FINAL PROJECT REPORT

Team Members:
1. Mehul Kumar (m9kumar@ucsd.edu)
2. Tanya Arora (taarora@ucsd.edu)

REPORT SUMMARY

The project involves implementing a control framework for a mobile manipulator to perform a pick-and-place task. The approach consists of three main components: trajectory generation, kinematics simulation, and feedback control. Each component plays a crucial role in ensuring accurate and stable execution of the task, with a focus on smooth motion and precise trajectory tracking.

The trajectory generation is handled by the TrajectoryGenerator function, which constructs an eight-segment reference trajectory for the end-effector. The motion is interpolated using either Cartesian or Screw-based interpolation, with a quintic time-scaling method applied to ensure smooth transitions between segments. The trajectory defines key configurations such as the initial and final positions of the cube, intermediate standoff positions, and gripper actions (open or closed). By specifying the number of reference configurations per timestep, the function ensures a well-defined trajectory that the controller can follow precisely.

The kinematics simulation is implemented through the NextState function, which updates the robot's configuration at each timestep. It follows a first-order Euler integration approach to propagate the robot's state based on given control inputs. The function accounts for velocity constraints to prevent excessive actuation, ensuring that both joint and wheel velocities remain within predefined limits. The chassis position is updated using a kinematic model derived from the wheel velocities, while the arm and wheel states are adjusted based on their respective velocities. This function serves as the foundation for predicting and simulating the robot's motion under different control commands.

The feedback control mechanism is designed using the FeedbackControl function, which employs a feedforward-plus-PI control strategy. The function computes the error between the current and desired end-effector configurations using which captures both rotational and translational discrepancies. Additionally, the integral of this error is maintained to enhance long-term accuracy. The commanded twist V is determined by combining the feedforward reference twist with proportional and integral corrections, ensuring effective trajectory tracking. The computed twist is then transformed into joint and wheel velocities using the Jacobian pseudoinverse, enabling precise execution of the desired motion.

The main script integrates these components by initializing the cube and robot configurations, setting proportional and integral gains (, and executing the control loop. The system is tested in CoppeliaSim by generating and running CSV files that dictate the robot's motion. By adjusting the control gains and trajectory parameters, different performance scenarios are analyzed, ensuring robust and efficient execution of the pick-and-place task.

In my approach to computing the manipulability factors, I first extract joint angles from the robot's recorded state at each timestep. Using these angles, I compute the full Jacobian matrix Je, which maps joint velocities to end-effector twist. To analyze manipulability, I split J_e into two key components: the rotational Jacobian A_w (first three rows, first five columns) and the

translational Jacobian A_v (last three rows, first five columns). This decomposition helps distinguish the robot's ability to rotate and translate independently. I then compute the smallest singular value of each Jacobian using Singular Value Decomposition (SVD), as the smallest singular value indicates how close the system is to a singularity. The full Jacobian's smallest singular value represents overall manipulability, while the values from A_w and A_v reflect the independent rotational and translational capabilities. Finally, I plot these values over time to observe how the robot's configuration affects its ability to move and rotate effectively.

The Jacobian J_e is a 6 x 9 matrix that maps joint velocities to end-effector twist, where the first three rows represent rotational velocity (ω_x, ω_y, ω_z) and the last three rows represent translational velocity (υ_x, υ_y, υ_z). Since the system has four wheels and five arm joints, the full Jacobian has nine columns. However, to analyze arm-only manipulability, I extract two submatrices: the rotational Jacobian (A_w) (first three rows, first five columns) and the translational Jacobian (A_v) (last three rows, first five columns). This selection isolates how the arm contributes to rotation and translation, excluding the base's effect. The four wheel-related columns are ignored because the wheels influence chassis movement rather than independent end-effector dexterity. Using Singular Value Decomposition (SVD), I compute the smallest singular value of each Jacobian, as this indicates the system's proximity to singularity. A lower value implies reduced dexterity, meaning the robot struggles with motion in certain directions at that configuration. This decomposition helps in understanding whether the arm is in a well-conditioned state for manipulation or nearing singularity, ensuring precise control adjustments.

