ME446 Lab 2: Reference trajectory following using PD and PID Joint control

Kevin Gim and Jasvir Virdi

March 29, 2019

Abstract

In this report, we demonstrate the result of position control of a CRS robot arm by implementing different controllers. The dynamic model has been established in MATLAB Simulink environment and PD controller is simulated with the simulation model. The position control in the joint space is done using PD, PID and Feedforward controller on the actual CRS robot arm. Lastly, the end-effector trajectory tracking is conducted to demonstrate trajectory tracking performance of the controller.

1 Introduction

Figure 1: CAD Drawing of CRS Robot Arm with an End-Effector

A CRS Robot arm is a 3 DOF manipulator comprising of 3 revolute joints. The objective of the lab is to perform position control of the robot arm for given trajectory. In order to make the CRS robot arm follow desired trajectory, it is important to understand the dynamics of robot arm. The dynamic equation of a 2 DOF planar robot arm is derived analytically using Lagrangian approach. In order to avoid complexity in dynamics equation, the dynamic analysis only accounts for the second and third joint. A complete dynamic analysis of a 3 DOF spatial manipulator is

also conducted using symbolic tool in MATLAB.
The robot arm has optical encoders to measure the angular position of each motor. The raw velocity of each joint is calculated by taking derivative of discrete position measurement which leads to error due to high noise in position measurements. Hence, proper filtering method needs to be applied to filter out high frequency noise from the raw velocity. These filters mainly act as low pass filters which filter out the high frequency noise and give out the underlying main signal as the output. In this case, Finite Average Filter and Infinite Average Filter are used and their performances are compared .
The derived 2 DOF dynamic model is implemented in Simulink to simulate PD position controller. A square wave step function is used as a reference trajecory. By tuning control gains of PD controller with the simulation model, we were able to find approximate control gain without directly testing on the actual hardware. The simulated gain is tuned to satisfy given requirement of $300ms$ rise time and 1% overshoot.
Position control of the actual hardware is also conducted using different control scheme. First, we start from PD control using the gain obtained from the controller simulation. Then an integral term is added to eliminate steady-state error from the system response. A Feedforward control is also tested for tracking a polynomial trajectory which is smooth. The results of each controllers are demonstrated.
Lastly, position control in end-effector space is conducted with a time varying "fun" trajectory. In order to perform end-effector position control, corresponding joint positions are derived by solving inverse kinematics on the generated trajectory.

2 Dynamic Model

2.1 2 DOF Planar Manipulator

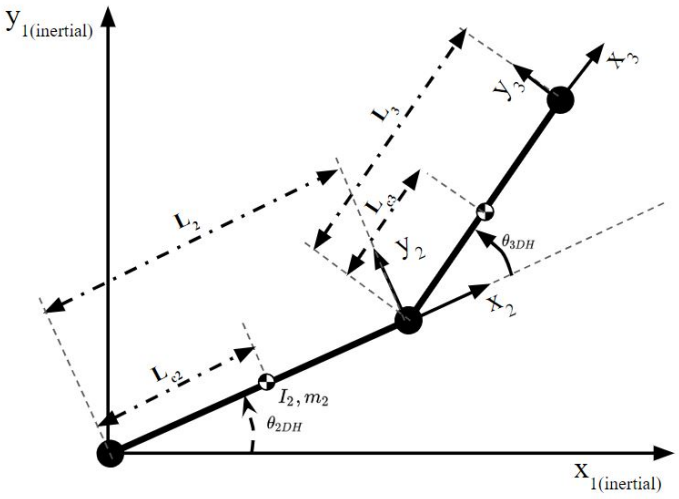

Figure 2: Two Link Planar Arm

The dynamic model developed below is for a 2 DOF planar manipulator with revolute joints as shown in Figure 2. In this analysis, the link 1 and its related dynamics are ignored to avoid complexity of the dynamic equation. Thus, we only account for the dynamics of two joints in a

same plane, i.e, θ_{2DH} and θ_{3DH} for links 2 and 3 respectively. The dynamic model is derived by solving the Eular-Lagrangian equation.

Firstly, we start with writing the Rotation matrices, Jacobians and some other preliminaries for each of the 2 links:

$$R_2^1 = \begin{bmatrix} \cos(\theta_{2DH}) & -\sin(\theta_{2DH}) & 0 \\ \sin(\theta_{2DH}) & \cos(\theta_{2DH}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \; R_3^1 = \begin{bmatrix} \cos(\theta_{2DH}+\theta_{3DH}) & -\sin(\theta_{2DH}+\theta_{3DH}) & 0 \\ \sin(\theta_{2DH}+\theta_{3DH}) & \cos(\theta_{2DH}+\theta_{3DH}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$J_{vc2} = \begin{bmatrix} -l_{c2}\sin(\theta_{2DH}) & 0 \\ l_{c2}\cos(\theta_{2DH}) & 0 \\ 0 & 0 \end{bmatrix} \; J_{wc2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \; q = \begin{bmatrix} \theta_{2DH} \\ \theta_{3DH} \end{bmatrix} \; \dot{q} = \begin{bmatrix} \dot{\theta_{2DH}} \\ \dot{\theta_{3DH}} \end{bmatrix}$$

$$J_{vc3} = \begin{bmatrix} -l_2\sin(\theta_{2DH}) - l_{c3}\sin(\theta_{2DH}+\theta_{3DH}) & -l_{c3}\sin(\theta_{2DH}+\theta_{3DH}) \\ l_2\cos(\theta_{2DH}) + l_{c3}\cos(\theta_{2DH}+\theta_{3DH}) & l_{c3}\cos(\theta_{2DH}+\theta_{3DH}) \\ 0 & 0 \end{bmatrix} \; J_{wc3} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$I_2 = \begin{bmatrix} I_{2xx} & 0 & 0 \\ 0 & I_{2yy} & 0 \\ 0 & 0 & I_{2zz} \end{bmatrix} \; I_3 = \begin{bmatrix} I_{3xx} & 0 & 0 \\ 0 & I_{3yy} & 0 \\ 0 & 0 & I_{3zz} \end{bmatrix}$$

After defining all the main parameters for the 2 links, we are in a position to state the total Kinetic Energy of the system which is sum of translational and rotational components. In this case, it is defined as follows:

$$K = \tfrac{1}{2}\dot{q}^T(m_2 J_{v,c2}^T J_{v,c2} + m_3 J_{v,c3}^T J_{v,c3} + J_{w,c2}^T R_2^1 I_2 {R_2^1}^T J_{w,c2} + J_{w,c3}^T R_3^1 I_3 {R_3^1}^T J_{w,c3})\dot{q}$$

Next, we write the potential energy for both the links. We take the reference as X_1 axis for defining potential energy.

$$V = m_2 g l_{c2}\sin(\theta_{2DH}) + m_3 g(l_2\sin(\theta_{2DH}) + l_{c3} sin(\theta_{2DH}+\theta_{3DH}))$$

Now, we will simplify the above expressions and bring all our parameters like mass, link lengths, inertia etc down to 5 main parameters. These are listed as follows:

$$\begin{aligned} p_1 &= m_2 l_{c2}^2 + m_3 l_2^2 + I_{2zz} \\ p_2 &= m_3 l_{c3}^2 + I_{3zz} \\ p_3 &= m_2 l_2 l_{c3} \\ p_4 &= m_2 l_{c2} + m_3 l_2 \\ p_5 &= m_3 l_{c3} \end{aligned}$$

Hence, after solving these equations in MATLAB, we get the following simplified expressions for kinetic and potential energy:

$$\begin{aligned} K &= \dot{\theta}_{3DH}^2(\frac{1}{2}p_2) + \dot{\theta}_{2DH}^2(\frac{1}{2}p_1 + \frac{1}{2}p_2 + p_3\cos(\theta_{3DH})) + \dot{\theta}_{2DH}\dot{\theta}_{3DH}(p_2 + p_3\cos(\theta_{3DH})) \\ V &= p_4 g\sin(\theta_{2DH}) + p_5 g\sin(\theta_{2DH}+\theta_{3DH}) \end{aligned}$$

Now, we derive the dynamic's equations from Lagrangian.The lagrangian is defined as:

$$L = K - V$$

The equation for dynamics is defined as:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i, \text{ where } i = 1, 2$$

On substituting, the respective expressions for K and V in the lagrangian equation and taking its partial derivatives, we get:

$$\begin{aligned}
\frac{\partial L}{\partial q_1} =& - gp_4 \cos(\theta_{2DH}) - gp_5 \cos(\theta_{2DH} + \theta_{3DH}) \\
\frac{\partial L}{\partial q_2} =& - p_3 \dot{\theta}_{2DH}^2 \sin(\theta_{2DH}) - p_3 \dot{\theta}_{2DH} \dot{\theta}_{3DH} \sin(\theta_{3DH}) - p_5 g \cos(\theta_{2DH} + \theta_{3DH}) \\
\frac{\partial L}{\partial \dot{q}_1} =& \dot{\theta}_{3DH}(p_2 + p_3 \cos(\theta_{3DH})) + \dot{\theta}_{2DH}(p_1 + p_2 + 2p_3 \cos(\theta_{3DH})) \\
\frac{\partial L}{\partial \dot{q}_2} =& p_2 \dot{\theta}_{3DH} + \dot{\theta}_{2DH}(p_2 + p_3 \cos(\theta_{3DH})) \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_1} =& \ddot{\theta}_{2DH}(p_1 + p_2 + 2p_3 \cos(\theta_{3DH})) + \ddot{\theta}_{3DH}(p_2 + p_3 \cos(\theta_{3DH})) \\
& - \dot{\theta}_{2DH} \dot{\theta}_{3DH}(2p_3 \sin(\theta_{3DH})) - (\dot{\theta}_{3DH})^2 (p_3 \sin(\theta_{3DH})) \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_2} =& \ddot{\theta}_{3DH}(p_2) + \ddot{\theta}_{2DH}(p_2 + p_3 \cos(\theta_{3DH})) - \dot{\theta}_{2DH} \dot{\theta}_{3DH}(p_3 \sin(\theta_{3DH}))
\end{aligned}$$

We now arrange, all these equations in a matrix form and get the following simplified expression:

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau$$

where

$$\begin{aligned}
\tau &= \begin{bmatrix} \tau_{2DH} \\ \tau_{3DH} \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_{2DH} \\ \theta_{3DH} \end{bmatrix} \\
D(\theta) &= \begin{bmatrix} p_1 + p_2 + 2p_3 \cos(\theta_{3DH}) & p_2 + p_3 \cos(\theta_{3DH}) \\ p_2 + p_3 \cos(\theta_{3DH}) & p_2 \end{bmatrix} \\
C(\theta, \dot{\theta}) &= \begin{bmatrix} -p_3 \sin(\theta_{3DH})\dot{\theta}_{3DH} & -p_3 \sin(\theta_{3DH})\dot{\theta}_{3DH} - p_3 \sin(\theta_{3DH})\dot{\theta}_{2DH} \\ p_3 \sin(\theta_{3DH})\dot{\theta}_{2DH} & 0 \end{bmatrix} \\
G(\theta) &= \begin{bmatrix} p_4 g \cos(\theta_{2DH}) + p_5 g cos(\theta_{2DH} + \theta_{3DH}) \\ p_5 g \cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix}
\end{aligned}$$

Now, we substitute all the DH angles in terms of motor angles in the above equations. Further, we convert the DH torques at each joint into motor torques.

