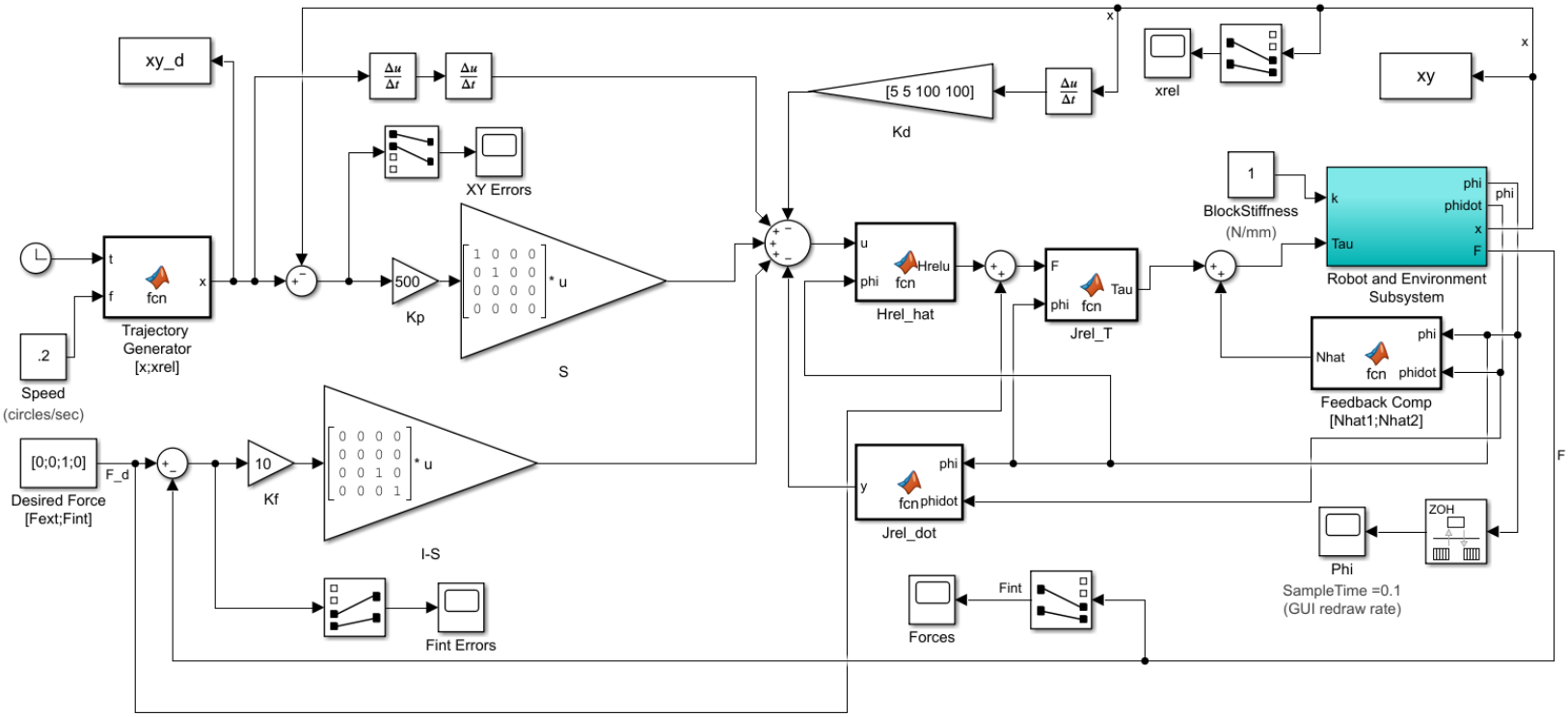
ME EN 6230
Problem Set 11
Ryan Dalby

1. 2-DOF Multi-Arm Hybrid Position/Force Control

Note all simulations are run at a trajectory speed of 0.2 circles/sec.

Model:



Code:

Hrel_hat block:

```
function Hrelu = 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;
g = 9.8; % gravitational constant
N1 = 70; % gear ratios
N2 = 70;

H_1_11 = N1^2*Jm1 + I1 + m2*a1^2;
H_1_12 = a1*r12*m2*cos(phi(2)-phi(1));
H_1_21 = H_1_12;
H_1_22 = N2^2*Jm2 + I2;
H_1 = [H_1_11 H_1_12; H_1_21 H_1_22];  %Inertia Matrix for Robot 1 in Joint
Space

H_2_11 = N1^2*Jm1 + I1 + m2*a1^2;
H_2_12 = a1*r12*m2*cos(phi(2)-phi(1));
H_2_21 = H_2_12;
H_2_22 = N2^2*Jm2 + I2;
H_2 = [H_2_11 H_2_12; H_2_21 H_2_22];  %Inertia Matrix for Robot 2 in Joint
Space

Hcomb = [H_1 zeros(size(H_1)); zeros(size(H_1)) H_2]; %Combined Inertia
Matricies for Two Robots in Joint Space

J_1_11 = -a1*sin(phi(1));
J_1_12 = -a2*sin(phi(2));
J_1_21 = a1*cos(phi(1));
J_1_22 = a2*cos(phi(2));
J_1 = [J_1_11 J_1_12; J_1_21 J_1_22]; %Jacobian for Robot 1
```

```
J_2_11 = -a1*sin(phi(3));
J_2_12 = -a2*sin(phi(4));
J_2_21 = a1*cos(phi(3));
J_2_22 = a2*cos(phi(4));
J_2 = [J_2_11 J_2_12; J_2_21 J_2_22]; %Jacobian for Robot 2

Jrel = [J_1 zeros(size(J_1)); -J_1 J_2]; %Relative Jacobian

Hrel = inv(Jrel')*Hcomb*inv(Jrel);   % Inertia Matrix for Two Robots in
Relative Task Space

Hrelu = Hrel*u;
```

Jrel_T block:

```
function Tau = fcn(F,phi)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.
a1=0.15;
a2=0.15;

J_1_11 = -a1*sin(phi(1));
J_1_12 = -a2*sin(phi(2));
J_1_21 = a1*cos(phi(1));
J_1_22 = a2*cos(phi(2));
J_1 = [J_1_11 J_1_12; J_1_21 J_1_22]; %Jacobian for Robot 1

J_2_11 = -a1*sin(phi(3));
J_2_12 = -a2*sin(phi(4));
J_2_21 = a1*cos(phi(3));
J_2_22 = a2*cos(phi(4));
J_2 = [J_2_11 J_2_12; J_2_21 J_2_22]; %Jacobian for Robot 2

Jrel = [J_1 zeros(size(J_1)); -J_1 J_2]; %Relative Jacobian

Tau = Jrel'*F;
```

