Robot Dynamics and Control

Lab Report

Kinematic Transformations

Amith Ramdas Achari
amithr3
Peng Han
penghan2
Section: Friday 9AM
Submitted on February 22, 2022

1 Introduction

A proportional–integral–derivative controller (PID controller or three-term controller) is a feedback-based control loop commonly used in industrial control systems and other applications that require continuously modulated control. A PID controller calculates an error value as the difference between a desired setpoint (SP) and a measured process variable (PV) on a continuous basis and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D, respectively).

Proportional action improves the response time and also reduces the steady-state error, but the value of overshoot increases. Derivative control improves damping, while keeping the speed of response the same. But the disadvantage of having a derivative control is that is amplifies high-frequency noise and disturbances. Integral control reduces the steady-state error, usually the error tends to zero, but at the same time it degrades the system's stability and the speed of response. This lab addresses how to implement PID control on the CRS Robot and how to tune each gain.

2 Tasks

• Using Lagrangian dynamics, derive the equations of motion for a two-link revolute linkage.

• Create a simulation of this linkage in Simulink.

• Implement velocity and integration calculations.

• Design and simulate two-link planar robot PD joint control.

• Implement PD joint control for three of the CRS robot joints and tune it to attain desired requirements.

• Integral control should be added to the PD joint control.

• Set up and fine-tune PD with Feedforward Control.

• Create a task space trajectory using joints 1, 2, and 3 to make the CRS robot's end effect trace a figure eight or something else fun.

3 Two-link equations of motion (Derivation)

3.1

3、1

Based on Figure 1

$$R_1^0 = \begin{bmatrix} \cos\theta_{2DH} & -\sin\theta_{2DH} & 0 \\ \sin\theta_{2DH} & \cos\theta_{2DH} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$x_{c2} = l_{c2}\cos\theta_{2DH}$, $y_{c2} = l_{c2}\sin\theta_{2DH}$, $z_{c2} = 0$

$$\therefore J_{v,c2} = \begin{bmatrix} \frac{d x_{c2}}{d\theta_{2DH}} & \frac{d x_{c2}}{d\theta_{3DH}} \\ \frac{d y_{c2}}{d\theta_{2DH}} & \frac{d y_{c2}}{d\theta_{3DH}} \\ \frac{d z_{c2}}{d\theta_{2DH}} & \frac{d z_{c2}}{d\theta_{3DH}} \end{bmatrix} = \begin{bmatrix} -l_{c2}\sin\theta_{2DH} & 0 \\ l_{c2}\cos\theta_{2DH} & 0 \\ 0 & 0 \end{bmatrix}$$

$x_{c3} = l_2\cos\theta_{2DH} + l_{c3}\cos(\theta_{2DH}+\theta_{3DH})$

$y_{c3} = l_2\sin\theta_{2DH} + l_{c3}\sin(\theta_{2DH}+\theta_{3DH})$, $z_{c3} = 0$

$$\therefore J_{v,c3} = \begin{bmatrix} \frac{d x_{c3}}{d\theta_{2DH}} & \frac{d x_{c3}}{d\theta_{3DH}} \\ \frac{d y_{c3}}{d\theta_{2DH}} & \frac{d y_{c3}}{d\theta_{3DH}} \\ \frac{d z_{c3}}{d\theta_{2DH}} & \frac{d z_{c3}}{d\theta_{3DH}} \end{bmatrix} = \begin{bmatrix} -l_2\sin\theta_{2DH} - l_{c3}\sin(\theta_{2DH}+\theta_{3DH}) & -l_{c3}\sin(\theta_{2DH}+\theta_{3DH}) \\ l_2\cos\theta_{2DH} + l_{c3}\cos(\theta_{2DH}+\theta_{3DH}) & l_{c3}\cos(\theta_{2DH}+\theta_{3DH}) \\ 0 & 0 \end{bmatrix}$$

$$J_{\omega,c2} = \left[R_0^0 u_1 \ , \ 0 \right] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

$$J_{\omega,c3} = \left[R_0^0 u_1 \ , \ R_1^0 u_2 \right] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} I_{2xx} & I_{2xy} & I_{2xz} \\ I_{2yx} & I_{2yy} & I_{2yz} \\ I_{2zx} & I_{2zy} & I_{2zz} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix}$$

$$\begin{bmatrix} I_{3xx} & I_{3xy} & I_{3xz} \\ I_{3yx} & I_{3yy} & I_{3yz} \\ I_{3zx} & I_{3zy} & I_{3zz} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_3 \end{bmatrix}$$

$$K = \frac{1}{2} m_2 \dot{q}^T J_{v,c2}^T J_{v,c2} \dot{q} + \frac{1}{2} \dot{q}^T J_{\omega,c2}^T R_1^0 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix} R_1^{0T} J_{\omega,c2} \dot{q}$$
$$+ \frac{1}{2} m_3 \dot{q}^T J_{v,c3}^T J_{v,c3} \dot{q} + \frac{1}{2} \dot{q}^T J_{\omega,c3}^T R_2^0 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_3 \end{bmatrix} R_2^{0T} J_{\omega,c3} \dot{q}$$

① $\frac{1}{2} m_2 \dot{q}^T J_{v,c2}^T J_{v,c2} \dot{q}$

$$= \frac{1}{2} m_2 \cdot [\dot{\theta}_{2DH} \ , \ \dot{\theta}_{3DH}] \cdot \begin{bmatrix} l_{c2}^2 \sin^2\theta_{2DH} + l_{c2}^2 \cos^2\theta_{2DH} & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2} m_2 \cdot [\, l_{c2}^2 \dot{\theta}_{2DH} \quad 0 \,] \cdot \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2} m_2 l_{c2}^2 \dot{\theta}_{2DH}^2$$

② $\frac{1}{2} m_3 \dot{q}^T J_{v,c3}^T J_{v,c3} \dot{q}$

$$J_{vc3}^T \cdot J_{vc3} = \begin{bmatrix} l_2^2 + l_{c3}^2 + 2 l_2 l_{c3} \cos\theta_{3DH} \ , & l_{c3}^2 + l_2 l_{c3} \cos\theta_{3DH} \\ l_{c3}^2 + l_2 l_{c3} \cos\theta_{3DH} \ , & l_{c3}^2 \end{bmatrix}$$

$$\frac{1}{2} m_3 \dot{q}^T J_{vc_3}^T J_{vc_3} \dot{q}$$

$$= \frac{1}{2} m_3 [\dot{\theta}_{2DH}, \dot{\theta}_{3DH}] \cdot \begin{bmatrix} (l_2^2 + l_{c_3}^2 + 2 l_2 l_{c_3} \cos\theta_{3DH})\dot{\theta}_{2DH} + (l_{c_3}^2 + l_2 l_{c_3} \cos\theta_{3DH}) \cdot \dot{\theta}_{3DH} \\ (l_{c_3}^2 + l_2 l_{c_3} \cos\theta_{3DH}) \dot{\theta}_{2DH} + l_{c_3}^2 \dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2} m_3 \left[(l_2^2 + l_{c_3}^2 + 2 l_2 l_{c_3} \cos\theta_{3DH}) \dot{\theta}_{2DH}^2 + 2(l_{c_3}^2 + l_2 l_{c_3} \cos\theta_{3DH}) \dot{\theta}_{2DH} \dot{\theta}_{3DH} + l_{c_3}^2 \dot{\theta}_{3DH}^2 \right]$$

③ $$\frac{1}{2} \dot{q}^T J_{\omega c_2}^T R_1^0 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix} R_1^0 J_{\omega c_2} \dot{q}$$

$$= \frac{1}{2} [\dot{\theta}_{2DH}, \dot{\theta}_{3DH}] \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{2DH} & -\sin\theta_{2DH} & 0 \\ \sin\theta_{2DH} & \cos\theta_{2DH} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{2DH} & -\sin\theta_{2DH} & 0 \\ \sin\theta_{2DH} & \cos\theta_{2DH} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2} [0 \quad 0 \quad \dot{\theta}_{2DH}] \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{2DH} \end{bmatrix}$$

$$= \frac{1}{2} I_2 \dot{\theta}_{2DH}^2$$

④ $$\frac{1}{2} \dot{q}^T J_{\omega,c_3}^T R_2^0 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_3 \end{bmatrix} R_2^{0T} J_{\omega,c_3} \dot{q}$$

$$= \frac{1}{2} [\dot{\theta}_{2DH}, \dot{\theta}_{3DH}] \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{2DH} & -\sin\theta_{2DH} & 0 \\ \sin\theta_{2DH} & \cos\theta_{2DH} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_{2DH} & -\sin\theta_{2DH} & 0 \\ \sin\theta_{2DH} & \cos\theta_{2DH} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} 0 & 0 & \dot{\theta}_{2DH}+\dot{\theta}_{3DH}\end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{2DH}+\dot{\theta}_{3DH} \end{bmatrix}$$

