INDEPENDENT JOINT CONTROL AND TRAJECTORY PLANNING

PUMA

CONTENTS

AIM OF PROJECT:

Robots are abundantly used in modern industries. Robots are making their way into the commercial district too. Robots' usefulness made them a valuable tool. As we put robots to perform harder and more intricate tasks, better control mechanism were a requirement. The control mechanism can be complex, calculation-intensive, and precise, for specifying tasks or, can be simple and generally implemented.

Our aim in this project is to delve into the control mechanism of manipulators. Implement an Independent Joint Controller for the PUMA robot. Plan the trajectory for the movement between two points and examine the controller performance.

METHODOLOGY:

We are using an Independent Joint controller schema to control the Motors. As the name suggests these controllers do not depend on the motion of the other joints. Each controller is independently controlling the position and speed of the motor.

The independent joint controller receives its input from the trajectory planner. The trajectory planner is a higher-level controller. The trajectory planner controls all the independent joint controllers in the robot.

The Trajectory planner receives its inputs from the user. The inputs are in the form of initial position and final position. The trajectory planner generates a time of angles from the initial to the final position. The planner uses the robot's inverse kinematic equations and joint space to generate its output.

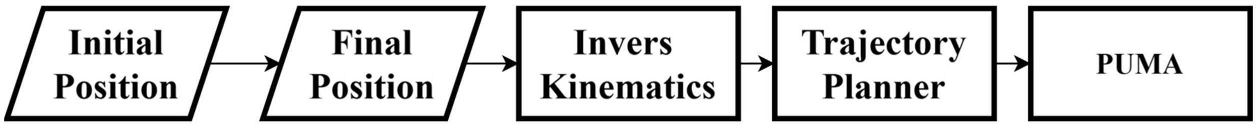


Figure 1 Shows the process flow for the PUMA robot.

The trajectory planner Inputs are initial and final position coordinates. These coordinates are passed to the Inverse Kinematic equitation of the robot. The equation outputs the joint angles of the robot. These joint angles are robot joint angles to reach the Initial and final position with the end effector.

After the inverse kinematics block, initial and final joint angles are passed to the joint space trajectory planner. The joint space planner fills in the joint angles between the initial and final joint angles for all the joints of the robot. The output is a time series of joint angles for each joint in the robot.

The time series of the joint is passed to its respective joint's independent joint controller.

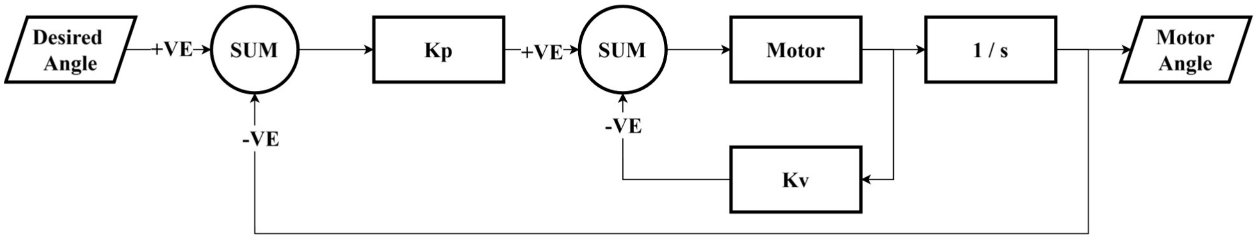


Figure 2 Shows the Independent Joint Controller diagram. This double feedback controller configuration allows control over the position as well as the speed of the motor. Where K_p and K_v are the gain values. The 1/S block is the integration.

The Independent joint controller takes the time series of joint angles and goes through the angles respective to the time provided by the time series. The independent joint controller has two feedback the feedback with K_p regulates the position of the motor. On the other hand, the feedback with K_v use regulates the rate of change of position. This controller is also called Proportional plus Velocity feedback controller.

The Joint controller rotates the joint from the initial to the final joint angle which ultimately causes the end effector to move and follow a trajectory. To find this trajectory we have taken the joint angles of the robotic arm and used the forward kinematic equation to get the position of the end effector.

By continuously plotting the position of the end effector, we will have a trajectory of the end effector in the 3D space.

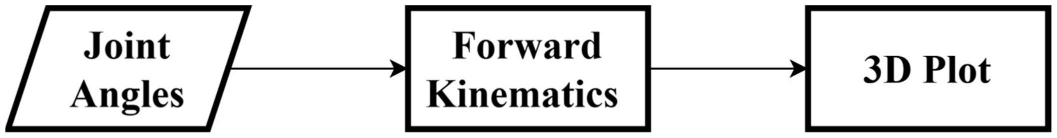


Figure 3 Shows the End Effector Position Plotting Process.

INDEPENDENT JOINT CONTROLLER CALCULATIONS:

Previous Joint Controller:

Following is the previous joint controller that was implemented on the Robot.