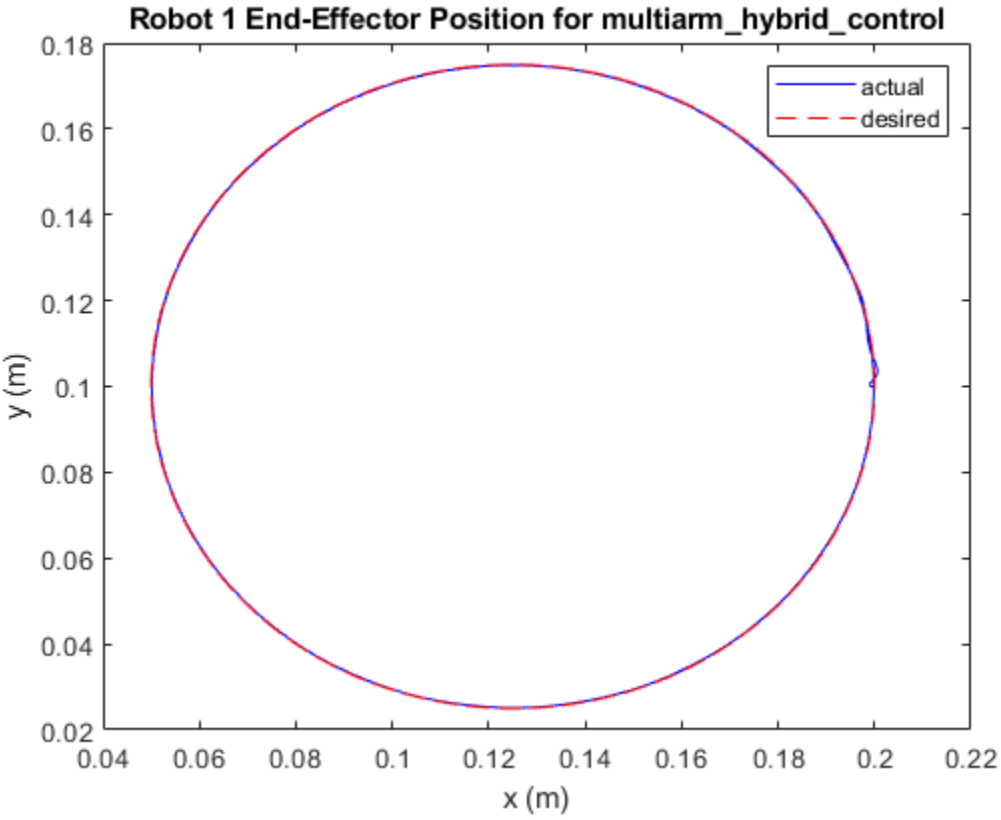
Jrel_dot block:

```
function y = fcn(phi, phidot)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.
a1=0.15;
a2=0.15;

Jdot_1_11 = -a1*cos(phi(1))*phidot(1);
Jdot_1_12 = -a2*cos(phi(2))*phidot(2);
Jdot_1_21 = -a1*sin(phi(1))*phidot(1);
Jdot_1_22 = -a1*sin(phi(2))*phidot(2);
Jdot_1 = [Jdot_1_11 Jdot_1_12; Jdot_1_21 Jdot_1_22]; %Jacobian Derivative
for Robot 1

Jdot_2_11 = -a1*cos(phi(3))*phidot(3);
Jdot_2_12 = -a2*cos(phi(4))*phidot(4);
Jdot_2_21 = -a1*sin(phi(3))*phidot(3);
Jdot_2_22 = -a1*sin(phi(4))*phidot(4);
Jdot_2 = [Jdot_2_11 Jdot_2_12; Jdot_2_21 Jdot_2_22]; %Jacobian Derivative
for Robot 2

Jreldot = [Jdot_1 zeros(size(Jdot_1)); -Jdot_1 Jdot_2]; %Relative Jacobian
Derivative
y = Jreldot*phidot;
```

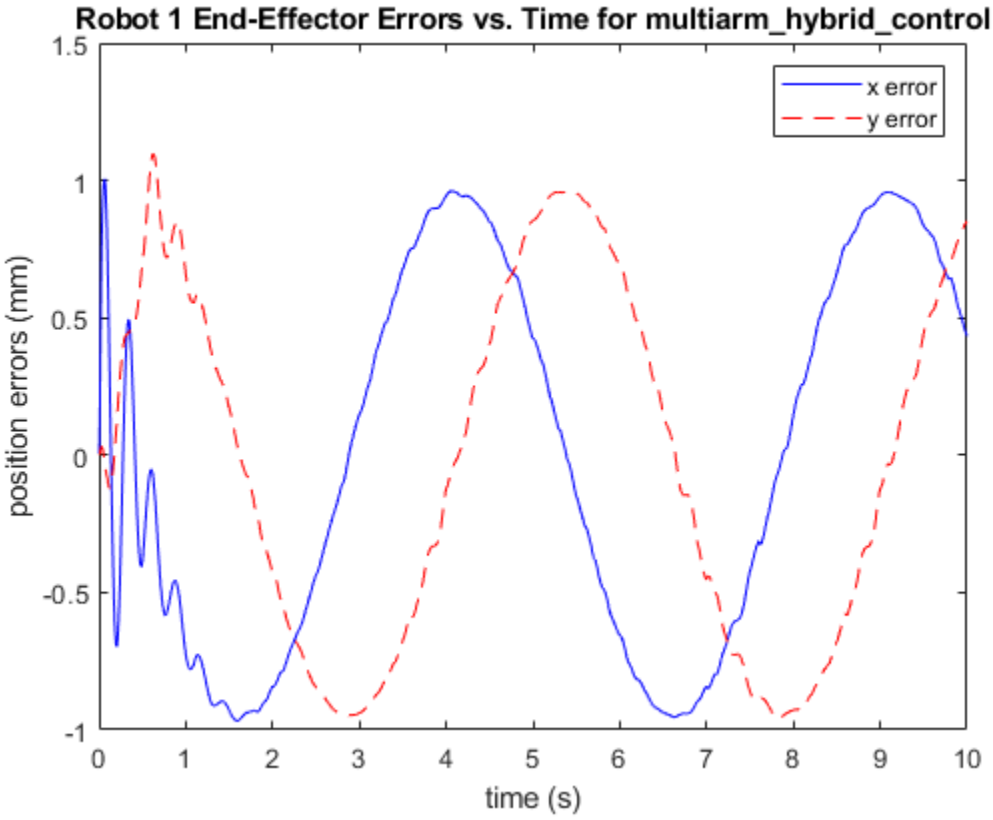
1.1. Block Stiffness

1.1.1. Soft Block(k=1N/mm)

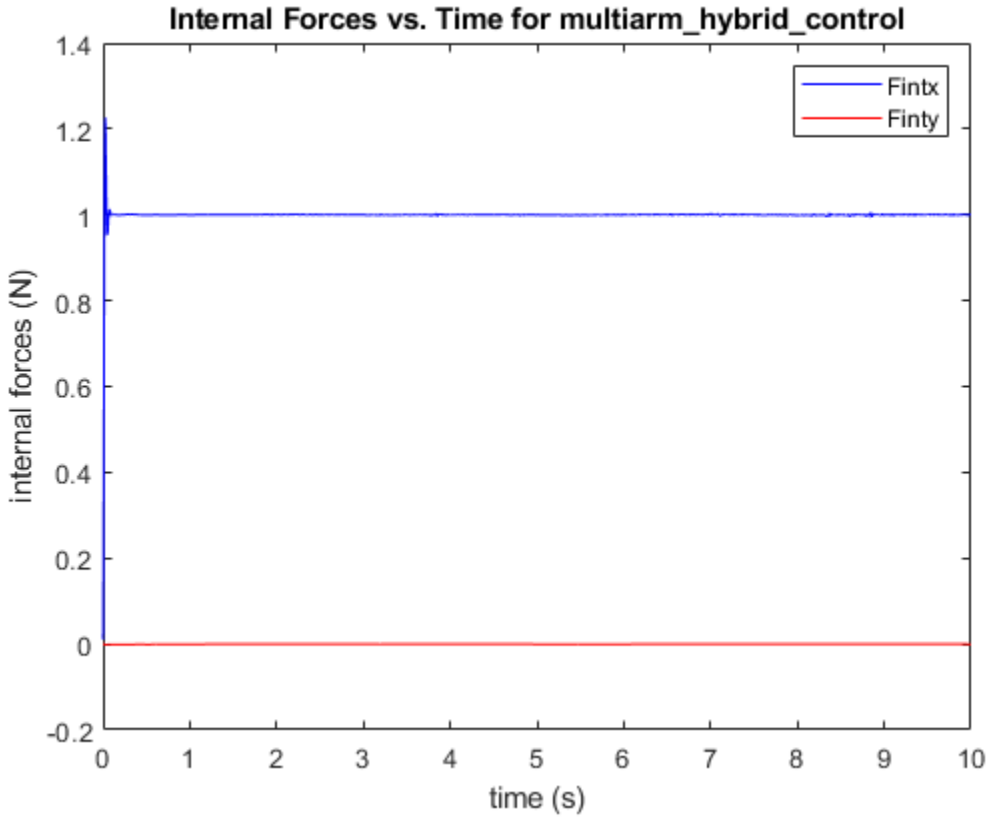
1.1.1.1. Position of Block (Robot 1 Pinned End-Effector)