$$= \frac{1}{2} I_3 \cdot (\dot{\theta}_{2DH}+\dot{\theta}_{3DH})^2$$

$$\therefore K = \textcircled{1} + \textcircled{2} + \textcircled{3} + \textcircled{4}$$

$$\left.\begin{aligned} K = {} & \frac{1}{2} m_2 l_{c2}^2 \dot{\theta}_{2DH}^2 + \frac{1}{2} m_3 l_{c3}^2 \cdot (\dot{\theta}_{2DH}^2 + 2\dot{\theta}_{2DH}\cdot\dot{\theta}_{3DH} + \dot{\theta}_{3DH}^2) \\ & + \frac{1}{2} m_3 l_2^2 \dot{\theta}_{2DH}^2 + m_3 l_2 l_{c3} \cos\theta_{3DH} \dot{\theta}_{2DH}^2 + m_3 l_2 l_{c3} \cos\theta_{3DH} \dot{\theta}_{2DH} \dot{\theta}_{3DH} \end{aligned}\right\} \textcircled{1} + \textcircled{2}$$

$$+ \underbrace{\frac{1}{2} I_2 \dot{\theta}_{2DH}^2}_{\textcircled{3}} + \underbrace{\frac{1}{2} I_3 (\dot{\theta}_{2OH} + \dot{\theta}_{3DH})^2}_{\textcircled{4}}$$

$$\begin{aligned} K = {} & \frac{1}{2} m_2 l_{c2}^2 \dot{\theta}_{2DH}^2 + \frac{1}{2} m_3 l_2^2 \theta_{2DH}^2 + \frac{1}{2} I_2 \dot{\theta}_{2DH}^2 \\ & + \frac{1}{2} I_3 (\dot{\theta}_{2OH} + \dot{\theta}_{3DH})^2 + \frac{1}{2} m_3 l_{c3}^2 (\dot{\theta}_{2DH}^2 + 2\dot{\theta}_{2OH}\dot{\theta}_{3OH} + \dot{\theta}_{3DH}^2) \\ & + m_3 l_2 l_{c3} \cos\theta_{3DH} \dot{\theta}_{2DH}^2 + m_3 l_2 l_{c3} \cos\theta_{3DH} \dot{\theta}_{2DH} \dot{\theta}_{3DH} \end{aligned}$$

$$\begin{aligned} K = {} & \frac{1}{2} \dot{\theta}_{2DH}^2 \cdot (m_2 l_{c2}^2 + m_3 l_2^2 + I_2) \\ & + \left(\frac{1}{2}\dot{\theta}_{2DH}^2 + \theta_{2OH}\cdot\theta_{3DH} + \frac{1}{2}\dot{\theta}_{3DH}^2\right) \cdot (I_3 + m_3 l_{c3}^2) \\ & + m_3 l_2 l_{c3} (\cos\theta_{3DH} \cdot \dot{\theta}_{2OH}^2 + \cos\theta_{3DH} \cdot \dot{\theta}_{2DH} \dot{\theta}_{3OH}) \end{aligned}$$

$\therefore$ Proved

$$\begin{aligned} K = {} & \frac{1}{2} \dot{\theta}_{2DH} P_1 + \left(\frac{1}{2}\dot{\theta}_{2DH}^2 + \theta_{2OH}\cdot\theta_{3DH} + \frac{1}{2}\dot{\theta}_{3DH}^2\right) \cdot P_2 \\ & + m_3 l_2 l_{c3} P_3 \end{aligned}$$

$$\begin{aligned} V &= -m_2 g \cdot l_{c2} \cdot \sin\theta_{2DH} - m_3 g \cdot (l_2 \sin\theta_{2DH} + l_{c3} \sin(\theta_{2OH}+\theta_{3DH})) \\ &= -(m_2 l_{c2} + m_3 l_2) g \sin\theta_{2OH} - m_3 l_{c3} \cdot g \sin(\theta_{2OH}+\theta_{3OH}) \\ &= -P_4 g \sin\theta_{2OH} - P_5 g \sin(\theta_{2DH}+\theta_{3DH}) \end{aligned}$$

3.2

3.2

$$\frac{\partial L}{\partial \dot{\theta}_{2DH}} = \frac{\partial (K-V)}{\partial \dot{\theta}_{2DH}} = P_1 \cdot \dot{\theta}_{2DH} + P_2 \dot{\theta}_{2DH} + P_2 \dot{\theta}_{3DH} + 2P_3 \cos\theta_{3DH} \cdot \dot{\theta}_{2DH}$$
$$+ P_3 \cos\theta_{3DH} \dot{\theta}_{3DH}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_{2DH}} = (P_1 + P_2 + 2P_3 \cos\theta_{3DH})\ddot{\theta}_{2DH} + (P_2 + P_3 \cos\theta_{3DH})\ddot{\theta}_{3DH}$$
$$- 2P_3 \dot{\theta}_{2DH} \cdot \sin\theta_{3DH} \cdot \dot{\theta}_{3DH} - P_3 \dot{\theta}_{3DH} \sin\theta_{3DH} \dot{\theta}_{3DH}$$

$$\frac{\partial L}{\partial \theta_{2DH}} = \frac{\partial (k-V)}{\partial \theta_{2DH}} = -P_4 g \cos\theta_{2DH} - P_5 g \cos(\theta_{2DH} + \theta_{3DH})$$

$$\therefore \tau_{M2DH} = \frac{d}{dt} \cdot \frac{\partial L}{\partial \dot{\theta}_{2DH}} - \frac{\partial L}{\partial \theta_{2DH}}$$
$$= (P_1 + P_2 + 2P_3 \cos\theta_{3DH})\ddot{\theta}_{2DH} + (P_2 + P_3 \cos\theta_{3DH})\ddot{\theta}_{3DH}$$
$$(-P_3 \dot{\theta}_{3DH} \cdot \sin\theta_{3DH} \cdot \dot{\theta}_{2DH}) + (-P_3 \dot{\theta}_{2DH} \cdot \sin\theta_{3DH} - P_3 \dot{\theta}_{3DH} \sin\theta_{3DH})\dot{\theta}_{3DH}$$
$$-P_4 g \cos\theta_{2DH} - P_5 g \cos(\theta_{2DH} + \theta_{3DH})$$

$$\frac{\partial L}{\partial \dot{\theta}_{3DH}} = P_2 \dot{\theta}_{2DH} + P_2 \dot{\theta}_{3DH} + P_3 \dot{\theta}_{2DH} \cos\theta_{3DH}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_{3DH}} = P_2 \ddot{\theta}_{2DH} + P_2 \ddot{\theta}_{3DH} + P_3 \cos\theta_{3DH} \ddot{\theta}_{2DH} + P_3 \dot{\theta}_{2DH} \sin\theta_{3DH} \dot{\theta}_{2DH}$$
$$= (P_2 + P_3 \cos\theta_{3DH})\ddot{\theta}_{2DH} + P_2 \ddot{\theta}_{3DH}$$
$$+ P_3 \sin\theta_{3DH} \cdot \dot{\theta}_{2DH}) \cdot \dot{\theta}_{2DH}$$

$$\frac{\partial L}{\partial \theta_{3DH}} = -P_5 g \cos(\theta_{2DH} + \theta_{3DH})$$

$$\because \tau_{M3DH} = \frac{d}{dt} \cdot \frac{\partial L}{\partial \dot{\theta}_{3DH}} - \frac{\partial L}{\partial \theta_{3DH}}$$

$$= (P_2 + P_3 \cos\theta_{3DH})\ddot{\theta}_{2DH} + P_2 \ddot{\theta}_{3DH}$$

$$+ (P_3 \sin\theta_{3DH} \cdot \dot{\theta}_{2DH}) \cdot \dot{\theta}_{2DH} + (-P_5 g \cos(\theta_{2DH} + \theta_{3DH}))$$

$\therefore$ Proved :

$$\begin{bmatrix} \tau_{M2DH} \\ \tau_{M3DH} \end{bmatrix} = \underbrace{\begin{bmatrix} P_1 + P_2 + 2P_3 \cos\theta_{3DH} & P_2 + P_3 \cos\theta_{3DH} \\ P_2 + P_3 \cos\theta_{3DH} & P_2 \end{bmatrix}}_{D(\theta)} \cdot \begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix} -P_3 \sin\theta_{3DH} \dot{\theta}_{3DH} & -P_3 \sin\theta_{3DH} \dot{\theta}_{3DH} - P_3 \sin\theta_{3DH} \dot{\theta}_{2DH} \\ P_3 \sin\theta_{3DH} \dot{\theta}_{2DH} & 0 \end{bmatrix}}_{C(\theta, \dot{\theta})} \cdot \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix} -P_4 g \cos\theta_{2DH} - P_5 g \cos(\theta_{2DH} + \theta_{3DH} \\ -P_5 g \cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix}}_{G(\theta)}$$

state space : $x_1 = \theta_{2DH}$, $x_2 = \dot{\theta}_{2DH} = \dot{x}_1$, $x_3 = \theta_{3DH}$, $x_4 = \dot{\theta}_{3DH} = \dot{x}_3$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_3 \\ \dot{x}_2 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \\ 0 & 0 & -D(\theta)^{-1} C(\theta, \dot{\theta}) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -D(\theta)^{-1} \end{bmatrix} \cdot \begin{bmatrix} \tau_{M2DH} \\ \tau_{M3DH} \end{bmatrix} - D(\theta)^{-1} G(\theta)$$