When testing for the overshoot case, I expected that increasing (K_i) would cause the robot to overshoot during pickup, but instead, it resulted in violent jerks and random movements. I started with ($K_p = 0.5$), ($Ki = 0$), and gradually increased to ($K_p = 2$), ($K_i = 10$), which gave a smooth and stable pick-and-place execution. However, when I pushed (K_i) to 100, the system became completely unstable. This is likely due to integrator windup, where the accumulated error led to excessively large corrective commands, causing erratic motion instead of controlled overshoot.

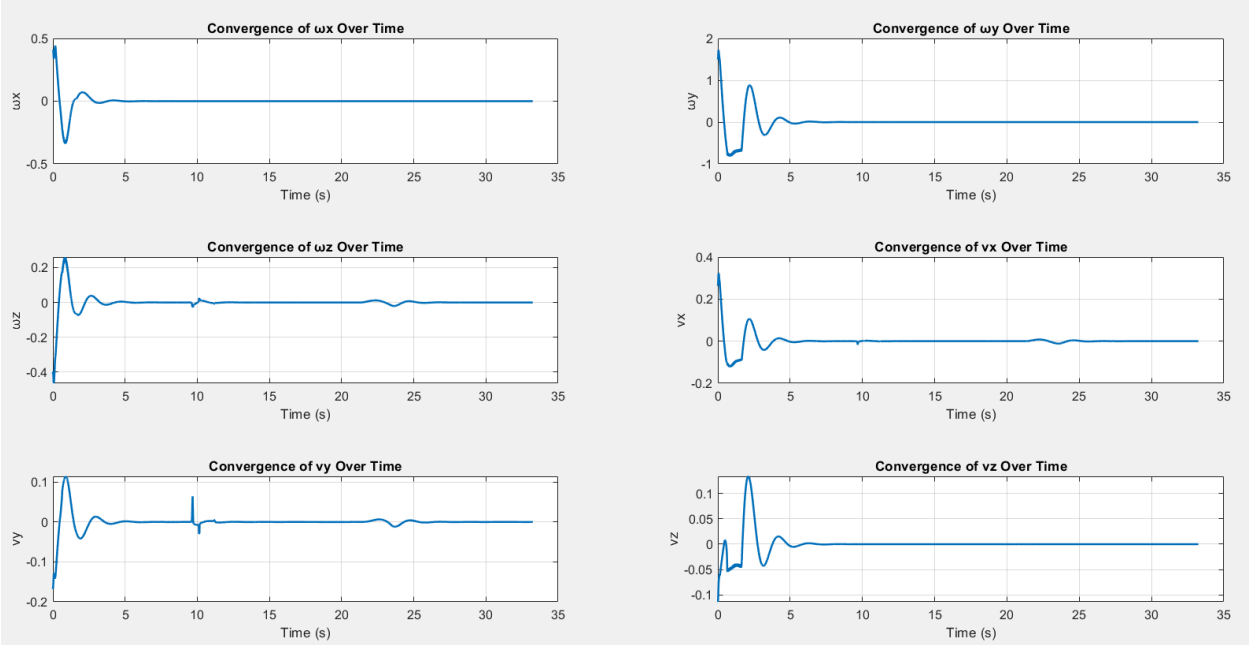
The control law computes the commanded twist V as ($V= Adj(X^{-1}X_d)\, V_d + K_p\, X_{err} + K_i \int X_{err}\, dt$), and with such a high K_i , the integral term dominated early on, leading to large and unrealistic velocity commands. This could have resulted in joint and wheel velocity saturation, causing the robot to behave unpredictably. Additionally, CoppeliaSim does not model perfect contact dynamics, so the robot might have overcompensated near the gripper, leading to instability. To diagnose this, I need to check if the computed joint and wheel velocities exceed limits, incrementally increase K_i (eg, 10 → 20 → 50) instead of jumping to 100, and inspect how X_{err} grows over time. Clamping the integral term could prevent windup, and plotting the commanded twist V should confirm if the control signals are fluctuating too aggressively. Since $K_p = 2$, $K_i = 10$ worked well, the instability at $K_i = 100$ suggests that the system is overcorrecting instead of smoothly following the trajectory. I need to find a moderate K_i value (around 20–30) where overshoot occurs without chaotic movement.

