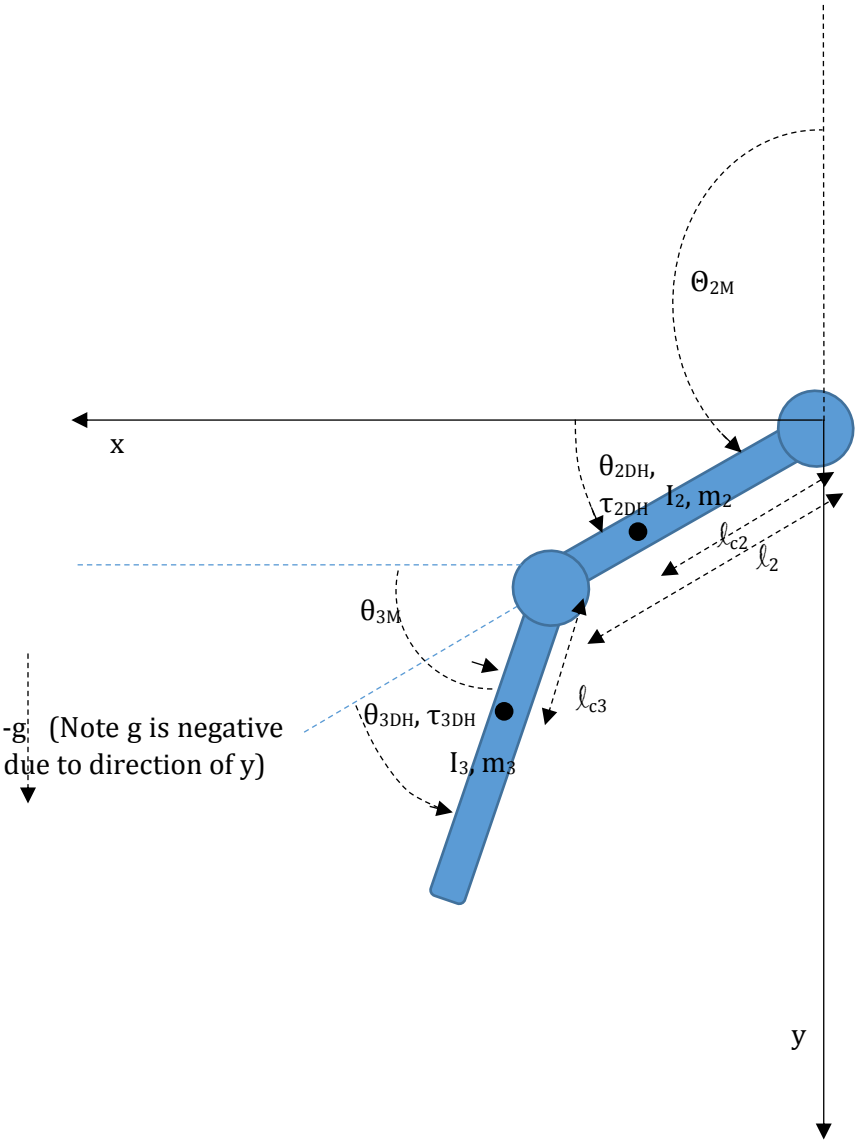
ME 446 Lab 2 Report

Diyu Yang dyang37
Dongbo Wang dwang49

Instructor: Daniel Block
Section: Wed. 2-5pm

Contents

Part 1: Equations of Motion of Two Link Planar Arm



1.1 Energy Equations

- **Kinetic Energy**

For each degree of freedom (each link), the kinetic energy can be expressed as:

$K_n = 1/2 I_n \omega_n^2 + 1/2 m_n v_n^2$, where n = 2, 3 for the 2 links respectively.

For link 2, we can express linear velocity as: $v_2 = l_{c2}\dot{\theta}_{2DH}$. The angular velocity is simply $\dot{\theta}_{2DH}$. So the kinetic energy of link 2 can be expressed as:

$$K_2 = 1/2 I_2 \omega \dot{\theta}_{2DH}^2 + 1/2 m_2 l_{c2}^2 \dot{\theta}_{2DH}^2 \quad (1)$$

For link 3, since the two links are rotating in the same plane, we can express angular velocity as follows:

$$\omega_3 = l_2 \dot{\theta}_{2DH} + l_{c3}\dot{\theta}_{3DH}$$

(2)

As for the linear velocity, we first express the position of center of mass of link 3 in base frame as follows:

$$\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} l_2 \sin(\theta_{2DH}) + l_{c3}\sin(\theta_{3M}) \\ l_3 \cos(\theta_{2DH}) + l_{c3}\cos(\theta_{3M}) \end{bmatrix}$$

Taking the derivative of the position vector w.r.t time, we will obtain the velocity vector as follows:

$$\vec{v}_3 = \begin{bmatrix} \dot{x}_3 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} l_2 \cos(\theta_{2DH}) + l_{c3}\cos(\theta_{3M}) \\ -l_3 \sin(\theta_{2DH}) - l_{c3}\sin(\theta_{3M}) \end{bmatrix} \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3M} \end{bmatrix}$$

Taking the norm, we get:

$$v_3^2 = \| \vec{v}_3 \|^2 = l_2^2 \dot{\theta}_{2DH}^2 + l_{c3}^2 \dot{\theta}_{3M}^2 + 2 l_2 l_{c3} \dot{\theta}_{2DH} \dot{\theta}_{3M} \cos(\theta_{3DH})$$

(3)

Where $\theta_{3M} = \theta_{2DH} + \theta_{3DH}$.

Taking the derivative w.r.t. time, we get: $\dot{\theta}_{3M} = \dot{\theta}_{2DH} + \dot{\theta}_{3DH}$.

As a result, after plugging in (2) and (3), we can express kinetic energy of link3 as follows:

$$K_3 = 1/2 I_3 (l_2 \dot{\theta}_{2DH} + l_{c3}\dot{\theta}_{3DH})^2 + 1/2 m_3 \left(l_2^2 \dot{\theta}_{2DH}^2 + l_{c3}^2 \dot{\theta}_{3M}^2 + 2 l_2 l_{c3} \dot{\theta}_{2DH} \dot{\theta}_{3M} \cos(\theta_{3DH}) \right)$$

(4)

Substituting $\dot{\theta}_{3M}$ with $\dot{\theta}_{2DH} + \dot{\theta}_{3DH}$, we get the following result for K_3:

$$K_3 = 1/2 I_3 (l_2 \dot{\theta}_{2DH} + l_{c3}\dot{\theta}_{3DH})^2 + 1/2 m_3 \left(l_2^2 \dot{\theta}_{2DH}^2 + l_{c3}^2 \left(\dot{\theta}_{2DH} + \dot{\theta}_{3DH} \right)^2 + 2 l_2 l_{c3} \dot{\theta}_{2DH} \left(\dot{\theta}_{2DH} + \dot{\theta}_{3DH} \right) \cos(\theta_{3DH}) \right) \quad (5)$$

Summing up equations (1) and (5), we get the desired result:

$$K = \tfrac{1}{2}\dot{\theta}_{2DH}^2 p_1 + (\tfrac{1}{2}\dot{\theta}_{2DH}^2 + \dot{\theta}_{2DH}\dot{\theta}_{3DH} + \tfrac{1}{2}\dot{\theta}_{3DH}^2) p_2 + (\cos\theta_{3DH}\dot{\theta}_{2DH}^2 + \cos\theta_{3DH}\dot{\theta}_{2DH}\dot{\theta}_{3DH}) p_3$$

Where

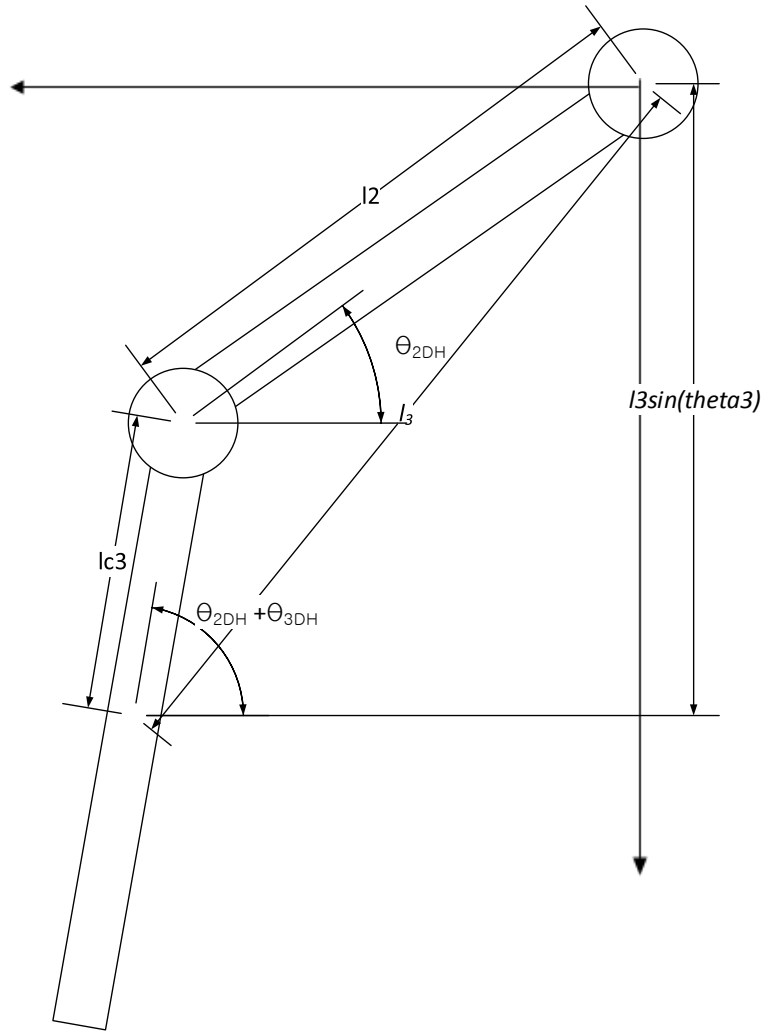
$$p_1 = m_2 l_{c2}^2 + m_3 l_2^2 + I_2$$
$$p_2 = m_3 l_{c3}^2 + I_3$$
$$p_3 = m_3 l_2 l_{c3}$$

- **Potential Energy**

  The potential energy of each link can be simply expressed as the following equation:

  $V = -mgl\sin\theta$, where l is the distance from the origin of base frame to the center of mass, and θ is the angle between the line connecting the origin to the center of mass and x axis of base frame.

