Capstone README

Srikanth Schelbert

About my code:

The code, as directed by the milestones, will calculate a trajectory for the end effector based on goal positions using the TrajectoryGenerator function. This employs the use of the CartesianTrajectory and ScrewTrajectory functions from the modern robotics library.

With the trajectory of the end effector, the function FeedbackControl takes the current, desired, and upcoming desired positions of the end effector along with the Ki and Kp gains to calculate the twist at each timestep of the robot using a control loop.

This code is then fed into the NextState function which calculates the next timestep's configuration of the robot. These functions are done in a for loop that is the length of the trajectory minus one. The configuration is updated at the end of the loop to be used in the next iteration as an input.

To accomplish this and lighten the load when writing the final code, I wrote some helper functions that can calculate some important things. Firstly, I use TrajtoMatrix to calculate the end effector position as a T matrix since the TrajectoryGenerator function provides these as a list. Second, I wrote a function called GetCurrentXJacobian which calculates the current configuration as a T matrix (which is used as an input for later functions) as well as calculating the pseudoinverse of the body jacobian which is used in later calculations. Lastly, I wrote a function called GetVelocities that calculates the current velocities of the joints using the pseudoinverse jacobian and the twist of the robot.

All of these functions together allowed me to simulate the robot moving in simulation to pick up a cube and deposit it at the specified location.

How to Run:

All of my code can be found in the "code" directory and running any of the files titled "best", "overshoot", or "newTask" will allow you to run and reproduce any file. The outputs produced should match the files in each of the sub directories titled "best", "overshoot", or "newTask" within the "results directory". Each sub directory contains the trajectory csv produced, the error csv file produced, an image of the error plot, a video of the simulation running in CoppeliaSim, a log file, and a readme with information about the run. Some may have extra things that are clearly named or explained in the readme.

There exists also a directory called “milestones” which contains the individual function scripts that can be run and tested. Each milestone was designed to test a certain portion of the code so that the whole task could be completed seamlessly.

Extra Credit Exploration:

In my code (for all full python scripts), there exists a block of code that is commented out around lines 306-312 within my GetCurrentXJacobian function. This code is intended to explore the ability for the robot to move with joint limits that avoid collisions with itself. The idea involved limiting joints 3 and 4 of the arm in such a way that their angles summed together could not equal pi (meaning that they would have large enough angles to collide with the body of the arm).

This exploration aimed to set the Jacobian value for those joints equal to 0 should this condition be met which would disallow them to move further into each other causing a collision. This however sometimes produced an error where the robot was able to hit unforeseen singularities. I suspect that this is due to the following issue. The robot, based on my code, checks the angles at the current state and becomes unable to move causing a singularity. The solution would be for the robot to check the upcoming state’s angles, and adjust the Jacobian accordingly to that, so that the robot plans ahead rather than acts in a reactionary manner getting stuck in a singularity.

The exploration of this can be found in the newTask directory where I have a video titled “collision_avoidance” and an error plot called “collision_avoidance” which shows the error spike in a couple places where the robot was not allowed to move the joints it wanted. The “X_error_co.csv” file saves this error. This run was successful however showing that the code did avoid collisions, even though it isn’t infallible.