CSCI 445 Final Project Writeup

Girish Mahajan, Lucas Makdessi, Efaz Muhaimen, Leo Zhuang

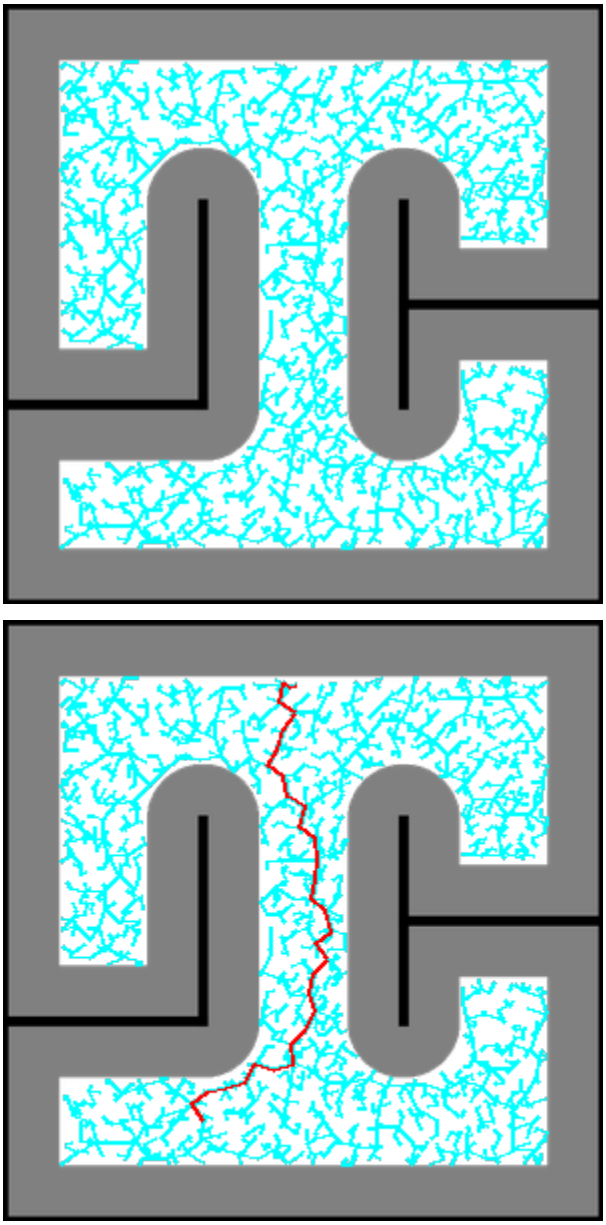
I. Overview

For the final project, we have been tasked with combining our previous work with the iRobot Create2 to develop a program that transports an object through a maze. Once the robot reaches the goal point, a robot arm will grab the object and place it on a shelf. To facilitate this goal, our group utilizes path finding, path following, and arm movement algorithms we have learned from class and developed in other assignments. These algorithms consist of RRT, Dijkstra's algorithm, PID controllers, odometry, particle filters, and inverse kinematics to efficiently and accurately achieve the goals of the project.

II. Path Finding

Our path finding algorithm uses a combination of RRT and Dijkstra's algorithm to find the shortest path from the robot's starting point to the goal point. We chose to use an RRT for path planning since it is designed to efficiently find paths and explore unexplored areas. To create our RRT, we randomly select a point in the map and find the closest point on the tree. We then choose a point on the line from the point on the tree to the random point such that the distance between the two is less than or equal to our step size. We then connect this point to our tree if there are no obstacles between it and repeat this for multiple iterations. Once our tree is completed, we run Dijkstra's algorithm to find the shortest path from the starting point to the goal point on the tree. The algorithm iterates through each node from the starting node to find the distance to the neighbors and get the closest neighbor for each node that hasn't already been explored. Once the algorithm finishes assigning distances and closest nodes, it runs through the completed map to find the path with the shortest distance between the start and goal nodes, or a node close to the goal which is extended to the goal node. To do this, we assume that the tree will reach a point close to the goal point because of the amount of iterations and step size we have hard coded. Our path finding algorithm was very successful at finding paths from the start to goal points in an efficient manner. From the image of our tree, you can see that it was able to explore the majority of the map and find the shortest path from the start to goal points. For our RRT, we set the number of iterations to 5000 and delta to 10. This kept each point close to each other so we did not have any parts of the RRT that went to walls. 5000 iterations meant that there would be enough branches so our optimal path would be close to the global optimal.

