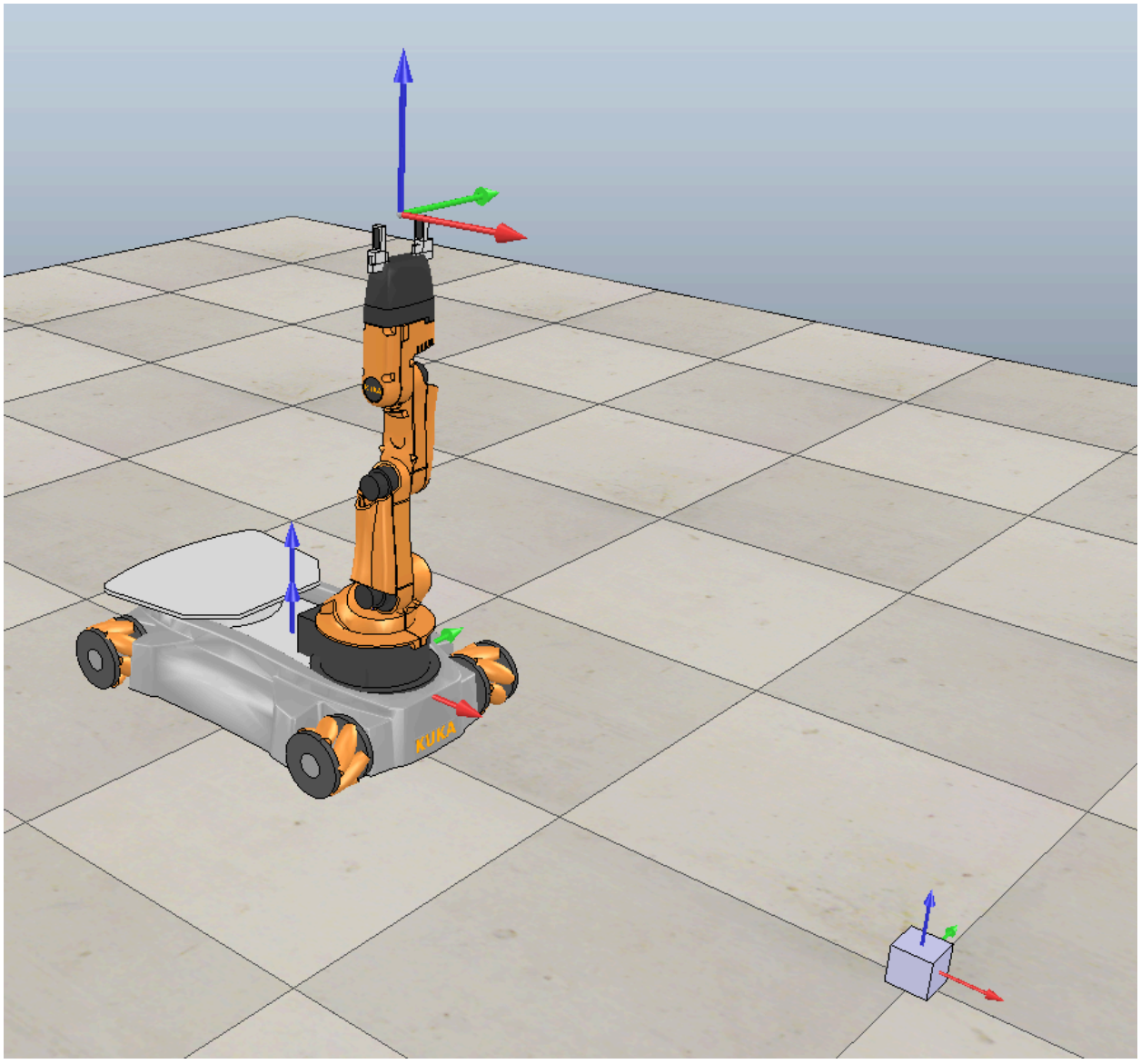
MAE204 - Final Project

Motion Control of a Mobile Manipulator Robot (YouBot)



Manoj Kumar Reddy Manchala

A59024975

SUMMARY

The project focuses on developing a comprehensive control system for a mobile manipulator robot, specifically the youBot, which is tasked with performing a pick-and-place operation. The main objective is to design and implement software components that enable the robot to autonomously navigate to a specified location, pick up an object, and place it at a target destination. This involves creating a simulation of the robot's kinematics, generating a reference trajectory for the task, and developing a feedback control strategy to ensure precise execution of the desired movements.

