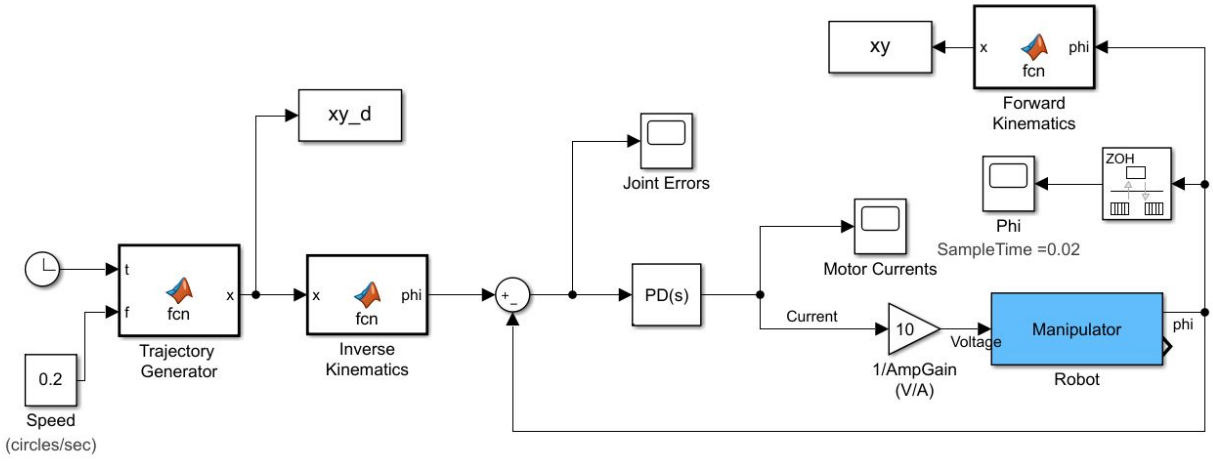
ME EN 6230
Problem Set 6
Ryan Dalby

1. Decentralized Control

Note: PD gains are Kp = 14 and Kd = 0.27 for all decentralized control. (These were found in PS5)

1.1. PD Control Only

1.1.1. Model

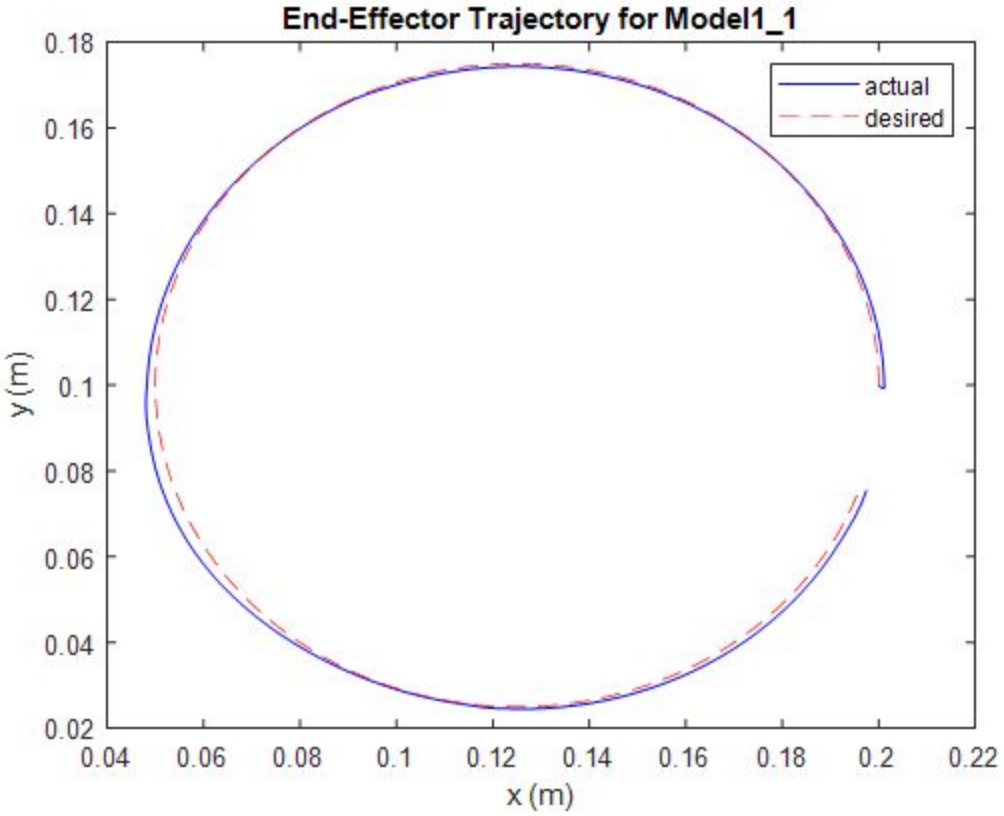


1.1.2. Code

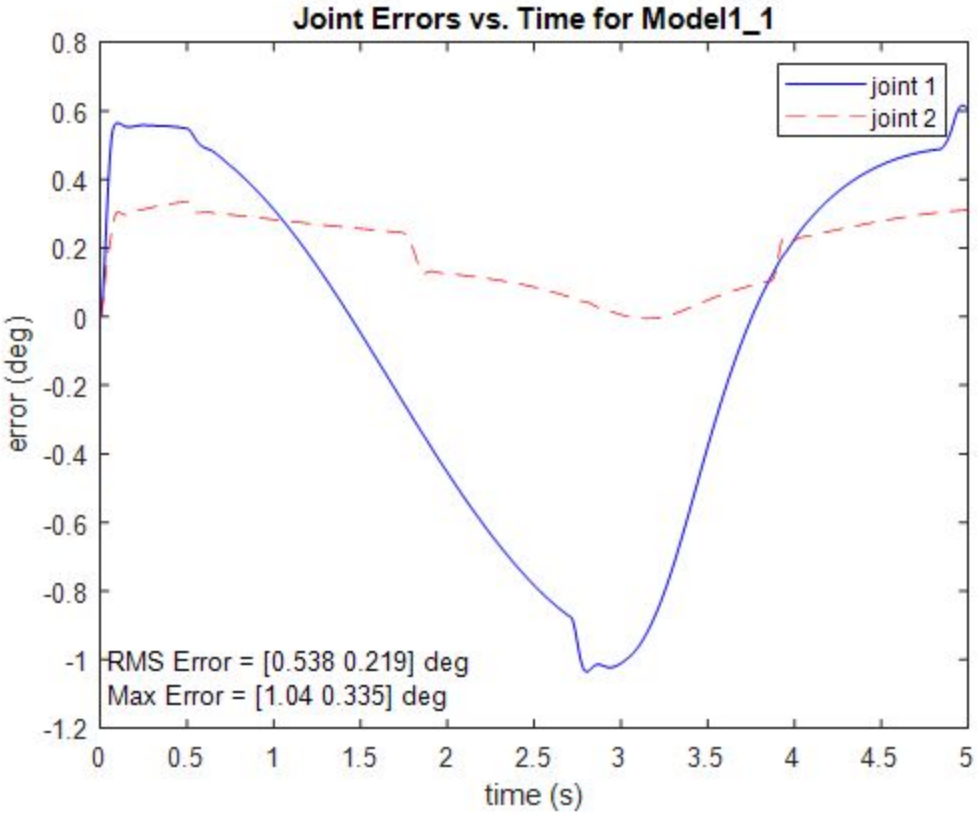
No code for model besides plotting code which is attached at the end of this document.