A. Image of tree and shortest path

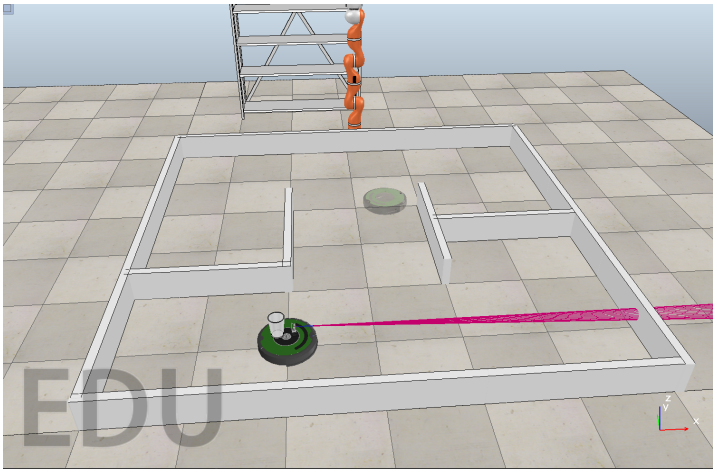


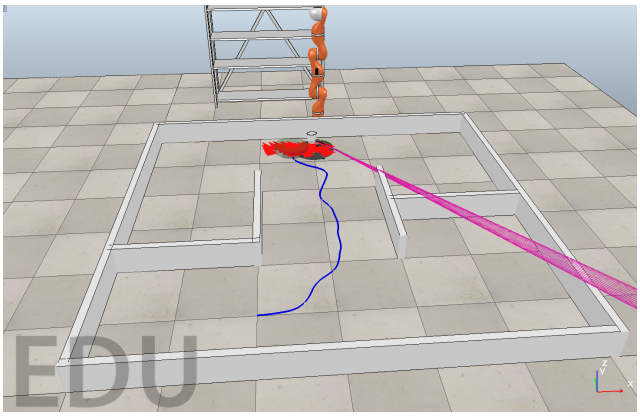
III. Path Following

To facilitate accurate path following, we utilize a PID controller with odometry as well as a particle filter to account for odometry inaccuracies that may arise over time. The PID controller minimizes instantaneous error, cumulative error, and oscillations since we need to maintain the path the robot follows. To use the PID controller, we integrate it with an odometry module to estimate the position and heading of the robot using encoders. The odometry module uses encoder counts from the motors and uses odometry formulas with the wheel size and distance to calculate the change in position and orientation of the robot. With the odometry module, we can determine how far away we are from the goal and combining it with our PID controller allows us to ensure the robot is heading in the right

direction. The PID controller uses errors between the calculated position from odometry and the goal position on the path to modify the values being sent to motors. To supplement our PID controller, we implemented a particle filter to ensure accuracy over time because of inaccuracies that odometry introduces. We set up certain intervals in the code where the robot would update its particle filter using odometry data and the distance to the wall in front of it measured by its sonar sensor. By passing this data to the particle filter, we can create a reasonable estimate of the robot's position using probabilistic localization. The particle filter keeps an array of particles that are affected by the locomotion and sensor readings of the actual robot with slight randomized variances to estimate the robot's position. This allows us to account for any small inaccuracies from components that can accumulate over time. For our PID controller, we set the kp to 90, ki to 1, kd to 60, ki was clamped to [-3, 3], output range was clamped to [-50, 50], and step size was 10. For our particle filter, we set the number of particles to 100 translation variance to 0.06, rotation variance to 0.05, and measurement variance to 0.05.