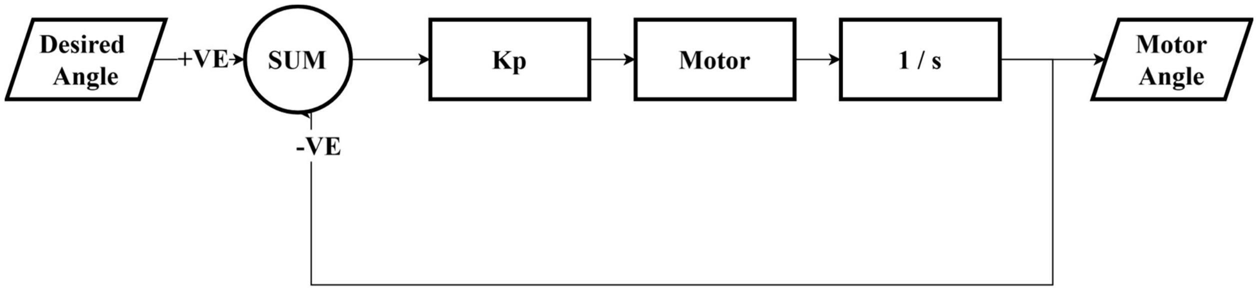


Figure 4 Shows the previous Joint controller.

The above controller only has one feedback loop. The controller keeps track of the current joint angle and based on the difference from the desired joint angle powers the motor. This is also called Proportional Controller (P-controller).

We have added velocity feedback making a proportional plus Velocity feedback controller.

The velocity feedback acts as a damping system. The velocity feedback reduces overshot and oscillation increasing the system stability.

K_p and K_v Calculation:

The figure 2 motor can be represented by:

$$\frac{K}{T_m s + 1}$$

Complete system including the feedback and gain represented by:

$$\frac{KK_p}{s(T_m s + K_v K + 1) + KK_p}$$

Thus, in this system frequency is given by:

$$\omega_n^2 = \frac{KK_p}{T_m}$$

$$K_p = \frac{(\omega_n^2 * T_m)}{K}$$

And damping is given by:

$$2\zeta\omega = \frac{KK_v + 1}{T_m}$$

$$K_v = \frac{2\zeta\omega * T_m - 1}{K}$$

Where T_m and K are motor constants given by:

$$T_m = \frac{R_a * I_{effective}}{R_a * f_{effective} + K_b K_a}$$

$$K = \frac{K_a}{R_a * f_{effective} + K_b * K_a}$$

Where the R_a is the motor coil resistance, K_b and K_a are the motor back-emf and torque constants respectively and $I_{effective}$ is the effective moment of inertia the motor will face and $f_{effective}$ is the joint resistance.

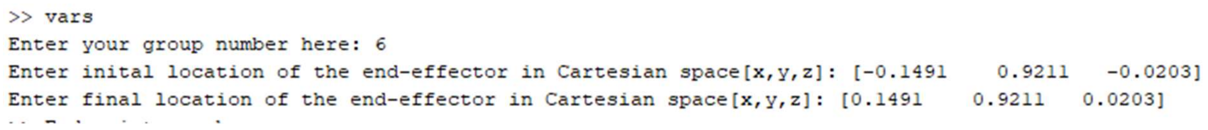
Simulation Setup:

We have one Simulink file named my_puma.mdl and two MATLAB files named vars.m and End_point_graph.m. The vars file does all the calculations of gain and prepares the joint angle time series.

The other MATLAB file End_point_graph.m is executed in the callback at the end of the Simulink simulation. This MATLAB script takes the joint angles of the robot from the Simulink simulation and runs through forward kinematics to find the endpoint x,y, and z coordinates of respective joint angles. The x,y, and z are plotted in plot3 to get the trajectory of the endpoint in 3D Space.

PLOTS:

We have applied the following commands to the MATLAB script.



```
>> vars
Enter your group number here: 6
Enter inital location of the end-effector in Cartesian space[x,y,z]: [-0.1491    0.9211   -0.0203]
Enter final location of the end-effector in Cartesian space[x,y,z]: [0.1491    0.9211   0.0203]
```

Figure 5 The initial and final points are defined in the command.

Initial Position = [-0.1491,0.9211,-0.023]

These points have come from the DH parameters.

Final position = [0.1491,0.9211,0.023]

