



UNIVERSITY OF ILLINOIS AT URBANA CHAMPAIGN

Control Design and Trajectory Tracking

Lab Report II

Karan Chawla karangc2@illinois.edu

Zhaoer Li zli91@illinois.edu

Zhichao Zhang zhichao3@illinois.edu

Course Instructor: Prof. Daniel Block & Prof. Hae Won Park

March 13, 2017

1 DERIVATION OF MOTION EQUATIONS

1.1 DERIVATION OF ENERGY EQUATIONS

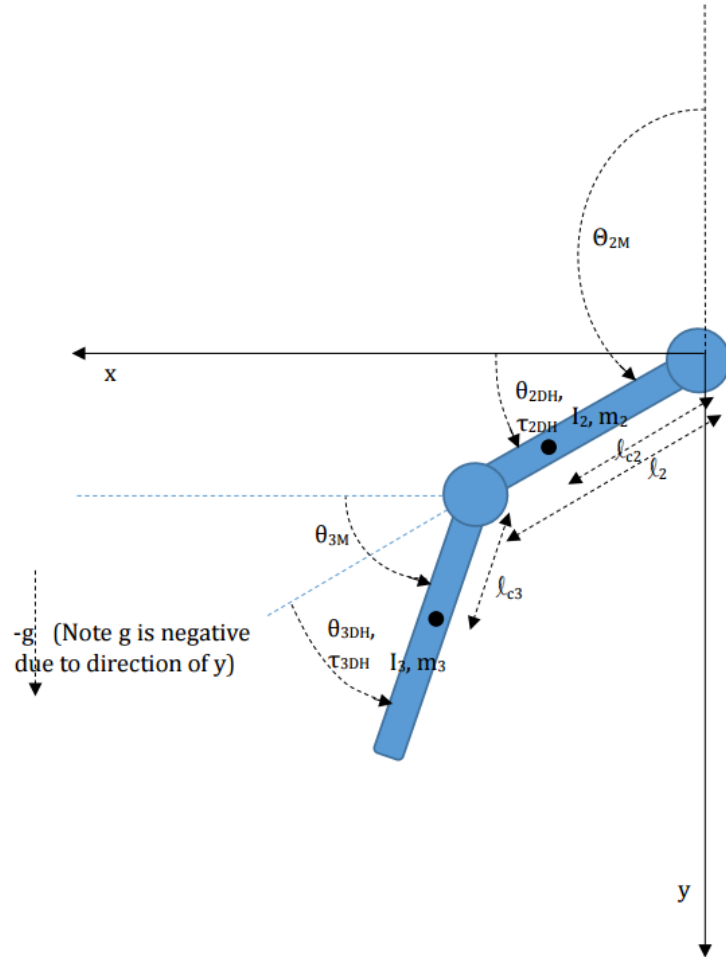


Figure 1.1: Physical Dimensions of Two Link Planar Arm (from me446 lab manual)

The system is shown in the figure 1.1. Jacobian needs to be evaluated to find the relationship between the energy with the angular velocity of each joints.

The Jacobian of mass center of link 2 is given by the following equation:

$$J_2 = \begin{bmatrix} -l_2 \sin(\theta_{2DH}) & 0 \\ l_2 \cos(\theta_{2DH}) & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

The Jacobian of mass center of link 3 is

$$J_3 = \begin{bmatrix} -l_2 \sin(\theta_{2DH}) - l_3 \sin(\theta_{2DH} + \theta_{3DH}) & -l_3 \sin(\theta_{2DH} + \theta_{3DH}) \\ l_2 \cos(\theta_{2DH}) + l_3 \cos(\theta_{2DH} + \theta_{3DH}) & l_3 \cos(\theta_{2DH} + \theta_{3DH}) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Therefore, the velocity of the mass center of link 2 is

$$v_2 = J_{2v} \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix} = \begin{bmatrix} -l_2 \sin(\theta_{2DH}) \dot{\theta}_{2DH} \\ l_2 \cos(\theta_{2DH}) \dot{\theta}_{2DH} \\ 0 \end{bmatrix}$$

The velocity of the mass center of link 3 is

$$v_3 = J_{3v} \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix} = \begin{bmatrix} (-l_2 \sin(\theta_{2DH}) - l_3 \sin(\theta_{2DH} + \theta_{3DH})) \dot{\theta}_{2DH} - l_3 \sin(\theta_{2DH} + \theta_{3DH}) \dot{\theta}_{3DH} \\ (l_2 \cos(\theta_{2DH}) + l_3 \cos(\theta_{2DH} + \theta_{3DH})) \dot{\theta}_{2DH} + l_3 \cos(\theta_{2DH} + \theta_{3DH}) \dot{\theta}_{3DH} \\ 0 \end{bmatrix}$$

The angular velocity of two links are $\dot{\theta}_{2DH}$ and $\dot{\theta}_{2DH} + \dot{\theta}_{3DH}$, respectively.

Hence, the Kinetic Energy of this system can be evaluated

$$\begin{aligned} K &= K_v + K_\omega = \frac{1}{2} m_2 |v_2|^2 + \frac{1}{2} m_3 |v_3|^2 + \frac{1}{2} I_2 (\dot{\theta}_{2DH})^2 + \frac{1}{2} I_3 (\dot{\theta}_{2DH} + \dot{\theta}_{3DH})^2 \\ &= \frac{1}{2} m_2 l_2^2 \dot{\theta}_{2DH}^2 + \frac{1}{2} m_3 l_2^2 \dot{\theta}_{2DH}^2 + \frac{1}{2} m_3 l_3^2 (\dot{\theta}_{2DH} + \dot{\theta}_{3DH})^2 \\ &\quad + l_2 l_3 \cos \theta_{3DH} \dot{\theta}_{2DH} (\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) + \frac{1}{2} I_2 (\dot{\theta}_{2DH})^2 + \frac{1}{2} I_3 (\dot{\theta}_{2DH} + \dot{\theta}_{3DH})^2 \\ &= \frac{1}{2} \dot{\theta}_{2DH}^2 p_1 + \frac{1}{2} (\dot{\theta}_{2DH} + \dot{\theta}_{3DH})^2 p_2 + \cos(\theta_{3DH}) \dot{\theta}_{2DH} (\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) p_3 \end{aligned} \quad (1.1)$$

where $p_1 = m_2 l_2^2 + m_3 l_2^2 + I_2$ $p_2 = m_3 l_3^2 + I_3$ $p_3 = m_3 l_2 l_3$.