3.3

1.3. $\theta_{2DH} = \theta_{2M} - \frac{\pi}{2}$

$\theta_{3DH} = \theta_{3M} - \theta_{2M} + \frac{\pi}{2}$

$\tau_{2DH} = \tau_{2M} + \tau_{3M}$

$\tau_{3DH} = \tau_{3M}$

Based on 1.2

$$\tau_{3DH} = \left(P_2 + P_3 \cos\theta_{3DH}\right) \cdot \ddot{\theta}_{2DH} + P_2 \ddot{\theta}_{3DH}$$
$$+ P_3 \sin\theta_{3DH}\, \dot{\theta}_{2DH} \cdot \dot{\theta}_{2DH} - P_5 g \cos(\theta_{2DH} + \theta_{3DH}) = \tau_{3M}$$

$\cos\theta_{3DH} = \cos(\theta_{3M} - \theta_{2M} + \frac{\pi}{2}) = -\sin(\theta_{3M} - \theta_{2M})$

$\sin\theta_{3H} = \sin(\theta_{3M} - \theta_{2M} + \frac{\pi}{2}) = \cos(\theta_{3M} - \theta_{2M})$

$\cos(\theta_{2DH} + \theta_{3DH}) = \cos\theta_{3M}$

$$\therefore \tau_{3M} = \left[P_2 - P_3 \sin(\theta_{3M} - \theta_{2M})\right] \ddot{\theta}_{2M} + P_2(\ddot{\theta}_{3M} - \ddot{\theta}_{2M})$$
$$+ P_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{2M} \cdot \dot{\theta}_{2M} - P_5 g \cos\theta_{3M}$$
$$= -P_3 \sin(\theta_{3M} - \theta_{2M}) \cdot \ddot{\theta}_{2M} + P_2 \ddot{\theta}_{3M}$$
$$+ P_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{2M} \cdot \dot{\theta}_{2M} - P_5 g \cos\theta_{3M}$$

$$\tau_{2DH} = \left[P_1 + P_2 - 2P_3 \sin(\theta_{3M} - \theta_{2M})\right] \ddot{\theta}_{2M}$$
$$+ \left[P_2 - P_3 \sin(\theta_{3M} - \theta_{2M})\right](\ddot{\theta}_{3M} - \ddot{\theta}_{2M})$$

$$-2P_3 \cos(\theta_{3M} - \theta_{2M})(\dot{\theta}_{3M} - \dot{\theta}_{2M}) \cdot \dot{\theta}_{2M} - P_3 \cos(\theta_{3M} - \theta_{2M})(\dot{\theta}_{3M} - \dot{\theta}_{2M})^2$$

$$-P_4 g \cos(\theta_{2M} - \frac{\pi}{2}) - P_5 g \cos\theta_{3M}$$

$$Z_{2M} = Z_{2DH} - Z_{3M}$$

$$= P_1 \ddot{\theta}_{2M} - P_3 \sin(\theta_{3M} - \theta_{2M}) \ddot{\theta}_{3M}$$

$$+ P_3 \cos(\theta_{3M} - \theta_{2M}) \cdot \dot{\theta}_{2M} \cdot \dot{\theta}_{2M} - P_4 g \sin\theta_{2M}$$

$\therefore$ Proved

$$\begin{bmatrix} Z_{2M} \\ Z_{3M} \end{bmatrix} = \underbrace{\begin{bmatrix} P_1 & -P_3 \sin(\theta_{3M} - \theta_{2M}) \\ -P_3 \sin(\theta_{3M} - \theta_{2M}) & P_2 \end{bmatrix}}_{D(\theta)} \cdot \begin{bmatrix} \ddot{\theta}_{2M} \\ \ddot{\theta}_{3M} \end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix} 0 & -P_3 \cos(\theta_{3M} - \theta_{2M}) \dot{\theta}_{3M} \\ P_3 \cos(\theta_{3M} - \theta_{2M}) \cdot \dot{\theta}_{2M} & 0 \end{bmatrix}}_{C(\theta, \dot{\theta})} \cdot \begin{bmatrix} \dot{\theta}_{2M} \\ \dot{\theta}_{3M} \end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix} -P_4 g \sin\theta_{2M} \\ -P_5 g \cos\theta_{3M} \end{bmatrix}}_{G(\theta)}$$

state space: $x_1 = \theta_{2M}$, $x_2 = \dot{\theta}_{2M} = \dot{x}_1$, $x_3 = \theta_{3M}$, $x_4 = \dot{\theta}_{3M} = \dot{x}_3$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_3 \\ \dot{x}_2 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \\ 0 & 0 & -D(\theta)^{-1} C(\theta, \dot{\theta}) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \\ -D(\theta)^{-1} \end{bmatrix} \cdot \begin{bmatrix} Z_{2M} \\ Z_{3M} \end{bmatrix}$$

$$- D(\theta)^{-1} G(\theta)$$

4 Simulink Simulation Model

With the parameters from the lab manual, a Simulink model was created for a nonlinear simulation of equations. The nonlinearity was implemented using the MATLAB function block. To find qdot and q respectively, we used two integrators for the signal generated by the MATLAB function qdotdot.

4.1 Implementation of velocity for PID Controller

We had to find the velocity required for the PID controller explicitly by using the optical encodes to measure the angles at each of its joints and then using that information to find velocity using the fact that this reading is taken every 1ms. The following formula was used to calculate the raw velocity of each joint.

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

The raw velocity was filtered using the Second Method of Filtering Velocity as prescribed in the manual. This was done in order to eliminate the noise. To do so, we simply used the averaging filter, which met our requirements and filtered the velocity noise. The important thing to remember here is that this filter was called every 1 millisecond. To find the gains, the same implementation was done in Simulink before being done in Code Composer using the closed loop system.

4.1.1 Simulation and PD control of link 2 and link 3 of the CRS Robot

The requirements of PD Control strategy are to achieve a rise time of less than 300ms and a percent overshoot of less than 1%, with minimal steady state error.

The gains of PD control, which are Kp1, Kd1 and Kp2 and Kd2 for each joint were identified using the simulation model (Simulink Model) we just created as shown in Figure 1 while considering the requirements. The Simulink model followed the control inputs as τ2M and τ3M respectively.

$$\tau_{2M} = K_{P1} * \left(\theta_{2M}^{d} - \theta_{2M}\right) - K_{D1}\dot{\theta}_{2M}$$
$$\tau_{3M} = K_{P2} * \left(\theta_{3M}^{d} - \theta_{3M}\right) - K_{D2}\dot{\theta}_{3M}$$

The desired motor angles where step input from 0 to pi/6 radians from t = 0s to t = 1s, followed by a step back to 0 from t = 1 to t = 2s. We even had to take care of saturation of the torque values for each joint as each joint can have a maximum torque of 5Nm. The desired reference step back and forth between 0 radians and pi/6 radians was given to all three joints. And finally the Kp, Kd values for all the three joints were tuned using the Code Composer Studio's watch expression. Proportional term reduced the rise time and steady state error. But increased the overshoot. While the derivative gain was used to reduce overshoot. The value we found in Simulink was Kp1 = Kp2 = 50 and Kd1 = Kd2 = 2. And we used these as starting gains to tune using the Code Composer expression window and finally found the values to be Kp2 = 50, Kp3 = 50 and Kd2 = 1.7 and Kd3 = 1.4.