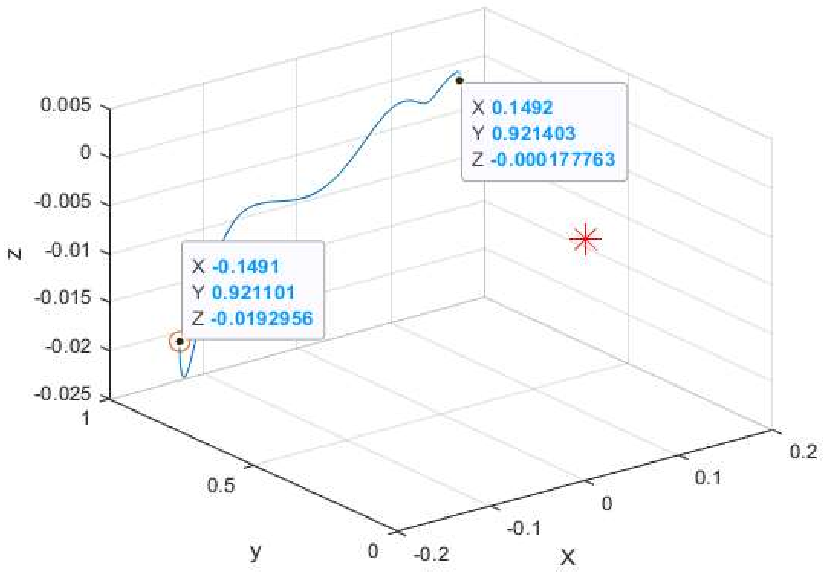


Figure 6 The graph shows the end effector movement in 3D space

The 3D graph shows the motion of the end effector from the initial to the final position. The starting point is marked with a circle. The red star shows the origin of the reference frame.

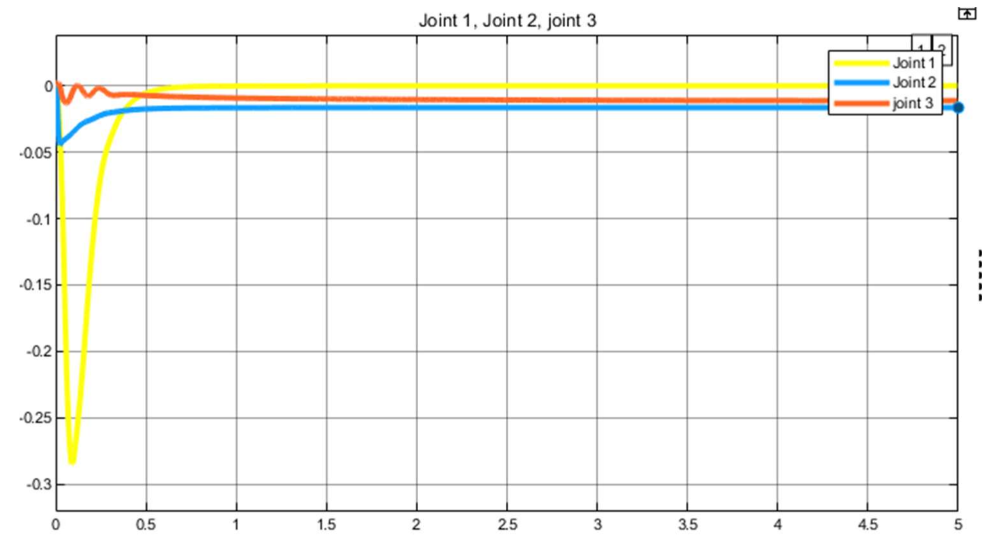


Figure 7 The graph shows the position error.

Following is the actual joint velocity graph.

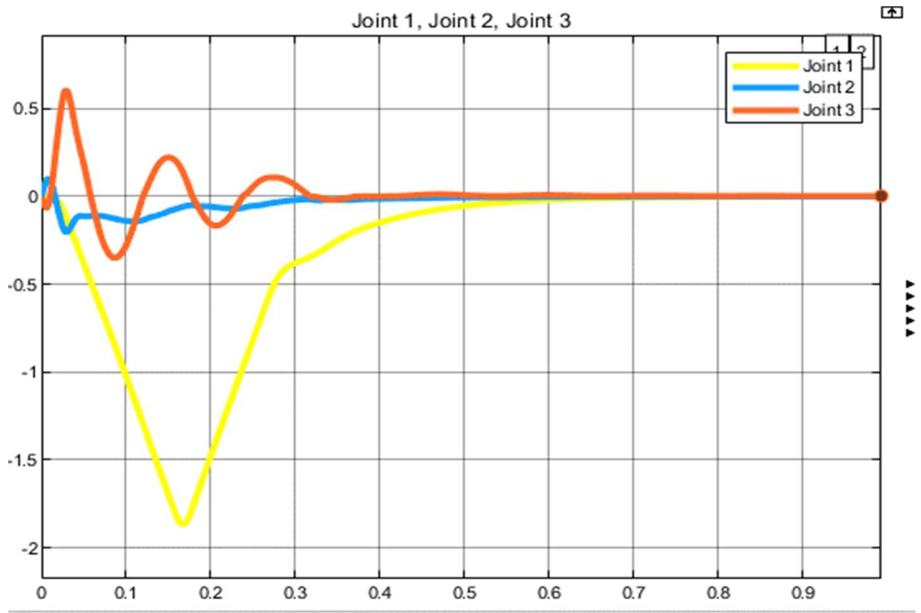
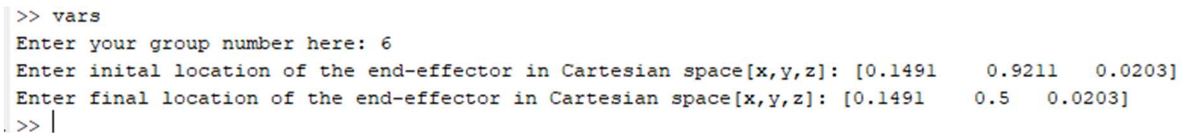


Figure 8 The graph shows the actual robot joint Velocity.



```
>> vars
Enter your group number here: 6
Enter inital location of the end-effector in Cartesian space[x,y,z]: [0.1491    0.9211   0.0203]
Enter final location of the end-effector in Cartesian space[x,y,z]: [0.1491    0.5   0.0203]
>> |
```

Figure 9 The providing the initial and final position.

