ME 449 Capstone Project

James Oubre

Explanation of the software

The three functions used to create this simulation are `TrajectoryGenerator`, `NextState`, and' `FeedbackControl`. The first step in generating this simulation is to define the homogeneous transformation matrices for the initial and final positions of the cube relative to the world frame (Tsc_in & Tsc_f), the initial position of the end effector relative to the world (Tse_in), and the stand off and grasping positions of the end effector relative to the cube's frame (Tce_so & Tce_g).

Next, `TrajectoryGenerator` can be called. This function takes Tse_in, Tsc_in, Tsc_f, Tce_g, Tce_so, and the number of reference trajectories per time step as arguments. Each of these matrices are used to achieve the different waypoints for the end effector to travel to. It first moves to the stand off position above the cube, down to the grasping position, back to the stand off, to the standoff above the cube's final position, down to the final position, and back to the final stand off. Each of these waypoints can be described in the world frame by using the function arguments and matrix multiplication, following subscript cancellation. Once a transformation matrix from the world to the waypoint is obtained, the matrix describing the current position of the end effector and the waypoint can be given to the Modern Robotics function `ScrewTrajectory` to generate a trajectory between points. This trajectory is then looped through and transformed so that the format of each step in the trajectory is `r11, r12, r13, r21, r22, r23, r31, r32, r33, px, py, pz, gripper state`. This process is repeated for the path between each waypoint and the trajectories are all appended to a list to return a matrix of desired end effector positions throughout the simulation.

After obtaining the desired trajectory, in the main body of the code, the list of trajectories are looped through and the current actual end effector configuration, desired end effector configuration, and desired next end effector configuration are put in the matrices X, Xd, and Xd_next, respectively. They are then passed to the `FeedbackControl` function alongside the controller gains and chosen time step.

In `FeedbackControl` the error between the actual and desired end effector position is calculated and used to calculate the command end effector twist with PID. It then returns this twist and the error.

Now, back in the main loop, the Jacobian of the system is calculated and multiplied by the returned command twist to obtain the velocity commands of the wheels and arm joints. This, along with the current configuration of the robot and time step are passed into `NextState`.

In `NextState`, the current configuration of the robot is broken up into the separate components representing the chassis, arm, and wheel configurations. Each of these configurations is updated by adding the respective command velocities multiplied by the time step. The output is then formatted as `chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5, W1, W2, W3, W4` and returned.

Finally, the current state is updated to be equal to the next state and this entire process is repeated for every time step in the desired trajectory.

Results

The simulation was able to complete the task in all three scenarios (best, overshoot, and next state). There was some wobbling of the arm while it was recovering from error. I believe this may have been caused by not implementing joints limits. The wobbling and error also became much more apparent when there is more overshoot. Overall, however, each simulation is able to reduce to zero error before the robot gets the the cube's standoff position.