A. Images of path followed, possibly graph of errors, estimated position

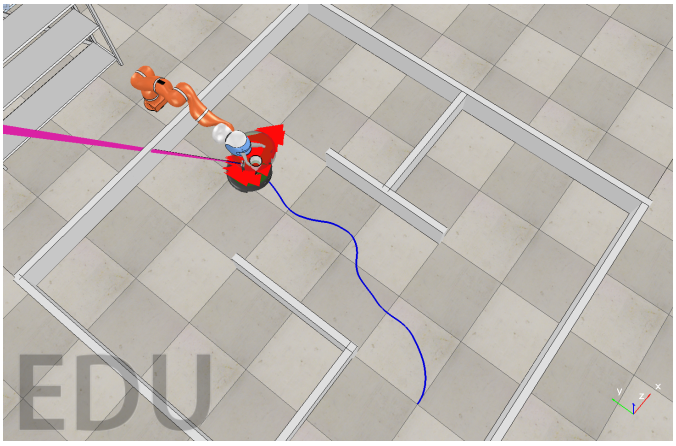


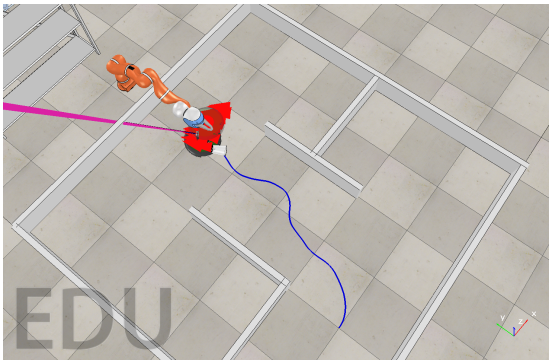


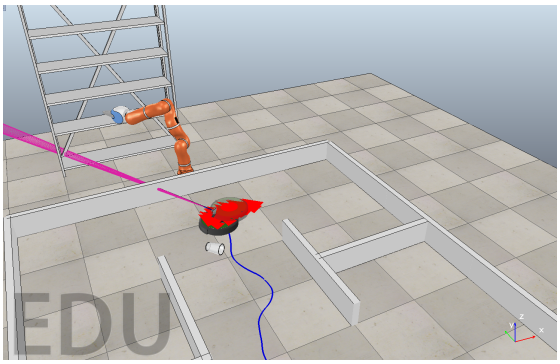
IV. Arm Movement

To control the robot arm in a 3D space and move the cup from the robot to a shelf, we utilize inverse kinematics as it allows us to accurately move the arm given a goal location. However, inverse kinematics is not solvable for more than 2 joints so we treat it as a revolving 2D plane rather than a 3D space. This allows the arm to move to a desired position even if the cup is not directly in front of the arm by rotating the bottommost actuator to ensure the cup is in the plane. Once the cup is in the plane, we use 2D inverse kinematics to move the gripper to the cup's location on the robot and lift it up. Then we rotate the plane once more so the goal location on the shelf is in the plane and rotate the bottommost actuator accordingly. We again use 2D inverse kinematics to move the cup to the goal location on the shelf and release it. Our inverse kinematics uses arms lengths measured from the simulation and applies them to the inverse kinematic equations from class. Because the arm is not at the center of the simulation, we have hard coded offset values to be used in our calculations based on its position. Although our arm movement works as expected, it seems challenging to grab the cup because of simulation challenges. In our tests, we get slight inaccuracies with the calculated final location of the robot which can be as small as 0.02 between actual and calculated position, but sometimes results in the gripper slightly touching the edge of the cup and launching it away. Although this is partially due to odometry inaccuracies, we still get very close to the cup's actual position and a real world test would probably succeed since it would just slightly touch the cup's edge. For our inverse kinematics calculations, we set the length of arm link 1 to 0.5, length of arm link 2 to 0.39, the initial x position to 1.6001, and the initial y position to 3.3999. Additionally we assume the cup is at a height of 0.2, and the 3 shelves that can be reached are at heights 0.21, 0.53, and 0.85.

A. Images of arm reaching cup and various locations on the shelf




EDU


EDU