The project was implemented in MATLAB, and the library from Modern Robotics Toolbox and its functions. For the simulation and visualization of the youBot's behavior, CoppeliaSim was utilized. This simulation software offers a versatile platform for testing robotics applications in a controlled, virtual environment, allowing for the detailed observation and analysis of the robot's performance under various conditions. The integration between MATLAB and CoppeliaSim was achieved through a CSV file-based communication, where the trajectory and control commands generated by the MATLAB scripts were executed within the CoppeliaSim environment to animate the robot's actions.
I have closely followed the structured approach outlined in the project document, which consisted of developing three main components:

Kinematics Simulator (NextState function):
This function is responsible for predicting the robot's next state based on its current state, velocities, and time step, using the Euler method for integration. This simulation provided the basis for planning and controlling the robot's movements.

Reference Trajectory Generator (Trajectory_Generator function):
This function generates the desired trajectory for the robot's end-effector to follow during the pick-and-place task. It computes a

sequence of configurations that the robot should achieve to successfully complete the task, taking into account the initial and final positions of the object and the robot's capabilities.

Feedback Control (FeedbackControl function):

This component implemented a feedforward + feedback(PI) (Proportional-Integral) control strategy to minimize the difference between the robot's actual path and the reference trajectory. It adjusted the robot's motions in real-time to correct deviations and ensure accurate task execution.

Enhancements and Observations:

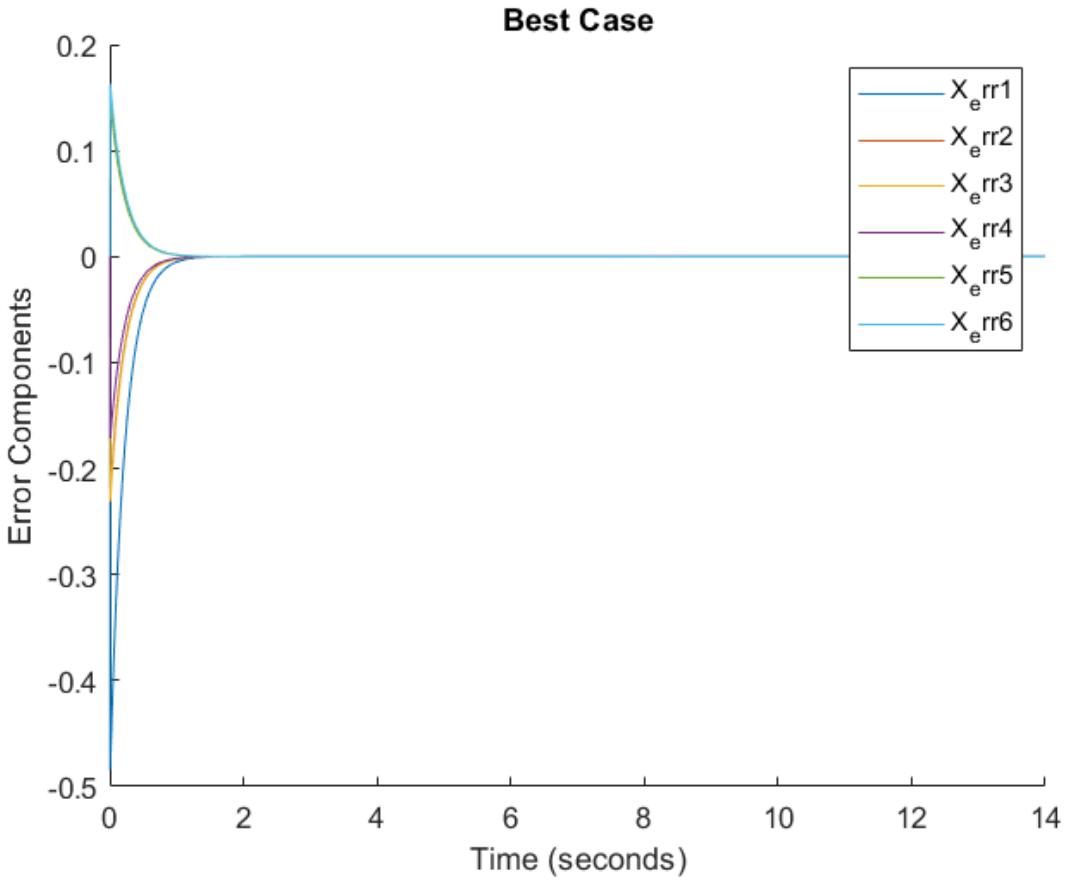
While the primary structure of the project followed the prescribed guidelines, some key observations were noted:

- During the implementation of the project, we have observed that to perform the exact task that is mentioned in the report, some components of the program can be hard-coded which will result in correct implementation but will not be robust to changes. We have implemented a completely adaptable program so that even when the task is changed, the program still works and implements the task.
- The calculation of the current state affects the implementation significantly when the initial state and the initial reference state differ significantly, which should be the case as the reference trajectory is entirely generated based on the reference initial state. But if both the initial state and initial reference state are the same, we can just update the current state of the robot as X = Xd and it will work.
- To avoid singularities when calculating the matrix inverse of the jacobian we implemented 'pinv' with a tolerance value.

RESULTS

In all the following cases, the initial configuration of the end-effector is at least 30 degrees of orientation error and 0.2 m of position error from the first configuration on the reference trajectory.

1. **Best Case:** A Feedward + Proportional Integral controller has been implemented for the best case. The proportional component helps to compensate for the error in the initial configuration of the robot(both end effector orientation and the position error) from the first configuration of the reference trajectory. The integral component helps to force the steady state to zero, but the integral constant is kept to a minimum so that it can avoid any overshoot. The same behavior can be observed from the error plot.



The values of Kp and Ki are tuned to be 1.5*Identity and 0.001*Identity. These specific values ensure that the error converges to zero before the end of segment 1, which ensures that the overall trajectory remains smooth.

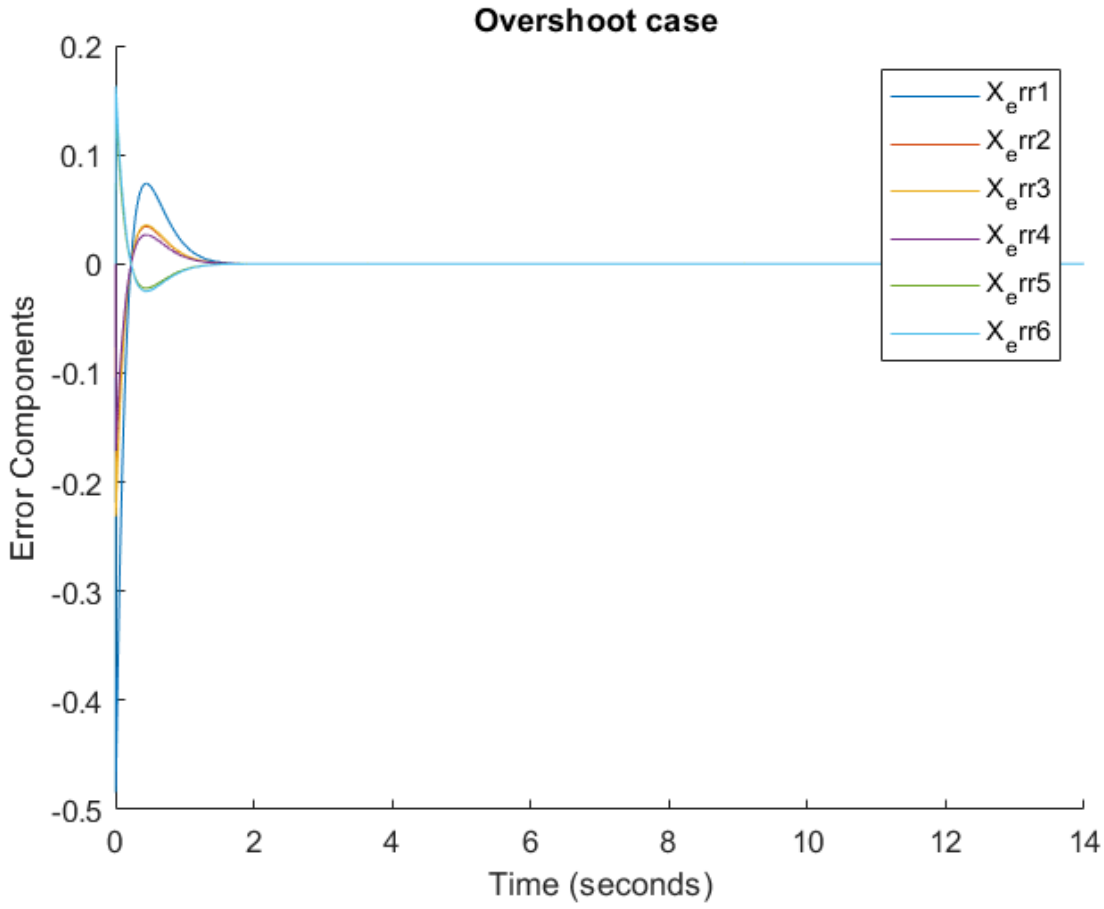
The link to the video is the following - Best_Case_video.avi