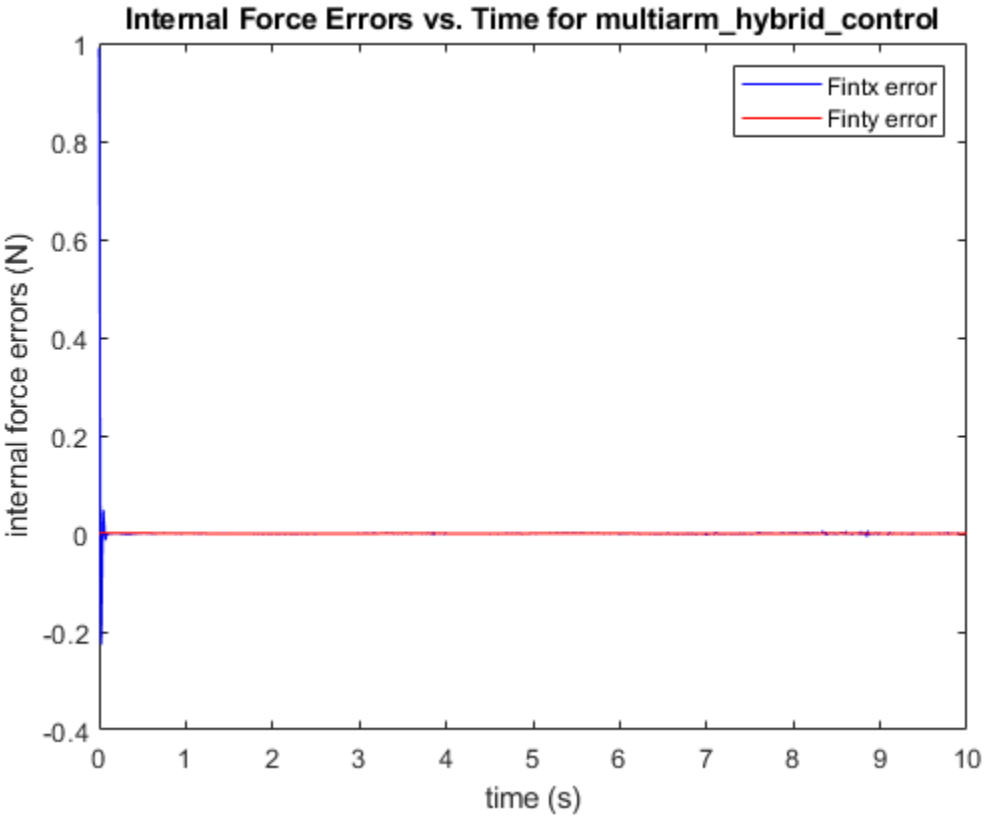
1.1.1.2. Position Errors of Block (Robot 1 Pinned End-Effector)



