Introduction

More and more people in the field of robotics are interested in developing safer and more flexible robotic arms. These arms are not like the typical stiff robots; they are referred to as compliant manipulators. They are more adept at adjusting to various environments and have smoother movement. This is especially helpful in situations when robots must operate in close proximity to humans or in disorganized environments.

This research is primarily concerned with a particular type of flexible robotic arm that can move in two different directions. These arms contain unique joints that let them to absorb shocks in different ways, which allows the arm to operate more precisely and intelligently. Creating intricate mathematical models to comprehend the behavior of these flexible joints is a major focus of this research. It's a challenging job because it's difficult to forecast the intricate movements of the joints. To ensure that these robotic arms function properly, this must be done correctly.

Determining the optimal strategy for managing these arms represents a significant component of the study. Since the arms are intended to be flexible, extremely sophisticated control systems are required. They have to be able to adapt to changes in the surroundings and guarantee that the arm travels precisely and steadily. The significance of the research findings lies in their potential to further the development of flexible robotic arms and their control. This is especially helpful in industries where humans and robots must collaborate closely, such medical robotics and advanced manufacturing.

Dynamic Modelling

Creating a mathematical model for a 2R robotic arm (with two rotating joints) involves a few key steps. Here's a simpler explanation of the process:

- **Recognize the Robot's Design:** Start by determining the robot arm's actual physical configuration. Be aware of the joint locations and lengths of each component. To begin modeling, you must have this basic information.
- **Calculate the Arm's Movements:** Apply a technique (such as the Denavit-Hartenberg convention) to derive equations illustrating how alterations in joint angles affect the position and angle of the arm's extremity.
- **Connect Movements and Veloces:** Locate a unique matrix known as the Jacobian matrix. This makes it easier to comprehend how variations in joint speed impact the arm's end's velocity and direction.
- **Motion and Speed Equations:** Utilizing the movement equations from step 2, use calculus to establish a connection between the joint and arm-end speeds.
- **Include Energy Formulas:** Put the Lagrangian equation in writing. This integrates the energy for each arm component derived from movement (kinetic energy) and location (potential energy).
- **Utilize the Lagrange Equations:** Apply Lagrange's formulas to the step five Lagrangian. In order to demonstrate how the robot arm responds to forces, gravity, and its own weight, this stage generates dynamic equations.
- **Resolve the Equations of Movement:** Solve these dynamic equations to have a complete understanding of the arm movements. Usually, this results in intricate equations that require simultaneous solution.
- **Simplify Assumptions:** Because the equations can be rather complex, they are frequently reduced in complexity or simplified to resemble a straight line in order to make them easier

to understand. This could entail overlooking insignificant impacts or assuming particular things about how the arm functions in particular circumstances.

Its mass distribution, geometry, joint configurations, and external forces or torques influence a manipulator's dynamics. The dynamic equation can be represented as:

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + \tau_g(q) = \tau} \quad 1$$

The dynamic modeling equation for the required manipulator is given below:

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D_l(\dot{q} - \dot{\theta}) + k(q - \theta) + \tau_g(q) = 0} \quad 2$$

$$\boldsymbol{J\ddot{\theta} + D_m(\dot{q} - \dot{\theta}) + k(q - \theta) = \tau_m} \quad 3$$

Where,

- M(q) represents the mass or inertia matrix
- $\mathrm{C}(\mathrm{q}, \dot{\mathrm{q}})$ represents the Coriolis and centrifugal effects
- D represents the damping effect
- $\mathrm{k}(\mathrm{q} - \theta)$ represents the nonlinear stiffness effect
- $\tau_{\mathrm{g}}(\mathrm{q})$ represents the vector of joint torques or forces
- J represents the inertia of the motor
- τ_{m} represents the torque of the motor

The matrices required to solve the general equation are the following:

$$\boldsymbol{M(q) = \begin{bmatrix} (m1+m2)l1^2 + m2l2^2 + 2m2l1l2\cos(q2) & m2l2^2 + m2l1l2\cos(q2) \\ m2l2^2 + m2l1l2\cos(q2) & m2l2^2 \end{bmatrix}}$$

$$\boldsymbol{C(q,\dot{q}) = \begin{bmatrix} -2l1l2m2\sin(q2)\,\dot{q2} & -l1l2m2\sin(q2)\,\dot{q2} \\ l1l2m2\sin(q2)\,\dot{q2} & 0 \end{bmatrix}}$$

$$\boldsymbol{\tau_g(q) = \begin{bmatrix} (m1+m2)gl1\sin(q1) + m2gl2\sin(q1+q2) \\ m2gl2\sin(q1+q2) \end{bmatrix}}$$

$$\boldsymbol{K = \begin{bmatrix} k1 & 0 \\ 0 & k2 \end{bmatrix}}$$

$$\boldsymbol{D_l = \begin{bmatrix} D_1^l & 0 \\ 0 & D_2^l \end{bmatrix}}$$

$$\boldsymbol{D_m = \begin{bmatrix} D_1^m & 0 \\ 0 & D_2^m \end{bmatrix}}$$

$$\boldsymbol{J = \begin{bmatrix} I_1 & 0 \\ 0 & I_2 \end{bmatrix}}$$

Where the values of the constant terms are:

Description	Symbol	Value	Unit
Link Mass	m_1	1	Kg
	m_2	1	
Link Length	l_1	0.4	M
	l_2	0.4	

Link Damping	D_1^l	0.3	Nms/rad
	D_2^l	0.3	
Motor Inertia	J_1	0.1	Nms^2/rad
	J_2	0.1	
Motor Damping	D_1^m	0.2	Nms/rad
	D_2^m	0.2	
Spring Stiffness	k_1	10	Nm/rad
	k_2	10	
Gravitational Acceleration	g	9.81	ms^{-2}

Control System Design

Combining PID controllers and feedforward dynamic models is a clever technique to manage machines or systems in control system design. The goal is to improve the functionality of these systems by incorporating a prediction component. As intelligent tuning tools, PID controllers respond more effectively thanks to this predictive component. This combination helps the system remain resilient and adaptable so that it can continue to function effectively.

Feedforward Dynamic Model

For control systems, the feedforward dynamic model functions as a kind of prophecy tool. It's a collection of mathematical formulas that predict how modifications to the system will impact the result. Because it takes into account intricate details like friction and the effort required to move objects (inertia), it is incredibly intelligent. The system is constructed according to the physical laws it adheres to. The feedforward model becomes more valuable with the addition of these intricate details, particularly in systems with complex behavior. This is how the math formula looks:

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D(t)\dot{q} + t_g(q) = \tau} \quad 4$$

PID Controllers

An integral component of the control system are PID (proportional, integral, and derivative) controllers. They operate by continuously changing the input for control in accordance with three distinct terms:

- **Proportional (P):** This component modifies the control in response to the present error, which is the discrepancy between the desired and actual outcomes. It works to lessen this mistake.
- **Integral (I):** This section adds up previous errors over time to assist correct persistent, long-term errors. It attempts to get rid of recurring discrepancies between the intended and real results.
- **Derivative (D):** This section forecasts future errors by examining how quickly the error is changing. Like a stabilizer, it aids in preventing the system from moving too far in either direction.

The PID mathematical model looks like this:

$$\boldsymbol{u = k_p \cdot e(t) + k_i \cdot \int e(t) + k_d \cdot \dot{e}(t)} \quad 5$$

Integration of Feedforward and PID

A well-rounded control system is produced by combining PID controllers with the feedforward dynamic model. By anticipating input reactions, the feedforward component aids in the system's adaptation to changes. PID controllers, on the other hand, continuously adjust inputs based on present mistakes in order to respond swiftly to any deviations from the intended course.

Benefits of Feedforward Dynamic Model-Based PID Control

- **Faster Response:** The system can react to changes more quickly thanks to the feedforward model's predictive power, which also shortens the time it takes to fix problems.
- **Improved Stability:** PID controllers modify inputs in response to faults, which helps to maintain stability. The feedforward model's predictions are included, making the system more controllable and stable.
- **Managing complicated Situations:** A feedforward model that takes nonlinear behaviors into account makes the system more adept at managing complicated dynamics.

Implementation

It is crucial to adjust the PID controller's parameters and confirm the accuracy of the feedforward model when configuring this. This process of fine-tuning guarantees the system performs optimally in various scenarios.

Inverse kinematics (Conversion from Joint to Cartesian Space)

Joint space was initially used to set up and regulate the 2-DOF robot system, meaning that the angles or movements at the robot's joints were crucial. These joint movements served as the foundation for the robot's regulations, which in turn served as a model for more research and control strategies.

The robot needed to go from joint to Cartesian space in order to become more flexible and useful. Cartesian space, sometimes referred to as operational or task space, is a global coordinate system that specifies the location and orientation of the robot's end (such as a hand or tool). With this modification, it becomes simpler to explain the robot's movements in a way that is more intuitive and natural by employing coordinates appropriate for the given task.

This switch is primarily based on inverse kinematics. It establishes a mathematical connection between the joint motions and the corresponding Cartesian space positions. Inverse kinematics allows you to specify the desired location of the robot's end in Cartesian space and provides the necessary joint movements.

Steps in the Inverse Kinematics Process

- **Establish Cartesian Variables:** Define variables in Cartesian space that specify the location and orientation of the robot's end.
- **Equations for Inverse Kinematics:** Create mathematical formulas that relate the joint movements of the robot to these Cartesian variables. The geometrical and physical properties of the robot are reflected in these equations.
- **Solve Kinematics Inverse:** Determine the joint movements required in Cartesian space to obtain the desired orientation and position. Depending on how sophisticated the robot is, either numerical methods or direct solutions can be used to accomplish this.
- **Enhance the Control Model by adding:** The joint movements are rapidly incorporated into the robot's overall control model after they have been determined. By utilizing Cartesian space in its control method, this aids the robot's ability to adapt to various jobs and surroundings.

By shifting from joint space to Cartesian space, inverse kinematics allowed for a more task-oriented and intuitive representation for the modeling and control of the 2-DOF robot system. This made a wide range of robust and flexible robotic applications possible. This conversion is required in scenarios where specifying motions and locations in Cartesian space makes more sense and is consistent with the robot's operating requirements. In mathematical terms, let

- q be the vector of joint space variables.

- x is the vector of Cartesian space variables.
- J be the Jacobian matrix.

The iterative update equation is given by:

$$\boldsymbol{q_{new} = q_{old} + \Delta q} \quad 6$$

Where

$$\mathbf{\Delta q = \alpha \cdot J^{-1} \cdot \delta x} \quad 7$$

Where

- α is a scalar that controls the step size in the joint space.
- δx is the desired change in Cartesian space.

The Jacobian matrix J relates the rate of change of the end-effector position and orientation to the rate of change of the joint variables:

$$\boldsymbol{\delta x = J \cdot \delta q} \quad 8$$

In this case, δq represents the differences in the joint space variables. The iteration continues until the difference between the intended and current Cartesian configurations is small enough. Recall that choosing an appropriate step size (α) is critical to the stability and convergence of the iterative process. Moreover, using the pseudoinverse or other methods to derive $J-1$ should be considered when the Jacobian might not be invertible.

Validation & Simulation Responses

Cartesian space and joint space were both utilized in the controller's design. As previously stated, the Cartesian space was subjected to inverse kinematics. Having two different kinds of controllers increases flexibility and enables the control strategy to be customized to the particular needs of the work. The equations from the dynamic model were initially used to represent the robot in Simulink.

Building a realistic simulation of the robotic system is the first stage in developing a control system. Simulink, a graphical programming environment built on top of MATLAB, is used for this. Typically, the modeling method entails the following steps:

- **Dynamic Representation of Models:** The dynamical equations of the robot, which are derived from its kinematics and physical properties, are used to build a Simulink model. The interactions between the joints, connectors, actuators, and sensors, among other components of the robotic system, are captured by this model. For the model shown in below differential calculus was used to derive the model and simulated in the MATLAB using mathematical tools available for simulation.

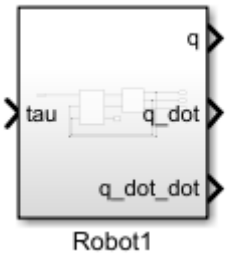

Figure 1: Dynamic Model of Robot being presented in Simulink

- **Subsystem Configuration:** Each subsystem that comprises the robot model represents a particular component such as manipulator and motor as shown in the below figure. This modular approach makes the Simulink model simpler to comprehend and update. Here manipulator model contains the mechanical properties of manipulator itself other than the actuators (motors) used. The inner model of manipulator is shown in the figure 4 where it can be seen that manipulator as described in the modelling section is developed in Simulink. Where as in figure 3, motor's model is given which contains interaction of mechanical components of an actuator and their relationship with manipulator using a spring and manipulator linkages.

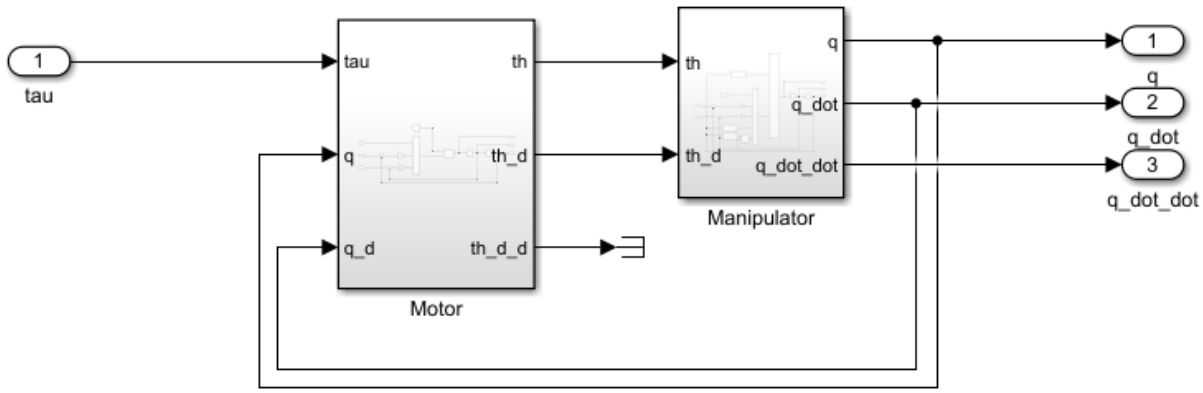

Figure 2: Robot Model as a Whole

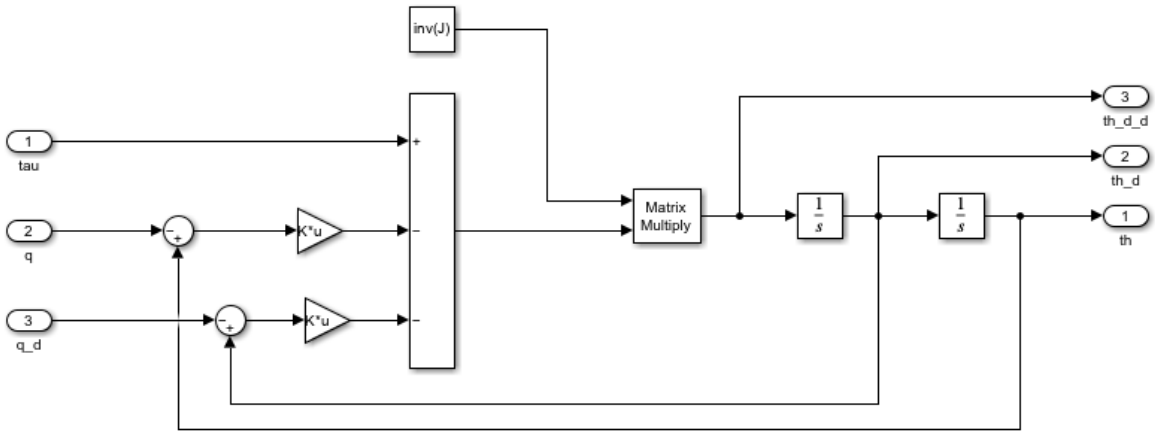

Figure 3: Motor Dynamics

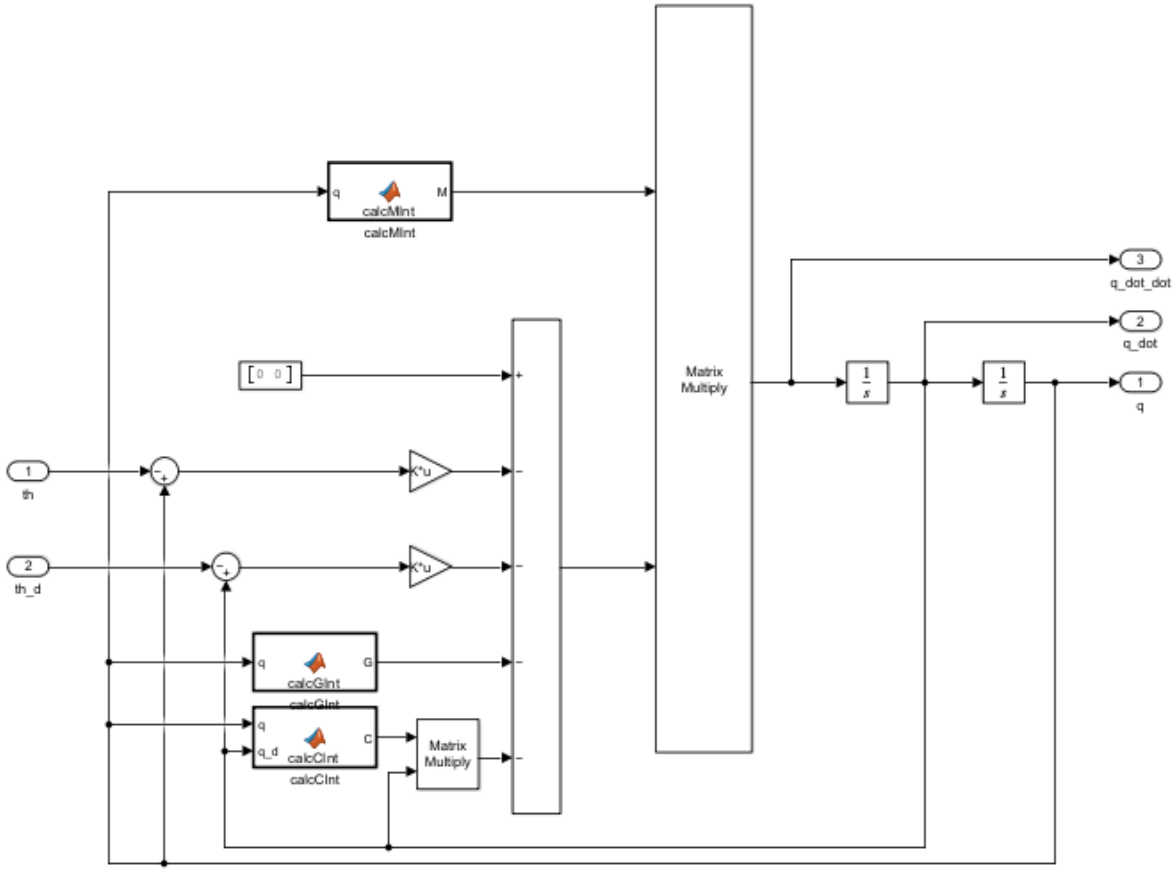

Figure 4: Manipulator Dynamics

- **Verification and Validation:** The Simulink model is verified and tested to ensure that it completely depicts the behavior of the robotic system in real life. To do this, it could be required to compare simulation results with experimental data or analytical solutions. For this robot one can easily assume that how robot should work for certain inputs given in real life. Such as for no input robot should not move as it was modeled in stable equilibrium position and for a little torque input it should move a certain angle, for an impulse input it should move and come back to stable position due to friction and gravity. In the following figures models for verification and results are being presented for the robot.

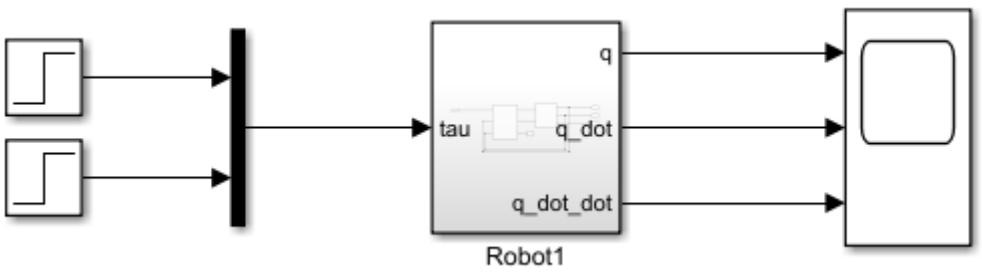

Figure 5: Model validation using step responses

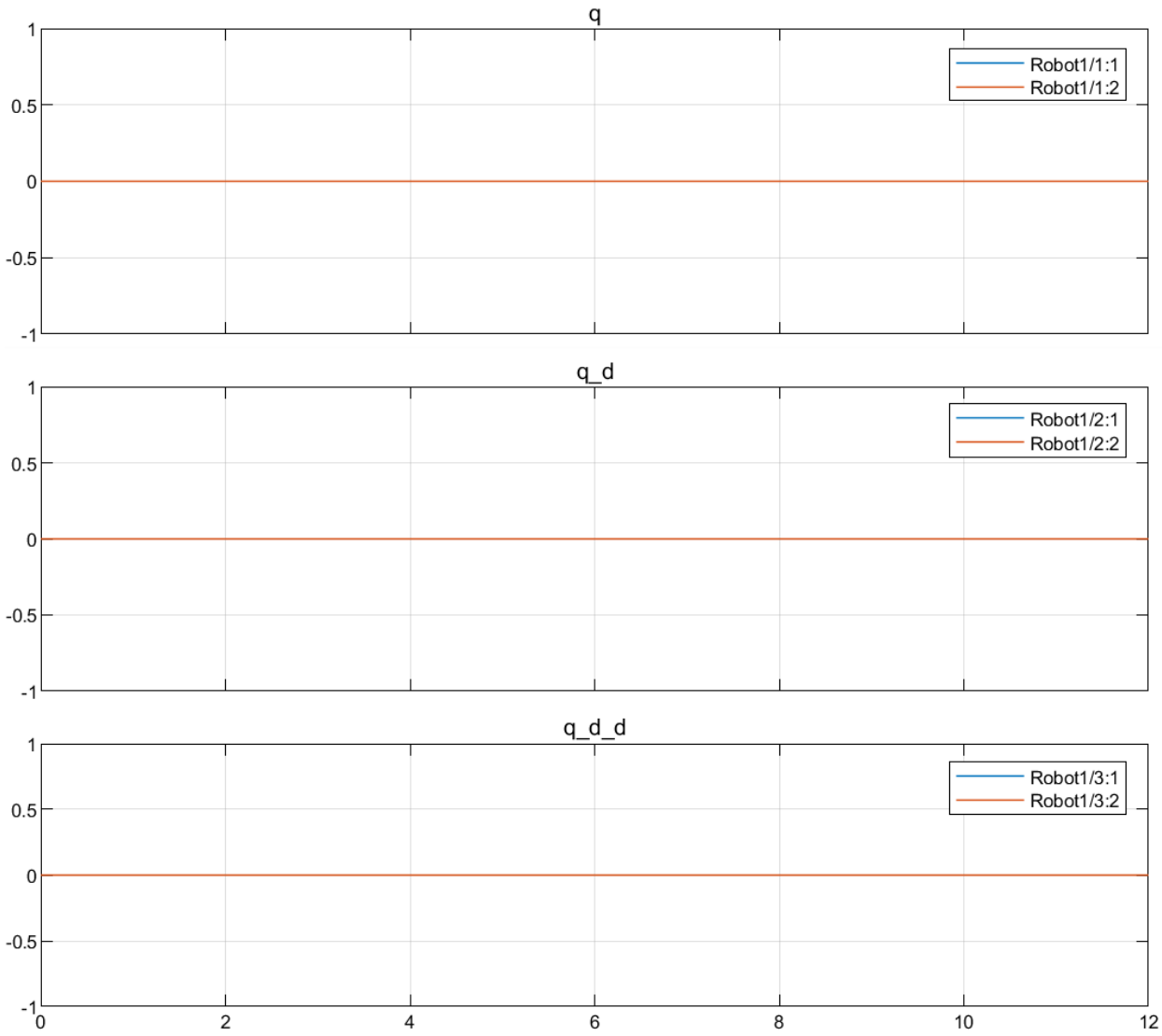

Figure 6: Output Response with zero input torque ($\tau = 0$)

For no input robot doesn't move which shows that it is in equilibrium position and zero torque input has no effect on the robot which is understandable according to real life situation. For figure shown below, an input torque was provided which makes the robot move up to certain amount of angle and due to friction and gravity the acceleration is reducing which is also showing a real life behavior.

Thus, it can be said that robot model is correct and is following the real life robot's behavior and is good to move on the next stage that is control system designing and simulation in Simulink and matlab.

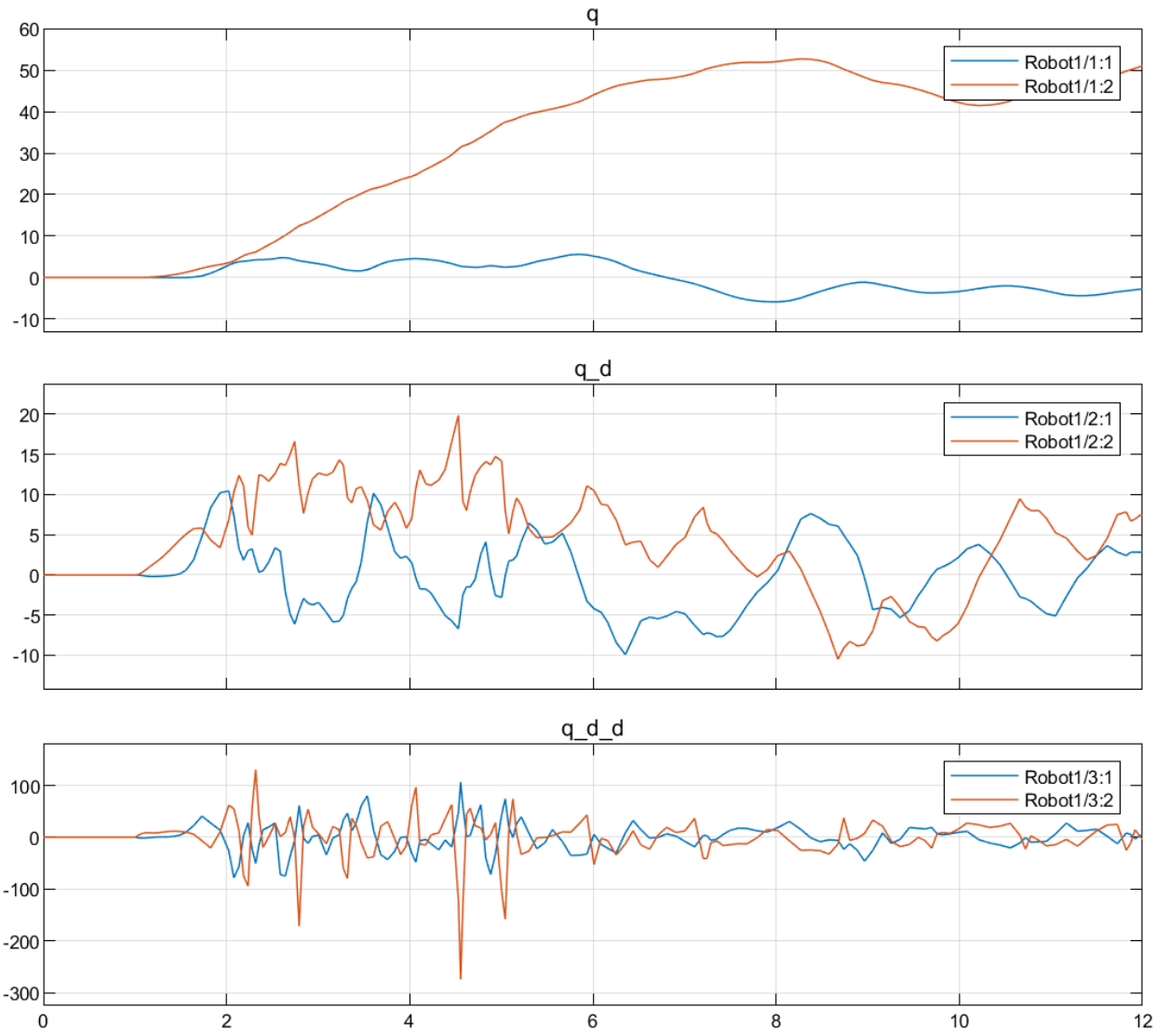

Figure 7: Output Response with step input torque input (τ=1)

Joint Space Control Model

An essential component of robotic system control is joint space control. It focuses on modifying every joint on the robot to perform a particular function or to move the robot's extremities (such as a hand or tool) along a predetermined path. By directly controlling the robot's motors or actuators, this technique modifies the angles of the joints, which in turn modifies the location and motion of the robot's end.

Sending the appropriate signals to each joint is the aim of joint space control. This facilitates the robot's path following. Usually, the angles or joint movements along this path indicate the expected movement of the robot.

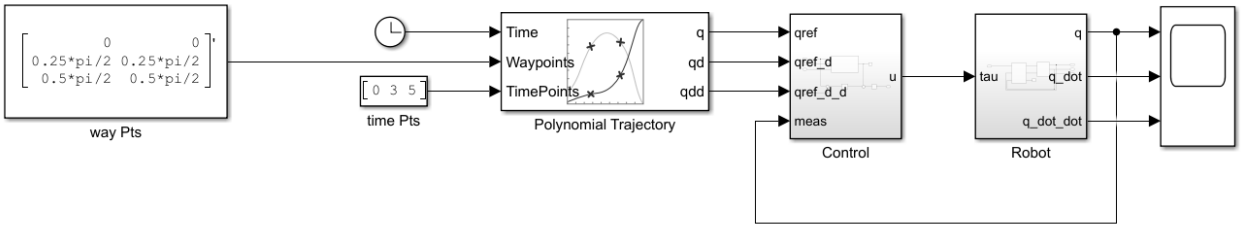

Figure 8: Top-level Controlled Model with Robot Dynamics and Trajectory Generation

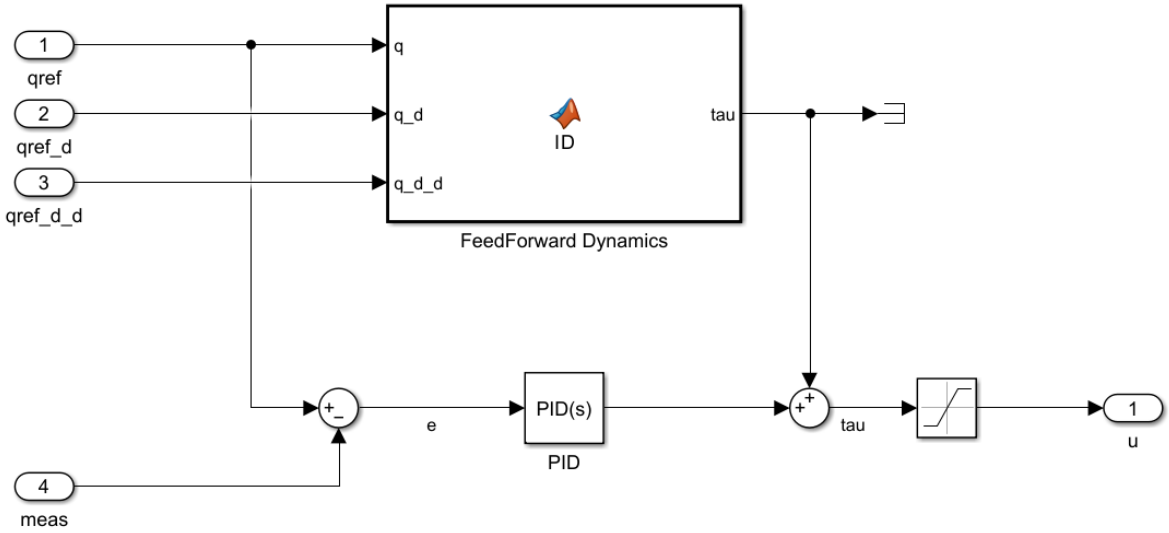

Figure 9: Robot Controller with Feedforward System and PID

Cartesian Space Control

The primary goal of the Cartesian space controller is to precisely control the location and orientation of the robot's extremities, such as its hands or tools. It models the robot's movement by connecting the joints' motions to locations in Cartesian space through the use of inverse kinematics. Joint movements are translated into high-level task needs by this controller. In order to accomplish this, it uses inverse kinematics to determine the proper joint movements by taking the necessary locations and angles in Cartesian space. Because the control inputs for this method are limited to the location and orientation of the robot within its working area, task planning is made easy.

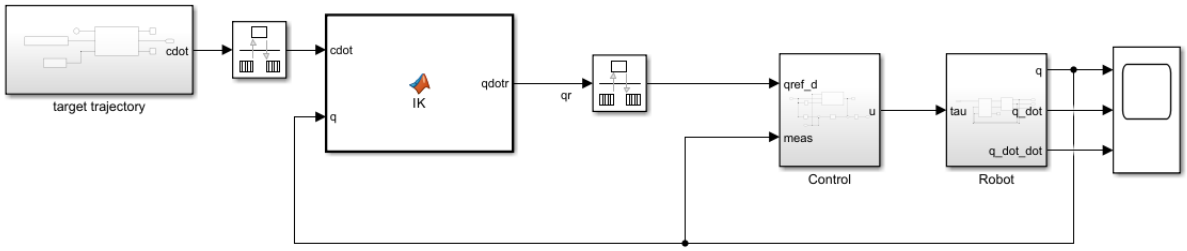

Figure 10: Top-level Model for Cartesian Space Controlled Robot Model

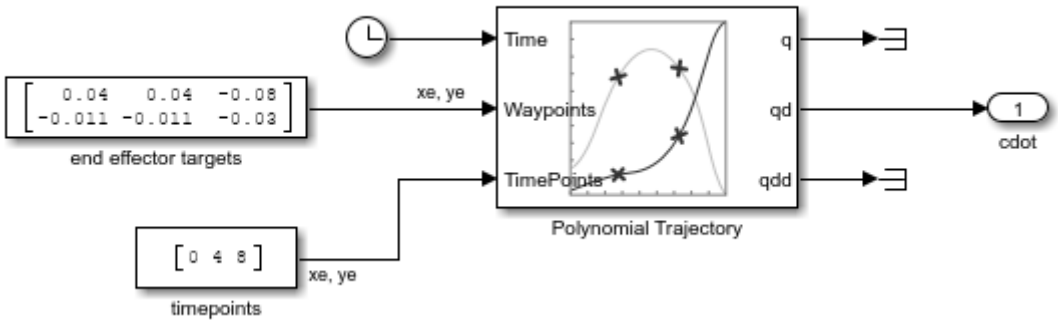

Figure 11: Trajectory Generation for Cartesian Space

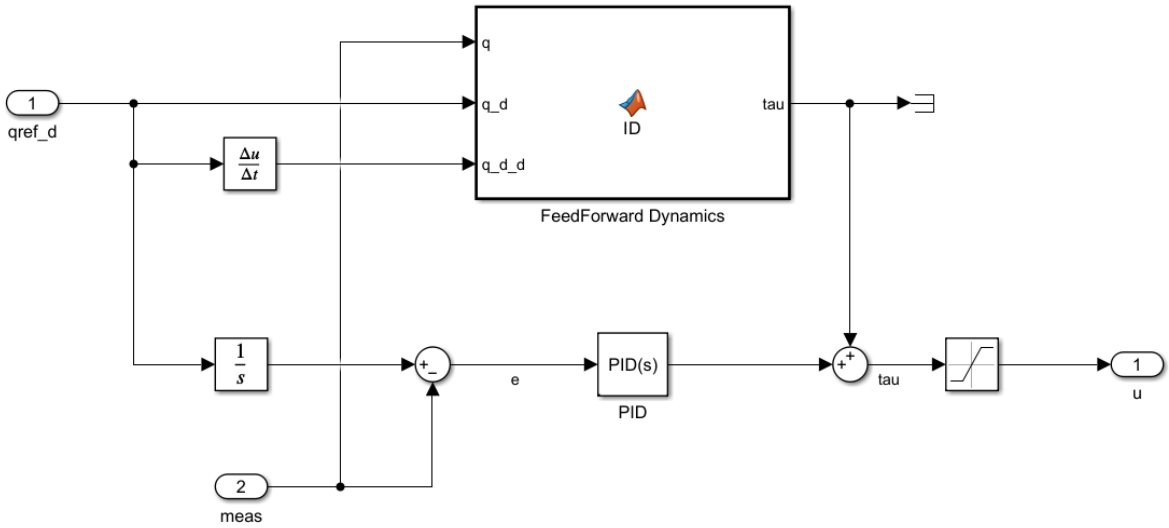

Figure 12: Controller Model for Cartesian Space Control

Controller Performance

These are the responses for the controller's performance when the model is given an input trajectory using MATLAB's input trajectory generation blocks from robotic systems toolbox. Here, it is clear that there is very little overshoot, a decent settling time for the controller, 0% steady-state error, and no oscillations in robot states like $q, \dot{q}$, and $\ddot{q}$, which makes it an excellent control setup for such a robot manipulator. The robot is following the trajectory provided with an acceptable amount of energy signal which can also be seen in the figure 14. That energy signal describes the amount of torque supplied to the robot to make the movement successful provided by input trajectory generator.

The oscillation of the system shows the vibrations in the robot which are also negligible, it can be seen that the acceleration oscillations are continuously decreasing which makes the controller performance more robust and effective in tracking the input signal.

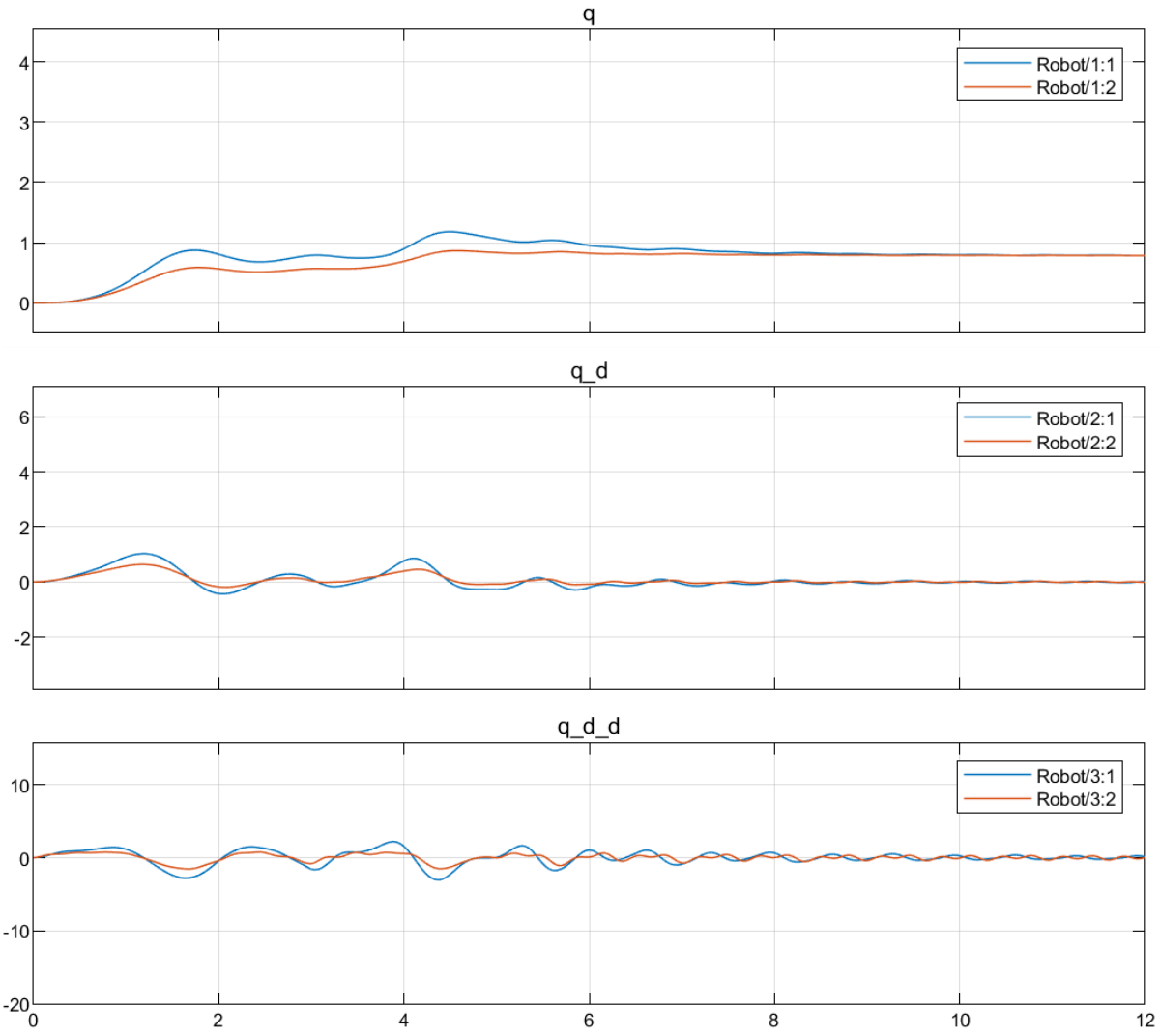

Figure 13: Controller response for Input Trajectories $(q, \dot{q}, \ddot{q})$

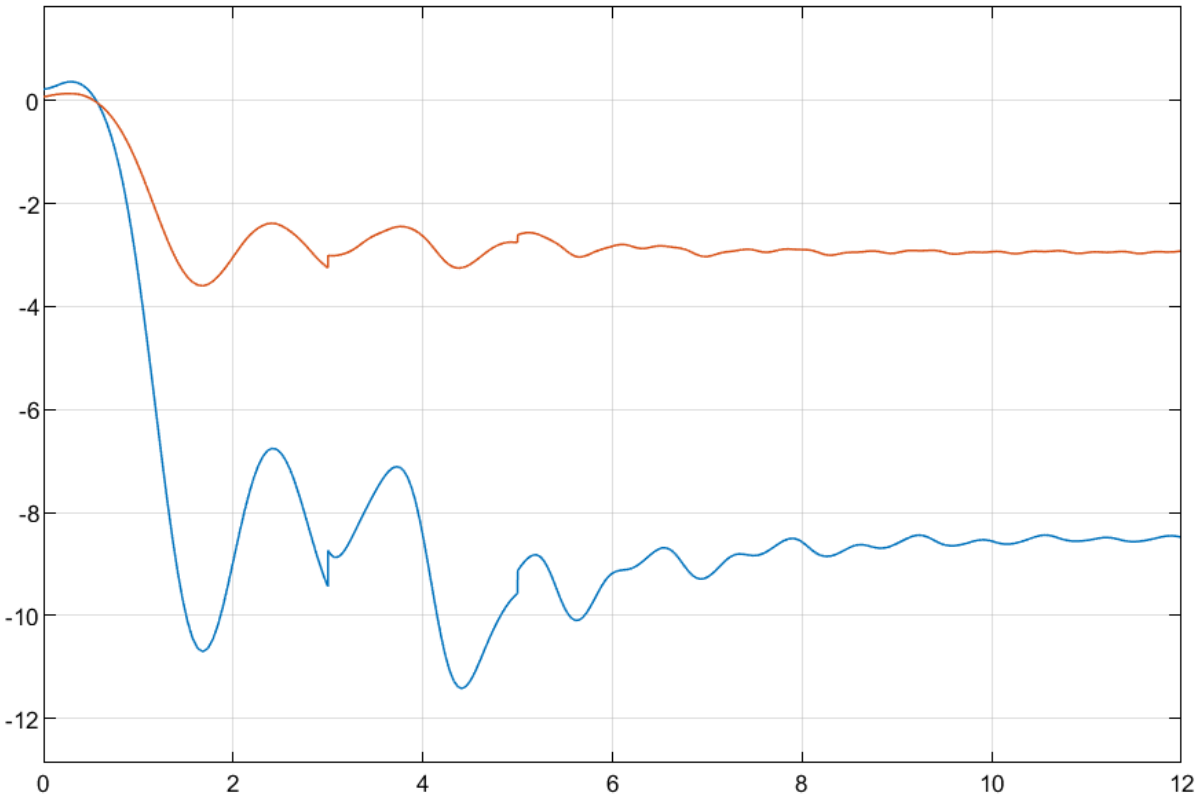

Figure 14: Controller Torque (u)

Mechanics-Based System in Joint and Cartesian Space

Here are joint and Cartesian space-based models with control systems developed using sim-mechanics in Simulink. This uses the exact robot specifications to show the robot in real-time with a sim-mechanics 3D viewer. Rendered version of the robot can be seen the figure 16, where the robot is presenting a posture which is provided using input trajectory generation block as shown in the figure 15 below.

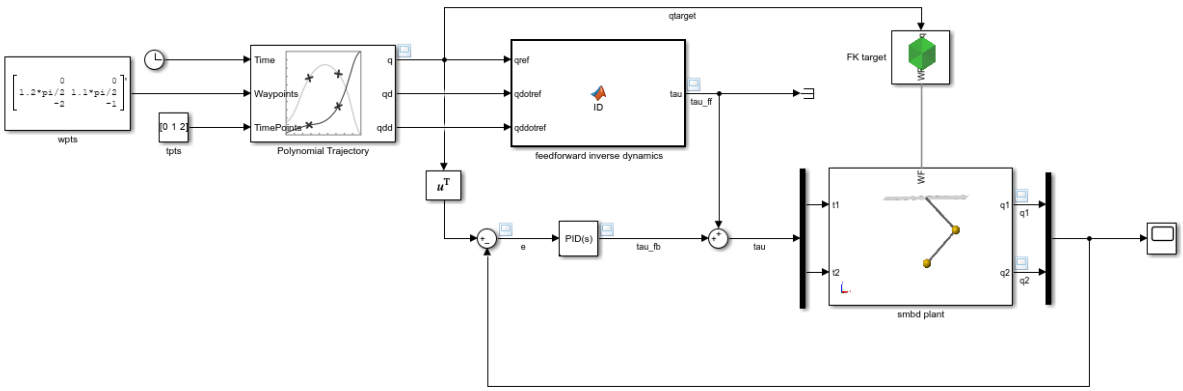

Figure 15: Joint space-based sim-mechanics Model with Control System

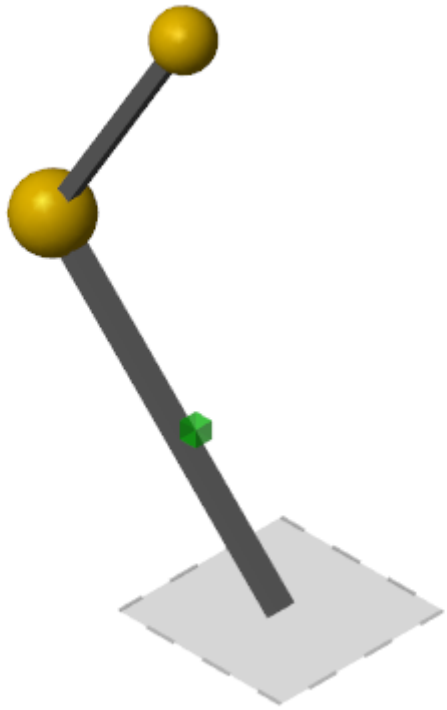

Figure 16: Sim-mechanics simulation for Joint Space Model

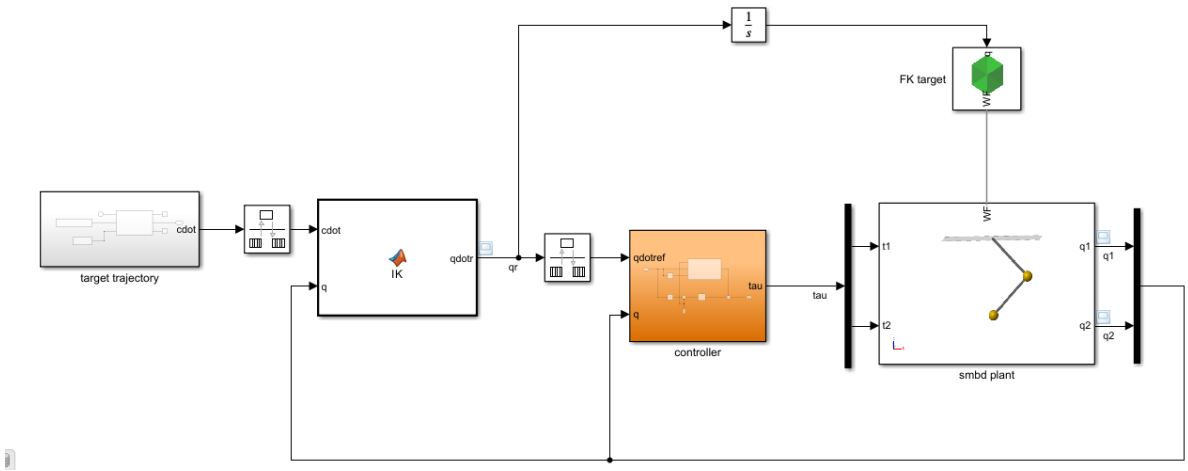

Figure 17: Cartesian space-based sim-mechanics Model with Control System

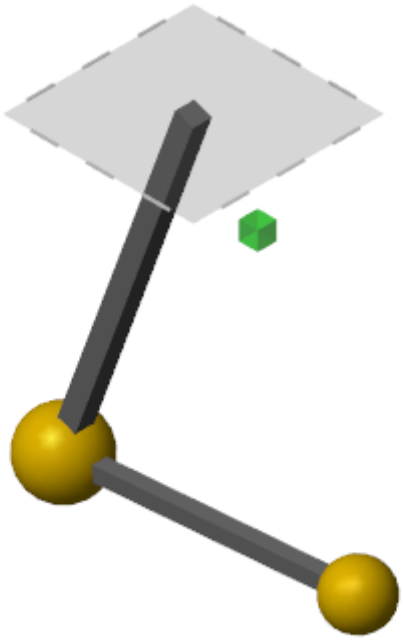

Figure 18: Sim-mechanics simulation for Cartesian Space Model

Singularities (Unstable Regions)

One type of singularity in a 2 Degrees of Freedom (DOF) compliant robot is a special configuration where the system's control or mathematics becomes complex. These configurations may have an impact on the robot's performance, stability, and controllability. Typical kinds include of:

- **Kinematic Singularities:** These occur when a robot loses some of its freedom of motion due to its joint configuration. This may result in pointless movements or difficulties accessing certain regions where it is intended to function.
- **Rigidity Singularities:** These arise when a robot's compliance, or its capacity to yield under force, becomes abnormal as a result of its stiffness and damping. They are frequently associated with a single stiffness matrix. For activities requiring a high degree of precision, too much compliance might cause unanticipated movements and make directing the robot challenging.

Avoiding these singularities requires careful planning, control, and design for obedient robots. These problems can be addressed by employing techniques like real-time feedback sensors and singularity-tolerant controllers. The nature of the singularity can vary depending on design decisions, control strategies, and intended robot usage, so it's critical to do in-depth study and simulations during the design and control stages.

Testing (Robustness & Dynamic Model Tunning)

The robustness testing step's objective was to evaluate the robot control system's performance under various real-world scenarios. The weight and length of the second portion of the robot arm were slightly adjusted in order to test the system's ability to adapt to changes in the robot's behavior. These modifications were reflected in the simulation by modifying the dynamic model within the Simulink

environment. The response of the control system to these novel circumstances was then observed using this modified model with joint space and Cartesian space controllers. The robot's balance and movement were impacted when the weight of the second arm component was raised. Additionally, the robot's shape was altered by significantly extending this section, which made controlling the arm's end more difficult.

Several metrics were used to assess the performance of the control system. It indicated a slight overshoot, indicating that it stayed rather close to the intended course. This was advantageous since it meant that there were no significant fluctuations in movement and that the system rapidly settled down. Additionally, the system demonstrated its ability to finish jobs quickly by getting to the desired location and staying there. The arm constantly arrived at the precise location it was meant to because there was no persistent mistake.

It was thought that the control energy used was satisfactory. The robot was being controlled with a decent amount of effort, according to the statistics of the control signals. It didn't push the robot's components too hard or consume excessive energy. Maintaining control actions within tolerances is critical to ensuring long-term robot functionality.

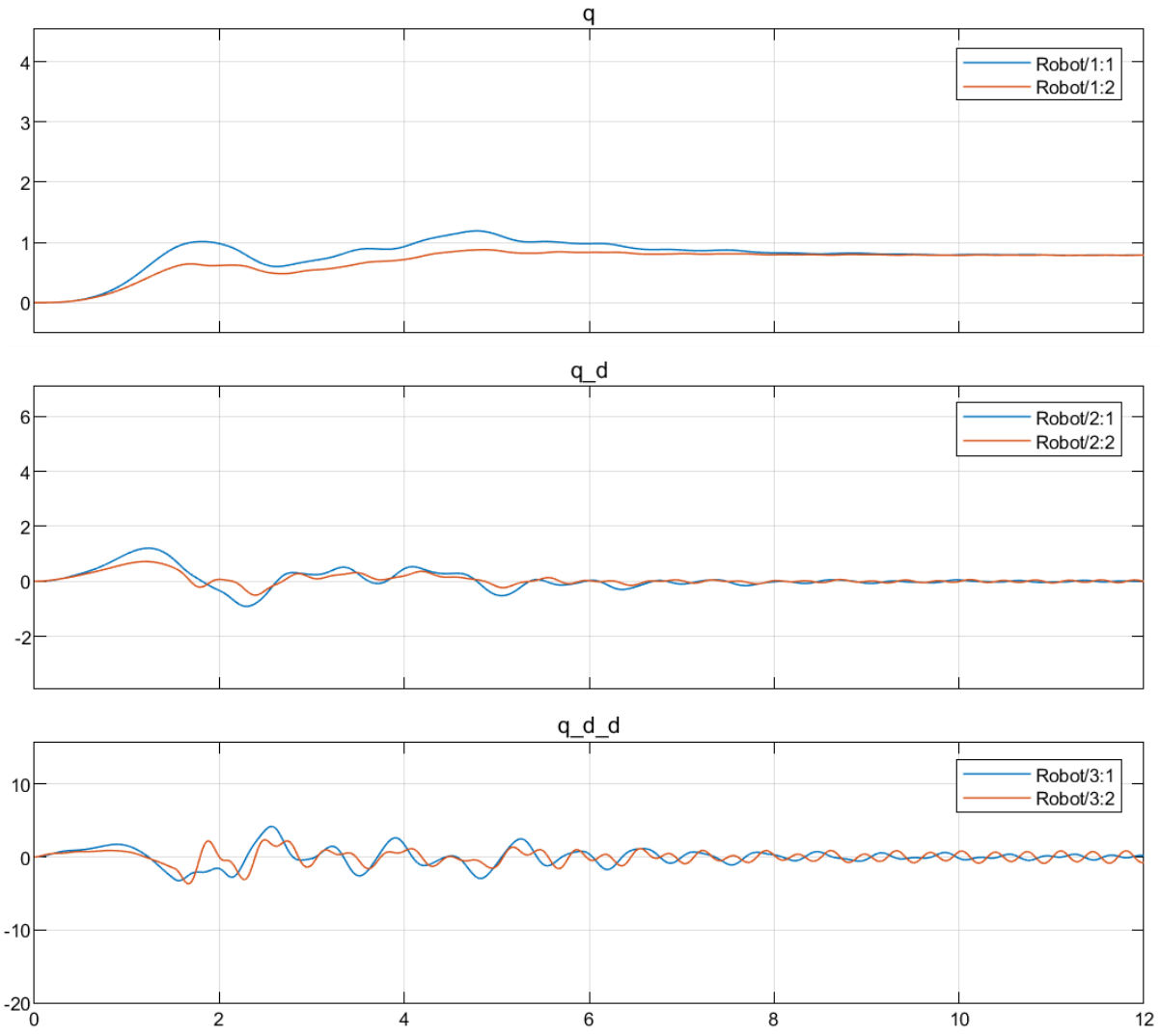

Figure 19: Robot Performance with Changed Parameters

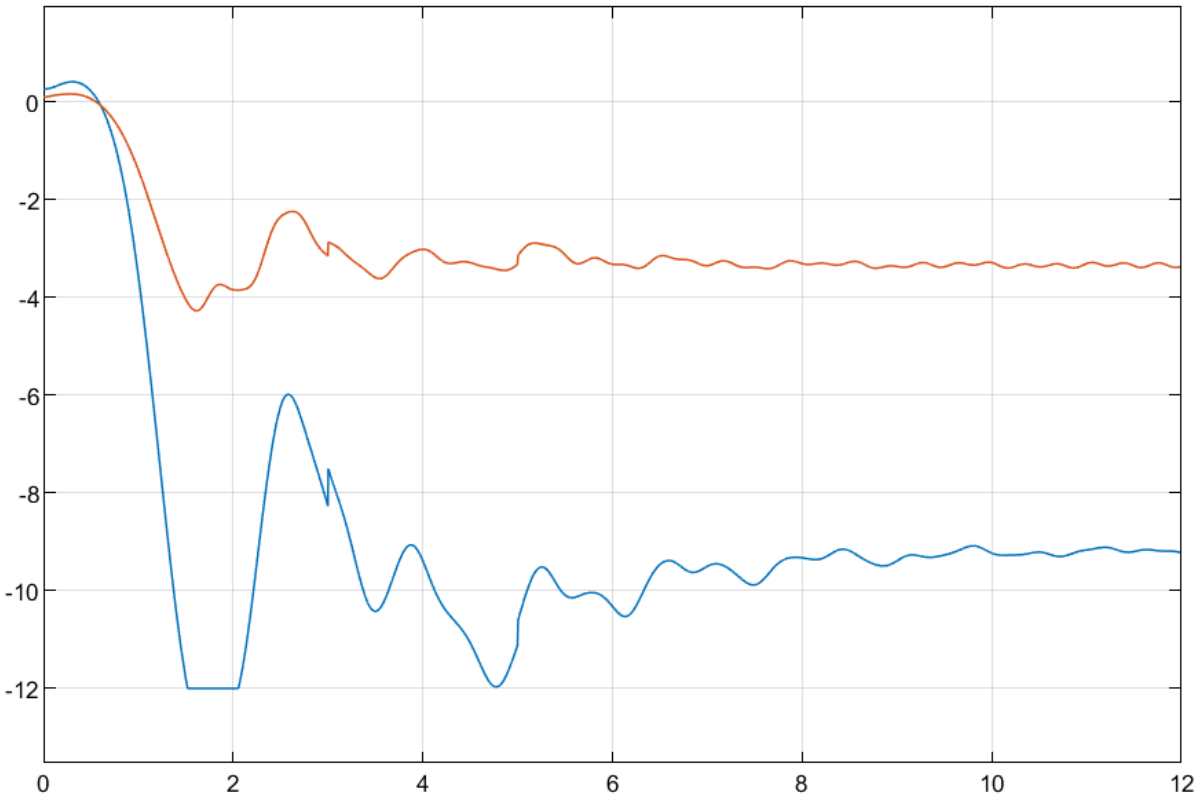

Figure 20: Control Signal with Changed Parameters

Conclusion

In this study, I explored the design and control of a robotic system with particular attention to joint space and Cartesian space control techniques. I worked hard to construct the dynamic model of the robot with state space representations and matrices. Our control system had a solid base because of this. For joint space control, I used a feedforward system with a PID approach, and for Cartesian space control, I used inverse kinematics.

The dynamic model—which was modeled using matrices and state space equations—was essential to comprehending the behavior of the robot. I were able to understand the dynamics of the system by using these matrices to help us capture the intricate relationships between the joint variables. The robot followed the intended pathways thanks to the joint space controller's ability to precisely coordinate joint movements when used in conjunction with the PID approach. Control was made intuitive by directly defining the robot's end position through the use of inverse kinematics in the Cartesian space controller.

Key performance metrics such overshoot, settling time, and steady-state error were used to assess the controllers. The combined space and Cartesian space controllers both performed admirably, which was encouraging. They demonstrated their capacity to precisely and reliably manage the robot by producing outputs with low steady-state error, slight overshooting, and quick settling times. I also conducted simulations to assess the robustness of the system and see how it would function in various physical scenarios. The controllers worked flawlessly even after I doubled the weight and length of the robot's second arm. They demonstrated their ability to adjust to variations in the dynamics of the robot while preserving low overshoot, fast settling times, and 0% steady-state error.

All things considered, our proposed control system demonstrated its flexibility and dependability in both routine and difficult situations. Our robotic system is flexible and resilient, suitable for a variety of applications thanks to its joint space and Cartesian space control methods, as well as its efficient application of inverse kinematics and PID controllers.