Final Project

MAE 204 : Robotics
Prof. Michael Tolley

Thursday March 20, 2025

Janna Aboudaher

Christian Parra

Report Summary

We started this coding project by developing the kinematics simulator function ‘NextState’. The purpose of this function is to use the Euler method, starting with the known configuration of the chassis arm and wheels and the command velocities at that instant to find the resulting configuration after a small time step. The traditional Euler method is used to find the resulting wheel and arm configuration simply using the command velocities. The chassis configuration can then be determined using odometry for the four-mecanum-wheel robot. The odometry calculation uses two cases to determine the change in chassis coordinates depending on whether the magnitude of the angular velocity of the chassis in the z direction is zero or not. In this function, instead of using zero, the restriction was set at an angular velocity of .001 rad. This is meant to account for the possibility of MATLAB not recognizing extremely small velocities that are effectively zero, for example if a number is to the order of 10^{-24} as zero. Another enhancement that this function implements is limiting velocity magnitudes at the beginning of the function ensuring that the change in configuration is reasonable. We also attempted to impose arm joint limits which were checked just after each euler step to ensure that the robot arm would not collapse on itself. In the end we deactivated these because the joint limits we established prevented the robot from reaching the cube.

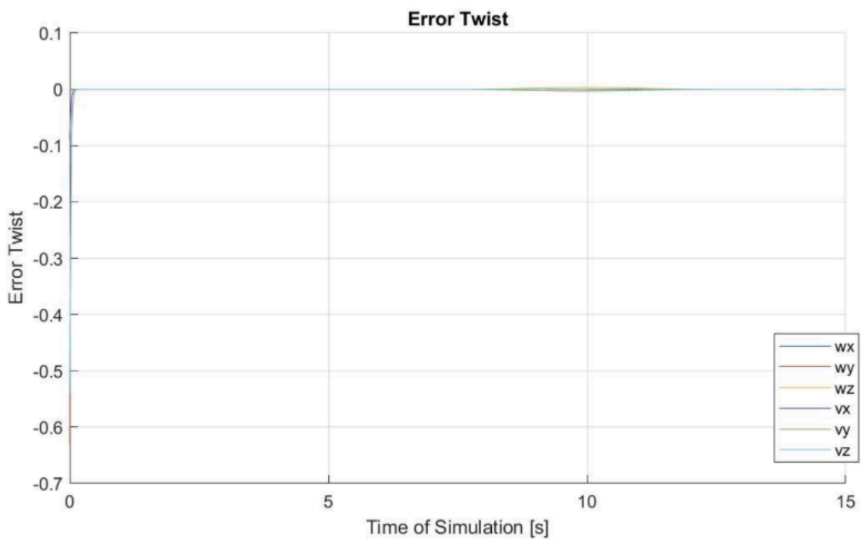
The next function that we built was the end effector trajectory generator ‘TrajectoryGenerator’. This function creates a sequence of desired end effector configurations, T_{se}, that will be used as a reference to guide the calculation of robot configurations through the necessary trajectories to accomplish the task. It includes generating trajectories from starting position to the standoff position to picking up the cube until finishing the task. We first defined a simulation length of 15 seconds for the entire set of tasks. Then each individual trajectory was categorized based on how long the change in configuration was estimated to take. Trajectory 1 and 5 were categorized as long movements as they required the most movement of the robot arm. Trajectories 2, 4, 6, and 8 were then categorized as short movements. The ScrewTrajectoy function from the MR textbook was then used to generate each section of the trajectory from the starting and intermediate end effector configurations. These trajectories were then converted into a set of rows with the corresponding gripper configurations and spliced together to create one matrix that sequentially described the entire robot motion in the form of End effector configurations, T_{se}.

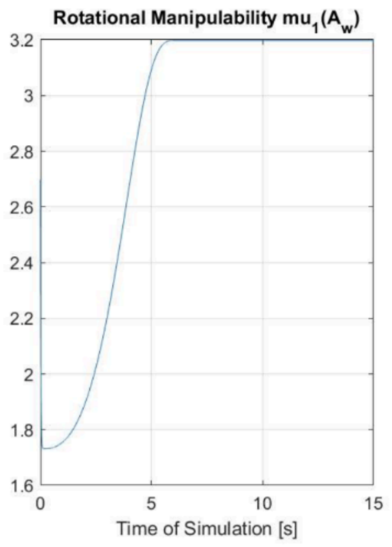
The feedforward plus feedback controller, ‘FeedbackControl’, was the last function we designed that uses the control law appearing in the MR textbook in equation 13.37 to determine the desired command joint velocities at each configuration to reach the next desired configuration. One enhancement that we made to this function involved incorporating a tolerance of .0001 to the Pseudoinverse of the wheel and joint jacobian. This recognized any number smaller than the tolerance as zero to prevent singularities.

