MAE 204 Final Project youBot

Winston Chou, Alexander Ivanov
Professor Michael Tolley
MAE 204 - Robotics
20 March 2025

Team

- Alexander Ivanov (a1ivanov@ucsd.edu)
- Winston Chou (w3chou@ucsd.edu)

Summary

The development of the code for the final project followed an incremental approach, aligning with the instructions provided. The process began with the implementation of the trajectory generation function, followed by the kinematics simulator, then the feedforward and feedback control functions, and finally a wrapper script to integrate all these components. Each of the sub-functions was tested individually to ensure that they produced the proper outputs and to alleviate debugging challenges in the future.

Trajectory Generation (`TrajectoryGenerator.m`)

The trajectory generation function was developed using the suggested inputs and outputs, adhering to the approach outlined in the problem statement. The standoff configuration for the end-effector was set at 20 centimeters above the cube's center before and after grasping. The trajectory generation uses the "`ScrewTrajectory`" function with a quintic polynomial for time scaling, resulting in a screw motion rather than a straight line in Cartesian space. The trajectory was then verified using CoppeliaSim to make sure that the configurations that were chosen allowed the robot arm to properly pick up and move the cube using the end-effector. No enhancements were implemented over the basic project description that was given for this section.

Kinematics Simulator (`NextState.m`)

The kinematics simulator was implemented as instructed, the inputs being: the current state of the robot, joint and wheel velocities, timestep size, and maximum joint and wheel velocity magnitude. These inputs were used in the function to compute the next state of the robot using a first-order forward Euler integration step. First, the joint and wheel velocities are checked to make sure that they sit within the bounds of the maximum joint and wheel velocities. In this case, the wheels and joints have the same maximum velocity because we are assuming that the joints and wheels are using the same motors. This acts as a saturation limit for all motors, which limits how much the motors can change their joint angles and wheel angles. Following velocity updates, the new chassis configuration was determined from odometry, and joint limits were checked to prevent singularities.

Feedback Controller (`FeedbackControl.m`)

The feedback control function followed the recommended steps, inputs, and outputs but included two additional inputs. The extra inputs are the current state vector, q, and a feedforward-enable flag. The current state vector was used to compute the transformation from the body frame to the space frame, as well as the transformation from the end-effector frame to the arm base frame. Once these two transformations were computed, the procedure was followed as outlined in the instructions. Additionally, checks were put into place for the joint limits and the singularities in the configuration. The joint limits are checked within the function to determine which entries in the Jacobian matrix are close to the singularity, and the Jacobian of those joints is ignored. In order to avoid singularities, a tolerance is placed on the "`pinv`" function in MATLAB in order to prevent it from producing unacceptably large values.

Wrapper (`main.m`)
Lastly, once all of the components were tested and verified that they performed accordingly, a wrapper script was created to call the functions in the proper order. At the beginning of the script, it makes sure that the integral error is reset to 0 and allows the user the opportunity to run whichever scenario they choose. Afterwards a trajectory is generated and the feedback control and "`NextState()`" functions are called in a loop. The outputs are plotted, then saved.