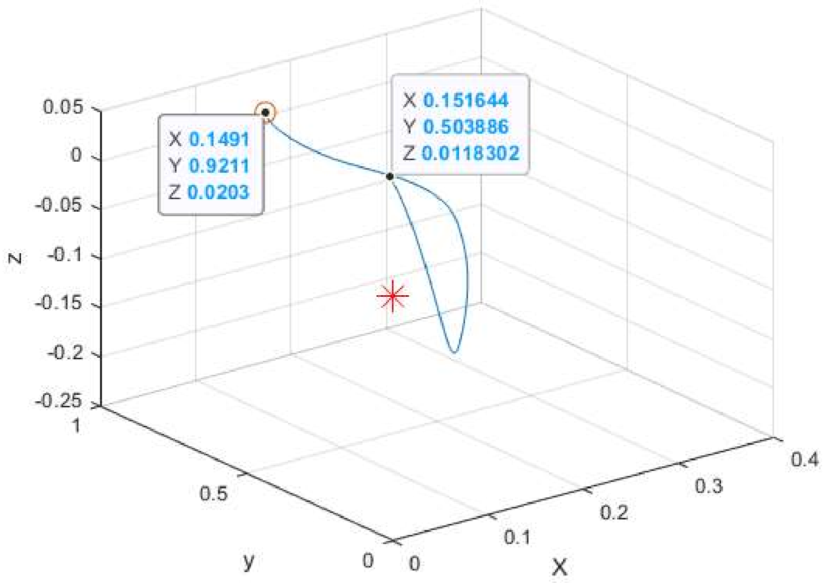


Figure 10 The graph shows the 3D plot of the end effector position.

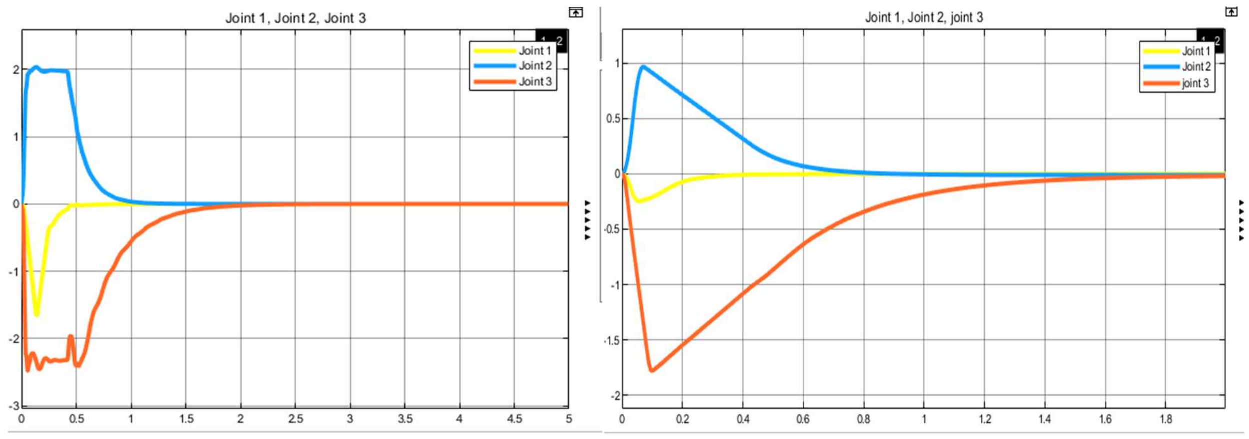


Figure 11 The left-side graph shows the robot joint velocity while in the right-side graph shows the position error.

DISCUSSION:

In the Figure 7 graph, we have plotted the position error vs time. We can see that there is some steady-state error in the joint 2 and 3. We can measure the error to be less than 0.025 radian.

In Figure 6 we can confirm by comparing the end effector final positions. In the command, the final position is [0.149, 0.921, 0.02]. but in the 3D plot, the end effector has a z value of about zero.

Desired -actual = error

0.02-0 = 0.02

This error of 0.02.

The Figure 8 graph shows the velocity of robot joints we can observe that join3 has under damped response.

We can trace this under-damped response to the end effector 3D plot. We can observe some oscillation in the movement from the initial to the final position.

Figure 9 shows the commands for a secondary run. This time the initial and final positions are [0.1491 0.9211 0.0203] and [0.1491 0.5 0.0203] respectively.

Figure 10 shows the end effector 3D plot. We can observe the initial and final positions. By comparing the given and observed final position we can state there is about 0.01 rad steady state error.

Figure 11 on the right-hand side graph we can see the velocity graph. The small oscillation of joint 3 visible in 1st run is still causing overshoot.

CONCLUSION:

We can conclude the independent joint controller is not good for very small and minute movements. Independent joint controllers are good for tasks such as picking and placing, welding, and large cutting.