1.1.3. Low Speed Simulation (f=0.2 circles/s)

1.1.3.1. X-y trajectory

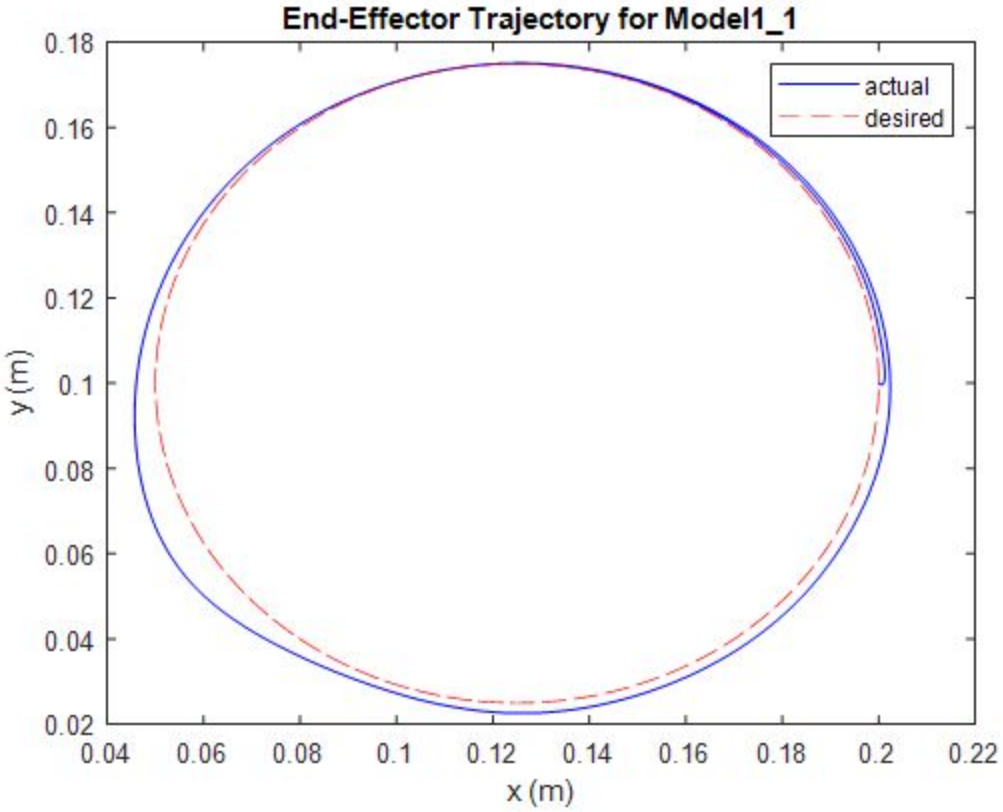


1.1.3.2. Joint angle errors

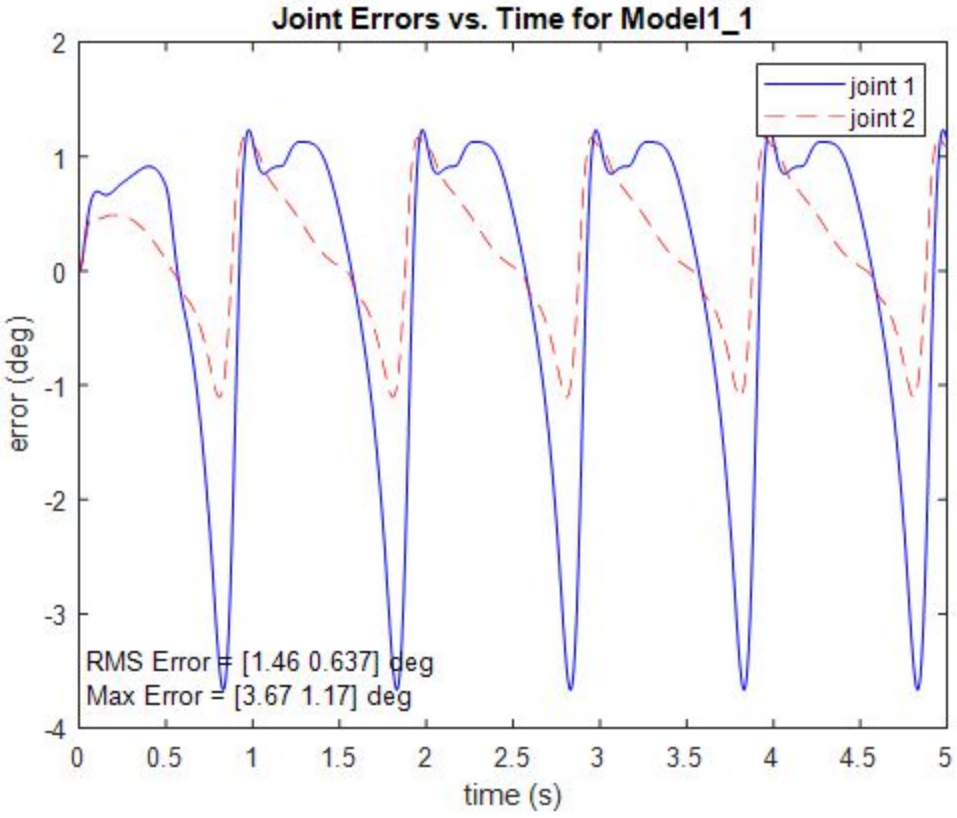


1.1.4. High Speed Simulation (f=1 circles/s)

1.1.4.1. X-y trajectory

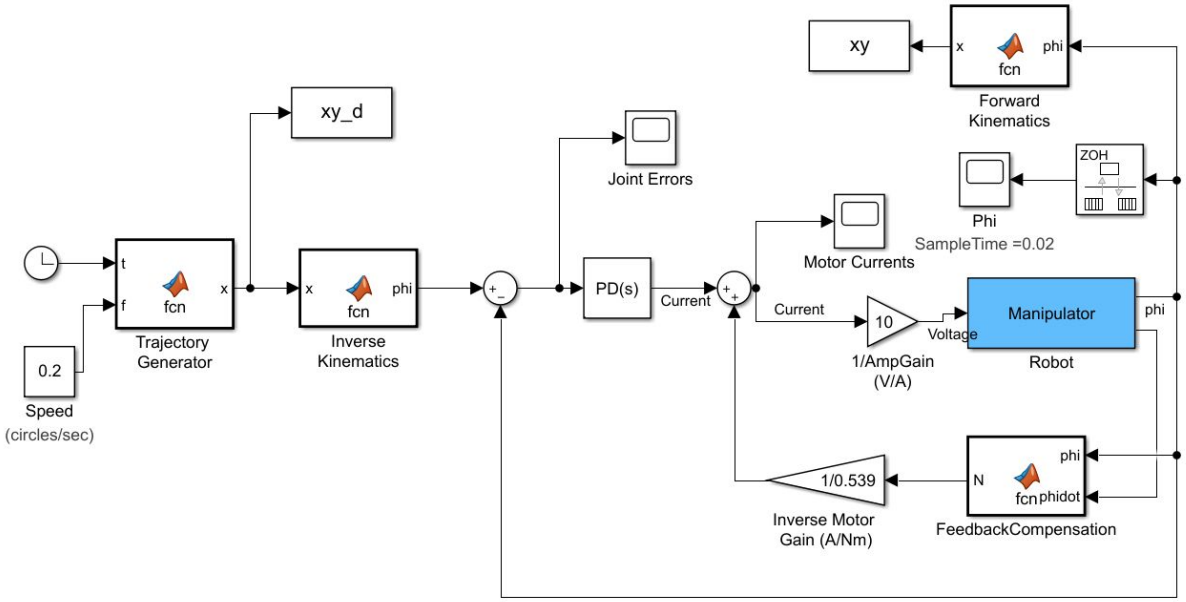


1.1.4.2. Joint angle errors



1.2. PD Control with gravity, friction, and centripetal/coriolis feedback compensation

1.2.1. Model



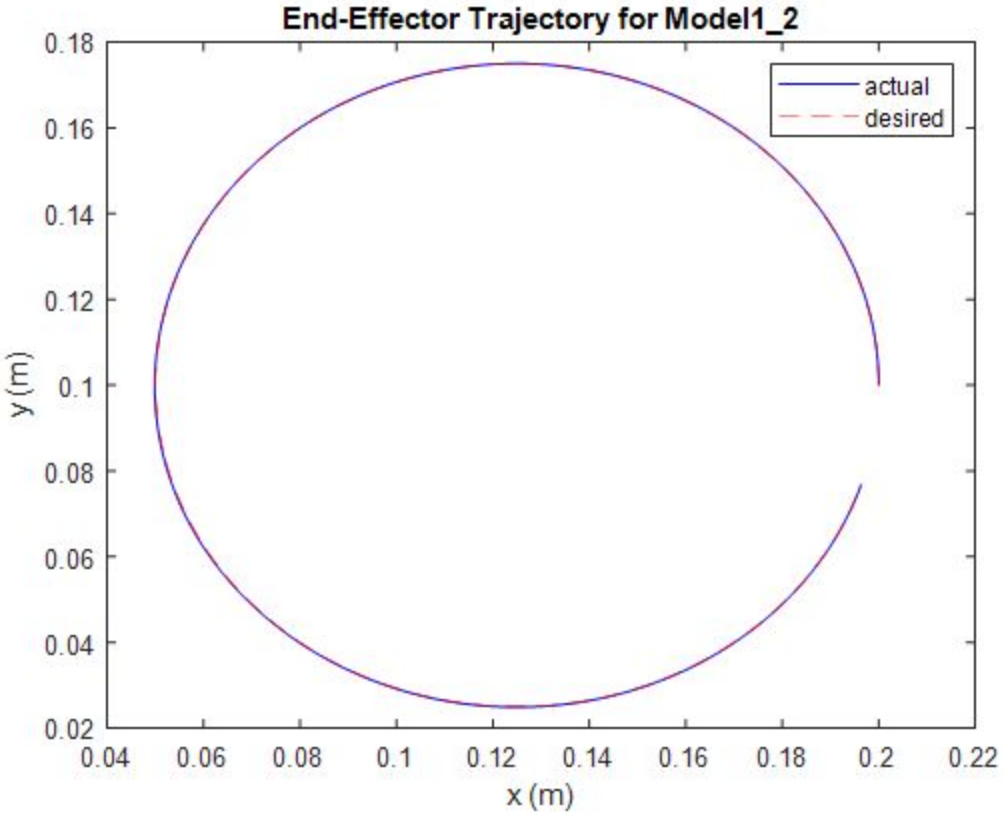
1.2.2. Code

Code for FeedbackCompensatoin Block:

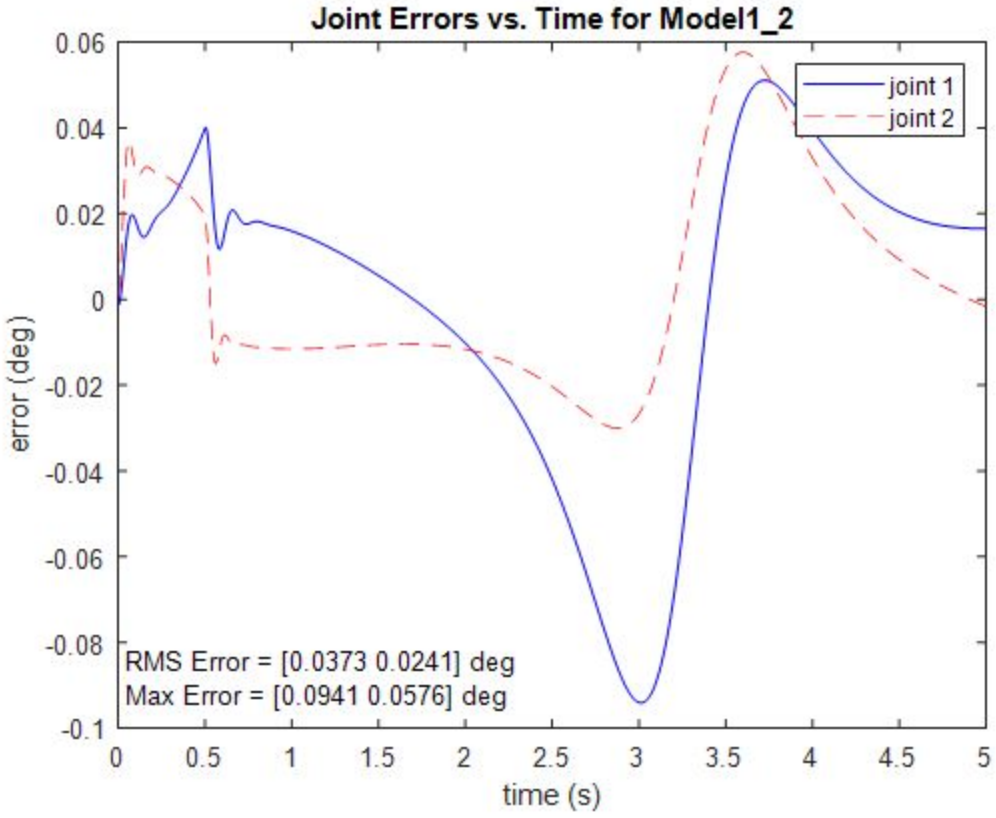
```
function N = fcn(phi, phidot)

% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length
m1 = 0.092;  % link 1 mass
m2 = 0.077; % link 2 mas
r01 = 0.062; % link 1 center of mass
r12 = 0.036; % link 2 COM
I1 = 0.64e-3;  % link 1 inertia
I2 = 0.30e-3;  % link 2 inertia
Jm1 = 0.65e-6; % motor inertias
Jm2 = 0.65e-6;
b1 = 3.1e-6; % viscous damping constants
b2 = 3.1e-6;
c1 = 0.0001; % coulomb friction constants
c2 = 0.0001;
g = 9.8; % gravitational constant
N1 = 70; % gear ratios
N2 = 70;

h = a1*r12*m2*sin(phi(2)-phi(1));
G1 = (r01*m1+a1*m2)*g*cos(phi(1));
G2 = r12*m2*g*cos(phi(2));
F1 = N1^2*b1*phidot(1) + N1*c1*sign(phidot(1));
F2 = N2^2*b2*phidot(2) + N2*c2*sign(phidot(2));

V = [0 -h ;h 0]*[phidot(1)^2;phidot(2)^2]; % centripetal torques
G = [G1;G2]; % gravity torques
F = [F1;F2]; % frictional torques

N = V + G + F;
```

1.2.3. Low Speed Simulation (f=0.2 circles/s)

1.2.3.1. X-y trajectory

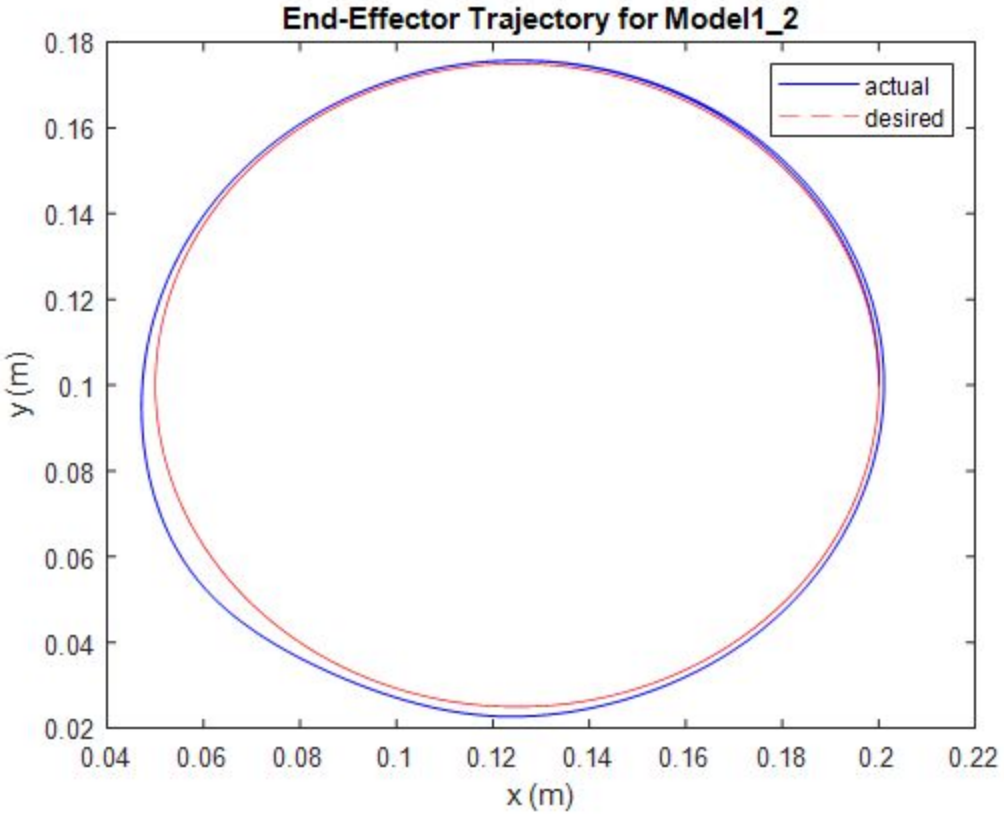


1.2.3.2. Joint angle errors

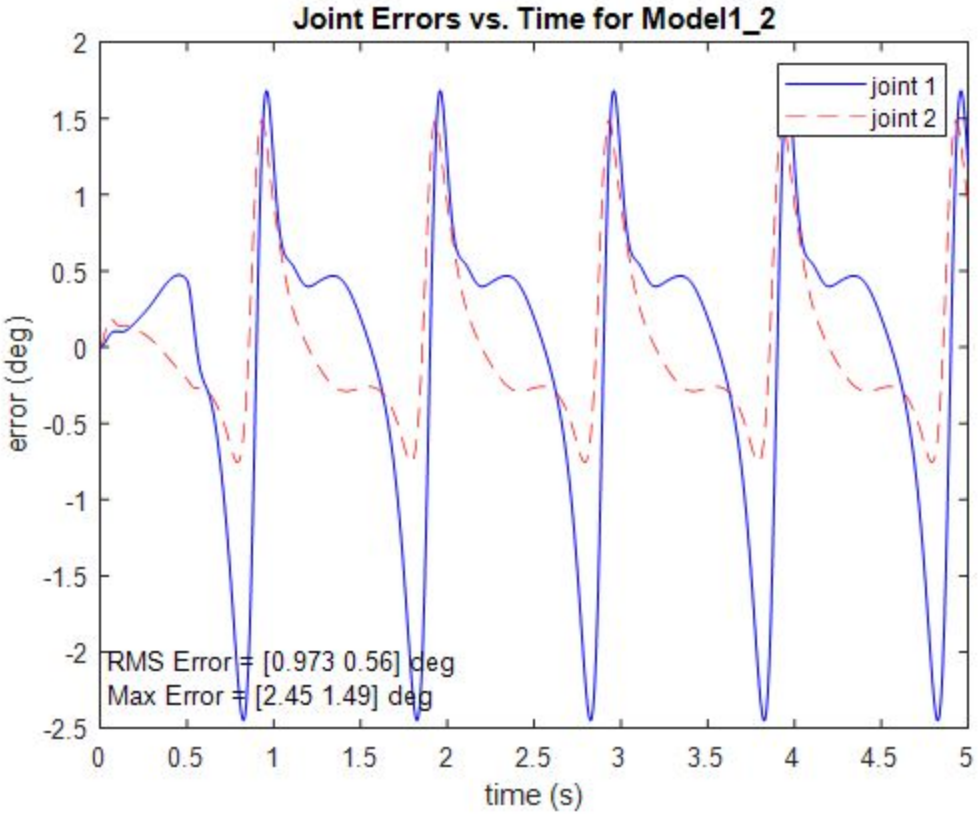


1.2.4. High Speed Simulation (f=1 circles/s)

1.2.4.1. X-y trajectory

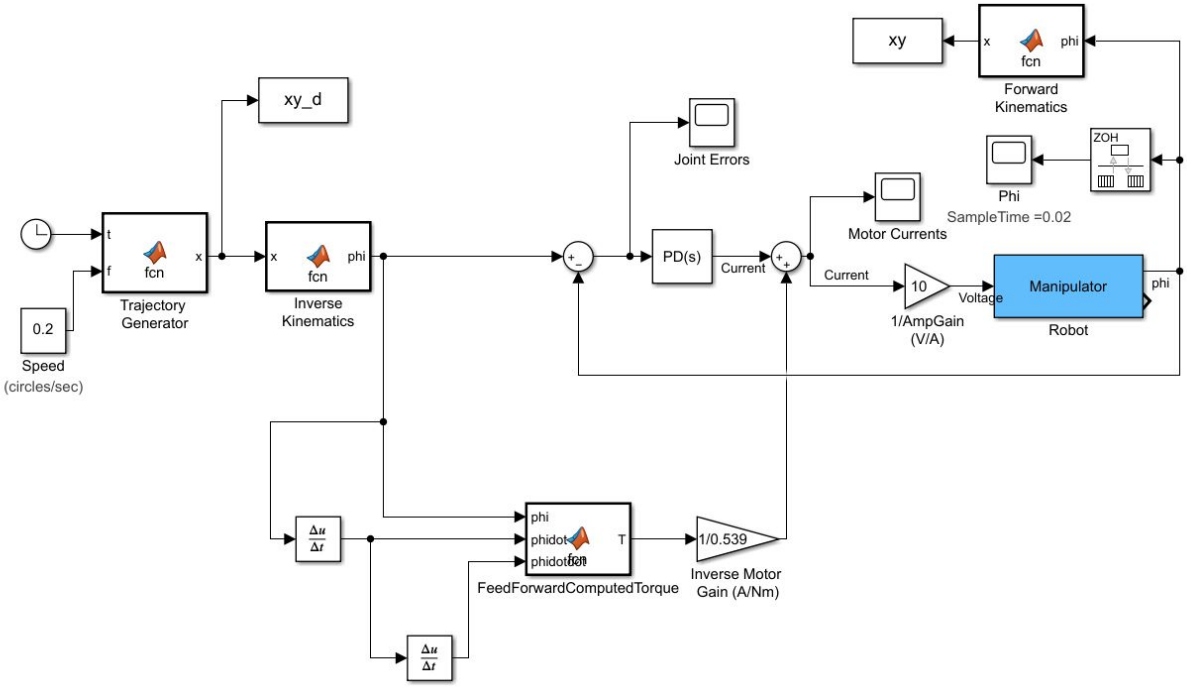


1.2.4.2. Joint angle errors



1.3. PD Control with Computed Torque Feedforward Control

1.3.1. Model



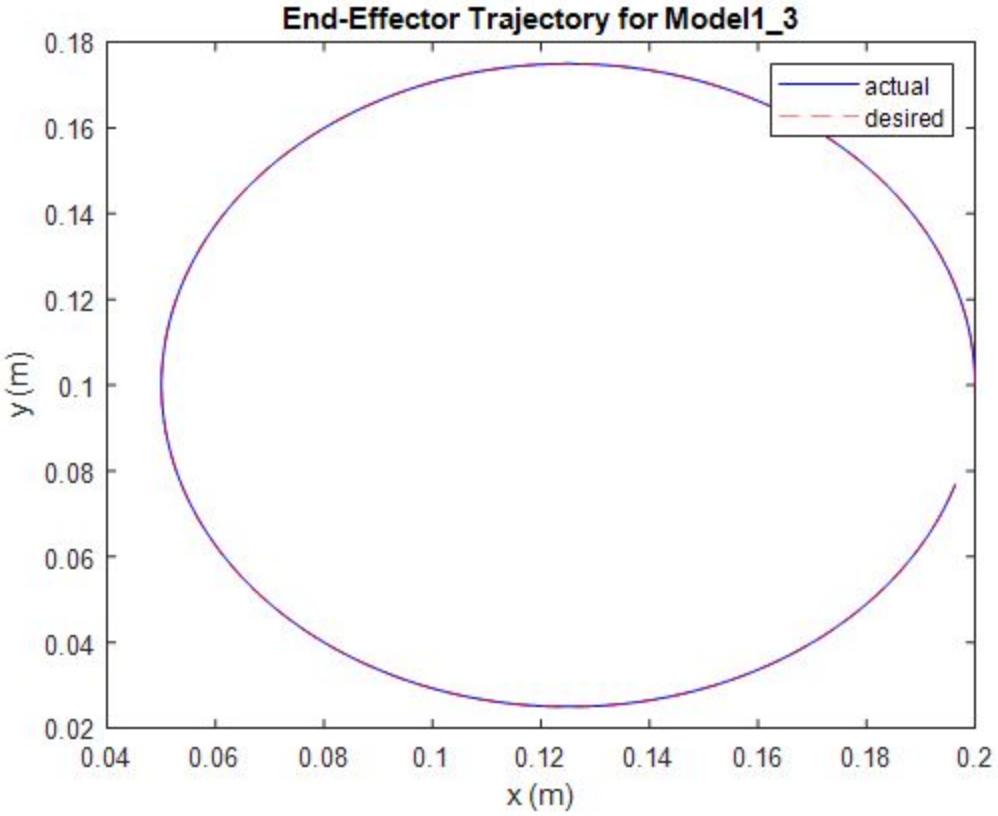
1.3.2. Code

Code for FeedForwardComputedTorque Block:

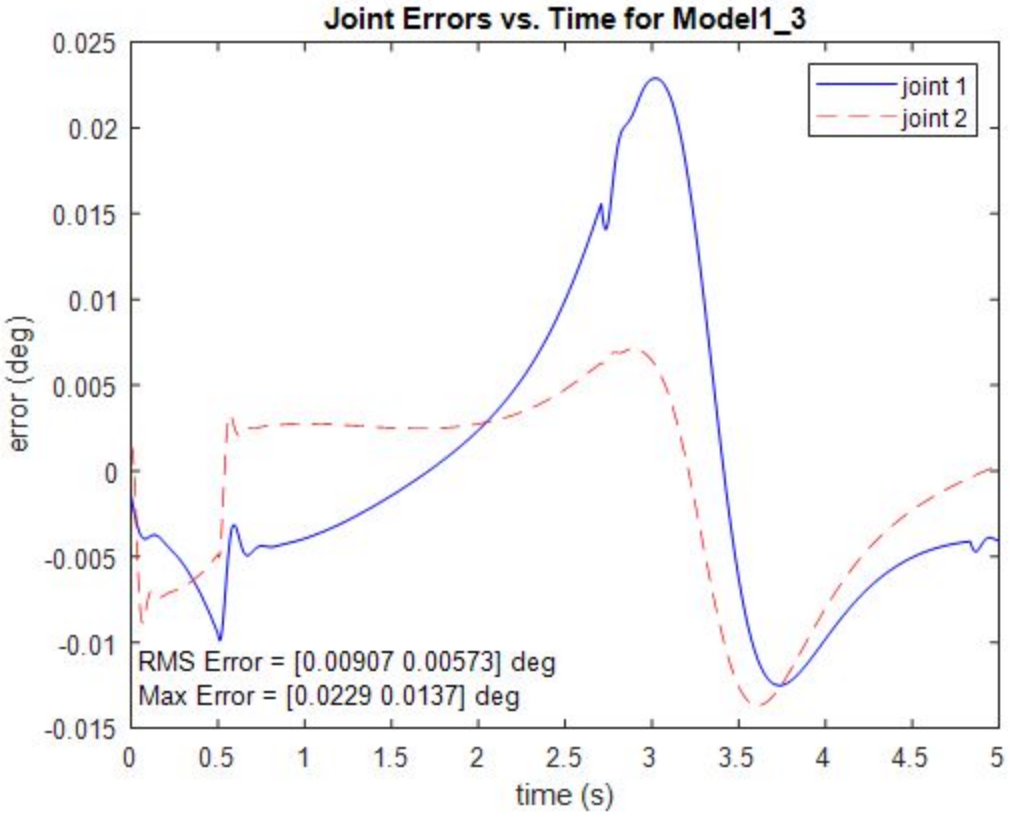
```
function T = fcn(phi,phidot,phidotdot)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length
m1 = 0.092;  % link 1 mass
m2 = 0.077; % link 2 mas
r01 = 0.062; % link 1 center of mass
r12 = 0.036; % link 2 COM
I1 = 0.64e-3;  % link 1 inertia
I2 = 0.30e-3;  % link 2 inertia
Jm1 = 0.65e-6; % motor inertias
Jm2 = 0.65e-6;
b1 = 3.1e-6; % viscous damping constants
b2 = 3.1e-6;
c1 = 0.0001; % coulomb friction constants
c2 = 0.0001;
g = 9.8; % gravitational constant
N1 = 70; % gear ratios
N2 = 70;

H11 = N1^2*Jm1 + I1 + m2*a1^2;
H12 = a1*r12*m2*cos(phi(2)-phi(1));
H21 = H12;
H22 = N2^2*Jm2 + I2;
h = a1*r12*m2*sin(phi(2)-phi(1));
G1 = (r01*m1+a1*m2)*g*cos(phi(1));
G2 = r12*m2*g*cos(phi(2));
F1 = N1^2*b1*phidot(1) + N1*c1*sign(phidot(1));
F2 = N2^2*b2*phidot(2) + N2*c2*sign(phidot(2));

H = [H11 H12; H21 H22]; % inertia matrix
V = [0 -h ;h 0]*[phidot(1)^2;phidot(2)^2]; % centripetal torques
G = [G1;G2]; % gravity torques
F = [F1;F2]; % frictional torques

T = H*phidotdot + V + G + F;
```

1.3.3. Low Speed Simulation (f=0.2 circles/s)

1.3.3.1. X-y trajectory

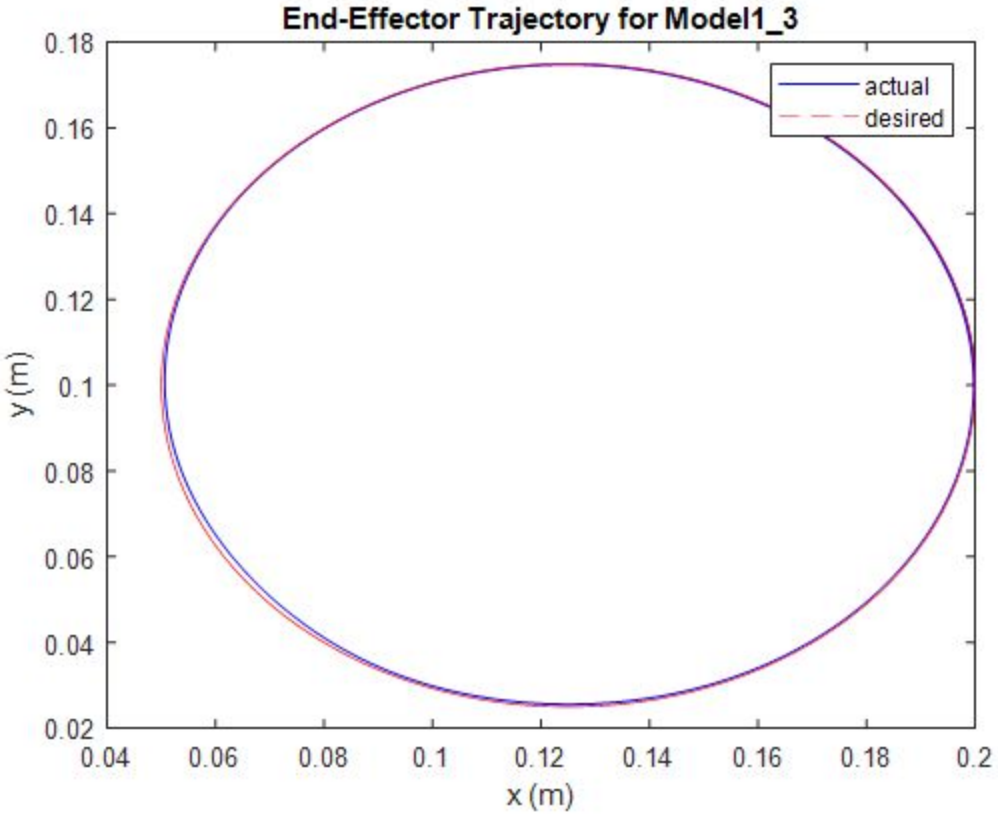


1.3.3.2. Joint angle errors

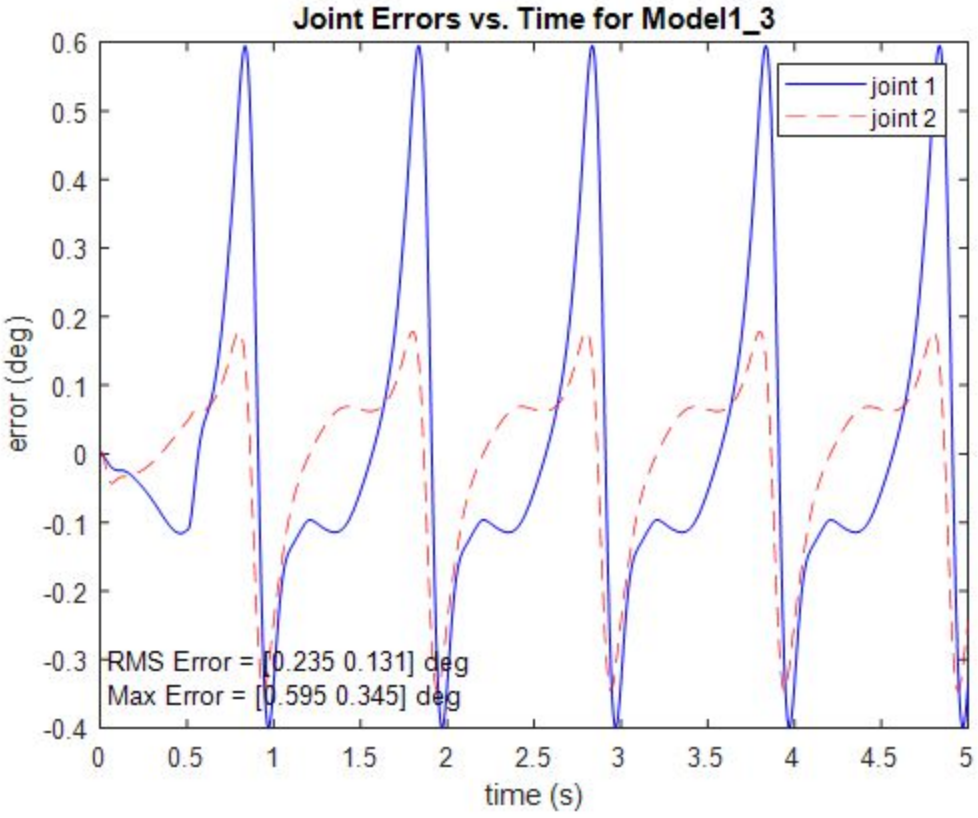


1.3.4. High Speed Simulation (f=1 circles/s)

1.3.4.1. X-y trajectory



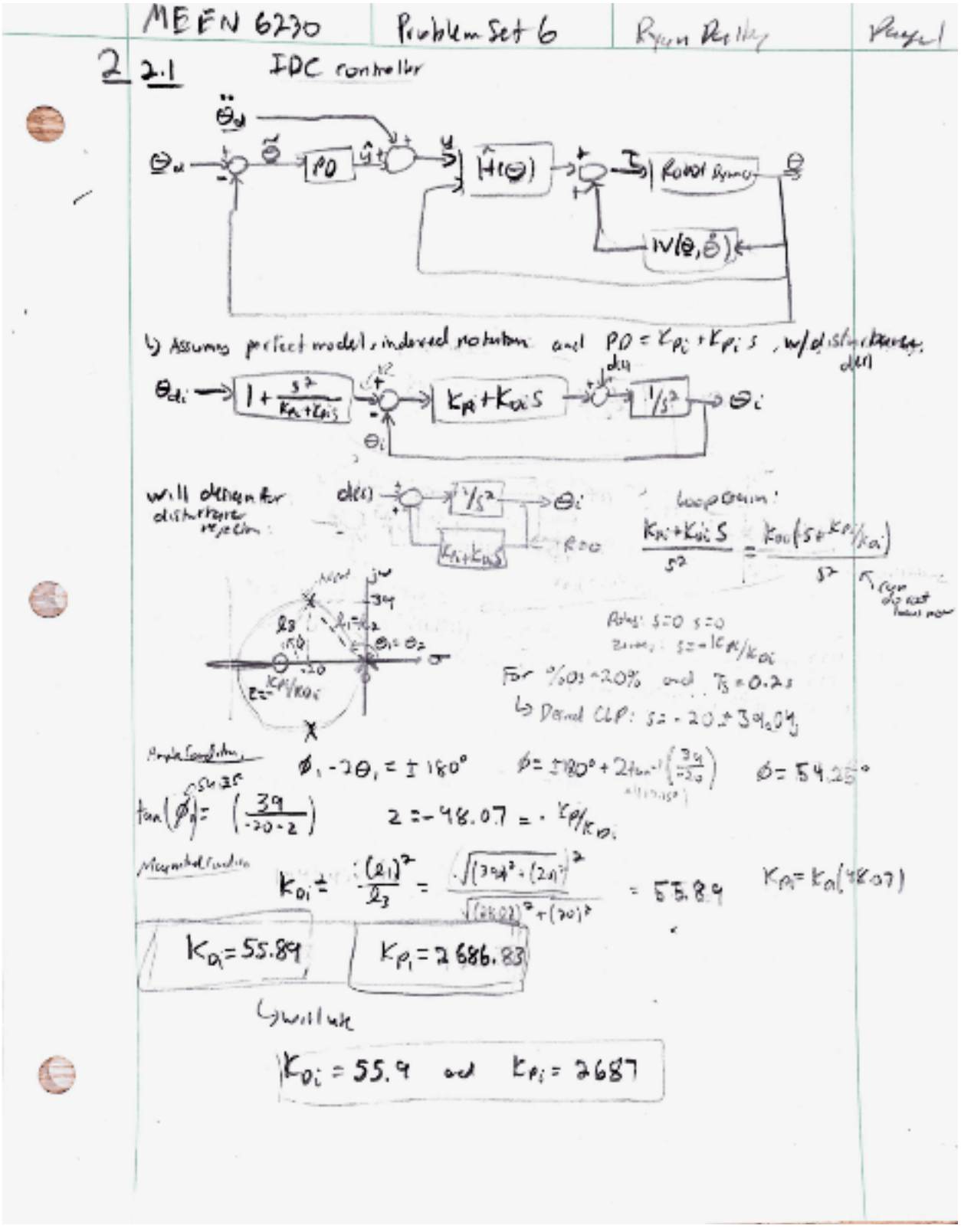
1.3.4.2. Joint angle errors



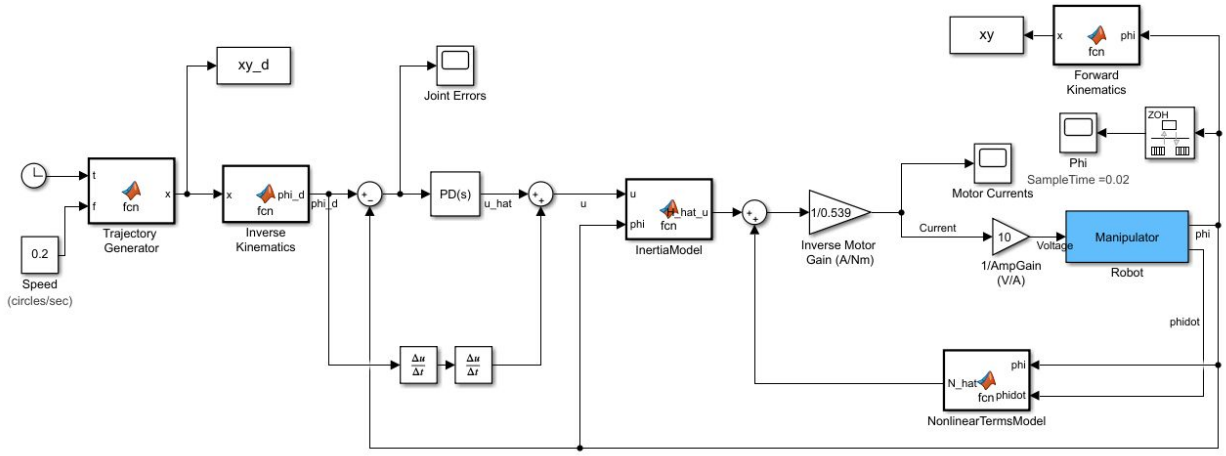
2. Centralized Control

2.1. Inverse Dynamics Controller

2.1.1. PD gain design



2.1.2. Model


xy_d
Joint Errors
xy
fcn
Forward Kinematics
ZOH
Phi
SampleTime =0.02
Motor Currents
0.2
Speed
(circles/sec)
Trajectory Generator
Inverse Kinematics
phi_d
PD(s)
u_hat
u
InertiaModel
1/0.539
Inverse Motor Gain (A/Nm)
Current
10
1/AmpGain (V/A)
Voltage
Manipulator
Robot
phi
phidot
N_hat
NonlinearTermsModel