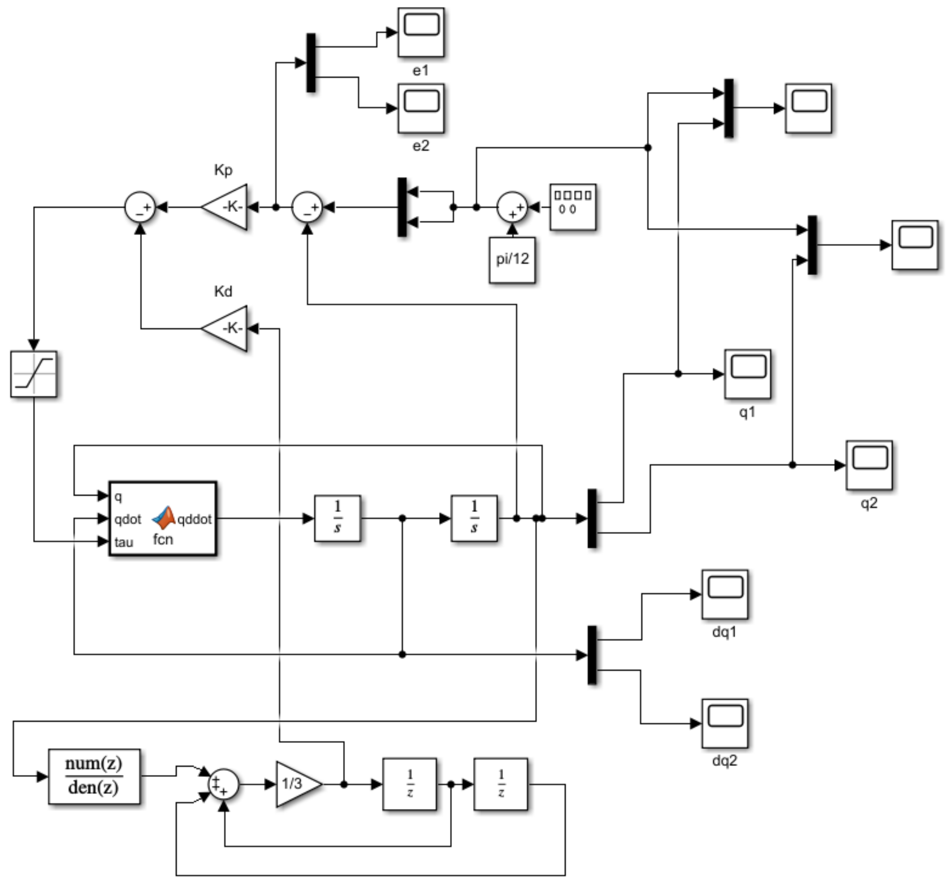


Figure 1. Simulink Model with Matlab Function and Averaging Filter

4.2 Controller Simulink Simulations

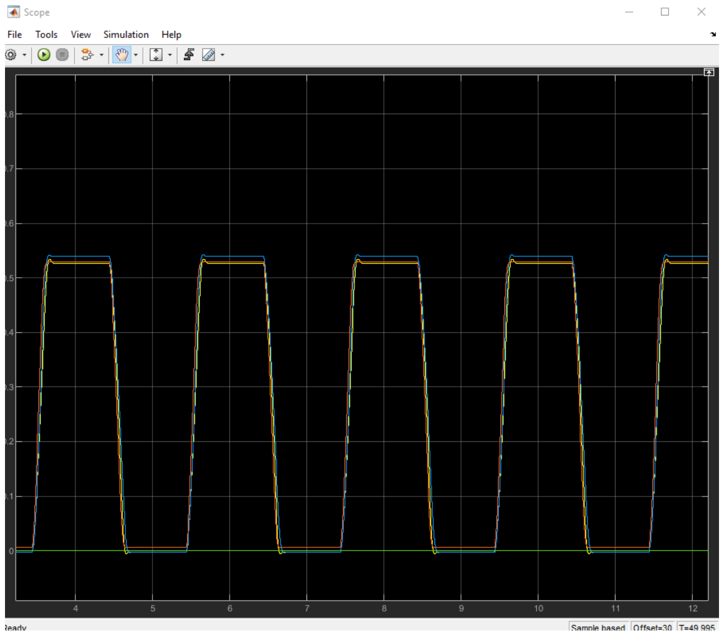
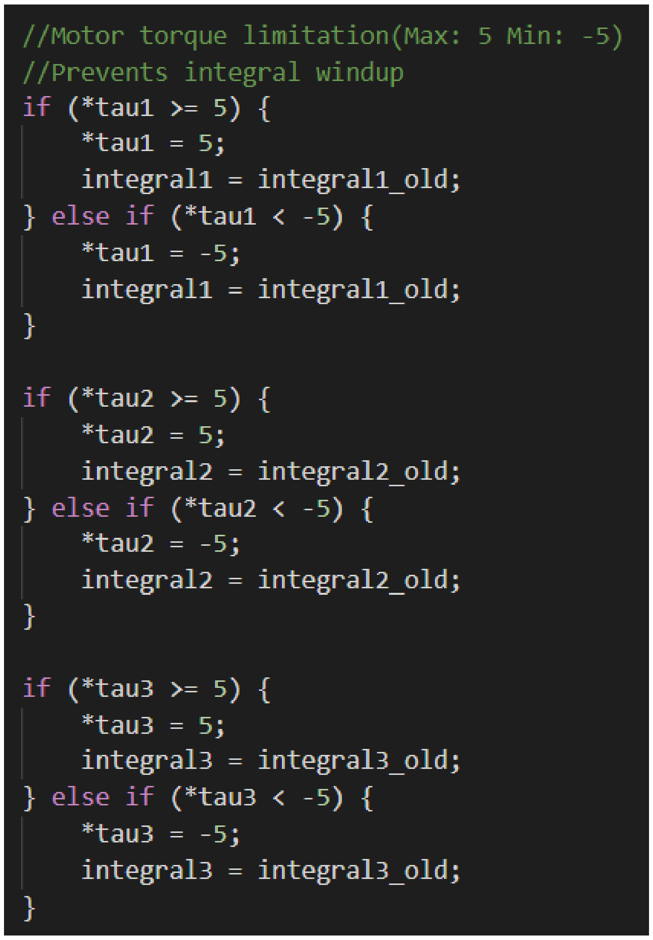


Figure 2. Step Response of System



```
//Motor torque limitation(Max: 5 Min: -5)
//Prevents integral windup
if (*tau1 >= 5) {
    *tau1 = 5;
    integral1 = integral1_old;
} else if (*tau1 < -5) {
    *tau1 = -5;
    integral1 = integral1_old;
}

if (*tau2 >= 5) {
    *tau2 = 5;
    integral2 = integral2_old;
} else if (*tau2 < -5) {
    *tau2 = -5;
    integral2 = integral2_old;
}

if (*tau3 >= 5) {
    *tau3 = 5;
    integral3 = integral3_old;
} else if (*tau3 < -5) {
    *tau3 = -5;
    integral3 = integral3_old;
}
```

Figure 3. Code Snippet to handle Torque Saturation and Integral Windup

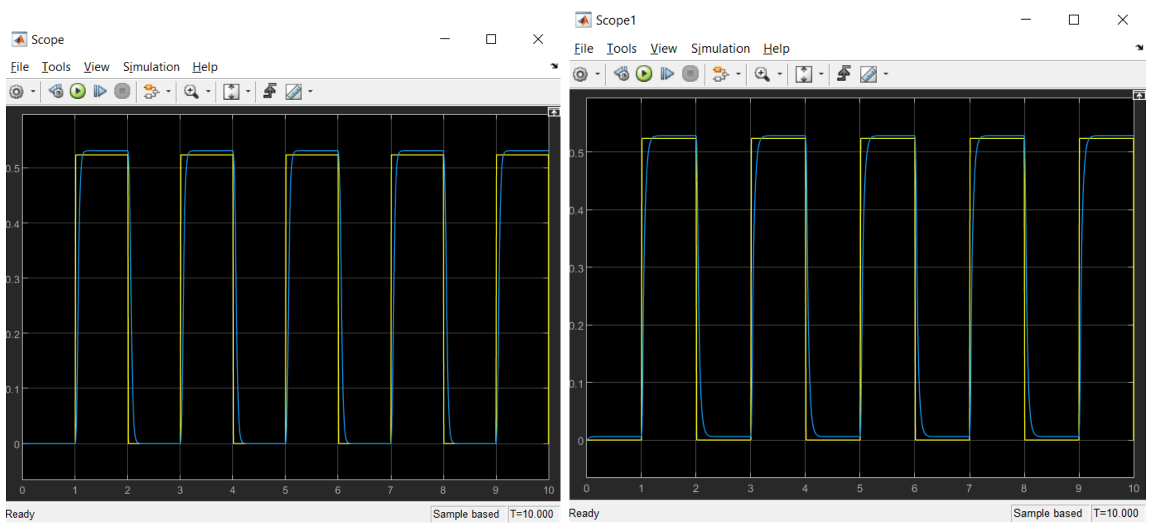


Figure 4. Response of Joint 2 and Joint 3 on Simulink Scope (Yellow: Reference, Blue: Output)