Results

Best Case

For the "best" case scenario, we use a feedforward-plus-P controller to remove the error in the system. The control system used for all three cases that will be discussed in this report uses logic to mitigate effects from singularities that could occur in the system. It does this by making use of the MATLAB "`pinv`" function to prevent the pseudoinverse from producing unacceptably large values. Furthermore, there are joint limits implemented in the control algorithm to prevent the jacobian matrix of the arm from being populated with values despite having reached their joint limits. A proportional gain K_p is set equal to a six-by-six diagonal matrix with 3's along the diagonal, while the integral gain is a six-by-six zero matrix, which means that with no integral gain there is no accumulated error over time. Despite the proportional controller only responding to instantaneous error, the feedforward component of the controller uses a feedforward term to provide correct baseline control action. In Figure 1, we can see that all 6 errors of the twist errors converge to zero using a feedforward-plus-P controller. In this figure, it can be seen that the transient response takes about two seconds to settle before the error goes to 0. Furthermore, there is no overshoot in this case because a feedforward-plus-P controller results in a first-order error dynamic equation. As we can see from Figure 1, all of the responses from the error terms match a first-order error response where there is no steady-state error and no overshoot. If we were to increase the K_p gain, that would result in a faster response, but can make the robot overreact to small errors, which can lead to jerky behavior. The advantage of feedforward control is that it provides a desired input; this is helpful because it provides an early corrective action, which leads to little overshoot and oscillations. These characteristics are seen in Figure 1. The drawbacks of solely using feedforward control are that the control system is dependent on the model, and any unmodeled disturbances that enter the system will not be accounted for because there is no feedback component. In this case, a feedforward-plus-P controller is implemented, and the feedback term accounts for any deviations in our response from the actual trajectory. Additionally, feedforward control requires a good initial condition in order to follow the correct trajectory, since it does not have a feedback term to remove the error in the system. The 5th-order polynomial that we use to develop the reference trajectory, ensures that there are no instantaneous accelerations throughout the motion.

In Figure 2 below, we can see the manipulability factors for the robot while running through the "best" case scenario. One thing that is noticeable is that the angular $\mu_1$is much smaller than the linear velocity's μ_1 the entire time. This denotes that the robot arm is able to easily rotate in any direction. Meanwhile, the μ_1 for the linear velocities are higher in number, which means that small changes in the linear velocity could cause large end-effector movements. This means that the robot is more sensitive when it comes to changing the linear velocity.

A video showing the results of this algorithm can be found at this link:
https://www.youtube.com/playlist?list=PLbabeYyHVxi0cFRawUQSwGy9YwoZjvt3B

Under the video called: "Task Scenario: Best"

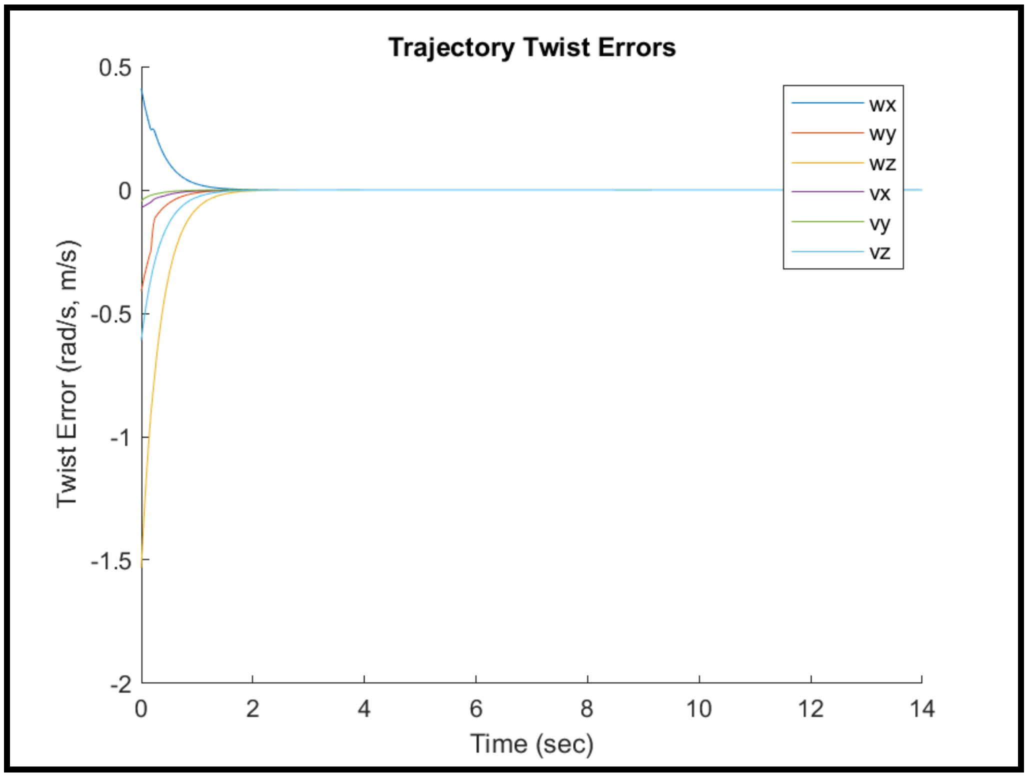

Figure 1: A plot of the six elements of X_{err} as a function of time for the “best” case.