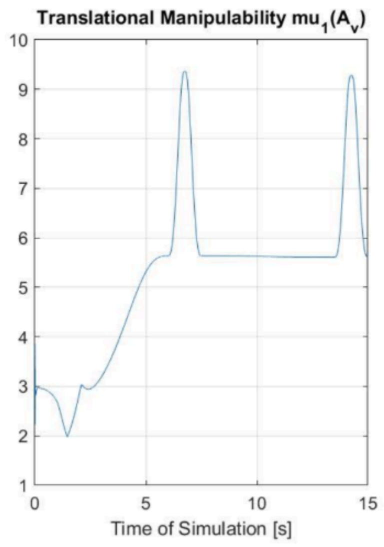
Lastly, the Wrapper function incorporates the 3 previous functions to create a reference trajectory made up of chassis, arm, and wheel configurations that the simulated robot can follow. First, all of the end effector configuration waypoints are defined and used by TrajectoryGenerator to create the reference trajectory. The desired gains for the proportional and integral controller are then defined along with the actual initial configuration of the robot. Then a loop is defined, starting by designating the desired and next desired end effector configurations for each step, which are then fed into FeedbackControl. Then all of the robot configuration angles and positions are updated using NextState and the actual end effector configuration T_{se} for each step recorded before repeating the loop. During this loop the velocity error, integral errors and positions are stored for each iteration to later be graphed.

Results

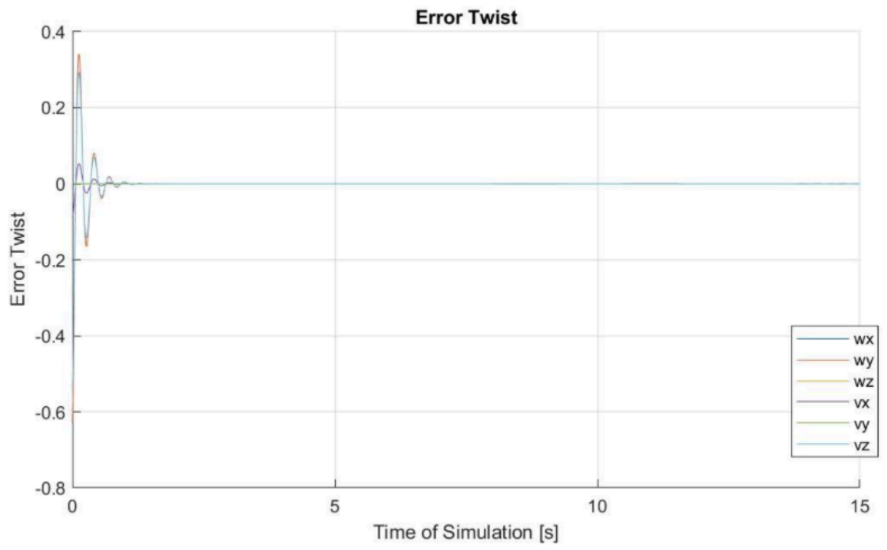
The “best” case involved FeedForward + P control in which kp = 5. The plot of the six error twist demonstrates that the error quickly is driven to zero with no overshoot. Both manipulability plots exhibit no singularities along the trajectory, with small spikes relating to the extensional movements of the robot arm. These movements are not singularities, however the robot is approaching a position in which it loses the ability to extend any further, causing a spike in the manipulability plot. The small variation in the error twist is attributed to some error once the robot begins turning to place the block, but is quickly reduced back to zero again. Adding an integral controller may reduce this error, however this would create overshoot as will be discussed in the next case. Tuning this controller consisted of changing the gains such that the time of correction needed to go between the input initial configuration and the desired initial condition was minimized with no overshoot. best_vid