  As a result, for link 2, $V_2 = -m_2 g l_{c2}\sin\theta_{2DH}$



From the graph above, we can see that for link 3, $l_3 \sin\theta_3 = l_2 \sin\theta_{2DH} + l_{c3} \sin(\theta_{2DH} + \theta_{3DH})$

As a result, $V_3 = -m_3 g \left(l_2 \sin\theta_{2DH} + l_{c3} \sin(\theta_{2DH} + \theta_{3DH}) \right)$

So total potential energy could be expressed as follows:

$$V = -p_4 g \sin\theta_{2DH} - p_5 g \sin(\theta_{2DH} + \theta_{3DH})$$

Where

$$p_4 = m_2 l_{c2} + m_3 l_2$$
$$p_5 = m_3 l_{c3}$$

1.2 Dynamic Equations in terms of DH Thetas.

The Euler-Lagrange Equations for torque calculation is

$$\begin{cases} L = K - V \\ \dfrac{d}{dt}\dfrac{\partial L}{\partial \dot{q}_i} - \dfrac{\partial L}{\partial q_i} = \tau_i \end{cases}$$

Substitute in K and V found in previous section, we have

$$\begin{cases} \dfrac{\partial L}{\partial \theta_{2DH}} = p_4 g \cos\theta_{2DH} + p_5 g \cos(\theta_{2DH} + \theta_{3DH}) \\ \dfrac{d}{dt}\dfrac{\partial L}{\partial \dot{\theta}_{2DH}} = p_1 \ddot{\theta}_{2DH} + p_2 \ddot{\theta}_{2DH} + p_2 \ddot{\theta}_{3DH} + 2 p_3 \cos\theta_{3DH} \ddot{\theta}_{2DH} - 2 p_3 \sin\theta_{3DH} \dot{\theta}_{3DH} \dot{\theta}_{2DH} \\ \qquad\qquad + p_3 \cos\theta_{3DH} \ddot{\theta}_{3DH} - p_3 \sin\theta_{3DH} \dot{\theta}_{3DH} \dot{\theta}_{3DH} \end{cases}$$

$$\begin{cases} \dfrac{\partial L}{\partial \theta_{3DH}} = -p_3 \sin\theta_{3DH} \dot{\theta}_{2DH}^2 - p_3 \sin\theta_{3DH} \dot{\theta}_{3DH} \dot{\theta}_{2DH} + p_5 g \cos(\theta_{2DH} + \theta_{3DH}) \\ \dfrac{d}{dt}\dfrac{\partial L}{\partial \dot{\theta}_{2DH}} = p_2 \ddot{\theta}_{2DH} + p_2 \ddot{\theta}_{3DH} + p_3 \cos\theta_{3DH} \ddot{\theta}_{2DH} - p_3 \sin\theta_{3DH} \dot{\theta}_{3DH} \dot{\theta}_{2DH} \end{cases}$$

The torques are

$$\begin{cases} \tau_2 = (p_1 + p_2 + 2 p_3 \cos\theta_{3DH}) \ddot{\theta}_{2DH} + (p_2 + p_3 \cos\theta_{3DH}) \ddot{\theta}_{3DH} + (-p_3 \sin\theta_{3DH} \dot{\theta}_{3DH}) \dot{\theta}_{2DH} \\ \quad + (-p_3 \sin\theta_{3DH} \dot{\theta}_{2DH} - p_3 \sin\theta_{3DH} \dot{\theta}_{3DH}) \dot{\theta}_{3DH} + (-p_4 g \cos\theta_{2DH} - p_5 g \cos(\theta_{2DH} + \theta_{3DH})) \\ \tau_3 = (p_2 + p_3 \cos\theta_{3DH}) \ddot{\theta}_{2DH} + (p_2) \ddot{\theta}_{3DH} + (p_3 \sin\theta_{3DH} \dot{\theta}_{2DH}) \dot{\theta}_{2DH} + (-p_5 g \cos(\theta_{2DH} + \theta_{3DH})) \end{cases}$$

Put into state-space matrix form, the equations of motion for this linkage are

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

With

$$\theta = \begin{bmatrix} \theta_{2DH} \\ \theta_{3DH} \end{bmatrix}, \dot{\theta} = \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}, \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix}, \tau = \begin{bmatrix} \tau_{M2DH} \\ \tau_{M3DH} \end{bmatrix}$$

And

$$D(\theta) = \begin{bmatrix} p_1 + p_2 + 2p_3\cos\theta_{3DH} & p_2 + p_3\cos\theta_{3DH} \\ p_2 + p_3\cos\theta_{3DH} & p_2 \end{bmatrix},$$

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

$$g(\theta) = \begin{bmatrix} -p_4 g\cos\theta_{2DH} - p_5 g\cos(\theta_{2DH} + \theta_{3DH}) \\ -p_5 g\cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix}$$

And finally in state space form can be written: (for serial linkages D is always invertible)

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

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

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

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

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

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

1.3 Dynamic Equations in terms of Motor Thetas

The relationship between DH thetas and motor thetas is

$$\theta_{2DH} = \theta_{2M} - {}^{\pi}/_{2}$$

$$\theta_{3DH} = \theta_{3M} - \theta_{2M} + {}^{\pi}/_{2}$$

Also since τ_{3M} acts relative to the same base as τ_{2M} we have the relationship that

$$\begin{cases} \tau_{2DH} = \tau_{2M} + \tau_{3M} \\ \tau_{3DH} = \tau_{3M} \end{cases} \Rightarrow \begin{cases} \tau_{2M} = \tau_{2DH} - \tau_{3DH} \\ \tau_{3M} = \tau_{3DH} \end{cases}$$

Substitute this relationship in to the torque equations

$$\begin{cases} \tau_{2M} = (p_2 + p_3\cos\theta_{3DH})\ddot{\theta}_{2DH} + (p_3\cos\theta_{3DH})\ddot{\theta}_{3DH} - p_3\sin\theta_{3DH}(\dot{\theta}_{2DH} + \dot{\theta}_{3DH})^2 + (-p_4 g\cos\theta_{2DH}) \\ \tau_{3M} = (p_2 + p_3\cos\theta_{3DH})\ddot{\theta}_{2DH} + (p_2)\ddot{\theta}_{3DH} + (p_3\sin\theta_{3DH}\dot{\theta}_{2DH})\dot{\theta}_{2DH} + (-p_5 g\cos(\theta_{2DH} + \theta_{3DH})) \end{cases}$$

Since

$$\begin{cases} \sin\theta_{3DH} = \sin(\theta_{3M} - \theta_{2M} + \dfrac{\pi}{2}) = \cos(\theta_{3M} - \theta_{2M}) \\ \cos\theta_{3DH} = \cos(\theta_{3M} - \theta_{2M} + \dfrac{\pi}{2}) = -\sin(\theta_{3M} - \theta_{2M}) \end{cases}$$

Therefore

$$\begin{cases} \tau_{2M} = (p_2 - p_3\sin(\theta_{3M} - \theta_{2M}))\ddot{\theta}_{2M} + (-p_3\sin(\theta_{3M} - \theta_{2M}))\ddot{\theta}_{3M} - p_3\cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{3M}^2 + (-p_4 g\sin\theta_{2M}) \\ \tau_{3M} = (p_2 - p_3\sin(\theta_{3M} - \theta_{2M}))\ddot{\theta}_{2M} + (p_2)\ddot{\theta}_{3M} + p_3\cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{2M}^2 + (-p_5 g\cos\theta_{2M}) \end{cases}$$

Put into state-space matrix form, the equations of motion for this linkage are

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

With

$$\theta = \begin{bmatrix} \theta_{2M} \\ \theta_{3M} \end{bmatrix}, \dot{\theta} = \begin{bmatrix} \dot{\theta}_{2M} \\ \dot{\theta}_{3M} \end{bmatrix}, \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_{2M} \\ \ddot{\theta}_{3M} \end{bmatrix}, \tau = \begin{bmatrix} \tau_{M2} \\ \tau_{M3} \end{bmatrix}$$

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

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

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

And finally in state space form can be written: (for serial linkages D is always invertible)

