ECE 5463 Final Project Obstacle-Aware Motion: Final Project Group 8

Krish Mehta, Ethan Summers, Ayush Chakravarty

https://github.com/thnsmmrs/CV-Based-Obstacle-Aware-Planning

1. Description of Task

The task for this project was to move from point A to B while avoiding a circular or rectangular obstacle in the workspace. The team went with a user-defined camera-based input approach to simulate a 3R robotic manipulator operating in a static obstacle workplace. A camera captures the workspace containing multiple physical obstacles placed in the area using background subtraction, pixel differentiation, and morphological operations. The user clicks anywhere in the video output workspace to specify an end goal for the robot (Point B). The system then runs an RRT adapted program to generate a tree of nodes at a given sampling frequency to find a path to the end goal. The motion planning algorithm is run 50 times with the given obstacle workspace and chooses the shortest one as the simulated path. The path is interpolated with a spline to minimize jerk and preserve hypothetical hardware. This spline is traced by the end effector of the end effector from start to finish. The program handles cases for out of bounds or assembly mode changing IK, and no path found RRT cases. Due to the nature of the planar workspace, the real-world setup of this design would use a prismatic joint on the end effector, and the 3R arm would operate above the obstacle plane for pick and place tasks.

2. FK and IK Method Summary

Forward kinematics is utilized to calculate the joint positions from angles using geometry. Starting at the base (x_0, y_0), each joint is calculated sequentially: $x_1 = x_0 + L_1 \cos(\theta_1)$, $x_2 = x_1 + L_2 \cos(\theta_1 + \theta_2)$, $x_3 = x_2 + L_3 \cos(\theta_1 + \theta_2 + \theta_3)$, with the y-counter parts being the same except using sin(). As a result, the forward kinematics function outputs all of the positions starting from the base to the end effector.

Inverse kinematics is utilized to calculate the joint angles $(\theta_1, \theta_2, \theta_3)$, to reach a desired target position (x_t, y_t). Since the robot is 3R, it has three DOF but only two position constraints. Because of this, the method treats links two and three (L_2, L_3) as a single combined link (L_{23}), thus making the problem simpler. The solution process is as follows:

1. Calculating distance and angle to target: $d = \sqrt{dy^2 + dx^2}, \gamma = \arctan 2(dy, dx)$
2. Checking the reachability: $|L_1 - L_{23}| \leq d \leq L_1 + L_{23}$
3. Using Law of Cosines to find α (angle at the base relative to the target direction) and β (angle between L_1 and L_{23}

4. Calculating θ_1 and θ_2 based on the elbow assembly mode configuration (righty or elbow up = $\theta_1 = \gamma - \alpha$, $\theta_2 = \pi - \beta$; lefty or elbow down = $\theta_1 = \gamma + \alpha$, $\theta_2 = \beta - \pi$)

5. Calculating joint two position using FK

6. Calculating θ_3 by finding the target angle first (angle from joint two to target): $\theta_t = \arctan 2(y_t - y_2, x_t - x_2)$ and then $\theta_3 = \theta_t - (\theta_1 + \theta_2)$

3. **Description of Control Strategy / Planner**

The Planner utilized in team 6's final project is the RRT planner, which utilizes multiple branches from the start point to explore different paths to optimize the best route while avoiding obstacles. The team decided to use this form of a planner to optimize time and the shortest distance needed to get from the start point to the end point while still maintaining a level of difficulty appropriate for the assignment. The team discussed the utilization of other planner strategies such as RRT*, but the team decided that the RRT method would be more effective while still being modular if necessary. The development of the RRT planner was done using inspiration from other GitHub repositories along with what was learned from class.