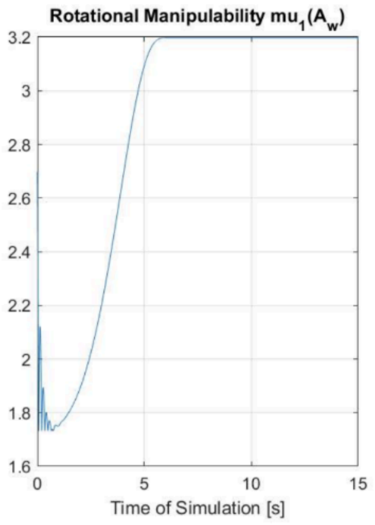


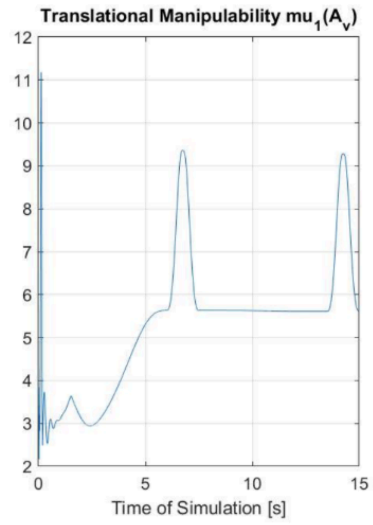




The “overshoot” case used FeedForward + PI control in which kp = 1 and ki = 5. In this case, there was the elimination of the small error seen around 7s of the “best” case, but very noticeable oscillations are present when the robot is correcting the actual initial location to the desired initial location. The same two peaks in the translational manipulability plot are present, again representing the extension of the robot arm towards a singularity orientation, but there is now another peak at the very beginning of the simulation. This peak, and the oscillations present just after it, represents the oscillating correction of the robot arm position as it corrects from the offset initial position to the desired one. There are also oscillations present in the rotational manipulability plot at the beginning, representing the oscillatory correction of the orientation of the end robot. While the overall error for this case avoids the small error in the middle of the simulation, the large overshoot at the beginning requires more of a correction than the “best” case simulation. overshoot_vid