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

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

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

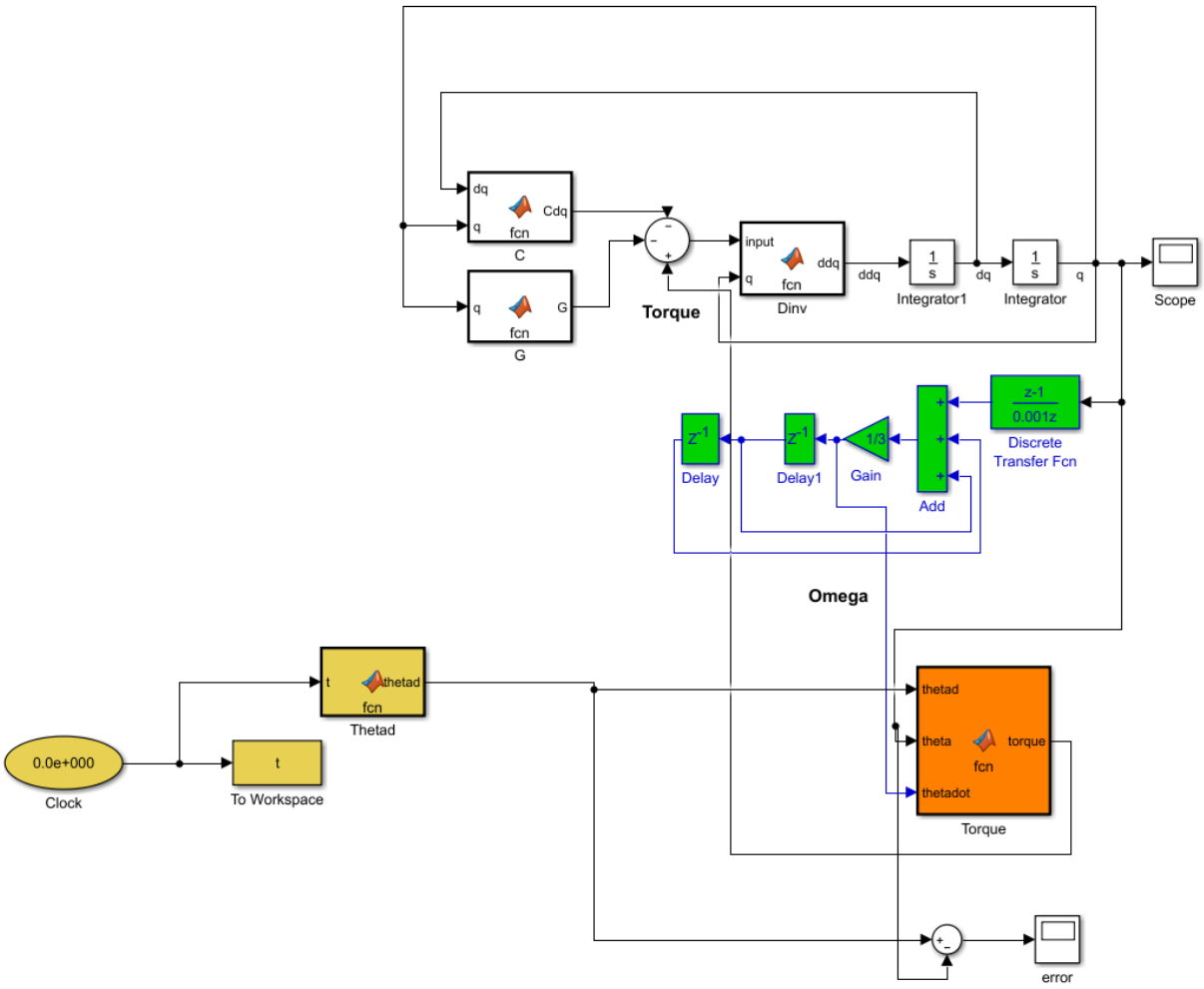
$$\dot{x}_2 = \ddot{\theta}_{2M}$$

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

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

Part 2: Simulation

Our Simulink model is as follows:



Detailed comments and explanations of each part of the simulation blocks are presented in the following sections:

Calculate and Implement the Velocity and Integration Components of the PID Controller:

The green blocks uses infinite average method to approximate $\dot{\theta} = \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$.

Generating step input:

The yellow blocks generate our step input to the system, where the function thetad is generated using the following code:

```
function thetad = fcn(t)
```

```
    thetad = [0 ; 0];
    thetad(1,1) = (t>0).*(t<2).*pi/6;
    thetad(2,1) = (t>0).*(t<2).*pi/6;
end
```

Where the variable t is from the clock. So basically our input rises to $\pi/6$ at time t = 0s, and falls back to 0 after t = 2s. (In **Part 4.1 Implementation of Feedforward PD Controller**, we replaced the input above with a smoother function that generates cubic polynomial trajectory. Details will be presented in later sections.)

PD controllers for control inputs

The orange block simulates PD controllers that control motor 2 and 3. Details about tuning PD parameters and simulation of PD control can be found in **Part 4**.

Nonlinear simulation of the equations of motion:

The white blocks are the actual simulation part of the equations of motion we derived in part I.

This part simulates the equation derived in part I:

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

Rewriting the equation above, we get the following equation:

$$D(\theta)\begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix} = \tau - C(\theta,\dot{\theta})\dot{\theta} - g(\theta)$$

(6)

Our system takes the torque generated by PD control as input. The function block "G" calculates $g(\theta)$, function block "C" calculates $C(\theta,\dot{\theta})\dot{\theta}$. The adder takes torque, output from "G", and output from "C" as input, calculates R.H.S. of equation (6) and output the result into block "Dinv". According to equation (6), the output of the adder should also be equal to $D(\theta)\begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix}$. Thus what "Dinv" does is simply taking the result from the output and multiply it with $D^{-1}(\theta)$. The output of Dinv therefore, should be simply $\begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix}$.

Code for block "Dinv", "C", and "G" can be found in Appendix A. The result of simulation is shown in Part 4.

Part 3: Velocity and Integration Components of the PID Controller.

3.1 How to find velocity.

This section presented three different ways of estimating angular velocities using angles. The C code for First two method of averaging filter can be found in Appendix B.

The basic algorithm for both methods is first using $\dot{\theta} = \frac{\theta_{current} - \theta_{previous}}{T}$ to calculate the raw velocity *Omega1_raw*, where $T = 1ms$, and then apply an average filter to *Omega1_raw* by taking the average value of *Omega1_raw*, *Omega1_old1,* and *Omega1_old2*, where the latter two are velocities in previous sample time *t-T* and *t-2T*.

The only difference between the two methods is that first method is using the raw velocity *Omega1_raw* as previous velocity when applying averaging filter, while the second method is inputting the filtered velocity *Omega1* instead of *Omega1_raw* into the averaging filter.

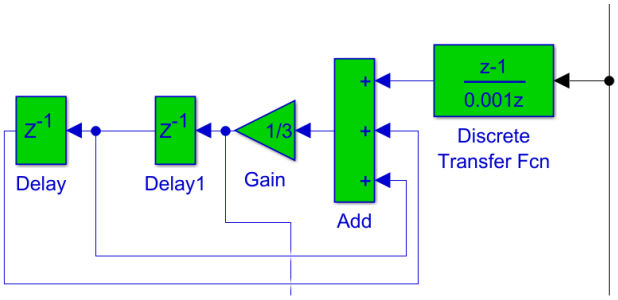
Question in the code: Theta1_old = thetamotor1; //**order matters here. Why??**

Answer: The order of updating variables "Omega1_old2 and Omega1_old1" matters because we need to first update *Omega1_old2*, the velocity at time *t-2T* by assigning *Omega1_old1* to it, and then update *Omega1_old1* by either assigning the filtered velocity (Method 2) or raw velocity (Method 1) at current time. If we do the variable update procedure in the reverse order, that is to update *Omega1_old1* first, then the velocity at time t-T would be lost and there's no way to update *Omega1_old2*.

We used the second method to approximate $\dot{\theta}$ in our C code. The code is as follows (take Omega1 as example)

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

We implemented the third method: infinite average method in the Simulink as follows:



Where the input into Discrete Transfer Fcn is $\theta = \begin{bmatrix} \theta_{2M} \\ \theta_{3M} \end{bmatrix}$ and the output is $\dot{\theta} = \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}$.

3.2 How to implement an integral approximation.

In this part we implemented the integral portion of the PID controller using trapezoidal approximation. The equation for integral approximation is given as follows: $I_K = I_{K-1} + \frac{e_K + e_{K-1}}{2} * T$.