$$\begin{aligned}
\theta_{2DH} &= \theta_{2M} - \frac{\pi}{2}, \qquad \theta_{3DH} = \theta_{3M} - \theta_{2M} + \frac{\pi}{2} \\
\tau_{2DH} &= \tau_{2M} + \tau_{3M}, \qquad \tau_{3DH} = \tau_{3M}
\end{aligned}$$

Substituting these four equations, along with the derivative and second derivative of θ_{2DH} and θ_{3DH},into the DH equations of motion, the new equations of motion become as follows:

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau$$

where

$$\tau = \begin{bmatrix} \tau_{2M} \\ \tau_{3M} \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_{2M} \\ \theta_{3M} \end{bmatrix}$$

$$D(\theta) = \begin{bmatrix} p_1 & -p_3 \sin(\theta_{3M} - \theta_{2M}) \\ -p_3 \sin(\theta_{3M} - \theta_{2M}) & p_2 \end{bmatrix}$$

$$C(\theta, \dot{\theta}) = \begin{bmatrix} 0 & -p_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{3M} \\ p_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{2M} & 0 \end{bmatrix}$$

$$G(\theta) = \begin{bmatrix} p_4 g \sin(\theta_{2M}) \\ p_5 g \cos(\theta_{3M}) \end{bmatrix}$$

Finally, we can express the dynamic equation in state space form as follows:

$$\begin{bmatrix} \ddot{\theta}_{2M} \\ \ddot{\theta}_{3M} \end{bmatrix} = D(\theta)^{-1}\tau - D(\theta)^{-1}C(\theta, \dot{\theta})\dot{\theta} - D(\theta)^{-1}G(\theta)$$

$$x_1 = \theta_{2M}, \quad x_2 = \dot{\theta}_{2M}, \quad x_3 = \theta_{3M}, \quad x_4 = \dot{\theta}_{3M}$$

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \ddot{\theta}_{2M}$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \ddot{\theta}_{3M}$$

2.2 3 DOF Spatial Manipulator

The dynamic equation of 3 DOF Spatial Manipulator is also established by accounting the ignored first joint in the previous analysis. By adding the yaw rotation joint, the model is able to describe the full dynamics of the CRS Robot arm. However, the existence of the first joint increase the complexity of the dynamic equation significantly so that it is not easy to be solved without a computational tool. Therefore, MATLAB symbolic tool is utilized to derive the dynamic equation of the 3 DOF Spatial Manipulator by computing the Euler-Lagrange equation. The MATLAB Scripts are attached in the Appendix. Due to the massively long expression of the dynamic equation, the derived dynamic equation is also presented in Appendix as well.

3 Simulation

Figure 3 displays the simulation model for the dynamics of 2 DOF planar manipulator with PD joint position controller in Simulink. The subsystem labeled 'Dynamic model' contains the derived dynamic model that takes the joint position, velocity and torque as inputs and computes joint acceleration based on the dynamic equation. By integrating the computed joint acceleration, joint position and torque are obtained and these values are recycled to the dynamic model. The script of subsystem is attached in Appendix. Using the dynamic simulation model, we attached a feedback controller module for PD joint position control of the robot.

The controller simulation model comprises of 3 main modules:

1. the robot dynamic's model which computes $\ddot{q}$
2. the infinite average filter which computes velocities based on the motor angles
3. the PD controller module for computing input torques based on current and desired motor angles

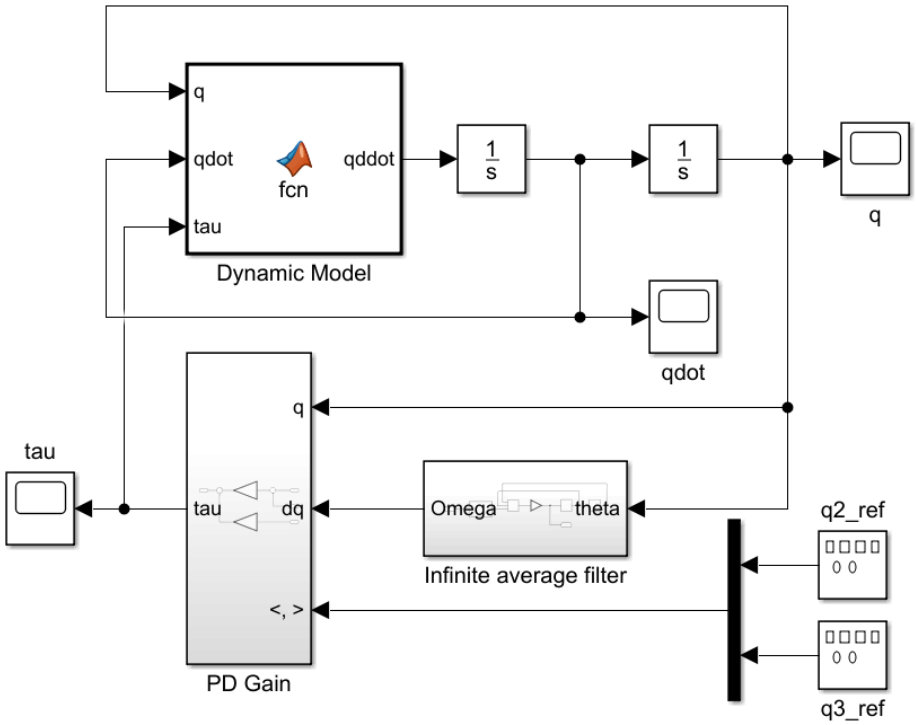

Figure 3: Simulink model for Dynamics of the 2 DOF Planar Manipulator

3.1 Velocity Noise filter

In the dynamic model, joint velocities are obtained by integrating the joint acceleration. However, the real robot doesn't have a velocity sensor or an acceleration sensor but only have an optical encoder on each joint as an angular position sensor. Since the robot only has the angular position sensor, the joint velocities are computed using the angular position values. As the sampling time of the code runs at constant interval, the approximated velocity can be simply calculated using the following equation:

$$\dot{\theta} = \frac{\theta_{current} - \theta_{previous}}{T}$$

However, this raw velocity is calculated from discrete two value which has the time interval between the measurement. Therefore, there is undesirable high frequency noise in the obtained raw velocity. Since the calculated velocity will be used in feedback controller, the noisy input signal will consequent noisy toque output as well. In order to alleviate the noise from the signal, two different filtering methods are proposed, Finite average filter and Infinite average filter.

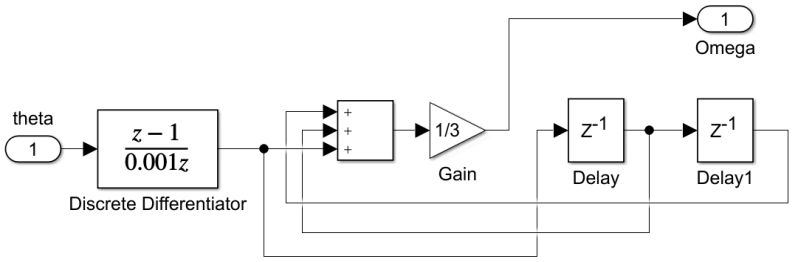

Figure 4: Simulink Subsystem of the Finite Average Filter

Figure 4 shows a Simulink subsystem of the first proposed method, Finite Average Filter. The main idea of the filter is straight forward that it is averaging sets of old value with current value to reduce noise signals. The Finite Average Filter has useful properties that it is inherently stable and requires no feedback. In the proposed filter, it takes angular position measurement and calculates the raw velocity by discrete differentiation of the value with $1ms$ interval, which is the loop time of the robot control system. Then the raw velocity is averaged with two prior raw velocity values to get the filtered velocity.

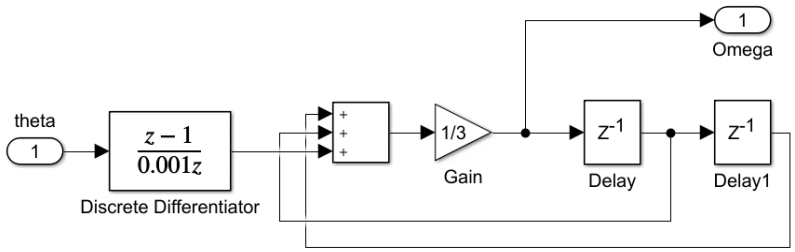

Figure 5: Simulink Subsystem of the Infinite Average Filter

The second proposed filter, Infinite Average Filter looks very similar with the Finite Average Filter as shown in Figure 5. However, instead using sets of raw velocity for averaging, two filtered values are stored and averaged with one raw value. Since the filtered value is recycled to average new value, the Infinite Average Filter contains feedback. Although the Infinite Average Filter is considered more complicated than Finite Average Filter, it has the advantage of efficient implementation by having a fewer number of calculations per time step.

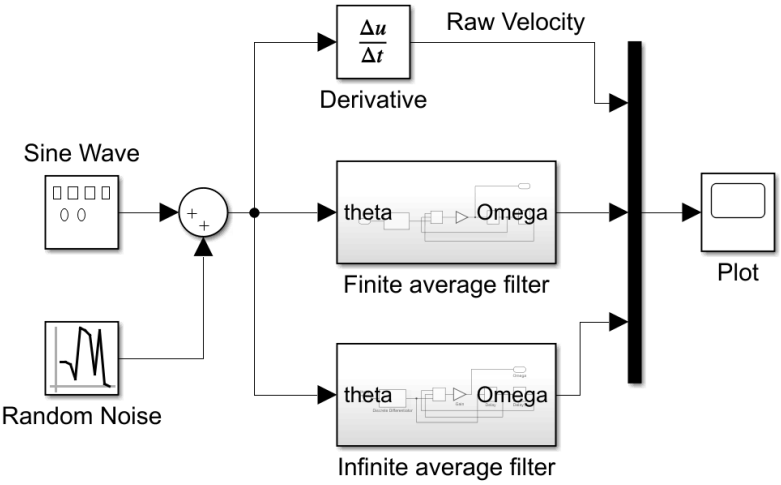

Figure 6: Simulink model for Filter Performance Comparison

Figure 6 and 7 present the Simulink model and result of performance comparison of the two filters. A reference raw signal is generated by adding random noise to sine wave. The resulting plot show that both Finite Average Filter and Infinite Average Filter reduces noise from the raw signal. Although two filtered signal seems quite similar, Infinite Average Filter result shows slightly better performance having less delay and noise amplitude. Moreover, we concluded that the Infinite Average Filter fits more to utilize for the actual robot due to limited computing capability of the DSP of the robot control processor.