1.1.1.3. Internal Forces

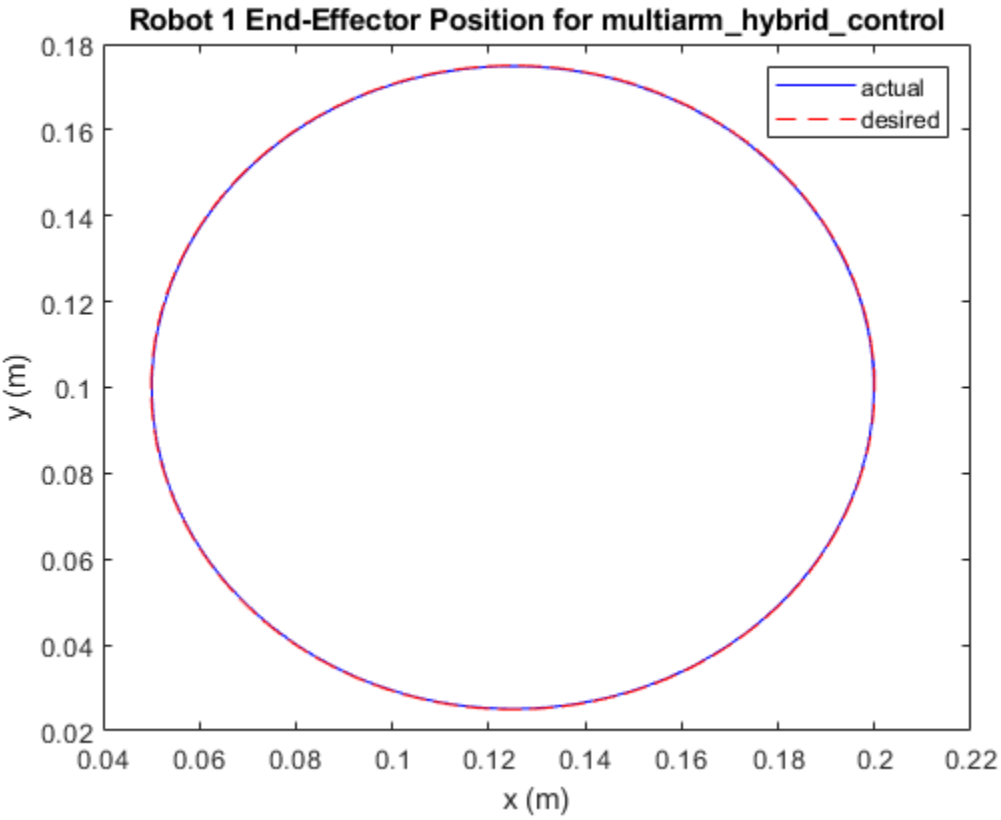


1.1.1.4. Internal Force Errors

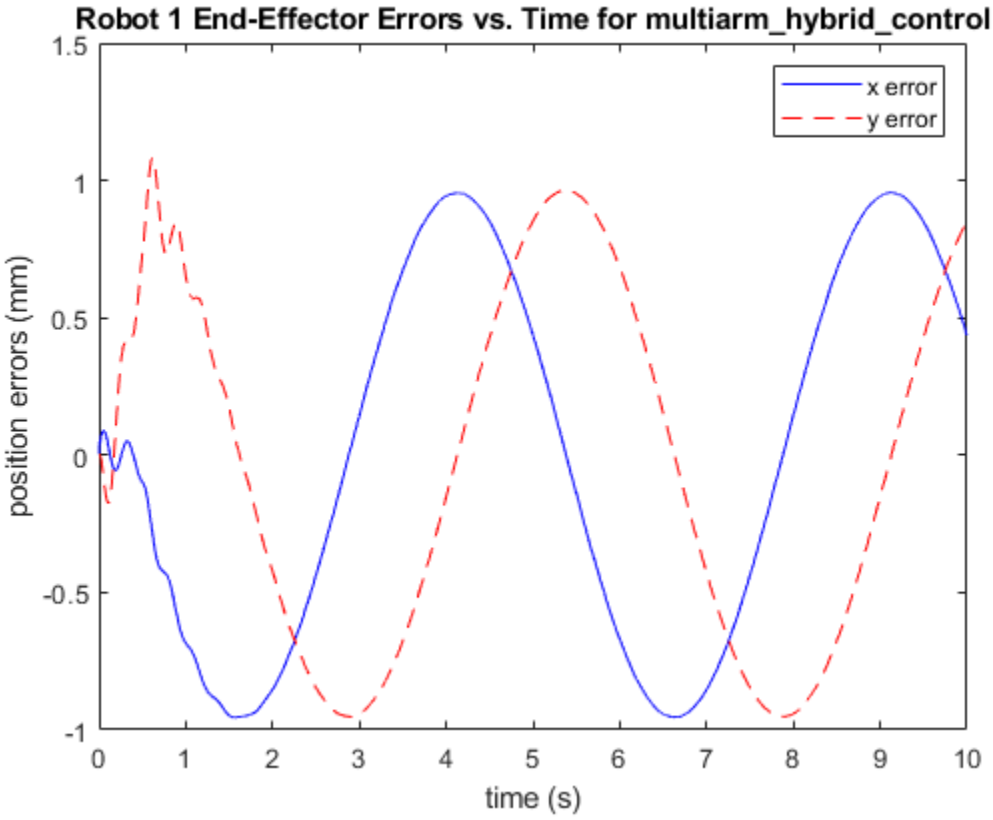


1.1.2. Hard Block (k=10N/mm)

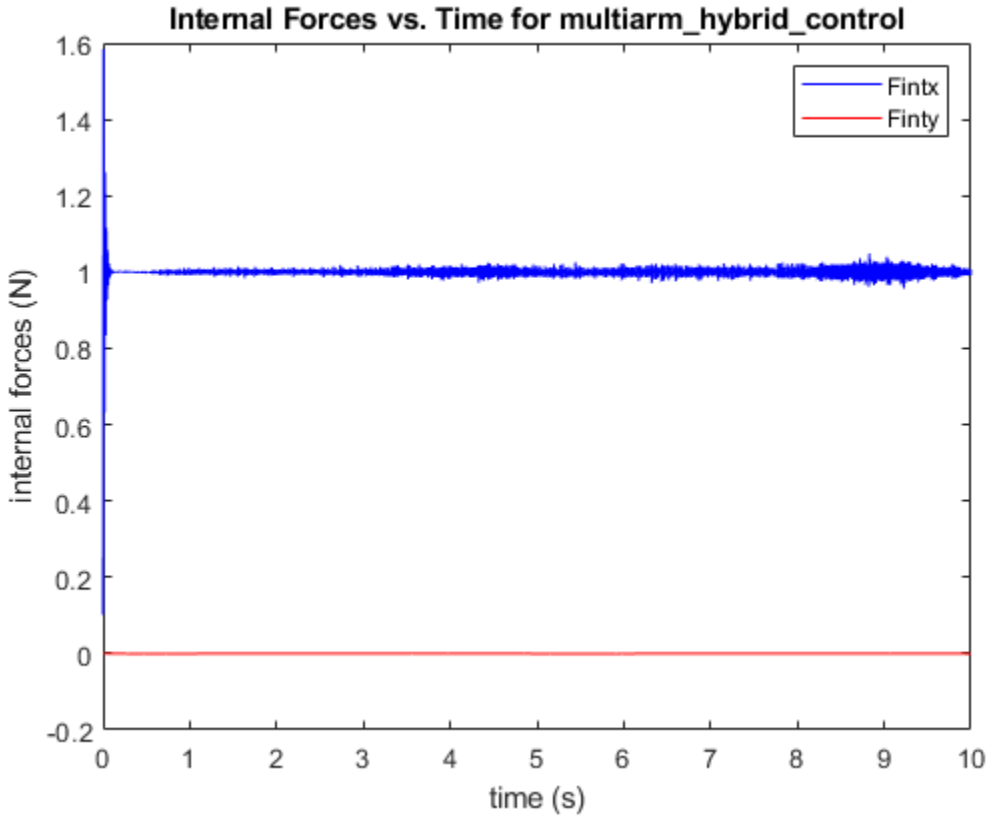
1.1.2.1. Position of Block (Robot 1 Pinned End-Effector)



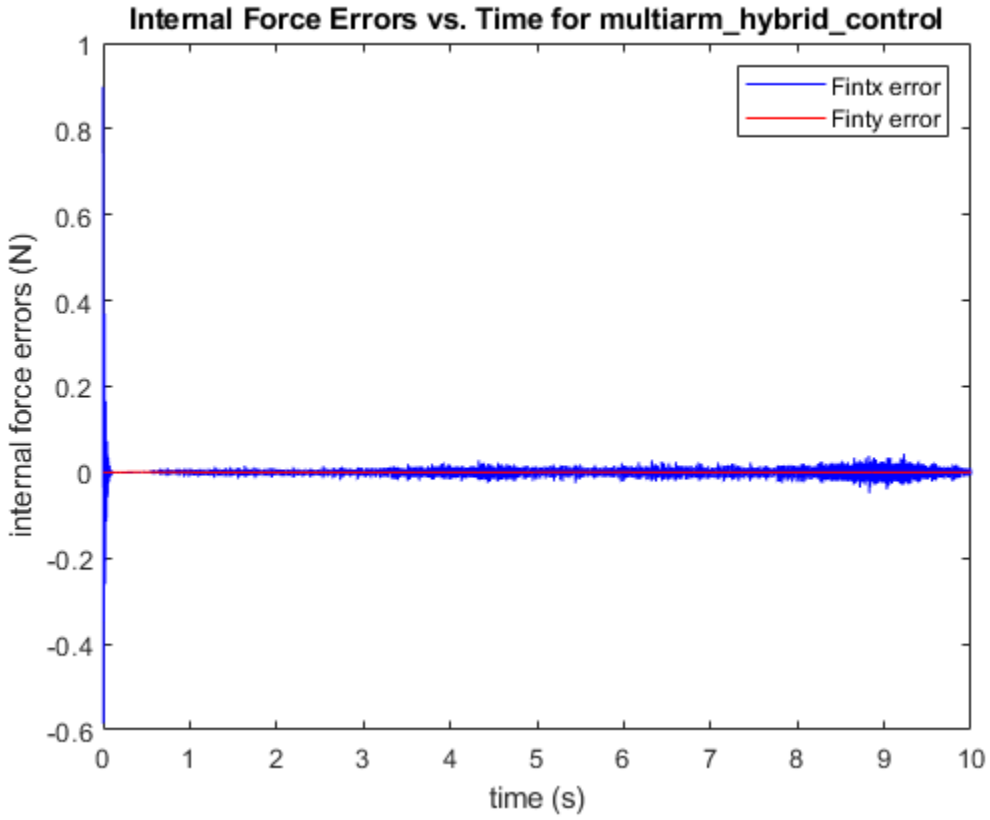
1.1.2.2. Position Errors of Block (Robot 1 Pinned End-Effector)



1.1.2.3. Internal Forces



1.1.2.4. Internal Force Errors



1.1.3. Analysis

Looking at the difference between a soft (k=1N/mm) and hard (k=10N/mm) block using a multi-arm hybrid position/force controller with a desired 1 N x-direction internal force and no y-direction internal force there are a few differences.

In terms of stability, the soft block is more stable as can be seen by the internal force errors plot which shows virtually no oscillatory noise for the soft block but shows some oscillatory noise that grows gradually for the hard block. Although this noise appears to grow continuously, with a simulation time of 50 seconds this noise does not appear to continue to grow meaning that control with the hard block is still stable but shows less stability than the soft block.

In terms of performance, both controllers provide good position and force tracking. The position tracking of the hard block controller appears to have more favorable transient behavior with a more damped transient response when compared to the soft block as shown by the end-effector errors versus time plot.

1.2. Desired Internal Force

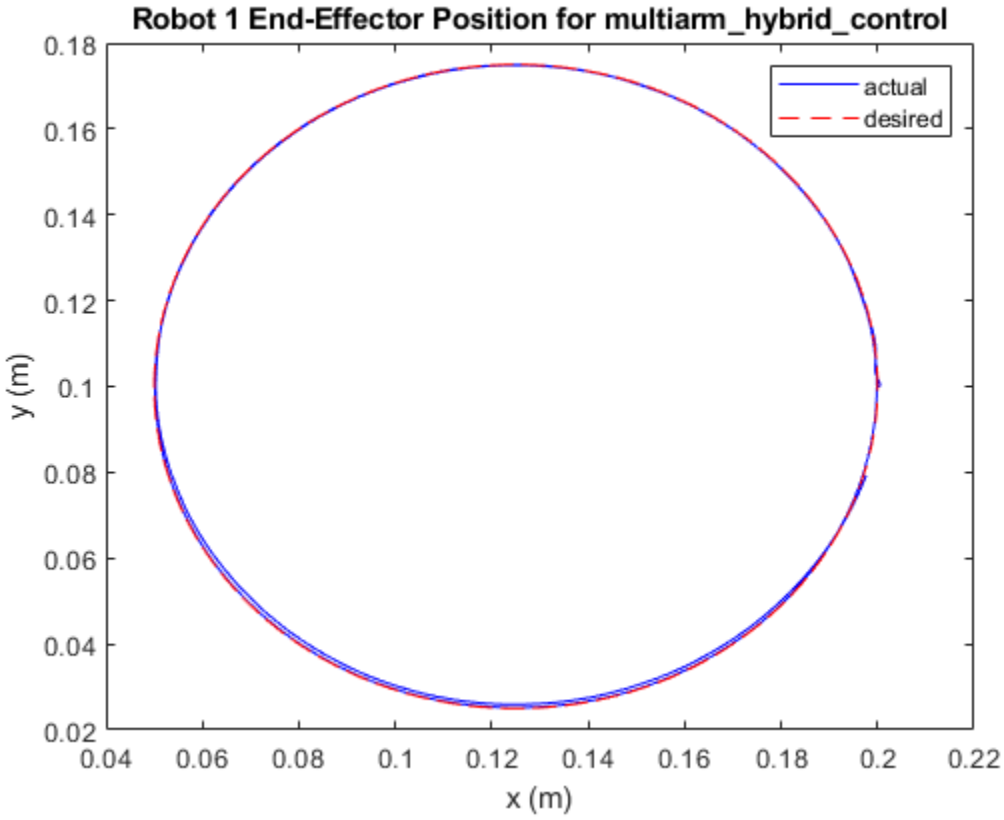
Note all simulations were run with block stiffness of k=1N/mm.