Error Plots:

$$e_1 = \theta_{2M}^d - \theta_{2M}, e_2 = \theta_{3M}^d - \theta_{3M}.$$

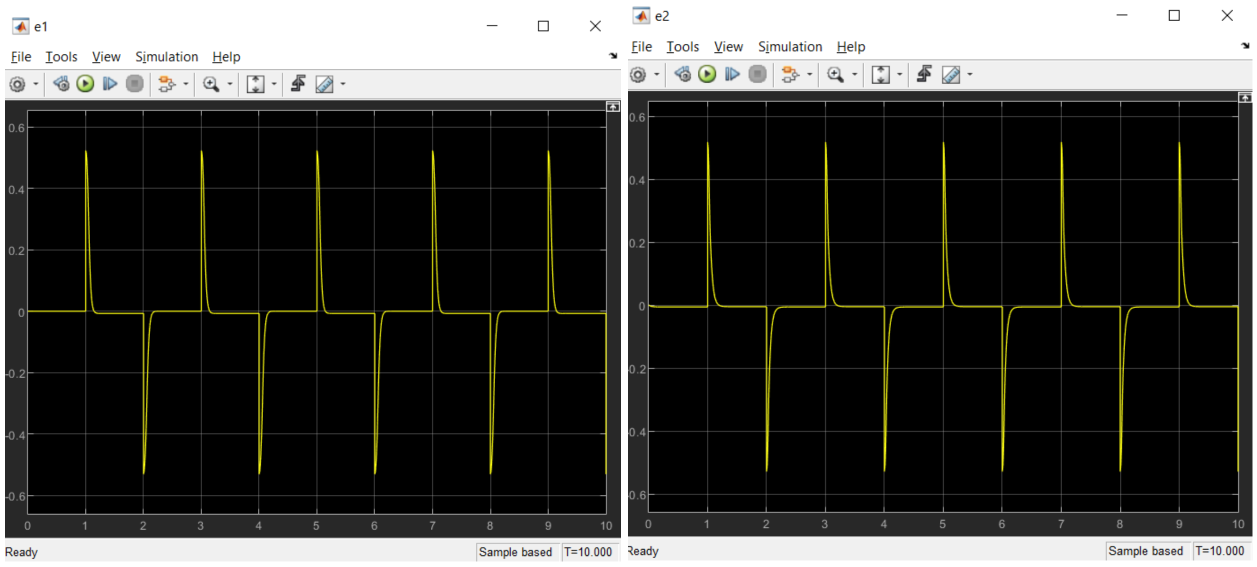
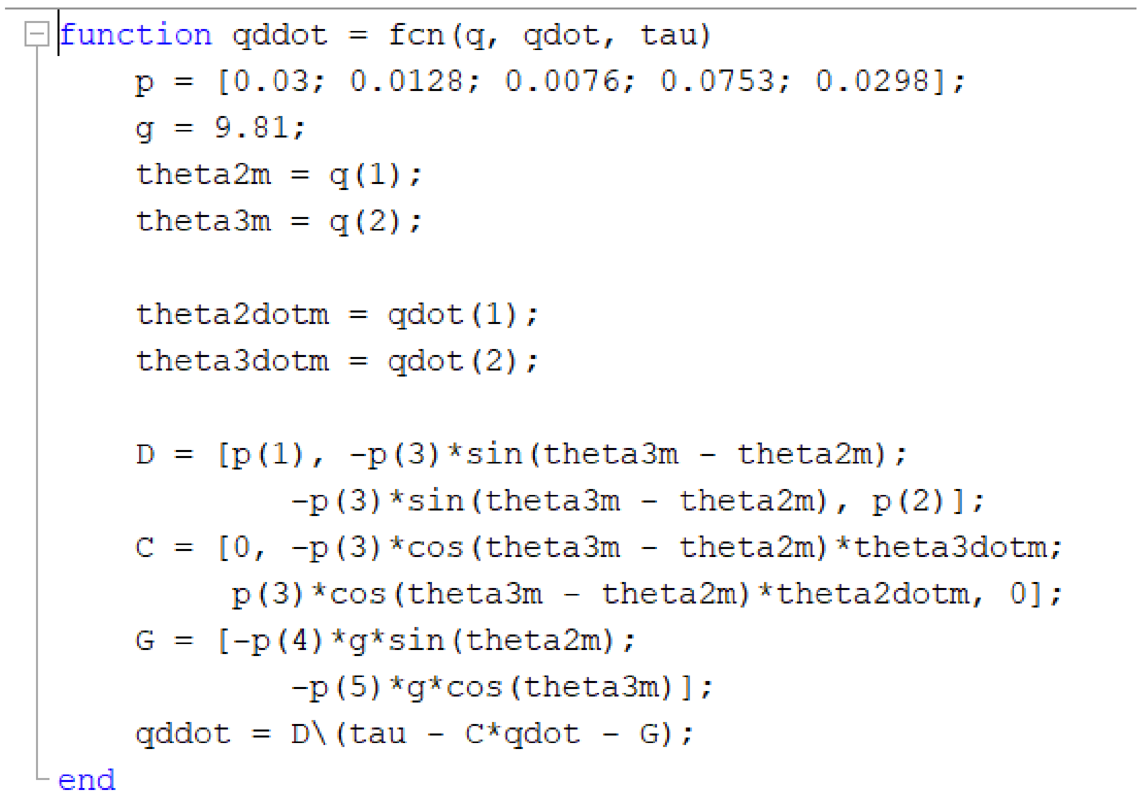


Figure 5. Error Plot of Joint 2 and Joint 3 on Simulink Scope



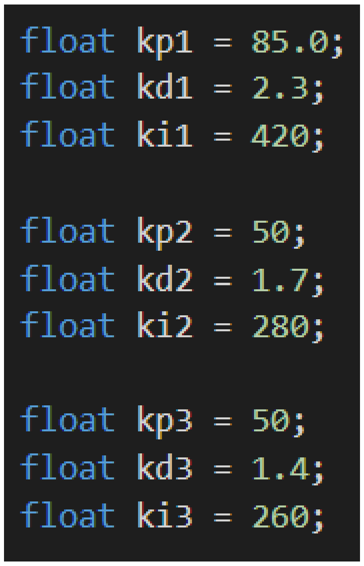
```
function qddot = fcn(q, qdot, tau)
    p = [0.03; 0.0128; 0.0076; 0.0753; 0.0298];
    g = 9.81;
    theta2m = q(1);
    theta3m = q(2);

    theta2dotm = qdot(1);
    theta3dotm = qdot(2);

    D = [p(1), -p(3)*sin(theta3m - theta2m);
            -p(3)*sin(theta3m - theta2m), p(2)];
    C = [0, -p(3)*cos(theta3m - theta2m)*theta3dotm;
         p(3)*cos(theta3m - theta2m)*theta2dotm, 0];
    G = [-p(4)*g*sin(theta2m);
            -p(5)*g*cos(theta3m)];
    qddot = D\(tau - C*qdot - G);
end
```

Figure 6. MATLAB Function Block used in Simulink Block Diagram

4.3 Implement Integral Control

Integral Control was added in the system to reduce the steady state error. The method used in the lab to find the integral term in PID control is with the trapezoidal approximation. The error signal is integrated using the trapezoidal rule at each new time step T. To take care of the integral windup, we turn on the integral term only when the error to desired position is small, this implementation can be seen in Figure 3. This is done by monitoring the torque command, where if the absolute value of torque is greater than 5, then we don't integrate and leave the integral value it had in the previous loop.

After tuning all the gains and checking for the response on Simulink scope, the final gains for Proportional, Integral and Derivative terms were as shown in the figure.



```
float kp1 = 85.0;
float kd1 = 2.3;
float ki1 = 420;

float kp2 = 50;
float kd2 = 1.7;
float ki2 = 280;

float kp3 = 50;
float kd3 = 1.4;
float ki3 = 260;
```

Figure 7. Kp, Ki, Kd values for each joint respectively after tuning

5 PID Plus Feedforward Control

$$\tau = J\ddot{\theta}^d + K_P(\theta^d - \theta) + K_I\int(\theta^d - \theta) + K_D(\dot{\theta}^d - \dot{\theta})$$

Once, the PID control was implemented we set the desired trajectory to a cubic polynomial trajectory to generate a trajectory from zero radians to 0.5 radians in one second and then 0.5 radians to 0 radians in the next one second. The coefficients were found in MATLAB as shown in Figure 8. This was followed by adding a feed forward control. It should be noted that the sign in the derivative term is positive now because there is the desired thetadot, in contrast to the case where thetadot was zero when doing a step response. Hence, it makes sense to add a positive sign in the Derivative term.

The coefficients of the cubic polynomial for the first half and second half are shown in the following Figure 8 respectively.

```
a1 = -1;          a2 = 1;
b1 = 1.5;         b2 = -4.5;
c1 = 0;           c2 = 6;
d1 = 0;           d2 = -2;
```

Figure 8. Coefficients of Cubic Polynomial

```
if (modeflag == 0)
    theta = a1*t^3+b1*t^2+c1*t+d1;
    theta_dot = 3*a1*t^2+2*b1*t+c1;
    theta_dotdot = 6*a1*t+2*b1;
elseif (modeflag == 1)
    theta = a2*t^3+b2*t^2+c2*t+d2;
    theta_dot = 3*a2*t^2+2*b2*t+c2;
    theta_dotdot = 6*a2*t+2*b2;
```

Figure 9. Polynomial equations for the first and second half of the trajectory

The equations hold true for first half, where modeflag is set to 0 and in the second half modeflag is set to 1. Figure 9 shows the plot of the desired trajectory and its first and second derivatives.