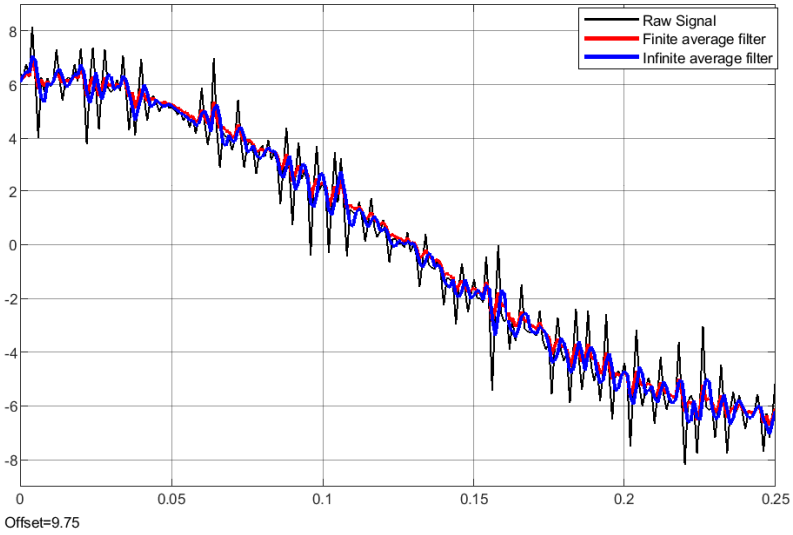

Figure 7: Filter Simulation result

The C code implementation of two filters are included in Appendix. In C code, it is very critical to have a proper order for updating Omega1_old2 and Omega1_old1. As shown in the attached C Code, Omega1_old2 must be updated prior to updating Omega1_old1 in order to keep storing two old values. If Omega1_old1 is updated with Omega1 before updating Omega1_old2, Omega1_old2 will takes updated Omega1_old1 value i.e, Omega1 and then the old values are flushed away. Therefore, it is important to put the first old value to Omega1_old2 and update the new value to Omega1_old1.

3.2 PD Controller Module

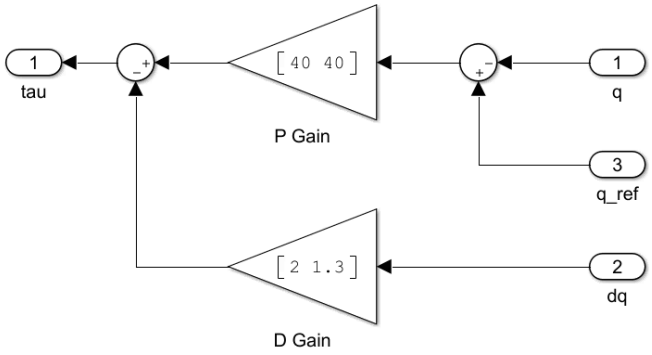

Figure 8: Simulink Subsystem of the PD Controller

As shown in Figure 8, a PD controller is implemented to perform joint position control of the 2 DOF planar manipulator. Gain tuning using physical hardware is a laborious job but also has potential safety issues that the robot or environment may be damaged from trials with inappropriate gains. Therefore, it is important to find approximate range and starting point before making trails with the physical system. By using the simulation model to tune the PD gains, we can find approximate PD gains for actual robot more efficiently and safely without operating the hardware.

The PD controller module computes motor torque by taking motor angular position and velocity value and desired position as shown in following equation:

$$\tau_{2M} = K_{p2}(\theta_{2ref} - \theta_{2M}) - K_{d2}\dot{\theta}_{2M}$$
$$\tau_{3M} = K_{p3}(\theta_{3ref} - \theta_{3M}) - K_{d3}\dot{\theta}_{3M}$$

In the PD controller, the output torque is computed by two components, proportional term and derivative term. The proportional term produces an output value with the position error which is difference between desired position and current position multiplied with the proportional gain K_p. By increasing K_p value, the system is able to reject the error more rapidly and sensitively.However large gain also increases overshoot and if it is too high, the system become unstable. Also, the proportional control is not able to completely eliminate a steady-state error. Therefore, choosing proper K_p gain has the highest priority for gain tuning of the system. The other term is Derivative term which is also considered as a damping term. The Derivative term produces the output torque with the velocity error multiplied with the derivative gain K_d. However, the reference velocities are set to zero because a step square wave is used for the reference signal that has either zero or infinite velocity. The derivative term reduces overshoot as a damper but not able to deal with a constant error.

3.3 Simulation result

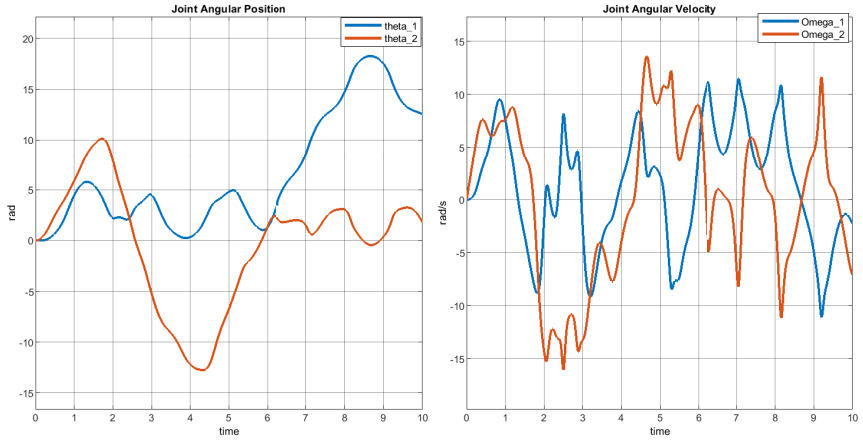

Figure 9: Position and Velocity Result of Zero Joint Torques Simulation

Figure 9 presents the simulation result of the dynamic model with zero joint torque input to verify the dynamic model. Note that the model is open-loop system so that the feedback controller is not activated and the dynamic model receives constant zero input for this specific case. The resulting plot show that the links are falling free and then swinging with enormous velocity which is expected since there is no friction or external force other than gravity.

Now, the PD feedback controller is introduced to the dynamic model conduct joint position control. Each joint takes step function as a desired trajectory that has $\frac{\pi}{6}$ amplitude and $2sec$ period. Figure 10 presents the result of the first trial using suggested initial the control gain K_{p2} and K_{p3} equal to 1 and K_{d2} and K_{d3} equal to zero. The result shows that the controlled system is not able to track the reference trajectory well due to the small proportional gains.

Through several iterations, proper PD gains has been found to have the required system response with a rise time less than $300ms$ and an overshoot less than 10%. The simulation result

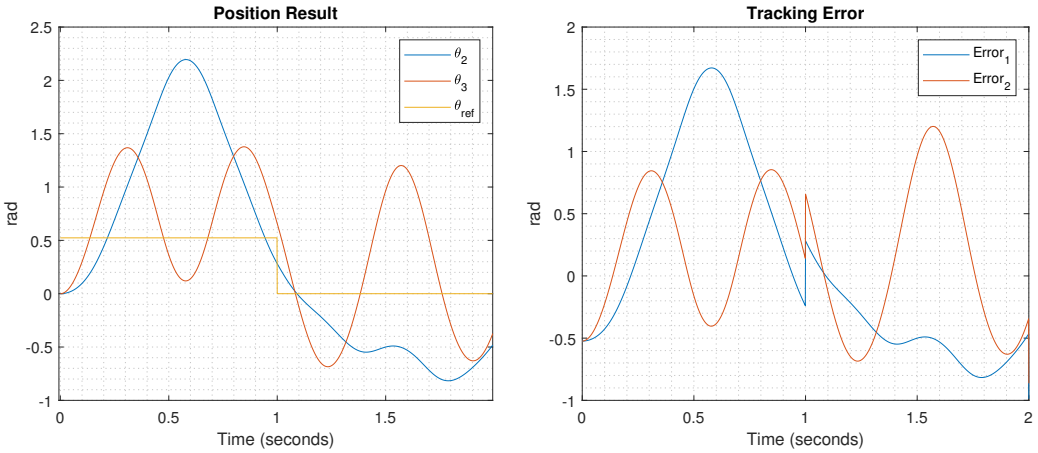

Figure 10: PD Control Simulation Result with Initial Gain
$K_{p2} = 1, K_{p3} = 1, K_{d2} = 0, K_{d3} = 0$

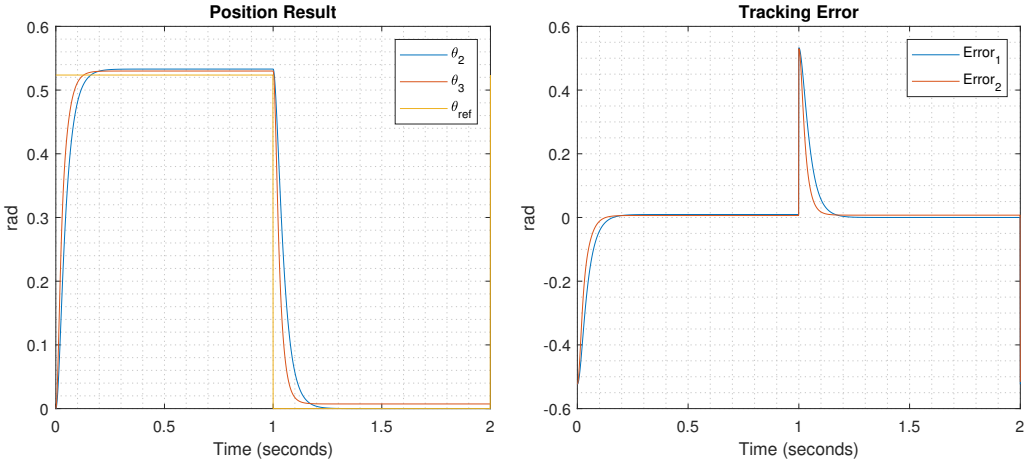

Figure 11: PD Control Simulation Result with Tuned Gain
$K_{p2} = 40, K_{p3} = 40, K_{d2} = 2, K_{d3} = 1.3$

of the tuned gain is shown in Figure 11. K_{p2} and K_{p3} are selected to 40 and K_{d2} and $_K d_3$ values equal to 2, and 1.3 respectively.

Figure 12 provides closer look of the joint position result with PD controller. It is verified that the system response has an overshoot of 0.95% and a 90%rise time of $60ms$ which satisfy the required system specification.

4 Hardware Implementation

After creating the simulation of the robot arm's dynamics in Simulink, we shift our focus on implementing different kinds of sensor based feedback control laws on the real robot arm. All these control laws are governing the system input in the form of motor torques to track the desired reference trajectory. In almost all of the cases, we first test the control law in Simulink, tune the gains until we get the desired response and then use these values on the real robot arm. Again, it is necessary to fine tune these gains on the real robot as we have not considered non linearities like friction for our simulation. In order to easily distinguish reference input of each joint, the

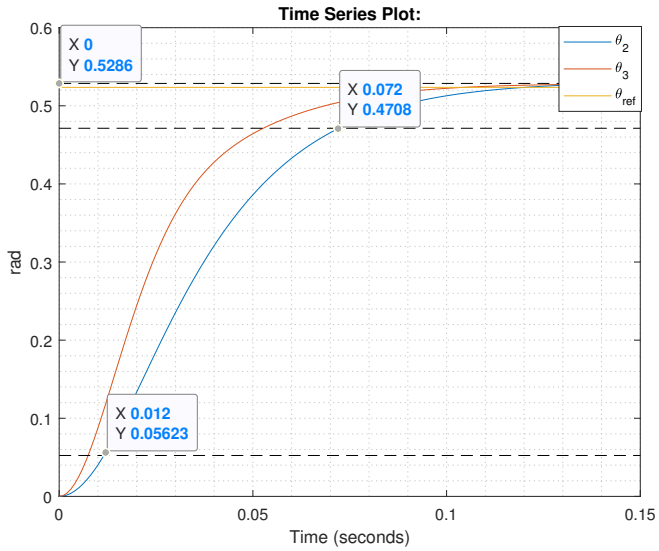

Figure 12: Closer look or PD Position Control Result with Tuned Gain

amplitude of the reference step signal are given to $0.3rad$, $0.4rad$, and $0.5rad$ respectively to the first, second and third motor.