1.2.1. +1 N X-Direction Internal Force and 0 N Y-Direction Internal Force

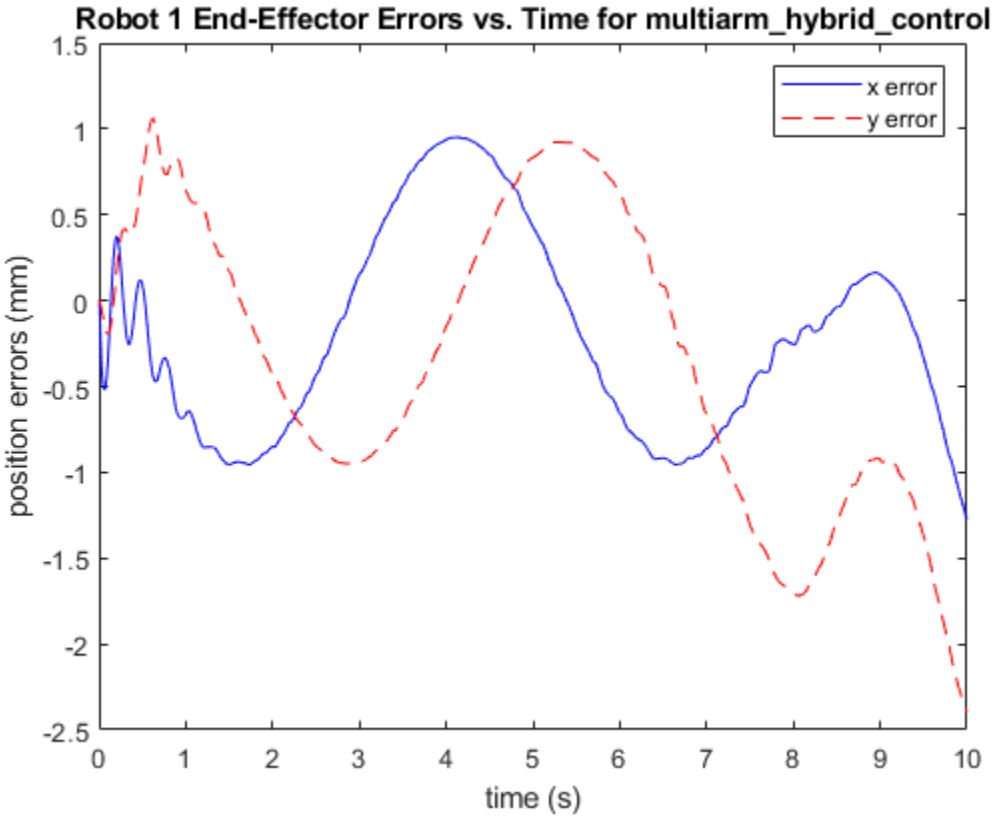
Plots are equivalent to 1.1.1.1-1.1.1.4.

1.2.2. -0.5 N X-Direction Internal Force and 0 N Y-Direction Internal Force

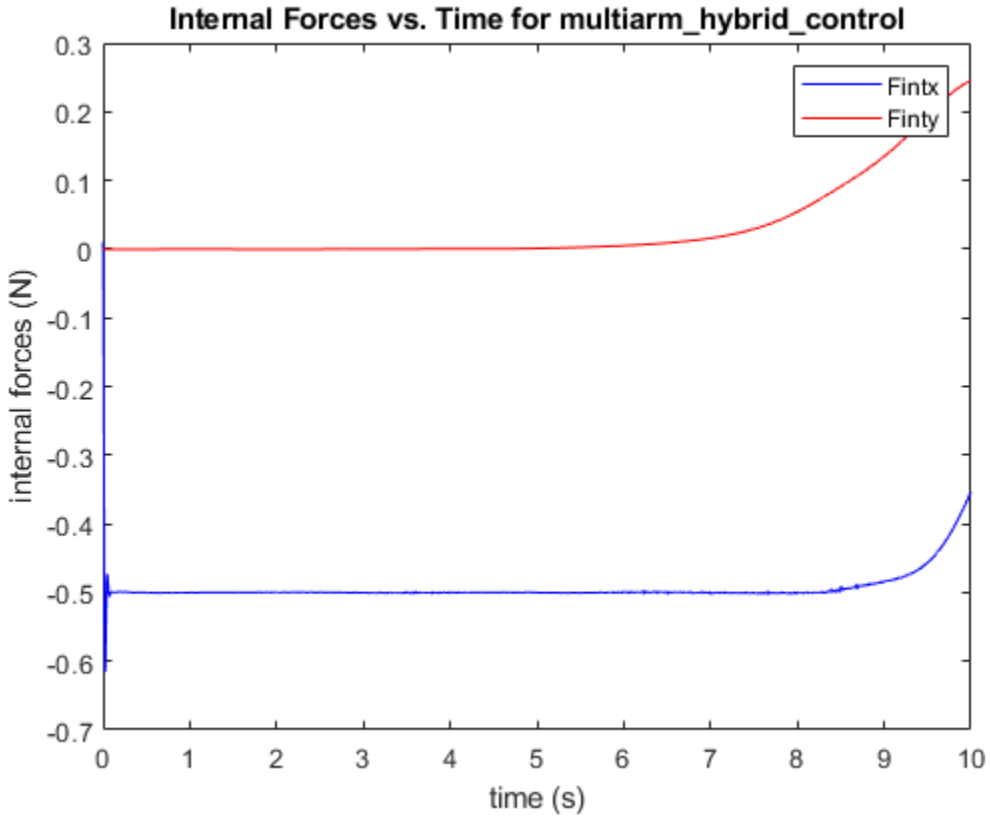
1.2.2.1. Position of Block (Robot 1 Pinned End-Effector)



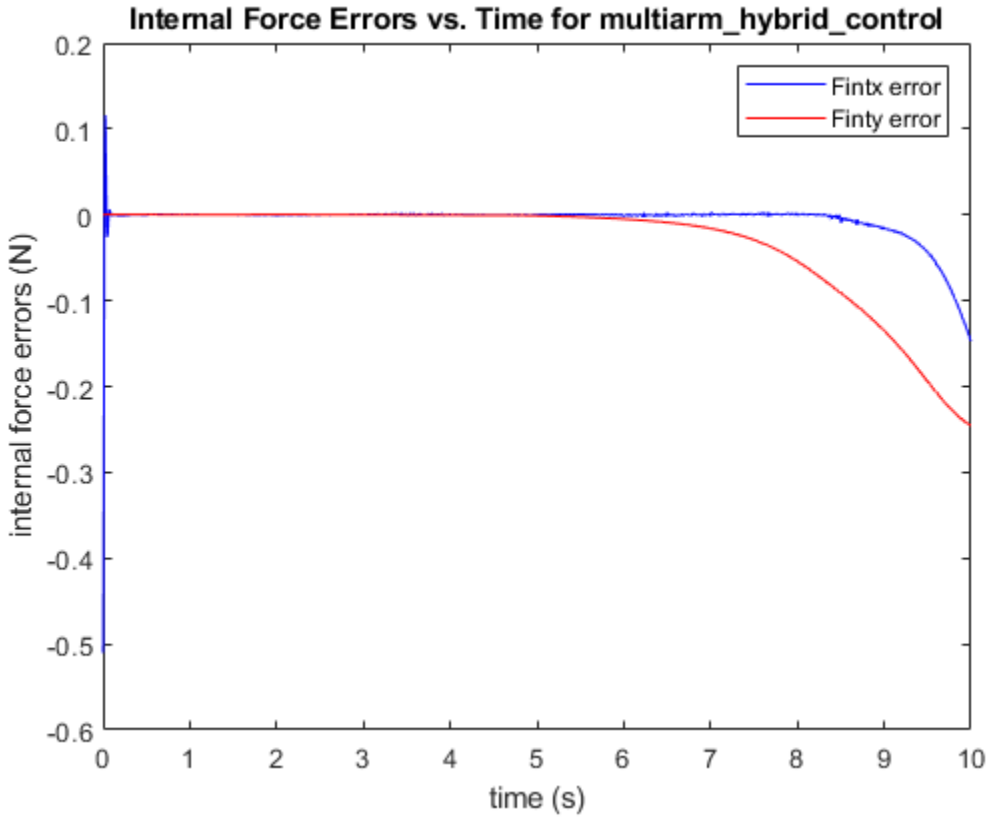
1.2.2.2. Position Errors of Block (Robot 1 Pinned End-Effector)