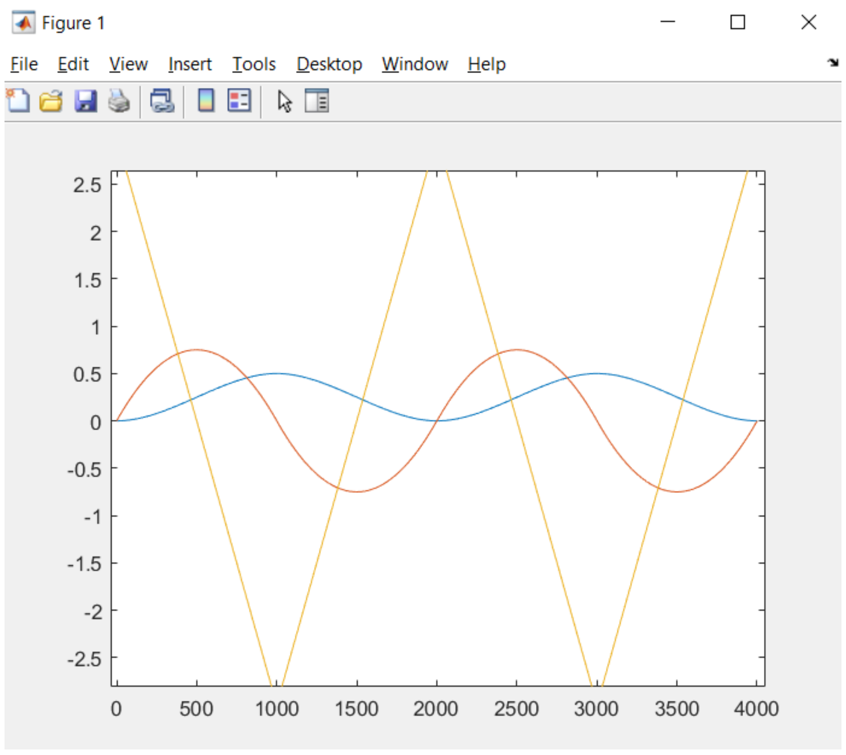


Figure 10. Plot of desired trajectory and its first and second derivatives

The cubic polynomial was implemented in the Code Composer Studio which returned the desired trajectory (Cubic Polynomial). The response of feed forward controller can be seen in Figure 11. We also noticed that the steady state error for step response of the system was zero because of the additional integral control as seen in Figure 11.

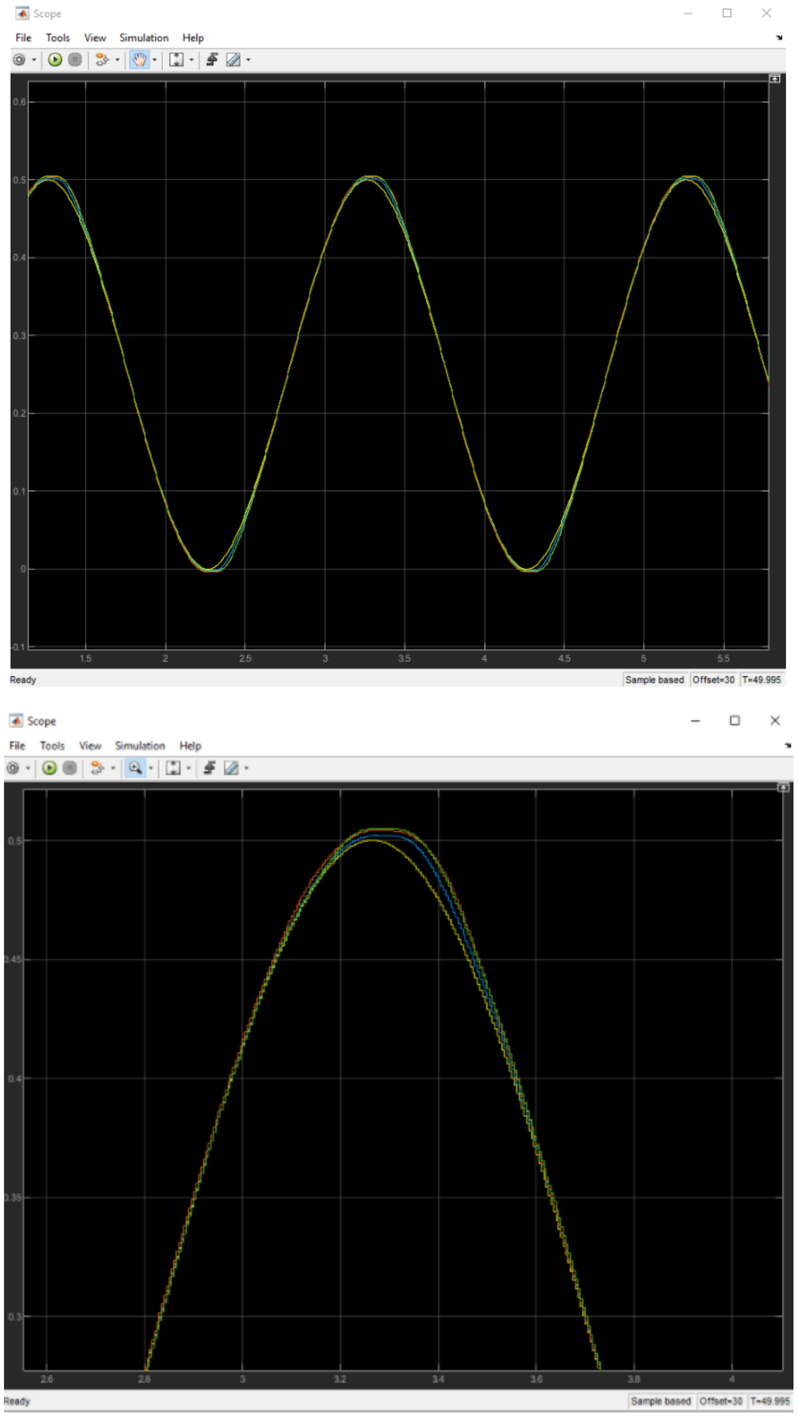


Figure 10. Plot of desired trajectory and its first and second derivatives

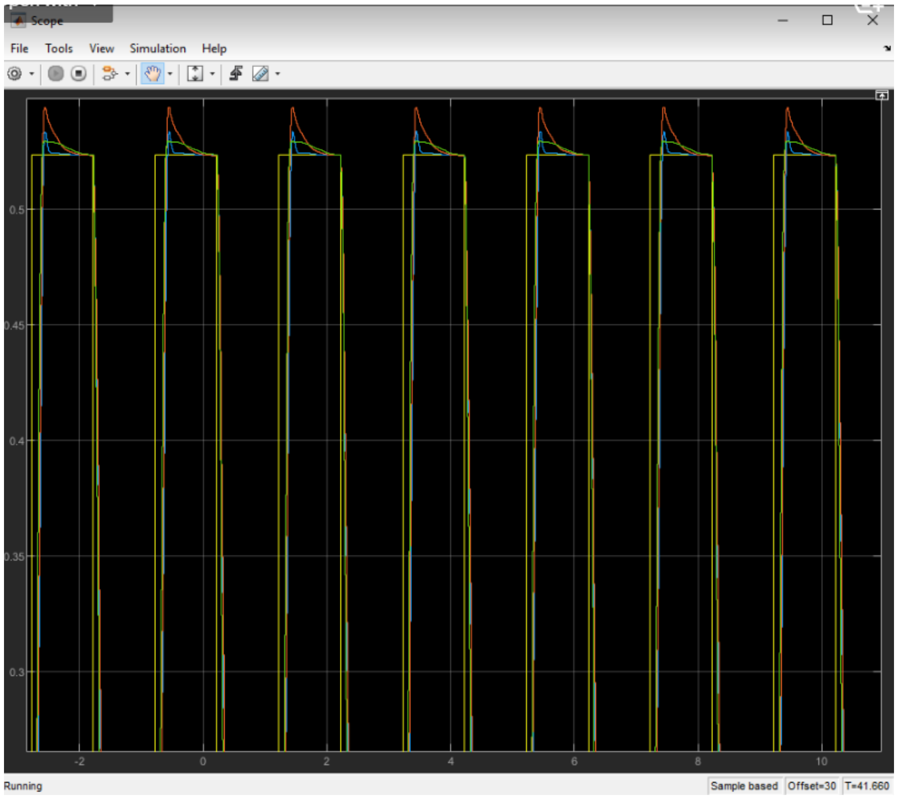
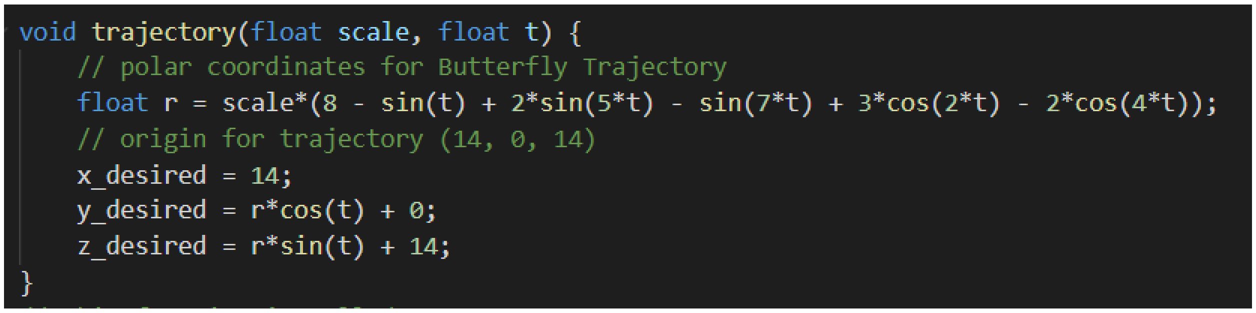