4.1 PD Control

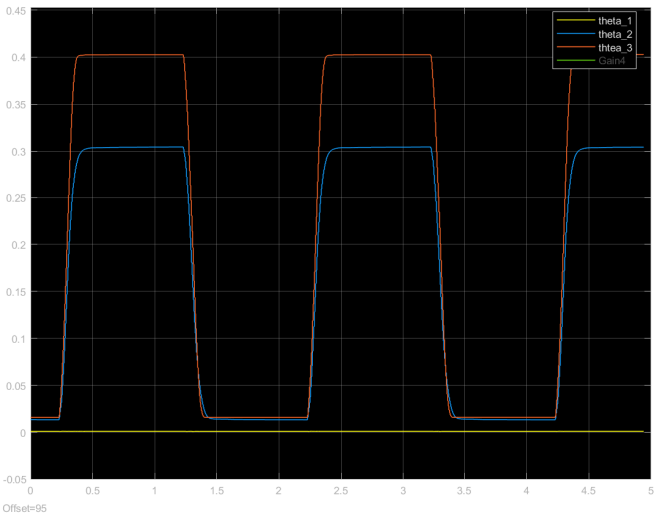

Figure 13: PD Position Control Result of θ_1, and θ_2 with Tuned Gain
$K_{p2} = 40,\ K_{p3} = 40,\ K_{d2} = 2,\ K_{d3} = 1.3$

In order to design PD controller for the CRS robot arm, we first find reasonable estimates of proportional and derivative gains for links 2 and 3 using the model created in Simulink. The reference trajectory is given as a square wave for both θ_{2M} and θ_{3M} with the same frequency but different amplitude. The control law governing the motor torque inputs is given as follows:

$$\tau = K_p(\theta_{des} - \theta_M) - K_d\dot{\theta}_M$$

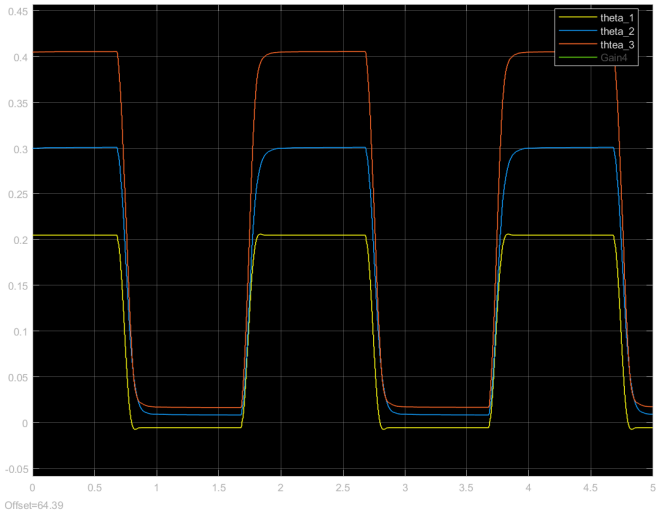

Figure 14: PD Control Result of θ_1, θ_2, and θ_3 with Simulated Gain
$K_{p1} = 50,\ K_{p2} = 40,\ K_{p3} = 40,\ K_{d1} = 2.1,\ K_{d2} = 2.1,\ K_{d3} = 1.5$

Here we consider $\dot{\theta}_{des} = 0$ for a step input, as the derivatives become infinte theoretically thereby making the the torque input values extremely large which could lead to damage of the motors. Keeping the gain values, as shown in Figure 13, we get a nice system response where the overshoot is kept to minimum and the system is quick to respond. However, the downside is that there is some steady state error observed for both the motor angles. Now, we use these gains on the real robot arm and keep the proportional and derivative gain values for link 1 to be of the same order as link 2 and 3. Figure 14 shows the actual system's response to a step function for each of the motor angles θ_{1M}, θ_{2M} and θ_{3M} with varying amplitude but same frequency. Here, the system again has a decent response time but the steady state error still exists.

4.2 PID Control

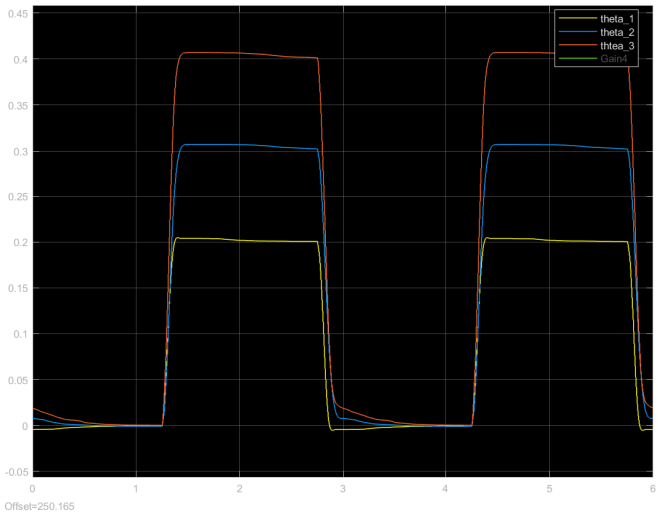

Figure 15: PID Control Result of the Joint Position with Simulated Gain
$K_{p1} = 51,\ K_{p2} = 40,\ K_{p3} = 50,\ K_{d1} = 2.5,\ K_{d2} = 2,\ K_{d3} = 1.3,$
K_I_1 = 0.05, K_I_2 = 0.05, K_I_3 = 0.05

In order to remove the shortcomings observed with implementing PD control, we shift to a new control law where an integral term is introduced. The main purpose of introducing this term is to eliminate the steady state error that was seen in applying just a PD control. The motor torque inputs are given as:

$$\tau = K_p(\theta_{des} - \theta_M) - K_d\dot{\theta}_M + K_I \sum(\theta_{des} - \theta_M)\Delta t$$

Here again, we keep $\dot{\theta}_{des} = 0$ for a step input for similar reasons as suggested above. As we can clearly see, the steady state error is reduced especially when the reference step input comes down towards 0. In this case as well, approximate values of gains were first obtained in Simulink and then these gain values were tuned on the actual robot. However, an important thing to remember while tuning the integral gain is that if it's values are kept high, it could result in the the overshoot of response coupled with increasing settling times. Hence, we need to be careful while tuning the integral gain. Since the integral term is accumulate error, it may consequent massive torque output especially for a step input and causing malfunction of the physical hardware. In order to prevent massive control output from the PID controller, we activate the integral term only if the error value is smaller than a preset threshold. Hence, the controller can avoid massive accumulation of error.

4.3 Feedforward Control

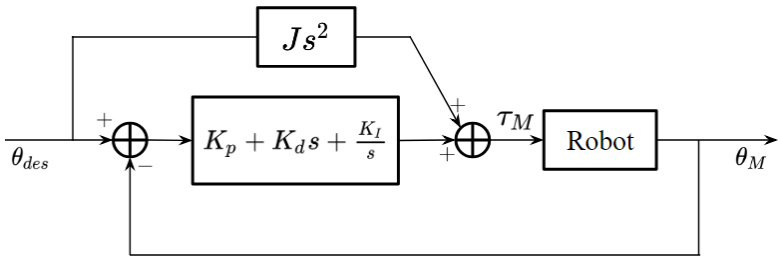

Figure 16: Feedforward Controller Block Diagram

In previous tasks, a square wave is used as a reference trajectory. Since the square wave has infinite or zero velocity, it is not possible to utilize reference velocity for control input. Moreover, the reference velocity is not differentiable to obtain the reference acceleration. Therefore, a polynomial trajectory is generated for introducing a feedforward control instead of the square wave trajectory. The requirements for the cubic polynomial trajectory are given by the lab manual as following:

$$\theta_{des}(t) = at^3 + bt^2 + ct + d$$

where

$$\begin{aligned} &\theta_{des}(0) = 0 \quad \theta_{des}(1) = 0.5 \quad \theta_{des}(2) = 0 \\ &\dot{\theta}_{des}(0) = 0 \quad \dot{\theta}_{des}(1) = 0 \quad \dot{\theta}_{des}(2) = 0 \end{aligned}$$

By plugging in the points to the equation, the coefficient of the cubic equation can be identified. The generated cubic trajectory is shown in Figure 17.

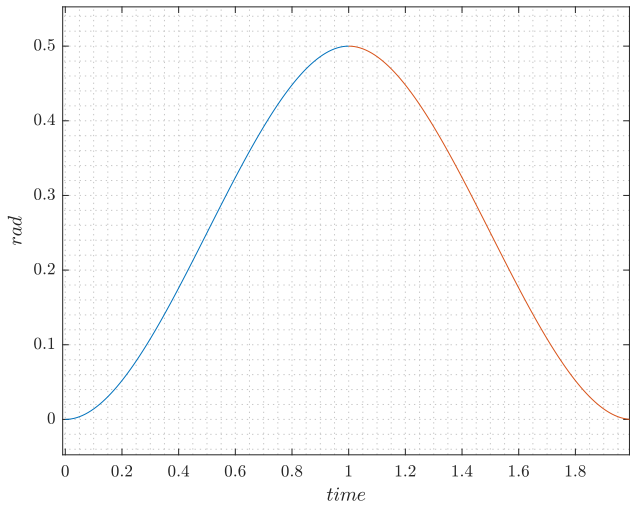

Figure 17: Reference Cubic Trajectory

Now, we introduce a feedforward controller to the PID controller. Feedforward control is a method to track time varying trajectories and reject time dependent disturbances such as inertial effect or viscous friction which are proportional to acceleration and velocity respectively. In the given task, only the inertial effect of each joint are rejected using the second time derivative of the reference trajectory. Figure 16 presents the block diagram of the PID controller with the feedforward compensator to reject inertial effect.

With the controllers, the resulting motor torque is computed as following:

$$\tau = K_p(\theta_{des} - \theta_M) + K_d(\dot{\theta}_{des} - \dot{\theta}_M) + K_I \sum(\theta_{des} - \theta_M)\Delta t + J\ddot{\theta}_{des}$$

The joint torque equation has similar expression with the torque from the PID controller but there are several differences. The second derivative of the reference trajectory is multiplied by the J value to reject the inertial effect. The J values are given by the lab manual, as $J_1 = 0.0167, J_2 = 0.03, and J_3 = 0.0128$. Since the polynomial trajectory is differentiable, the reference velocity is no longer considered as zero.

It was very challenging task to eliminate the position error while tracking the square wave trajectory with the PID controller as shown in Figure 15. This was due to sharp edges of the trajectory, discontinuous reference velocity, and corresponding inertial effect. Now, taking the smoothed cubic trajectory, the hardware implement result shown in Figure 18 shows that all three motors are tracking reference trajectory precisely by successfully rejecting the position error.

