Final Project - Robotics I

Wyeth Michaelsen , Ethan Nguyen

Abstract

The goal of this project was, given a set of locations, to create an algorithm that designed a trajectory for the UR5 robot to maneuver its end-effector between two shelves of a shopping aisle in order to move objects from one shelf to another. The algorithm is given an array of numbers each corresponding to one of twelve preset locations on the aisle (three on the top and three on the bottom of each shelf), and outputs a trajectory which demonstrates the robot's ability to move to and from the specified items. To make this a reality, we used the Modern Robotics libraries and functions presented in class; as well as the python files UR5 and Serialrobot which were provided to us by Professor Roth.

1 Approach and Method

As we outlined our plan for this project, our biggest technical concern was trying to make the UR5 behave as realistically as possible. This meant we had to determine the most effective trajectory generation method to allow the robot to move between the shelves while keeping the end-effector in the upright position. We also knew that the robot couldn't skip or jump during the trajectory as that would never be a practical implementation of this type of robot in the real world. The first objective we tackled was to streamline the generation process by condensing our trajectory generation into one function. Building off of a previous assignment, we decided to continue using the end-effector frame which contains the rotation and translation matrix to express each of the 12 positions. Resulting in a easy way to scale the number and locations of positions.

The program generates the trajectory through the implementation of three functions from the Modern Robotics package: *JointTrajectory*, *ScrewTrajectory*, and *IKinBody*. As shown below, the function *movePositions* takes the start and end configurations of the end-effector and generates the screw trajectory. The result is passed to the *inverseKin* function to calculate the joint angles. The last set of joint angles are then used to recalculate the trajectory using the *jointTrag*

function which is then passed to the caller to be animated. Additionally, this function returns the set of joint angles in order to be used in the next position command.

```
def movePositions(Xstart, Xend, currAngles, frameInst=100,
   quarterInt=2.5):
  se3Res = mr.ScrewTrajectory(Xstart,Xend,quarterInt,frameInst,5)
  screwAngleResult = inverseKin(se3Res,currAngles)
  joinTrag = jointTrag(screwAngleResult[-1], currAngles)
  return joinTrag, screwAngleResult[-1]
```

Code Snippet 1: Trajectory Generation Function

```
def inverseKin(se3Res, prevTheta = np.zeros(6), eomg=0.1, ev=0.1):
  thetaResCir = []
  for i~in range(len(se3Res)):
    thetalist, success = mr.IKinBody(UR5.BList, UR5.Mb, se3Res[i],
        prevTheta, eomg, ev)
    thetaResCir.append(thetalist)
    prevTheta = thetalist.copy()
  return thetaResCir
```

Code Snippet 2: Inverse Kinematics Function

```
def jointTrag(targetJointAngles, currJointAngles, frameInst=100):
  curr_pos_mod = np.mod(currJointAngles+np.pi, 2*np.pi)-np.pi
  targ_pos_mod = np.mod(targetJointAngles+np.pi, 2*np.pi)-np.pi
  joinTraj = mr.JointTrajectory(curr_pos_mod, targ_pos_mod, 2.5,
     frameInst,5)
  return joinTraj
```

Code Snippet 3: Joint Trajectory Generation

2 Results

Through the process of using the screw trajectory to then input into the joint trajectory, our animation of the robot moving between shelves looks smooth and works well with only one major caveat. Originally, our goal was to maintain the end-effector's upright position throughout the duration of the robot's movements and for the most part the UR5 does that.

However, we discovered in some edge cases the trajectory generation would result in a joint moving in the opposite direction, the long way around, to accomplish the move command. The end-effector no longer maintaining an upright position as outlined in our project goal.

We speculate that this issue is caused by a joint angle from the inverse kinematics that was out of phase by 2pi with the angle the robot was currently in. Due to the limited time, we were unable to implement a solution. We believe one successful approach would be to check the phase of each joint relative to the robot's current configuration; adjusting as necessary to provide the angle with the shortest travel.

3 Code Instructions

The files provided are as follows:

- Demo.py
  - This is the generation code and should be run with the terminal command $python\ Demo.py$ after prerequisite libraries are installed
  - Two files are then created with the trajectories. "Final.csv" is formatted for the UR5/matplotlib animation library while "coppelia_final.csv" is formatted for the Scene 2 scene of CoppeliaSim.
- The following files are libraries "Demo.py" utilizes
  - serialRobot.py
  - UR5.py
- The following videos are provided to demonstrate the project
  - "E399 Final Demo Video.mp4"
    - This video illustrates successful movement in line with the project goals.
  - "E399 Final Demo Video - Transistion Bug.mp4"
    - This video illustrates the main issue we encountered with this project.