EECS 195: Autonomous Systems - Spring 2020
Assignment #3
Due Date: 6/7/2020

Problem 1: Tune PID Controllers (40 points)

In this problem, we are going to use TurtleBot3 simulator to tune a PID controller. In your workspace, create a new node named "`PID_Controller`". This node shall subscribe to the following topics:
/reference_point: this topic contains information about the new pose (x,y,theta) that the robot shall visit. It also contains another parameter called "mode" (described below)
/slam_pose: this topic contains information about the current robot position as estimated by the Hector SLAM algorithm.

This node shall publish to the topic **/cmd_vel** which is used to assign linear and angular velocities. The node shall implement the PID controller that was discussed in Lecture 12 to move the robot from its current pose to the reference pose. In particular, it should implement two PID controllers, one that controls the angular motion, while the other one controls the linear motion.

Whenever the "mode" is set to zero, then the "`PID_Controller`" shall activate the PID for the angular velocity first until the robot faces the reference point, followed by activating the linear velocity controller until the robot gets to the reference point, and finally activate the angular velocity controller again to turn the robot towards the final angle. Whenever the "mode" is set to 1, then the `PID_Controller` shall activate both the angular and the linear controller simultaneously trying to control both the angle and the position of the robot to get to the final pose.

You need to pick different combinations of the P, I, and D for the angular velocity controller and the linear controller. You need to try different values until you get an acceptable result.

Evaluation:
The grader is going to publish messages on the **/reference_point** topic and compute the time for the robot to get to the reference point using both modes. The grader will perform this step 5 times for different reference points and grades are going to be assigned based on the time the robot takes to get to the point.

Problem 2: Path Planning Using RRT (40 points)

In this problem, we are going to implement the RRT algorithm for path planning. In your workspace, create a new ROS node named "`RRT_node`". This node should subscribe to the following topics:
/map: this topic is used to publish the current occupancy grid computed by the Hector SLAM
/slam_pose: this topic contains information about the current robot position as estimated by the Hector SLAM algorithm.
/target_pose: this topic is used by the user to specify the target point for which the robot should visit.

And publishes to the following topic:
/trajectory: this topic should contain the trajectory (an array of points) that needs to be visited by the robot in order to move from its current position to the target point.

This node shall implement the RRT algorithm discussed in the lecture (assume that the robot is a circle of width = 0.5 meters). As discussed in the lecture, the RRT algorithm computes a tree data structure with its root is set to the current robot pose. It then expands the tree by adding more nodes to it. To structure your code, you should implement the following functions that are needed for RRT algorithm to work:

- Random_configuration: this function generates a new random configuration in the configuration space. As discussed in the lecture, the configuration space is 2-dimensional. So this function is going to generate a random (x,y) point
- Nearest_vertex: this function takes as input the randomly generated configuration. It shall go through all the nodes in the tree and compute the euclidean distance with respect to the randomly generated configuration. The function should return the nearest node in the tree.
- New_configuration: this function takes as input the randomly generated configuration along with its nearest node in the tree. If the Euclidean distance between the two is smaller than a certain threshold (you are free to pick this threshold), then this function should return the randomly generated configuration. If not, then it should compute the nearest point to the randomly generated configuration whose Euclidean distance is less than the threshold.
- Add_vertex: this function should add the node computed by the "New_configuration" to the tree.

Your algorithm shall use the four functions above to add nodes to the tree until you add a node that is "close" to the target pose (i.e., the Euclidean distance between one of the nodes in the tree and the target pose is less than the threshold) or you reach a max number of iterations. Finally, your algorithm should find the trajectory between the root and the target. This can be done by starting from the target node in your tree and go up in the tree until you get to the root.

Evaluation:
The grader is going to publish messages on the **/target_pose** topic and compute the time your algorithm will take to compute the final trajectory. The grader will perform this step 5 times for different target points and grades are going to be assigned based on the time the robot takes to compute the trajectory along with the correctness of the computed trajectory.

Problem 3: Full Path Planning Stack (20 points)

In this problem, we are going to integrate the code from Problem 1 and Problem 2 to implement a full path planning stack. In your workspace, create a new ROS node named "`PathPlanner_node`". This node should subscribe to the following topics:

/trajectory: this topic is used by the "`RRT_node`" node to publish the computed trajectory
/slam_pose: this topic contains information about the current robot position as estimated by the Hector SLAM algorithm.

This node should publish to the following topics:
/reference_point: this topic contains the next pose (x,y,theta) that the robot shall visit along with the "mode" parameter.

In particular, this node is responsible for waiting for trajectories to be computed by the `RRT_node` and publishing the poses of this trajectory one at a time on the /reference_point topic. In particular, this node should monitor the pose of the robot to decide when to publish the next reference point. [Optional] this node can also use the **/map** or **/scan** topic to decide on the "mode" that shall be used to go from one point to another.

Evaluation:
The grader is going to publish messages on the **/target_pose** topic and monitors the robot movement. The grader will perform this step 5 times for different target points and grades are going to be assigned based on the time the robot takes to move along the trajectory.