4.4 Trajectory Tracking

In the previous tasks, we conducted position control of individual joint by tracking the desired joint position trajectory. In addition to the position control in joint space, position control in end-effector space is conducted by following the desired trajectory of the end-effector. The end-effector position control is performed through following steps:

1. Generate desired trajectory as a time varying function to be followed by the end-effector
2. Calculate $\theta_{1des}, \theta_{2des}, \theta_{3des}$ by solving Inverse Kinematics for the point in reference trajectory for the current time step

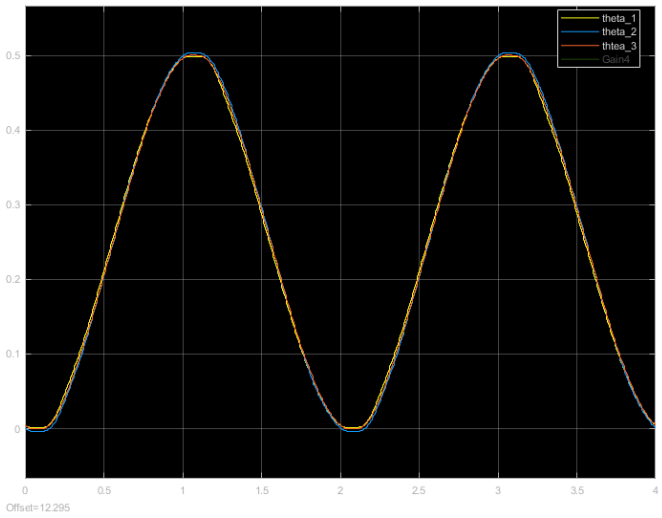

Figure 18: Feedforward Control Result of the Joint position

3. Perform PID position control for $\theta_{1des}, \theta_{2des}, \theta_{3des}$ in the joint space

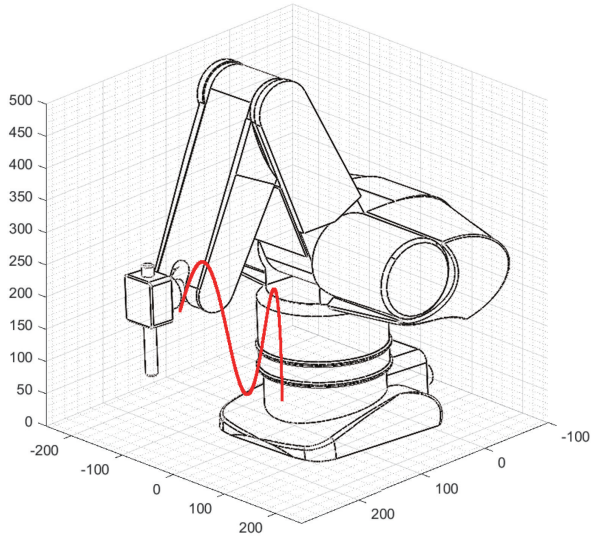

Figure 19: End-effector trajectory and Joint trajectory

Since the controller of the CRS robot arm runs with the constant loop time of $1ms$, the joint position values are updated every time step by solving inverse kinematics for the position in the reference trajectory at the time step. Then the PID controller takes the calculated joint position values for the control input to produce output torque for the each motor.

We decide to make a reference trajectory to draw a sin wave with $90mm$ amplitude on the yz plane maintaining x position to $280mm$ with reducing period dependent to y displacement. The

equation of the end-effector position with respect to the base frame can be expressed as following:

$$x = 280$$
$$y = 200 \sin\left(\frac{2\pi t}{10}\right)$$
$$z = 250 + 90 \sin\left(\frac{2\pi}{1.5}\right)$$

Figure 20 shows the reference end-effector trajectory and the joint trajectory. It is confirmed from the joint trajectory plot that the desired angle of each joint never exceed the range limit. The Reference End-effector trajectory also shown in Figure 19 with the CAD model of the robot arm.

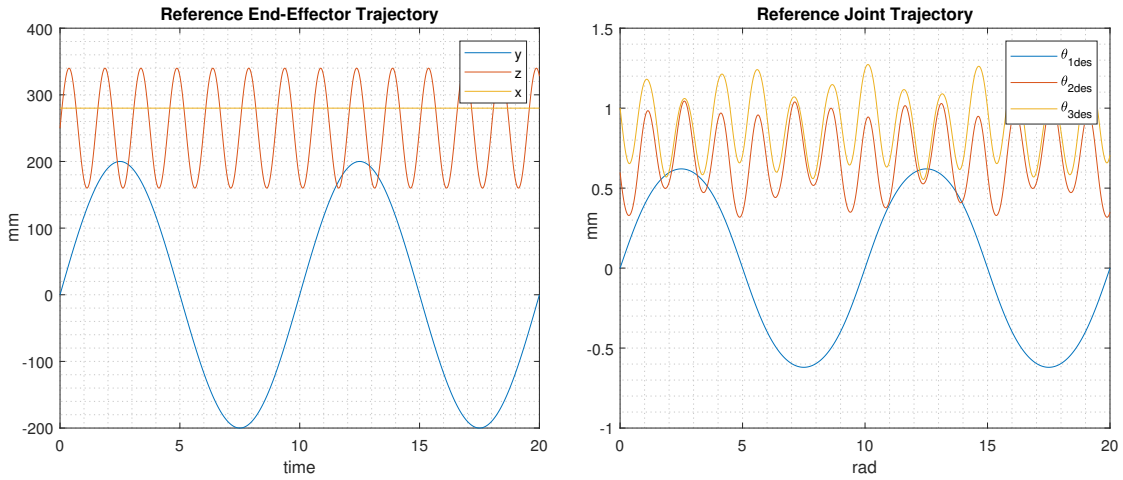

Figure 20: End-effector trajectory and Joint trajectory

The reference desired joint angles are now converted to joint torque by the PID controller. Since the derivative of the inverse kinematic solution is difficult to be obtained analytically, the PID controller is utilized to control the motor although the Feedforward controller presented better performance than the PID controller by rejecting the influence of the inertia. The same PID gain is applied which are tuned in the previous PID controller experiment. Using Forward Kinematics the joint position result is converted to End-Effector position to verify the trajectory tracking performance. The result is presented in Figure 21

Figure 22 also demonstrate a snapshot of the trajectory tracking experiment.

5 Discussion

In this report, we have conducted position control of a 3-axis robot arm with PD, PID, and PID with Feedforward controller. The PD controller showed good performance to track the given trajectory yet it can't eliminate steady-state error due to the inherent characteristic of the controller. By adding the integral term, the PID controller can pile up error to completely reject error from steady-state as well. Since the integral term can cause massive error accumulation when there is a drastic change in the reference signal, the threshold has been set for activating the integral term only for small error value. Although there is a possibility of being unstable due to the integral term, having proper gain and threshold improve the system response but also eliminate

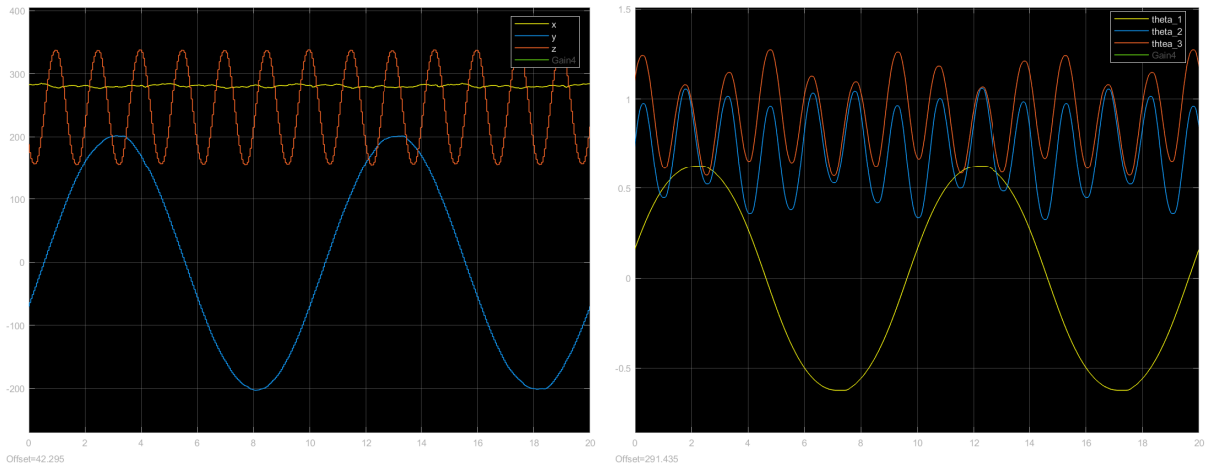

Figure 21: End-Effector Position and Joint Position Result

steady-state error. Now, the Feedforward controller is introduced to compensate inertial effect of the robot arm. In order to conduct the Feedforward control, a reference trajectory needs to have the second derivative. Therefore, the cubic polynomial desired trajectory has been generated and the effective inertia of the links are multiplied to the second derivative of reference trajectory and added to the control input. The result presents that the controller demonstrates the most successful error rejection performance. According to the End-Effector position control result, it is clear that the End-Effector of the robot arm is following given trajectory successfully. However, there is some error observed in the End-Effector position result, especially in x position result.To improve the control performance we can add a Feedforward controller. However, there is an obstacle that it is very complicated to compute the time derivative of the inverse kinematics solutions in an analytical way. If there is enough computing power of the DSP, it is possible to numerically compute the time derivative and utilize it to compensate inertial effect.

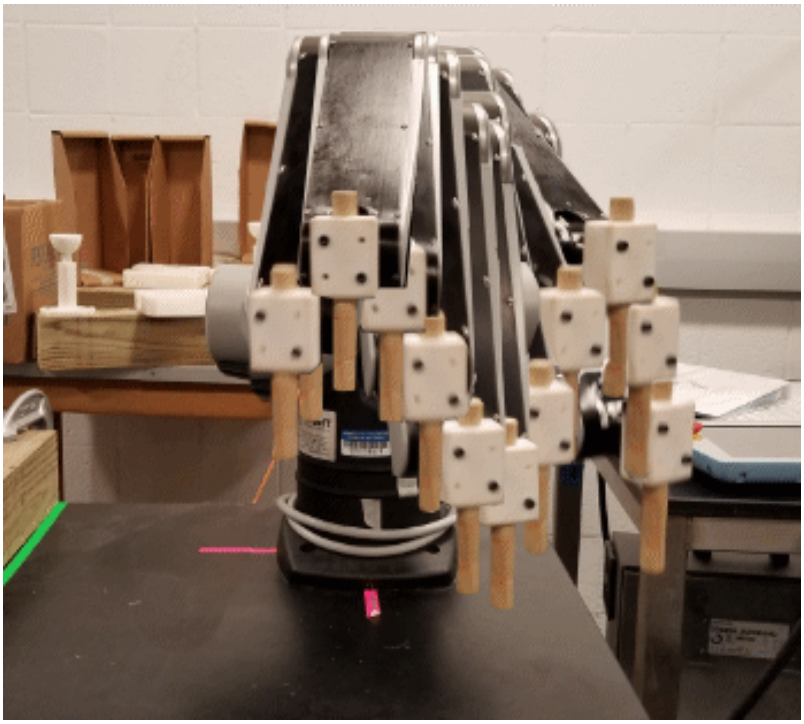

Figure 22: Snapshot of the Trajectory Tracking Implementation

6 Appendix

Dynamic Model of 2 DOF Planar Manipulator

```
function qddot = fcn(q, qdot, tau)

    g = 9.8;
    p = [ 0.0300; 0.0128; 0.0076; 0.0753; 0.0298];

    D = [p(1),     p(3)*sin(q(2) q(1));
         p(3)*sin(q(2) q(1)),  p(2)];
    C = [0,  p(3)*cos(q(2) q(1))*qdot(2);
        p(3)*cos(q(2) q(1))*qdot(1), 0];
    G = [ p(4)*g*sin(q(1));
         p(5)*g*cos(q(2))];

    qddot = D\(tau   C*qdot   G) ;
```