2. **Overshoot Case:**

A less-tuned Feedward + Proportional Integral controller has been implemented for the overshoot case. The proportional component helps to compensate for the error in the initial configuration of the robot(both end effector orientation and the position error) from the first configuration of the reference trajectory. The integral component helps to force the steady state to zero. But the Integral constant is significant in this case which causes significant overshoot from the reference trajectory. As a result the proportional constant also needed to be updated to a higher value in order to make sure that the error converges before the end of segment 1. The same behavior can be observed from the error plot.

The values of Kp and Ki are tuned to be 2.85*Identity and 2.37*Identity. These specific values ensure that the error converges to zero before the end of segment 1, which ensures that the overall trajectory remains smooth. We can clearly see that there is an overshoot from the reference trajectory and it can be observed in the following video.

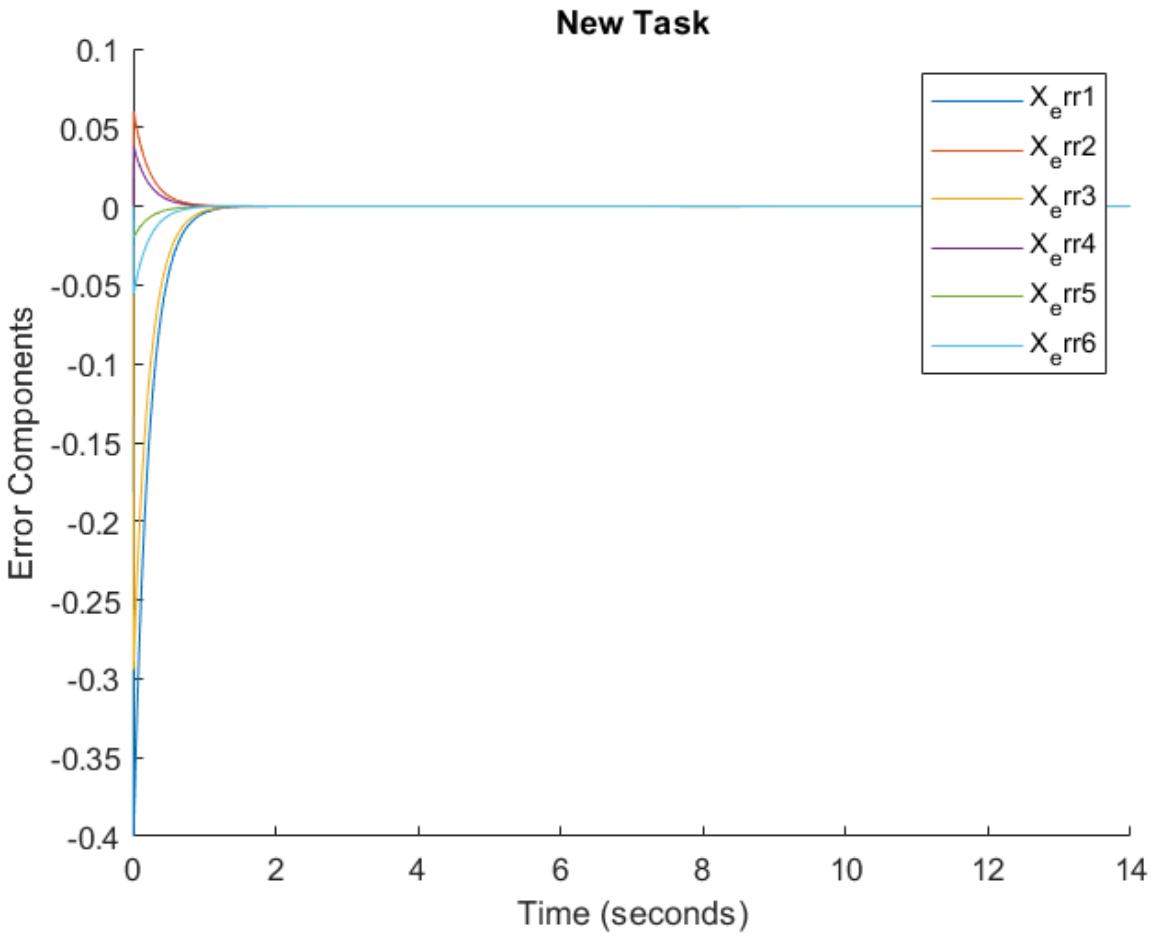
The link to the video is the following - Overshoot_Video.avi