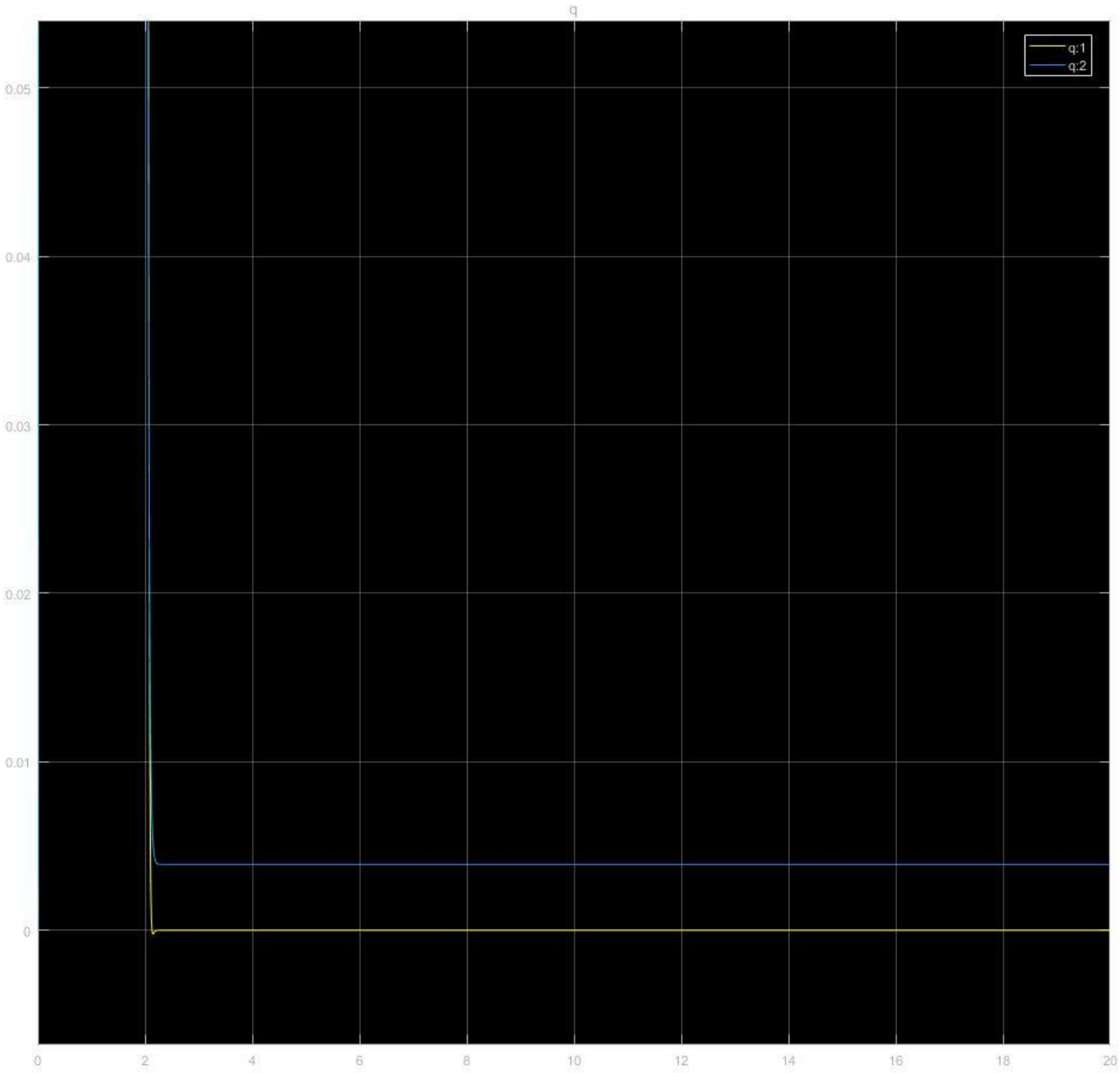
In addition, we filtered the $\frac{e_K + e_{K-1}}{2} * T$ part by setting a threshold value = 0.1. If the absolute value for $\frac{e_K + e_{K-1}}{2} * T$ is greater than threshold value, then we reset I_K to 0. The pseudocode for our integral approximation is given as follows:

```
Initialize tau1, tau2, tau3 = 0, 0, 0
for i in [1, 2, 3]:
    tau_i += Kpi * (theta_i_d - theta_i_motor)
    d_theta_i_motor = filter(theta_i_motor)
    tau_i += Kdi * (d_theta_i_d - d_theta_i_motor)
    delt_integral = theta_i_d - theta_i_motor
    if delt_integral > threshold:
        integral = 0
        delt_integral = 0
    tau_i += Kii * (integral + delt_integral)
    if abs(tau_i) < 5:
        integral += delt_integral
    else:
        tau_i -= Kii * delt_integral
    tau_i += Ji * dd_theta_i_d
return
```

Where threshold = 0.1 is our threshold value, and *delt_integral* = $\frac{e_K + e_{K-1}}{2} * T$.

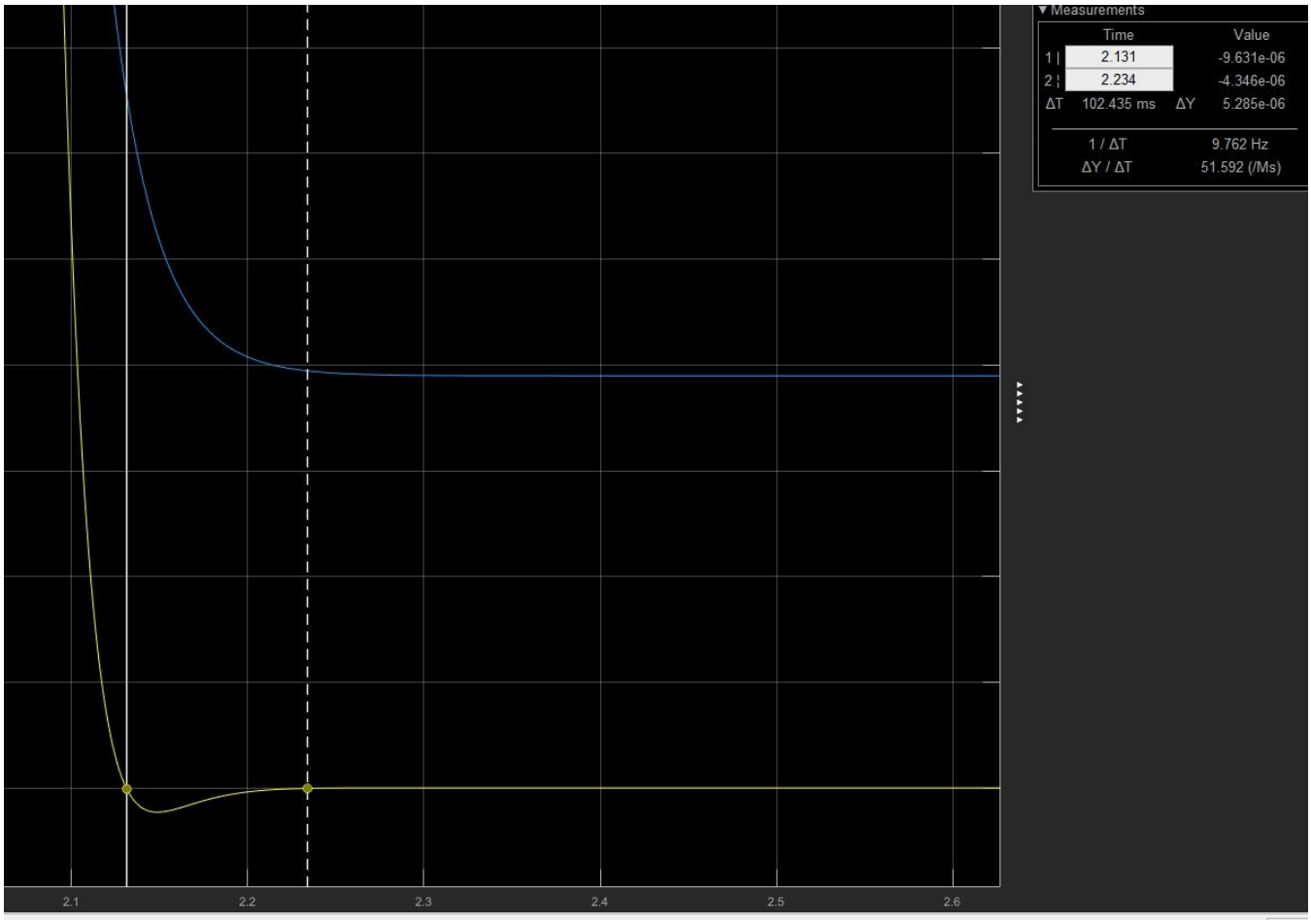
Part 4: Simulation and PD control of Link two and Link three of the CRS Robot.

The simulation result for $\theta = \begin{bmatrix} \theta_{2M} \\ \theta_{3M} \end{bmatrix}$ is shown in the plot below:



