**Documentation for Problem 2**

**By Manish Kundu**

This problem involves using RRT for planning a path from a source point to a destination point in IIT Kharagpur.

**IMP – When executing the program, you need to press any key on the keyboard twice to get to the window showing the path generation.**

1. The first step involves processing the google map. There are many labels on google maps which are undesirable. So, an online tool which removes labels from google maps was used to get a map without labels.
2. The roads in the image without the labels already have roads in white colour. So, an array of the dimensions of the map was created filled with 0s (The map is a coloured h\*w\*3 array but since we need a binary image, our new array is 2d with dim h\*w).
3. We then traverse through the map and then if we get [255,255,255], then we get the corresponding coordinates in our binary image to 255 for highlighting the roads with white. All the buildings and other things are in black, acting as obstacles.
4. Now we implement our tree. We have our tree variables like parent, children list for each node, the coordinates of the node and the image on which we are drawing the circles and lines to represent the nodes and edges. We also have list for storing the cost of each node from source and size of the tree.
5. The add\_node(self, coordinates, cur\_parent) tree function simply takes the coordinates of the node and current parent, adds an empty list to the children list for that node and adds the node’s coordinates in the coordinates list. The node is added to its parent’s children list and the cost of reaching the node is also stored.
6. The ch\_parent(self, index, new\_parent) is used to update the parent while running RRT\* when we add a new node whose cost is more efficient with some other neighbourhood node than the current parent. This is known as rewiring the tree
7. The add\_dest(self, cur\_parent) is executed when we reach in the radius of the destination. It adds the destination as a node sets the variable dest\_reached as True.
8. The sample\_point(self) simple returns any random coordinate on the map.
9. The find\_nearest(self, coordinates) takes any coordinate as input, then checks all possible valid nodes in the existing map to which an edge can be drawn and then returns the nearest valid node(if any). To check the valid nodes, Bresenham’s Algorithm is used to generate the intermediate pixels between the two nodes.
10. The Bresenham’s Algorithm finds the intermediate pixels on the line without using floating point arithmetic and rounding to integer which makes it more efficient. We always increase x by 1 and for y we keep track of the slope error. If the slope error becomes greater than 0.5, then the line moves upwards by one pixel and we increment y by 1. Otherwise y is not incremented.
11. The line\_segment\_is\_unobstructed() function uses Bresenham’s Algorithm to get intermediate points and checks whether the point lies on path or on obstacle.
12. The Render(self) function is simply used to mark the nodes on the map, mark the paths traversed in green and finally when destination is reached, it marks the path found by RRT\*.
13. The movement\_towards\_unit\_vector(point1, point2, move\_distance) takes two valid nodes and a threshold. Then if the distance between the two nodes is less than the threshold, it returns point2. Otherwise, it returns the point in direction of the line made by point1 and point2 but with distance from point1 as the threshold.
14. The RRT() function implements the RRT algorithm for path planning. We generate a new point and check whether it is a valid node. Then we find the neartest existing node and attach it to the new node by chaining a link of discretized nodes. We keep checking if our chain is valid or not (passing through obstacle or not). Otherwise we sample a new point and continue.

**Challenges:**

1. RRT does not find the most optimal path always. RRT\* is better, more optimised version of RRT which theoretically is ran for infinite time gives the most optimal path.