1.2.2.3. Internal Forces

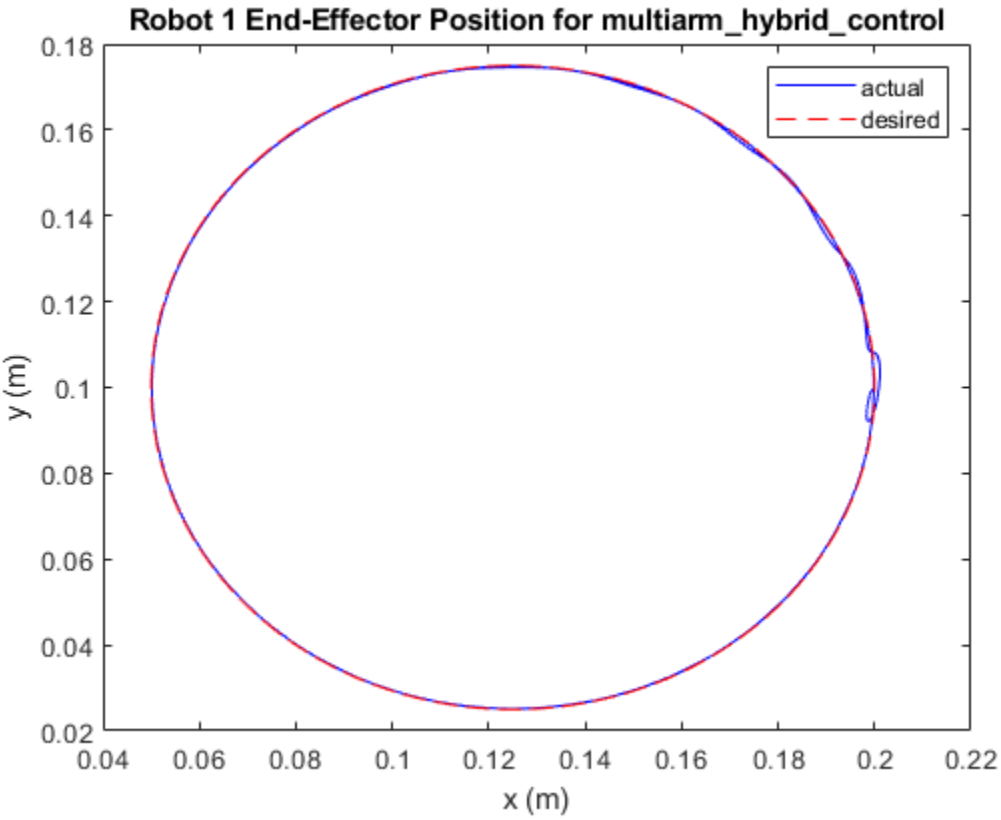


1.2.2.4. Internal Force Errors

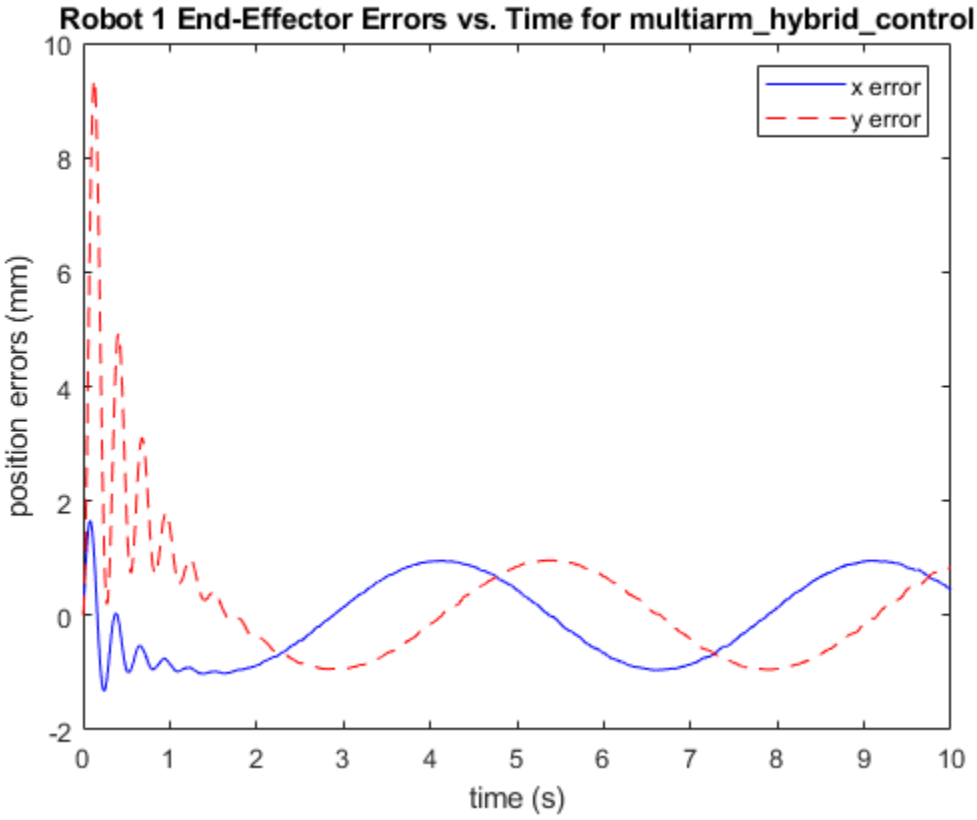


1.2.3. +1 N X-Direction Internal Force and +0.5 N Y-Direction Internal Force

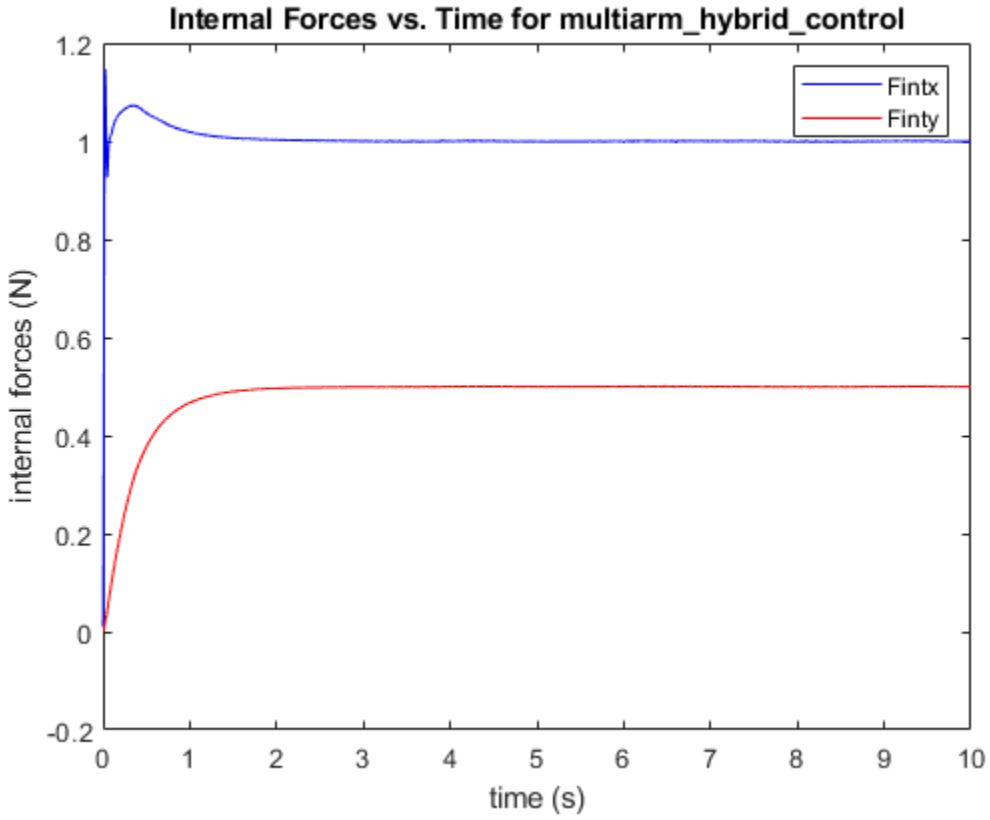
1.2.3.1. Position of Block (Robot 1 Pinned End-Effector)



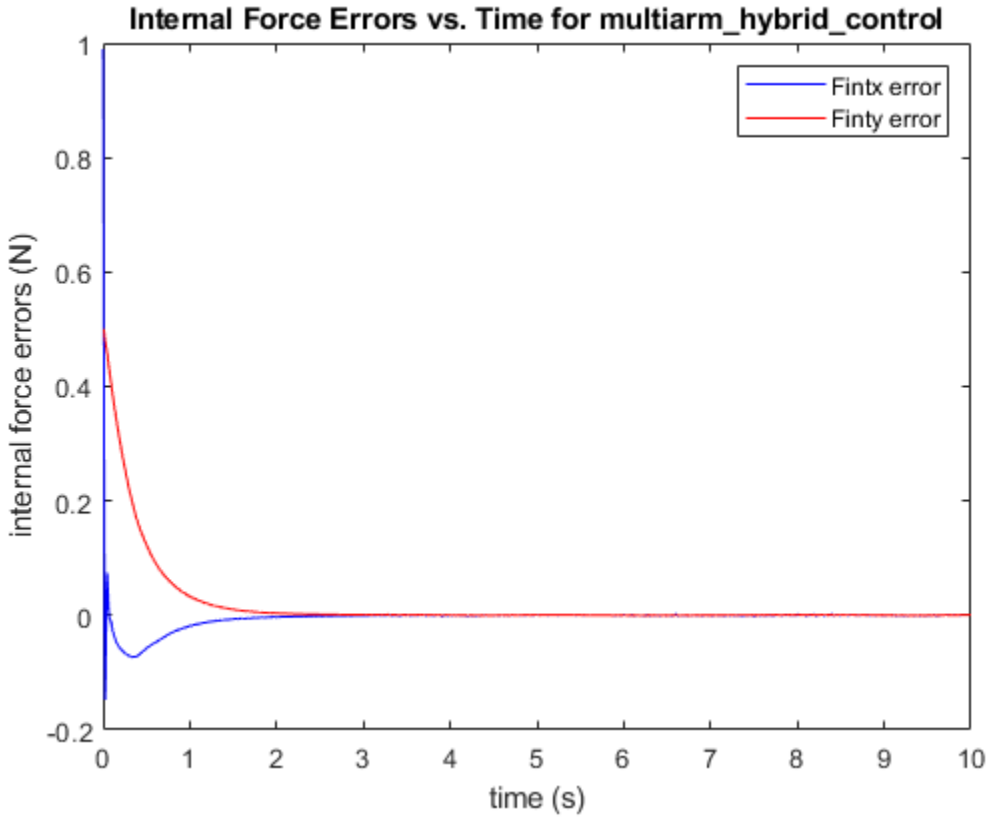
1.2.3.2. Position Errors of Block (Robot 1 Pinned End-Effector)



1.2.3.3. Internal Forces

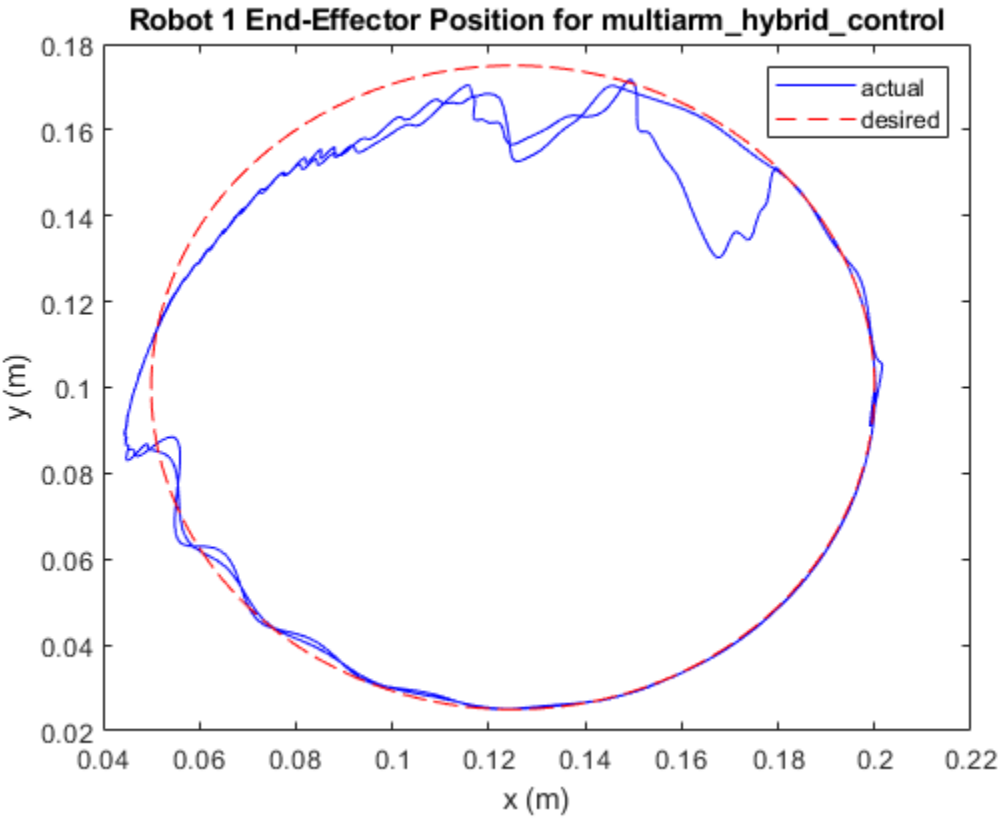


1.2.3.4. Internal Force Errors

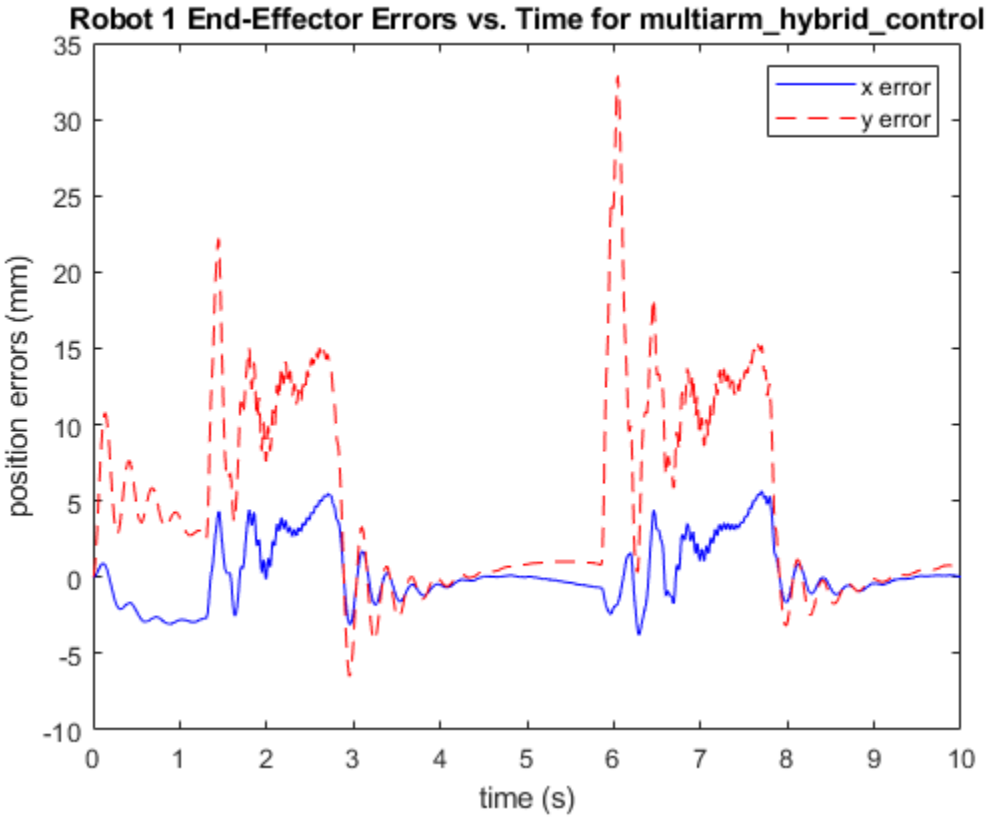


1.2.4. 0 N X-Direction Internal Force and +0.5 N Y-Direction Internal Force

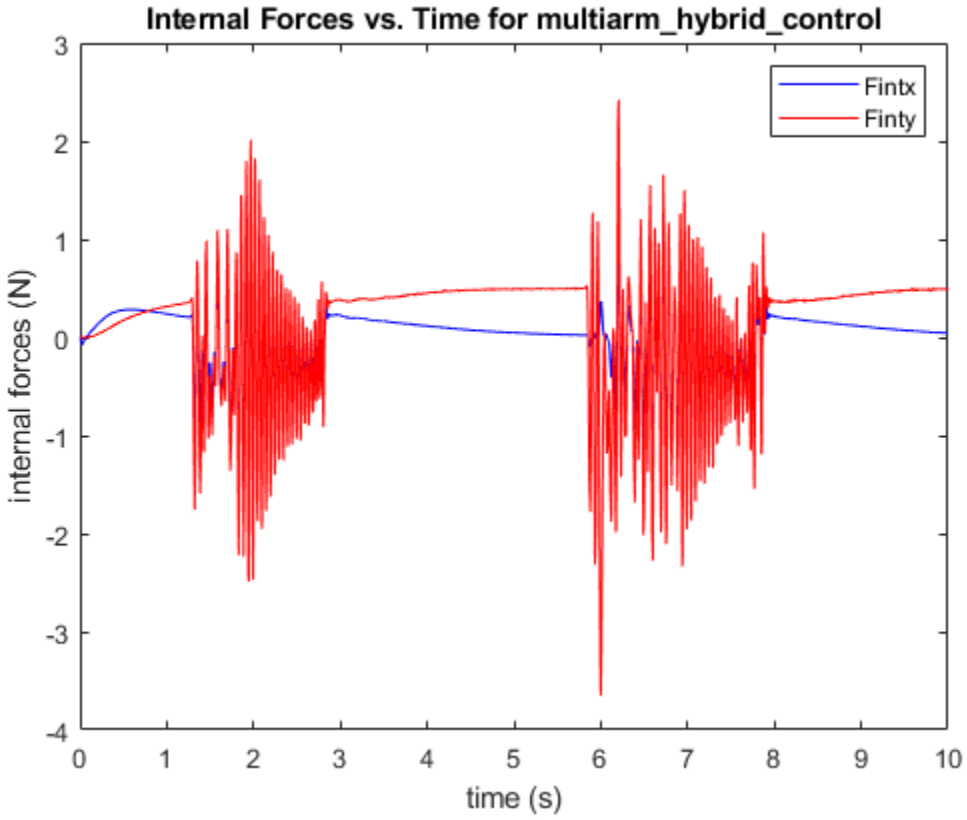
1.2.4.1. Position of Block (Robot 1 Pinned End-Effector)



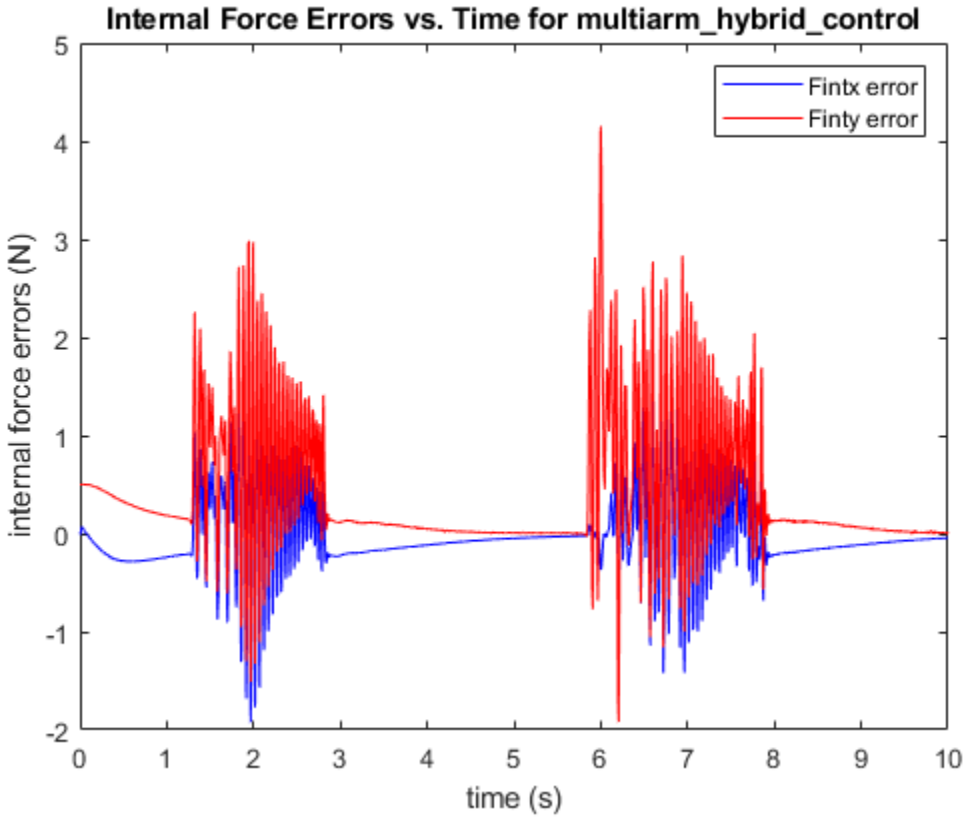
1.2.4.2. Position Errors of Block (Robot 1 Pinned End-Effector)



1.2.4.3. Internal Forces



1.2.4.4. Internal Force Errors



1.2.5. Analysis

Comparing the different desired internal force values there are some obvious limitations to what desired internal forces are possible for the multi-arm hybrid position/force controller.

Setting the desired internal force to a positive x-direction value (implying tension) and no y-direction value results in favorable performance position and force tracking and good stability (this is shown by plots 1.1.1.1-1.1.1.4). This is contrasted by setting the desired internal force to a negative x-direction value (implying compression) and no y-direction value which results in satisfactory tracking if the x-direction value is small (like 0.5 N as shown by 1.2.2) but is inherently unstable as shown by 1.2.2.4 where the force errors begin to diverge and increase rapidly at the end of the 10 second simulation time. If the x-direction values are set to values more negative than -0.5 N then the controller goes unstable and exhibits poor position and force tracking. This shows that keeping the block in compression is very difficult, which makes intuitive sense since for this robot a small error can result in a large moment couple which is not possible when in tension.

Setting the desired internal force to a positive x-direction value and a positive y-direction value results in tilting the block and with a reasonably small y-direction value the tracking and stability are still good as seen in the plots of 1.2.3. The transient response of the position and internal force is not as favorable as having no y-direction value but still results in stable control. If the desired internal force is set to a positive y-direction value with no x-direction value the controller exhibits instability, edge of workspace issues, and toggling behavior near the fully extended position. This behavior is shown by 1.2.4 and the 1.2.4.1 plot shows how at the top left of the circular trajectory the position tracking is poor since robot 2 is fully extended. The force errors are also oscillatory when robot 2 is fully extended which is partly due to the rapid toggling between elbow-up and elbow-down positions of robot 2.

Appendix- Plotting Code (Used to make simulation plots)

```
%% ME EN 6230 Problem Set 11 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 = xy_errors.time; % s
model_title = extractBefore(xy_errors.blockName, "/");
position = xy; % m (Note: xy sequentially contains: x, y, xrel, yrel)
desired_position = xy_d; % m
position_errors = xy_errors.signals.values*1000; % mm
internal_forces = F.signals.values; % N
internal_forces_errors = ForceErrors.signals.values; % N

% Plot Position of Block (Robot 1 Pinned End-Effector)
figure;
plot(position(:,1), position(:,2), 'b-');
hold on;
plot(desired_position(:,1), desired_position(:,2), 'r--');
title(strcat("Robot 1 End-Effector Position for ", model_title));
xlabel("x (m)");
ylabel("y (m)");
legend("actual", "desired");

% Plot Position Errors of Block (Robot 1 Pinned End-Effector)
figure;
plot(time, position_errors(:,1), 'b-');
hold on;
plot(time, position_errors(:,2), 'r--');
title(strcat("Robot 1 End-Effector Errors vs. Time for ", model_title));
xlabel("time (s)");
ylabel("position errors (mm)");
legend("x error", "y error");

% Plot Internal Forces
figure;
plot(time, internal_forces(:,1), 'b-');
hold on;
plot(time, internal_forces(:,2), 'r-');
title(strcat("Internal Forces vs. Time for ", model_title));
xlabel("time (s)");
ylabel("internal forces (N)");
```

```
legend("Fintx", "Finty");

% Plot Internal Force Errors
figure;
plot(time, internal_forces_errors(:,1), 'b-');
hold on;
plot(time, internal_forces_errors(:,2), 'r-');
title(strcat("Internal Force Errors vs. Time for ", model_title));
xlabel("time (s)");
ylabel("internal force errors (N)");
legend("Fintx error", "Finty error");
```