CODE:

File vars.m:

```
% file 1
% This function assigns the global values required to run the PUMA
% simulation.  It should be called before the simulation can be performed.
% In the function, 'thr10' is the initial angular position (in radius) of the
first joint
% of the PUMA.  Consequently, 'thr20' and 'thr30' represent the initial
positions
% of the second and third joints of the PUMA.
% 'ng1' is the gear ratio of the first joint driving train >1.
% 'ng2' is the gear ratio of the 2nd joint driving train >1.
% 'ng3' is the gear ratio of the 3rd joint driving train >1.
%  this program must be run before the simulation.
%
% -0.1491    0.9211   -0.0203

global gn thr10 thr20 thr30 ng1 ng2 ng3

gn=input('Enter your group number here: '); % get your group number

% gets the required data
inital_loc = input('Enter inital location of the end-effector in Cartesian
space[x,y,z]: ');
final_loc  = input('Enter final location of the end-effector in Cartesian
space[x,y,z]: ');




% these values below are the reciprocals of the gear ratio numbers given in
% the assignment sheet. Keep them as greater than 1.
ng1=40+gn*4;
ng2=100-gn*2;
ng3=60+gn;


%% group parameters

m1 = 13 + gn/2;
m2 = 21 + gn/4;
m3 = 5  + gn/6;
Mload = 1  + gn/5;
%Nominal maximum velocity ratio
u_v = 0.3-gn*0.02;
%Nominal maximum acceleration ratio
u_a = 0.3+gn*0.02;

%% Motor Controle Parameter

% damping parameter
```

```
zeta1 = 1;
zeta2 = 1;
zeta3 = 5;

%% given data from the Table
% DH-parameter a and d paramter for the Robot
a1 = 0;
a2 = 0.4318;
a3 = 0.0203;
d1 = 0;
d2 = 0.1491;
d3 = 0;
d4 = 0.43307;
d6 = 0.05625;

robo_param = [a1,a2,a3,d1,d2,d3,d4,d6];

%--------------------------------------------
% Centre of mass for the 1st link
x1 = 0;
z1 = 0.3088;
y1 = 0.0389;

% Centre of mass for the 2nd link
x2 = -0.3289;
y2 = 0;
z2 = 0.2038;

% Centre of mass for the 3rd link
x3 = 0.0204;
y3 = 0.0137;
z3 = 0.1244;

%--------------------------------------------
% Moment of inertia for the 1st link
i1xx = 2.35;
i1yy = 0.2;
i1zz = 2.35;

% Moment of inertia for the 2nd link
i2xx = 1.33;
i2yy = 3.03;
i2zz = 3.38;

% Moment of inertia for the 3rd link
i3xx = 0.3128;
i3yy = 0.3148;
i3zz = 0.01;

%--------------------------------------------
% Moment of inertia of the motor armature driving the joints
Im = 0.0005;

% Effective viscous friction coefficient of the joints
```

```
F_eff = 0.002;

% Motor armature resistance
R_a = 1.2;

% Motor torque constant
K_b = 0.2531;
K_a = 0.2531;

% Rated voltage of the motor
rated_voltage_motor1 = 41;
rated_voltage_motor2 = 37;
rated_voltage_motor3 = 32.5;

%Rated torque of the motor
rated_torque_motor1 = 1.25;
rated_torque_motor2 = 1.5;
rated_torque_motor3 = 1.75;






%% organizing Data
% Centre of mass for the links
cm1 = [x1; y1; z1 ;1];
cm2 = [x2; y2; z2 ;1];
cm3 = [x3 ;y3 ;z3 ;1];


%%  Calculation

% Transfer function Calculation from the DH-parameter
syms thi1 thi2 thi3; % The variable for theta in DH-parameter

%transform function is user-defined you can find its code in the end of
%this code under the user defined function section
t1 = transform(d1, thi1+(pi/2), a1 ,-pi/2);
t2 = transform(d2, thi2,a2, 0);
t3 = transform(d3, thi3-(pi/2),a3,-pi/2);
t4 = transform((d4+d6),   0, 0,  0);

% solving the syms matrix to get the transfer function
% at angles thi1 = thi2 = thi3 = 0 the inital position
T1 = vpa(subs(t1,0));
T2 = vpa(subs(t2,0));
T3 = vpa(subs(t3,0));
T4 = vpa(subs(t4,0));


%% Motor 1 Load
% In these calculations we will find the load faced by the Motor 1
% Center of mass of link 2 "cm2" mapped to refrence frame zero
cm2_0 = double(T1*cm2);

% Center of mass of link 3 "cm3" mapped to refrence frame zero
cm3_0 = double(T1*T2*cm3);

% Center of mass of load mapped to refrence frame zero
% Considering the load’s center of mass is at the end of link 3.
% thus (0,0,0) poistion