Cubic Polymonial Trajectory Generation Script

```
% ME446 Report 2
% Cubic Trajectory Generation

t1 = linspace(0,1,100);
t2= linspace(1,2,100);

A_1 = [1, 1, 1, 1;
       0, 0, 0, 1;
       3, 2, 1, 0;
       0, 0, 1, 0];
```

```
11
12  B_1 = [0.5; 0; 0; 0];
13
14  coff_1 = inv(A_1)*B_1
15
16
17  v1 = polyval(coff_1, t1)
18
19  A_2 = [1, 1, 1, 1;
20         3, 2, 1, 0;
21         8, 4, 2, 1;
22        12, 4, 1, 0];
23  B_2 = [0.5; 0; 0; 0];
24
25  coff_2 = inv(A_2)*B_2
26
27  v2 = polyval(coff_2, t2);
28
29  plot(t1, v1)
30  hold on;
31  grid minor;
32
33  plot(t2, v2)
```

3 DOF Spatial Manipulator Dynamics Script

```
%=========================================================================
% ME446
% Report 2
% Kevin Gim & Jasvir Virdi
% Dynamics of 3 DOF Spatial Manipulator with Lagrangian Method
%=========================================================================
%%
clc
clear all
close all

syms t q1 q2 q3 q1d q2d q3d q1dd q2dd q3dd m1 m2 m3 g l1 l2 l3 fx fy fz nx ny nz...
     Pc1x Pc1y Pc1z Pc2x Pc2y Pc2z Pc3x Pc3y Pc3z Ix1 Iy1 Iz1 Ix2 Iy2 Iz2 Ix3 Iy3 Iz3

startup_rvc

% DH Parameter of Robot Arm
DH = [0,  pi/2, l1, q1;
      l2,  0,   0,  q2;
      l3,  0,   0,  q3;];


[k,l] = size(DH);
m=[m1,m2,m3];
Pc = [Pc1x Pc1y Pc1z; Pc2x Pc2y Pc2z; Pc3x Pc3y Pc3z].';
I = [diag([Ix1 Iy1 Iz1]) diag([Ix2 Iy2 Iz2]) diag([Ix3 Iy3 Iz3])];
grav=[0;0;g];


 % Generate Link structure to define each link
for i= 1:size(DH,1)
     a = DH(i,1);
    al = DH(i,2);
     d = DH(i,3);
    th = DH(i,4);

    L(i) = Link([th, d, a, al,], 'standard'); % Define link using the DH parameters

    L(i).m=m(i); % Link Mass
    L(i).r=Pc(:,i); % Position of COM
    L(i).I=I(:,3*(i-1)+1:3*i);  % Link Inertia

end

robot = SerialLink(L)

% Assume there is no external force and torque other than gravity
fi=[0;0;0]; % External Force
ni=[0;0;0]; % External Torque


% Dynamics of the manipulator with Lagrangian

tau_lag = lagrangian(robot,grav,3,fi,ni).'

% Convert MATLAB Result to Latex
```

```
tau_latex_1 = latex(tau_lag(1))
tau_latex_2 = latex(tau_lag(2))
tau_latex_3 = latex(tau_lag(3))
```

Dynamics Equation of a 3 DOF Spatial Manipulator:

$$
\begin{aligned}
\tau_1 = {} & \mathrm{Iz}_1\,\mathrm{qdd}_1 + \mathrm{Iz}_2\,\mathrm{qdd}_1 + \mathrm{Iz}_2\,\mathrm{qdd}_2 + \mathrm{Iz}_3\,\mathrm{qdd}_1 + \mathrm{Iz}_3\,\mathrm{qdd}_2 + \mathrm{Iz}_3\,\mathrm{qdd}_3 + \frac{{l_2}^2\,m_2\,\mathrm{qdd}_1}{2} + \frac{{l_2}^2\,m_3\,\mathrm{qdd}_1}{2} \\
& + \frac{{l_3}^2\,m_3\,\mathrm{qdd}_1}{2} + \mathrm{Pc1x}^2\,m_1\,\mathrm{qdd}_1 + \frac{\mathrm{Pc2x}^2\,m_2\,\mathrm{qdd}_1}{2} + \mathrm{Pc1z}^2\,m_1\,\mathrm{qdd}_1 + \frac{\mathrm{Pc2y}^2\,m_2\,\mathrm{qdd}_1}{2} \\
& + \frac{\mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_1}{2} + \mathrm{Pc2z}^2\,m_2\,\mathrm{qdd}_1 + \frac{\mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_1}{2} + \mathrm{Pc3z}^2\,m_3\,\mathrm{qdd}_1 + \frac{\mathrm{Pc2x}^2\,m_2\,\mathrm{qdd}_1\,\cos(2\,q_2)}{2} \\
& - \frac{\mathrm{Pc2y}^2\,m_2\,\mathrm{qdd}_1\,\cos(2\,q_2)}{2} + \frac{{l_2}^2\,m_2\,\mathrm{qdd}_1\,\cos(2\,q_2)}{2} + \frac{{l_2}^2\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2)}{2} + \frac{\mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+2\,q_3)}{2} \\
& - \frac{\mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+2\,q_3)}{2} + \frac{{l_3}^2\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+2\,q_3)}{2} + \mathrm{Pc2x}\,l_2\,m_2\,\mathrm{qdd}_1 + \mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_1 \\
& - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qdd}_1\,\sin(2\,q_2+q_3) + \mathrm{Pc2x}\,\mathrm{Pc2z}\,m_2\,{\mathrm{qd}_2}^2\,\cos(q_2) + l_2\,l_3\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+q_3) \\
& - \mathrm{Pc2x}\,\mathrm{Pc2y}\,m_2\,\mathrm{qdd}_1\,\sin(2\,q_2) - \mathrm{Pc2y}\,\mathrm{Pc2z}\,m_2\,{\mathrm{qd}_2}^2\,\sin(q_2) + \mathrm{Pc2x}\,l_2\,m_2\,\mathrm{qdd}_1\,\cos(2\,q_2) \\
& + \mathrm{Pc2z}\,l_2\,m_2\,{\mathrm{qd}_2}^2\,\cos(q_2) + \mathrm{Pc3z}\,l_2\,m_3\,{\mathrm{qd}_2}^2\,\cos(q_2) - \mathrm{Pc2y}\,l_2\,m_2\,\mathrm{qdd}_1\,\sin(2\,q_2) \\
& - \mathrm{Pc3x}\,\mathrm{Pc3y}\,m_3\,\mathrm{qdd}_1\,\sin(2\,q_2+2\,q_3) + \mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+2\,q_3) - \mathrm{Pc3y}\,l_3\,m_3\,\mathrm{qdd}_1\,\sin(2\,q_2+2\,q_3) \\
& + \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_2\,\cos(q_2+q_3) + \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_3\,\cos(q_2+q_3) + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_2\,\sin(q_2+q_3) \\
& + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_3\,\sin(q_2+q_3) - \mathrm{Pc2x}^2\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2) + \mathrm{Pc2y}^2\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2) \\
& + \mathrm{Pc3z}\,l_3\,m_3\,\mathrm{qdd}_2\,\sin(q_2+q_3) + \mathrm{Pc3z}\,l_3\,m_3\,\mathrm{qdd}_3\,\sin(q_2+q_3) + \mathrm{Pc2y}\,\mathrm{Pc2z}\,m_2\,\mathrm{qdd}_2\,\cos(q_2) \\
& - {l_2}^2\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2) - {l_2}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2) + \mathrm{Pc2x}\,\mathrm{Pc2z}\,m_2\,\mathrm{qdd}_2\,\sin(q_2) \\
& + \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qdd}_1\,\cos(q_3) - \mathrm{Pc3x}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+2\,q_3) - \mathrm{Pc3x}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+2\,q_3) \\
& + \mathrm{Pc3y}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+2\,q_3) + \mathrm{Pc3y}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+2\,q_3) + \mathrm{Pc2z}\,l_2\,m_2\,\mathrm{qdd}_2\,\sin(q_2) \\
& - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qdd}_1\,\sin(q_3) + \mathrm{Pc3z}\,l_2\,m_3\,\mathrm{qdd}_2\,\sin(q_2) + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,{\mathrm{qd}_2}^2\,\cos(q_2+q_3) \\
& + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,{\mathrm{qd}_3}^2\,\cos(q_2+q_3) + l_2\,l_3\,m_3\,\mathrm{qdd}_1\,\cos(q_3) - {l_3}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+2\,q_3) \\
& - {l_3}^2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+2\,q_3) - \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,{\mathrm{qd}_2}^2\,\sin(q_2+q_3) - \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,{\mathrm{qd}_3}^2\,\sin(q_2+q_3) \\
& + \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qdd}_1\,\cos(2\,q_2+q_3) + \mathrm{Pc3z}\,l_3\,m_3\,{\mathrm{qd}_2}^2\,\cos(q_2+q_3) + \mathrm{Pc3z}\,l_3\,m_3\,{\mathrm{qd}_3}^2\,\cos(q_2+q_3) \\
& - 2\,\mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\sin(q_2+q_3) + 2\,\mathrm{Pc3z}\,l_3\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\cos(q_2+q_3) - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\cos(q_3) \\
& - \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(q_3) - l_2\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(q_3) - 2\,\mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\cos(2\,q_2+q_3) \\
& - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\cos(2\,q_2+q_3) - 2\,\mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+q_3) - \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+q_3) \\
& - 2\,\mathrm{Pc2x}\,\mathrm{Pc2y}\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\cos(2\,q_2) - 2\,l_2\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+q_3) - l_2\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+q_3) \\
& - 2\,\mathrm{Pc2y}\,l_2\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\cos(2\,q_2) - 2\,\mathrm{Pc2x}\,l_2\,m_2\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2) - 2\,\mathrm{Pc3x}\,\mathrm{Pc3y}\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\cos(2\,q_2+2\,q_3) \\
& - 2\,\mathrm{Pc3x}\,\mathrm{Pc3y}\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\cos(2\,q_2+2\,q_3) - 2\,\mathrm{Pc3y}\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\cos(2\,q_2+2\,q_3) \\
& - 2\,\mathrm{Pc3y}\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\cos(2\,q_2+2\,q_3) - 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_2\,\sin(2\,q_2+2\,q_3) \\
& - 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qd}_1\,\mathrm{qd}_3\,\sin(2\,q_2+2\,q_3) + 2\,\mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\cos(q_2+q_3)
\end{aligned}
$$