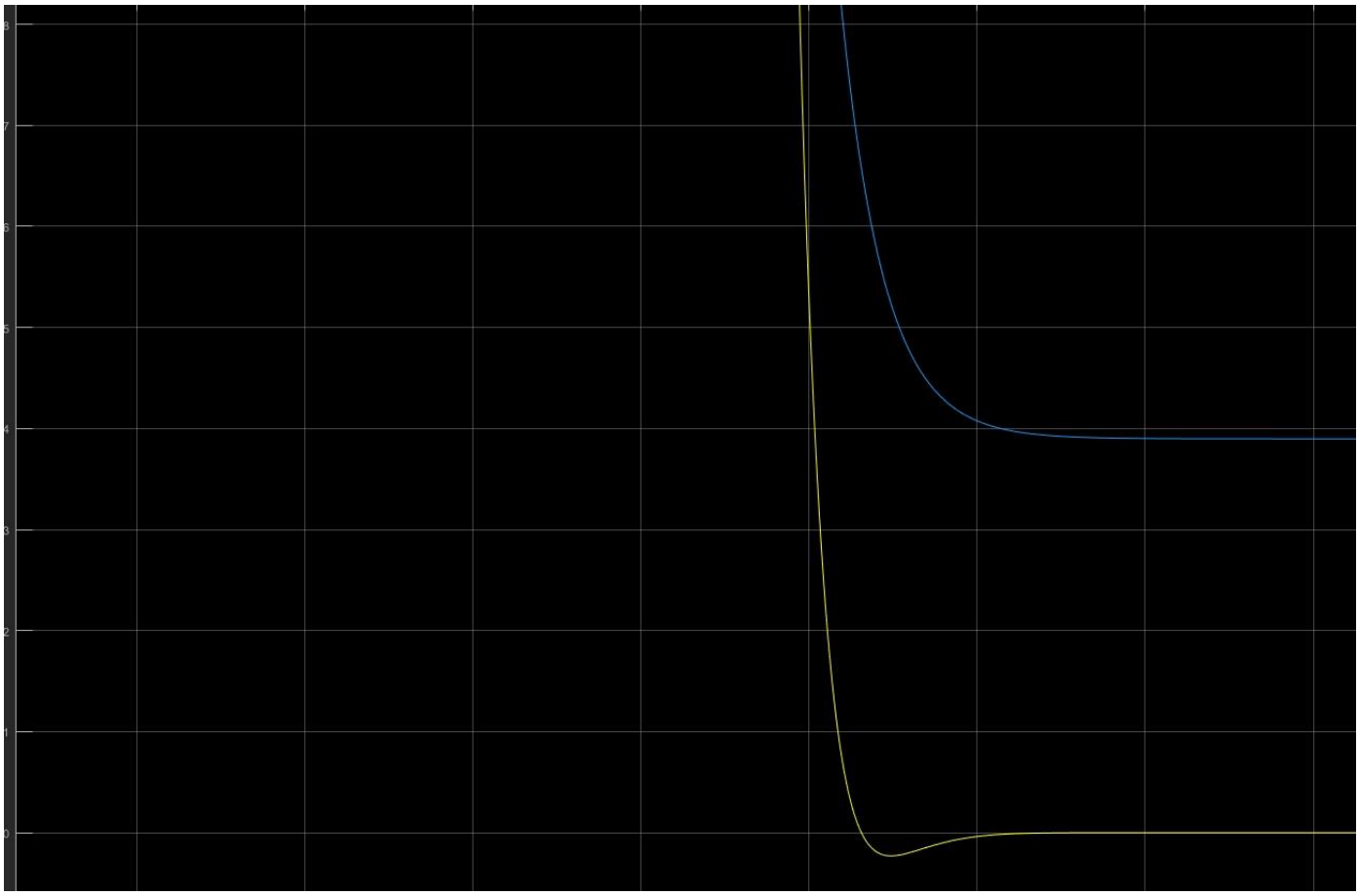
Parameters: $K_{D1} = K_{D2} = 2$; K_{p1} =60, K_{p2}=75.

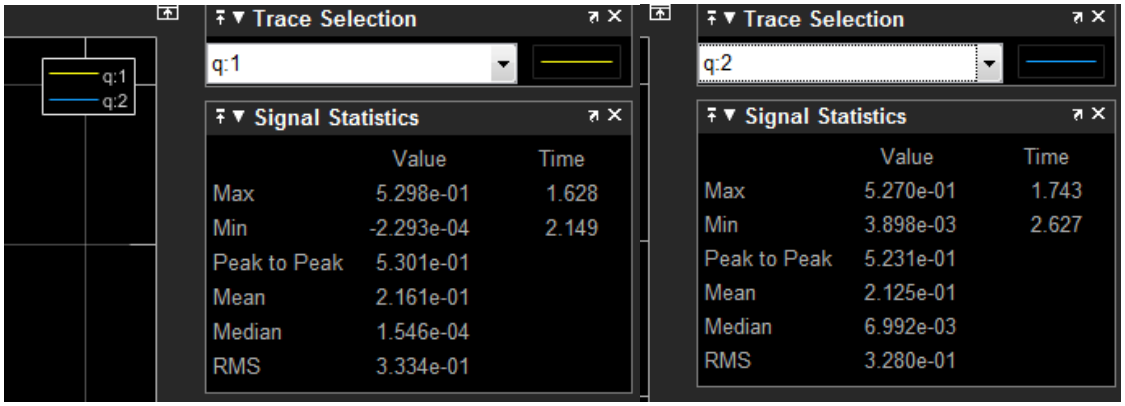
Risetime is shown in the following figure



From the cursor in the figure we can see that risetime = 0.131s for θ_{2M} , and risetime = 0.234s for θ_{3M} . Both rise time values are less than 300 ms.

Figure and statistics for overshoot can be found as follows

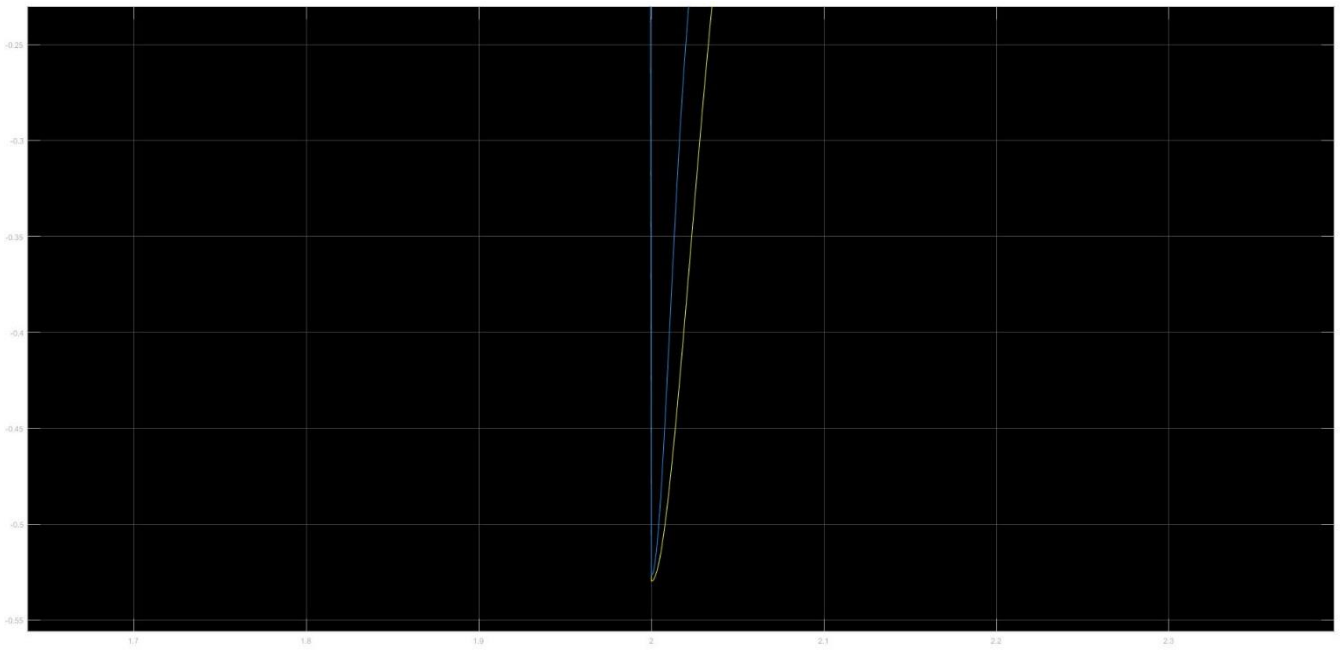




From the statistics above, we see that overshoot = 2.293e-4 for θ_{2M}, and overshoot = 3.898e-3 for θ_{3M}. Both satisfy the requirement of percent overshoot less than 1%.

Plots for tracking errors:

Zoom in:



We can see that our errors are negligible.

Part 3 (Should be Part 5): PD and PID Control of the CRS Robot.

In this section, we summarize our C implementation in Code Composer for the control of the robot arm.

We used the following notation in our code. theta1d stands for the 'theta1 destination' value, d_theta1d stands for the 'derivative of theta1 destination' value and dd_theta1d stands for the 'second derivative of theta1 destination'.

We follow the exact descriptions in the lab manual to implement the PID control with feed forward term. The entire code consists of 4 blocks as listed below.

PD Control

Proportional Block

This is the simplest block, where the weighted difference between the destination theta and real theta is added to the torque as

$$\tau_i \mathrel{+}= Kp_i * (theta\_i\_d - theta\_i\_motor)$$

Where Kp_i is the parameter we tunned to be (Kp1, Kp2, Kp3) = (70, 60, 75).

When the control function is called every millisecond, the theta destination value is given by the user defined trajectory and the real theta motor is given by the function input. The torque is updated with the weighted difference without any exceptions.

Derivative Block

This is the block that add the weighted difference between the derivative of theta destination and the derivative of the real theta to the torque.

We used the first method in the lab manual to filter the theta derivative values. This method keep tracking the latest three raw theta derivative values and use the average value as the filtered derivative value.

Then the weighted difference between the derivative of theta destination and the filtered derivative of the real theta motor is added to the torque as

$$\tau_i \mathrel{+}= Kd_i * (d\_theta\_i\_d - d\_theta\_i\_motor)$$

Where Kd_i is the parameter we tunned to be (Kd1, Kd2, Kd3) = (2, 2, 2).

When the control function is called every millisecond, the derivative of theta destination is given by the user defined trajectory directly and the filtered real theta motor derivative is computed using the theta motor values. The torque is updated with the weighted difference without any exceptions.