load_0= double(T1*T2*T3*T4*[0;0;0;1]);


% for projecting moment of inertia of a body at any point
% we will use the parallel-axis theorem

% projecting of link 1 moment of inertia to motor 1
IL1_1 = i1zz + m1.*(sum(cm1(1:3).^2));

% projecting of link 2 moment of inertia to motor 1
IL1_2 = i2yy + m2.*(sum(cm2_0(1:3).^2));

% projecting of link 3 moment of inertia to motor 1
IL1_3 = i3yy + m3.*(sum(cm3_0(1:3).^2));

% projecting of Load moment of inertia to motor 1
IL1_4 = Mload.*(sum(load_0(1:3).^2));

% Total robot moment of inertia seen by the motor 1
IL1 = IL1_1 + IL1_2 + IL1_3 + IL1_4;

% effectice moment of inertial seen by the motor armature
I_eff1 = Im + ((1/ng1)^2).*(IL1);

%% Motor 1 gain Calculatoin
% in these calculaton we used the motor transfer function to calculate
% gains for out controllers

T_m1 = (R_a .* I_eff1)./(R_a.*F_eff + K_a.*K_b);
T_m1 = double(T_m1);
K1    = (K_a)./(R_a.*F_eff + K_a.*K_b);


w_n1 = 20;% natural Frequency <0.5*Resonant_Frequency


% Propotional controller Gain
k_p1 = (w_n1^2).*T_m1 ./ K1;

% Velocity Feedback Gain
k_v1 = 2*w_n1*zeta1*T_m1./(K1+1);


%% Motor 2 Load
```

```
% In these calculations we will find the load faced by the Motor 2

% Center of mass of link 3 "cm3" mapped to refrence frame 1
cm3_1 = double(T2*cm3);

% Center of mass of load mapped to refrence frame 1
% Considering the load’s center of mass is at the end of link 3.
% thus (0,0,0) poistion
load_1= double(T2*T3*T4*[0;0;0;1]);


% for projecting moment of inertia of a body at any point
% we will use the parallel-axis theorem

% projecting of link 2 moment of inertia to motor 2
IL2_2 = i2zz + m2.*(sum(cm2(1:3).^2));

% projecting of link 3 moment of inertia to motor 2
IL2_3 = i3zz + m3.*(sum(cm3_1(1:3).^2));

% projecting of Load moment of inertia to motor 2
IL2_4 = Mload.*(sum(load_1(1:3).^2));

% Total robot moment of inertia seen by the motor 2
IL2   = IL2_2 + IL2_3 + IL2_4;

% effectice moment of inertial seen by the motor armature
I_eff2 = Im + ((1/ng2)^2).*(IL2);

%% Motor 2 gain Calculatoin
% in these calculaton we used the motor transfer function to calculate
% gains for out controllers

T_m2 = (R_a .* I_eff2)./(R_a.*F_eff + K_a.*K_b);
T_m2 = double(T_m2);
K2   = (K_a)./(R_a.*F_eff + K_a.*K_b);

w_n2 = 20;% natural Frequency <0.5*Resonant_Frequency

% Propotional controller Gain for motor 2
k_p2 = (w_n2^2).*T_m2 ./ K2;

% Velocity Feedback Gain for motor 2
k_v2 = 2*w_n2*zeta2*T_m2./(K2+1);

%% Motor 3 Load
% In these calculations we will find the load faced by the Motor 2

% Center of mass of load mapped to refrence frame 1
% Considering the load’s center of mass is at the end of link 3.
% thus (0,0,0) poistion
load_2= double(T3*T4*[0;0;0;1]);
```

```
% for projecting moment of inertia of a body at any point
% we will use the parallel-axis theorem

% projecting of link 3 moment of inertia to motor 3
IL3_3 = i3zz + m3.*(sum(cm3(1:3).^2));

% projecting of Load moment of inertia to motor 3
IL3_4 = Mload.*(sum(load_2.^2));

% Total robot moment of inertia seen by the motor 3
IL3 = IL3_3 + IL3_4;

% effectice moment of inertial seen by the motor armature
I_eff3 = Im + ((1/ng3)^2).*(IL3);

%% Motor 3 gain Calculatoin
% in these calculaton we used the motor transfer function to calculate
% gains for out controllers

T_m3 = (R_a .* I_eff3)./(R_a.*F_eff + K_a.*K_b);
T_m3 = double(T_m3);
K3   = (K_a)./(R_a.*F_eff + K_a.*K_b);

w_n3 = 30;% natural Frequency <0.5*Resonant_Frequency

% Propotional controller Gain for motor 2
k_p3 = (w_n3^2).*T_m3 ./ K3;

% Velocity Feedback Gain for motor 2
k_v3 = 2*w_n3*zeta3*T_m3./(K3+1);

%% Forward Kinmatics

% the 4th colom in the DH matrix are the forward kinmatic equations
temp = t1*t2*t3*t4;
forward_kinmatics = temp(1:3,4);



%% Inverse Kinmatics
%[-0.1491,d4+d6+a2,a3] coordinates at angle [pi/2,0,pi/2]
ARM = -1;   % -1 for left arm +1 for right arm
ELBOW = -1; %  +1 for above arm -1 for below arm

% the functions for Inverse Kinmatics are in the user defined function
% section



%% the joint positions can be calculated using inverse kinematics equations

% assigning the value of inital location x,y,z
xi = inital_loc(1);
```

```
yi = inital_loc(2);
zi = inital_loc(3);

% assigning the value of final location x,y,z
xf = final_loc(1);
yf = final_loc(2);
zf = final_loc(3);

% using the Function Theta1 Theta2 and Theta3.
% these are the inverst Kinmatics functions
% these function take position as input and outputs the joint angles.
joint_angles_motor1_inital = theta1(xi,yi,zi,robo_param,ARM,ELBOW);
joint_angles_motor2_inital = theta2(xi,yi,zi,robo_param,ARM,ELBOW);
joint_angles_motor3_inital = theta3(xi,yi,zi,robo_param,ARM,ELBOW);

inital_angles = double([joint_angles_motor1_inital,...
    joint_angles_motor2_inital, joint_angles_motor3_inital]);

% Change theta initial positions
thr10 = inital_angles(1);
thr20 = inital_angles(2);
thr30 = inital_angles(3);

joint_angles_motor1_final = theta1(xf,yf,zf,robo_param,ARM,ELBOW);
joint_angles_motor2_final = theta2(xf,yf,zf,robo_param,ARM,ELBOW);
joint_angles_motor3_final = theta3(xf,yf,zf,robo_param,ARM,ELBOW);

final_angles = double([joint_angles_motor1_final,...
    joint_angles_motor2_final, joint_angles_motor3_final]);

%% Motors speed

% motor max speed
w_max_motor1 = rated_voltage_motor1 ./ K_a;
w_max_motor2 = rated_voltage_motor2 ./ K_a;
w_max_motor3 = rated_voltage_motor3 ./ K_a;

% motor max acceleration.
alpha_max_1 = rated_torque_motor1/I_eff1;
alpha_max_2 = rated_torque_motor2/I_eff2;
alpha_max_3 = rated_torque_motor3/I_eff3;

alpha_max = double([alpha_max_1,alpha_max_2,alpha_max_3]);
speeds  = [w_max_motor1, w_max_motor2, w_max_motor3 ].*u_v;

ratio_1 = (speeds(1).^2) / alpha_max_1;
ratio_2 = (speeds(2).^2) / alpha_max_2;
ratio_3 = (speeds(3).^2) / alpha_max_3;
ratio = [ratio_1, ratio_2, ratio_3];

dif = final_angles - inital_angles;
comparison = (abs(dif) > ratio);
%% trajectory planing
```

```
% using the joint space samll and large movement controller for motots

n = 1;
syms t
for i = comparison % compation checks if the movemment is large or samell.
    if(i == 1)

        %large Movement
        % calculationg lenght of the input singnal in time.
        t_c = (abs(dif(n))-(0.5 * ratio(n)* 2)) / speeds(n);
        t_f = t_c + 2 * (speeds(n) / alpha_max(n));
        t_a = speeds(n) / alpha_max(n);
        syms t;

        % piceing togather  the different graphs formaing a movement
        % pattren
        pice_1 = inital_angles(n) + (0.5*alpha_max(n) *t.^2)*sign(dif(n));
        pice_2 = inital_angles(n) + ...
        (0.5*alpha_max(n) *t_a.^2)*sign(dif(n)) +...
        (speeds(n)*(t - t_a))*sign(dif(n));
        pice_3 = inital_angles(n) + (0.5*alpha_max(n) *t_a.^2)*sign(dif(n)) +
...
        (speeds(n)*(t - t_a))*sign(dif(n)) - ...
        (0.5*alpha_max(n)*(t - t_a - t_c).^2)*sign(dif(n));
        q = piecewise(((t>0)&(t <= t_a)), pice_1 , ((t_a < t)&(t <= (t_a +
t_c))), pice_2 , (((t_a + t_c) < t) & (t<= t_f)), pice_3);
    end

    if(i == 0)

        %small Movement

        t_f = 2.*(abs(dif(n)) ./ alpha_max(n)).^(1/2);
        pice_1 = inital_angles(n) + (0.5*alpha_max(n) *t.^2)*sign(dif(n));
        pice_2 = inital_angles(n) + ...
        (0.5*alpha_max(n) *(0.5*t_f).^2)*sign(dif(n)) +...
        (alpha_max(n)*(0.5*t_f)*(t - 0.5*t_f))*sign(dif(n)) - ...
        (0.5*alpha_max(n)*(t - 0.5*t_f).^2)*sign(dif(n));

        q = piecewise((t <= 0.5*t_f),pice_1,((0.5*t_f < t)&(t < t_f) )
,pice_2);
    end
    % creating a time vector
        time = linspace(0,t_f,1000);
        % solving the equations to for its values.
        z = double(subs(q,time));
        z = [transpose(time),transpose(z)];
        z(isnan(z)) = 0;