3. **New Task:**

The same “Best case” Feedward + Proportional Integral controller has been implemented for this case. The initial and final positions of the cube are changed to [0.5,0] and [1,-0.5] respectively. The orientations of the cube are kept the same as the best case. This is intentionally done, as now there is a change of 90 deg in the orientation but the angle between the initial and final position vectors is less than 90 deg. We can see that the robot adjusts its approach directions towards the cube in order to handle this change. The proportional component helps to compensate for the error in the initial configuration of the robot(both end effector orientation and the position error) from the first configuration of the reference trajectory. The integral component helps to force the steady state to zero, but the integral constant is kept

to a minimum so that it can avoid any overshoot. The same behavior can be observed from the error plot.

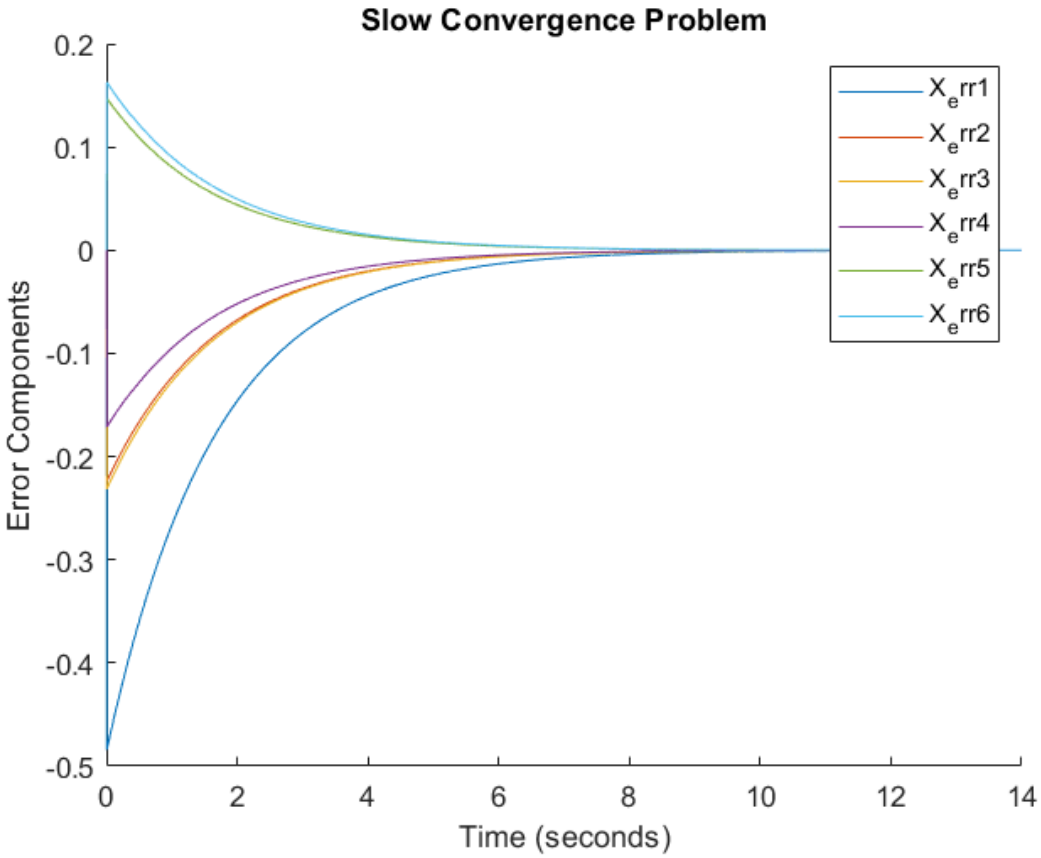


The values of Kp and Ki are tuned to be 1.5*Identity and 0.001*Identity. These specific values ensure that the error converges to zero before the end of segment 1, which ensures that the overall trajectory remains smooth.

The link to the video is the following - New_Task_video.avi