The potential energy of this system is determined by change of height of mass center of two links. The height change of links are

$$\Delta h_2 = l_2 \sin(\theta_{2DH}) \quad \Delta h_3 = l_2 \sin(\theta_{2DH}) + l_3 \sin(\theta_{2DH} + \theta_{3DH})$$

Therefore, the potential energy of this system is

$$V = -m_2gh_2 - m_3gh_3 = -p_4g\sin(\theta_{2DH}) - p_5g\sin(\theta_{2DH} + \theta_{3DH})$$

where $p_4 = m_2l_{c2} + m_3l_2$ $p_5 = m_3l_{c3}$.

1.2 DERIVATION OF DYNAMIC EQUATIONS IN TERMS OF DH THETAS

Using the Lagrangian method

$$\frac{d}{dt} \frac{\partial(K - V)}{\partial \dot{\theta}} - \frac{\partial(K - V)}{\partial \theta} = \tau$$

Substitute the previous K and V in this equation,

$$\begin{aligned} \frac{\partial(K - V)}{\partial \dot{\theta}} &= \begin{bmatrix} p_1\dot{\theta}_{2DH} + p_2(\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) + p_3\cos(\theta_{3DH})(2\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) \\ p_2(\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) + p_3\cos(\theta_{3DH})\dot{\theta}_{2DH} \end{bmatrix} \\ \frac{\partial(K - V)}{\partial \theta} &= \begin{bmatrix} p_4g\cos(\theta_{2DH}) + p_5g\cos(\theta_{2DH} + \theta_{3DH}) \\ -p_3\sin(\theta_{3DH})\dot{\theta}_{2DH}(\dot{\theta}_{2DH} + \dot{\theta}_{3DH}) + p_5g\cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix} \\ \tau &= \frac{d}{dt} \frac{\partial(K - V)}{\partial \dot{\theta}} - \frac{\partial(K - V)}{\partial \theta} \\ &= \begin{bmatrix} (p_1 + p_2 + 2\cos(\theta_{3DH})p_3)\ddot{\theta}_{2DH} + (p_2 + \cos\theta_{3DH}p_3)\ddot{\theta}_{3DH} \\ (p_2 + \cos(\theta_{3DH})p_3)\ddot{\theta}_{2DH} + p_2\ddot{\theta}_{3DH} \end{bmatrix} \\ &+ \begin{bmatrix} -p_3\sin(\theta_{3DH})\dot{\theta}_{3DH}\dot{\theta}_{2DH} - p_3\sin(\theta_{3DH})(\dot{\theta}_{2DH} + \dot{\theta}_{3DH})\dot{\theta}_{3DH} \\ p_3\sin(\theta_{3DH})\dot{\theta}_{2DH}^2 \end{bmatrix} \quad (1.2) \\ &+ \begin{bmatrix} -p_4g\cos(\theta_{2DH}) - p_5g\cos(\theta_{2DH} + \theta_{3DH}) \\ -p_5g\cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix} \\ &:= D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + g(\theta) \\ D(\theta) &= \begin{bmatrix} (p_1 + p_2 + 2\cos(\theta_{3DH})p_3) & (p_2 + \cos\theta_{3DH}p_3) \\ (p_2 + \cos(\theta_{3DH})p_3) & 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}_{2DH} + \dot{\theta}_{3DH}) \\ p_3\sin(\theta_{3DH})\dot{\theta}_{2DH} & 0 \end{bmatrix} \\ g(\theta) &= \begin{bmatrix} -p_4g\cos(\theta_{2DH}) - p_5g\cos(\theta_{2DH} + \theta_{3DH}) \\ -p_5g\cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix} \\ \ddot{\theta} &= \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) \end{aligned}$$

Therefore, we found all dynamic equations for this system. The relationship between the angular acceleration of each joint and the torque on each joint can be evaluated by this equation.

1.3 DERIVATION OF DYNAMIC EQUATIONS IN TERMS OF MOTOR ANGLES

In the previous lab, the relationship between the motor angle and the DH angles has been evaluated.

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

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

Besides, when a torque is applied on link 3, this moment of torque will also be applied on link 2, therefore

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

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

Substitute the equations above into the equation 2, a new group of D, C, g can be evaluated according to motor angle.

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

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

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

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

2 NONLINEAR SIMULATION AND PID CONTROL IMPLEMENTATION

2.1 SIMULINK SIMULATION OF CRS ARM

A nonlinear simulation of the equations of motion of the robot was generated via Simulink in order to further implement proportional and derivative control. The block diagram of the simulation is shown below.

The subsystem block called 'space form equations' store the space form equations of motion. Using the acceleration outputs $\ddot{\theta}_{2m}$ and $\ddot{\theta}_{3m}$ two pairs of integrators calculate the velocities from accelerations, and positions from velocities, of motor 2 and 3. 'tau2' and 'tau3' were torque inputs and were set to zero. The Matlab code to simulate the 'Space form equations' subsystem block is shown in the section below:

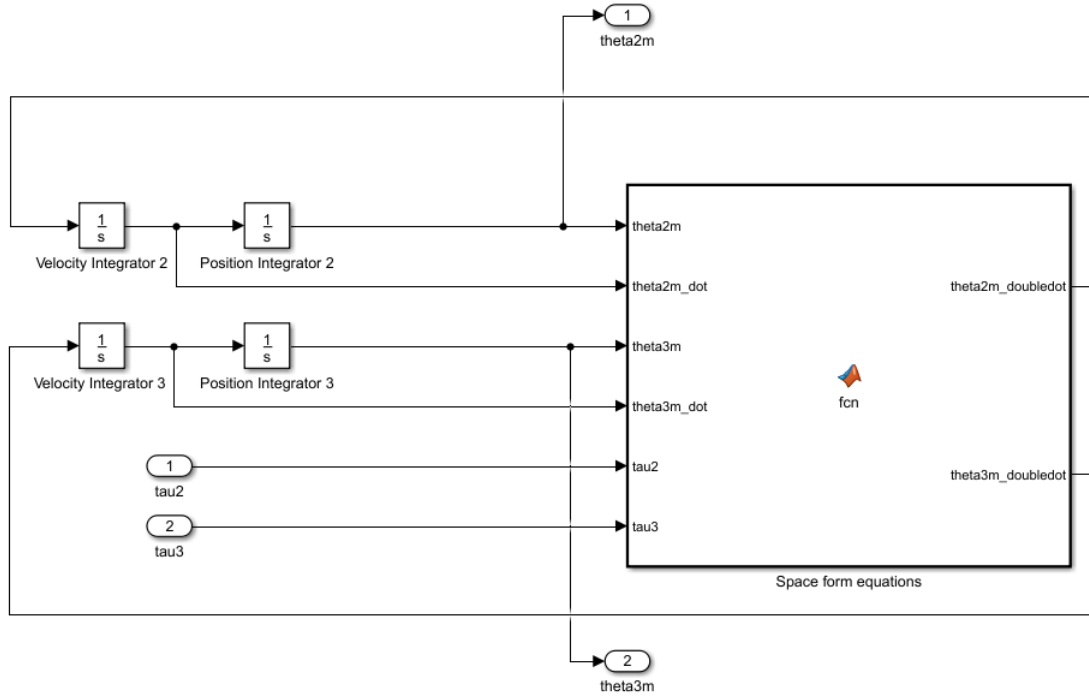


Figure 2.1: Simulink Simulation Block Diagram of Robot System

```

function [theta2m_doubledot,theta3m_doubledot]= fcn(theta2m, theta2m_dot,
    theta3m, theta3m_dot, tau2,tau3)
p = [0.0300;0.0128;0.0076;0.0753;0.0298];
p1 = p(1);
p2 = p(2);
p3 = p(3);
p4 = p(4);
p5 = p(5);
g = 9.81;
Fic = [-0.04*theta2m_dot; -0.037*theta3m_dot];
tau = [tau2;tau3];
D = [p1 -p3*sin(theta3m - theta2m); -p3*sin(theta3m - theta2m) p2];
C = [0 -p3*cos(theta3m - theta2m)*theta3m_dot;
    p3*cos(theta3m-theta2m)*theta2m_dot 0];
G = [-p4*g*sin(theta2m); -p5*g*cos(theta3m)];
out = inv(D)*tau - inv(D)*C*[theta2m_dot;theta3m_dot] - inv(D)*G + inv(D)*Fic;
theta2m_doubledot = out(1);
theta3m_doubledot = out(2);
end

```

which is the exact representation of the dynamic equations in section 1.3. Note that

array p was a predefined one with values given by the lab instruction manual.

Plots of θ_2 vs t and θ_3 vs t were generated in order to verify the simulation. From fig. 2.1 we can tell that the equilibrium position of the robot is at $\theta_2 = \pi$ and $\theta_3 = \pi/2$, where both links are pointing straight down towards the ground due to gravitational force. To illustrate how the robot oscillated from non-equilibrium position to equilibrium position, we also plotted the robot's links' responses when both links started horizontally, namely, pointing towards the positive X direction of the inertia frame.

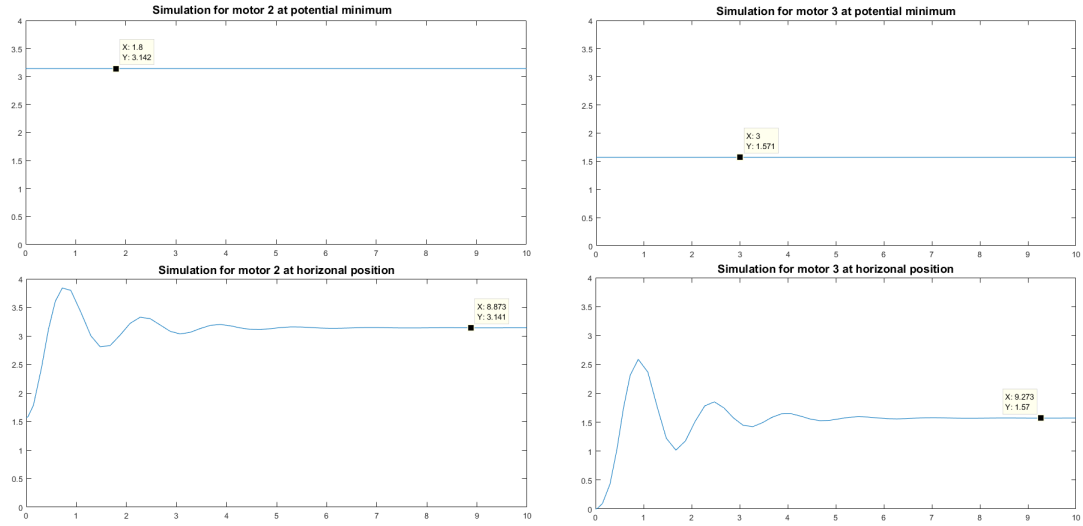


Figure 2.2: 2 Simulated Natural Response of Robot System (Motor angle (Radian) vs Time (s))

From the 2.1 we can observe that in both cases, the simulation showed that the robot always stayed at or went to the equilibrium position, depending on different initial conditions. This completes the verification of the simulation model with the experiments.

2.2 IMPLEMENTATION OF PID CONTROL

2.2.1 SIMULATION OF PD CONTROL

After we obtained the nonlinear simulation of the robot, we went one step further, to implement controls on motor torque inputs to realize position control. The PD-control-added block diagram is shown in 2.2.1.

where the previous simulation model was stored into the subsystem block called 'system'. One constant block and one control block were implemented for each motor torque input, e.g., 'theta2md' referred to the reference value of θ_{2M} and 'PD control2' the PD control block. The scoped parameters were: reference values, simulated motor angles, torque inputs (control outputs), tracking errors, which was the difference between simulated and reference values.

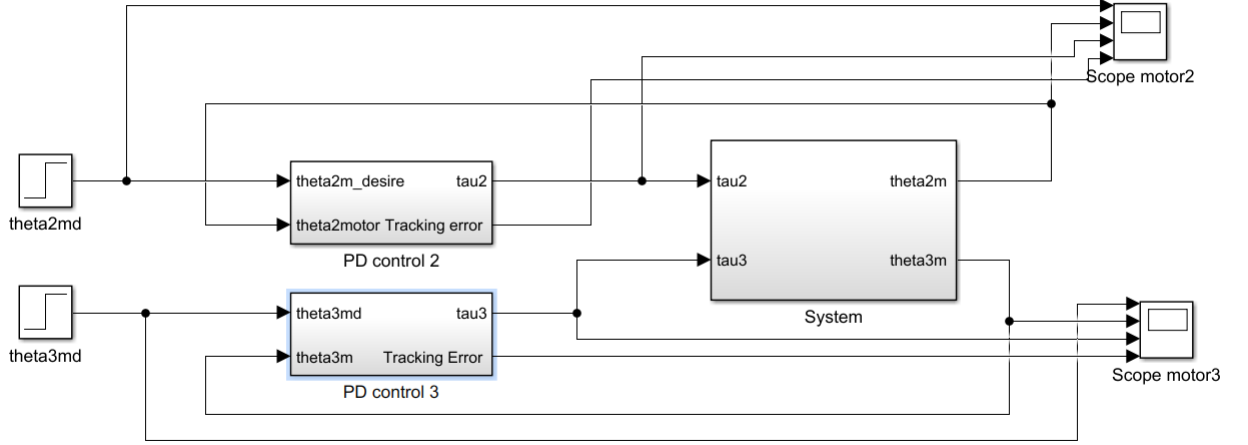


Figure 2.3: PD Control Block Diagram of Robot System

To implement PD control, we created the PD control subsystem block following the equations listed below:

$$\tau_{2M} = K_{P1}(\theta_{2d} - \theta_{2M}) - K_{D1}\dot{\theta}_{2M} \quad (2.1)$$

$$\tau_{3M} = K_{P2}(\theta_{3d} - \theta_{3M}) - K_{D2}\dot{\theta}_{3M} \quad (2.2)$$

where τ_{2M} is motor 2 torque output, K_{P1} is proportional gain for motor 2, K_{D1} is derivative gain for motor 2, θ_{2d} is reference value for θ_{2M} , motor 2 angle. Similar naming convention holds for Eqn. 2.2.

Here’s how the PD control subsystem block, taking the block associated with τ_{2M} as an example. Fig. 2.2.1 shows the aforementioned block diagram.

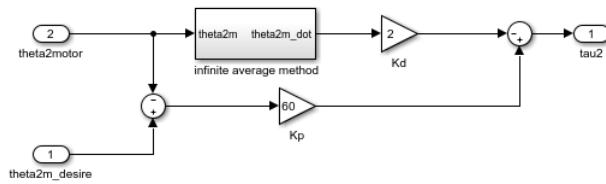


Figure 2.4: PD Control Block Diagram of motor 2

To calculate $\dot{\theta}_{2M}$, we applied the infinite average method, which is a filtering method to make the velocity approximation less noisy. Its block diagram is shown in Fig. 2.2.1.

By averaging the already-averaged velocity values, this method produced velocity values that were actually weighted averages of all their previous raw velocity values, which

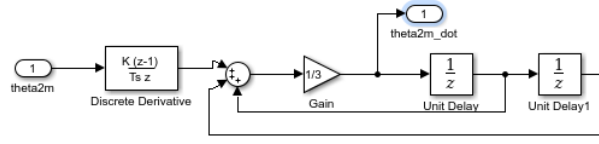


Figure 2.5: Infinite Average Method Block Diagram

made the velocity output less noisy.

C Code for the Infinite Average Filter

```
static inline float velocityFilter(float thetamotor, vel_filter* vel_filter)
{
    vel_filter->omega = (thetamotor - vel_filter->theta_old)/0.001;
    vel_filter->omega = (vel_filter->omega + vel_filter->omega_old1 +
        vel_filter->omega_old2)/3.0;

    vel_filter->theta_old = thetamotor;

    vel_filter->omega_old2 = vel_filter->omega_old1;
    vel_filter->omega_old1 = vel_filter->omega;

    return vel_filter->omega;
}
```

2.2.2 K_P AND K_D TUNING

We tuned the K_P and K_D gains in order to reach the system performance requirements, which were rise time under 300 milliseconds, overshoot less than 1% and steady state minimal. Starting from $K_P=1$ and $K_D=0$, we tuned K_P and K_D gradually and obtained the desired system performance with $K_P=60$ and $K_D=2$.

As we can see from Fig. 2.2.2, the simulated and experimental results were comparable with the same K_P and K_D gain values.

At this point, it was more efficient to perform PD gain tuning with the real system rather than simulated system. We then continued to implement full PID control on the system, as expressed below.

$$\tau_M = K_P(\theta_M^d - \theta_M) - K_D\dot{\theta}_M + K_I \int (\theta_M^d - \theta_M)$$

2.3 INTEGRAL APPROXIMATION

We want to control the torque of the CRS robot arm with reference including integration of error within some error threshold range to get rid of the system error left by pure

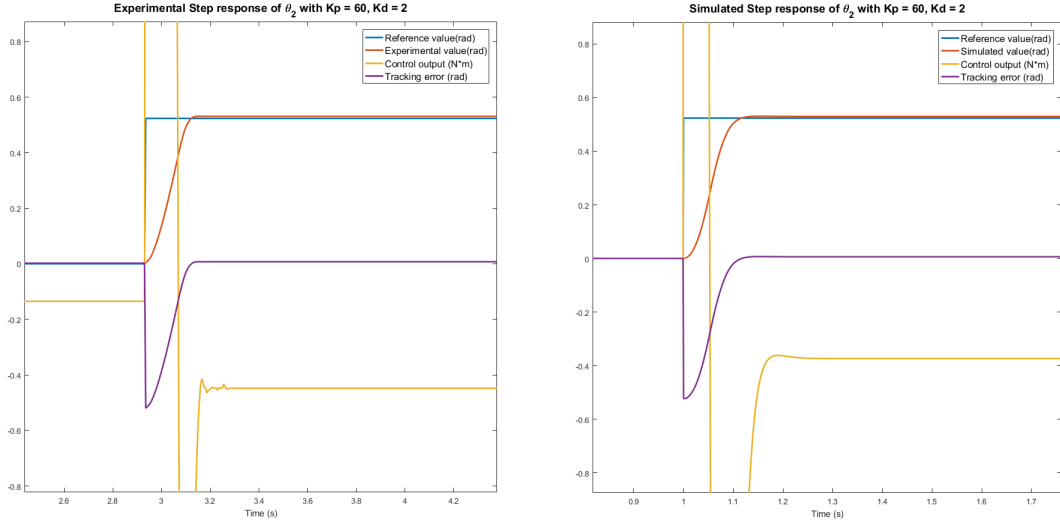


Figure 2.6: Simulated and Experimental Step Response Comparison

PD control. We used the trapezoidal rule at each new time step T to sum up the new trapezoid sliver of area under the curve giving the equation 2.3

$$I_K = I_{K-1} + \frac{e_K + e_{K-1}}{2} T \quad (2.3)$$

Since integration is a summing operator and must be monitored otherwise it could sum up to very large values. We need to set up a threshold for the integrated error. Namely, we set up an error threshold, if the tracking error is greater than the threshold, the integral I_K and any previous I_{K-1} will be zeroed. A graphic representation of the integral approximation is shown in fig. 2.3.

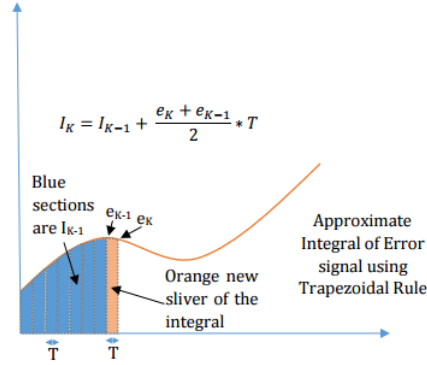


Figure 2.7: Simulated and Experimental Step Response Comparison

With integral control and previous PD control algorithm, we implemented the PID

control function in Code Composer Studio.

```
static inline float PIDController(float thetamotor, float theta_desired, float
    thetamotor_dot, PID *ctrl, float* curr_error, float* prev_error, float
    *total_error)
{
    //Integration error
    *curr_error = (theta_desired-thetamotor);
    *total_error = *total_error + (*prev_error + *curr_error)*.001/2;
    float int_error = *total_error;
    //Use the PID controller close to desired angles
    float tau = 0;
    if(fabs(*curr_error) < 0.2) {
        tau = ctrl->Kp*(theta_desired-thetamotor) + ctrl->Kd*(-thetamotor_dot) +
            ctrl->Ki*(int_error);
    } else {
        //Use PD controller over everything else
        *total_error = 0;
        tau = ctrl->Kp*(theta_desired-thetamotor) + ctrl->Kd*(-thetamotor_dot) +
            ctrl->Ki*(0);
    }








    *prev_error = *curr_error;

    //Saturate output torque
    if(tau>5)
        return 5.0;
    else if(tau < -5)
        return -5.0;
    else
        return tau;
}
```

The saturations were applied for integral control effort and torque output, to prevent integral windup and due to hardware limit, respectively. Table 2.1 shows the performance of the controller as it was gradually tuned.

From the Table 2.1, it can be seen that an ideal proportional gain largely decreased overshoot, and steady state error, it was not able to optimize system performance by itself. With K_D added, both rise time and overshoot were further reduced and eventually satisfied the system performance requirements, despite of the fact that mere PD control could not further reduce steady state error. As integral control came into play, steady state error was reduced to less than 1%, though with a tolerable trade off of rise time.

Table 2.1: Robot System PID Tuning and System Performance Results

K_P	K_D	K_I	Rise Time	Overshoot(%)	Steady State Error(%)	Step Response
30	0	0	0.59	22.05	4.89	
60	0	0	0.58	19.29	1.70	
60	1	0	0.33	8.47	1.41	
60	2	0	0.22	0.00	1.41	
60	2	0.3	0.22	0.00	1.41	
60	2	10	0.24	0.00	1.38	
60	2	20	0.25	1.20	0.76	

3 PID PLUS FEED FORWARD CONTROL

A cubic polynomial of the form

$$\theta_i = at^3 + bt^2 + ct + d \quad (3.1)$$

was obtained, identifying the values and derivatives on each step in the trajectory. This allowed us to solve for two sets of parameters $[a, b, c, d]$, one from 0 to 0.5 radians and another from 0.5 radians to 1 radian.(going back to the starting point). The section below shows the Matlab code that was used for deriving the aforementioned parameters.

```

function [y, ydot, yddot] = spline(t)

for i = 0:0.01:1000
    if(i<1)
        x = [0;0;1.5000;-1.0000];
        y = x(1) + x(2)*i + x(3)*i^2 + x(4)*i^3;
        ydot = x(2) + 2*x(3)*i + 3*x(4)*i^2;
        yddot = 2*x(3) + 6*x(4)*i;
        plot(i,ydot,'b+');
        hold on;
    end

    if(i>=1 && i<=2)
        x = [-2.0000;6.0000;-4.5000;1.0000];
        y = x(1) + x(2)*i + x(3)*i^2 + x(4)*i^3;
        ydot = x(2) + 2*x(3)*i + 3*x(4)*i^2;
        yddot = 2*x(3) + 6*x(4)*i;
        plot(i,ydot,'r+');
        hold on;
    end

    if(i>2)
        y = 0;
        ydot = 0;
        yddot = 0;
    end

end
end

```

```

t0 = 0;
tf = 1;
qf = 0.5;
q0 = 0;
v0 = 0;
vf = 0;

A = [1 t0 t0^2 t0^3;
     0 1 2*t0 3*t0^2;
     1 tf tf^2 tf^3;
     0 1 2*tf 3*tf^2];
b = [q0 v0 qf vf]';
x = inv(A)*b

for i = 0:0.01:1
    y = x(1) + x(2)*i + x(3)*i^2 + x(4)*i^3;
    ydot = x(2) + 2*x(3)*i + 3*x(4)*i^2;
    yddot = 2*x(3) + 6*x(4)*i;

```

```

    plot(i,ydot,'r+');
    hold on;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t0 = 1;
tf = 2;
qf = 0;
q0 = 0.5;
v0 = 0;
vf = 0;

A = [1 t0 t0^2 t0^3;
     0 1 2*t0 3*t0^2;
     1 tf tf^2 tf^3;
     0 1 2*tf 3*tf^2];
b = [q0 v0 qf vf]';
x = inv(A)*b

for i = 1:0.01:2
    y = x(1) + x(2)*i + x(3)*i^2 + x(4)*i^3;
    ydot = x(2) + 2*x(3)*i + 3*x(4)*i^2;
    yddot = 2*x(3) + 6*x(4)*i;
    plot(i,ydot,'b+');
    hold on;
end

```

The figure below shows the result:

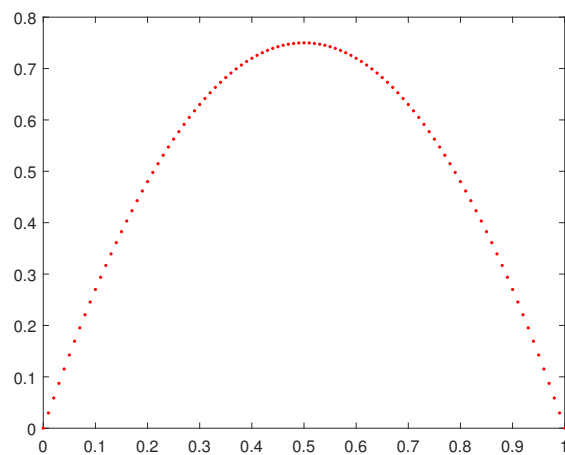


Figure 3.1: Desired $\dot{\theta}$ for the first half of cubic spline trajectory

The cubic polynomial represents a smooth trajectory without any discontinuities in its

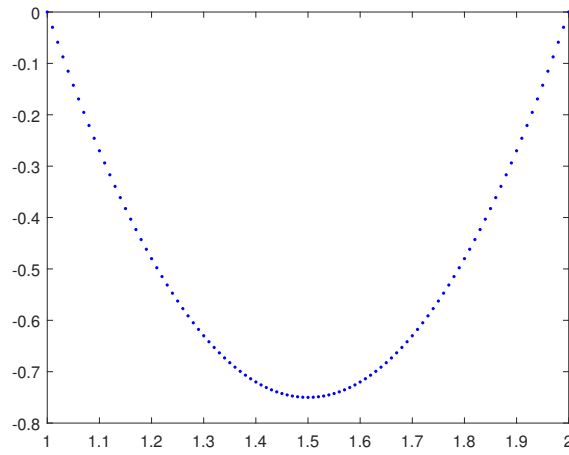


Figure 3.2: Desired $\dot{\theta}$ for the second half cubic spline trajectory

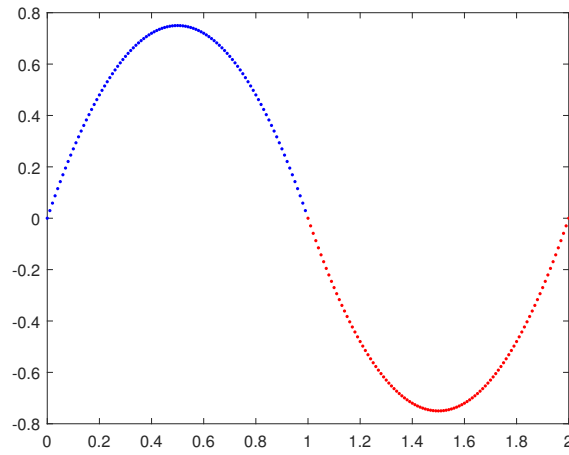


Figure 3.3: Desired $\dot{\theta}$ for the complete cubic spline trajectory

derivatives. This makes trajectory tracking easy for the controller and thus the results shown are a very good match. In this case, the first and second derivatives are needed in the control as they are nonzero. In the previous case, the derivatives are zero, although θ is discontinuous, thus these values are not in the expression.

4 CUSTOM TRAJECTORIES

4.1 INVERSE KINEMATICS

This sub section provides a brief summary of the equations that were derived for the inverse kinematics of the CRS robot arm in the first report. The inverse kinematics for the CRS robot arm are listed below:

$$\begin{aligned}\theta_1 &= a \tan 2(y, x) \\ R &= \sqrt{x^2 + y^2} \\ d &= \sqrt{R^2 + z^2} \\ \beta &= a \tan 2(z, R) \\ \gamma &= a \tan 2\left(\sqrt{L^2 - \frac{d^2}{4}}, \frac{d}{2}\right) \\ \theta_2 &= \frac{\pi}{2} - \gamma - \beta \\ \theta_3 &= \frac{\pi}{2} + \gamma - \beta\end{aligned}$$

4.2 MATLAB CODE FOR INVERSE KINEMATICS

```
function [theta1, theta2, theta3] = inverseK(x, y, z, L)
theta1 = atan2(y,x);
R = sqrt(x^2 + y^2);
d = sqrt(R^2 + z^2);
beta = atan2(z,R);
gamma = atan2(sqrt(L^2 - d^2/4), d/2);
theta2 = pi/2 - gamma - beta;
theta3 = gamma - beta;
```

4.3 FORWARD KINEMATICS FORMULA

The following section gives a brief summary of the equations that were derived for the forward kinematics for the CRS robot arm:

$$\begin{aligned}\theta_3 &= \theta_3 + \frac{\pi}{2} \\ R &= L(\sin(\theta_2) + \sin(\theta_3)) \\ z &= L(\cos(\theta_2) + \cos(\theta_3)) \\ x &= R \cos(\theta_1) \\ y &= R \sin(\theta_1)\end{aligned}$$

4.4 MATLAB CODE FOR FOWARD KINEMATICS

```
function [x, y, z] = forwardK(theta1, theta2, theta3, L)
R = L * ( sin(theta2) + sin(theta3) );
z = L * ( cos(theta2) + cos(theta3) );
x = R .* cos(theta1);
y = R .* sin(theta1);
```

4.5 VERIFICATION OF INVERSE AND FORWARD KINEMATICS

```
>> [theta1,theta2,theta3] = inverseK(13.5,0,8,10)
>> theta1 = 0
>> theta2 = 0.3671
>> theta3 = 1.7046
>> [x,y,z] = forwardK(theta1,theta2,theta3)
>> x = 13.5, y = 0, z = 8
```

4.6 JOINT ANGLE DERIVATIVE CALCULATIONS

To reduce the computation payload on the controller, a numerical approach was adopted for calculating the derivatives and double derivatives of the joint angles of the robot arm. These are required as an input to the controller for trajectory tracking. The computation of the derivatives takes place online as the controller is running. The following steps detail the method used to numerically solve the derivatives of the joint angles.

Expanding the value of θ using the Taylor's series expansion,

$$\theta(\delta t) = \theta_n + \dot{\theta}_n \delta t + \frac{1}{2} \ddot{\theta}_n \delta t^2 + O(\delta t^2) \quad (4.1)$$

In this approximation, θ_n derivatives will be approximated using the three point approach i.e.

$$\delta t_{n-1} = -\delta t$$

$$\delta t_n = 0$$

$$\delta t_{n+1} = +\delta t$$

whose values are:

$$\theta(\delta t_{n-1}) = \theta_{n-1}$$

$$\theta(\delta t_n) = \theta_n$$

$$\theta(\delta t_{n+1}) = \theta_{n+1}$$

The aforementioned equations thus result in the following three equations:

$$\theta_{n-1} = \theta_n - \dot{\theta}_n \delta t + \frac{1}{2} \ddot{\theta}_n \delta t^2 \quad (4.2)$$

$$\theta_{n-1} = \theta_n \quad (4.3)$$

$$\theta_{n+1} = \theta_n - \dot{\theta}_n \delta t + \frac{1}{2} \ddot{\theta}_n \delta t^2 \quad (4.4)$$

Solving these for $\dot{\theta}_n$ and $\ddot{\theta}_n$ we get:

$$\dot{\theta}_n = \frac{\theta_{n+1} - \theta_{n-1}}{2\delta t} \quad (4.5)$$

$$\ddot{\theta}_n = \frac{\theta_{n+1} - 2\theta_n + \theta_{n-1}}{\delta t^2} \quad (4.6)$$

4.7 TRAJECTORY EQUATIONS

So for this section of the lab, two custom trajectories were programmed and tested. The first is defined by the following set of equations and represents a circle in the x-z plane with a radius of 3 inches.