In addition, we saturated the torque between +/-5 as required in the lab manual. The code is as follows:

```
*tau1 = J1*dd_theta1d + Kp1*(theta1d-theta1motor) + Kd1*(d_theta1d-Omega1) + Ki1*(Ik1+delt_IK1);
*tau2 = J2*dd_theta2d + Kp2*(theta2d-theta2motor) + Kd2*(d_theta2d-Omega2) + Ki2*(Ik2+delt_IK2);
*tau3 = J3*dd_theta3d + Kp3*(theta3d-theta3motor) + Kd3*(d_theta3d-Omega3) + Ki3*(Ik3+delt_IK3);
```

```
        // Boundary Check
        if (fabs(*tau1) < 5)
                Ik1 = Ik1+delt_IK1;
        else
                *tau1 = *tau1-Ki1*delt_IK1;
        if (fabs(*tau2) < 5)
                Ik2 = Ik2+delt_IK2;
        else
                *tau2 = *tau2-Ki2*delt_IK2;
        if (fabs(*tau3) < 5)
                Ik3 = Ik3+delt_IK3;
        else
        *tau3 = *tau3-Ki3*delt_IK3;
```

So basically we first computed the torque and then check for torque saturation. For each of the three torque values, if absolute value for the torque is smaller than 5, then update the integration approximation part as described in **Part 3.2**. If the absolute value is larger than 5, than saturate the torque by "un-update" the torque by subtract the integral control part "`Ki*delt_IK`" from the torque.

Integral Block

This is the block that add the integral of the theta error to the torque. The theta error is defined as the difference between the theta destination value and the real theta motor value. However, if the current value is larger than our threshold 0.1, we then reset the integral value to 0. Then the weighted integral value is added to the torque as

$$\tau_i \mathrel{+}= Ki_i * (Integral + \Delta Integral)$$

where Ki_i is the parameter we tunned to be (Ki1, Ki2, Ki3) = (1, 1, 1).

When the control function is called every millisecond, the new delta integral is calculated and added to the torque. If after calculation, torque absolute value is smaller than 5, we update the integral value with the new delta integral; otherwise, we do not update integral value and subtract the delta integral from the torque.

The code for integration wind up check is as follows:

```
      double threshold = 0.1;          // CHange threshold val HERE
       if (fabs(theta1d-Theta1_old) < threshold)
              delt_IK1 = (2*theta1d-theta1motor-Theta1_old)/2*0.001;
       else
              Ik1 = 0;
       if (fabs(theta2d-Theta2_old) < threshold)
              delt_IK2 = (2*theta2d-theta2motor-Theta2_old)/2*0.001;
       else
              Ik2 = 0;
       if (fabs(theta3d-Theta3_old) < threshold)
```

```
        delt_IK3 = (2*theta3d-theta3motor-Theta3_old)/2*0.001;
    else
    Ik3 = 0;
```

Feed Forward Block

This is the block where the torque is further updated with the second derivative of the theta destination value as

$$\tau_i \mathrel{+}= J_i * dd\_theta\_i\_d$$

where J_i is the constant parameter s (J1, J2, J3) = (0.0167, 0.03, 0.0128).

When the control function is called every millisecond, the second derivative of the theta destination value is given by the used defined trajectory directly. The torque is updated with the weighted difference without any exceptions.

Overall Pseudocode

```
Initialize tau1, tau2, tau3 = 0, 0, 0
for i in [1, 2, 3]:
    tau_i += Kpi * (theta_i_d - theta_i_motor)
    d_theta_i_motor = filter(theta_i_motor)
    tau_i += Kdi * (d_theta_i_d - d_theta_i_motor)
    delt_integral = theta_i_d - theta_i_motor
    if delt_integral > threshold:
        integral = 0
        delt_integral = 0
    tau_i += Kii * (integral + delt_integral)
    if abs(tau_i) < 5:
        integral += delt_integral
    else:
        tau_i -= Kii * delt_integral
    tau_i += Ji * dd_theta_i_d
return
```

Part 4 (Should be Part 6): PID Plus Feedforward Control.

Part 4.1 Implementation of Feedforward PD Controller.

In this part, we replace the step input with a cubic polynomial trajectory $\theta^d(t)$, where $\theta^d(t)$ satisfies the following condition:

$$\theta^d(0) = 0 \ ; \ \dot{\theta}^d(0) = 0$$
$$\theta^d(1) = 0.5 \ ; \ \dot{\theta}^d(1) = 0$$
$$\theta^d(2) = 0 \ ; \ \dot{\theta}^d(2) = 0$$

Therefore, suppose

$$\theta_1{}^d(t) = a_1 t^3 + b_1 t^2 + c_1 t + d_1; \ 0 \le t \le 1$$
$$\dot{\theta_1}{}^d(t) = 3a_1 t^2 + 2b_1 t + c_1; \ 0 \le t \le 1$$
$$\theta_2{}^d(t) = a_2 t^3 + b_2 t^2 + c_2 t + d_2; \ 1 \le t \le 2$$
$$\dot{\theta_2}{}^d(t) = 3a_2 t^2 + 2b_2 t + c_2; \ 0 \le t \le 1$$

We have the following matrix equation after plugging in the conditions in the previous page:

$$\begin{bmatrix} \theta_1{}^d(0) \\ \dot{\theta_1}{}^d(0) \\ \theta_1{}^d(1) \\ \dot{\theta_1}{}^d(1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{bmatrix}$$

Linear algebra: take A = $\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0.5 \\ 3 & 2 & 1 & 0 & 0 \end{bmatrix}$;

Taking Reduced Row Echelon Form of matrix A, we get the following result:

$$rref(A) = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1.5 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Therefore

$$\theta_1{}^d(t) = -t^3 + 1.5t^2$$

.

We solved $\theta_2{}^d(t)$ in the similar way. The result for $\theta_2{}^d(t)$ is as follows:

$$\theta_2{}^d(t) = t^3 - 4.5t^2 + 6t - 2$$

Therefore, our MATLAB function for $\theta^d(t), \dot{\theta}^d(t), \ddot{\theta}^d(t)$ is as follows:

```
function [thetad, d_thetad, dd_thetad] = cubic_torque(t)

thetad = (-t.^3+1.5.*t.^2)*(t>=0)*(t<=1) + (t.^3-4.5.*t^2+6.*t-2)*(t>1)*(t<=2);
d_thetad = (-3.*t.^2+3.*t)*(t>=0)*(t<=1) + (3.*t.^2-9.*t+6)*(t>1)*(t<=2);
dd_thetad = (-6.*t+3)*(t>=0)*(t<=1) + (6.*t-9)*(t>1)*(t<=2);
end
```

Code implementation for our PID plus feedforward control can be found in **Appendix C**, where each single part is already separately explained in the section above.

Question in the lab manual: Explain why it makes sense that with step input trajectories as used in sections 2 and 3 the KD term is appropriate to be $-K_D\dot{\theta}$ but now that we are generating polynomial trajectories the derivative term is $+K_D(\dot{\theta}^d - \dot{\theta})$.

Answer: Previously we set the trajectory to be a constant value for θ^d from t=0s to t=2s. Therefore during this time period, our previous $\dot{\theta}^d$ would always be 0. Therefore the equation $K_D(\dot{\theta}^d - \dot{\theta})$ becomes $K_D(0 - \dot{\theta}) = -K_D\dot{\theta}$.

Part 5 (Should be Part 7): Follow a repeating trajectory

The trajectory we designed consisted with four circles, shown in **Figure** 1.

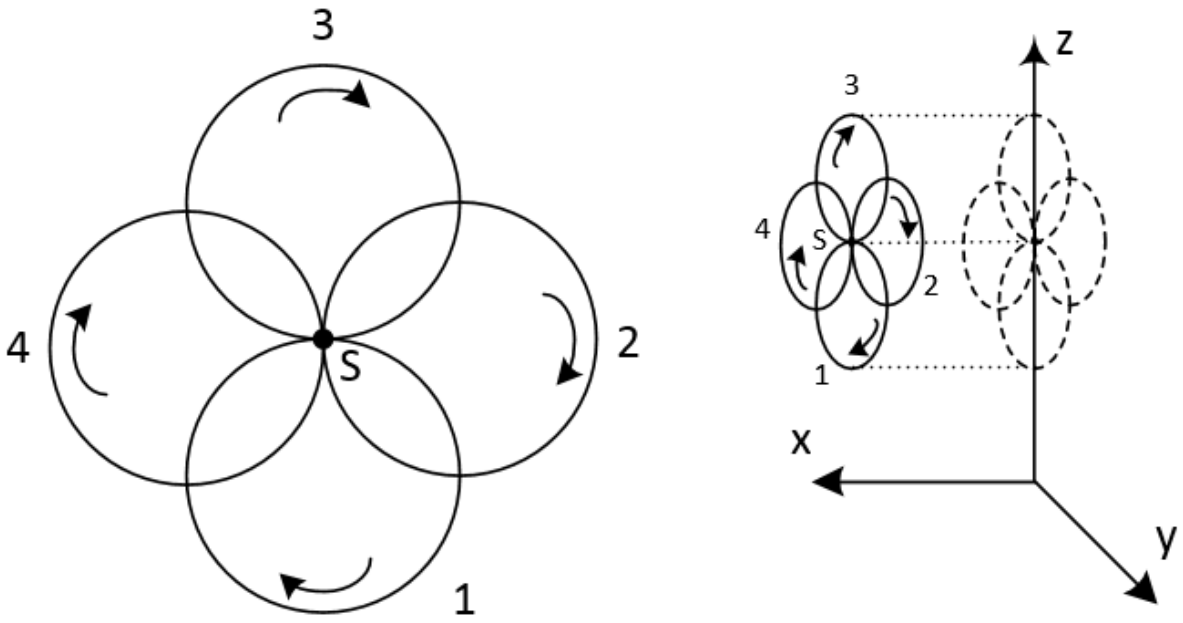


Figure 1. Trajectory of Four Circles

The entire trajectory lies in the y-z plane where x = 25.4 cm, the start point S is at location (x, y, z) = (25.4, 0, 50.8), all the circles has the radius r = 8.0 cm. The robot arm go through the four circles from #1 to #4 one by one in the clockwise direction repeatedly. For example, the end effector start at point S, go through circle #1, get back to point S and go through circle #2. Each circle takes 4 seconds to run and the entire trajectory takes 16 seconds to complete. To derive the trajectory data, we first derive the trajectory function of a single circle. Since we want this trajectory to be a function of time, we used the formula below

$$\begin{cases} y = y_c + r * \sin(wt) \\ z = z_c + r * \cos(wt) \end{cases}$$

where (y_c, z_c) is the circle center, r is the radius and w is the angular velocity.