$$
\begin{aligned}
\tau_2 = {} & \mathrm{Iz}_2\,\mathrm{qdd}_1 + \mathrm{Iz}_2\,\mathrm{qdd}_2 + \mathrm{Iz}_3\,\mathrm{qdd}_1 + \mathrm{Iz}_3\,\mathrm{qdd}_2 + \mathrm{Iz}_3\,\mathrm{qdd}_3 + {l_2}^2\,m_2\,\mathrm{qdd}_2 + {l_2}^2\,m_3\,\mathrm{qdd}_2 + {l_3}^2\,m_3\,\mathrm{qdd}_2 \\
& + {l_3}^2\,m_3\,\mathrm{qdd}_3 + \mathrm{Pc2x}^2\,m_2\,\mathrm{qdd}_2 + \mathrm{Pc2y}^2\,m_2\,\mathrm{qdd}_2 + \mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_2 + \mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_3 + \mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_2 \\
& + \mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_3 + \mathrm{Pc3x}\,g\,m_3\,\cos(q_2+q_3) - \mathrm{Pc3y}\,g\,m_3\,\sin(q_2+q_3) + g\,l_3\,m_3\,\cos(q_2+q_3) \\
& + \mathrm{Pc2x}\,g\,m_2\,\cos(q_2) - \mathrm{Pc2y}\,g\,m_2\,\sin(q_2) + g\,l_2\,m_2\,\cos(q_2) + g\,l_2\,m_3\,\cos(q_2) + \frac{\mathrm{Pc2x}^2\,m_2\,{\mathrm{qd}_1}^2\,\sin(2\,q_2)}{2} \\
& - \frac{\mathrm{Pc2y}^2\,m_2\,{\mathrm{qd}_1}^2\,\sin(2\,q_2)}{2} + \frac{{l_2}^2\,m_2\,{\mathrm{qd}_1}^2\,\sin(2\,q_2)}{2} + \frac{{l_2}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2)}{2} + \frac{\mathrm{Pc3x}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} \\
& - \frac{\mathrm{Pc3y}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} + \frac{{l_3}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} + 2\,\mathrm{Pc2x}\,l_2\,m_2\,\mathrm{qdd}_2 \\
& + 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_2 + 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_3 - \mathrm{Pc3y}\,l_2\,m_3\,{\mathrm{qd}_3}^2\,\cos(q_3) - \mathrm{Pc3x}\,l_2\,m_3\,{\mathrm{qd}_3}^2\,\sin(q_3) \\
& - l_2\,l_3\,m_3\,{\mathrm{qd}_3}^2\,\sin(q_3) + \mathrm{Pc3y}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+q_3) + \mathrm{Pc3x}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+q_3) \\
& + \mathrm{Pc2x}\,\mathrm{Pc2y}\,m_2\,{\mathrm{qd}_1}^2\,\cos(2\,q_2) + \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_1\,\cos(q_2+q_3) + l_2\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+q_3) \\
& + \mathrm{Pc2y}\,l_2\,m_2\,{\mathrm{qd}_1}^2\,\cos(2\,q_2) + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_1\,\sin(q_2+q_3) + \mathrm{Pc2x}\,l_2\,m_2\,{\mathrm{qd}_1}^2\,\sin(2\,q_2) \\
& + \mathrm{Pc3x}\,\mathrm{Pc3y}\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+2\,q_3) + \mathrm{Pc3z}\,l_3\,m_3\,\mathrm{qdd}_1\,\sin(q_2+q_3) + \mathrm{Pc2y}\,\mathrm{Pc2z}\,m_2\,\mathrm{qdd}_1\,\cos(q_2) \\
& + \mathrm{Pc3y}\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+2\,q_3) + \mathrm{Pc2x}\,\mathrm{Pc2z}\,m_2\,\mathrm{qdd}_1\,\sin(q_2) + \mathrm{Pc3x}\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3) \\
& + 2\,\mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qdd}_2\,\cos(q_3) + \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qdd}_3\,\cos(q_3) + \mathrm{Pc2z}\,l_2\,m_2\,\mathrm{qdd}_1\,\sin(q_2) + \mathrm{Pc3z}\,l_2\,m_3\,\mathrm{qdd}_1\,\sin(q_2) \\
& - 2\,\mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qdd}_2\,\sin(q_3) - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qdd}_3\,\sin(q_3) + 2\,l_2\,l_3\,m_3\,\mathrm{qdd}_2\,\cos(q_3) + l_2\,l_3\,m_3\,\mathrm{qdd}_3\,\cos(q_3) \\
& - 2\,\mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\cos(q_3) - 2\,\mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\sin(q_3) - 2\,l_2\,l_3\,m_3\,\mathrm{qd}_2\,\mathrm{qd}_3\,\sin(q_3)
\end{aligned}
$$

$$
\begin{aligned}
\tau_3 = {} & \mathrm{Iz}_3\,\mathrm{qdd}_1 + \mathrm{Iz}_3\,\mathrm{qdd}_2 + \mathrm{Iz}_3\,\mathrm{qdd}_3 + {l_3}^2\,m_3\,\mathrm{qdd}_2 + {l_3}^2\,m_3\,\mathrm{qdd}_3 + \mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_2 + \mathrm{Pc3x}^2\,m_3\,\mathrm{qdd}_3 \\
& + \mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_2 + \mathrm{Pc3y}^2\,m_3\,\mathrm{qdd}_3 + \mathrm{Pc3x}\,g\,m_3\,\cos(q_2+q_3) - \mathrm{Pc3y}\,g\,m_3\,\sin(q_2+q_3) \\
& + g\,l_3\,m_3\,\cos(q_2+q_3) + \frac{\mathrm{Pc3x}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} - \frac{\mathrm{Pc3y}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} \\
& + \frac{{l_3}^2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3)}{2} + 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_2 + 2\,\mathrm{Pc3x}\,l_3\,m_3\,\mathrm{qdd}_3 + \frac{\mathrm{Pc3y}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\cos(q_3)}{2} \\
& + \mathrm{Pc3y}\,l_2\,m_3\,{\mathrm{qd}_2}^2\,\cos(q_3) + \frac{\mathrm{Pc3x}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\sin(q_3)}{2} + \mathrm{Pc3x}\,l_2\,m_3\,{\mathrm{qd}_2}^2\,\sin(q_3) + \frac{l_2\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\sin(q_3)}{2} \\
& + l_2\,l_3\,m_3\,{\mathrm{qd}_2}^2\,\sin(q_3) + \frac{\mathrm{Pc3y}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+q_3)}{2} + \frac{\mathrm{Pc3x}\,l_2\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+q_3)}{2} \\
& + \mathrm{Pc3y}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_1\,\cos(q_2+q_3) + \frac{l_2\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+q_3)}{2} + \mathrm{Pc3x}\,\mathrm{Pc3z}\,m_3\,\mathrm{qdd}_1\,\sin(q_2+q_3) \\
& + \mathrm{Pc3x}\,\mathrm{Pc3y}\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+2\,q_3) + \mathrm{Pc3z}\,l_3\,m_3\,\mathrm{qdd}_1\,\sin(q_2+q_3) + \mathrm{Pc3y}\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\cos(2\,q_2+2\,q_3) \\
& + \mathrm{Pc3x}\,l_3\,m_3\,{\mathrm{qd}_1}^2\,\sin(2\,q_2+2\,q_3) + \mathrm{Pc3x}\,l_2\,m_3\,\mathrm{qdd}_2\,\cos(q_3) - \mathrm{Pc3y}\,l_2\,m_3\,\mathrm{qdd}_2\,\sin(q_3) + l_2\,l_3\,m_3\,\mathrm{qdd}_2\,\cos(q_3)
\end{aligned}
$$

Lagrangian Script

```
function tau = lagrangian(Robot,grav,nlink,fi,ni)
% Lagrangian Dynamics
% Solves for the torque on each joint of a Revolute Robot using the
% Lagrangian method.
% Takes inputs DH (modified), inertia tensor I, position of link center of mass Pc
% and gravity vector grav.


syms t q1 q2 q3 qd1 qd2 qd3 qdd1 qdd2 qdd3 q4 q5 q6 fx fy fz nx ny nz

g=grav(3);
q=[q1;q2;q3];
qd  = [qd1,qd2,qd3]; % theta dot vector
qdd = [qdd1,qdd2,qdd3]; % theta double dot vector
Vc0 = sym(zeros(3,nlink)); % initialize Velocity of center of mass with respect to the base frame
omega = sym(zeros(3,nlink)); % initialize angular velocity of links wrt to LINK frame
Pc0     = sym(zeros(4,nlink)); %Position of center of mass wrt base
dPdq    = sym(zeros(4,nlink));
KE  = sym(zeros(1,nlink));
PE  = sym(zeros(1,nlink));


for i=1:nlink
    Trans(:,:,i)=Robot.A([i],[q]);
    R(:,:,i)=Trans(:,:,i).R;
end

T2=Trans;

for i=1:nlink 1
    T2(:,:,i+1)=T2(:,:,i)*T2(:,:,i+1);
end


% Calculate Position of Center of Mass
for i = 1:nlink
    Pc0(:,i)    = T2(:,:,i).T * [Robot.links(i).r.';1]; % Position of each center of mass wrt 0

    if(i == 1)
        omega(:,i) = [0;0;qd(i)];
    else
        omega(:,i) = R(:,:,i).'*omega(:,i 1)+[0;0;qd(i)]; % Use transpose for i_R_i 1
    end
end

% Calculate velocity
for i = 1:nlink
    dPdq(:,:,i) = diff(Pc0,q(i))*qd(i); % differentiate position vector, wrt base frame
    Vc0     = Vc0 + dPdq(1:3,:,i); % velocity of mass_i is velocity of mass plus the velocity
end


% Calculate kinetic and potential energies
for i = 1:nlink
    PE(i)   = Robot.links(i).m*g*Pc0(3,i);
    KE(i)   = 0.5*(Robot.links(i).m*Vc0(:,i).'*Vc0(:,i)+omega(:,i).'*Robot.links(i).I*omega(:,i));
```

```
end

KE = collect(simplify(expand(KE),10),[qdd qd.^2]);

L   = sum(KE) sum(PE);


jacob=Robot.jacobe([q1;q2;q3]);

fe=[fi;ni];
Q=simplify(jacob.'*fe);

% calculate torques on joints using the Lagrangian
for i = 1:nlink
    tau(i) = timeDeriv2n(diff(L,qd(i)),q,qd,qdd)   diff(L,q(i));
end
tau=tau Q.';

tau = collect(simplify(expand(tau),10),[qdd qd.^2,g]);
end
```

Time derivative Script

```
function dfdt = timeDeriv2n(f,q,qd,qdd)
% takes the time derivative of up to second order polynomials that have
% the symbols q.

dfdt = sym(zeros(size(f,1)));

for i = 1: length(q)
    dfdt = diff(f,q(i))*qd(i) + diff(f,qd(i))*qdd(i) + dfdt;
end


end
```

End-Effector Trajectory Verification

```
clear all
close all
clc

t = linspace(1,2500,1000);
x(1:length(t)) = 280;

for i = 1:length(t)
    y(i) = 200*sin((2*pi)/10*t(i)/1000);
    z(i) = 250 + 90*sin((2*pi)/1.5*t(i)/1000);
    plot3(x(i),y(i),z(i),'r.')
    xlim([ 100 290])
    ylim([ 250 250])
    zlim([0 500])

    hold on
    grid on
end
grid minor
```

Velocity Filter C code Implementation

```
//First Method of Filtering Velocity, FIR

float Theta1_old = 0;
float Omega1_raw = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;

void lab(float thetamotor1,float thetamotor2,float thetamotor3,float *tau1,
    float *tau2,float *tau3) {
     Omega1_raw = (thetamotor1 - Theta1_old)/0.001;
     Omega1 = (Omega1_raw + Omega1_old1 + Omega1_old2)/3.0;
     Theta1_old = thetamotor1;
     Omega1_old2 = Omega1_old1;
     Omega1_old1 = Omega1_raw;
}

//Second Method of Filtering Velocity, IIR
float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;

void lab(float thetamotor1,float thetamotor2,float thetamotor3,float *tau1,
    float *tau2,float *tau3) {
     Omega1 = (thetamotor1 - Theta1_old)/0.001;
     Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;
     Theta1_old = thetamotor1;
     Omega1_old2 = Omega1_old1;
     Omega1_old1 = Omega1;
}
```