        if n == 1
            movement = z;
        end
        if n > 1
            movement = [movement , z];
        end
        n = n+1;
end
% removing the last listing in the list.
movement = movement(1:end-1,:);
movement(1,:) = [0,inital_angles(1),0,inital_angles(2),0,inital_angles(3)];
% the indiemdence data.
% multipliing the gair ratio.
movement(:,2) = movement(:,2)*ng1;
movement(:,4) = movement(:,4)*ng2;
movement(:,6) = movement(:,6)*ng3;


Motor1_angles = movement(:,[1,2]);
Motor2_angles = movement(:,[3,4]);
Motor3_angles = movement(:,[5,6]);




%% user Defined Function

% function for creating Transfer function matrix from the DH-parameters
function T = transform(d,thi,a,alpha)
T = [
     cos(thi),  -cos(alpha).*sin(thi),  sin(alpha).*sin(thi), a.*cos(thi)
     sin(thi),   cos(alpha).*cos(thi), -sin(alpha).*cos(thi), a.*sin(thi)
     0,          sin(alpha),            cos(alpha),           d
     0,          0,                     0,                    1
     ];
end


function angle = theta3(x,y,z,robo_param,ARM,ELBOW)

    a1 = robo_param(1);
    a2 = robo_param(2);
    a3 = robo_param(3);
    d1 = robo_param(4);
    d2 = robo_param(5);
    d3 = robo_param(6);
    d4 = robo_param(7);
    d6 = robo_param(8);

    R = sqrt(power(x,2)+power(y,2)+power(z,2)-power(d2,2));
    if(z>=0)
        cos_phi = (a2^2+((d4+d6)^2+a3^2)- R^2) / (2*a2*sqrt((d4+d6)^2 +
a3^2));
        sin_phi = ARM*ELBOW*sqrt(1-power(cos_phi,2));
        sin_beta = (d4+d6)/sqrt((d4+d6)^2+a3^2);
        cos_beta = abs(a3)/sqrt((d4+d6)^2+a3^2);
        angle = atan2(sin_phi*cos_beta-cos_phi*sin_beta ,
cos_phi*cos_beta+sin_phi*sin_beta);
    end
    if(z<0)
```

```
        cos_phi = (a2^2+((d4+d6)^2+a3^2)- R^2) / (2*a2*sqrt((d4+d6)^2 +
a3^2));
        sin_phi = ARM*ELBOW*sqrt(1-power(cos_phi,2));
        sin_beta = (d4+d6)/sqrt((d4+d6)^2+a3^2);
        cos_beta = abs(a3)/sqrt((d4+d6)^2+a3^2);
        angle = atan2(sin_phi*cos_beta-cos_phi*sin_beta ,
cos_phi*cos_beta+sin_phi*sin_beta);
        angle = angle + (angle-(pi/2));
    end
end

function angle = theta2(x,y,z,robo_param,ARM,ELBOW)

    a1 = robo_param(1);
    a2 = robo_param(2);
    a3 = robo_param(3);
    d1 = robo_param(4);
    d2 = robo_param(5);
    d3 = robo_param(6);
    d4 = robo_param(7);
    d6 = robo_param(8);

    if(z<0)
        alpha = acos(-ARM*sqrt(power(x,2)+power(y,2)-d2^2) /
sqrt(power(x,2)+power(y,2)+power(z,2)-d2^2));
    end
    if(z>=0)
        alpha = acos(-ARM*sqrt(power(x,2)+power(y,2)-d2^2) /
sqrt(power(x,2)+power(y,2)+power(z,2)-d2^2));
        alpha = -(alpha);
    end
    beta = acos( (a2^2+x^2+y^2+z^2-d2^2-((d4+d6)^2+a3^2)) /
(2*a2*sqrt(power(x,2)+power(y,2)+power(z,2)-d2^2)));
    angle = atan2( sin(alpha+ARM*ELBOW*beta) , cos(alpha+ARM*ELBOW*beta)  );
end

function angle = theta1(x,y,z,robo_param,ARM,ELBOW)

    a1 = robo_param(1);
    a2 = robo_param(2);
    a3 = robo_param(3);
    d1 = robo_param(4);
    d2 = robo_param(5);
    d3 = robo_param(6);
    d4 = robo_param(7);
    d6 = robo_param(8);

    angle = atan2(-ARM*y*sqrt(power(x,2)+power(y,2)-d2^2)- x*d2 ,...
    -ARM*x*sqrt(power(x,2)+power(y,2)-d2^2)+y*d2);
end
```

File: End_point_graph.m:

```
% angles from the simulink
% ang1,ang2 and ang3 are robot joint ange from the simulink
% we are using the forward kinmatics equation to get the position of end
% effector for respective joint angel.
end_effector_pos = double(vpa(subs(forward_kinmatics,{thi1,thi2,thi3},{ang1'-
(pi/2),ang2',ang3'-(pi/2)}),3));
X = end_effector_pos(1,:); Y = end_effector_pos(2,:); Z =
end_effector_pos(3,:);
%ploting the X,Y,Z coordinates of the end effector.
plot3(X,Y,Z);
grid on
hold on
% marking the starting point.
plot3(X(1),Y(1),Z(1),'o','MarkerSize',10)
% marking the refrence frame origen
plot3(0,0,0,'*r','MarkerSize',15)
grid on
xlabel X
ylabel y
zlabel z
hold off
```