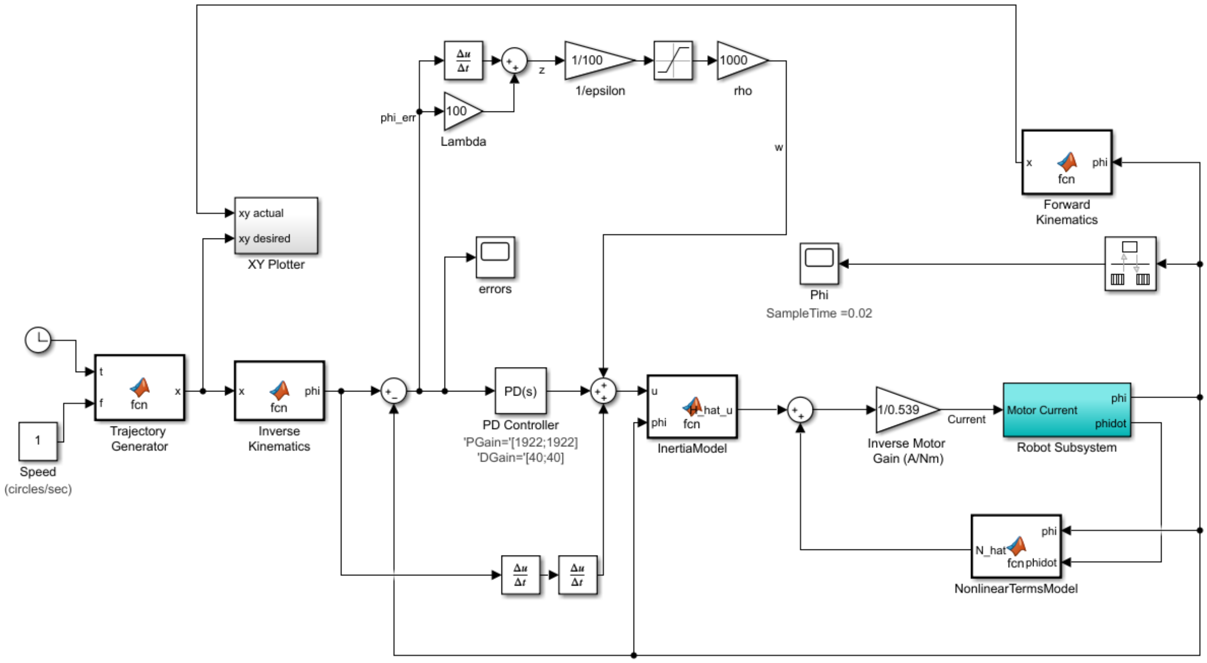
ME EN 6230
Lab 2
Ryan Dalby

Note that lab station 2 was used for this lab. The amp bias used was [-0.042; 0.065] and the current sense bias used was [0.02; 0.037].

1. Sliding Mode Control

Note: The sliding mode control parameters were modified from simulation to function well on the real-world robot. Epsilon was set to 100 to increase the boundary layer thickness and prevent chattering and saturation of the amplifier. Gamma was also increased to 1000 to compensate for the change in epsilon and to get closer to using the full range of the amplifier.

1.1. Model



1.2. Code

Code for InertiaModel Block:

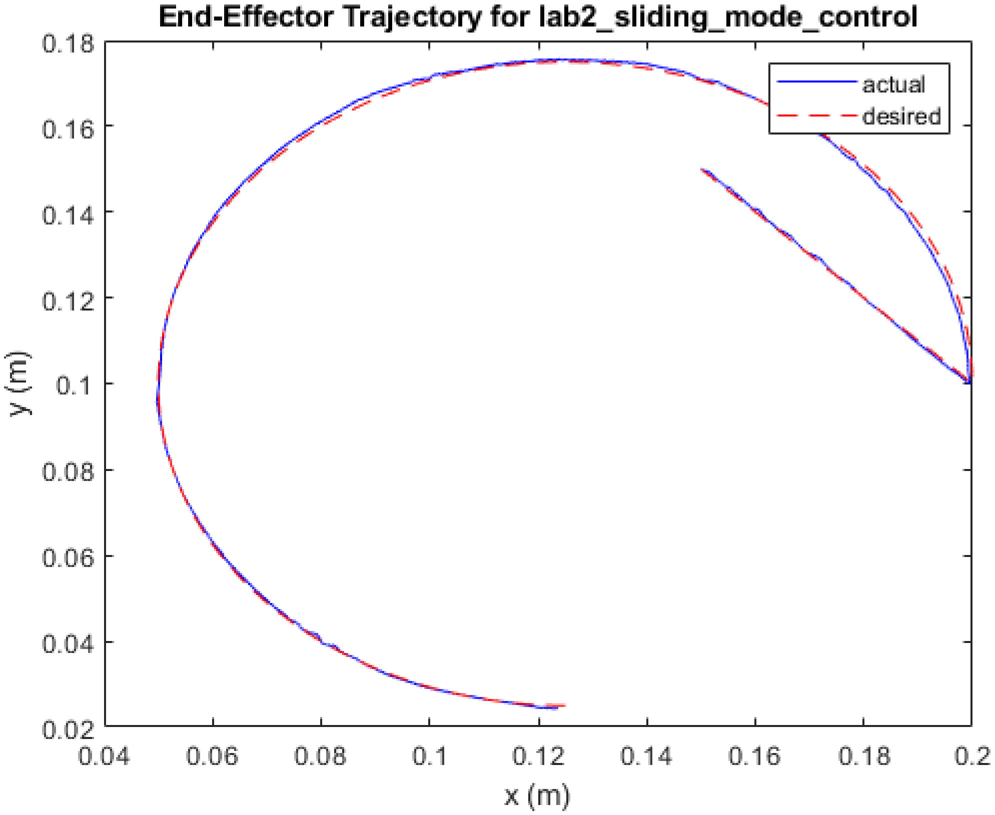
```
function H_hat_u = fcn(u,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

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:

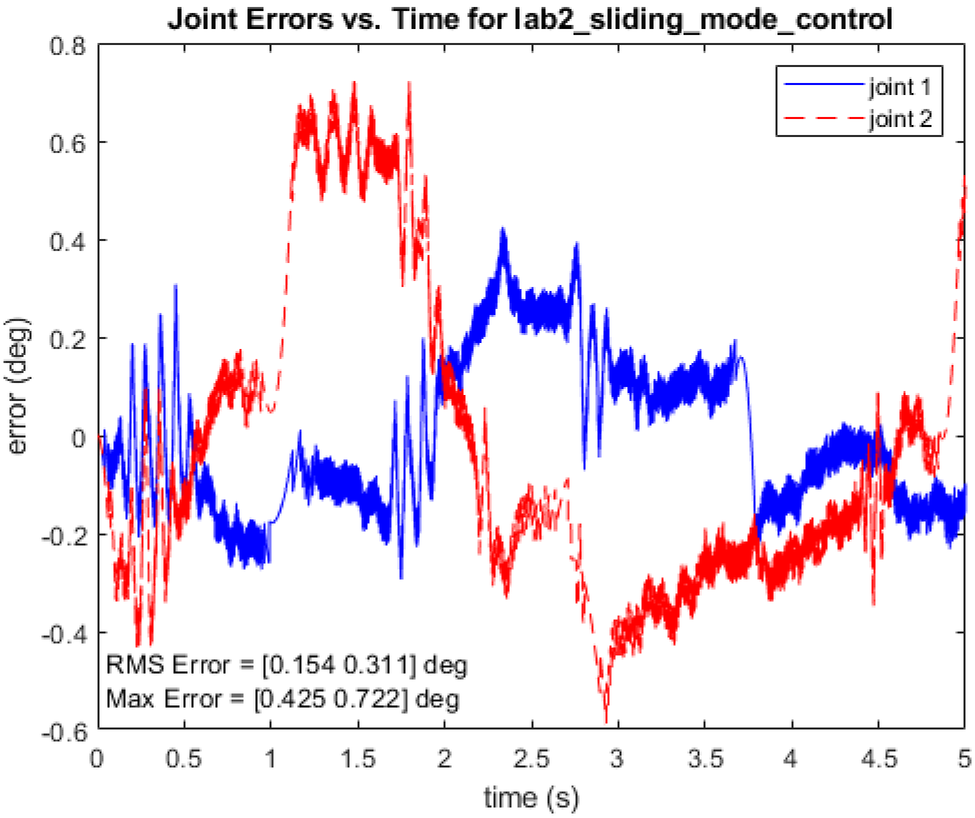
```
function N_hat = fcn(phi,phidot)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

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;
```

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

1.3.1. X-y trajectory

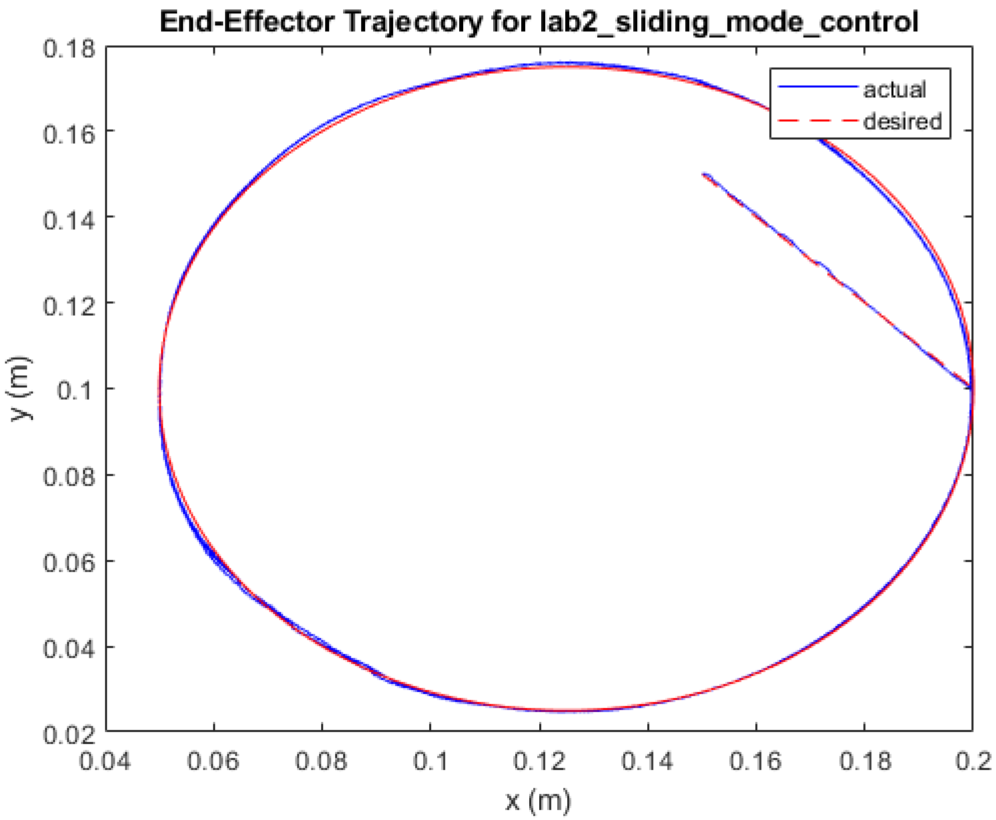


1.3.2. Joint angle errors

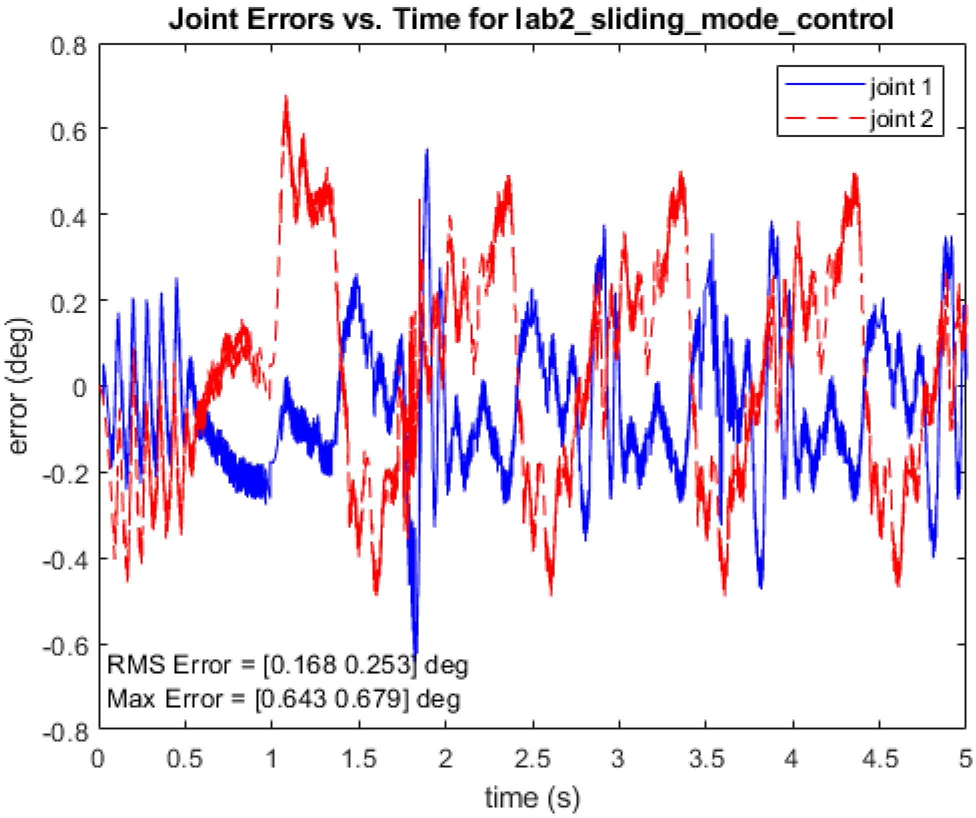


1.4. High Speed Simulation (f=1 circles/s)

1.4.1. X-y trajectory

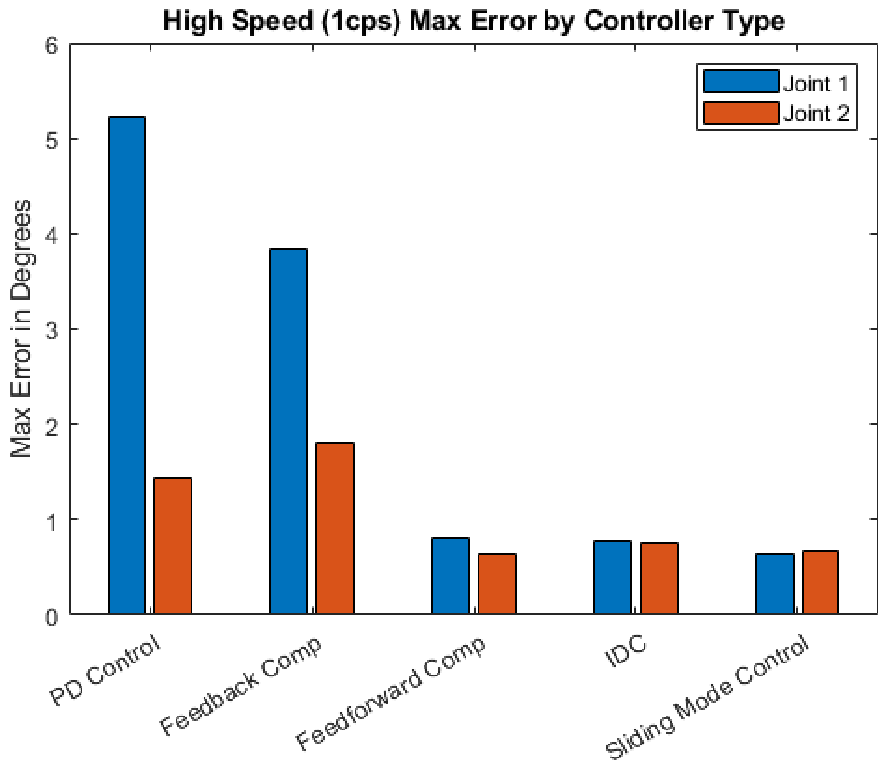


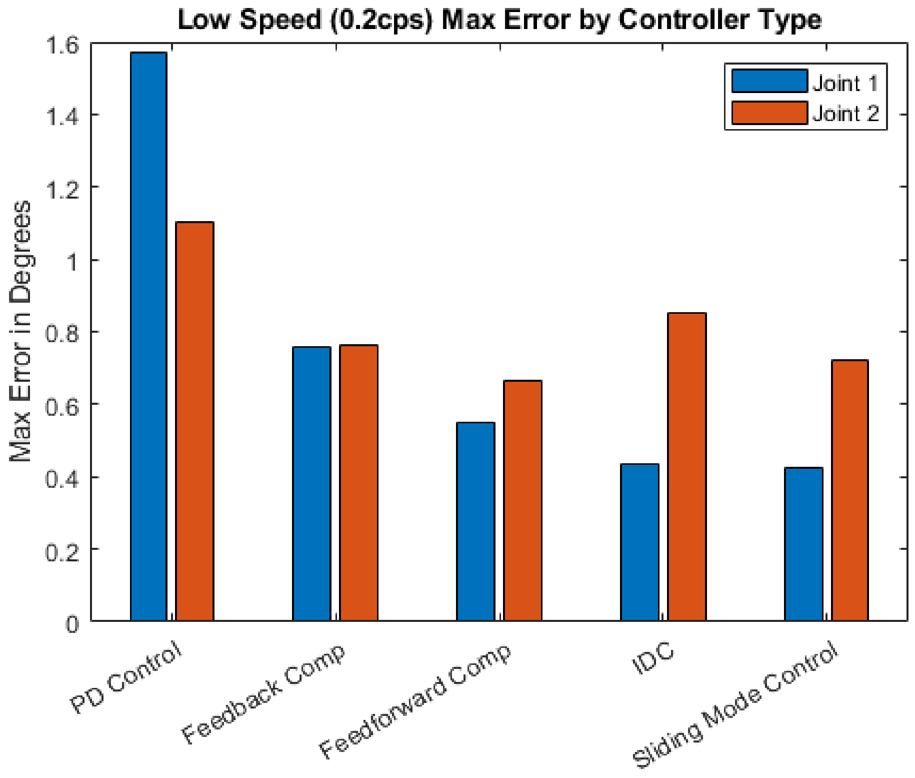
1.4.2. Joint angle errors



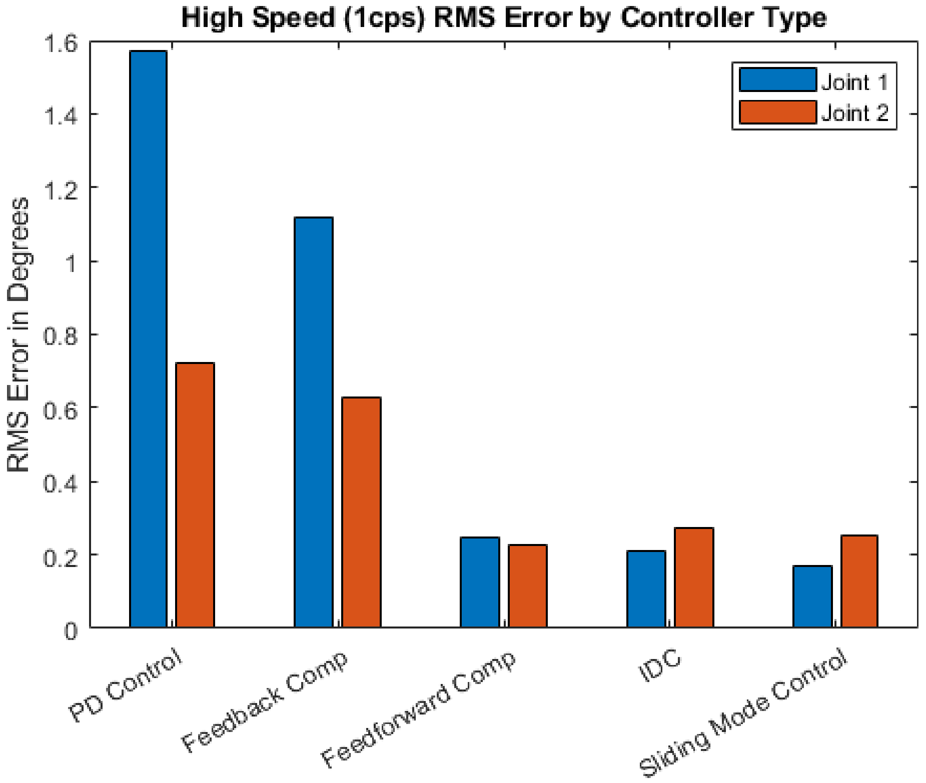
1.5. Controller comparison

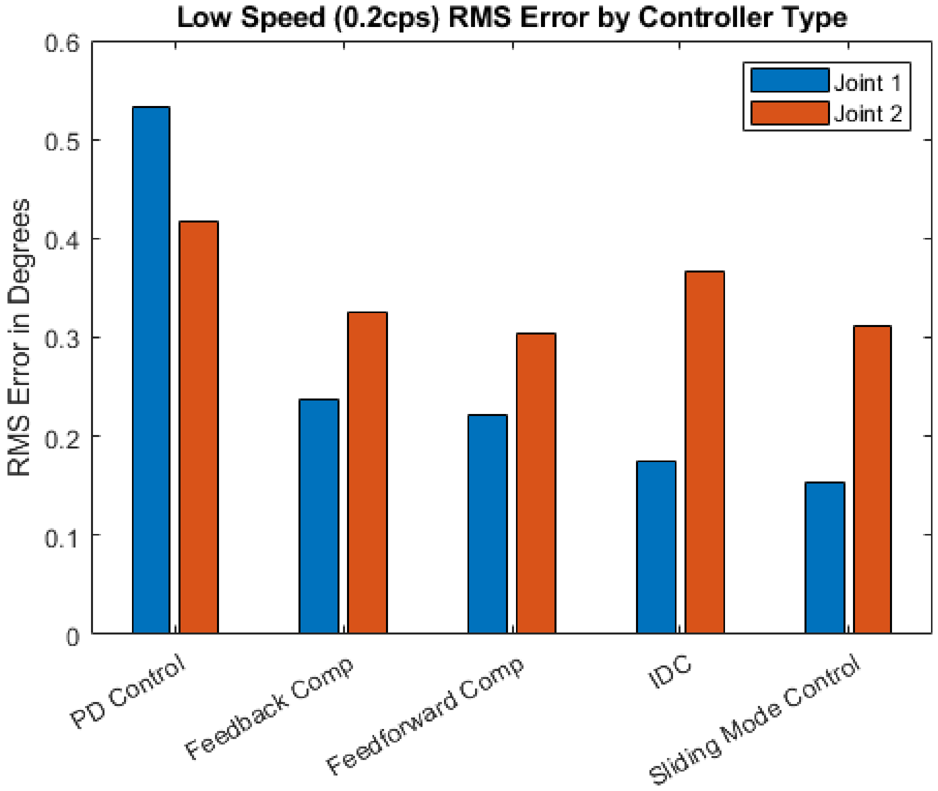
1.5.1. Peak joint errors





1.5.2. Root-Mean-Square joint errors





1.5.3. Comparison

As can be seen above, for low speed trajectories feedback comp, feedforward comp, inverse dynamics control, and sliding mode control are all comparable in terms of RMS and peak error on both joints. For low speeds PD control has the highest RMS and peak error on both joints. For high speed trajectories feedforward comp, inverse dynamics control, and sliding mode control are all comparable with feedback comp and PD control behind these controllers with the highest RMS and peak error on both joints. Overall, for high speed trajectories it does seem that sliding mode control has a slight edge over feedforward comp and inverse dynamics control with the lowest joint 1 RMS and peak error and comparable joint 2 RMS and peak error.

1.5.4. Comparison Code

```matlab
%% ME EN 6230 Lab 2 Ryan Dalby
% Controller Comparison
% close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from becoming subscript

% errors are in degrees
% PD Controller
pd_slow_rms = [0.532 0.417];
pd_slow_max = [1.57 1.1];
pd_fast_rms = [1.57 0.722];
pd_fast_max = [5.23 1.44];

% PD feedback comp
feedbackcomp_slow_rms = [0.237 0.326];
feedbackcomp_slow_max = [0.756 0.766];
feedbackcomp_fast_rms = [1.12 0.627];
feedbackcomp_fast_max = [3.84 1.8];

% PD feedforward comp
feedforwardcomp_slow_rms = [0.222 0.304];
feedforwardcomp_slow_max = [0.548 0.665];
feedforwardcomp_fast_rms = [0.246 0.228];
feedforwardcomp_fast_max = [0.805 0.638];

% IDC
idc_slow_rms = [0.174 0.366];
idc_slow_max = [0.435 0.853];
idc_fast_rms = [0.209 0.276];
idc_fast_max = [0.768 0.760];

% Sliding mode control
slidingmodecontrol_slow_rms = [0.154 0.311];
slidingmodecontrol_slow_max = [0.425 0.722];
slidingmodecontrol_fast_rms = [0.168 0.253];
slidingmodecontrol_fast_max = [0.643 0.679];

slow_rms = [pd_slow_rms; feedbackcomp_slow_rms; feedforwardcomp_slow_rms; idc_slow_rms; slidingmodecontrol_slow_rms];
fast_rms = [pd_fast_rms; feedbackcomp_fast_rms; feedforwardcomp_fast_rms; idc_fast_rms; slidingmodecontrol_fast_rms];
slow_max = [pd_slow_max; feedbackcomp_slow_max; feedforwardcomp_slow_max;
idc_slow_max; slidingmodecontrol_slow_max];
fast_max = [pd_fast_max; feedbackcomp_fast_max; feedforwardcomp_fast_max;
idc_fast_max; slidingmodecontrol_fast_max];

controller_labels = {'PD Control', 'Feedback Comp', 'Feedforward Comp',
'IDC', 'Sliding Mode Control'};
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');
```

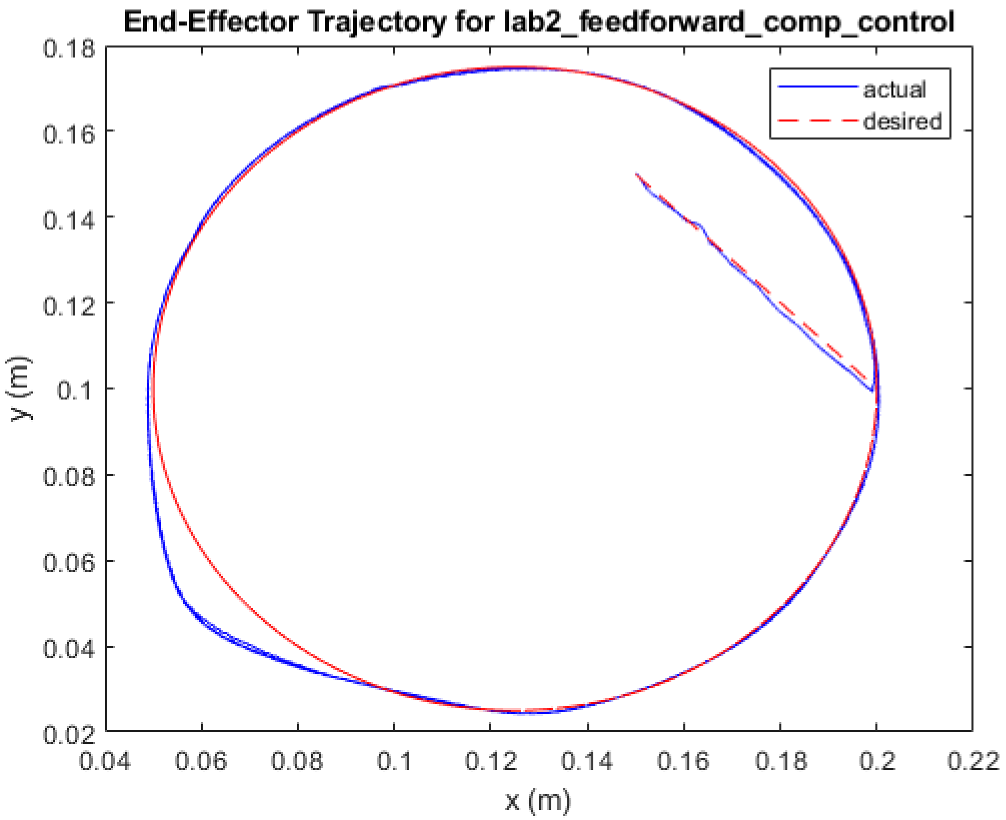
2. Robustness

Note: 2 weights on the 2nd link were used to simulate a disturbance.

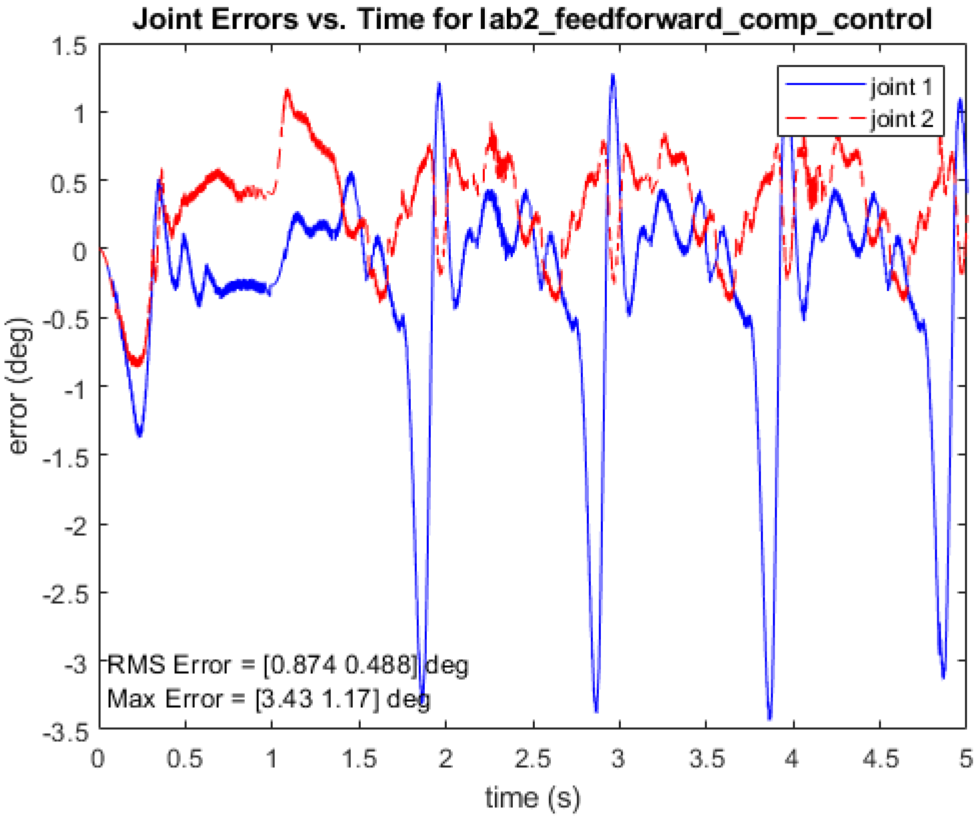
2.1. Feedforward compensation with disturbance

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

2.1.1.1. X-y trajectory



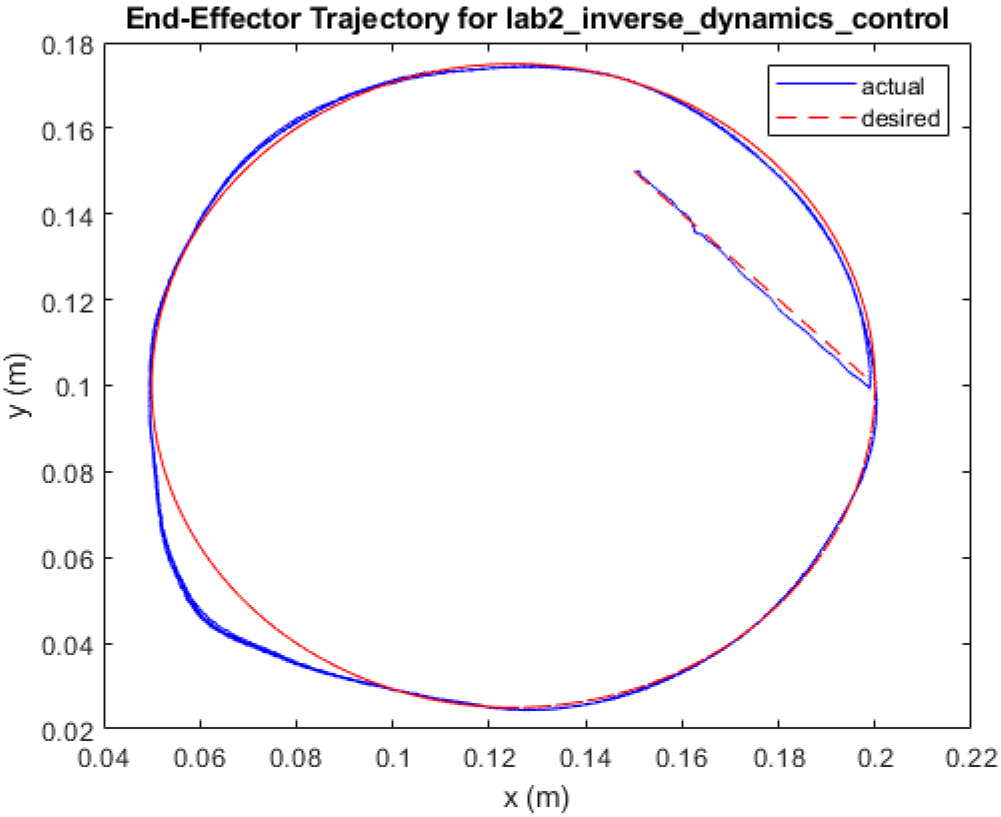
2.1.1.2. Joint angle errors



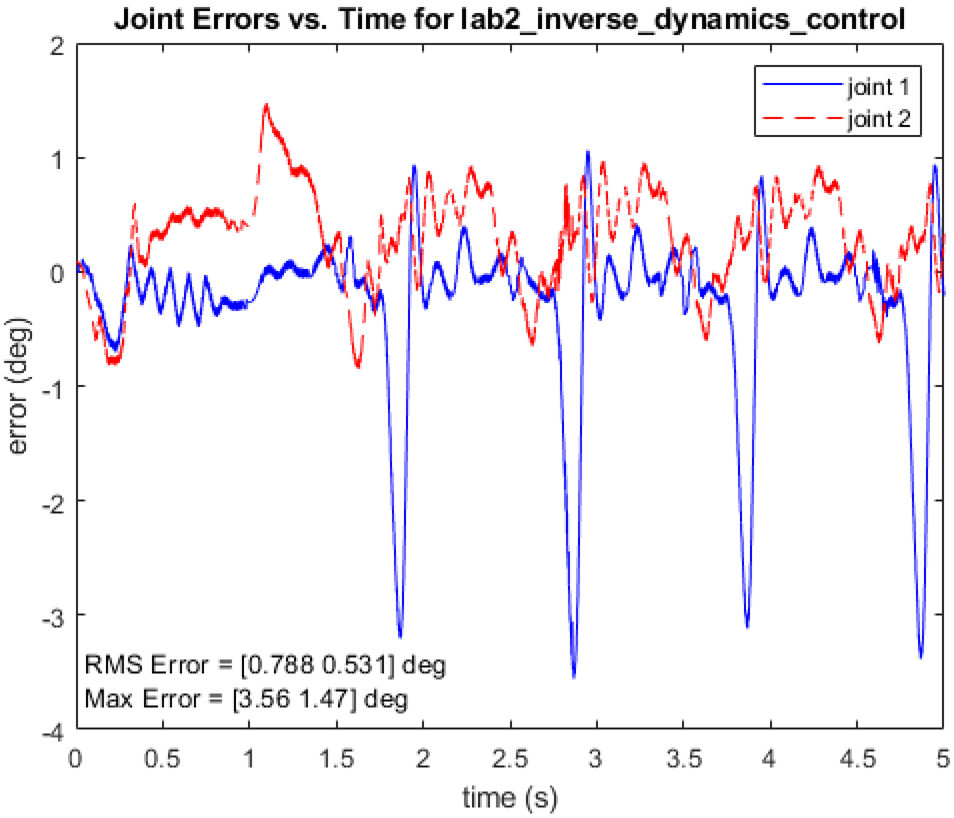
2.2. Inverse dynamics control with disturbance

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

2.2.1.1. X-y trajectory



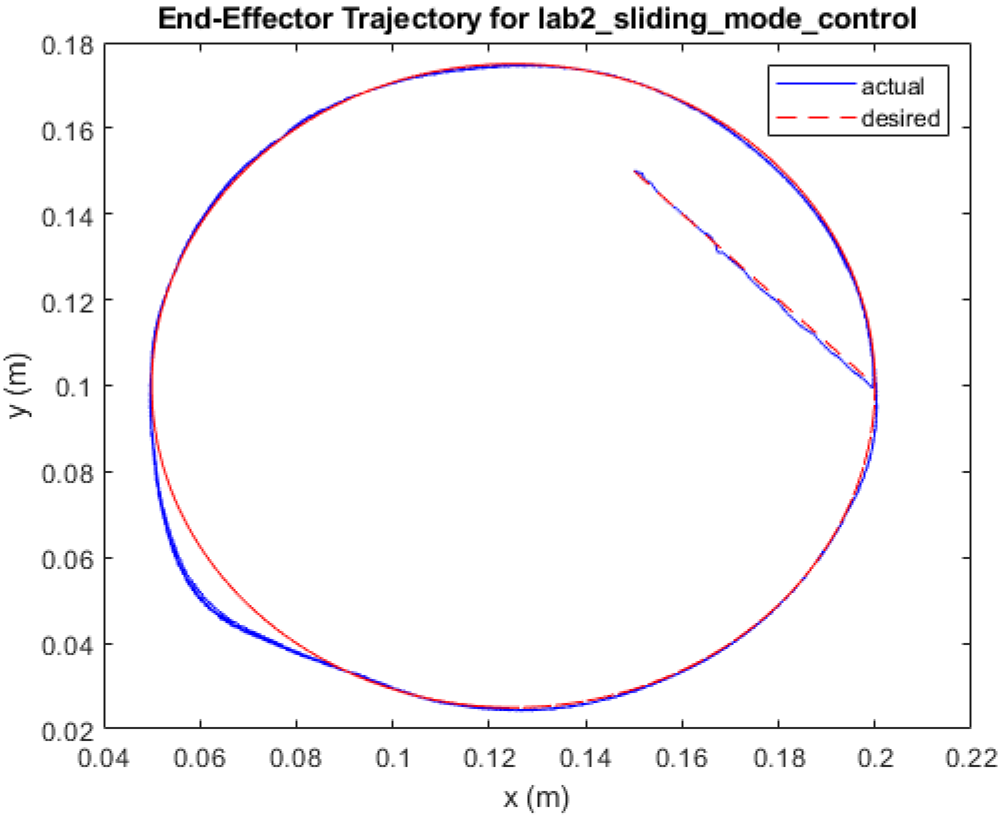
2.2.1.2. Joint angle errors



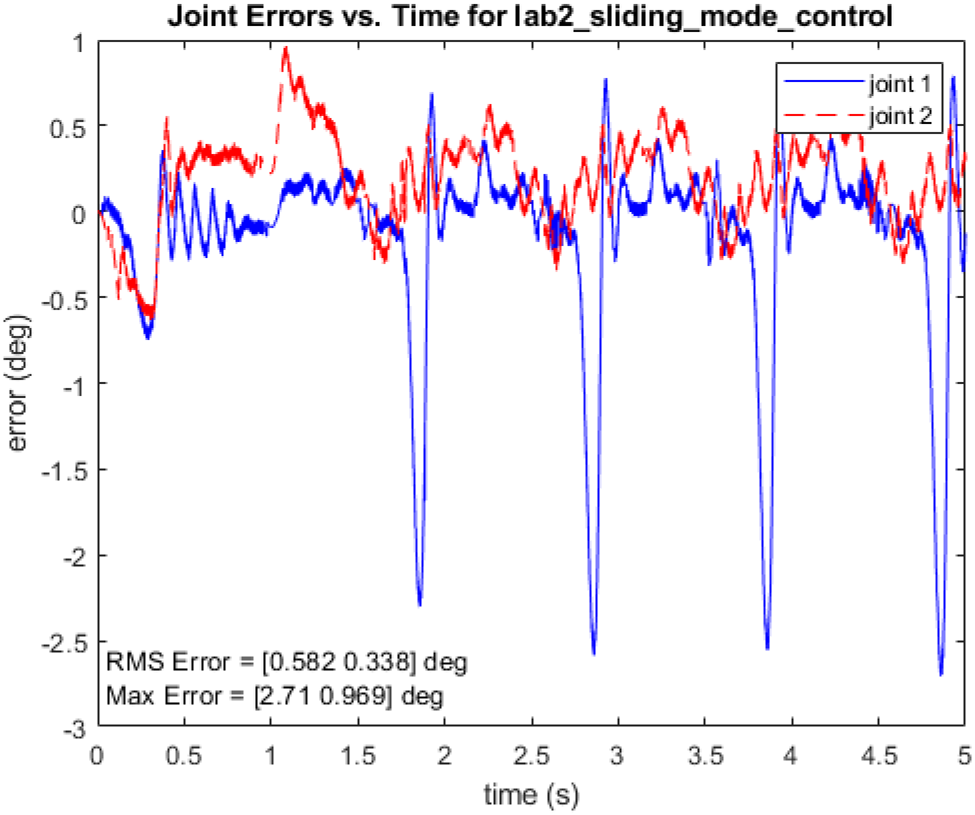
2.3. Sliding mode control with disturbance

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

2.3.1.1. X-y trajectory

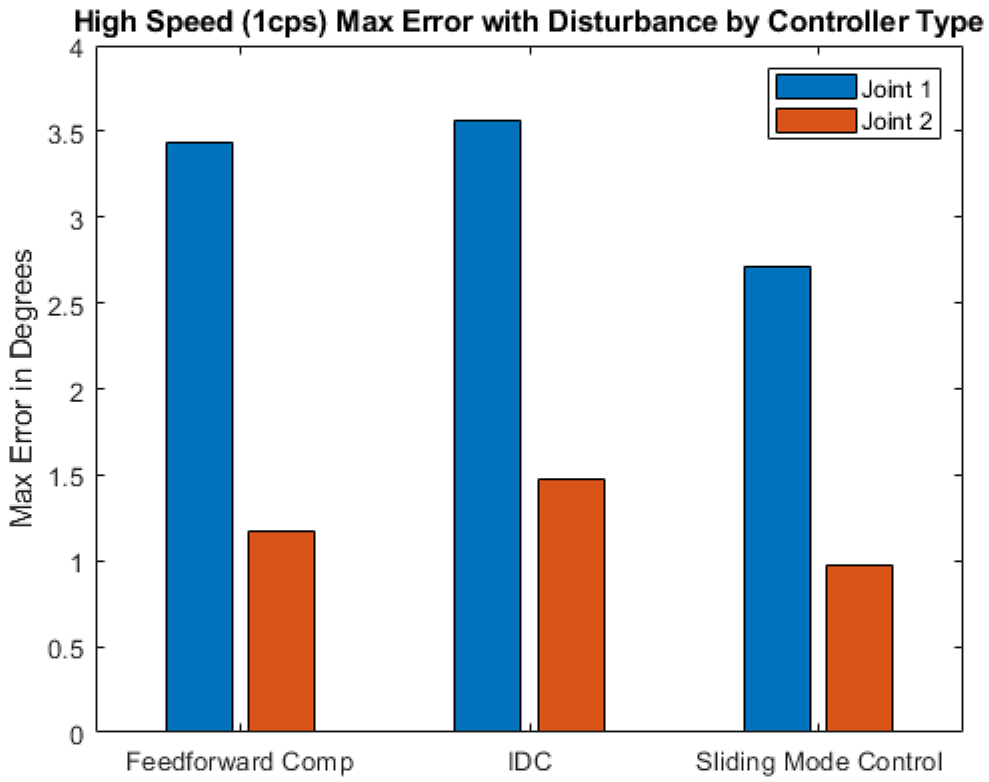


2.3.1.2. Joint angle errors

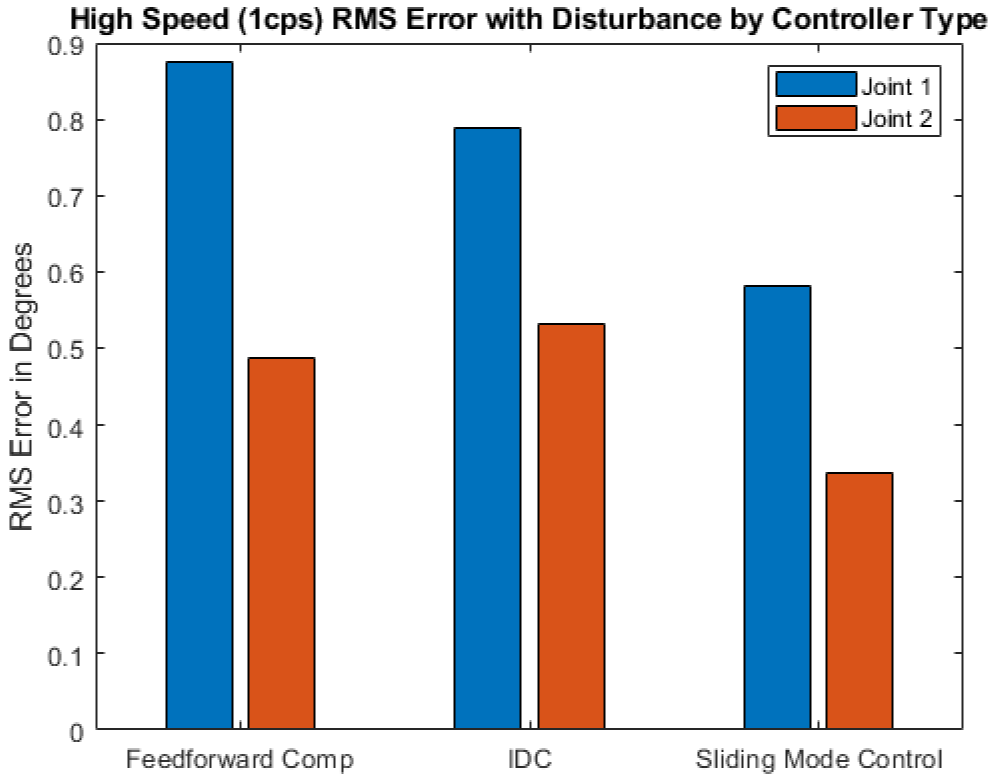


2.4. Controller robustness comparison

2.4.1. Peak joint errors



2.4.2. Root-Mean-Square joint errors



2.4.3. Comparison

As can be seen above feedforward comp and inverse dynamics control have similar errors in terms of RMS and max error with feedforward comp having less max error and inverse dynamics control having less RMS error. On the other hand sliding mode control clearly has the lowest RMS and max error. Sliding mode control deals with model uncertainty much better than the other two controllers and as a result with a disturbance (extra mass on link 2) the controller has the ability to deal with this disturbance robustly.

2.4.4. Comparison Code

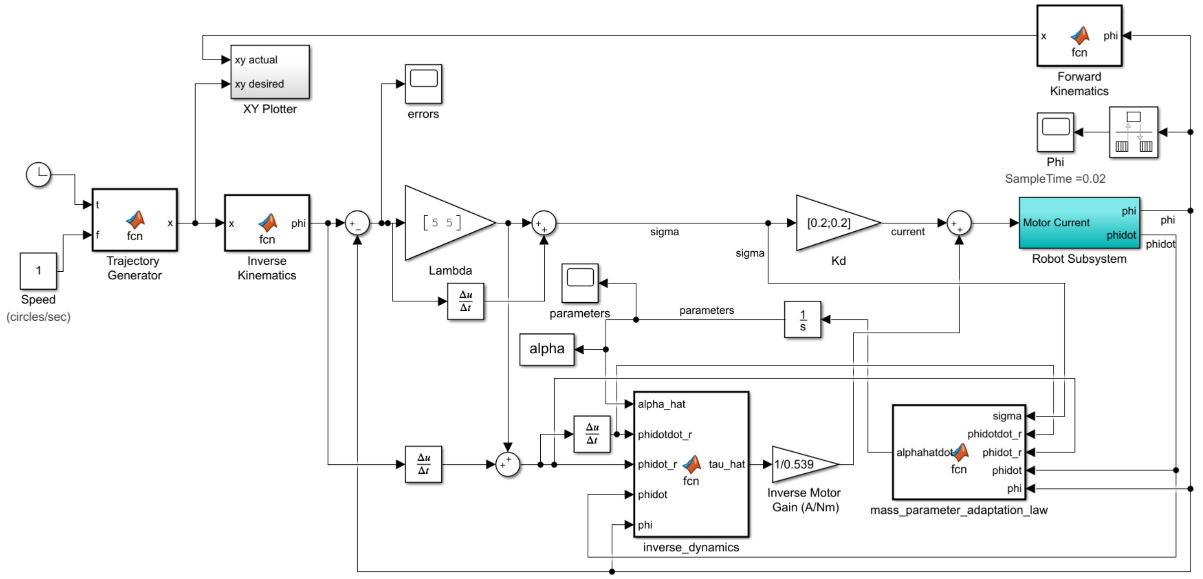
```
%% ME EN 6230 Lab 2 Ryan Dalby
% Robustness Comparison
% close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from
becoming subscript

% errors are in degrees
% PD feedforward comp w/ disturbance
feedforwardcomp_fast_rms = [0.874 0.488];
feedforwardcomp_fast_max = [3.43 1.17];

% IDC w/ disturbance
idc_fast_rms = [0.788 0.531];
idc_fast_max = [3.56 1.47];

% Sliding mode control w/ disturbance
slidingmodecontrol_fast_rms = [0.582 0.338];
slidingmodecontrol_fast_max = [2.71 0.969];

fast_rms = [feedforwardcomp_fast_rms; idc_fast_rms;
slidingmodecontrol_fast_rms];
fast_max = [feedforwardcomp_fast_max; idc_fast_max;
slidingmodecontrol_fast_max];

controller_labels = {'Feedforward Comp', 'IDC', 'Sliding Mode Control'};
controller_labels_cat = categorical(controller_labels);
controller_labels_cat = reordercats(controller_labels_cat,
string(controller_labels_cat));

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

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

3. Adaptive Control

Note: A diagonal 4x4 gamma matrix was used with values of 2000. (For the run with a disturbance (weights) the gamma for the diagonal corresponding to r01*m1 + a1*m2 was 200 to get faster convergence for this parameter)

A lambda value of [5 5] was used to not saturate the amplifier and a Kd value of [0.2 0.2] was used. These are not equivalent to the PD gains used in feedforward control but the scaled down lambda prevents saturation of the amplifier.

3.1. Model



3.2. Code

Code for inverse_dynamics Block:

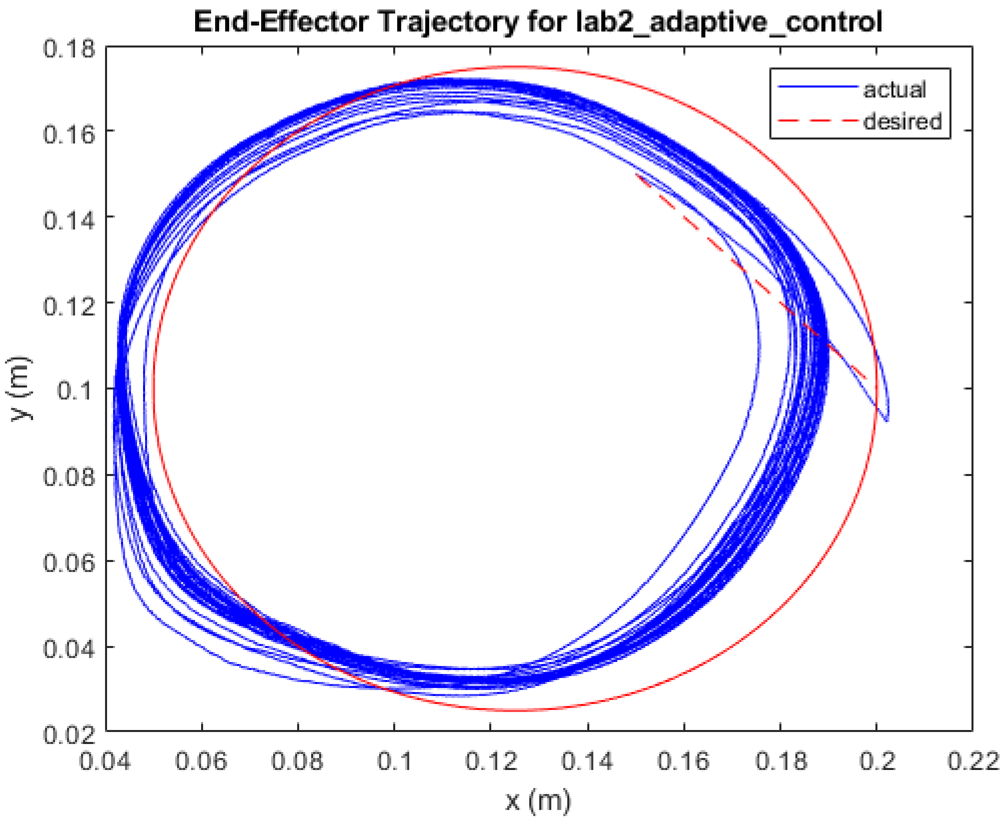
```
function tau_hat = fcn(alpha_hat,phidotdot_r,phidot_r,phidot,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length
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;

F1 = N1^2*b1*phidot(1) + N1*c1*sign(phidot(1));
F2 = N2^2*b2*phidot(2) + N2*c2*sign(phidot(2));

F = [F1;F2]; % frictional torques

Y11 = phidotdot_r(1);
Y12 = a1*cos(phi(2)-phi(1))*phidotdot_r(2) -
a1*sin(phi(2)-phi(1))*phidot(2)*phidot_r(2);
Y13 = g*cos(phi(1));
Y14 = 0;
Y21 = 0;
Y22 = a1*cos(phi(2)-phi(1))*phidotdot_r(1) +
a1*sin(phi(2)-phi(1))*phidot(1)*phidot_r(1) + g*cos(phi(2));
Y23 = 0;
Y24 = phidotdot_r(2);

Y = [Y11 Y12 Y13 Y14; Y21 Y22 Y23 Y24];

tau_hat = Y*alpha_hat + F;
```

Code for mass_parameter_adaptation_law Block:

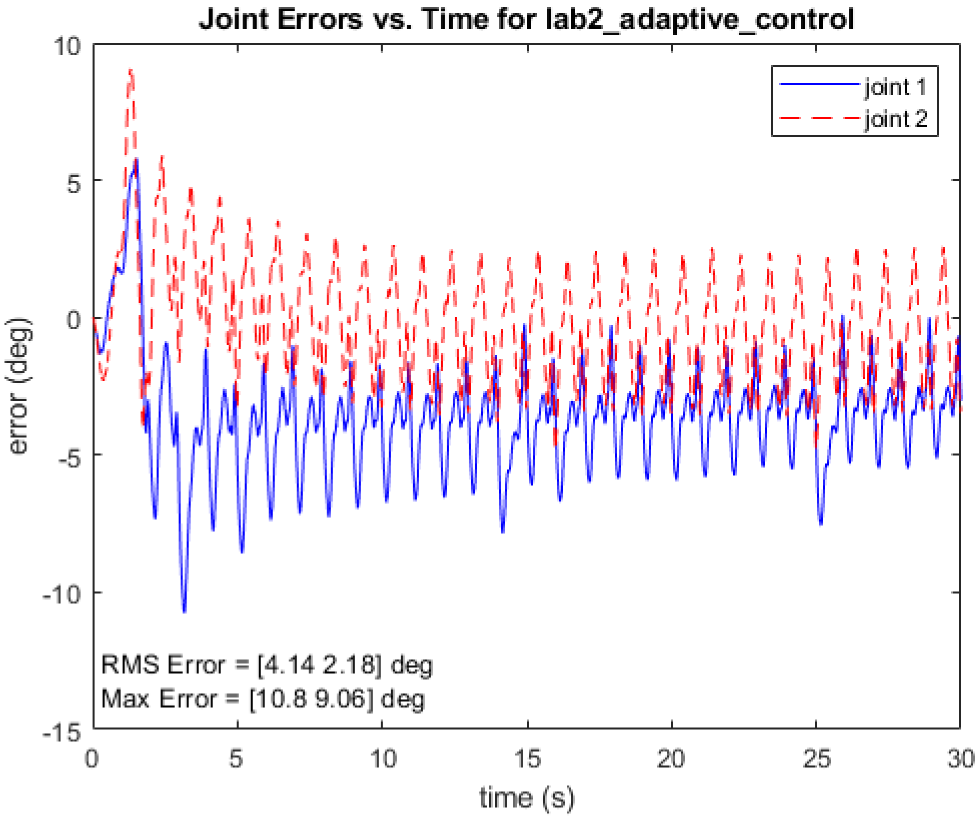
```
function alphahatdot = fcn(sigma,phidotdot_r,phidot_r,phidot,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

a1 = 0.15;  % link 1 length
a2 = 0.15;  % link 2 length

g = 9.8; % gravitational constant

Y11 = phidotdot_r(1);
Y12 = a1*cos(phi(2)-phi(1))*phidotdot_r(2) -
a1*sin(phi(2)-phi(1))*phidot(2)*phidot_r(2);
Y13 = g*cos(phi(1));
Y14 = 0;
Y21 = 0;
Y22 = a1*cos(phi(2)-phi(1))*phidotdot_r(1) +
a1*sin(phi(2)-phi(1))*phidot(1)*phidot_r(1) + g*cos(phi(2));
Y23 = 0;
Y24 = phidotdot_r(2);

Y = [Y11 Y12 Y13 Y14; Y21 Y22 Y23 Y24];

gamma_val = 15;
gamma = [gamma_val 0 0 0; 0 gamma_val 0 0; 0 0 gamma_val 0; 0 0 0
gamma_val];

alphahatdot = inv(gamma) * transpose(Y) * sigma;
```

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

3.3.1. X-y trajectory

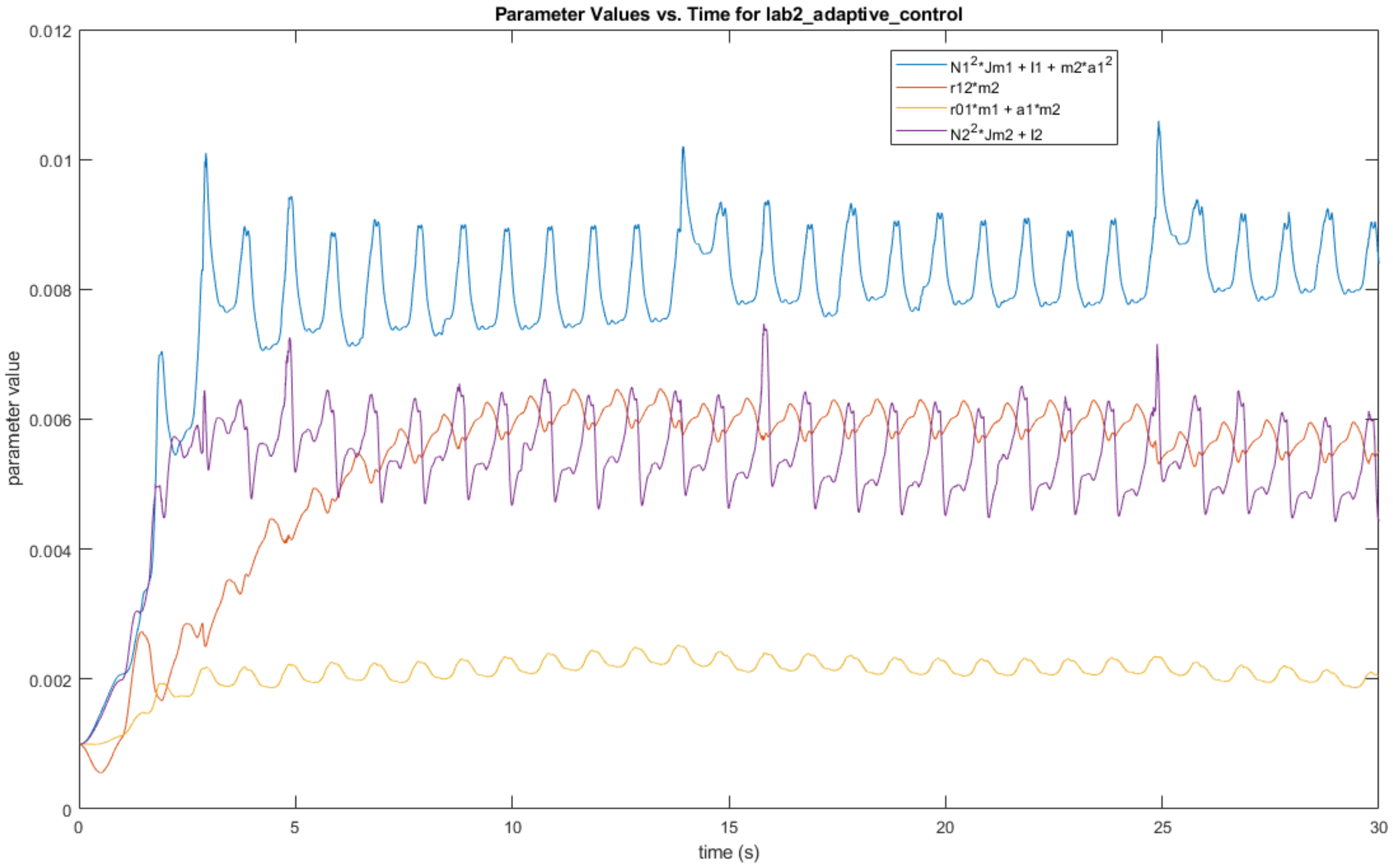


3.3.2. Joint angle errors

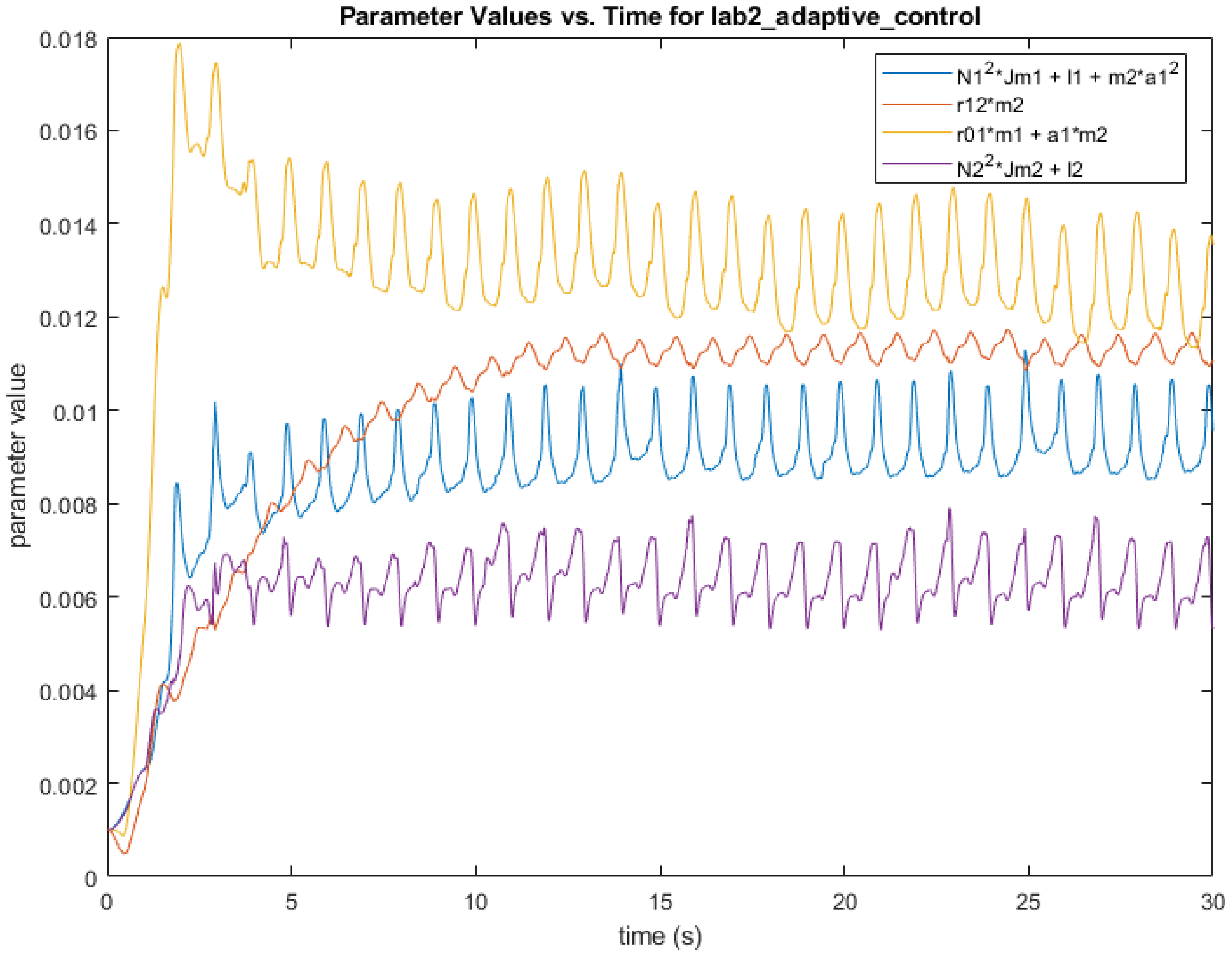


3.3.3. Parameter estimates vs. time

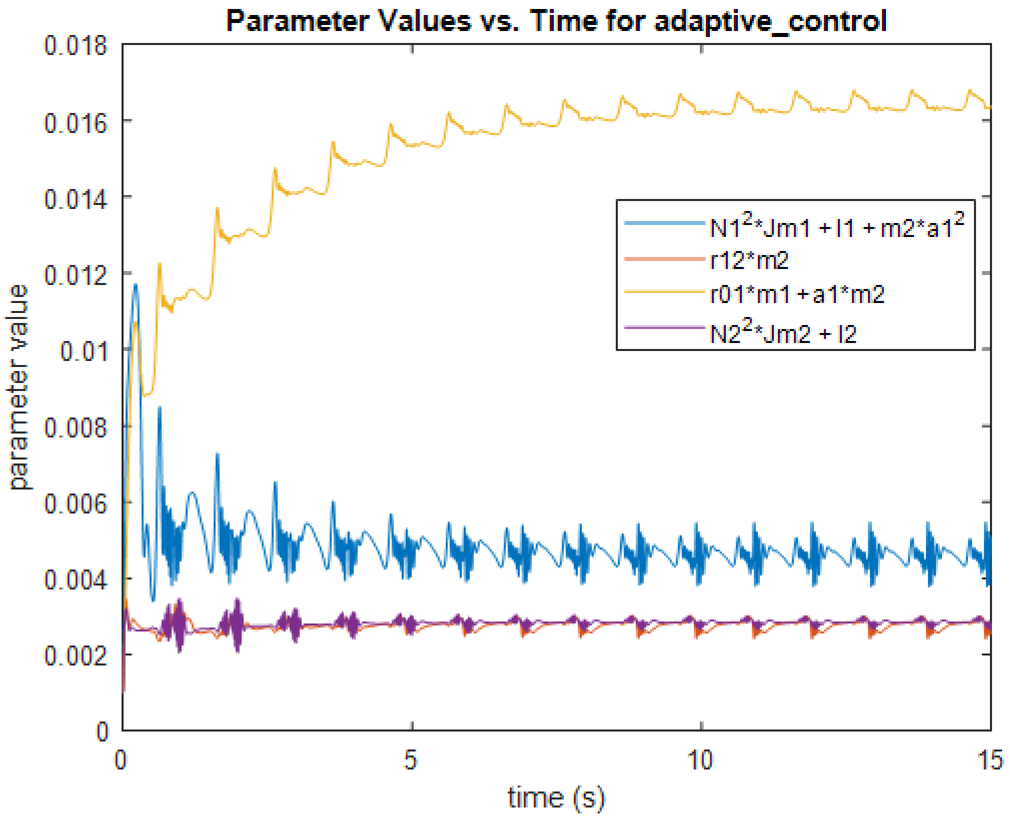
3.3.3.1. In lab



3.3.3.2. In lab with disturbance (brass weights)



3.3.3.3. In simulation



3.4. Analysis

Looking at joint angle errors the adaptive controller performs overall much worse than the feedforward compensation when compared to the feedforward compensation controller in the plot from 1.5. This is because the adaptive controller has to adapt its feedforward compensation from initial values of 0. As a result, in the lab the controller does not converge to a steady state estimate of the parameters since the controller tries to adjust the parameter values as it passes through different parts of the trajectory. Thus, the controller oscillates around the converged parameter value and continues to have persistent error. It was interesting to realize that the adaptive controller continued to converge to a trajectory that was offset from the desired trajectory even after trying many different gamma and PD values. This is a sort of steady state error and may be able to be addressed by adding an integral term to the PD controller or performing more tuning of the controller tuning parameters.

The controller converged to parameter values of [0.008; 0.006; 0.002; 0.0055] which is not the same converged parameter values as the true parameter values of [0.0055575; 0.002772; 0.017254; 0.003485] or the simulation parameter values of [0.0045; 0.0027; 0.0162; 0.0027]. With two extra masses on link 2 acting as a disturbance the converged values were [0.009; 0.011; 0.013; 0.006]. These parameter values vary from the parameter values without the disturbance but the controller appears to adapt to the change in the mass of link 2. For example, term 2 (r12*m2) is directly proportional mass 2 and has a substantially different parameter value with the added mass disturbance.

Appendix- Plotting Code (Used to make simulation plots)

```
%% ME EN 6230    Lab 2 Plot Joint Error and Trajectory    Ryan Dalby
set(groot, "DefaultTextInterpreter", "none") % Prevents underscore from
becoming subscript

% Extract necessary data, will error if the data does not exist
time = errors.time; % s
time_datapoints = length(time);
model_title = extractBefore(errors.blockName, "/errors");
joint_errors = rad2deg(transpose(reshape(errors.signals.values, [2,
time_datapoints]))); % 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;
rms_string = mat2str(RMS_error,3);
text(0.01,0.10,strcat("RMS Error = ", rms_string," deg"), "Units",
"normalized");
hold on;
max_string = mat2str(max_error,3);
text(0.01,0.05,strcat("Max Error = ", max_string," deg"), "Units",
"normalized");
title(strcat("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(strcat("End-Effector Trajectory for ", model_title));
```

```
xlabel("x (m)");
ylabel("y (m)");
legend("actual", "desired");

% If model_title is adaptive_control plot parameter adaptation
if strcmp(model_title, "lab2_adaptive_control")
    figure;
    for i = 1:size(alpha, 2)
        plot(time, alpha(:,i));
        hold on;
    end
    title(strcat("Parameter Values vs. Time for ", model_title))
    xlabel("time (s)");
    ylabel("parameter value");
    legend("N1^2*Jm1 + I1 + m2*a1^2","r12*m2","r01*m1 + a1*m2","N2^2*Jm2 +
I2");
end
```