstacles will reduce the number of nodes required and make the graph more sparse. A large distance between two nodes suggest that the space does not have a collision, so the algorithm can move the chair in a straight line towards the node. Upon reaching a more dense section of the graph, the algorithm will use the nodes near obstacles to navigate around it.

- Sample Inside a Passage

  The obstacles for our problem can be sparse, with large distances between obstacles. Thus this strategy would not be an optimal choice.

- Using Workspace Information

  This strategy also looks promising, as we are given the complete workspace. Thus a possible solution is to sample the workspace randomly to create a discrete grid. However this will create a lot of nodes, which will slow down the search algorithm. Furthermore, this sampling strategy will create nodes in the empty spaces in the corners of the workspaces, these nodes are irrelevant and will only slow the search algorithm.

The Sampling Near Obstacles was chosen because of the configuration of the workspace and the chair.

2.1 Implementation

The strategy was implemented using the following pseudocode:

```
N = Number of required samples
samples = List of valid samples

while samples.length < N:

    x = Random configuration near an obstacle
    if x is a valid configuration:
        add x to samples
```

The end result of this will be a list of valid configurations in the workspace. The algorithm will then calculate the distance from each sample to the destination use that information to sort the list of samples. Thus at the end we have a list of samples with the closest first.

By having the closest samples first in the list, the algorithm will check these points for a solution first, which will provide a solution which is quicker to compute and, in theory, produce a more optimal solution.

Additionally a queue is created for running a BFS search through the nodes. The algorithm then iterates through the list looking for valid paths between the nodes. If a valid path is found, it will be added to the queue. The program will continue to search through the queue until the destination is found. Once it is found, the algorithm will backtrace to find the solution.

3 What will cause the program to succeed

The algorithm is highly dependant on the sampling strategy, if the random distrubution of points is good, the algorithm is more likely to succeed.

In theory as long as there is a number of valid configurations between the start and destination, the program will succeed. However the chances are increased if the number of samples is larger.

Because of the sampling strategy used, the less joints the arm has the easier it is to find a valid path. However the algorithm will attempt to correct the joints and gripper after every move, allowing the algorithm to find valid paths in the harder problems with more joints.

This can be shown through the results of the algorithm. Running the `1_joint.txt` takes less than a second, while `4_joints.txt` takes 21 seconds, which is a significant difference. Thus increasing the number of joints will make the problem harder to solve.

4 What will cause the program to fail

The program will fail if the number of samples is insufficient. Furthermore, since it depends on sampling near obstacles, if the obstacles are too close together the sampling will fail.

The program is more likely to fail if the number of joints is high, since it will attempt to find a valid path between random configurations, and as the number of joints increase so does the difficulty in moving from one configuration to another along a path.

Additionally, if the workspace is extremely big it will take a long time to search through each node. Meaning the program will likely run out of memory before finishing.

As with before, it can be shown through experimentation that increasing the complexity will make it harder to complete, for instance the `4_joints.txt` takes 21 seconds, while the `gripper_4_joints.txt` takes minutes to complete.