Since we want each circle to take 4 seconds to complete, the angular velocity is w = PI/2. Using this formula, it is straight forward to derive the entire trajectory. We only need to pay attention to the circle center, time delay and angle shift. The circle center is always 8 cm away from the starting point S at (x, y, z) = (25.4, 0, 50.8) in y or z direction; the time delay could be introduced by setting t to (t-4), (t-8) and (t-12); the angle shift specify the starting point on the circle which could be 0, -PI/2, PI or PI/2.

The entire trajectory is thus

$$y = \begin{cases} 8 * \sin\left(\frac{\pi}{2}t\right) & 0 \le t < 4 \\ 8 + 8 * \sin\left(\frac{\pi}{2}(t-4) - \frac{\pi}{2}\right) & 4 \le t < 8 \\ 8 * \sin\left(\frac{\pi}{2}(t-8) + \pi\right) & 8 \le t < 12 \\ -8 + 8 * \sin\left(\frac{\pi}{2}(t-12) + \frac{\pi}{2}\right) & 12 \le t < 16 \end{cases}$$

$$z = \begin{cases} 42.8 + 8 * \cos\left(\frac{\pi}{2}t\right) & 0 \le t < 4 \\ 50.8 + 8 * \cos\left(\frac{\pi}{2}(t-4) - \frac{\pi}{2}\right) & 4 \le t < 8 \\ 58.8 + 8 * \cos\left(\frac{\pi}{2}(t-8) + \pi\right) & 8 \le t < 12 \\ 50.8 + 8 * \cos\left(\frac{\pi}{2}(t-12) + \frac{\pi}{2}\right) & 12 \le t < 16 \end{cases}$$

Now that we have the trajectory function of time t, all other processes are similar. The pseudocode is as follows

```
Initialize all thetas and dthetas to 0
for t in 0:0.001:16 seconds:
    x = 25.4
    y,z = calculateTrajectory(t)
    theta1d,theta2d,theta3d = inverseKinematics(x,y,z)
    compute thetad derivative with new thetad and old thetad
    constrain thetad derivatives absolute value < 0.5
    calculate tau1,tau2,tau3 with PID control
return
```

Since the second derivatives are hard to calculate, we drop the feed forward term and only use the PID control we developed in the section before to control the trajectory.

After executing the program, we are pleased to find that our robot works well. The end effector remains in the same y-z plane and follows the trajectory in a smooth and steady manner.

Conclusion

In this lab we derived the equations of motions for our robot arm, simulated the equations of motion for our robot arm, designed and implemented PID plus feedforward control system for robot arm, and we tested the result of our control system by using both a given trajectory and a trajectory designed by ourselves using inverse kinematics. This is a time consuming lab, but we are glad that we learned a lot from it.

Appendix A: MATLAB Code for Simulink Blocks

C

```
function Cdq = fcn(dq,q)
    p3 = 0.0076;
    Cdq = [0 -p3*cos(q(2)-q(1))*dq(2); p3*cos(q(2)-q(1))*dq(1) 0]*dq;
    Cdq = Cdq + [0.04*dq(1); 0.037*dq(2)];
end
```

Dinv

```
function ddq = fcn(input,q)
    p3 = 0.0076;
    p2 = 0.0128;
    p1 = 0.03;
    D = [p1 -p3.*sin(q(2)-q(1)); -p3.*sin(q(2)-q(1)) p2];
    ddq = inv(D)*input;
end
```

G

```
function G = fcn(q)
     g = 9.81;
     p4 = 0.0753;
     p5 = 0.0298;
     G = [-p4*g*sin(q(1)); -p5*g*cos(q(2))];

end
```

Appendix B: C Code for Averaging Filter

First Method of Filtering Velocity

```
float Theta1_old = 0;
float Omega1_raw = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;
// This function is called every 1 ms
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;
//order matters here. Why??
Omega1_old2 = Omega1_old1;
Omega1_old1 = Omega1_raw;
}
```

Second Method of Filtering Velocity

```
float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;
// This function is called every 1 ms
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;
//order matters here. Why??
Omega1_old2 = Omega1_old1;
Omega1_old1 = Omega1;
}
```

Appendix C: Code Implementation for PID plus feedforward control

```
      *tau1 = 0;
       *tau2 = 0;
       *tau3 = 0;

       //Motor torque limitation(Max: 5 Min: -5)

       // DH Frame Angle
       float theta1dh = theta1motor;
       float theta2dh = theta2motor - PI/2;
       float theta3dh = theta3motor-theta2motor+PI/2;

       // Forward Kinematics
       float endx = 25.4*cos(theta1dh)*(cos(theta2dh) + cos(theta2dh+theta3dh));
       float endy = 25.4*sin(theta1dh)*(cos(theta2dh) + cos(theta2dh+theta3dh));
       float endz = 25.4*(1-sin(theta2dh)-sin(theta2dh+theta3dh));

       // Forward Kinematics
       float theta1inv = atan2(endy,endx);
       float armlength = sqrt(endx*endx+endy*endy+(endz-25.4)*(endz-25.4));
       float theta2inv = - atan2(endz-25.4,sqrt(endx*endx+endy*endy)) -
acos(armlength*armlength/2/25.4/armlength);
       float theta3inv = 2 * acos(armlength*armlength/2/25.4/armlength);

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

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

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

       }

       /*
       float theta1d = theta1val((mycount%4000)/1000.0);
       float theta2d = thetaval((mycount%4000)/1000.0);
       float theta3d = thetaval((mycount%4000)/1000.0);

       float d_theta1d = d_theta1val((mycount%4000)/1000.0);
       float d_theta2d = d_thetaval((mycount%4000)/1000.0);
       float d_theta3d = d_thetaval((mycount%4000)/1000.0);

       float dd_theta1d = dd_theta1val((mycount%4000)/1000.0);
       float dd_theta2d = dd_thetaval((mycount%4000)/1000.0);
       float dd_theta3d = dd_thetaval((mycount%4000)/1000.0);
       */
       thetaval(mycount%16000 / 1000.0);
```

```
	// Calculate Integral
	double threshold = 0.1;		// CHange threshold val HERE
	if (fabs(theta1d-Theta1_old) < threshold)
		delt_IK1 = (2*theta1d-theta1motor-Theta1_old)/2*0.001;
	else
		Ik1 = 0;
	if (fabs(theta2d-Theta2_old) < threshold)
		delt_IK2 = (2*theta2d-theta2motor-Theta2_old)/2*0.001;
	else
		Ik2 = 0;
	if (fabs(theta3d-Theta3_old) < threshold)
		delt_IK3 = (2*theta3d-theta3motor-Theta3_old)/2*0.001;
	else
		Ik3 = 0;

	// Filtering of Theta
	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;

	// Calculate Torque
	*tau1 = J1*dd_theta1d + Kp1*(theta1d-theta1motor) + Kd1*(d_theta1d-Omega1) +
Ki1*(Ik1+delt_IK1);
	*tau2 = J2*dd_theta2d + Kp2*(theta2d-theta2motor) + Kd2*(d_theta2d-Omega2) +
Ki2*(Ik2+delt_IK2);
	*tau3 = J3*dd_theta3d + Kp3*(theta3d-theta3motor) + Kd3*(d_theta3d-Omega3) +
Ki3*(Ik3+delt_IK3);

	// Boundary Check
	if (fabs(*tau1) < 5)
		Ik1 = Ik1+delt_IK1;
	else
		*tau1 = *tau1-Ki1*delt_IK1;
	if (fabs(*tau2) < 5)
		Ik2 = Ik2+delt_IK2;
	else
		*tau2 = *tau2-Ki2*delt_IK2;
	if (fabs(*tau3) < 5)
		Ik3 = Ik3+delt_IK3;
	else
	*tau3 = *tau3-Ki3*delt_IK3;
```