$$x = x_0 + r_0 \sin\left(\frac{2\pi t}{T}\right)$$

$$y = 0$$

$$z = z_0 + r_0 \cos\left(\frac{2\pi t}{T}\right)$$

4.8 MATLAB CODE FOR CIRCULAR TRAJECTORY

```

xr0 = 10.0;
r0 = 4.0;
yr0 = 0;
zr0 = 8.0;
T = 10;

td1n = 0;
td2n = 0;
td3n = 0;
delta = 0.005;

td10 = 0;
td20 = 0;
td30 = 0;
```

```
figure;
```

```

%Infinity trajectory
for t = 0:0.05:20
    phi = 2*pi*t/T;

    x = xr0 + r0*sin(phi);
    y = yr0;
    z = zr0 + r0*cos(phi);

    x0 = xr0 + r0*sin(phi - 2*pi*delta/T);
    y0 = yr0;
    z0 = zr0 + r0*cos((phi - 2*pi*delta/T));

    x2 = xr0 + r0*sin(phi + 2*pi*delta/T);
    y2 = yr0;
    z2 = zr0 + r0*cos((phi + 2*pi*delta/T));

    [td1,td2,td3] = inverseK(x, y, z, 10.0);
    [td10,td20,td30] = inverseK(x0, y0, z0, 10.0);
    [td1n,td2n,td3n] = inverseK(x2,y2,z2,10);

    td1dot = (td1 - td10)/delta/2;
    td2dot = (td2 - td20)/delta/2;
    td3dot = (td3 - td30)/delta/2;

    td1ddot = (td1 - 2*td1n + td10)/(delta*delta);
    td2ddot = (td2 - 2*td2n + td20)/(delta*delta);
    td3ddot = (td3 - 2*td3n + td30)/(delta*delta);

    subplot(3,1,1)
    plot(t,td1,'r. ');
    plot(t,td2,'g. ');
    plot(t,td3,'b. ');
    title('Theta dot')
    hold on;

    subplot(3,1,2)
    plot(t,td1ddot,'r. ');
    plot(t,td2ddot,'g. ');
    plot(t,td3ddot,'b. ');
    hold on;
    title('Theta doubledot')

    subplot(3,1,3)
    plot(x2,z2,'r. ');
    title('Trajectory')
    hold on;

end

```

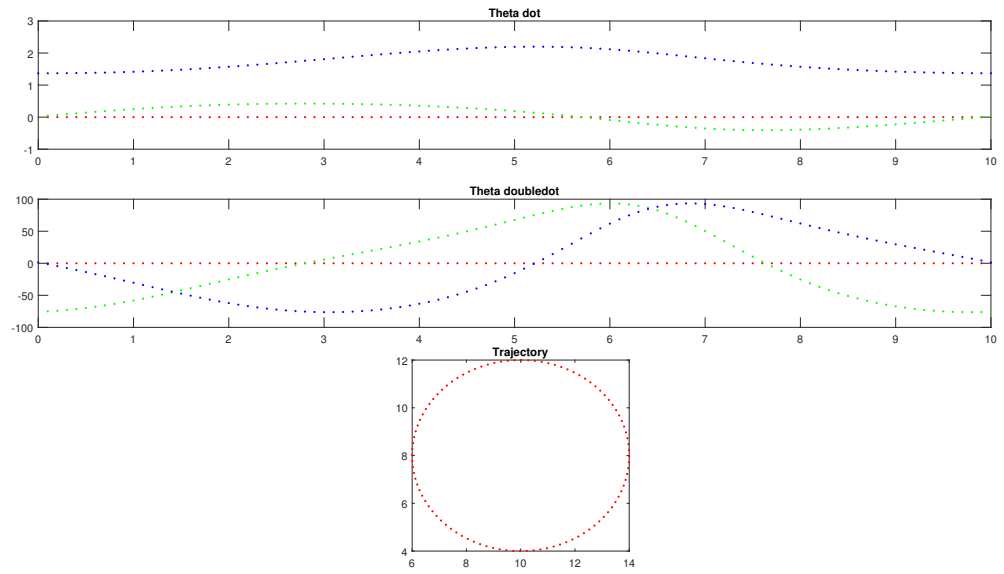


Figure 4.1: Desired $\dot{\theta}$, $\ddot{\theta}$ and Trajectory

4.9 MATLAB CODE FOR *Infinity* SHAPED TRAJECTORY

```

xr0 = 10.0;
r0 = 6.0;
yr0 = 0;
zr0 = 8.0;
T = 10;

td1n = 0;
td2n = 0;
td3n = 0;
delta = 0.005;

td10 = 0;
td20 = 0;
td30 = 0;

figure;

%Infinity trajectory
for t = 0:0.05:60

    %define time period
    phi = 2*pi*t/T;

    scale = 2/(3 - cos(2*phi));

```

```

x = xr0 + r0*scale * cos(phi);
y = yr0;
z = zr0 + r0*scale * sin(2*phi)/ 2;

x0 = xr0 + r0*scale * cos(phi - 2*pi*delta/T);
y0= yr0;
z0 = zr0 + r0*scale * sin(2*(phi - 2*pi*delta/T))/2;

x2 = xr0 + r0*scale * cos(phi + 2*pi*delta/T);
y2= yr0;
z2 = zr0 + r0*scale * sin(2*(phi + 2*pi*delta/T))/2;

[td1,td2,td3] = inverseK(x, y, z, 10.0);
[td10,td20,td30] = inverseK(x0, y0, z0, 10.0);
[td1n,td2n,td3n] = inverseK(x2,y2,z2,10);

td1dot = (td1 - td10)/delta/2;
td2dot = (td2 - td20)/delta/2;
td3dot = (td3 - td30)/delta/2;

td1ddot = (td1 - 2*td1n + td10)/(delta*delta);
td2ddot = (td2 - 2*td2n + td20)/(delta*delta);
td3ddot = (td3 - 2*td3n + td30)/(delta*delta);

subplot(3,1,1)
plot(t,td1,'r. ');
plot(t,td2,'g. ');
plot(t,td3,'b. ');
title('Theta dot')
hold on;

subplot(3,1,2)
plot(t,td1ddot,'r. ');
plot(t,td2ddot,'g. ');
plot(t,td3ddot,'b. ');
hold on;
title('Theta doubledot')

subplot(3,1,3)
plot(x2,z2,'r. ');
title('Trajectory')
hold on;
end

```

The radius used for the "infinity" trajectory was 6.0 inches and radius for the circular trajectory was set as 4.0 inches. The values of x_0, y_0, z_0 were adjusted on each case to ensure the path didn't reach out of the safe volume.

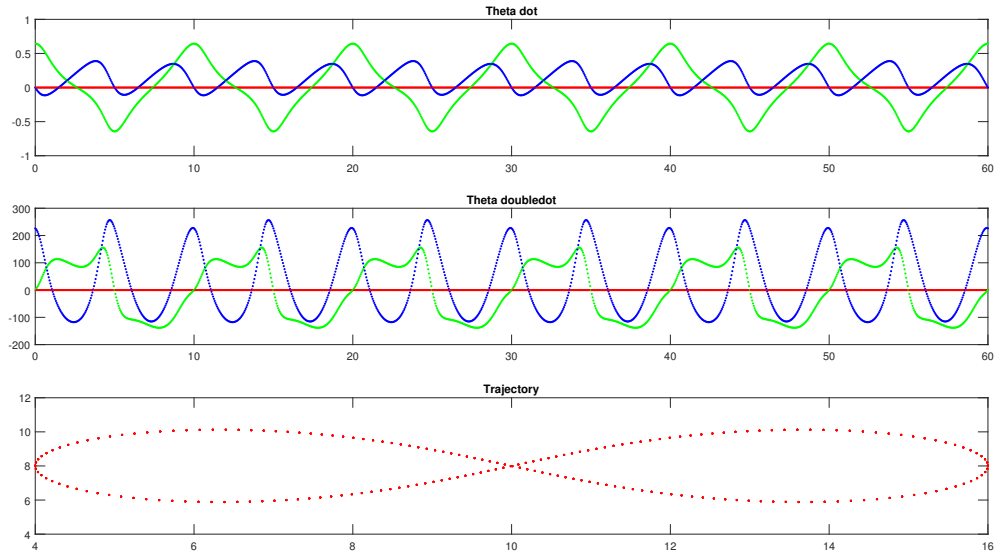


Figure 4.2: Desired $\dot{\theta}$, $\ddot{\theta}$ and Trajectory

4.10 IMPLEMENTATION AND RESULTS

The cubic spline trajectory generation and following was studied in depth to gain an intuitive understanding of the system. Next step was to program the microcontroller for trajectory generation along with the numerical derivation of the desired joint angles, at each instant in the trajectory. This made the θ_d , $\dot{\theta}_d$ and $\ddot{\theta}_d$ available for our control algorithm which uses a combination of PD, PID and feedforward controllers.

Using a serial connection between Matlab Simulink and the microcontroller, the motor angles as the robot moves through the trajectory, are recorded using the Scope. The following figures show long exposure pictures of the trajectories that were thus created with the arm.



Figure 4.3: Long exposure photo of the circle trajectory

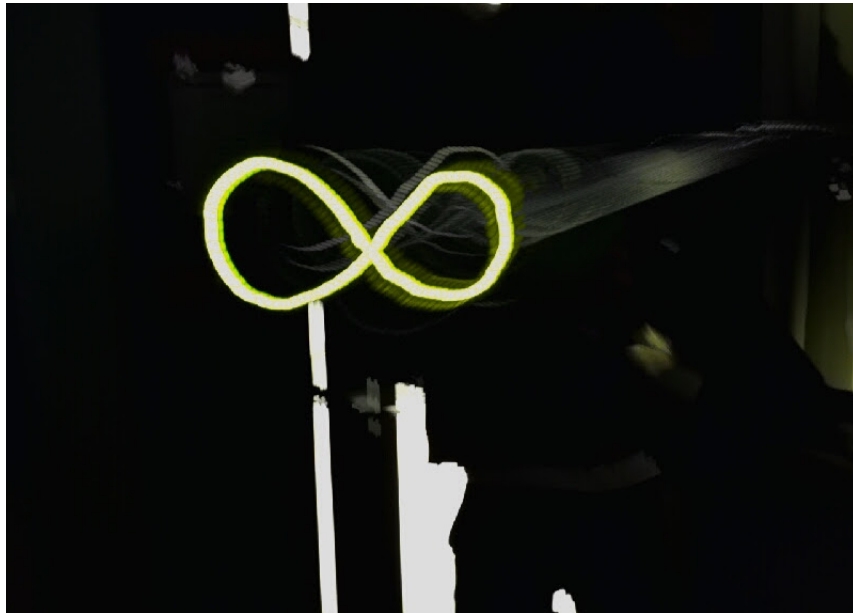


Figure 4.4: Long exposure photo of the *infinity* trajectory

5 APPENDIX

This section gives the microcontroller code that was written to program the CRS arm.

```
/*
Function trajectoryGenerator takes in mycount as its input
Updates theta, thetaDot and thetaDoubleDot based on trajectory values
*/
static inline void trajectoryGenerator(float t, char c)
{
    float time = 0.001*t; //input t is mycount 1000*mycount = 1 sec
    float phi = 2*PI*time/period; //constant definition
    float delta = 0.005; //Sampling time period
    if(c=='c') //For circular trajectory
    {
        if( time <= 100*period) //run the trajectory a hundred times
        {
            //Sample trajectory at T_n
            x = xr0 + Rr0 * sin(phi);
            y = y0;
            z = zr0 + Rr0 * cos(phi);

            //Sample trajectory at T_{n-1}
            float x0 = xr0 + Rr0* sin(phi - PI*delta*2/period);
            y0 = 0;
            float z0 = zr0 + Rr0*cos(phi - PI*delta*2/period);

            //Sample trajectory at T_{n+1}
            float x2 = xr0 + Rr0* sin(phi + PI*delta*2/period);
            float y2 = 0;
            float z2 = zr0 + Rr0*cos(phi + PI*delta*2/period);

            //Get desired theta values using the inverse kinematics of the robot
            arm
            InverseK(x, y, z, 10.0, tdn[0], tdn[1], tdn[2]);
            InverseK(x0, y0, z0, 10.0, td0[0], td0[1], td0[2]);
            InverseK(x2, y2, z2, 10.0, td[0], td[1], td[2]);

            //Calculate thetaDot and thetaDoubleDot using discretization of
            derivative
            //and double derivative
            for (int i = 0; i < 3; ++i)
            {
                tdDot[i] = (td[i]-td0[i])/delta/2;
                tdDoubleDot[i] = (td[i] - 2*tdn[i] + td0[i])/(delta*delta);
            }
        }
    }
    else if(c=='i') //For infinity trajectory
```



```

{
    if( time <= 100*period)
    {
        //Sample trajectory at T_n
        x = xr0 + Rr0 * scale *cos(phi);
        y = 0;
        z = zr0 + Rr0 * scale*sin(2*phi)/2;

        //Sample trajectory at T_{n-1}
        float x0 = xr0 + Rr0 * scale*cos(phi - PI*delta*2/period);
        y0 = 0;
        float z0 = zr0 + Rr0 *scale*sin(2*(phi - PI*delta*2/period))/2;

        //Sample trajectory at T_{n+1}
        float x2 = xr0 + Rr0*scale*cos(phi + PI*delta*2/period);
        float y2 = 0;
        float z2 = zr0 + Rr0*scale*sin(2*(phi + PI*delta*2/period))/2;

        //Get desired theta values using the inverse kinematics of the robot
        arm
        InverseK(x, y, z, 10.0, tdn[0], tdn[1], tdn[2]);
        InverseK(x0, y0, z0, 10.0, td0[0], td0[1], td0[2]);
        InverseK(x2, y2, z2, 10.0, td[0], td[1], td[2]);

        //Calculate thetaDot and thetaDoubleDot using discretization of
        derivative
        //and double derivative
        for (int i = 0; i < 3; ++i)
        {
            tdDot[i] = (td[i]-td0[i])/delta/2;
            tdDoubleDot[i] = (td[i] - 2*tdn[i] + td0[i])/(delta*delta);
        }
    }
}
else //Otherwise throw an error
    printf("Trajectory Name unrecognized\n");
}

```