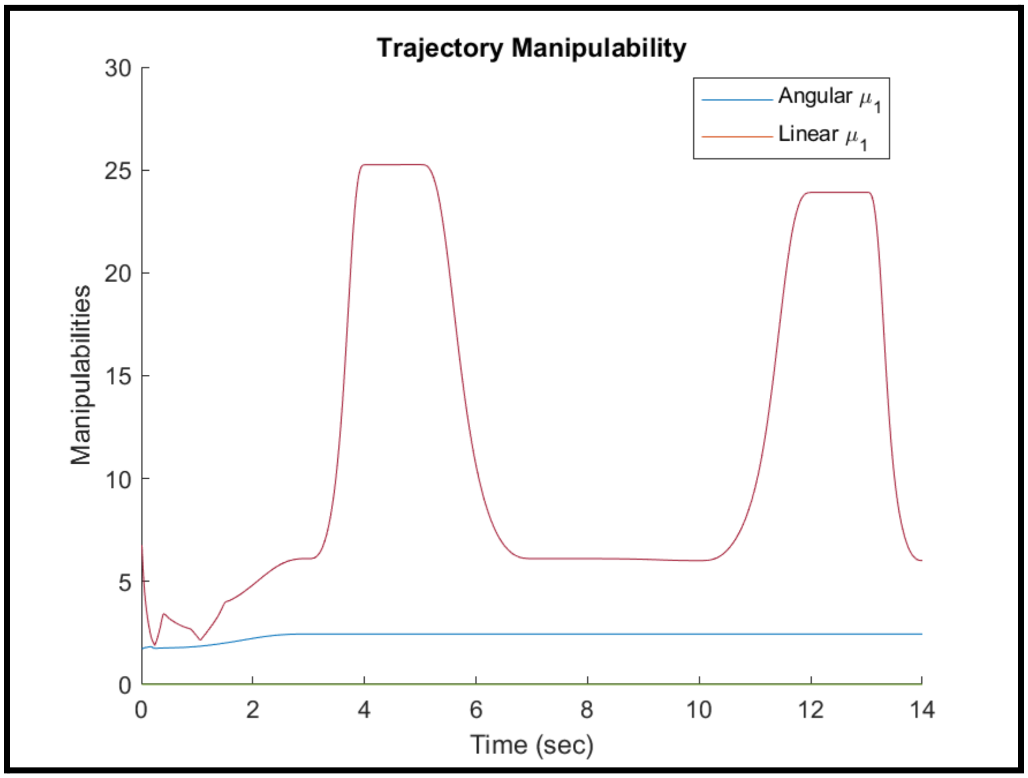

Figure 2: A plot of the two manipulability factors (i.e., $\mu_1(A_w)$ and $\mu_1(A_v)$) as a function of time for the “best” case.