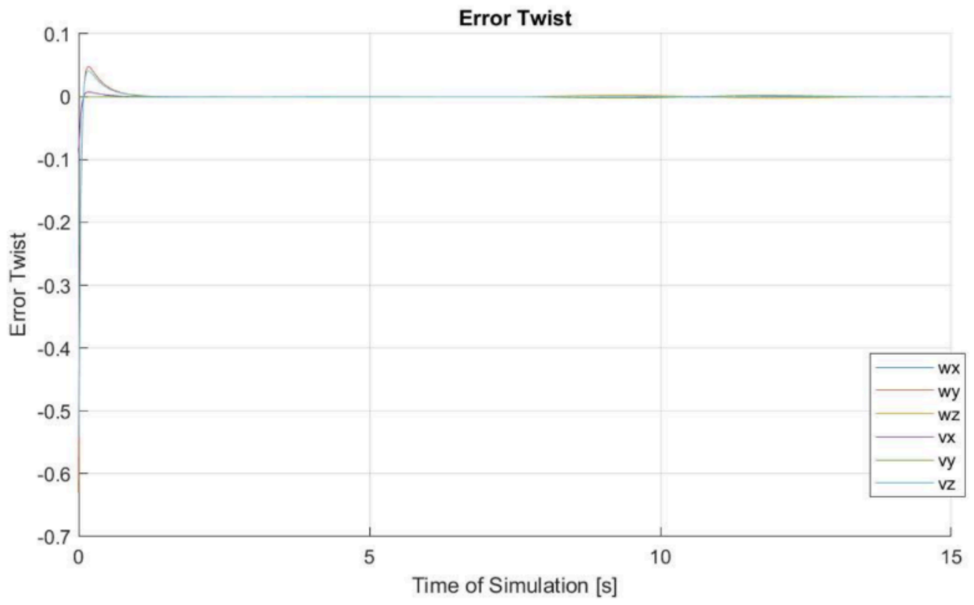


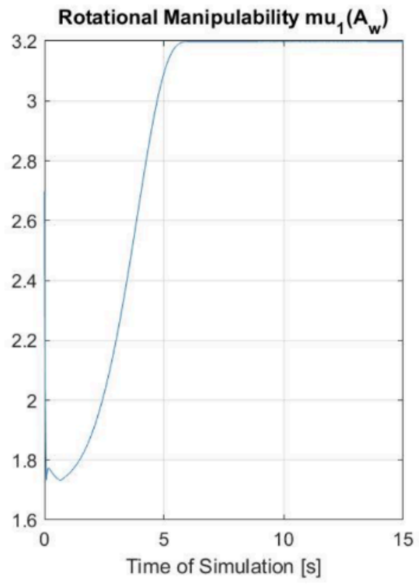


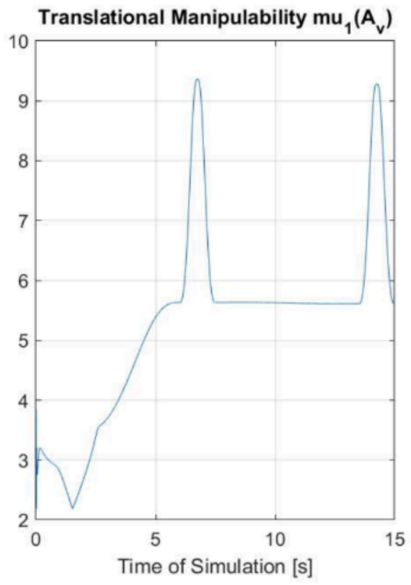
The "New Configuration" case saw the block moved to the following locations:

$$r_{ci} = (0.75, 0.25, 0)\ \&\ r_{cf} = (0, -2, 0)$$

Using FeedForward + PI control with controllers of kp = 3 and Ki = 1, the robot successfully corrected to the desired trajectory and moved the block from its new initial state to its new final state. There was minimal overshoot, and in turn, minor oscillations in the transient stage of the robot's motion. The same small error occurs in the middle of the trajectory, much like in the "best" case, but it is again corrected back to an error of zero. Tuning this controller was relatively easy and followed the same process as the "best" case. Minimizing oscillations, overshoot, and correction time (rise time) were the goal, and these controllers provide an effective way of completing the task. newConfig_vid







Discussion

1. Increasing the integrator term does a good job of reducing steady state error quickly, however can introduce oscillations. The integral controller sums all of the past errors to correct the output to be closer to the steady state error. If the initial error is large, that large stored error will introduce overshoot in the error plot as the oscillation decays towards the steady state value. While this does mean there should be much less or zero steady state error, these oscillations produce odd physical movements of the robot while it is correcting to the desired trajectory. It takes time, and joint motion to correct for these oscillations, which could potentially strain the robot or the task being performed.

2. The manipulability factors became larger as the robot extended. The fully extended position is a singularity, and as the robot's joint extended to their maximum values they approached this singularity position. As the arm extended, it approached configurations in which it could not extend any further, as seen in the initial extension of the arm in all videos. The configuration the robot is in is not a singularity, as seen in the lack of a spike in either manipulability plot, because there was always some joint not fully extended, allowing the robot to perform the task without losing the ability to move in any needed direction.

3. If maximum joint velocities are set at low values, then the error will increase again after picking up a cube. This happens because when the gripper closes to picking up the cube, there is no required arm movement. Once the arm begins to move again, the reference end effector trajectories will require a certain joint velocity in order for the arm to closely follow the desired path. When the joint velocities are set too low, the robot arm is physically not able to move fast enough in order to closely track the desired path. This therefore causes the error to increase.

4. The types of tasks where a UR5 robot would use velocity control is when it needs to precisely move to a specific point or if it needs to move in a smooth manner. One example of this is if the robot were used for soldering small electrical components. Here the position of the end effector requires high precision. The types of tasks where a UR5 robot would use Torque control is in tasks that manipulate the environment and require the robot to be gentle with said environment to reduce damage. One example of this is if the robot had to tighten screws. These tasks require the screw to not be over or under tightened to work properly therefore torque control is best used to ensure that they are not damaged and placed properly.

5. Some additional parameters you would need to know in order to compute required torque are: the gravity vector, the list of link frames relative to the home position, spatial inertia matrices Gi of the links, and the reference joint variables, velocities, and accelerations in addition to the actual joint velocities and variables. I would use the function ComputedTorque to find the required torques.