Also utilized in the final project was the use of a PD controller for velocity propagation purposes. This was used due to the use of a three linked robot that follows the path created by the RRT planner. The development of this controller was inspired by each team member's PA#3 assignment along with the use of splines in PA#2. These programs were altered for the purposes of this project.
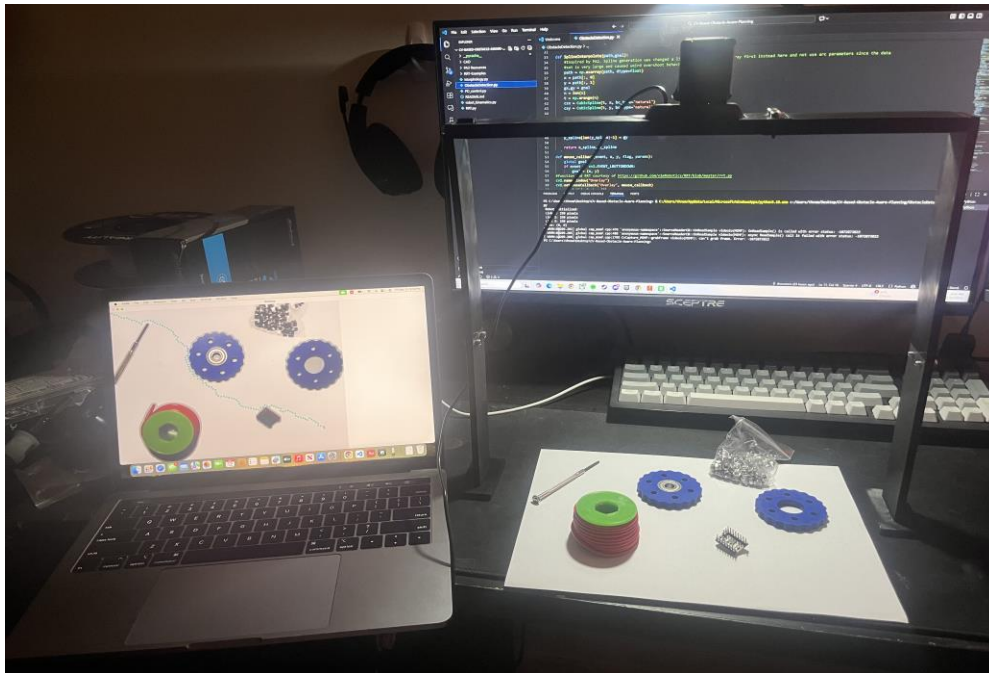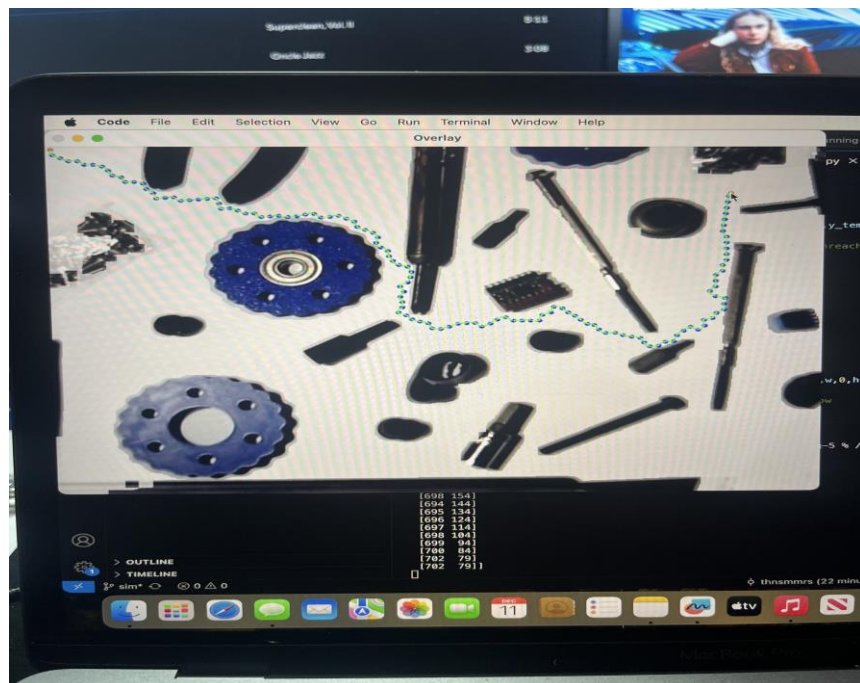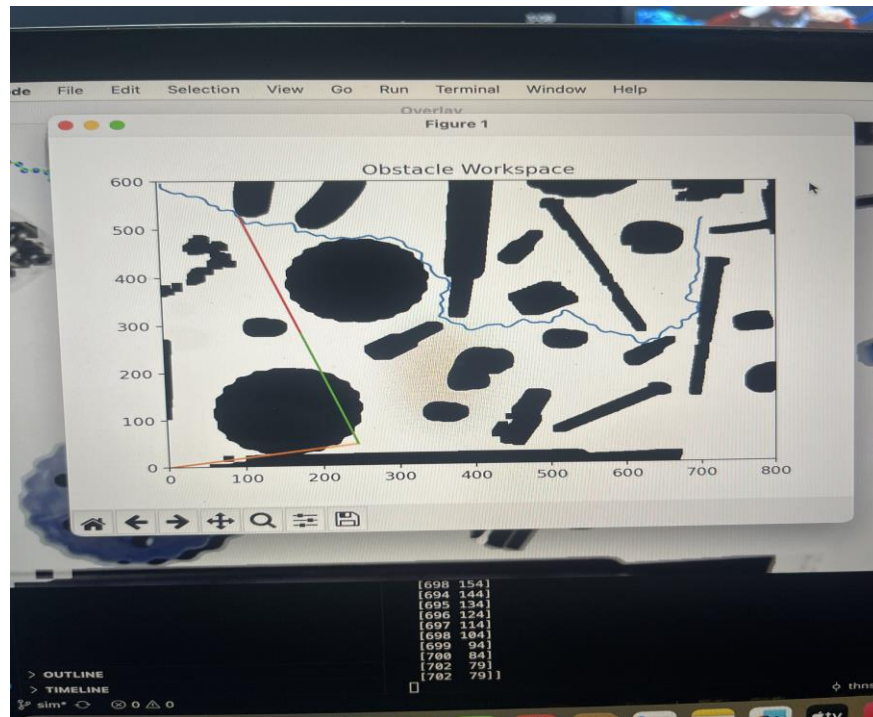
4. Screenshots of Simulation / Output Plots

