Assessment task 2: Plan a path for a mobile robot

By Md Shawkath Hossain (Student Id 12631558)

Introduction

This assignment is planning a path for a mobile robot within an environment. To do so, I have followed the tasks assigned in the assignment sheet. To achieve the goal of path planning for a mobile robot, I have completed 5 assigned tasks which are Find the start and goal nodes, Implement A*, Generate the path, Smooth the path, and Create a PRM graph. After implementation, I have built the Python code and run the code into a simulator to simulate the robot path planning. According to the simulation it was working well. However, the PRM graph is not working well in my implementation. I have created a demo video of the simulation which could be found in the following link:
https://www.youtube.com/watch?v=Bu06QRZx-Y4

Finding the start and goal nodes

In this section I identified two points first by iterating over the nodes array then identified Euclidean distance between each pair of points.

```
292             ## Implemented ##
293             ##########################
294
295             point1 = np.array((self.nodes_[i].x, self.nodes_[i].y))
296
297             dist = np.linalg.norm(point1 - point2)
298             if dist < best_dist:
299                 best_dist = dist
300                 best_index = self.nodes_[i].idx
301
302
```

Implement A*

Here implemented A* algorithm for searching in graph. This is efficient than Dijkstra's algorithm because it reduces the number of nodes visit. It uses heuristic, which is a function that provides an estimated value for the quality of a node. Here it has two arrays for visited and unvisited nodes. These arrays are updated as search grows.

```
        ##########################
        ## YOUR CODE GOES HERE ##
        ##########################

        node_idx = self.get_minimum_cost_node(unvisited_set)
        node_idx = unvisited_set[node_idx]
```

```
        ##########################
        ## YOUR CODE GOES HERE ##
        ##########################

        visited_set.append(node_idx)
        unvisited_set.remove(node_idx)



    ##########################
    ## YOUR CODE GOES HERE  ##
    ## FIX THE IF CONDITION ##
    ##########################
    if node_idx == goal_idx:
        rospy.loginfo("Goal found!")
        return



          ##########################
          ## YOUR CODE GOES HERE  ##
          ## FIX THE ?? BELOW     ##
          ##########################
          if cost < neighbour.cost :
              neighbour.parent_node = self.graph_.nodes_[node_idx]
              neighbour.cost = cost




      else:

          # Add it to the unvisited set
          unvisited_set.append(neighbour.idx)

          # Initialise the cost and the parent pointer
          # hint: this will be similar to your answer above

          ##########################
          ## YOUR CODE GOES HERE  ##
          ## FIX THE ?? BELOW     ##
          ##########################
          neighbour.parent_node = self.graph_.nodes_[node_idx]
          neighbour.cost = cost



    ##########################
    ## YOUR CODE GOES HERE  ##
    ##########################

    cost = self.graph_.nodes_[node_idx].cost + neighbour_cost + self.heuristic_weight_ *
        neighbour.distance_to(self.graph_.nodes_[goal_idx])
```

Generate the path

After A* search is completed, I have got the nodes with less costs from start to goal. Then looped backward to the start from goal to present the path.

```
        ##########################
        ## YOUR CODE GOES HERE ##
        ##########################

        while current.parent_node is not None:
            path.append(current.parent_node)
            current = current.parent_node
```

Smooth the Path

After drawing the path, it shows that there are so many sharp turns in the path which is not friendly for a robot. So, using the equation provided in the Assignment sheet, I have smoothened the path. In demo it is shown as red path.

```
changes = 555.0
e = 0.001

while changes >= e:
    changes = 0.0
    for x in range(1, len(path_smooth)-1):
        newPointX = path_smooth[x].x - (alpha + 2*beta)*path_smooth[x].x + alpha*path_nodes[x].x +
            beta*path_smooth[x-1].x + beta*path_smooth[x].x
        newPointY = path_smooth[x].y - (alpha + 2*beta)*path_smooth[x].y + alpha*path_nodes[x].y +
            beta*path_smooth[x-1].y + beta*path_smooth[x].y

        #path_smooth[index] = np.subtract(current, addition)

        blocked = is_occluded(self.graph_.map_.image_, [path_smooth[x-1].x, path_smooth[x-1].y], [newPointX,
            newPointY])

        if blocked == False:
            changes += (newPointX - path_smooth[x].x)**2 + (newPointY - path_smooth[x].y)**2
            path_smooth[x].x = newPointX
            path_smooth[x].y = newPointY
```

Create a PRM graph

I have created nodes for PRM graph, but it is not properly working. When I test without PRM graph the path smoothing works very well. However, with PRM graph it is not working

properly as there are some issue in the implementation with random placement of nodes in free space.

```
        ########################
        ## YOUR CODE GOES HERE ##
        ########################

        for x in xrange(self.map_.min_x_, self.map_.max_x_-1, self.grid_step_size_):
            for y in xrange(self.map_.min_y_, self.map_.max_y_-1, self.grid_step_size_):

                if rospy.is_shutdown():
                    return

                # Check if it is occupied
                occupied = self.map_.is_occupied(x,y)

                # Create the node
                if not occupied:
                    self.nodes_.append(Node(x,y,idx))
                    idx = idx + 1
```

Conclusion

The implementation shows it is working well except few situations. It could be improved by implemented PRM graph properly.