Overshoot Case

For the “overshoot” case, a PI controller was used to remove the error in the system. Similar to the “best” case scenario, this controller also checks for joint limits and makes sure that in the case of singularities, the “pinv” function in MATLAB is prevented from producing unreasonably large values for pseudo-inverse. The proportional gain that was used in this case is a six-by-six matrix with 5’s across the diagonal, and the integral gain is a six-by-six matrix with 0.5 across the diagonal. The combination of the proportional and integral gains chosen results in a control response with overshoot for each of the error terms. It is important to balance the integral gain so that there is not too much overshoot, which is an undesirable trait to have in a robot. One thing when comparing Figure 3 to Figure 1 is that with the integral gain, the initial error converges more quickly to 0 than the proportional gain case. Another point of interest is that with the small integral term chosen, the controller accumulates error slowly, which means that it converges on the steady state error more slowly as well. The overshoot in the plots can be reduced with smaller proportional gain values as well, but it is important to keep the system responsive to new errors that appear in the system. Figure 4 shows the manipulability factors for the angular and linear velocities for the PI controller. In this case, the factors look similar to the feedforward-plus-P case where the angular manipulability is relatively small, which denotes that the manipulability ellipsoid is close to the shape of a sphere and allows for control of the end-effector in any direction. The velocities, on the other hand, show that they are approaching a singularity because their manipulability factor rises drastically, which means that small changes in velocity result in large changes for the end-effector. Overall, the PI controller that is implemented in the overshoot case provides the advantage that it is able to adjust to errors and disturbances that appear in the system. This also makes it less susceptible to poor initial conditions, since it can adjust for the errors over time. However, introducing the integral term also makes the controller more difficult to tune properly. It is also important to keep in mind that without a feedforward component or a derivative component in the controller, a PI controller can be slow to react to fast dynamics and therefore take a while for it to react to the error in the system.

A video showing the results for the PI controller can be seen here:
https://www.youtube.com/watch?v=vfzlKWDpwc8&list=PLbabeYyHVxi0cFRawUQSwGy9YwoZjvt3B&index=3

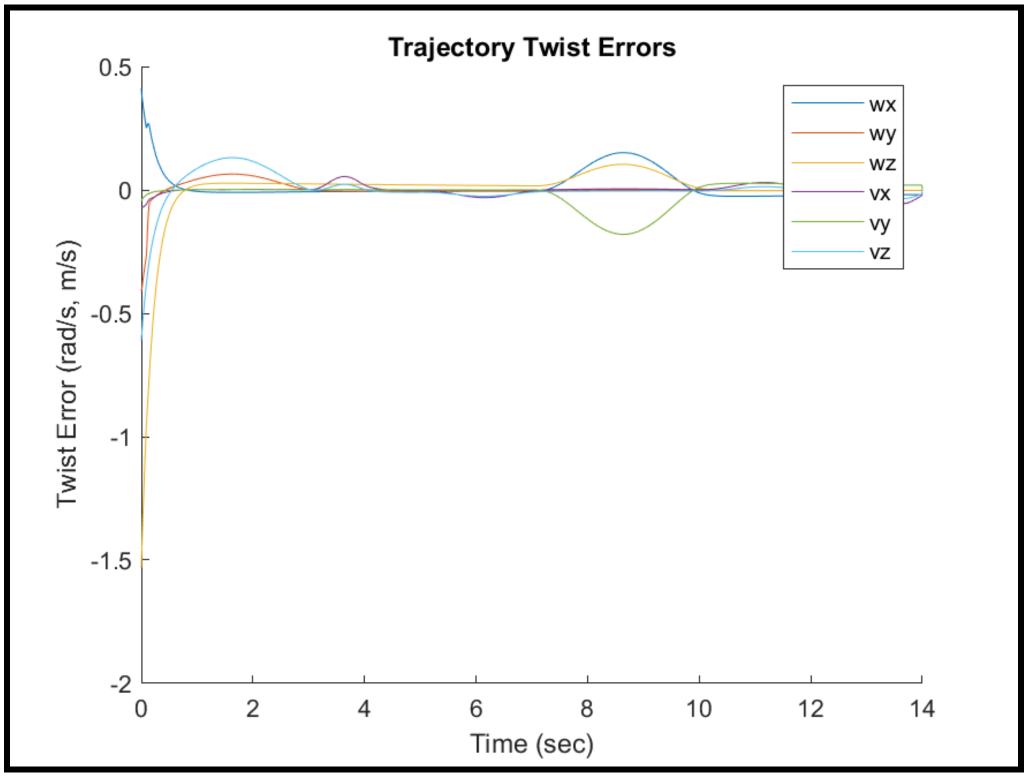

Figure 3: A plot of the six elements of X_{err} as a function of time for the "overshoot" case.