Figure 1: Computer Vision Rendering with RRT

Figure 2: Computer Vision Workspace with RRT Path

Figure 2: 3R robot arm following spline in planned workspace

5. Challenges Faced

The main challenges the team came across were implementing PD control, creating a global coordinate system, and accounting for link collisions in the RRT program. Several attempts were made to create a functioning algorithm for PD control with the arrays created from IK/FK, using the main PD control law and by tuning Kd and Kp we were able to get some damping but when plotting the steady state errors we saw that they never were damped to 0 and had an almost opposite and equal error reaction between theta 1 and theta 2.

The coordinate system challenge that came up repeatedly was something we eventually were able to solve but was one of the bigger time sinks of the project. A global coordinate system in this context is necessary because we need the points used for start and goal position capture, the vision window, the RRT program, and eventually the final plot with the animation, to all match to avoid scaling issues, IK/FK discrepancies or incorrect motion planning interfering with obstacles. We were able to take advantage of the OpenCV built in coordinate system options of using pixels and the camera input resolution (w, h) as (x, y), locking the resolution at 720p and getting a resultant plane of size 800x600. By using this, we were able to keep consistent data points between the first few steps of the process but when plotting we realized that the vision

window sets +y to facing down, creating a plot that was flipped from what we expected; we were able to fix this with a simple function that flipped every y value using h – y.

Accounting for link collisions was the most disappointing part coming out of the final deliverables because it was the only original goal that could not be met. Several attempts were made, and although we only wanted to implement one obstacle in the workspace, we were never able to achieve success with the tries. The last attempt included another function in the RRT function that along with checking the EE for collision against obstacles (non-free pixels), would run IK/FK and sample multiple points on each link to test the node for reachability with no collisions. The only possible way we found to make this method work with obstacles was decreasing our node step size so much that the generated paths were no longer optimized. After brainstorming and checking out other projects online, I believe using joint space could have been the solution to this problem but for our use case it is hard to be sure that it is even possible to do what we had imagined.