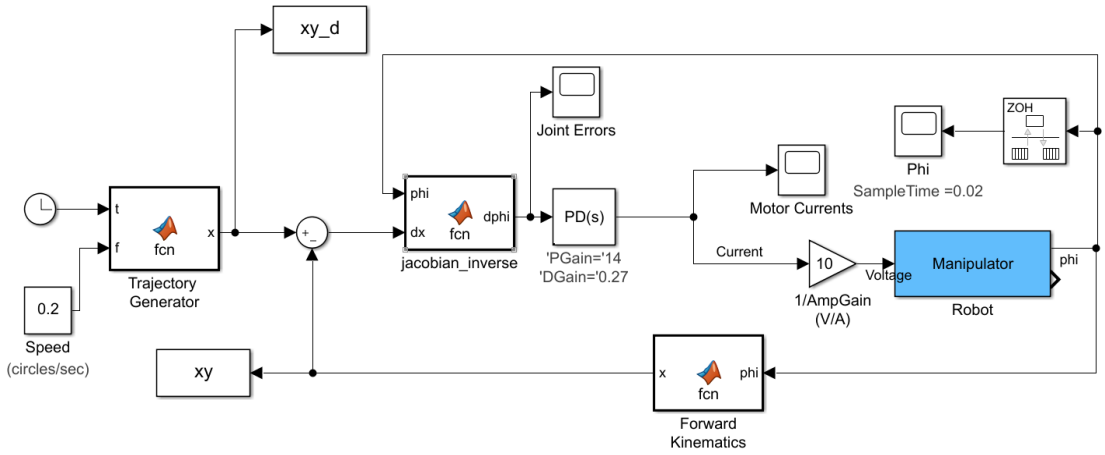
ME EN 6230
Problem Set 8
Ryan Dalby

1. Jacobian Inverse Control

PD gains are Kp = 14 and Kd = 0.27 for jacobian inverse control.

1.1. Model



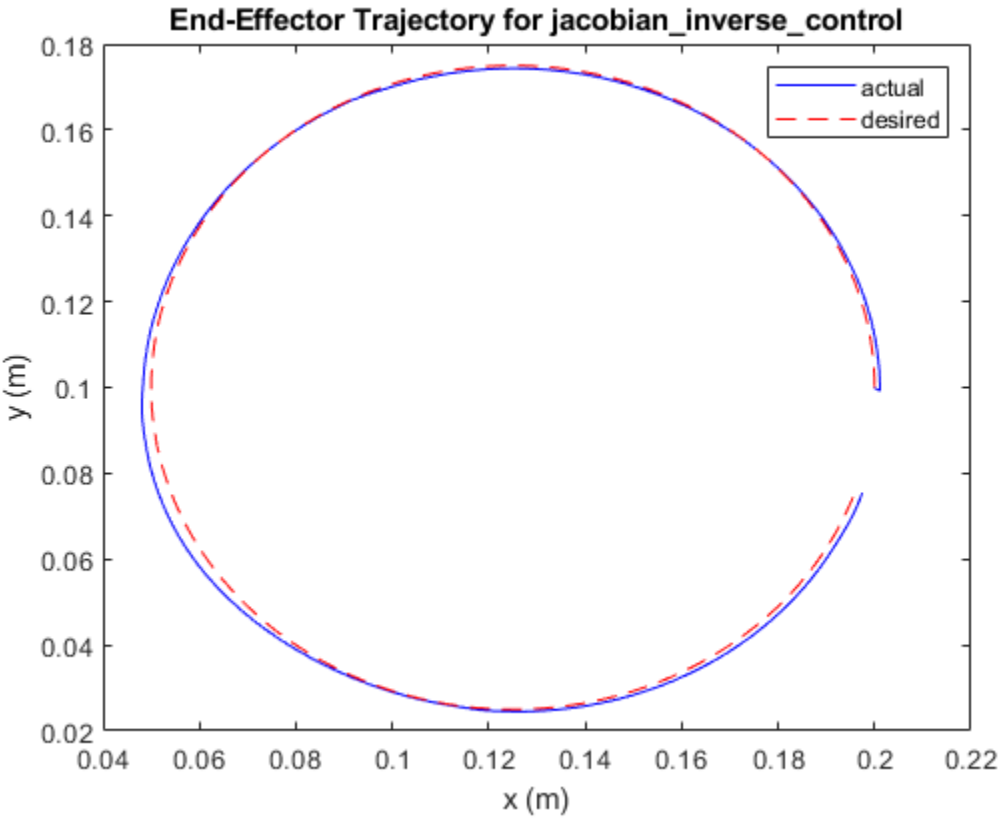
1.2. Code

Code for jacobian_inverse block:

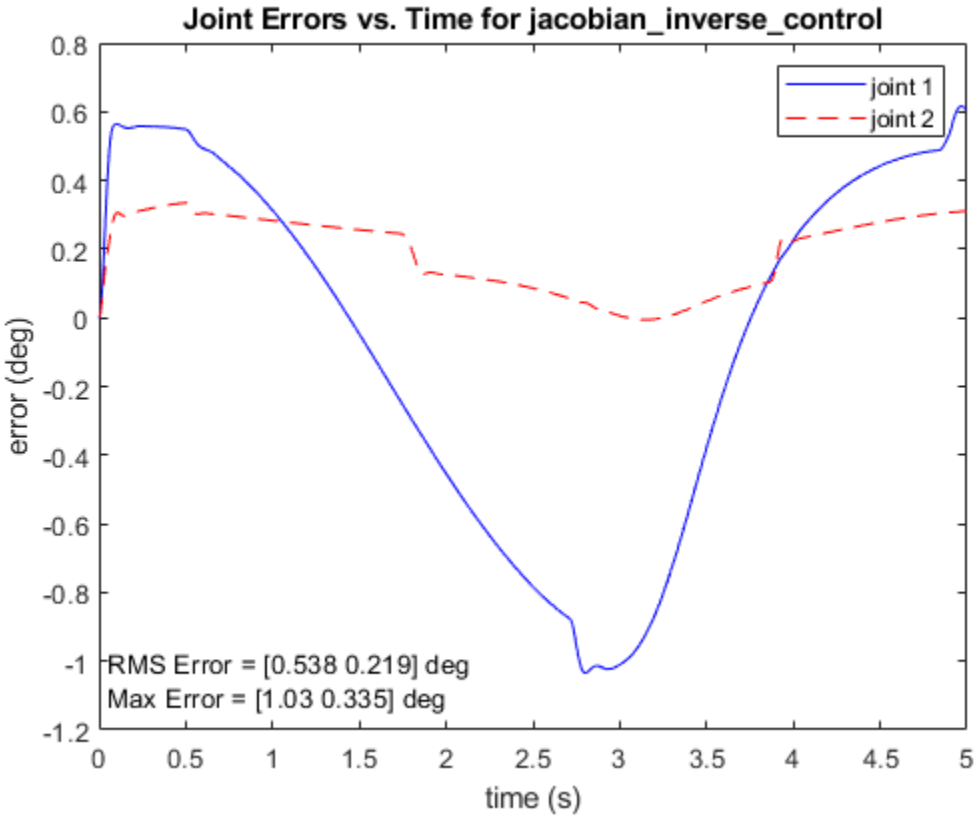
```
function dphi = fcn(phi, dx)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

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

% Describe combined relationship of jacobian from phi to x using jacobian
% (Combines transmission jacobian from phi to theta and manipulator
% jacobian from theta to x)
J11 = -a1*sin(phi(1));
J12 = -a2*sin(phi(2));
J21 = a1*cos(phi(1));
J22 = a2*cos(phi(2));
J = [J11 J12; J21 J22];

dphi = inv(J)*dx;
```

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

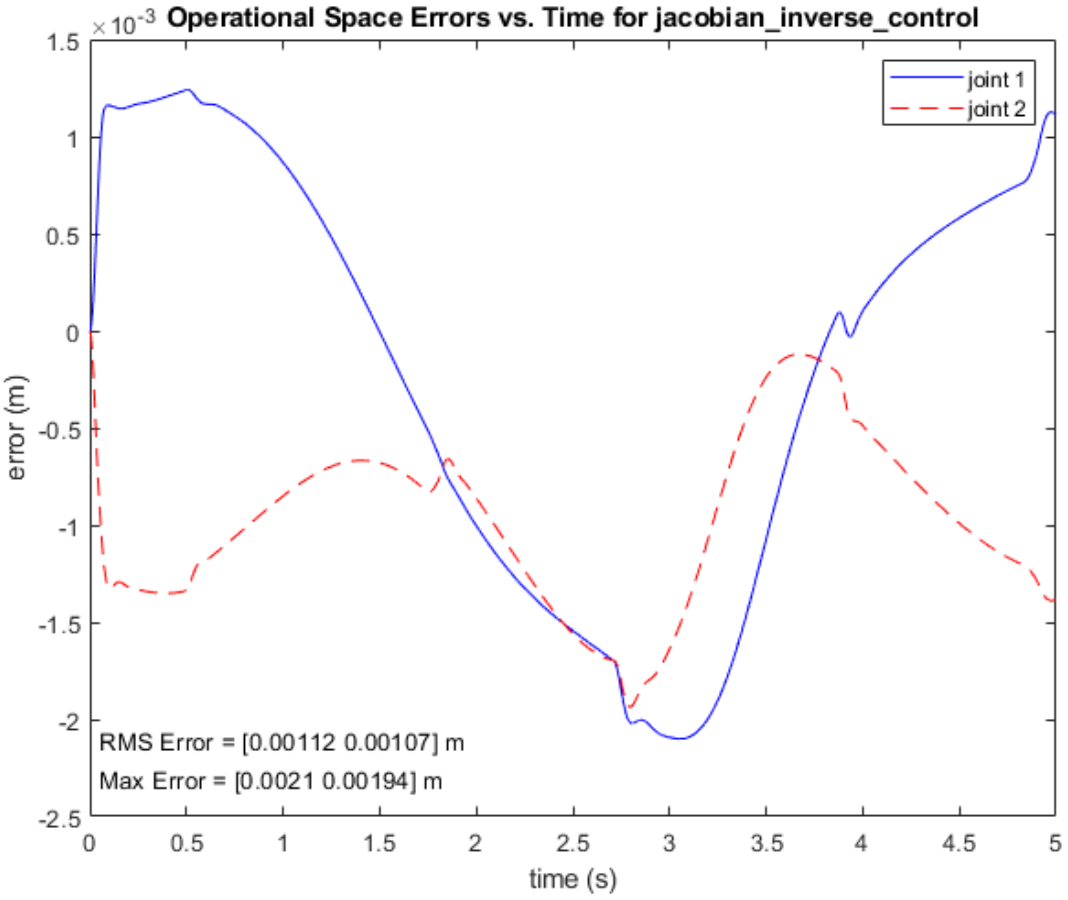
1.3.1. X-Y Trajectory



1.3.2. Joint Angle Errors

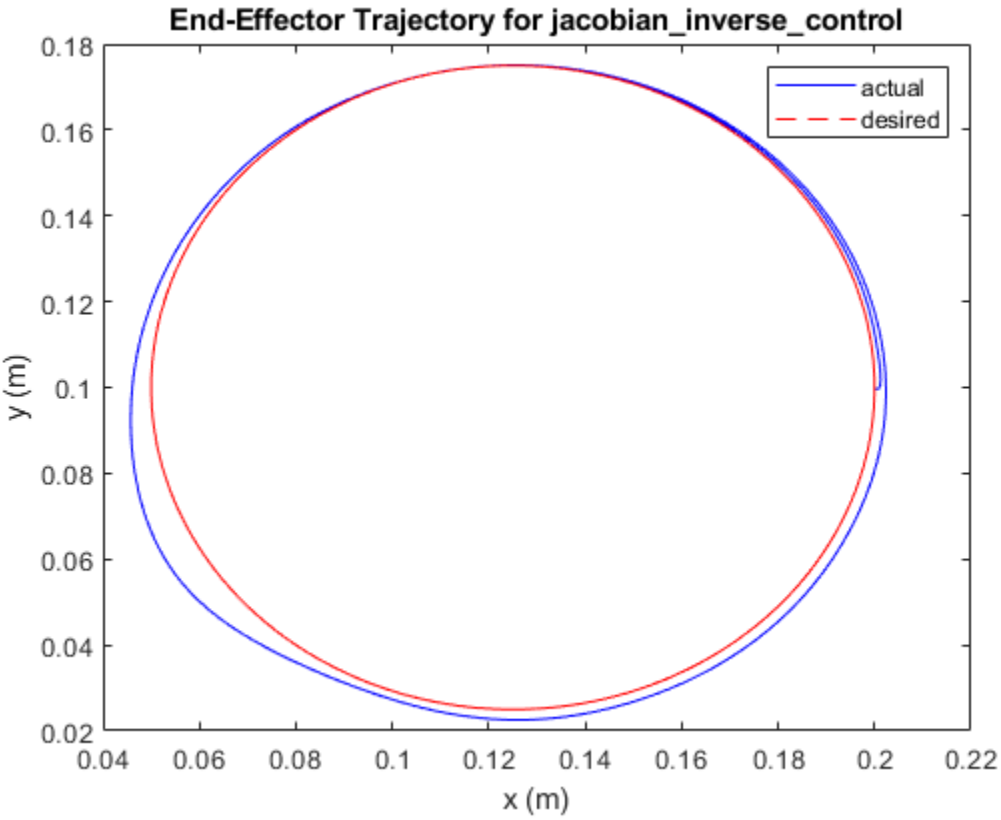


1.3.3. Operational Space Errors

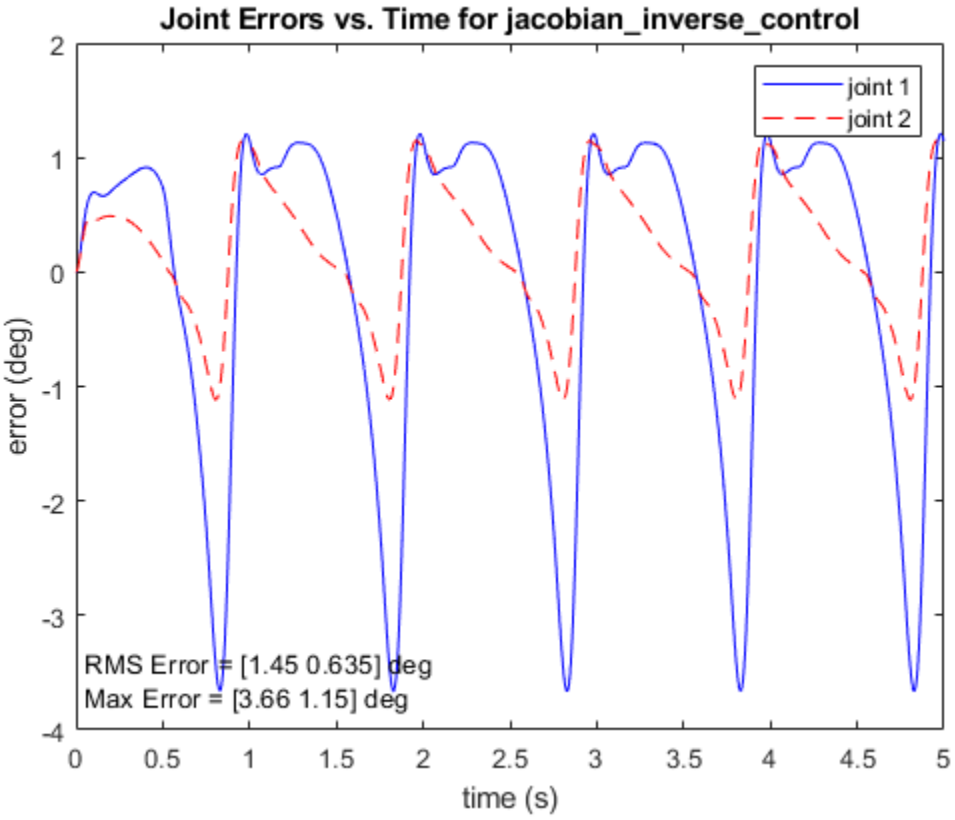


1.4. High Speed Simulation (f=1 circles/s)

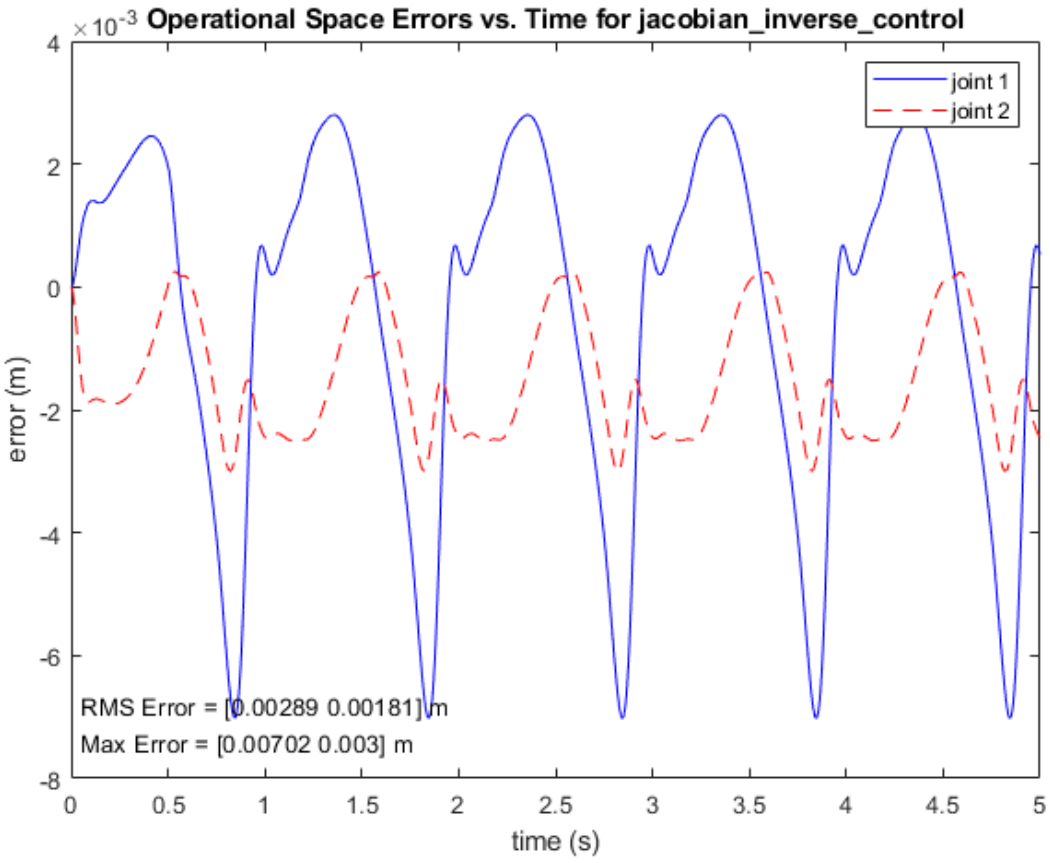
1.4.1. X-Y Trajectory



1.4.2. Joint Angle Errors

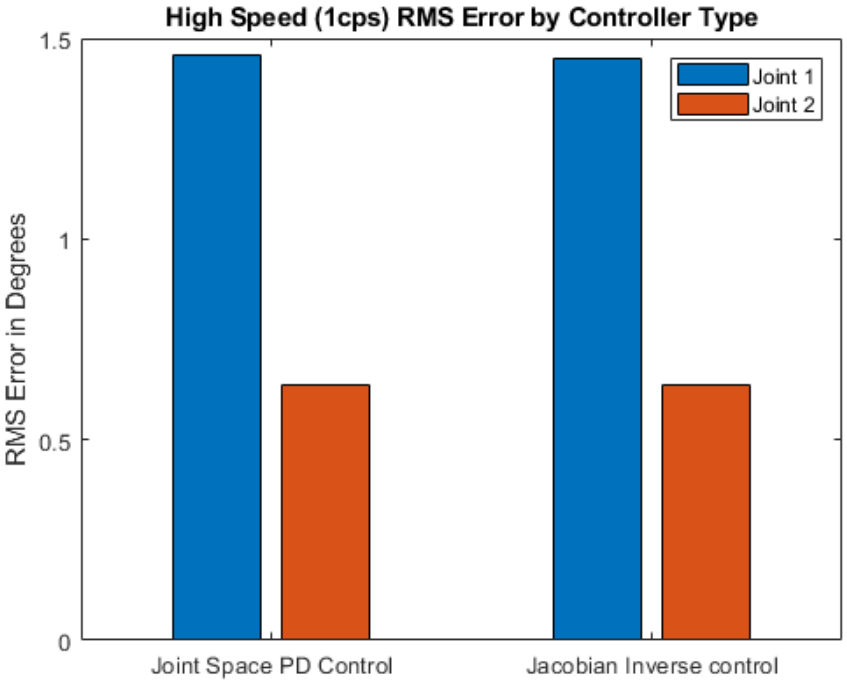


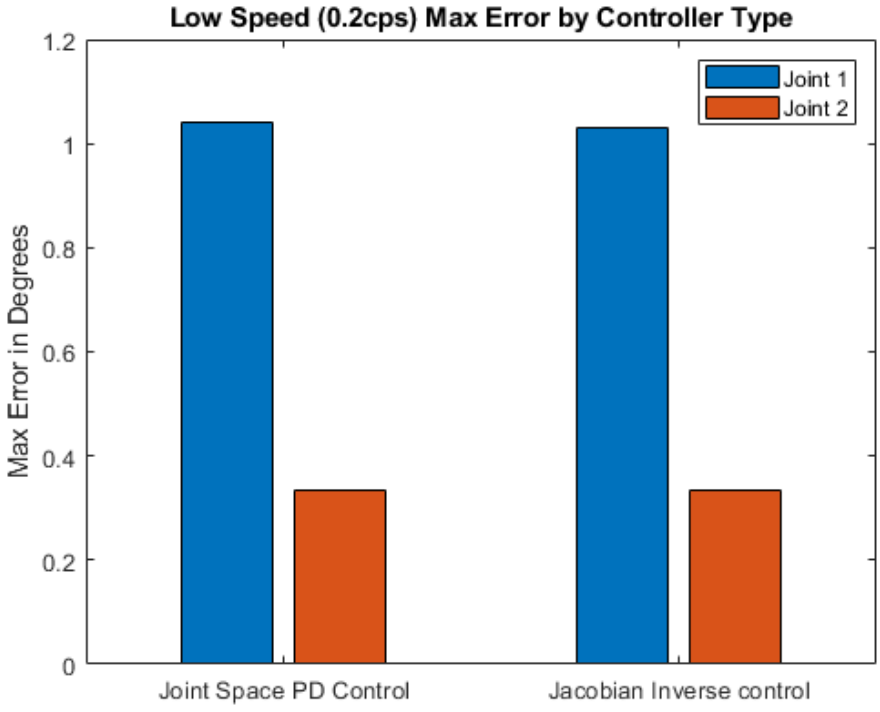
1.4.3. Operational Space Errors



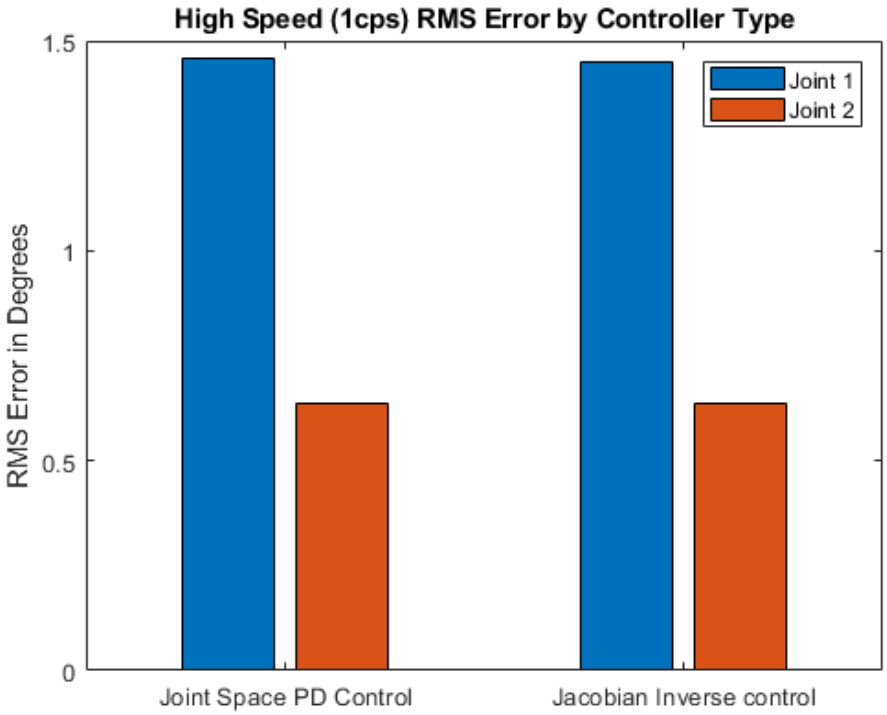
1.5. Controller Comparison

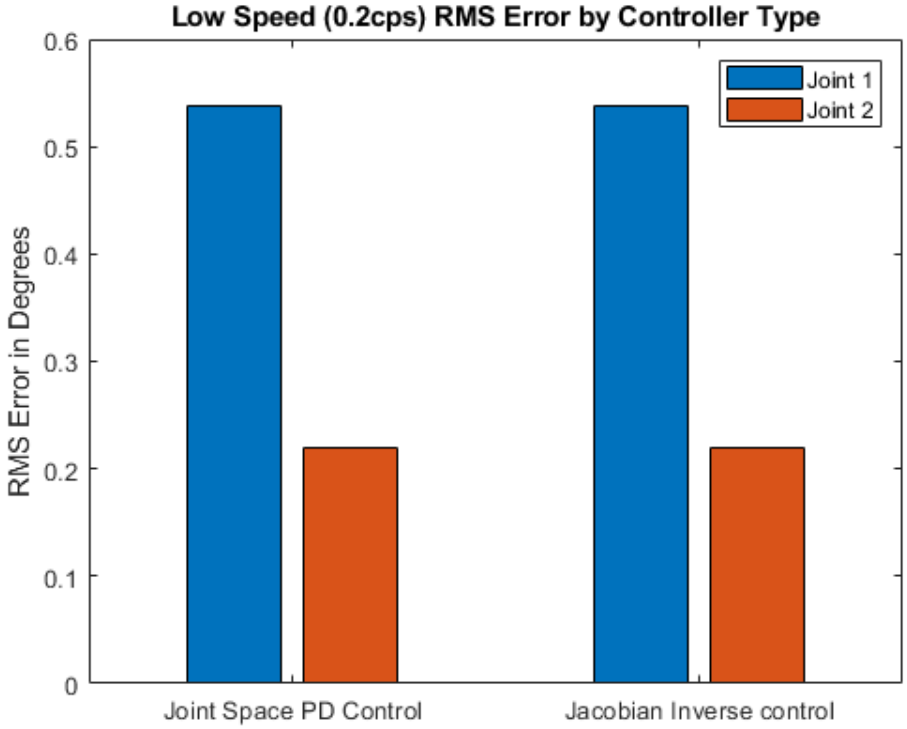
1.5.1. Peak Joint Errors





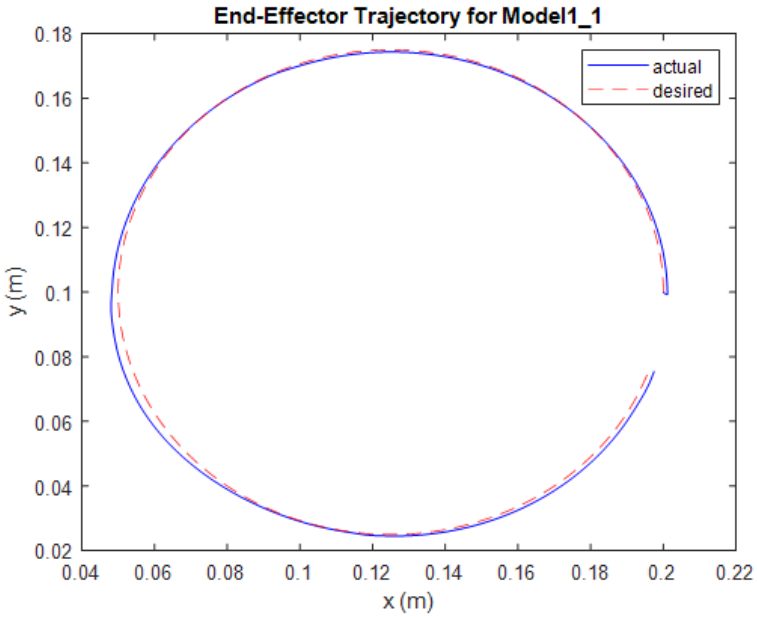
1.5.2. Root-Mean-Square Joint Errors



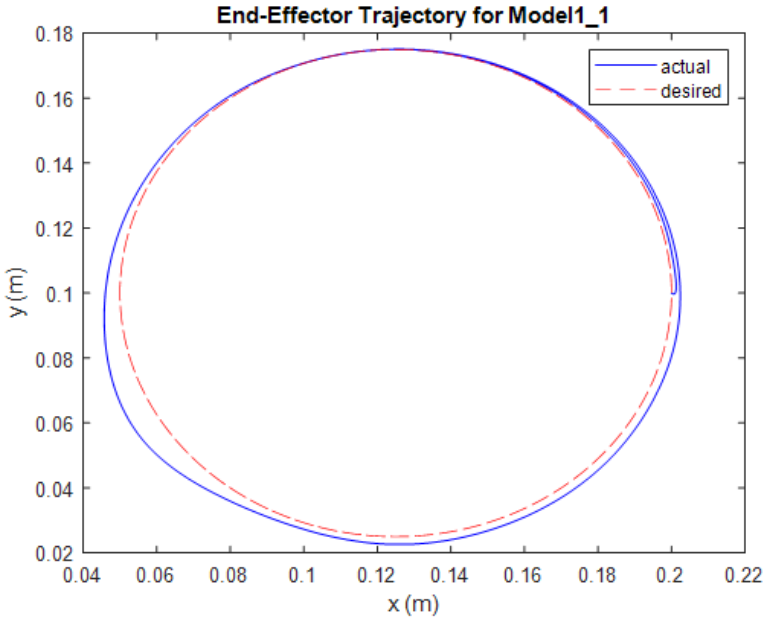


1.5.3. Trajectories

1.5.3.1. PD Control Low Speed Simulation (f=0.2 circles/s) (Compare to 1.3.1.)



1.5.3.2. PD Control High Speed Simulation (f=1 circles/s) (Compare to 1.4.1.)



1.5.4. Comparison

Comparing jacobian inverse control to joint space PD control with the same PD gains the simulation results are virtually identical. As can be seen in the trajectory plots in 1.5.3 the trajectories appear to be virtually equivalent with the same sort of trajectory. As can be seen in 1.5.1 and 1.5.2, both the RMS and max errors are also virtually the same.

1.5.5. Comparison Code

```
%% ME EN 6230 Problem Set 8 Ryan Dalby
% Jacobian Inverse and Joint Space PD Controller Comparison
% close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from
becoming subscript

% errors are in degrees
% Joint Space PD Controller
pd_slow_rms = [0.538 0.219];
pd_slow_max = [1.04 0.335];
pd_fast_rms = [1.46 0.637];
pd_fast_max = [3.67 1.17];

% Jacobian Inverse Controller
feedbackcomp_slow_rms = [0.538 0.219];
feedbackcomp_slow_max = [1.03 0.335];
feedbackcomp_fast_rms = [1.45 0.635];
feedbackcomp_fast_max = [3.66 1.15];


slow_rms = [pd_slow_rms; feedbackcomp_slow_rms];
fast_rms = [pd_fast_rms; feedbackcomp_fast_rms];
slow_max = [pd_slow_max; feedbackcomp_slow_max];
fast_max = [pd_fast_max; feedbackcomp_fast_max];

controller_labels = {'Joint Space PD Control', 'Jacobian Inverse 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');
```

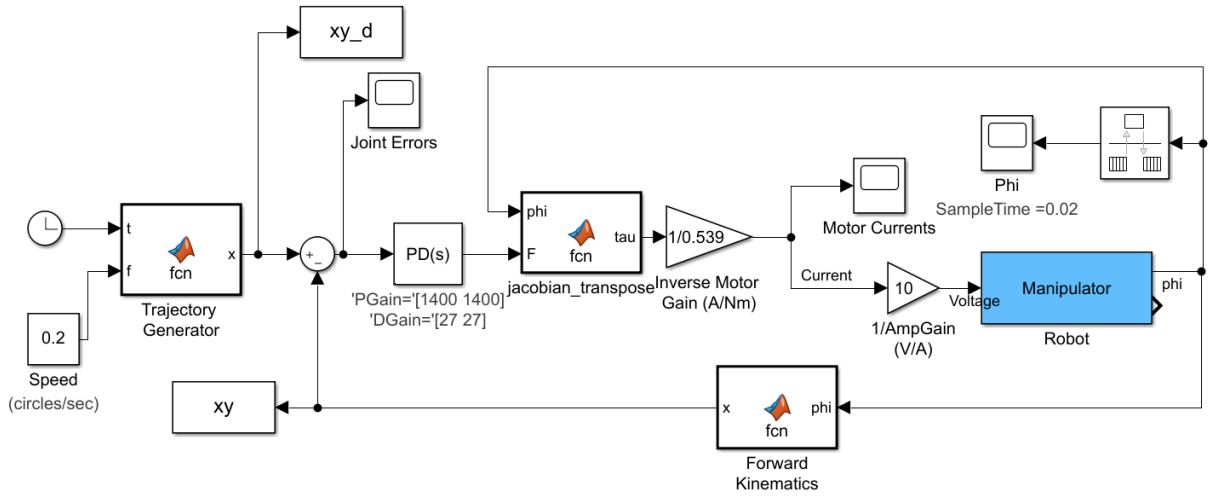
```
figure;
bar(controller_labels_cat,slow_max);
title('Low Speed (0.2cps) Max Error by Controller Type');
legend('Joint 1', 'Joint 2');
ylabel('Max Error in Degrees');

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. Jacobian Transpose Related Control

PD gains are Kp = 1400 and Kd = 27 for all jacobian transpose related control

2.1. Jacobian Transpose Control

2.1.1. Model



2.1.2. Code

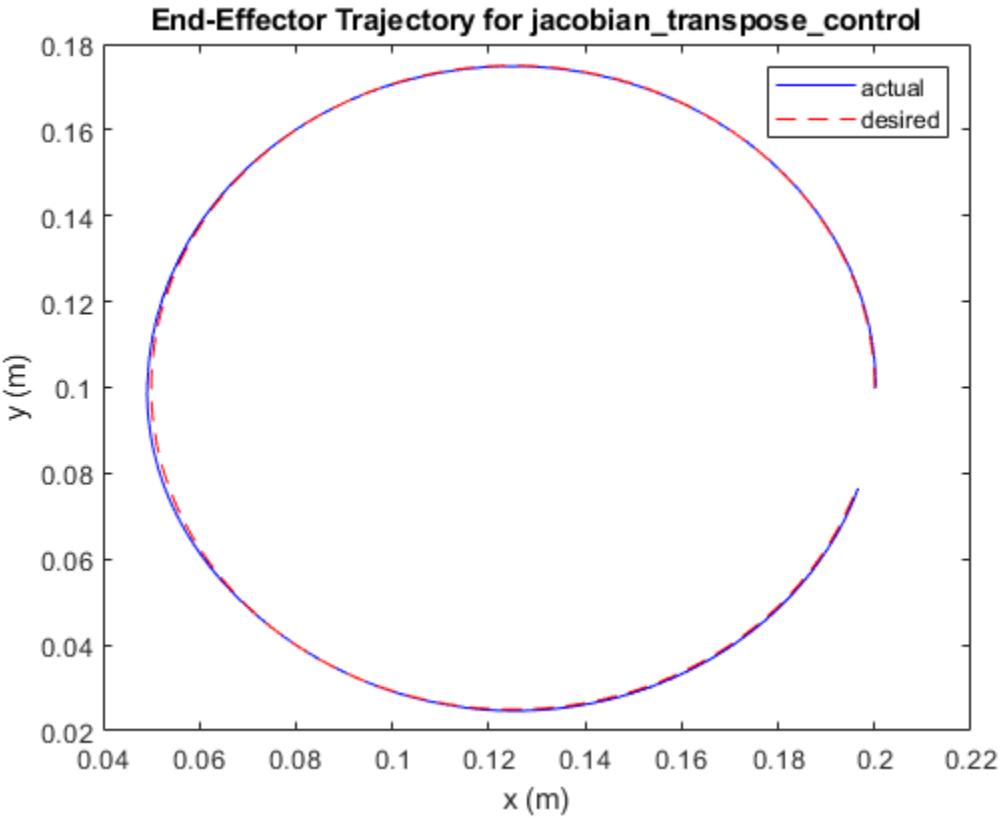
Code for the jacobian_transpose block (also used for inverse dynamics control in operational space and robust control in operational space):

```
function tau = fcn(phi, F)
% 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

% Describe combined relationship of jacobian from phi to x using jacobian
% (Combines transmission jacobian from phi to theta and manipulator
% jacobian from theta to x)
J11 = -a1*sin(phi(1));
J12 = -a2*sin(phi(2));
J21 = a1*cos(phi(1));
J22 = a2*cos(phi(2));
J = [J11 J12; J21 J22];

tau = transpose(J)*F;
```

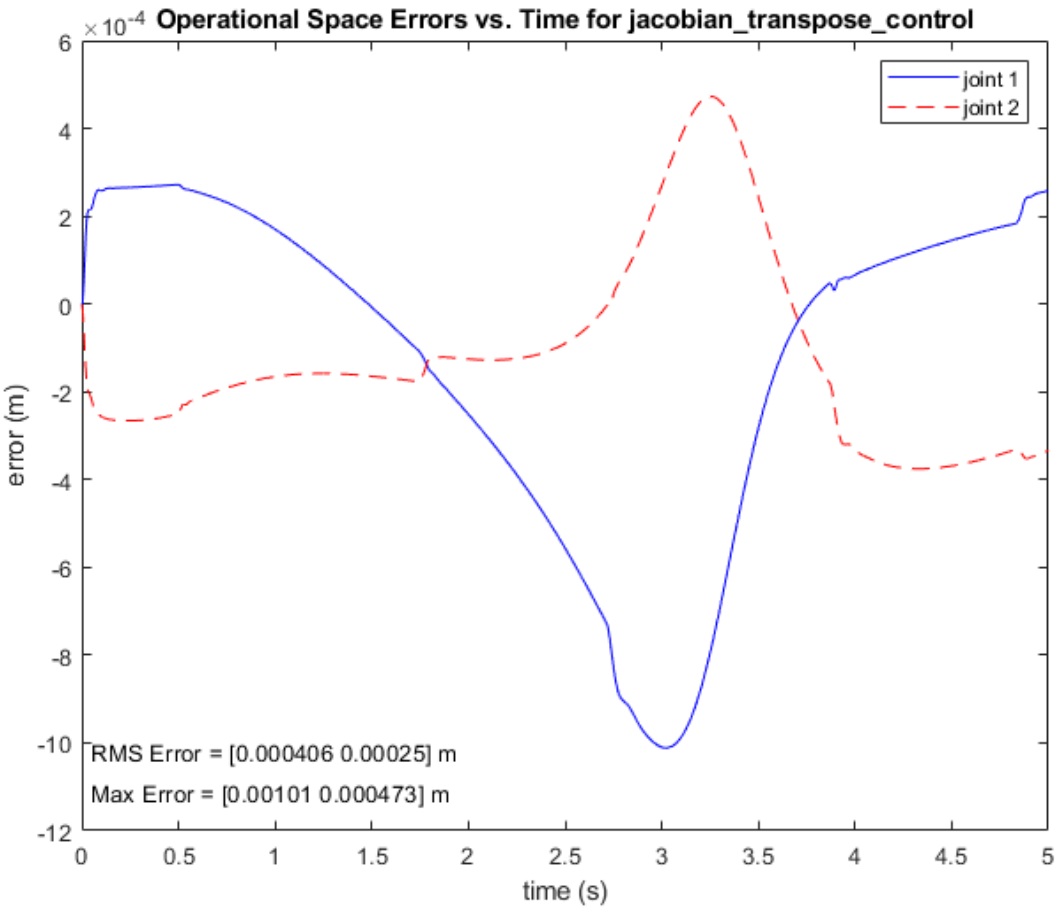
]

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

2.1.3.1. X-Y Trajectory

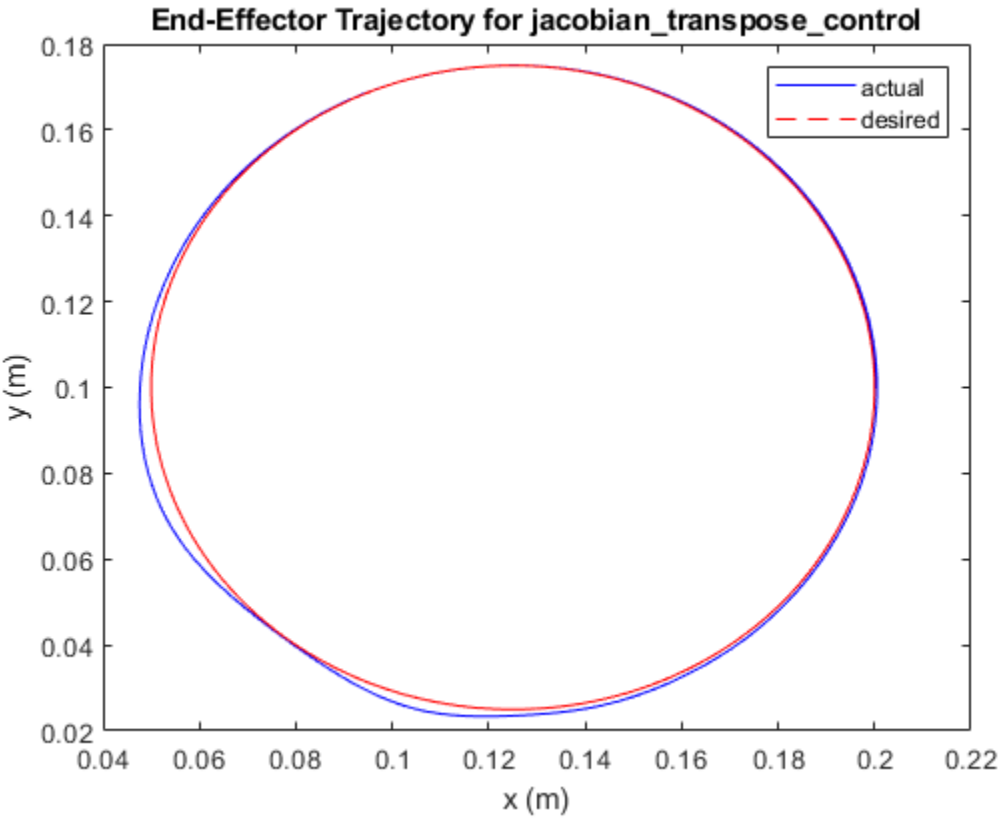


2.1.3.2. Operational Space Errors

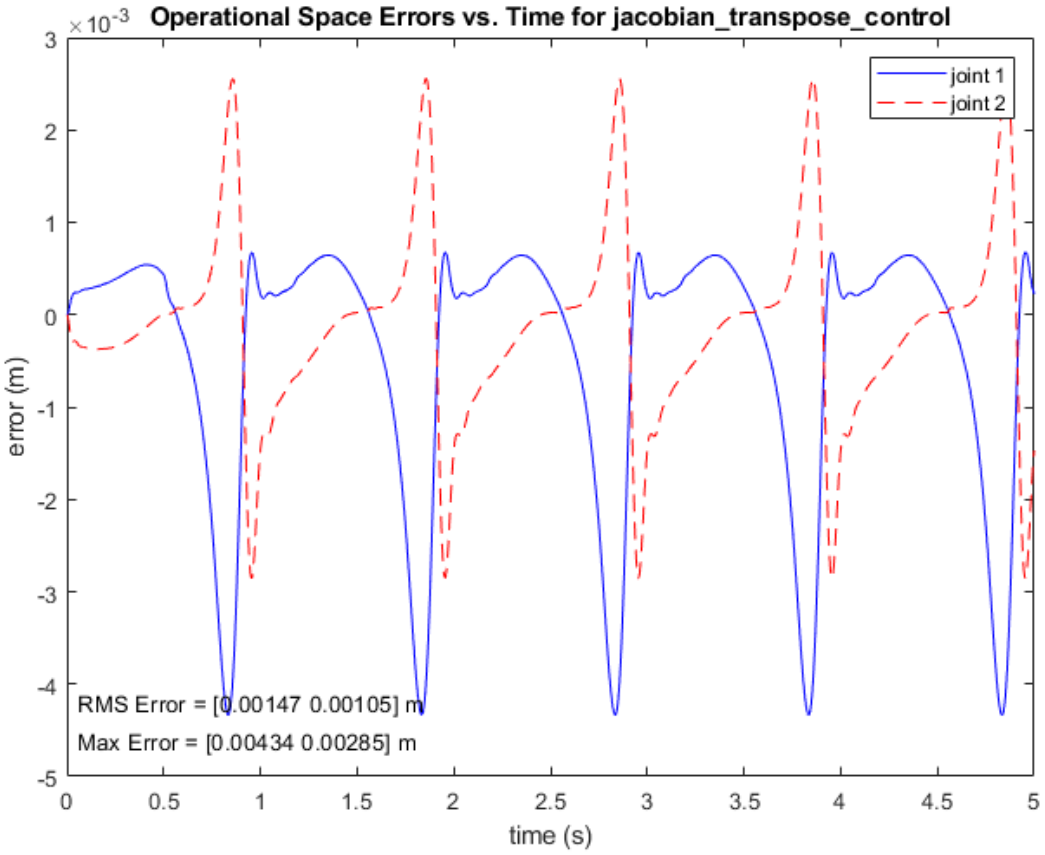


2.1.4. High Speed Simulation (f=1 circles/s)

2.1.4.1. X-Y Trajectory

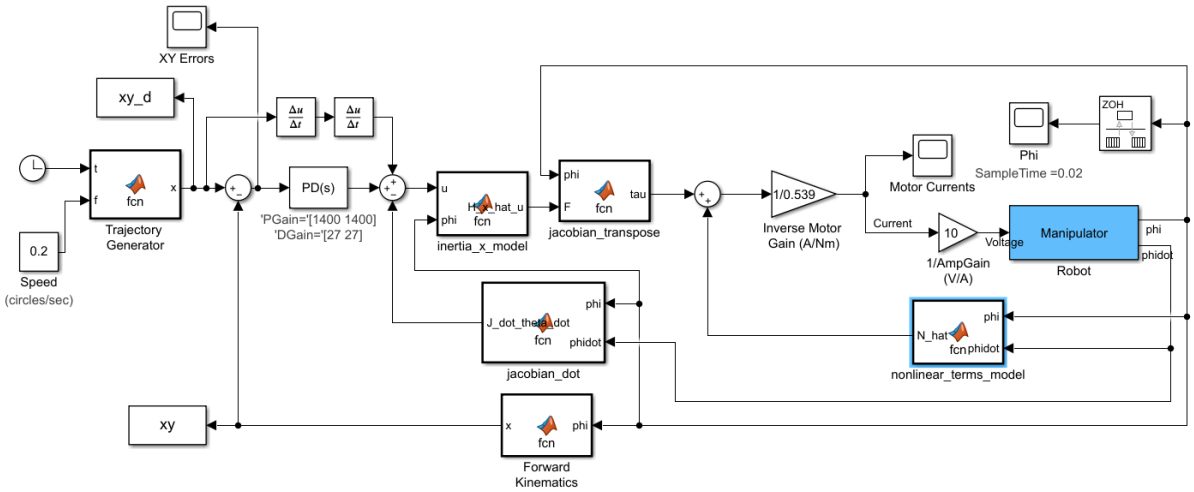


2.1.4.2. Operational Space Errors



2.2. Inverse Dynamics Control in Operational Space

2.2.1. Model



2.2.2. Code

Code for the jacobian_dot block:

```
function J_dot_theta_dot = 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

J11_dot = -a1*cos(phi(1))*phidot(1);
J12_dot = -a2*cos(phi(2))*phidot(2);
J21_dot = -a1*sin(phi(1))*phidot(1);
J22_dot = -a2*sin(phi(2))*phidot(2);

J_dot = [J11_dot J12_dot; J21_dot J22_dot];

J_dot_theta_dot = J_dot * phidot;
```

Code for the inertia_x_model block:

```
function H_x_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;

a1 = 0.15;  % link 1 length
```

```
a2 = 0.15;  % link 2 length
% Describe combined relationship of jacobian from phi to x using jacobian
% (Combines transmission jacobian from phi to theta and manipulator
% jacobian from theta to x)
J11 = -a1*sin(phi(1));
J12 = -a2*sin(phi(2));
J21 = a1*cos(phi(1));
J22 = a2*cos(phi(2));
J = [J11 J12; J21 J22];

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_x_hat = inv(transpose(J)) * H_hat * inv(J);

H_x_hat_u = H_x_hat * u;
```

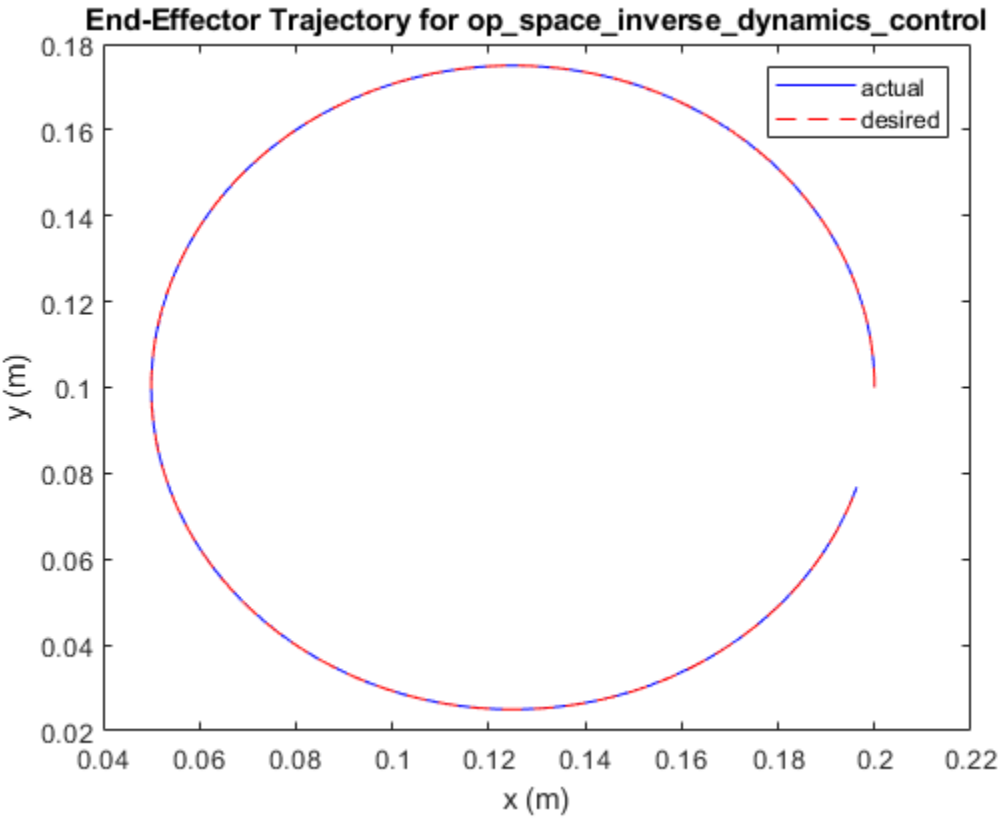
Code for nonlinear_terms_model block:

```
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
```

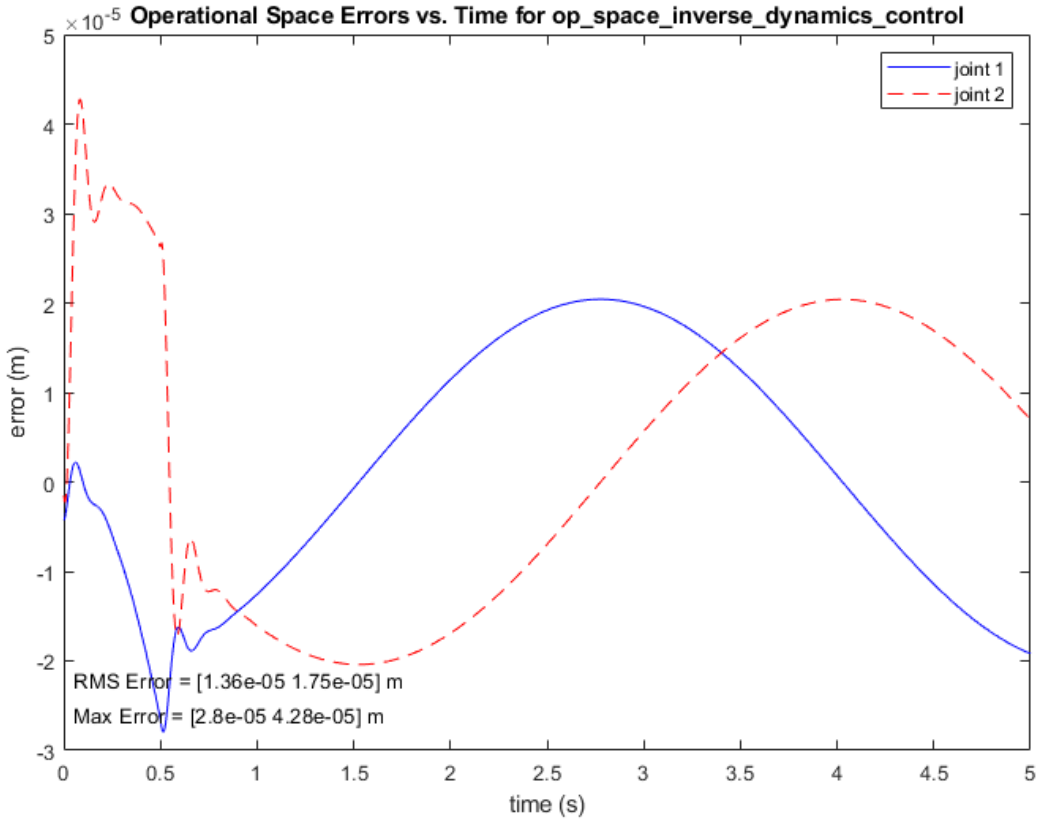
```
N2 = 70;

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;
```

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

2.2.3.1. X-Y Trajectory

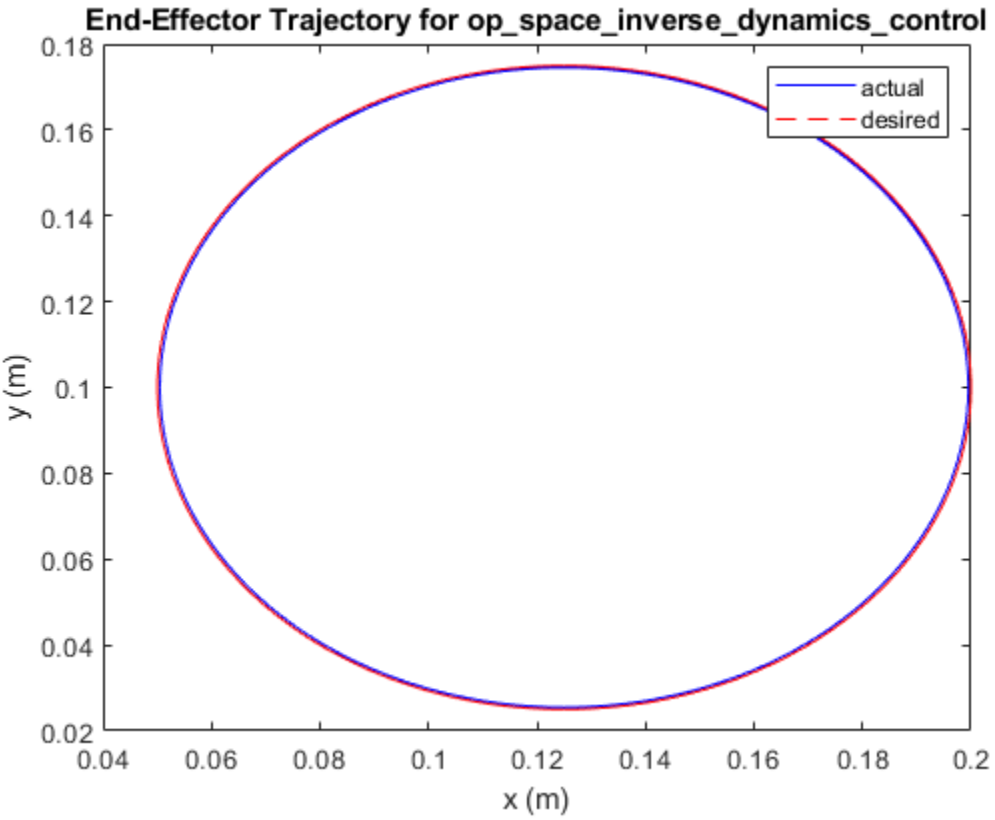


2.2.3.2. Operational Space Errors

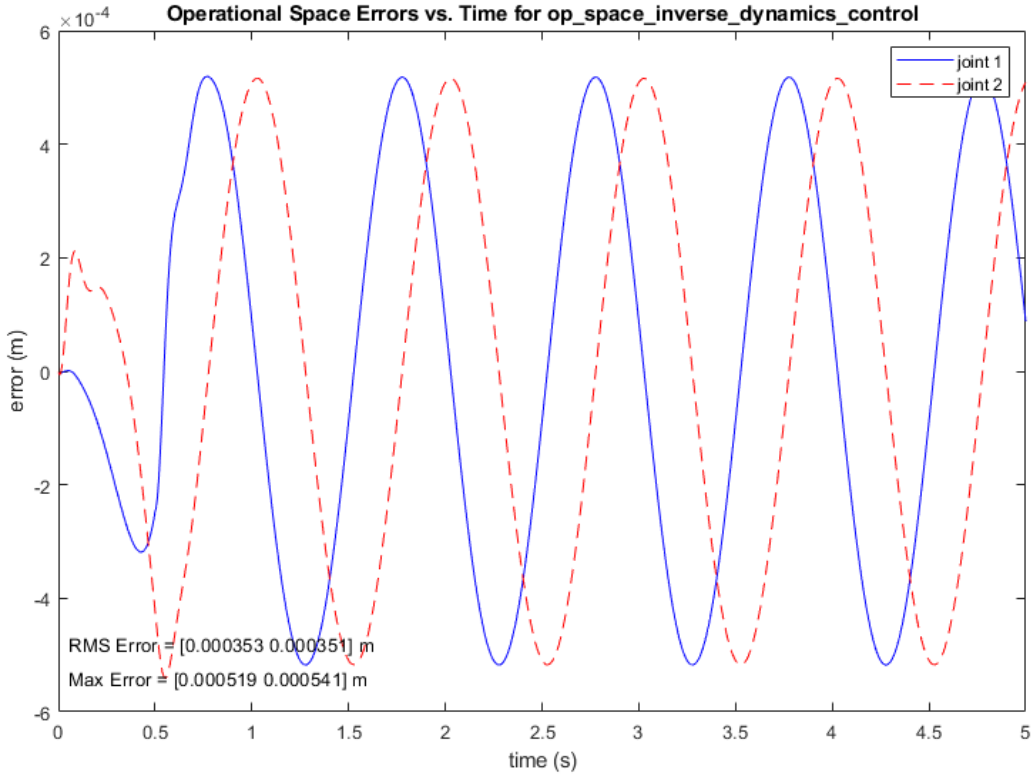


2.2.4. High Speed Simulation (f=1 circles/s)

2.2.4.1. X-Y Trajectory

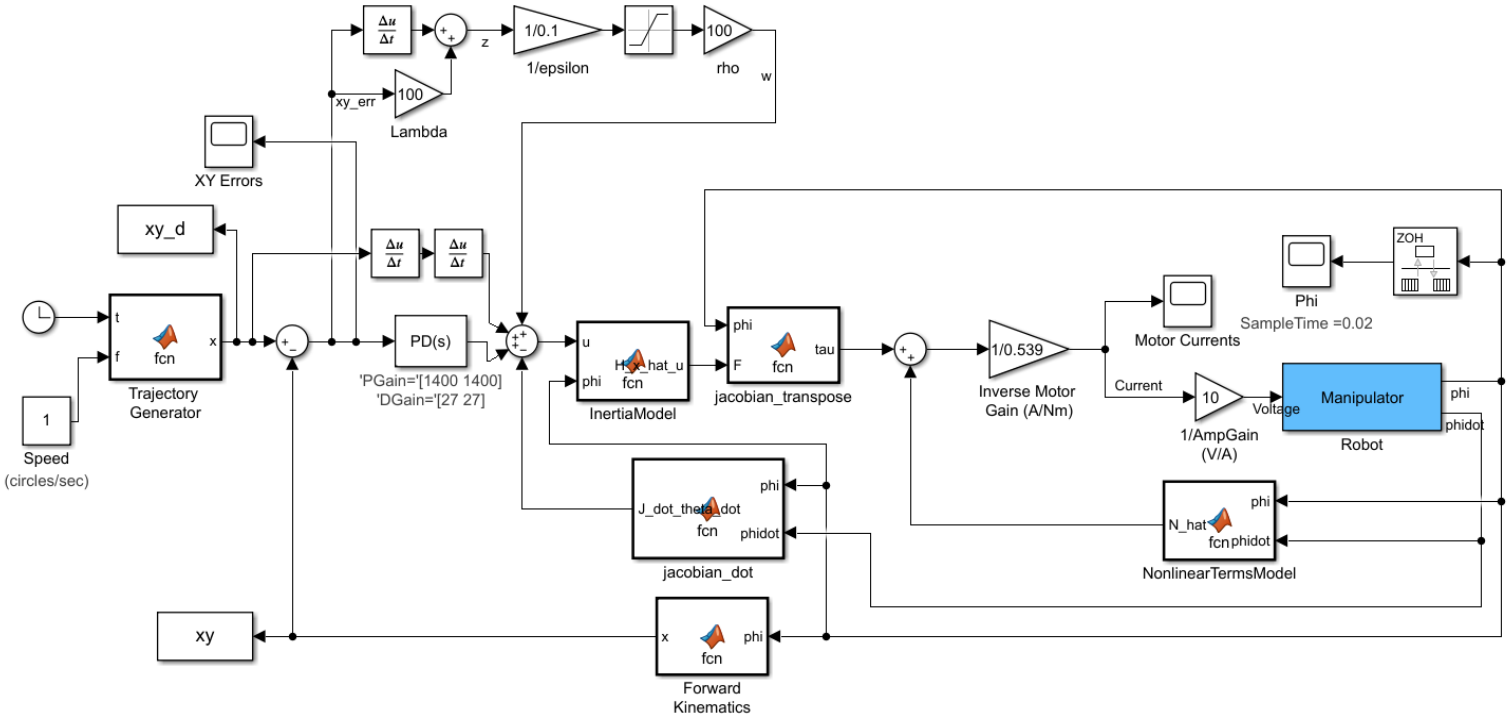


2.2.4.2. Operational Space Errors



2.3. Robust Control in Operational Space

2.3.1. Model

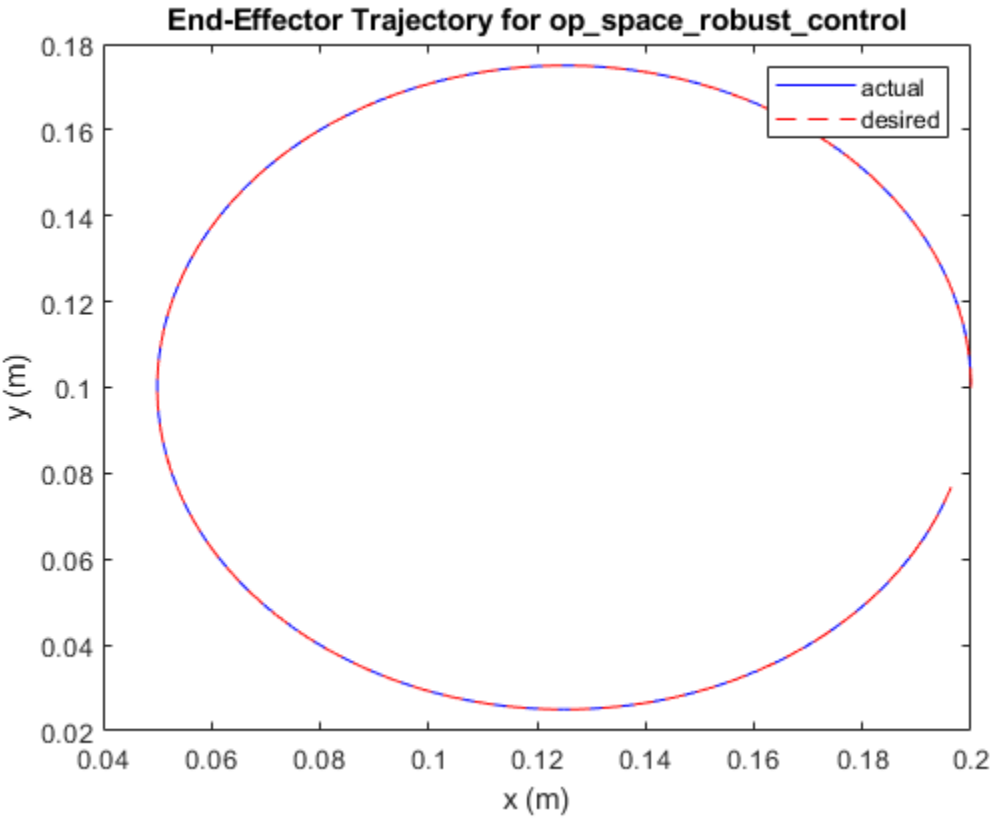


2.3.2. Code

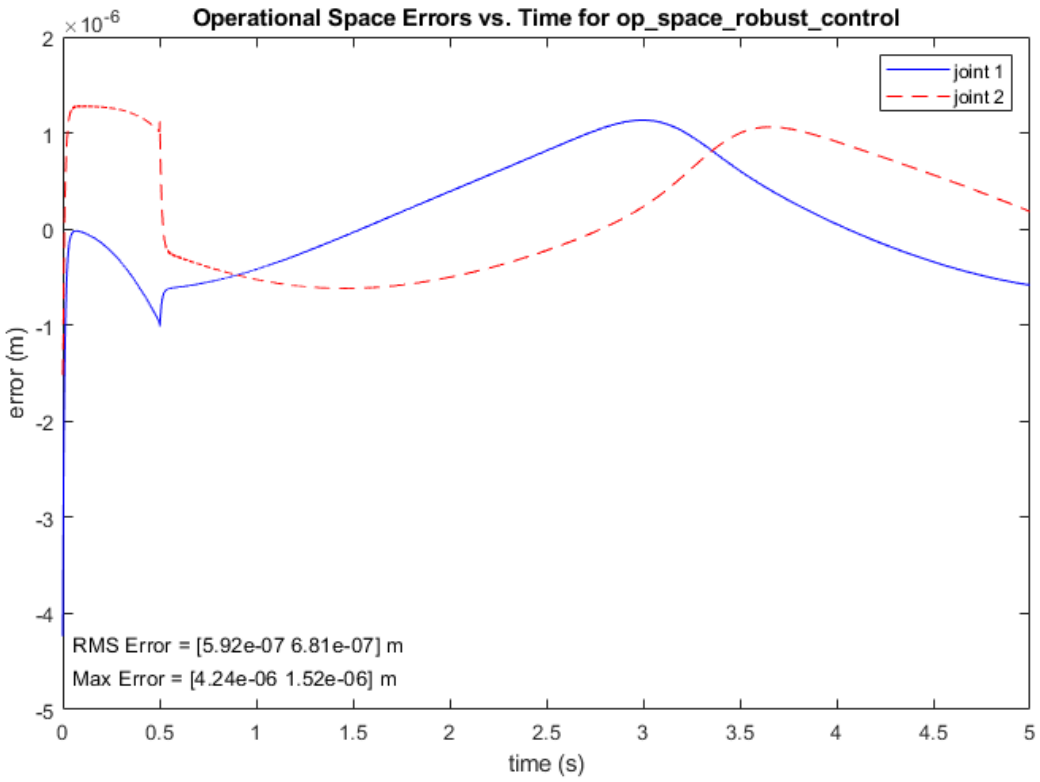
There are no new MATLAB function blocks that have not been described previously..

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

2.3.3.1. X-Y Trajectory

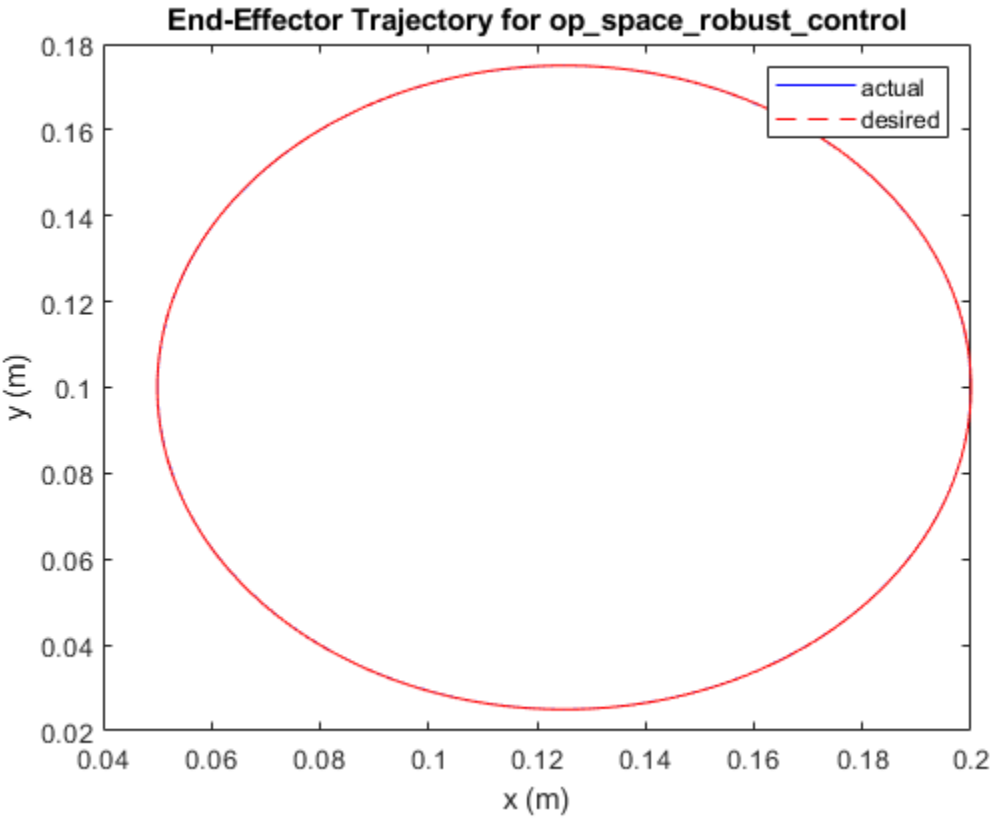


2.3.3.2. Operational Space Errors

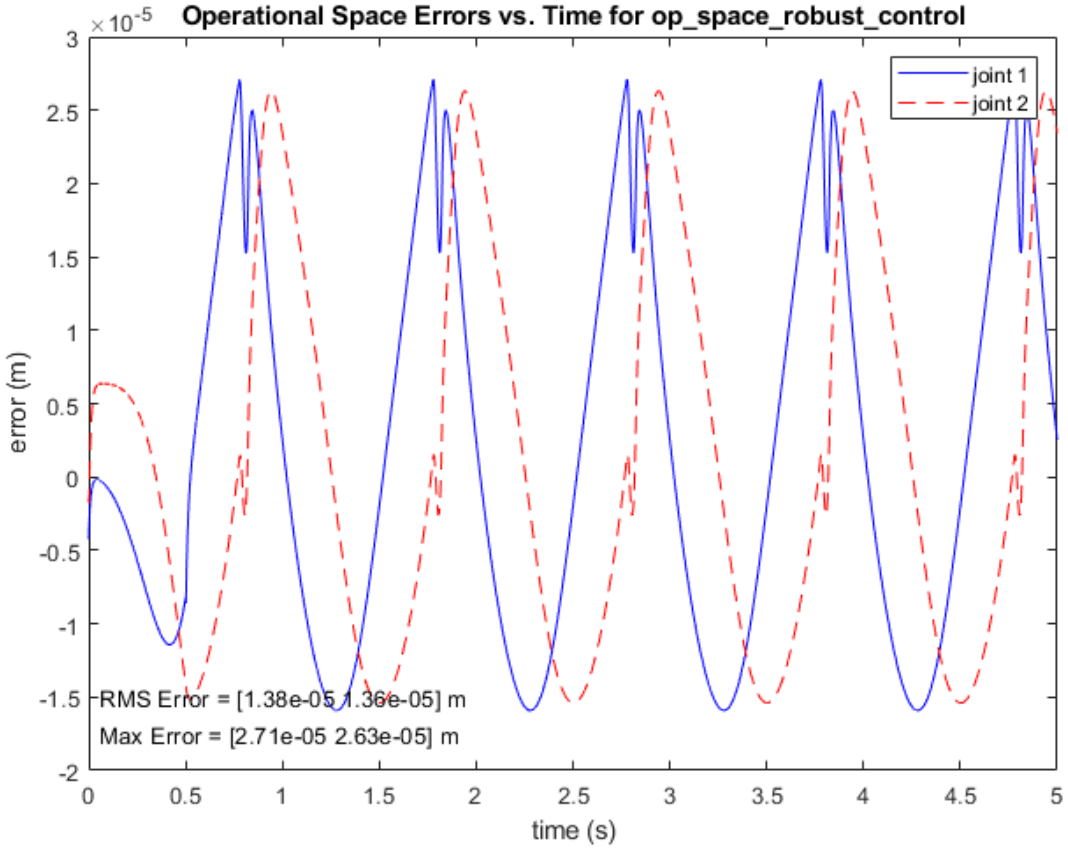


2.3.4. High Speed Simulation (f=1 circles/s)

2.3.4.1. X-Y Trajectory

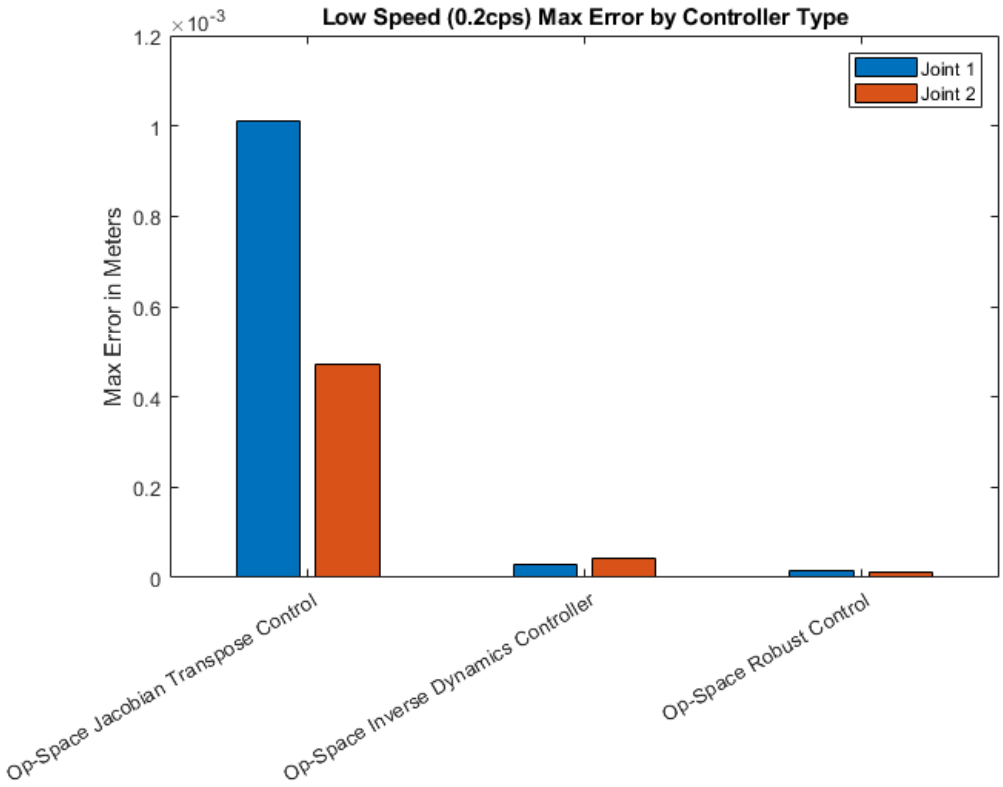


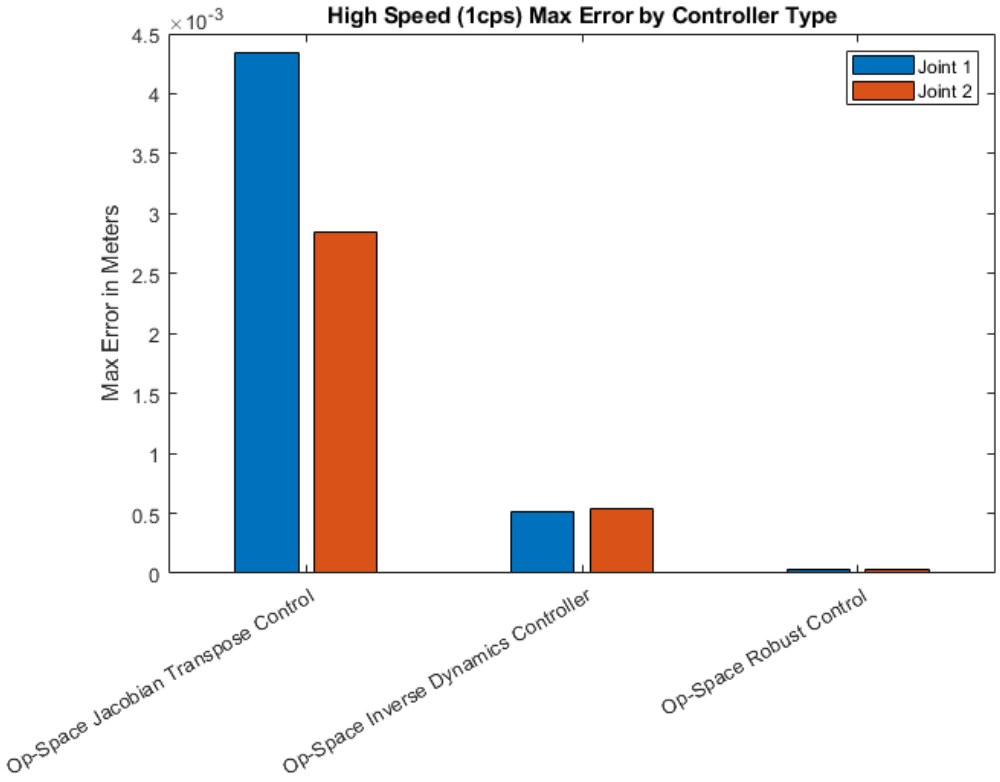
2.3.4.2. Operational Space Errors



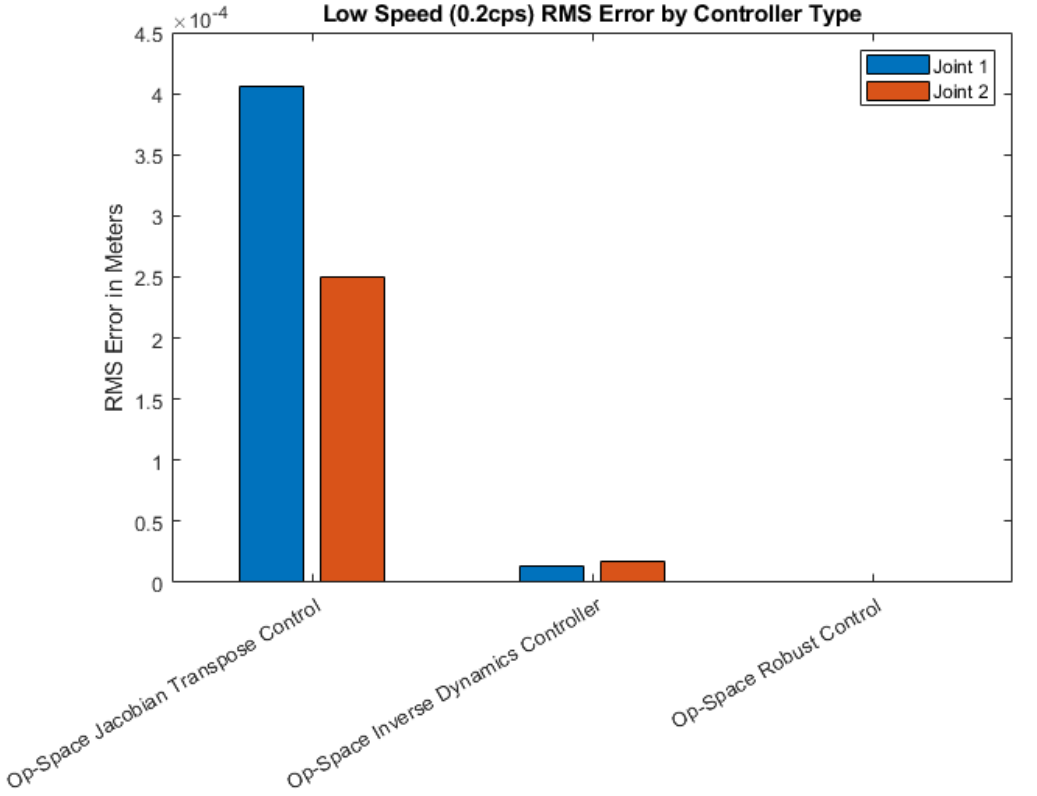
2.4. Controller Comparison

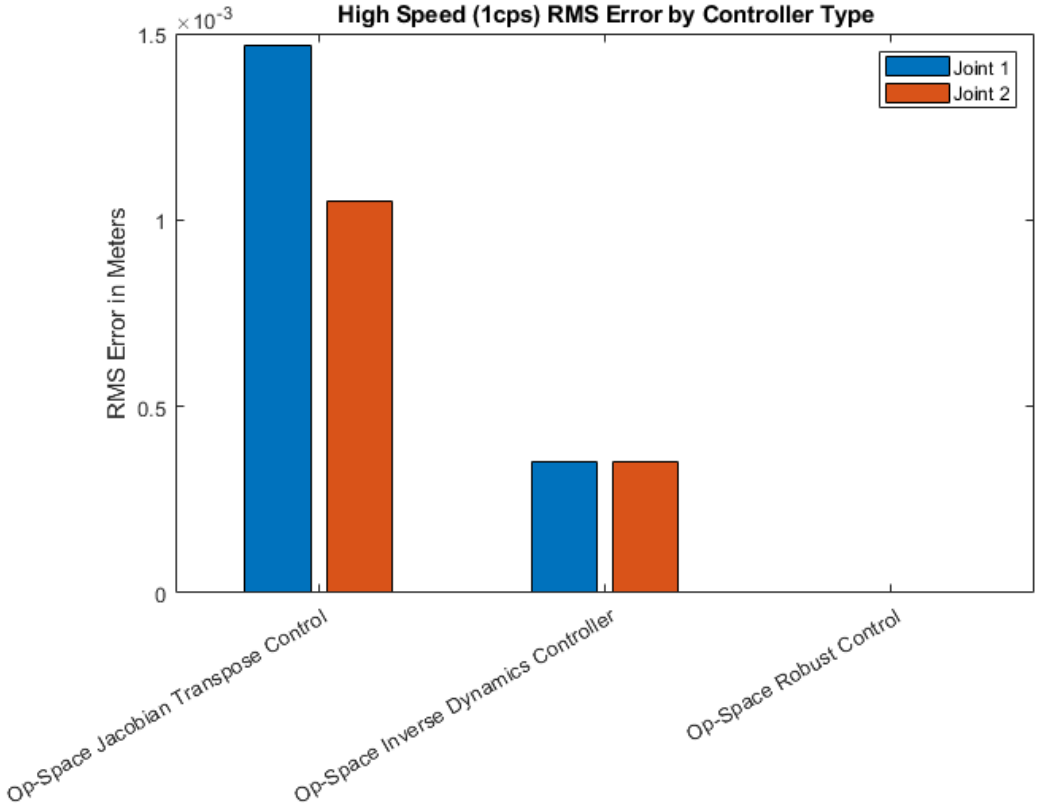
2.4.1. Peak Operational Space Errors





2.4.2. Root-Mean-Square Operational Space Errors





2.4.3. Comparison

As can be seen above the operational space controller had the highest RMS and max error, followed by the operational space inverse dynamics controller, and then the operational space robust controller which had the lowest RMS and max error. This follows the trends of the equivalent controllers in joint space. This shows that the controllers designed in operational space give comparable performance to the controllers which were previously designed in joint space.

2.4.4. Comparison Code

```
%% ME EN 6230 Problem Set 8 Ryan Dalby
% Jacobian Transpose Related Controller Comparison
% close all;
set(groot, 'DefaultTextInterpreter', 'none') % Prevents underscore from
becoming subscript

% errors are in meters
% Operational space jacobian transpose controller
jacobian_transpose_slow_rms = [0.000406 0.00025];
jacobian_transpose_slow_max = [0.00101 0.000473];
jacobian_transpose_fast_rms = [0.00147 0.00105];
jacobian_transpose_fast_max = [0.00434 0.00285];

% Operational space inverse dynamics controller
op_space_idc_slow_rms = [1.36e-05 1.75e-05];
op_space_idc_slow_max = [2.8e-05 4.28e-05];
op_space_idc_fast_rms = [0.000353 0.000351];
op_space_idc_fast_max = [0.000519 0.000541];

% Operational space inverse dynamics controller
op_space_robust_slow_rms = [5.92e-07 6.81e-07];
op_space_robust_slow_max = [4.24e-06 1.52e-06];
op_space_robust_fast_rms = [1.38e-05 1.36e-05];
op_space_robust_fast_max = [2.71e-05 2.63e-05];


slow_rms = [jacobian_transpose_slow_rms; op_space_idc_slow_rms;
op_space_robust_slow_rms];
fast_rms = [jacobian_transpose_fast_rms; op_space_idc_fast_rms;
op_space_robust_slow_max];
slow_max = [jacobian_transpose_slow_max; op_space_idc_slow_max;
op_space_robust_fast_rms];
fast_max = [jacobian_transpose_fast_max; op_space_idc_fast_max;
op_space_robust_fast_max];

controller_labels = {'Op-Space Jacobian Transpose Control', 'Op-Space
Inverse Dynamics Controller', 'Op-Space Robust 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 Meters');

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 Meters');

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 Meters');

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 Meters');
```

Appendix- Plotting Code (Used to make simulation plots)

```
%% ME EN 6230 Problem Set 8 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 = rad2deg(errors.signals.values); % deg
op_errors = xy - xy_d; % m
actual_trajectory = xy; % m
desired_trajectory = xy_d; % m

% RMS Joint Errors
RMS_joint_error = rms(joint_errors); % deg
% