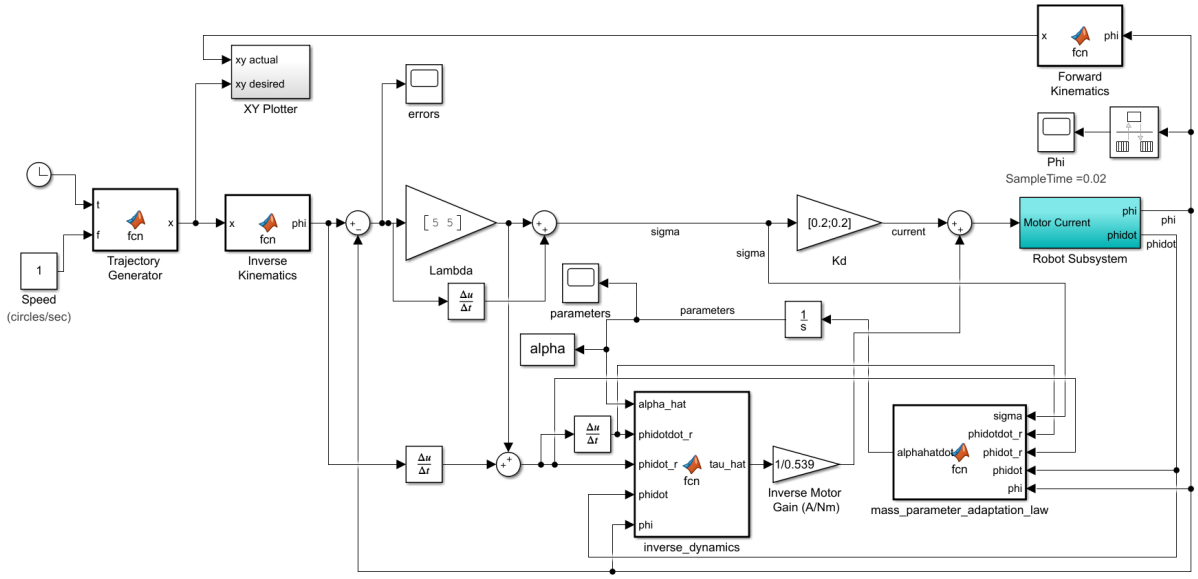
3. Adaptive Control

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:

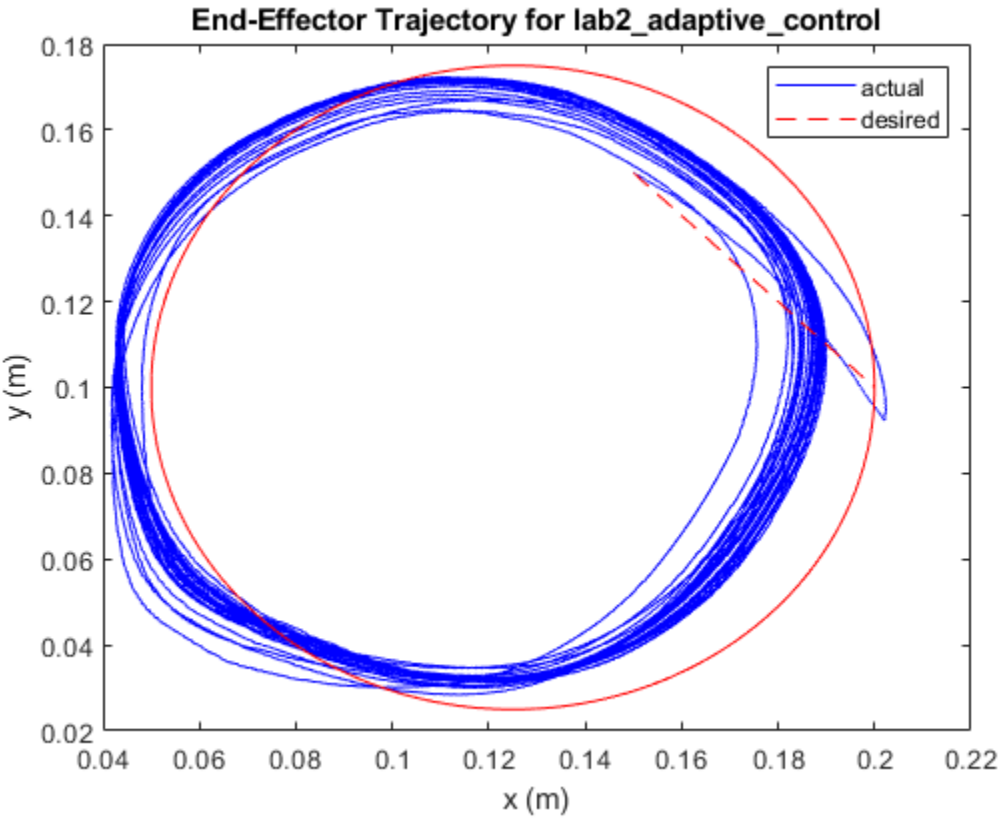
```
function tau_hat = fcn(alpha_hat,phidotdot_r,phidot_r,phidot,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

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:

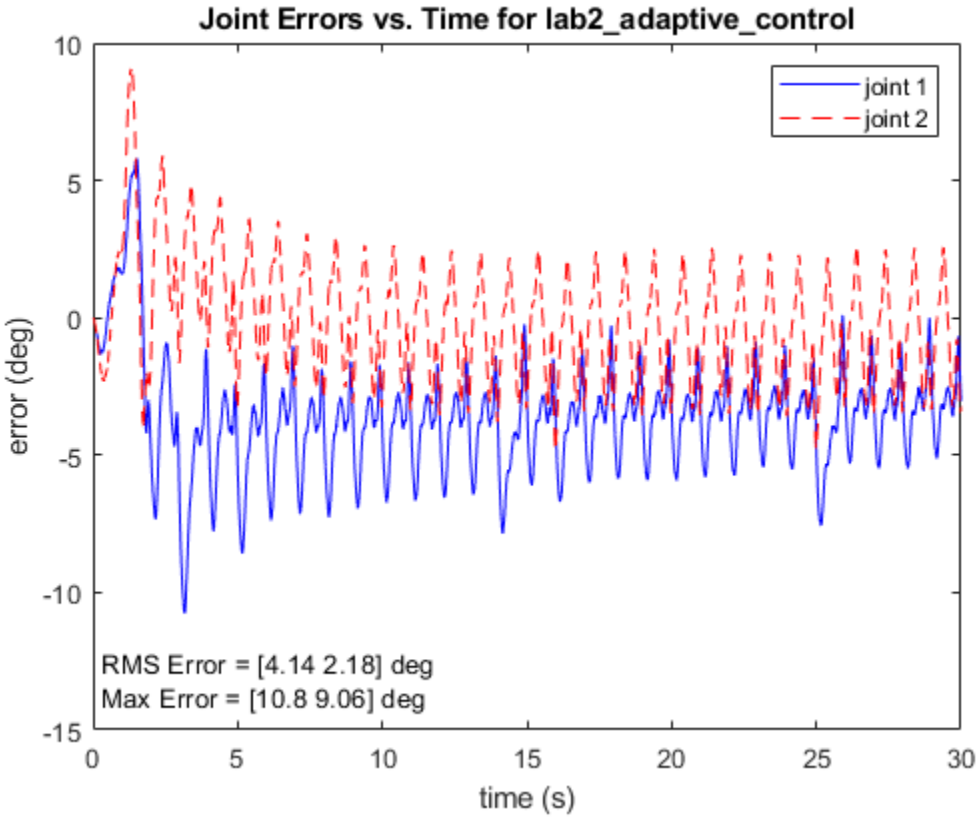
```
function alphahatdot = fcn(sigma,phidotdot_r,phidot_r,phidot,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

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

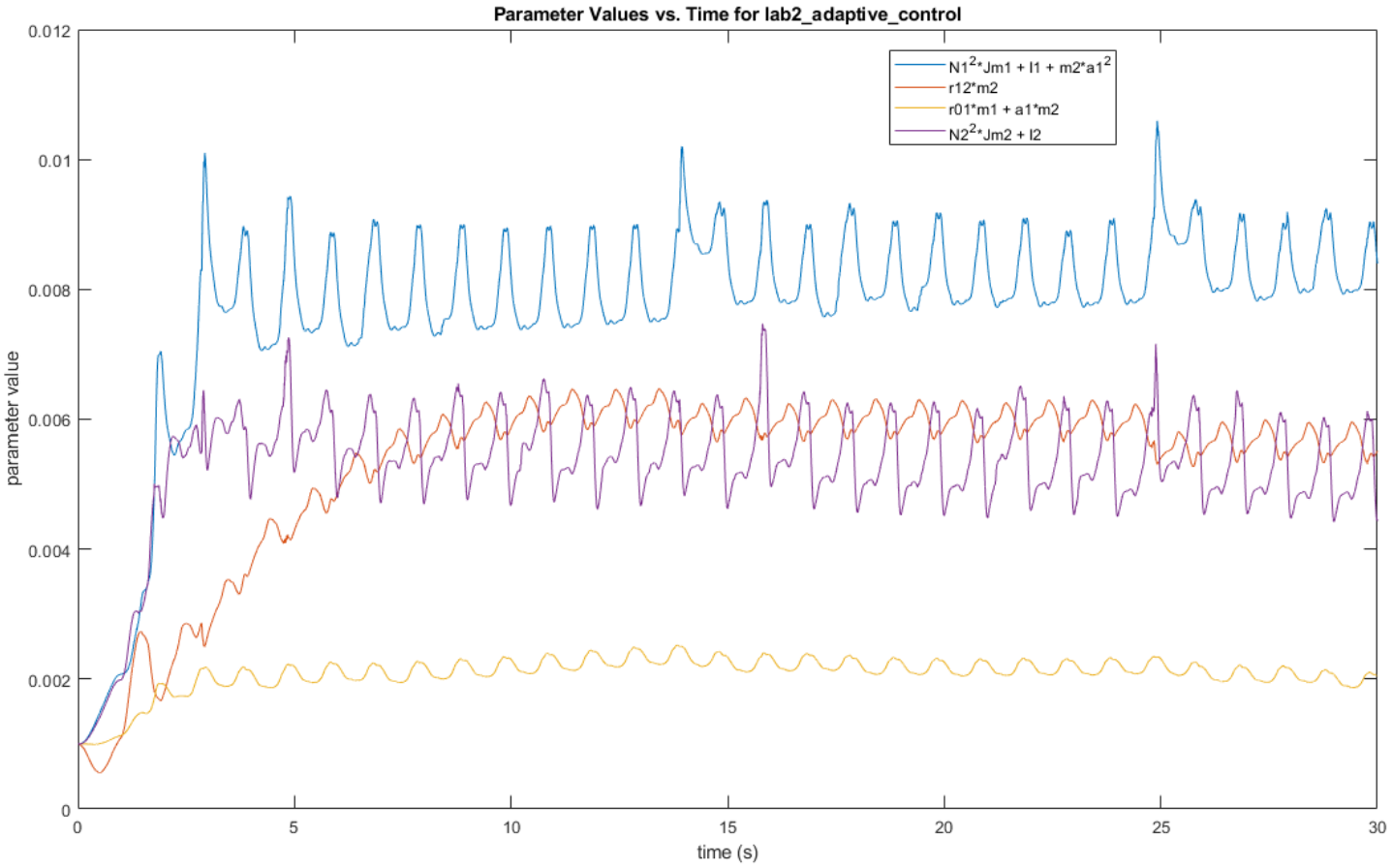


3.3.2. Joint angle errors

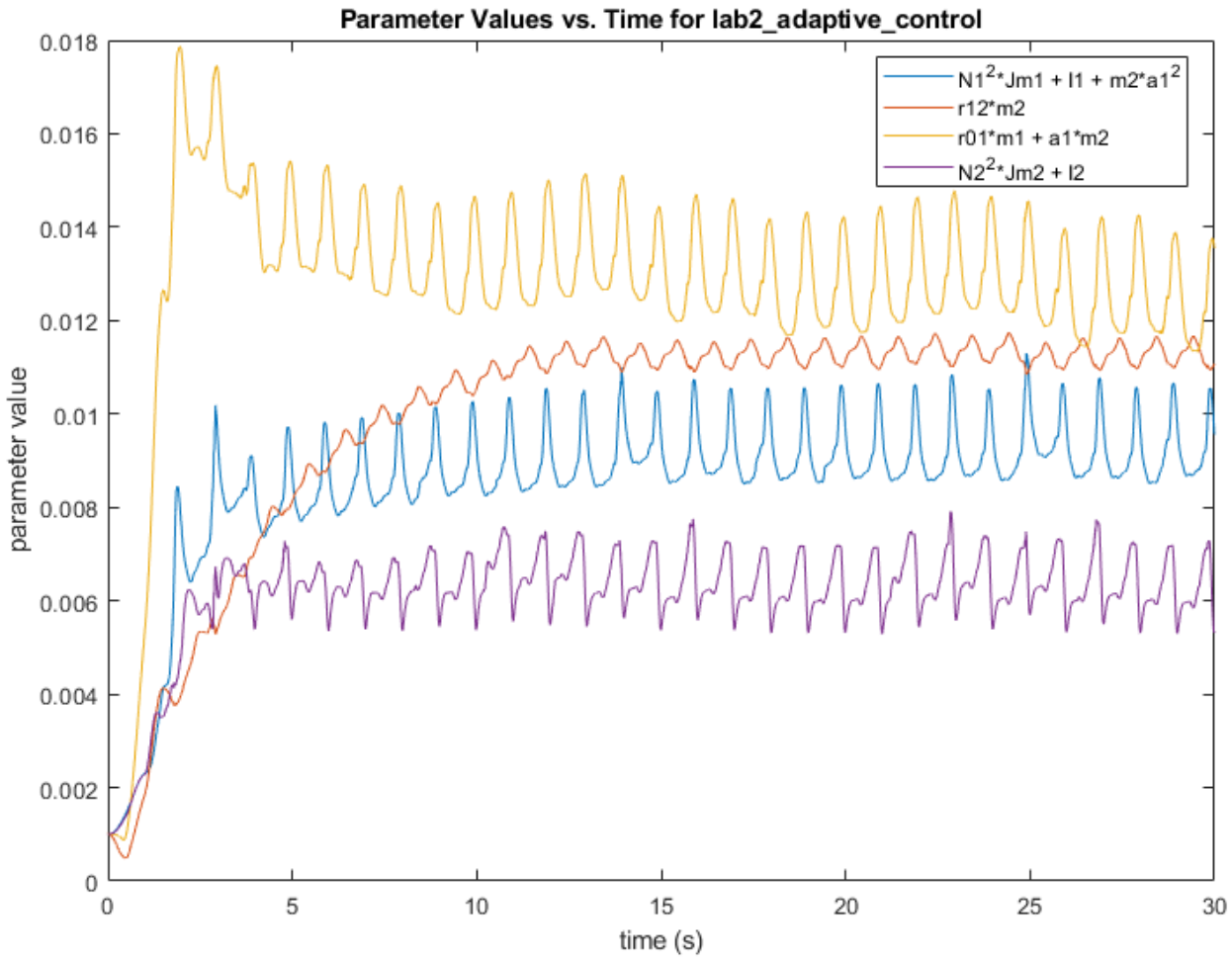


3.3.3. Parameter estimates vs. time

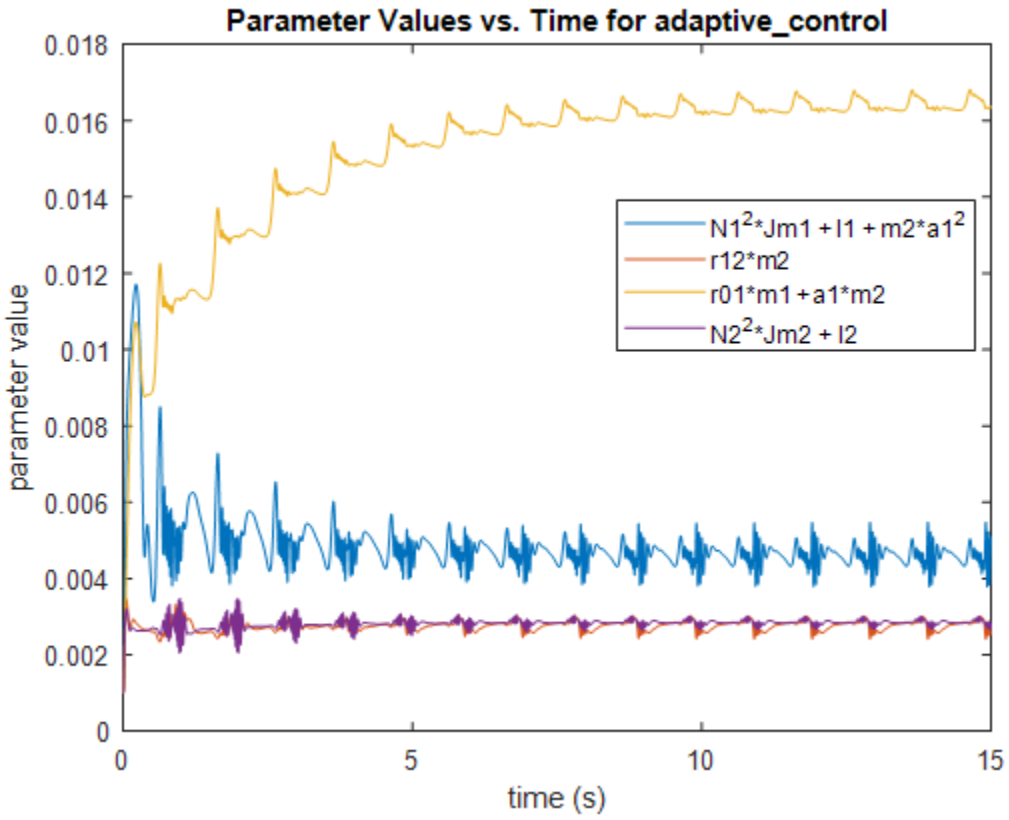
3.3.3.1. In lab



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
```