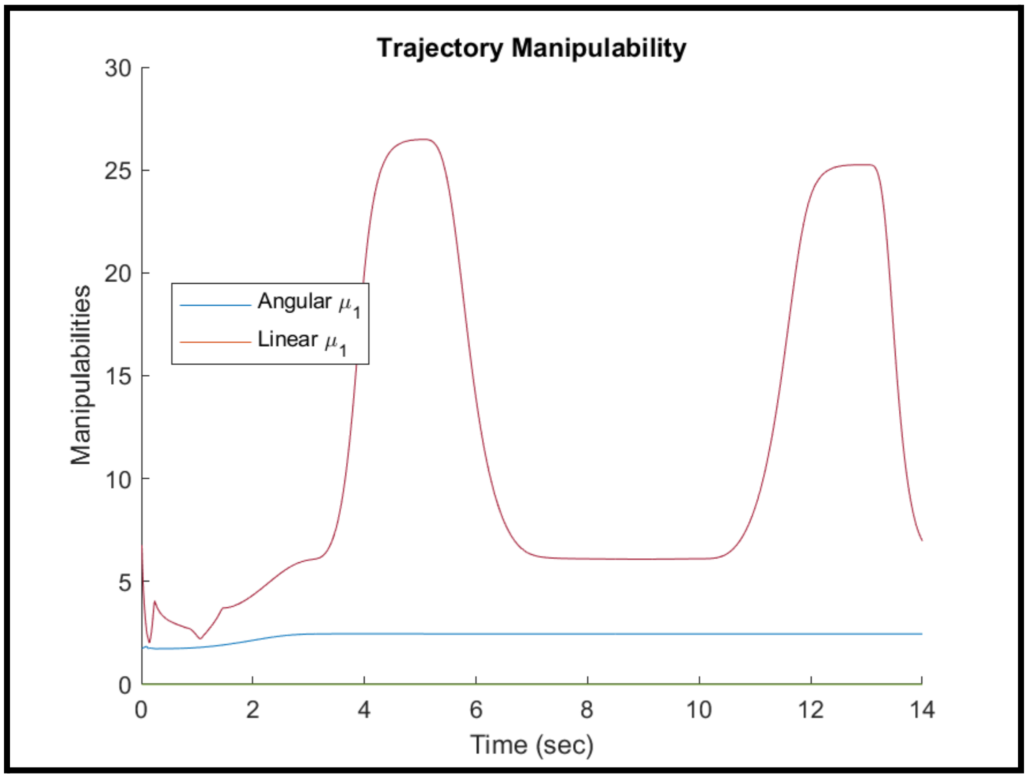

Figure 4: A plot of the two manipulability factors (i.e., $\mu_1(A_w)$ and $\mu_1(A_v)$) as a function of time for the "overshoot" case.

New Task Case

Lastly, we will take a look at the case labelled “new task” with a video of the results shown here: https://www.youtube.com/watch?v=Mrw1Q1xuohg&list=PLbabeYyHVxi0cFRawUQSwGy9YwoZjvt3B&index=4

Figures 5 and 6 below show that the controller used for the “new task” case is a feedforward-plus-P controller. In this case, the proportional gain matrix is a six-by-six identity matrix, and there is no integral gain. This is supported by Figure 5, where the twist errors approach 0 with exhibiting a first-order error response. Unlike the “best” case scenario that we saw before, in this case the block is placed at a different initial location than the first case. In this scenario, the robot has to use its wheels initially to pick up the block and transport it to a different location. Once again, the trajectory that was generated for this case also used the 5th-order polynomial to ensure that there are no instantaneous accelerations throughout the motion, which prevents jerky motion in the robot.

The initial and final configurations of the cube are defined as:

```
cube_initial = [1, 1, 0];
cube_final = [1, -1, -pi/3];
T_sc_initial
 = [[cos(cube_initial(3)), -sin(cube_initial(3)), 0,  cube_initial(1)];
    [sin(cube_initial(3)),  cos(cube_initial(3)), 0,  cube_initial(2)];
    [                   0,                     0, 1,            0.025];
    [                   0,                     0, 0,                1]];
T_sc_final
 = [[cos(cube_final(3)), -sin(cube_final(3)), 0,  cube_final(1)];
    [sin(cube_final(3)),  cos(cube_final(3)), 0,  cube_final(2)];
    [                 0,                   0, 1,          0.025];
    [                 0,                   0, 0,              1]];
```

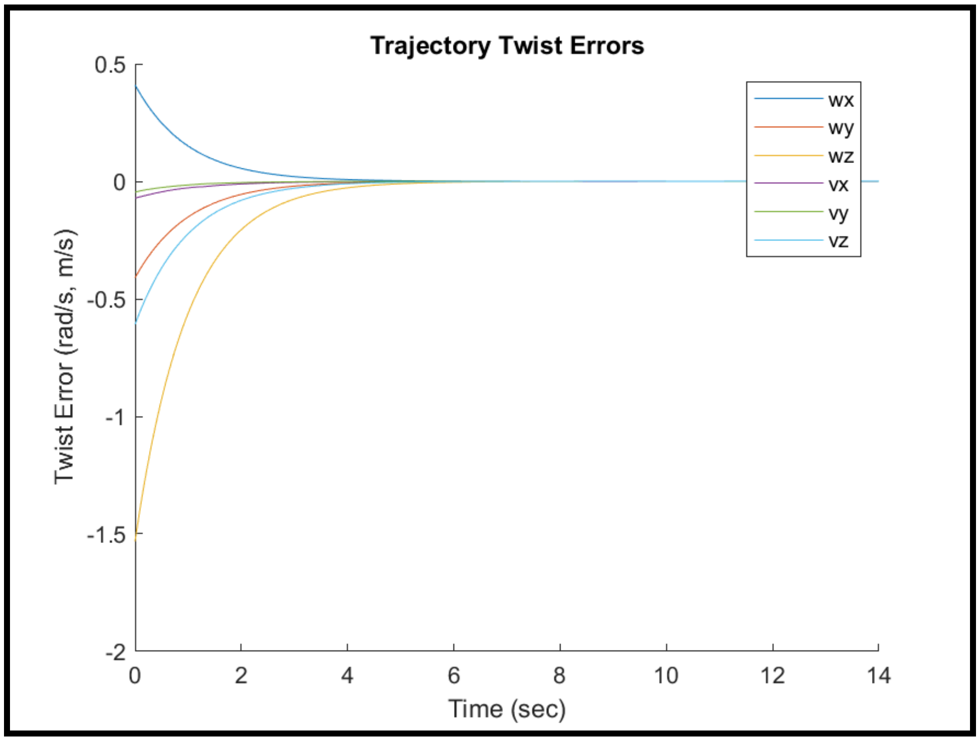

Figure 5: A plot of the six elements of X_{err} as a function of time for the “new task” case.

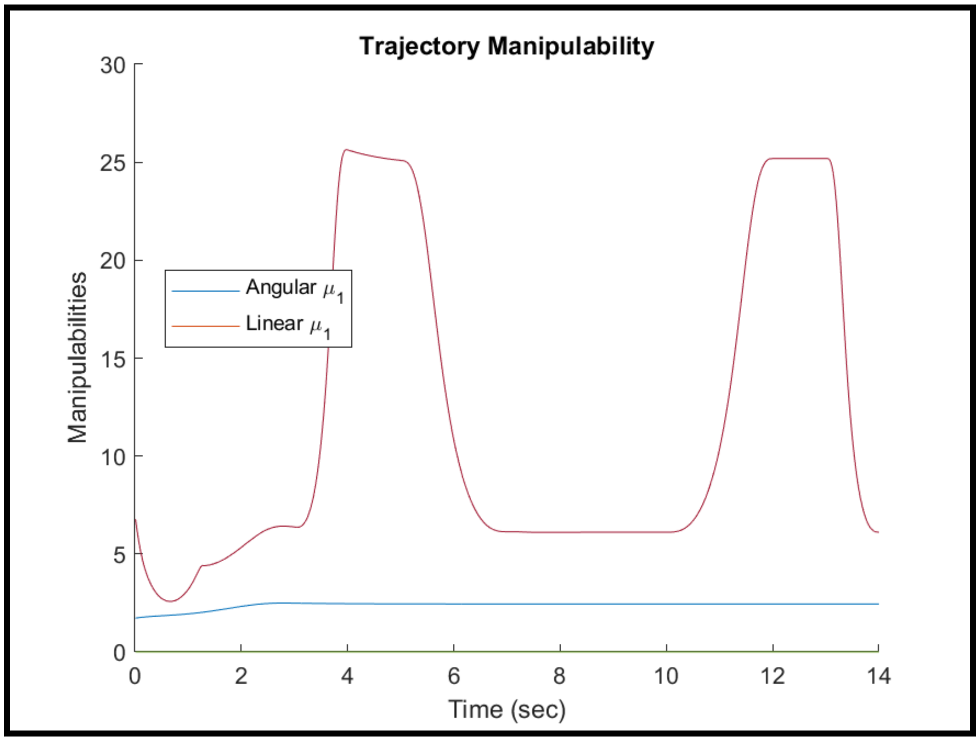

Figure 6: A plot of the two manipulability factors (i.e., $\mu_1(A_w)$ and $\mu_1(A_v)$) as a function of time for the “new task” case.

Discussion

1. One of the main advantages of increasing the integrator term in a controller is being able to eliminate the steady-state error much quicker. This is because with a higher integrator term, the error accumulates more quickly and the controller is able to drive the offset or steady-state error to 0 in a shorter amount of time. A higher integrator term comes with the trade-off that the system becomes second-order which allows for the presence of overshoot and oscillations in the control response. Due to a high integrator term, if the initial error term is large, the errors accumulate quickly and the controller “overreacts” in its corrective action. If there is not sufficient damping in the system, then this can lead to oscillations or even instability in the system. In the “best” case, we see that with a first-order error response, it takes longer for the error to reach 0, and in the “overshoot” all of the error terms overshoot their target of 0 error while slowly driving the steady state error down to 0.

2. The linear manipulability factors for the velocity became large when the robot picked up the cube and released the cube at the final configuration. The spike can be attributed to a favorable geometric configuration of the robot’s joints when it is about to pick up the cube, which means that small joint angles produce large linear velocities. This occurs because the robot arm is nearly extended in a straight line, which is a singularity in the robot arm configuration since it loses its ability to move in a direction. This explains the increase in the manipulability for the linear velocity.

3. If you reduce the maximum joint velocities to low values, this causes the error to increase because the robot arm cannot follow the desired trajectories as quickly. This is particularly relevant after picking up the cube because the robot’s moment of inertia changes, which in turn could require larger joint velocities to follow the desired trajectory.

4. We think velocity control is a realistic solution for the application that we saw in this project, which required picking up and placing an object somewhere. Applications where velocity control would not be beneficial would be applications where torque control or force control is required to achieve a goal. If the robot was required to assemble delicate parts, then force control would be necessary in order to prevent damaging any of the parts.

5. The torque control would need the following inputs: reference position, reference velocity, reference acceleration, the controller gains (Kp, Ki, Kd) depending on whether a P, PI, PD, or PID controller is used, the current acceleration, current

velocity, and position. We also need to make use of a dynamics model for the forward and inverse dynamics. We would use the function 11.35 from the MR textbook.

6. Generative AI, Perplexity, was used only for checking proper usage for certain MATLAB functions, such as plotting and UI-related functions.