End-Effector Trajectory Tracking C code

```
1
2  // ME446 Robot Dynamics and Controls
3  // Lab 2: Kinematic Transformation
4  // Kevin Gim & Jasvir Virdi
5
6  #include <tistdtypes.h>
7  #include <coecsl.h>
8  #include <user_includes.h>
9  #include <math.h>
10
11 float offset_Enc2_rad = -0.36;
12 float offset_Enc3_rad = 0.27;
13
14 // Your global varialbes.
15 long mycount = 0;
16 long arrayindex = 0;
17
18 float printtheta1motor = 0;
19 float printtheta2motor = 0;
20 float printtheta3motor = 0;
21
22 float DHtheta1 = 0;
23 float DHtheta2 = 0;
24 float DHtheta3 = 0;
25
26 float x = 0;
27 float y = 0;
28 float z = 0;
29
30 float IKtheta1DH = 0;
31 float IKtheta2DH = 0;
32 float IKtheta3DH = 0;
33
34 float IKthetam1 = 0;
35 float IKthetam2 = 0;
36 float IKthetam3 = 0;
37
38 float r1 = 0;
39 float r2 = 0;
40
41 float l1 = 254;
42 float l2 = 254;
43 float l3 = 254;
44
45 float Theta1_old = 0;
46 float Omega1_old1 = 0;
47 float Omega1_old2 = 0;
48 float Omega1 = 0;
49
50 float Theta2_old = 0;
51 float Omega2_old1 = 0;
52 float Omega2_old2 = 0;
53 float Omega2 = 0;
54
55 float Theta3_old = 0;
56 float Omega3_old1 = 0;
```

```
float Omega3_old2 = 0;
float Omega3 = 0;

float theta1_des = 0;
float theta2_des = 0;
float theta3_des = 0;
float Omega1_des = 0;
float Omega2_des = 0;
float Omega3_des = 0;
float Omega1d_des = 0;
float Omega2d_des = 0;
float Omega3d_des = 0;

float J1 = 0.0167;
float J2 = 0.03;
float J3 = 0.0128;

float error_1 = 0.0;
float error_2 = 0.0;
float error_3 = 0.0;

float traperror_1 = 0.0;
float traperror_2 = 0.0;
float traperror_3 = 0.0;

float I_1 = 0.0;
float I_2 = 0.0;
float I_3 = 0.0;

float thresh_1 = 0.05;
float thresh_2 = 0.05;
float thresh_3 = 0.03;

float a_1 = -1;
float b_1 = 1.5;
float a_2 = 1;
float b_2 = -4.5;
float c_2 = 6;
float d_2 = -2;

float t = 0.0;

float x_f = 0.0;
float y_f = 0.0;
float z_f = 0.0;

// Control Gain Values
////Just PD
//float KP_1 = 50;
//float KP_2 = 40;
//float KP_3 = 40;
//float KD_1 = 2.1;
//float KD_2 = 2.1;
//float KD_3 = 1.5;

// //Just PID
float KP_1 = 51;
```

```
114 float KP_2 = 40;
115 float KP_3 = 50;
116 float KD_1 = 2.5;
117 float KD_2 = 2.0;
118 float KD_3 = 1.3;
119 float KI_1 = 250.0;
120 float KI_2 = 200.0;
121 float KI_3 = 300.0;
122
123 // PID for cubic reference trajectory
124 //float KP_1 = 50;
125 //float KP_2 = 50;
126 //float KP_3 = 50;
127
128 //float KD_1 = 1.0;
129 //float KD_2 = 1.0;
130 //float KD_3 = 1.0;
131
132 //float KI_1 = 100.0;
133 //float KI_2 = 100.0;
134 //float KI_3 = 100.0;
135
136 // Assign these float to the values you would like to plot in Simulink
137 float Simulink_PlotVar1 = 0;
138 float Simulink_PlotVar2 = 0;
139 float Simulink_PlotVar3 = 0;
140 float Simulink_PlotVar4 = 0;
141
142
143 // This function is called every 1 ms
144 void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,
       float *tau2,float *tau3, int error) {
145         //Motor torque limitation(Max: 5 Min: -5)
146         // save past states
147         if ((mycount%50)==0) {
148
149                 theta1array[arrayindex] = theta1motor;
150
151                 if (arrayindex >= 100) {
152                         arrayindex = 0;
153                 } else {
154                         arrayindex++;
155                 }
156
157         }
158
159         if ((mycount%50)==0) {
160                 if (whattoprint > 0.5) {
161                         serial_printf(&SerialA, I love robotics);
162                 } else {
163                         printtheta1motor = theta1motor;
164                         printtheta2motor = theta2motor;
165                         printtheta3motor = theta3motor;
166                         DHtheta1 = theta1motor;
167                         DHtheta2 = theta2motor-PI*0.5;
168                         DHtheta3 = theta3motor-theta2motor+PI*0.5;
169                         SWI_post(&SWI_printf); //Using a SWI to fix SPI issue
```

```
                                from sending too many floats.
170                 }
171                 GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Blink LED on Control
                       Card
172                 GpioDataRegs.GPBTOGGLE.bit.GPIO60 = 1; // Blink LED on
                       Emergency Stop Box
173         }
174
175 //// Step input
176 //   t = (mycount%3000)/1500.;
177 //  if ((mycount%3000)<1500) {
178 //      theta1_des = 0.2;
179 //      theta2_des = 0.3;
180 //      theta3_des = 0.4;
181 //  }
182 //  else{
183 //      theta1_des = 0;
184 //      theta2_des = 0;
185 //      theta3_des = 0;
186 //
187 //  }
188
189 //// Smooth motor trajectory (cubic)
190 //          t = (mycount%2000)/1000.;
191 //      if ((mycount%2000)<1000) {
192 //
193 //          theta1_des = a_1*t*t*t + b_1*t*t;
194 //          theta2_des = a_1*t*t*t + b_1*t*t;
195 //          theta3_des = a_1*t*t*t + b_1*t*t;
196 //
197 //          Omega1_des = 3*a_1*t*t + 2*b_1*t;
198 //          Omega2_des = 3*a_1*t*t + 2*b_1*t;
199 //          Omega3_des = 3*a_1*t*t + 2*b_1*t;
200 //
201 //          Omega1d_des = 6*a_1 + 2*b_1;
202 //          Omega2d_des = 6*a_1 + 2*b_1;
203 //          Omega3d_des = 6*a_1 + 2*b_1;
204 //
205 //          }
206 //      else {
207 //          theta1_des = a_2*t*t*t + b_2*t*t + c_2*t + d_2;
208 //          theta2_des = a_2*t*t*t + b_2*t*t + c_2*t + d_2;
209 //        theta3_des = a_2*t*t*t + b_2*t*t + c_2*t + d_2;
210 //
211 //        Omega1_des = 3*a_2*t*t + 2*b_2*t + c_2;
212 //        Omega2_des = 3*a_2*t*t + 2*b_2*t + c_2;
213 //        Omega3_des = 3*a_2*t*t + 2*b_2*t + c_2;
214 //
215 //        Omega1d_des = 6*a_2 + 2*b_2;
216 //        Omega2d_des = 6*a_2 + 2*b_2;
217 //        Omega3d_des = 6*a_2 + 2*b_2;
218 //      }
219
220     // Reference Sine wave trajectory
221     t = (mycount)/1000.;
222     x = 280;
223     y = 0.0 + 200*sin((2*PI)*t/10);
```

```
    z = 250 + 90*sin((2*PI)*t/1.5);

    //Inverse Kinematics
    IKtheta1DH = atan2(y,x);
    r1 = z-l1;
    r2 = sqrt(r1*r1 + x*x + y*y);
    IKtheta3DH = PI - acos((l2*l2+l3*l3-r2*r2)/(2*l2*l3));
    IKtheta2DH = -(IKtheta3DH)/2 - asin((r1/r2));

    IKthetam1= IKtheta1DH;
    IKthetam2=IKtheta2DH +(PI/2);
    IKthetam3=IKtheta3DH + IKtheta2DH;

    theta1_des = IKthetam1;
    theta2_des = IKthetam2;
    theta3_des = IKthetam3;

// Select which value to display in Simulink plot

        Simulink_PlotVar1 = theta1_des;
        Simulink_PlotVar2 = theta1motor;
        Simulink_PlotVar3 = theta2_des;
        Simulink_PlotVar4 = theta2motor;

        //Lab 2 PD control

        // Theta1 velocity
    Omega1 = (theta1motor - Theta1_old)/0.001;
    Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;

    Theta1_old = theta1motor;

    Omega1_old2 = Omega1_old1;
    Omega1_old1 = Omega1;

    Omega1 = (theta1motor - Theta1_old)/0.001;
    Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;

    // Theta2 velocity
        Omega2 = (theta2motor - Theta2_old)/0.001;
        Omega2 = (Omega2 + Omega2_old1 + Omega2_old2)/3.0;

        Theta2_old = theta2motor;

        Omega2_old2 = Omega2_old1;
        Omega2_old1 = Omega2;

        // Theta3 velocity
    Omega3 = (theta3motor - Theta3_old)/0.001;
    Omega3 = (Omega3 + Omega3_old1 + Omega3_old2)/3.0;

    Theta3_old = theta3motor;

    Omega3_old2 = Omega3_old1;
    Omega3_old1 = Omega3;

    error_1 = theta1_des - theta1motor;
```

```
    error_2 = theta2_des - theta2motor;
    error_3 = theta3_des - theta3motor;

    // Integral Control

    if (fabs(*tau1) > 5){
        traperror_1 = 0;}
    else{
        traperror_1 = error_1/2*0.001;
        I_1 += traperror_1;
    }

    if (fabs(*tau2) > 5){
        traperror_2 = 0;}
    else{
        traperror_2 = error_2/2*0.001;
            I_2 += traperror_2;
        }

    if (fabs(*tau3) > 5){
        traperror_3 = 0;}
    else{
        traperror_3 = error_3/2*0.001;
                I_3 += traperror_3;
            }

    if (fabs(error_1)>thresh_1){
        I_1 = 0;
    }

    if (fabs(error_2)>thresh_2){
            I_2 = 0;
       }

    if (fabs(error_3)>thresh_3){
            I_3 = 0;
       }

//    PD control(w_des = 0)
//    *tau1 = KP_1 * (error_1) - KD_1 * Omega1;
//    *tau2 = KP_2 * (error_2) - KD_2 * Omega2;
//    *tau3 = KP_2 * (error_3) - KD_3 * Omega3;

//    PID control(w_des = 0)
    *tau1 = KP_1 * (error_1) - KD_1 * Omega1 + KI_1 * I_1;
    *tau2 = KP_2 * (error_2) - KD_2 * Omega2 + KI_2 * I_2;
    *tau3 = KP_2 * (error_3) - KD_3 * Omega3 + KI_3 * I_3;

//    Feedforward Control(w_des != 0)
//    *tau1 = KP_1 * (error_1) + KD_1 * (Omega1_des - Omega1) + J1*Omega1d_des;
//    *tau2 = KP_2 * (error_2) + KD_2 * (Omega2_des - Omega2) + J2*Omega2d_des;
//    *tau3 = KP_2 * (error_3) + KD_3 * (Omega3_des - Omega3) + J3*Omega3d_des;

        mycount++;
}
}
```