Appendix D: Code for lab.c

```
#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 = -0.4238;
float offset_Enc3_rad = 0.2571;


// Your global varialbes.

long mycount = 0;

#pragma DATA_SECTION(whattoprint, ".my_vars") ///visible by matlab
float whattoprint = 0.0;

#pragma DATA_SECTION(theta1array, ".my_arrs") //visible by matlab
float theta1array[100];

#pragma DATA_SECTION(theta2array, ".my_arrs") //visible by matlab
float theta2array[100];

long arrayindex = 0;

float printtheta1motor = 0;
float printtheta2motor = 0;
float printtheta3motor = 0;
float printtheta1dh = 0;
float printtheta2dh = 0;
float printtheta3dh = 0;
float printendx = 0;
float printendy = 0;
float printendz = 0;
float printtheta1inv = 0;
float printtheta2inv = 0;
float printtheta3inv = 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;
float Theta1_old = 0;
float Omega1_raw = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;
```

```
float Theta2_old = 0;
float Omega2_raw = 0;
float Omega2_old1 = 0;
float Omega2_old2 = 0;
float Omega2 = 0;

float Theta3_old = 0;
float Omega3_raw = 0;
float Omega3_old1 = 0;
float Omega3_old2 = 0;
float Omega3 = 0;

float Ik1 = 0;
float Ik2= 0;
float Ik3 = 0;
float delt_IK1;
float delt_IK2;
float delt_IK3;


int Kp1 = 70;
int Kp2 = 60;
int Kp3 = 75;
int Kd1 = 2;
int Kd2 = 2;
int Kd3 = 2;
int Ki1 = 1;
int Ki2 = 1;
int Ki3 = 1;
float J1 = 0.0167;
float J2 = 0.03;
float J3 = 0.0128;



//Destination points
//float theta1d = PI/4;
//float theta2d = PI/4;
//float theta3d = PI/2;
/*
 * float thetaval(float t)
{
      float thetad = 0 + (-t*t*t+1.5*t*t)*(t>=0)*(t<=1) + (t*t*t-4.5*t*t+6*t-
2)*(t>1)*(t<=2);
      return thetad;
}

float d_thetaval(float t)
{
      float thetad = 0+(-3*t*t+3*t)*(t>=0)*(t<=1) + (3*t*t-9*t+6)*(t>1)*(t<=2);
      return thetad;
}

float dd_thetaval(float t)
{
```

```
	float thetad = (-6*t+3)*(t>=0)*(t<=1) + (6*t-9)*(t>1)*(t<=2);
	return thetad;
}
 *
 */
float theta1d = 0;
float theta2d = 0;
float theta3d = 0;

float d_theta1d = 0;
float d_theta2d = 0;
float d_theta3d = 0;


float dd_theta1d = 0;
float dd_theta2d = 0;
float dd_theta3d = 0;

void thetaval(float t)
{
	float r = 8.0;
	float x = 25.4;
	float y = r*sin(1/4.0*2*PI*t)*(t>=0)*(t<4) + (r + r*sin(1/4.0*2*PI*(t-4) -
PI/2))*(t>=4)*(t<8);
	float z = (50.8 - r +r*cos(1/4.0*2*PI*t))*(t>=0)*(t<4) + (50.8 + r*cos(1/4.0*2*PI*(t-
4) - PI/2))*(t>=4)*(t<8);

	y += (r*sin(1/4.0*2*PI*(t-8) + PI))*(t>=8)*(t<12) + (-r + r*sin(1/4.0*2*PI*(t-12) +
PI/2))*(t>=12)*(t<16);
	z += (50.8 + r + r*cos(1/4.0*2*PI*(t-8) + PI))*(t>=8)*(t<12) + (50.8 +
r*cos(1/4.0*2*PI*(t-12) + PI/2))*(t>=12)*(t<16);

	d_theta1d = (atan2(y,x)-theta1d)/0.001;
	d_theta1d = d_theta1d > 0.5? 0.5 : d_theta1d;
	d_theta1d = d_theta1d < -0.5? -0.5 : d_theta1d;
	theta1d = atan2(y,x);

	float armlength = sqrt(x*x+y*y+(z-25.4)*(z-25.4));
	d_theta2d = (theta2d + atan2(z-25.4,sqrt(x*x+y*y)) +
acos(armlength*armlength/2/25.4/armlength))/0.001;
	d_theta2d = d_theta2d > 0.5? 0.5 : d_theta2d;
	d_theta2d = d_theta2d < -0.5? -0.5 : d_theta2d;
	theta2d = PI/2 - atan2(z-25.4,sqrt(x*x+y*y)) -
acos(armlength*armlength/2/25.4/armlength);

	d_theta3d = d_theta2d + (theta3d - 2 *
acos(armlength*armlength/2/25.4/armlength))/0.001;
	d_theta3d = d_theta3d > 0.5? 0.5 : d_theta3d;
	d_theta3d = d_theta3d < -0.5? -0.5 : d_theta3d;
	theta3d = theta2d + 2 * acos(armlength*armlength/2/25.4/armlength) - PI/2;

	return;
}
```

```
// This function is called every 1 ms
void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,float
*tau2,float *tau3, int error) {


	*tau1 = 0;
	*tau2 = 0;
	*tau3 = 0;

	//Motor torque limitation(Max: 5 Min: -5)

	// DH Frame Angle
	float theta1dh = theta1motor;
	float theta2dh = theta2motor - PI/2;
	float theta3dh = theta3motor-theta2motor+PI/2;

	// Forward Kinematics
	float endx = 25.4*cos(theta1dh)*(cos(theta2dh) + cos(theta2dh+theta3dh));
	float endy = 25.4*sin(theta1dh)*(cos(theta2dh) + cos(theta2dh+theta3dh));
	float endz = 25.4*(1-sin(theta2dh)-sin(theta2dh+theta3dh));

	// Forward Kinematics
	float theta1inv = atan2(endy,endx);
	float armlength = sqrt(endx*endx+endy*endy+(endz-25.4)*(endz-25.4));
	float theta2inv = - atan2(endz-25.4,sqrt(endx*endx+endy*endy)) -
acos(armlength*armlength/2/25.4/armlength);
	float theta3inv = 2 * acos(armlength*armlength/2/25.4/armlength);

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

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

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

	}

	/*
	float theta1d = theta1val((mycount%4000)/1000.0);
	float theta2d = thetaval((mycount%4000)/1000.0);
	float theta3d = thetaval((mycount%4000)/1000.0);

	float d_theta1d = d_theta1val((mycount%4000)/1000.0);
	float d_theta2d = d_thetaval((mycount%4000)/1000.0);
	float d_theta3d = d_thetaval((mycount%4000)/1000.0);

	float dd_theta1d = dd_theta1val((mycount%4000)/1000.0);
	float dd_theta2d = dd_thetaval((mycount%4000)/1000.0);
```

```
	float dd_theta3d = dd_thetaval((mycount%4000)/1000.0);
	*/
	thetaval(mycount%16000 / 1000.0);

	// Calculate Integral
	double threshold = 0.1;		// CHange threshold val HERE
	if (fabs(theta1d-Theta1_old) < threshold)
		delt_IK1 = (2*theta1d-theta1motor-Theta1_old)/2*0.001;
	else
		Ik1 = 0;
	if (fabs(theta2d-Theta2_old) < threshold)
		delt_IK2 = (2*theta2d-theta2motor-Theta2_old)/2*0.001;
	else
		Ik2 = 0;
	if (fabs(theta3d-Theta3_old) < threshold)
		delt_IK3 = (2*theta3d-theta3motor-Theta3_old)/2*0.001;
	else
		Ik3 = 0;

	// Filtering of Theta
	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;

	// Calculate Torque
	*tau1 = J1*dd_theta1d + Kp1*(theta1d-theta1motor) + Kd1*(d_theta1d-Omega1) +
Ki1*(Ik1+delt_IK1);
	*tau2 = J2*dd_theta2d + Kp2*(theta2d-theta2motor) + Kd2*(d_theta2d-Omega2) +
Ki2*(Ik2+delt_IK2);
	*tau3 = J3*dd_theta3d + Kp3*(theta3d-theta3motor) + Kd3*(d_theta3d-Omega3) +
Ki3*(Ik3+delt_IK3);

	// Boundary Check
	if (fabs(*tau1) < 5)
		Ik1 = Ik1+delt_IK1;
	else
		*tau1 = *tau1-Ki1*delt_IK1;
	if (fabs(*tau2) < 5)
		Ik2 = Ik2+delt_IK2;
	else
		*tau2 = *tau2-Ki2*delt_IK2;
```

```
	if (fabs(*tau3) < 5)
		Ik3 = Ik3+delt_IK3;
	else
		*tau3 = *tau3-Ki3*delt_IK3;


	if ((mycount%500)==0) {
		if (error != 0){
			serial_printf(&SerialA, "error position\n\r");
		}
		else{
		if (whattoprint > 0.5) {
			serial_printf(&SerialA, "I love robotics\n\r");
		} else {
			printtheta1motor = theta1motor;
			printtheta2motor = theta2motor;
			printtheta3motor = theta3motor;
			printtheta1dh = theta1dh;
			printtheta2dh = theta2dh;
			printtheta3dh = theta3dh;
			printendx = endx;
			printendy = endy;
			printendz = endz;
			printtheta1inv = theta1inv;
			printtheta2inv = theta2inv;
			printtheta3inv = theta3inv;
			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 = theta1motor;
	Simulink_PlotVar2 = theta2motor;
	Simulink_PlotVar3 = theta3motor;
	Simulink_PlotVar4 = 0;

	mycount++;
}

void printing(void){
	serial_printf(&SerialA, "Motor Angle: %.2f %.2f,%.2f
\n\r",printtheta1motor*180/PI,printtheta2motor*180/PI,printtheta3motor*180/PI);
	serial_printf(&SerialA, "Joint Angle: %.2f %.2f,%.2f
\n\r",printtheta1dh*180/PI,printtheta2dh*180/PI,printtheta3dh*180/PI);
	serial_printf(&SerialA, "End Position: %.2f %.2f,%.2f
\n\r",printendx,printendy,printendz);
	serial_printf(&SerialA, "Inverse Angle: %.2f %.2f,%.2f
\n\r",printtheta1inv*180/PI,printtheta2inv*180/PI,printtheta3inv*180/PI);
}
```