2.1.3. Code

Code for InertiaModel Block:

```
function H_hat_u = fcn(u,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length
m1 = 0.092;  % link 1 mass
m2 = 0.077; % link 2 mas
r01 = 0.062; % link 1 center of mass
r12 = 0.036; % link 2 COM
I1 = 0.64e-3;  % link 1 inertia
I2 = 0.30e-3;  % link 2 inertia
Jm1 = 0.65e-6; % motor inertias
Jm2 = 0.65e-6;
b1 = 3.1e-6; % viscous damping constants
b2 = 3.1e-6;
c1 = 0.0001; % coulomb friction constants
c2 = 0.0001;
g = 9.8; % gravitational constant
N1 = 70; % gear ratios
N2 = 70;

H11 = N1^2*Jm1 + I1 + m2*a1^2;
H12 = a1*r12*m2*cos(phi(2)-phi(1));
H21 = H12;
H22 = N2^2*Jm2 + I2;

H_hat = [H11 H12; H21 H22]; % inertia matrix

H_hat_u = H_hat*u;
```

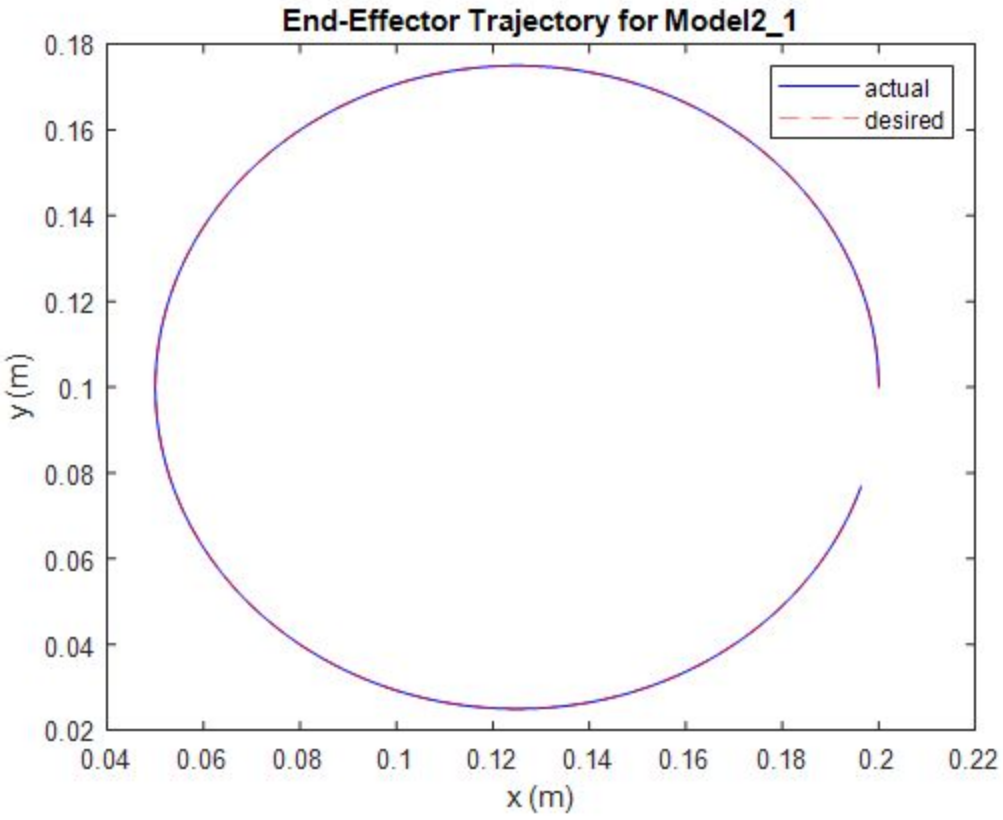
Code for NonlinearTermsModel Block:

```
function N_hat = fcn(phi,phidot)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length
m1 = 0.092;  % link 1 mass
m2 = 0.077; % link 2 mas
r01 = 0.062; % link 1 center of mass
r12 = 0.036; % link 2 COM
I1 = 0.64e-3;  % link 1 inertia
I2 = 0.30e-3;  % link 2 inertia
Jm1 = 0.65e-6; % motor inertias
Jm2 = 0.65e-6;
b1 = 3.1e-6; % viscous damping constants
b2 = 3.1e-6;
c1 = 0.0001; % coulomb friction constants
c2 = 0.0001;
g = 9.8; % gravitational constant
N1 = 70; % gear ratios
N2 = 70;

h = a1*r12*m2*sin(phi(2)-phi(1));
G1 = (r01*m1+a1*m2)*g*cos(phi(1));
G2 = r12*m2*g*cos(phi(2));
F1 = N1^2*b1*phidot(1) + N1*c1*sign(phidot(1));
F2 = N2^2*b2*phidot(2) + N2*c2*sign(phidot(2));

V_hat = [0 -h ;h 0]*[phidot(1)^2;phidot(2)^2]; % centripetal torques
G_hat = [G1;G2]; % gravity torques
F_hat = [F1;F2]; % frictional torques

N_hat = V_hat + G_hat + F_hat;
```

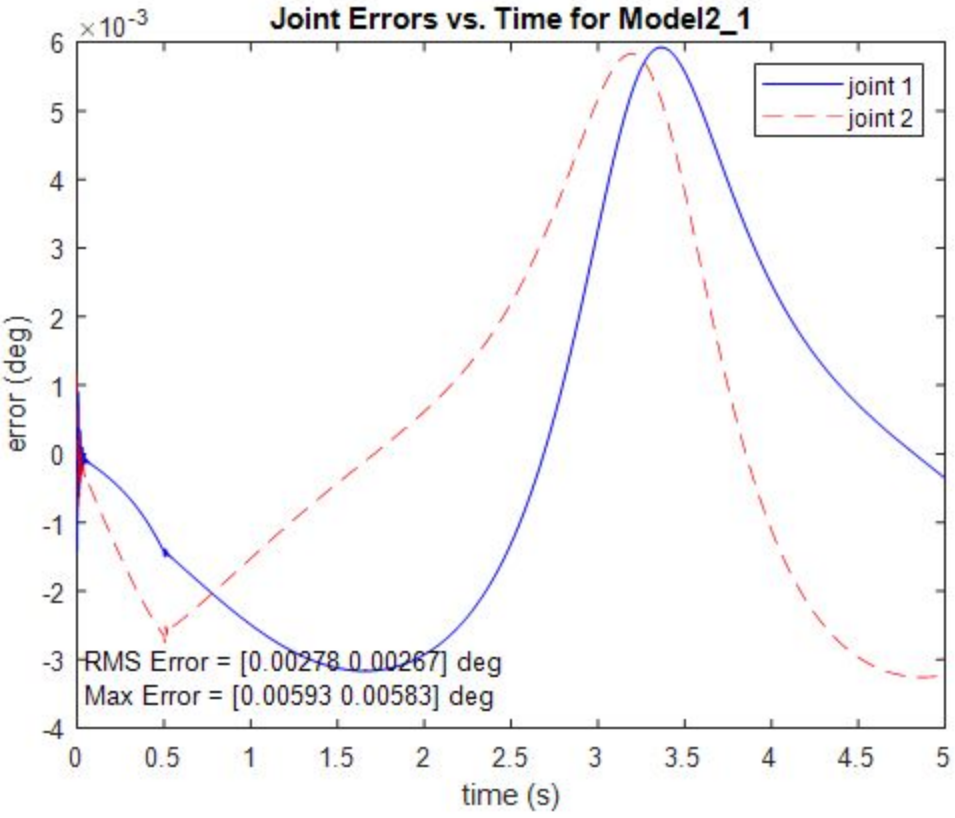
2.2. Inverse Dynamics Controller Simulation

2.2.1. Low Speed Simulation (f=0.2 circles/s)

2.2.1.1. X-y trajectory

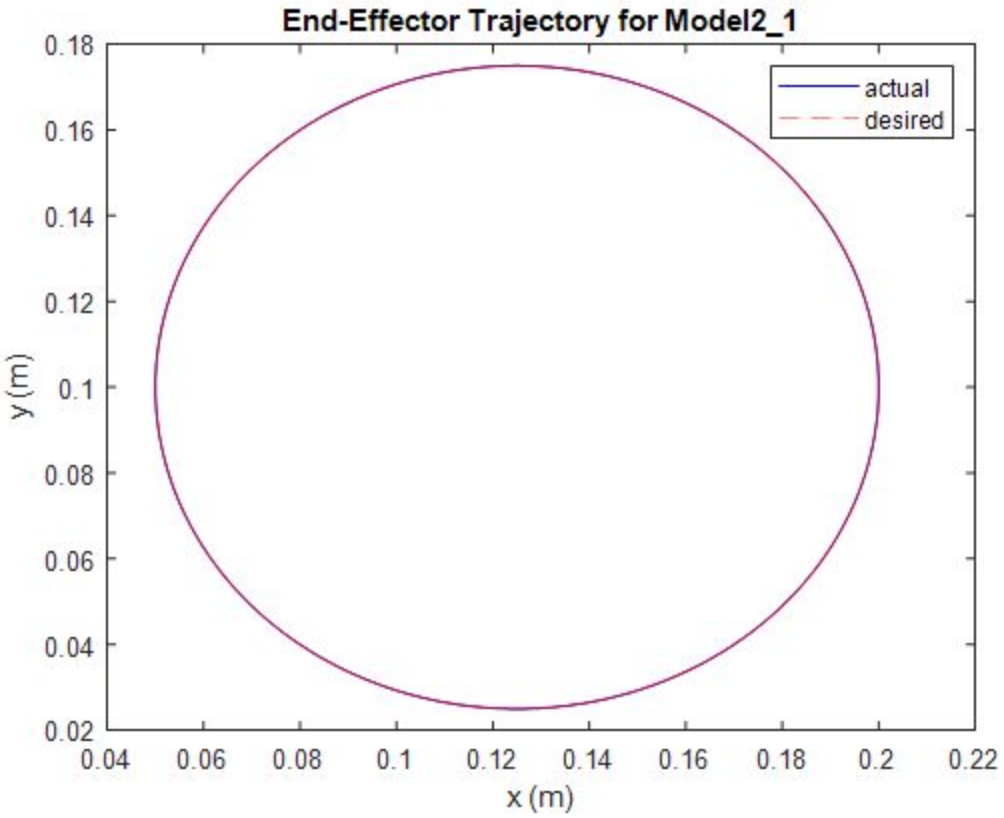


2.2.1.2. Joint angle errors

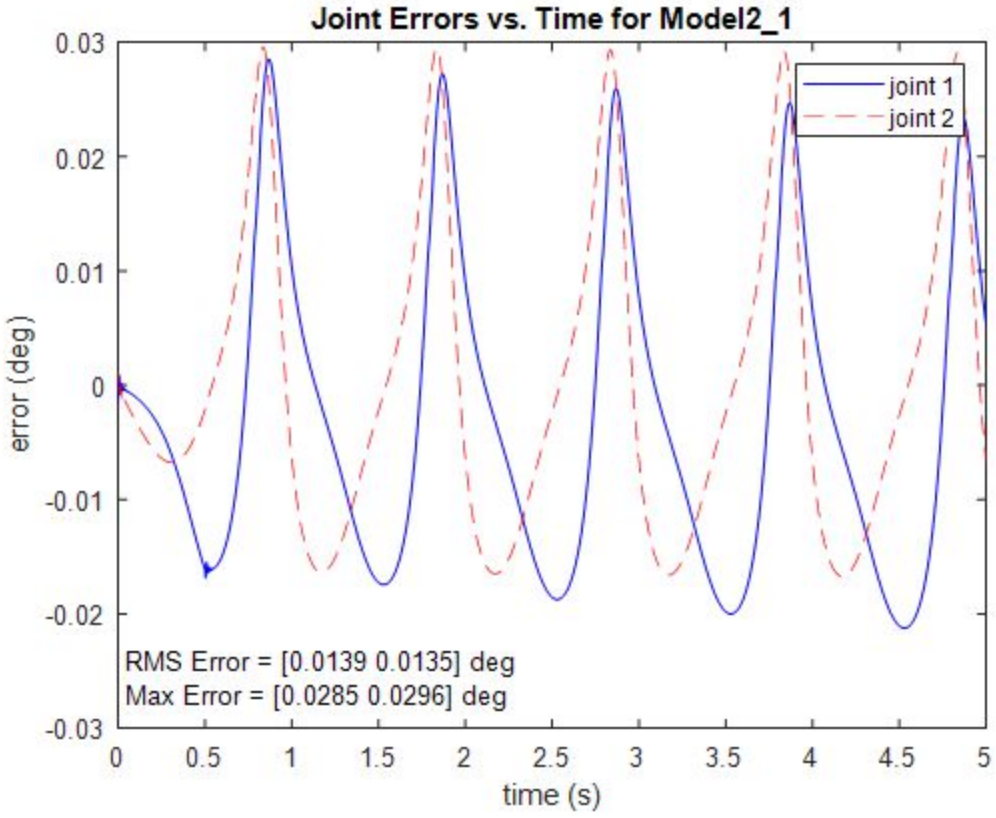


2.2.2. High Speed Simulation (f=1 circles/s)

2.2.2.1. X-y trajectory

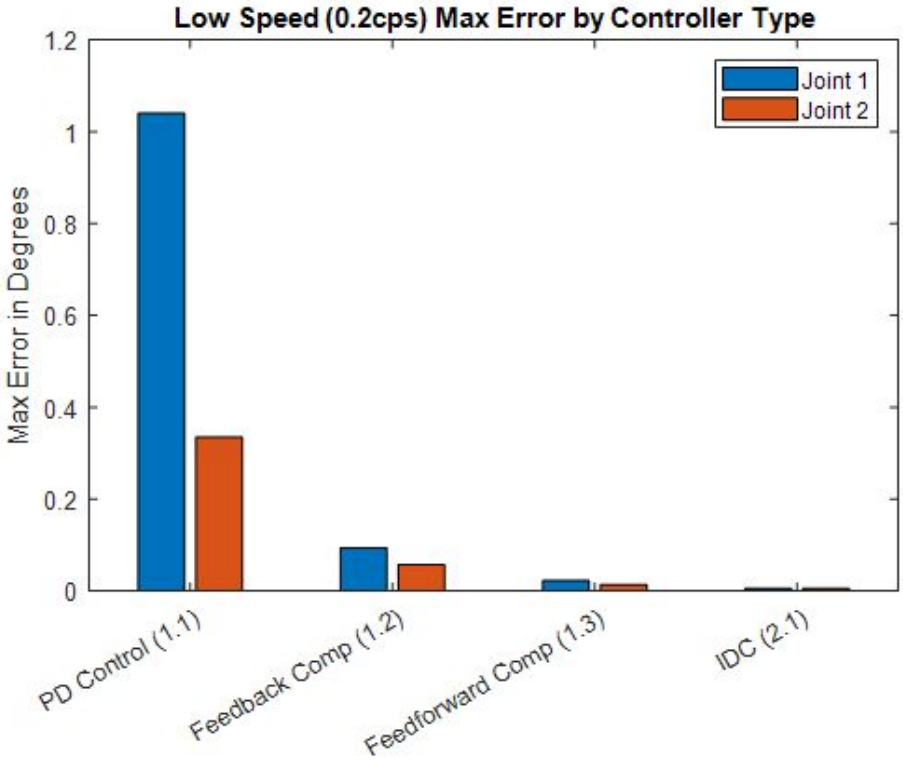


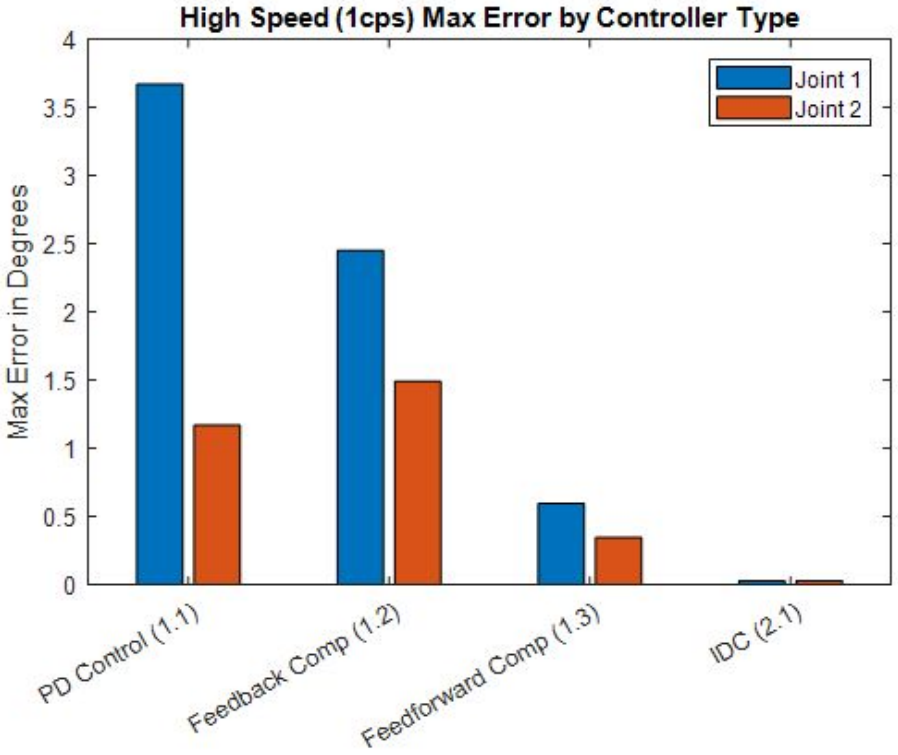
2.2.2.2. Joint angle errors



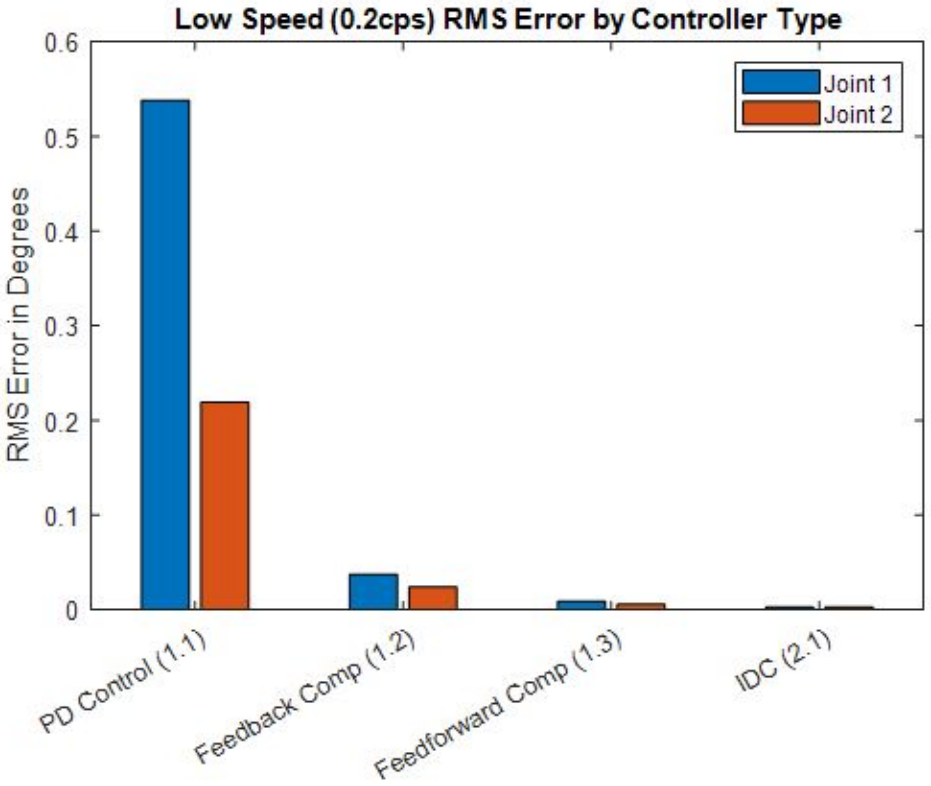
3. Controller Comparison

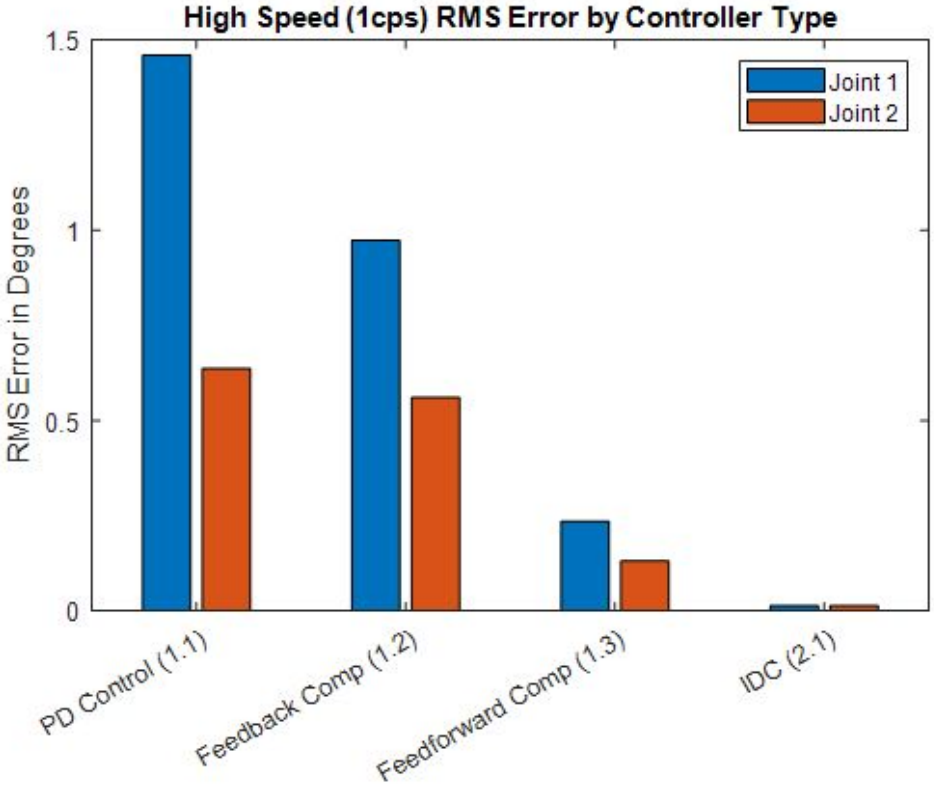
3.1. Peak joint errors





3.2. Root-Mean-Square joint errors





3.3. Comparison

As can be seen above the inverse dynamics controller (IDC) was the best performing controller in terms of the least RMS error and least max error for both joints at both high and low speeds. The next best controller was the feedforward compensator, then the feedback compensator, and lastly the PD controller did the worst with the highest RMS error and highest max error for both joints at both high and low speeds.

The first joint virtually always had more error, this is likely due to it having to support the second joint. All the controllers performed better with a lower speed trajectory, and with some quick tests it was noted that all the controllers quickly went unstable above 1.5cps. This was likely due to amp saturation resulting in the amp not able to provide the necessary current to maintain control.

3.4. Comparison Code

```
%% ME EN 6230 Problem Set 6 Ryan Dalby
% Controller Comparison
close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from
becoming subscript

% errors are in degrees
% Model 1.1- PD Controller
m11_slow_rms = [0.538 0.219];
m11_slow_max = [1.04 0.335];
m11_fast_rms = [1.46 0.637];
m11_fast_max = [3.67 1.17];

% Model 1.2- PD feedback comp
m12_slow_rms = [0.0373 0.0241];
m12_slow_max = [0.0941 0.0576];
m12_fast_rms = [0.973 0.56];
m12_fast_max = [2.45 1.49];

% Model 1.3- PD feedforward comp
m13_slow_rms = [0.00907 0.00573];
m13_slow_max = [0.0229 0.0137];
m13_fast_rms = [0.235 0.131];
m13_fast_max = [0.595 0.345];

% Model 2.1- IDC
m21_slow_rms = [0.00278 0.00267];
m21_slow_max = [0.00593 0.00583];
m21_fast_rms = [0.0139 0.0135];
m21_fast_max = [0.0285 0.0296];

slow_rms = [m11_slow_rms; m12_slow_rms; m13_slow_rms; m21_slow_rms];
fast_rms = [m11_fast_rms; m12_fast_rms; m13_fast_rms; m21_fast_rms];
slow_max = [m11_slow_max; m12_slow_max; m13_slow_max; m21_slow_max];
fast_max = [m11_fast_max; m12_fast_max; m13_fast_max; m21_fast_max];

controller_labels = {'PD Control (1.1)', 'Feedback Comp (1.2)',
'Feedforward Comp (1.3)', 'IDC (2.1)'};
controller_labels_cat = categorical(controller_labels);
controller_labels_cat = reordercats(controller_labels_cat,
string(controller_labels_cat));
```

```
figure;
bar(controller_labels_cat,slow_rms);
title('Low Speed (0.2cps) RMS Error by Controller Type');
legend('Joint 1', 'Joint 2');
ylabel('RMS Error in Degrees');

figure;
bar(controller_labels_cat,fast_rms);
title('High Speed (1cps) RMS Error by Controller Type');
legend('Joint 1', 'Joint 2');
ylabel('RMS Error in Degrees');

figure;
bar(controller_labels_cat,slow_max);
title('Low Speed (0.2cps) Max Error by Controller Type');
legend('Joint 1', 'Joint 2');
ylabel('Max Error in Degrees');

figure;
bar(controller_labels_cat,fast_max);
title('High Speed (1cps) Max Error by Controller Type');
legend('Joint 1', 'Joint 2');
ylabel('Max Error in Degrees');
```

Appendix- Plotting Code (Used to make simulation plots)

```
%% ME EN 6230 Problem Set 6 Ryan Dalby
% close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from
becoming subscript

% Extract necessary data, will error if the data does not exist
time = errors.time; % s
model_title = extractBefore(errors.blockName, "/Joint Errors");
joint_errors = rad2deg(errors.signals.values); % deg
actual_trajectory = xy; % m
desired_trajectory = xy_d; % m

% RMS Joint Errors
RMS_error = rms(joint_errors);
% Max Joint Errors
max_error = max(abs(joint_errors));

% Plot Joint Errors vs Time
figure;
plot(time, joint_errors(:,1), 'b-');
hold on;
plot(time, joint_errors(:,2), 'r--');
hold on;
text(0.01,0.10,append('RMS Error = ', mat2str(RMS_error,3),' deg'),
'Units', 'normalized');
hold on;
text(0.01,0.05,append('Max Error = ', mat2str(max_error,3),' deg'),
'Units', 'normalized');
title(append('Joint Errors vs. Time for ', model_title));
xlabel('time (s)');
ylabel('error (deg)');
legend('joint 1', 'joint 2');

% Plot End-Effector Trajectory
figure;
plot(xy(:,1), xy(:,2), 'b-');
hold on;
plot(xy_d(:,1), xy_d(:,2), 'r--');
title(append('End-Effector Trajectory for ', model_title));
xlabel('x (m)');
ylabel('y (m)');
legend('actual', 'desired');
```