MISCELLANEOUS

4. **Slow Convergence:** A Feedward + Proportional Integral controller has been implemented for this case. The proportional component helps to compensate for the error in the initial configuration of the robot(both end effector orientation and the position error) from the first configuration of the reference trajectory. But in this case, the proportional constant has been set to a significantly lower value than the best case. The effect of the same is that even though the error eventually is reduced, it is not converging before the segments 1 & 2, which means there is an error in the robot configuration at the time of grasping the cub, which results in the robot not picking up the cube.



The same observation can be confirmed through the plot that even though eventually it converged to the reference trajectory, there is significant error after segment 1 & 2 which resulted in the robot not grasping the cube. The same can be seen in the following video.

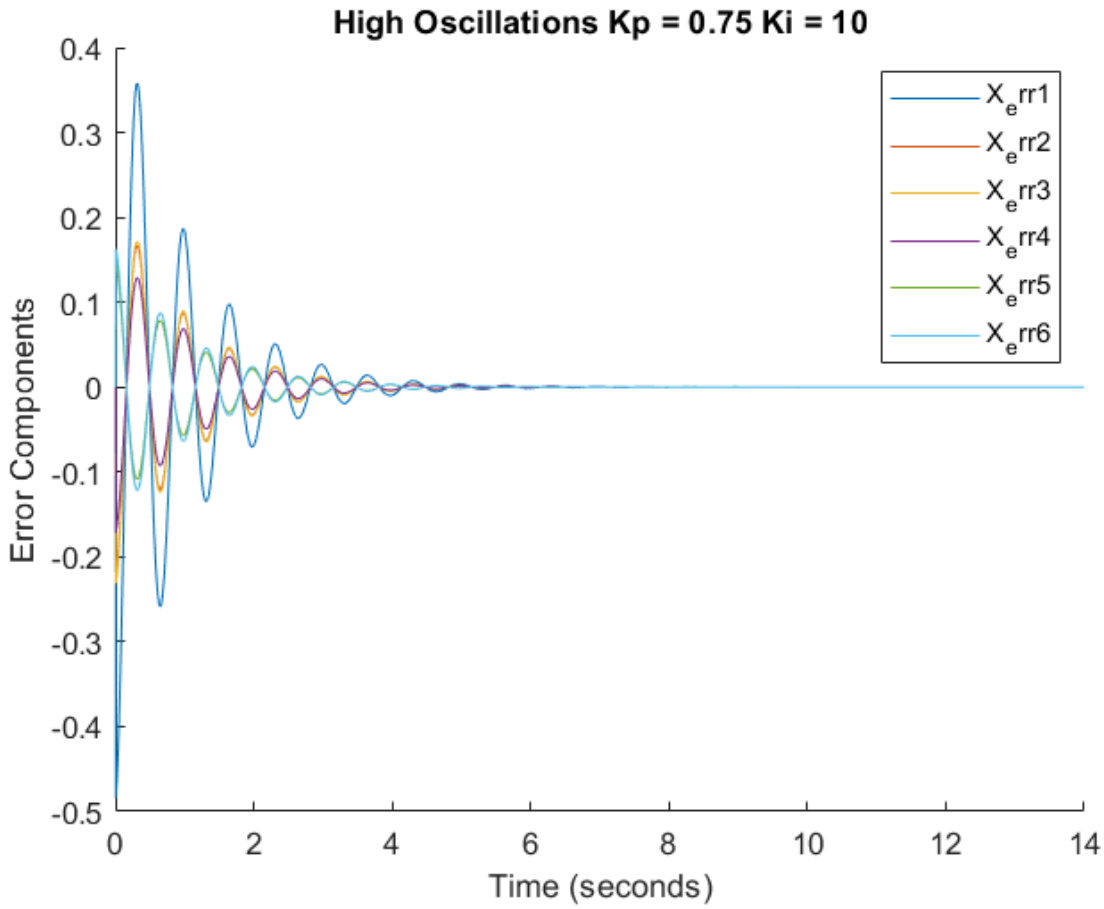
The values of Kp and Ki are tuned to be 0.2*Identity and 0*Identity. These specific values ensure that the error converges to zero but the convergence is slow, which is at the end of the trajectory, the bot reaches the desired location but without the cube.

The link to the video is the following - Slow_Convergence.avi

5. **High Oscillations:** A Feedward + Proportional Integral controller has been implemented for this case. The proportional component helps to compensate for the error in the initial configuration of the robot(both end effector orientation and the position error) from the first configuration of the reference trajectory. But in this case, the proportional constant has been set to a significantly lower value than the best case but more than the slow convergence case. The integral component helps to force the steady state to zero. But the Integral constant is significant in this case, even more than the overshoot case which causes significant oscillations in the trajectory. The same can be observed from the error plot. But the bot eventually converges to the reference trajectory and successfully completes the task. But this type of oscillatory behavior is often not safe.

   The values of Kp and Ki are tuned to be 0.75*Identity and 10*Identity. These specific values ensure that the error converges to zero but the convergence is slow, which is at the end of the trajectory, the bot reaches the desired location but without the cube.

   The link to the video is the following - High_Oscillations.avi



DISCUSSION

1. Incorporating an integrator term in a controller, transitioning from feedforward plus proportional (P) control to feedforward plus proportional-integral (PI) control,helps eliminate steady-state error, thus improving system accuracy by integrating errors over time ensuring that the control action is adjusted to bring the error to zero. However this can introduce overshoot and potentially longer settling times in the system's response. Overshoot occurs because the integrator accumulates error over time, leading to an overcompensation in the control action. When the system tries to correct for even small errors, the integrator can cause the output to

exceed the desired setpoint, leading to an oscillatory response before stabilizing, especially if the system has fast dynamics or if the integrator gain is too high. Therefore, while the integrator improves steady-state accuracy, it requires careful tuning to prevent excessive overshoot and oscillations.

2. Reducing the maximum joint velocities to low values leads to an increase in error after picking up the cube because the robot's ability to respond to dynamic changes in the environment or to correct its trajectory in real-time is constrained. When the robot handles an object like a cube, its dynamics change due to the added weight and altered center of mass. If the joint velocities are capped at low levels, the robot cannot adjust its movements quickly or accurately enough to compensate for these changes. This results in slower response times and less precise control, making it challenging to maintain the desired trajectory or end-effector position. Consequently, the system struggles to correct deviations promptly, leading to increased errors, especially in tasks requiring precise manipulation.

3. Directly prescribing the joint velocities (theta_dot) to the robot's joints is feasible in real robots that operate under velocity control mode which is possible when the robot's actuators (like servo motors) are equipped with velocity controllers that can accurately set and maintain specified joint speeds. Such a setup is common in robots designed for smooth, continuous movements, where the control system needs to dynamically adjust joint velocities to follow a trajectory or maintain a certain speed profile. It's particularly applicable in scenarios where precise control of movement timing and speed is more critical than the exact final position. However, this approach assumes the robot has a well-tuned velocity control system and can respond effectively to the commanded velocities, which is more likely in industrial robots.

4. Implementing torque control for the youBot, as opposed to speed control, necessitates the FeedbackControl function to have additional information, primarily related to the robot’s dynamics. This includes parameters like the mass and inertia of each link, the center of mass locations, and the friction characteristics of the joints. These parameters are crucial for computing the dynamic model of the robot, which relates the joint torques to the accelerations, velocities, and positions of the joints. The function that could be utilized is the InverseDynamics function that calculates the necessary joint torques given the current states of the robot and the desired accelerations. This function uses the robot's kinematic and dynamic parameters to solve the inverse dynamics problem, thereby providing the torques needed to achieve the desired end-effector trajectory and motion control.

5. ChatGPT has been used for this project to debug the code, and to understand the project requirements in a better way.