Figure 11. Response of the system for PID control (Yellow: Reference)

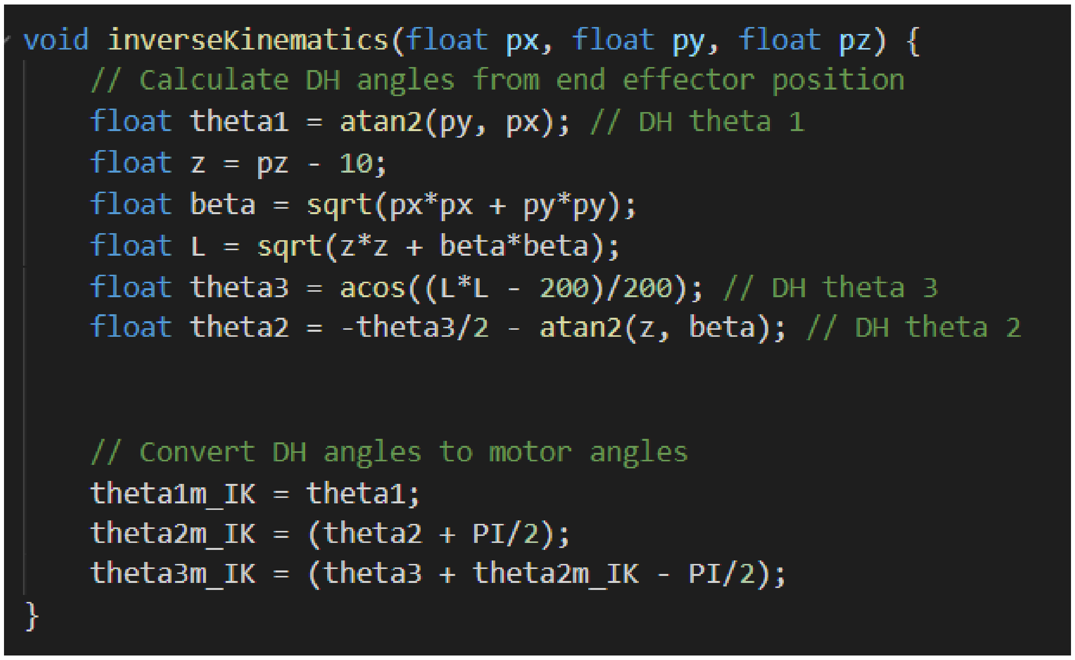
6 Fun Trajectory

The final exercise of this lab was to generate a Fun trajectory and we decided to do a butterfly trajectory. This is done by creating a function which updates the values of x,y,z and then the inverse kinematics(Figure 12) for these coordinates if found every 1ms. The butterfly trajectory was done using the following function as seen in Figure 11.



```
void trajectory(float scale, float t) {
    // polar coordinates for Butterfly Trajectory
    float r = scale*(8 - sin(t) + 2*sin(5*t) - sin(7*t) + 3*cos(2*t) - 2*cos(4*t));
    // origin for trajectory (14, 0, 14)
    x_desired = 14;
    y_desired = r*cos(t) + 0;
    z_desired = r*sin(t) + 14;
}
```

Figure 11. Butterfly trajectory function updating x,y,z values



```
void inverseKinematics(float px, float py, float pz) {
    // Calculate DH angles from end effector position
    float theta1 = atan2(py, px); // DH theta 1
    float z = pz - 10;
    float beta = sqrt(px*px + py*py);
    float L = sqrt(z*z + beta*beta);
    float theta3 = acos((L*L - 200)/200); // DH theta 3
    float theta2 = -theta3/2 - atan2(z, beta); // DH theta 2


    // Convert DH angles to motor angles
    theta1m_IK = theta1;
    theta2m_IK = (theta2 + PI/2);
    theta3m_IK = (theta3 + theta2m_IK - PI/2);
}
```

Figure 12. Inverse Kinematics Function used to update Theta Motor values

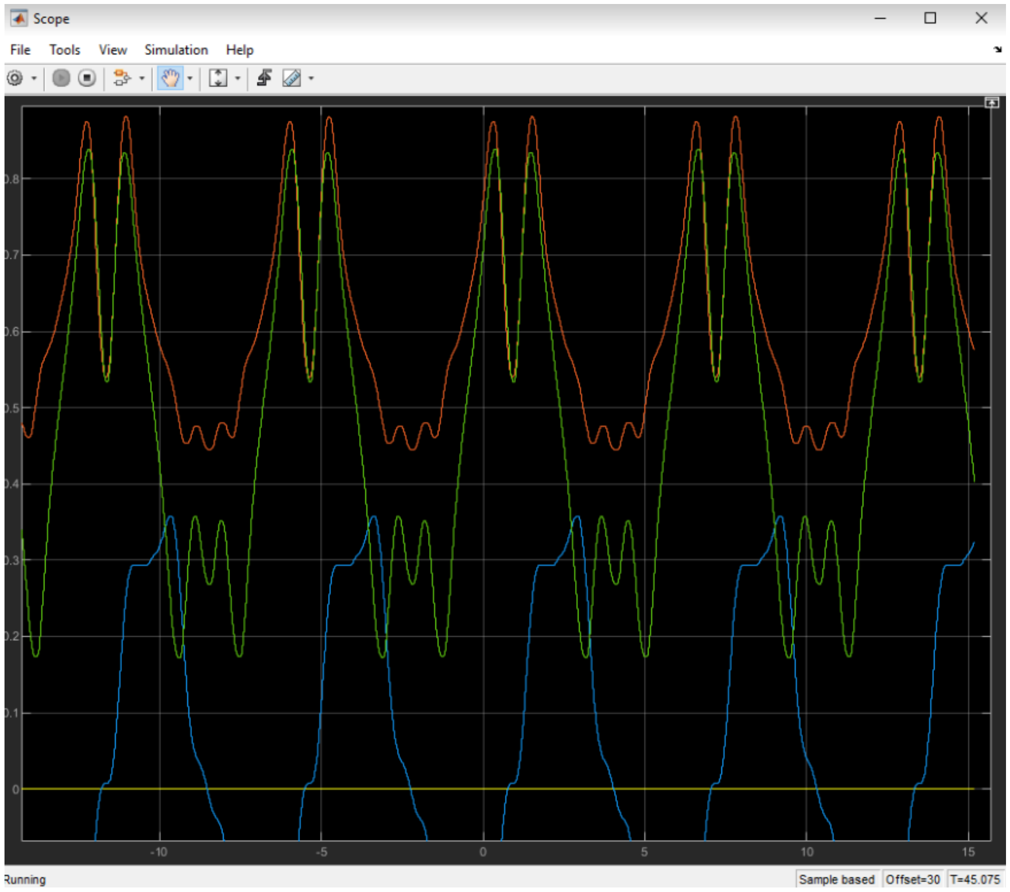


Figure 13. Response of the system for Feed Forward Control for Butterfly trajectory

7 Conclusion

Several control strategies, beginning with PD, were implemented in the lab, with the Integral term being used to reduce the steady state value. Both of these controllers were tuned to achieve the desired rise time of less than 300 milliseconds and a percent overshoot of less than 1% with minimal steady state error. Then we added a feed forward control that made the CRS Robot behave as if there was no friction. Then, to smooth out the trajectory, we used a cubic trajectory. We were able to combine all of the previous lessons into one final Fun trajectory module, in which we created a butterfly trajectory and used inverse kinematics to find the joints.