To ensure numerical stability and avoid large, unrealistic joint velocities near singular configurations, we computed the pseudoinverse of the combined Jacobian J_e using a small tolerance (1×10^{-6}) to suppress near-zero singular values. The full Jacobian was constructed by transforming the base Jacobian (from wheel motions) into the end-effector frame using adjoint mappings and concatenating it with the arm Jacobian derived from screw axes. This setup allows the end-effector twist to be mapped accurately to joint and wheel velocities while mitigating risks of singularities and self-collisions.

RESULTS

BEST CASE *Video Link*: https://youtu.be/3pnTQD-HKLs

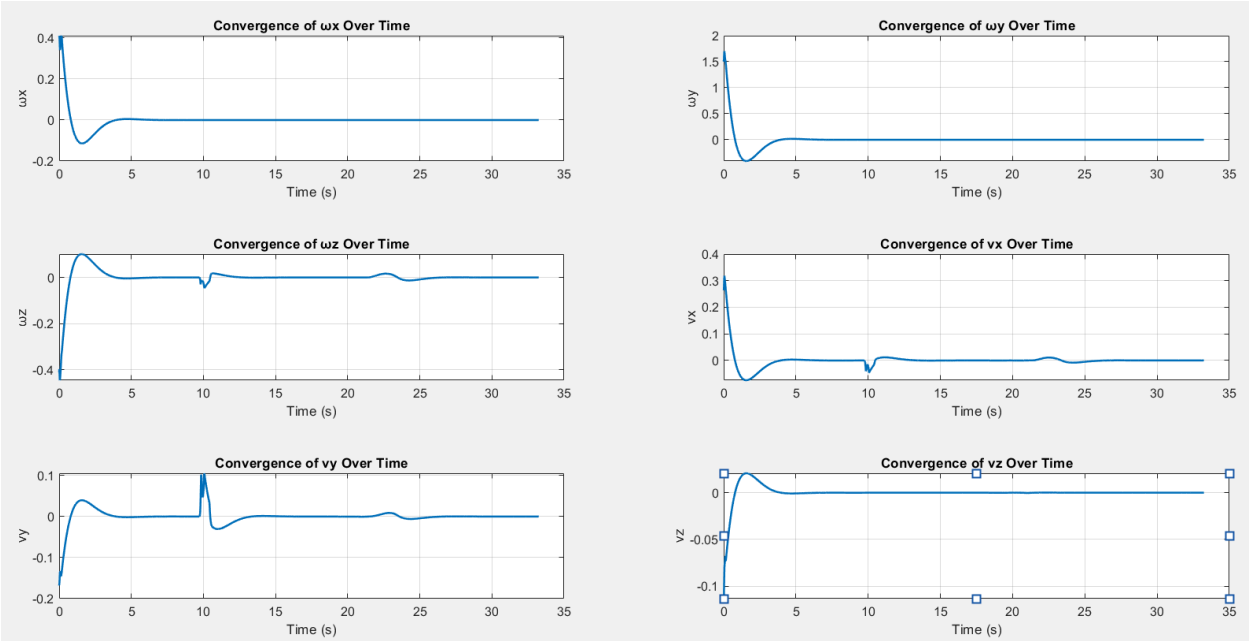
1.



(Kp = 2, Ki = 10)

- Error quickly converges and stays zero with time.

2.



(Kp = 2, Ki = 2)

- Error Converges but deviates seldom with time.

Well-Tuned Controller- for the best-case scenario, I used a well-tuned feedforward-plus-PI controller with K_p=2 and K_i=10. The robot started from a deviated initial position, ensuring it had to correct its trajectory before reaching the target. The feedback controller computed the error between the actual and desired end-effector configurations using the logarithm of the transformation matrix:

$$Xerr = \log(X^{-1}X_d)$$

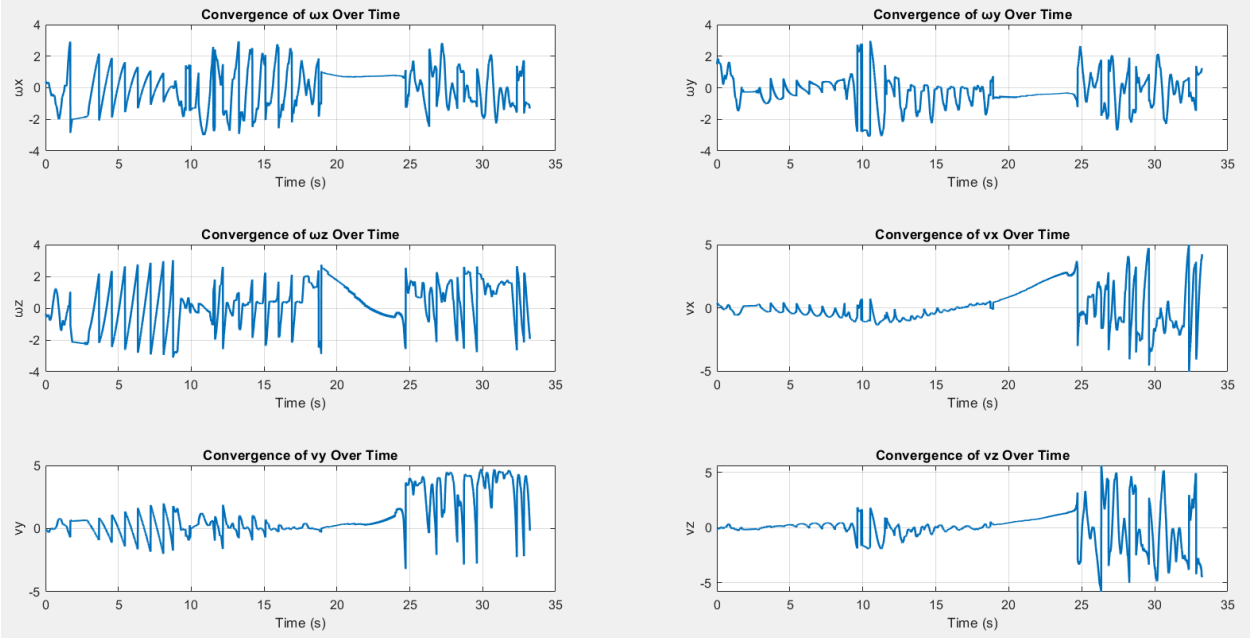
This error was used to compute the commanded twist V, incorporating both feedforward and feedback terms:

$$V = \text{Adj}(X^{-1}\ X_d)\ V_d + K_p X_{err} + K_i \int X_{err}\ dt$$

With this tuning, the error plot showed a rapid convergence to zero, maintaining stability after the object was placed. The motion was smooth, with minimal fluctuations, and the robot successfully executed the pick-and-place task efficiently in CoppeliaSim without any noticeable jerks.

OVERSHOOT CASE

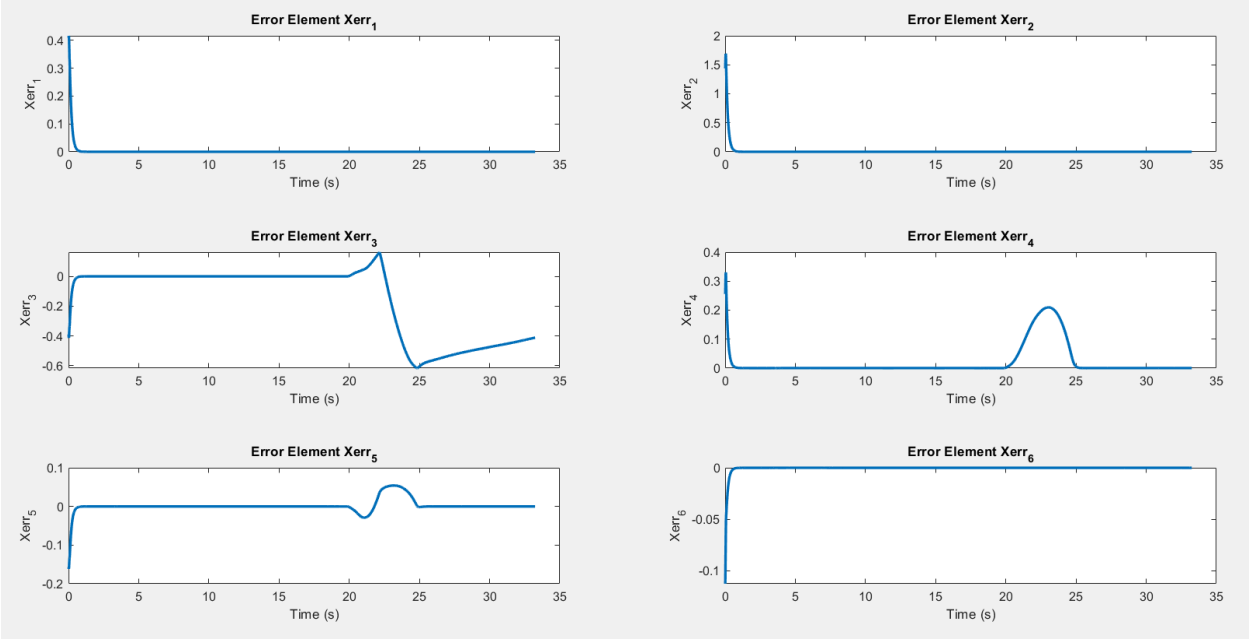
Video Link: https://youtu.be/Ly-pmArrMQ4



Excessive Integral Gain: for the overshoot case, I set K_p = 2, K_i = 100 to observe the effects of an excessively high integral gain. While the controller still attempted to track the trajectory, the error plot exhibited high-frequency oscillations, never fully stabilizing. This was due to the integral term accumulating too much error, leading to overcorrection and instability.

NEW TASK CASE

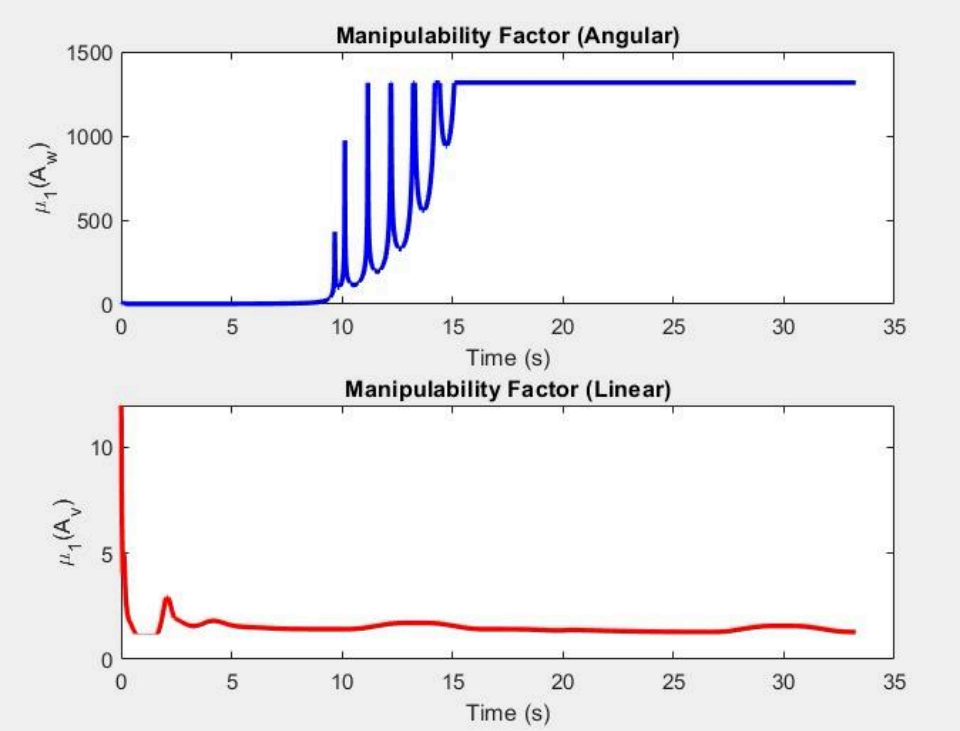
Video Link: https://youtu.be/7uTHa5xaxOg



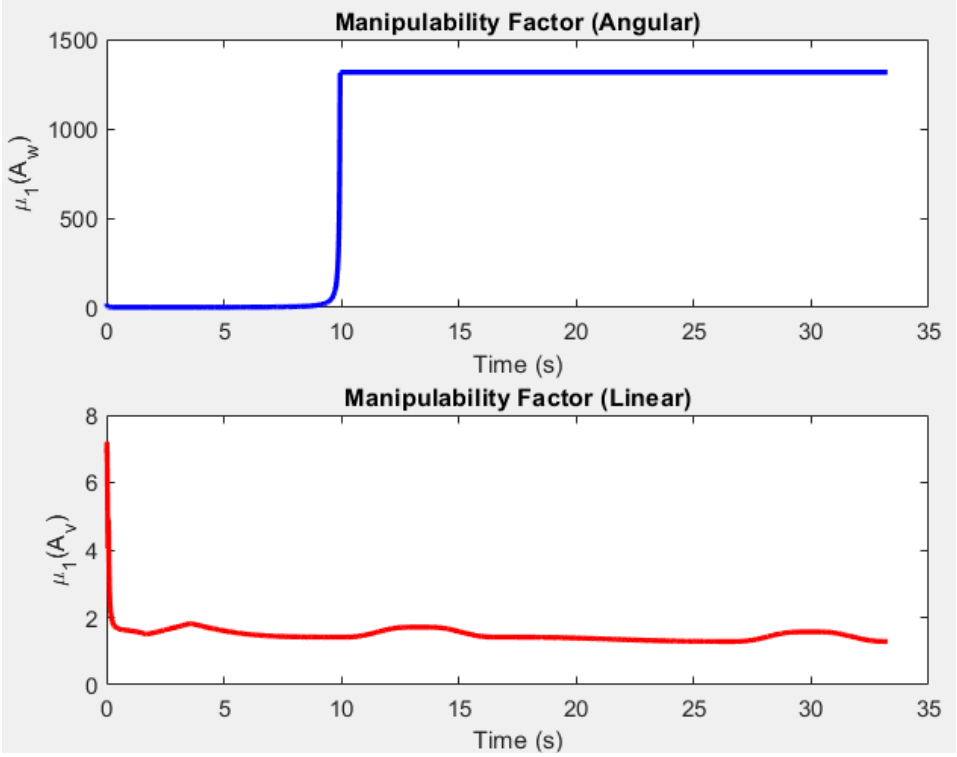
Cube Initial : x = 1, y = 1.2, z = 0.0248
Cube Final : x = 0.5, y = -1.047, z = 0.75
K_p = 7, K_i = 0, Velocity Limit = 9

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

BEST CASE:



NEW POSITION:



DISCUSSION

1. Referring to your error plots for the best and overshoot cases, based on your results, what are the advantages and disadvantages of increasing the integrator term of your controller? Explain why this causes the overshoot that you observe in the Overshoot case.

Increasing the integrator gain Ki improves steady-state error correction, as seen in the best case (Kp=2,Ki=10), where the error converged quickly and remained at zero. However, excessively high Ki (Kp=2,Ki=100) led to high-frequency oscillations and instability, as the accumulated error resulted in overcompensation, leading to jerky, uncontrolled movements in CoppeliaSim.

2. In your results, when did the manipulability factors, $\mu_1(A)$, become small and large? Explain why this happened.

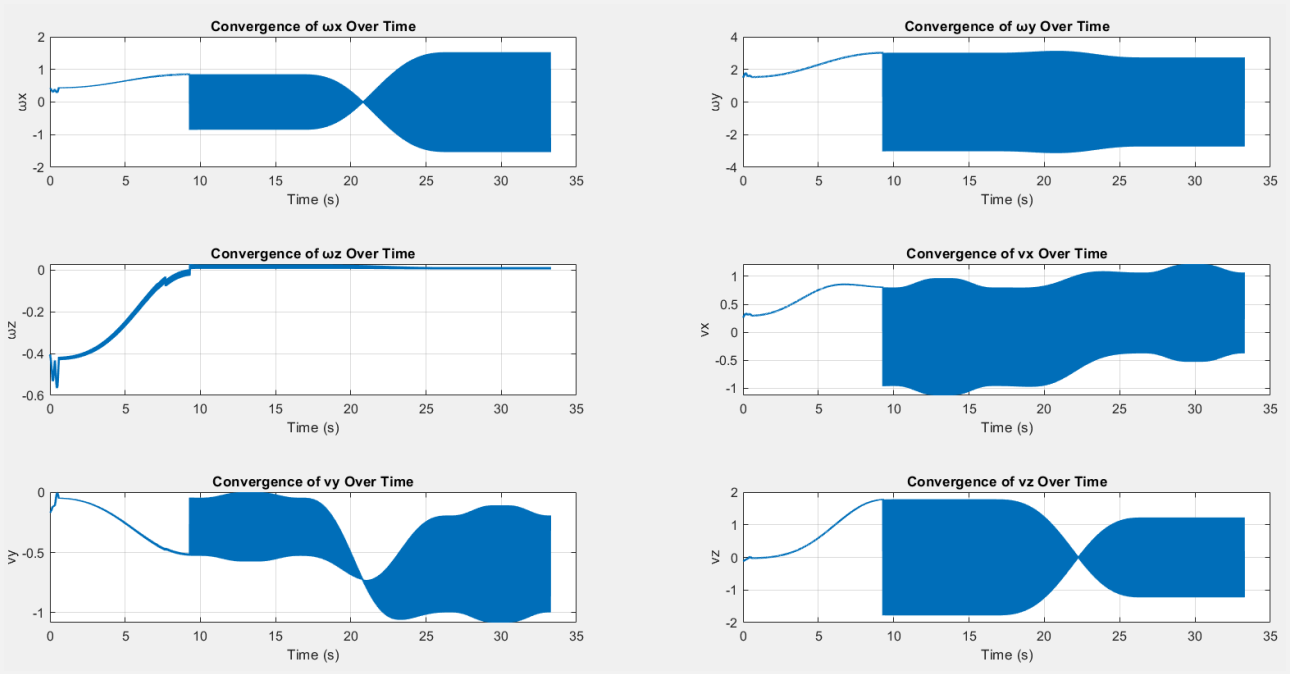
The translational manipulability ($\mu_1(A_v)$ starts high, near 2, because the robot initially has maximum freedom to move its end-effector in all translational directions. As the robot moves toward the cube, the manipulability decreases since the end-effector follows a constrained trajectory to precisely align with the cube for grasping. This reduction reflects the limited freedom in translation due to task-specific constraints. Once the robot successfully picks up and places the cube, the translational manipulability converges back to 2, indicating that the robot has regained full translational freedom after completing the task.

The rotational manipulability ($\mu_1(A_w)$ starts close to 0 at the beginning of the task, as the robot is in a configuration with limited rotational freedom, likely due to joint alignment or an initial pose restricting end-effector rotation. As the robot moves its arm and wrist to approach the cube, the rotational manipulability increases, reflecting greater freedom in orientation adjustments necessary for grasping. During this phase, the factor approaches 1, indicating improved rotational flexibility. When the robot reaches for the cube, a sudden spike occurs due to a singularity, where the arm is in a stretched or degenerate configuration, temporarily increasing rotational freedom at the cost of translational maneuverability.

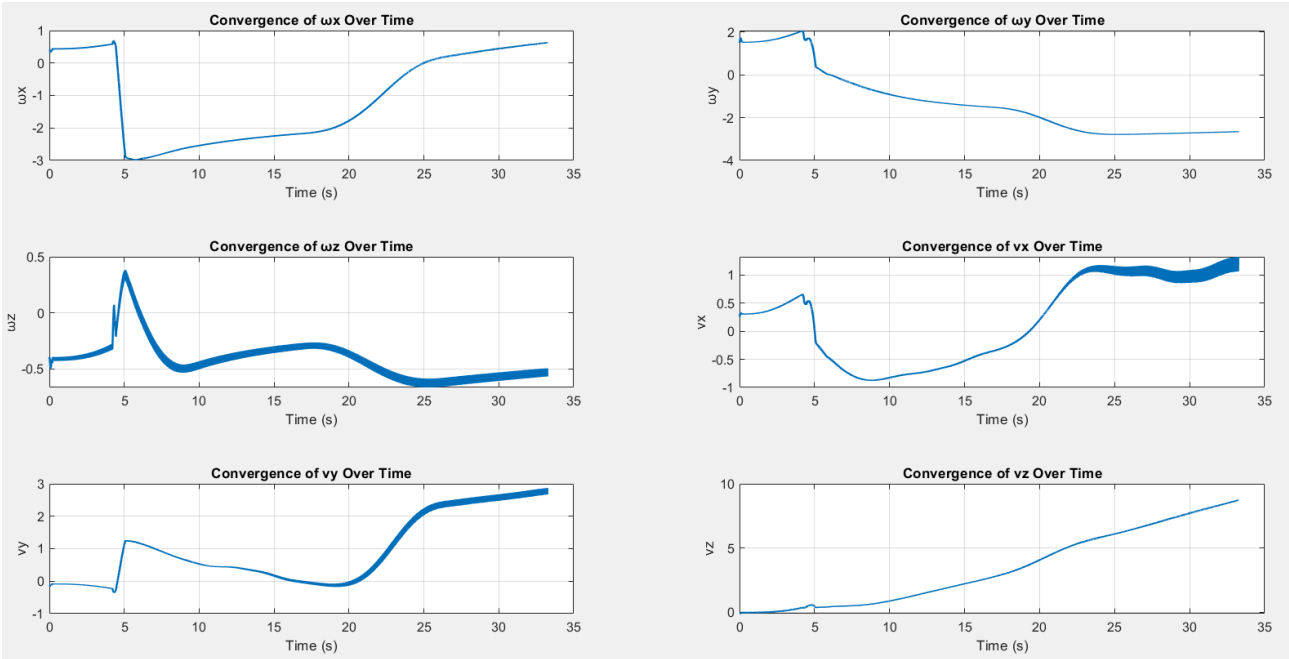
3. If you reduce the maximum joint velocities to low values, you should see error increase again after picking up the cube. Why does this happen?

Upon running the simulation with reduced maximum joint velocities, I observed a significant increase in end-effector tracking error, particularly after the robot picks up the cube. This degradation in performance is due to the robot's limited ability to respond to rapid changes in the desired end-effector pose, especially under increased payload conditions. The added mass alters the system dynamics and requires higher joint velocities to maintain trajectory tracking. However, due to velocity saturation, the controller’s commanded velocities are clipped,

preventing the end-effector from achieving the required motion. This causes the actual trajectory to lag behind the desired path, resulting in elevated configuration-space error X_{err}, notably during fast motions like lifting or transitioning with the cube. Additionally, the persistent error buildup led to signs of integrator windup in the controller, which, if unmitigated, can further destabilize the system and exacerbate tracking inaccuracy.



Velocity Limit = 2



Velocity Limit = 4

4. In this project, our controller directly prescribes the $\dot{\theta}$ to the joints of the robot. For this robot, give an example of a task in which velocity control would be realistic, and another example of a task in which velocity control would not be possible.

Examples of Velocity Control Suitability:
Realistic: Continuous surface polishing, where joint velocities can be regulated for smooth tool movement.
Not Possible: Precise pick-and-place tasks requiring exact positioning, as velocity control alone cannot ensure accurate final configurations without an additional position feedback loop.

5. If we were to implement torque control for the youBot (rather than speed control), what additional information (i.e., input parameters and/or pre-defined constants) would the FeedbackControl function need to know to compute the required torques? Which function from the MR textbook code would it use?

Implementing torque control requires knowledge of joint dynamics, including mass, inertia, Coriolis, and gravity terms. The FeedbackControl function would need the robot's mass matrix M(θ) Coriolis matrix C(θ,), and gravity torques G(θ) The MR textbook function InverseDynamics would be used to compute the required joint torques.

6. Did you use generative AI in solving this project?

Yes, we used generative AI as a supplementary tool to enhance our work. It helped in debugging the code, improving code documentation through comments, and structuring the report. Additionally, we used it to clarify certain technical concepts to ensure accuracy and completeness in my explanations.