8 C Code

```
#include <tistdtypes.h>
#include <coecsl.h>
#include "user_includes.h"
#include "math.h"

// These two offsets are only used in the main file user_CRSRobot.c  You just need to create them here and
find the correct offset and then these offset will adjust the encoder readings
float offset_Enc2_rad = -0.415; //-0.37;
float offset_Enc3_rad = 0.233; //0.27;

// Your global variables.
float px = 0; // x coordinate of end effector
float py = 0; // y coordinate of end effector
float pz = 0; // z coordinate of end effector
float theta1m_IK = 0; // theta1 motor from inverse kinematics
float theta2m_IK = 0; // theta2 motor from inverse kinematics
float theta3m_IK = 0; // theta3 motor from inverse kinematics
long mycount = 0;
long counter = 0;
int cycle = 0;

// Omega calculations
float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;
float e1_old = 0;
float integral1_old = 0;

float Theta2_old = 0;
float Omega2_old1 = 0;
float Omega2_old2 = 0;
float Omega2 = 0;
float e2_old = 0;
float integral2_old = 0;

float Theta3_old = 0;
float Omega3_old1 = 0;
float Omega3_old2 = 0;
float Omega3 = 0;
float e3_old = 0;
float integral3_old = 0;


// Constants
float desired = 0;
float dt = 0.001;
float threshold1 = 0.01;
float threshold2 = 0.06;
float threshold3 = 0.02;

float kp1 = 85.0;
float kd1 = 2.3;
float ki1 = 420;

float kp2 = 50;
float kd2 = 1.7;
float ki2 = 280;

float kp3 = 50;
float kd3 = 1.4;
float ki3 = 260;

float theta_dotdot = 0.0;

float e1 = 0;
float e2 = 0;
float e3 = 0;

float x_desired = 0;
float y_desired = 0;
float z_desired = 0;

#pragma DATA_SECTION(whattoprint, ".my_vars")
float whattoprint = 0.0;

#pragma DATA_SECTION(whatnottoprint, ".my_vars")
float whatnottoprint = 0.0;

#pragma DATA_SECTION(theta1array, ".my_arrs")
float theta1array[100];

#pragma DATA_SECTION(theta2array, ".my_arrs")
float theta2array[100];

long arrayindex = 0;

float printtheta1motor = 0;
float printtheta2motor = 0;
float printtheta3motor = 0;

// Assign these float to the values you would like to plot in Simulink
float Simulink_PlotVar1 = 0;
float Simulink_PlotVar2 = 0;
float Simulink_PlotVar3 = 0;
float Simulink_PlotVar4 = 0;

void inverseKinematics(float px, float py, float pz) {
    // Calculate DH angles from end effector position
    float theta1 = atan2(py, px); // DH theta 1
    float z = pz - 10;
    float beta = sqrt(px*px + py*py);
    float L = sqrt(z*z + beta*beta);
    float theta3 = acos((L*L - 200)/200); // DH theta 3
    float theta2 = -theta3/2 - atan2(z, beta); // DH theta 2


    // Convert DH angles to motor angles
    theta1m_IK = theta1;
    theta2m_IK = (theta2 + PI/2);
    theta3m_IK = (theta3 + theta2m_IK - PI/2);
}

// Velocity estimation
void omega(float theta1motor, float theta2motor, float theta3motor) {
    Omega1 = (theta1motor - Theta1_old)/0.001;
    Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;

    Theta1_old = theta1motor;

    Omega1_old2 = Omega1_old1;
    Omega1_old1 = Omega1;

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

    Theta2_old = theta2motor;

    Omega2_old2 = Omega2_old1;
    Omega2_old1 = Omega2;

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

    Theta3_old = theta3motor;

    Omega3_old2 = Omega3_old1;
    Omega3_old1 = Omega3;
}

float cubic_func(float a1, float a2, float a3, float a4, float t) {
    return a1*t*t*t + a2*t*t + a3*t + a4;
}
float dot(float a1, float a2, float a3, float t) {
    return 3*a1*t*t + 2*a2*t + a3;
}
```

```
float doubledot(float a1, float a2, float t) {
    return 6*a1*t + 2*a2;
}

void trajectory(float scale, float t) {
    // polar coordinates for Butterfly Trajectory
    float r = scale*(8 - sin(t) + 2*sin(5*t) - sin(7*t) + 3*cos(2*t) - 2*cos(4*t));
    // origin for trajectory (14, 0, 14)
    x_desired = 14;
    y_desired = r*cos(t) + 0;
    z_desired = r*sin(t) + 14;
}
// This function is called every 1 ms
void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,float *tau2,float *tau3, int error) {
    // Forward Kinematics
    px = 10*cos(theta1motor)*(cos(theta3motor) + sin(theta2motor));
    py = 10*sin(theta1motor)*(cos(theta3motor) + sin(theta2motor));
    pz = 10*cos(theta2motor) - 10*sin(theta3motor) + 10;

    // Butterfly Trajectory
//    float time = counter*dt;
//    trajectory(0.5, time); // determine desired end-effector position
//    inverseKinematics(x_desired, y_desired, z_desired); // determine desired thetas

    // PID + Feedforward with Cubic Trajectory

    //
    if (counter % 2000 == 0) {
        counter = 0;
    }
    if (counter % 1000 == 0) {
        cycle++;
    }

    float time = counter*dt;
    float desired_doubledot = 0;
    float desired_dot = 0;
    if (cycle % 2 != 0) {
        // First cubic function
//        desired = cubic_func(-1, 1.5, 0, 0, time);
//        desired_dot = dot(-1, 1.5, 0, time);
//        desired_doubledot = doubledot(-1, 1.5, time);
//
        desired = 0;
        desired_dot = 0;
        desired_doubledot = 0;
    } else {
        // Second cubic function
//        desired = cubic_func(1, -4.5, 6, -2, time);
//        desired_dot = dot(1, -4.5, 6, time);
//        desired_doubledot = doubledot(1, -4.5, time);
        desired = PI/6;
        desired_dot = 0;
        desired_doubledot = 0;

    }

    // Calculate omegas
    omega(theta1motor, theta2motor, theta3motor);

  // Desired theta - current theta
    float e1 = desired - theta1motor;
    float e2 = desired - theta2motor;
    float e3 = desired - theta3motor;


//    e1 = theta1m_IK - theta1motor;
//    e2 = theta2m_IK - theta2motor;
//    e3 = theta3m_IK - theta3motor;

    // Integral approximation
    float integral1 = integral1_old + (e1 + e1_old) * dt / 2;
    float integral2 = integral2_old + (e2 + e2_old) * dt / 2;
    float integral3 = integral3_old + (e3 + e3_old) * dt / 2;

    // Prevents integral windup
    if (fabs(e1) > threshold1) {
        integral1 = 0;
        integral1_old = 0;
    }

    if (fabs(e2) > threshold2) {
            integral2 = 0;
            integral2_old  = 0;
    }

    if (fabs(e3) > threshold3) {
            integral3 = 0;
            integral3_old = 0;
    }
    // Desired theta_dot and theta_dot_dot for Butterfly Trajectory
//    float desired_dot = 0;
//    float desired_doubledot = 0;

    // Feedforward + PID Controller
            *tau1 = 0.0167 * desired_doubledot + kp1 * e1 + kd1 * (desired_dot - Omega1) + ki1 * integral1;
            *tau2 = 0.03 * desired_doubledot + kp2 * e2 + kd2 * (desired_dot - Omega2) + ki2 * integral2;
            *tau3 = 0.0128 * desired_doubledot + kp3 * e3 + kd3 * (desired_dot - Omega3)+ ki3 * integral3;

            //Motor torque limitation(Max: 5 Min: -5)
            //Prevents integral windup
            if (*tau1 >= 5) {
                *tau1 = 5;
                integral1 = integral1_old;
            } else if (*tau1 < -5) {
                *tau1 = -5;
```

```
			integral1 = integral1_old;
		}

    if (*tau2 >= 5) {
        *tau2 = 5;
        integral2 = integral2_old;
    } else if (*tau2 < -5) {
        *tau2 = -5;
        integral2 = integral2_old;
    }

    if (*tau3 >= 5) {
        *tau3 = 5;
        integral3 = integral3_old;
    } else if (*tau3 < -5) {
        *tau3 = -5;
        integral3 = integral3_old;
    }

    e1_old = e1;
    e2_old = e2;
    e3_old = e3;
    integral1_old = integral1;
    integral2_old = integral2;
    integral3_old = integral3;

		// save past states
		if ((mycount%50)==0) {

			theta1array[arrayindex] = theta1motor;
            theta2array[arrayindex] = theta2motor;

			if (arrayindex >= 100) {
				arrayindex = 0;
			} else {
				arrayindex++;
			}

		}

		if ((mycount%500)==0) {
			if (whattoprint > 0.5) {
				serial_printf(&SerialA, "I love robotics\n\r");
			} else {
				printtheta1motor = theta1motor;
				printtheta2motor = theta2motor;
				printtheta3motor = theta3motor;
				SWI_post(&SWI_printf); //Using a SWI to fix SPI issue from sending too
many floats.
			}

			GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Blink LED on Control Card
			GpioDataRegs.GPBTOGGLE.bit.GPIO60 = 1; // Blink LED on Emergency Stop Box
```

```
            }


            Simulink_PlotVar1 = desired; //yellow
            Simulink_PlotVar2 = theta1motor; //blue
            Simulink_PlotVar3 = theta2motor; //orange
            Simulink_PlotVar4 = theta3motor; //green

            mycount++;
    counter++;
}

void printing(void){
    // Printing (theta1, theta2, theta3, px, py, pz) and then on a new line (theta1_IK, theta2_IK, theta3_IK)
            serial_printf(&SerialA, "px: %.2f, py: %.2f, pz: %.2f \n\r", px, py, pz);
}
```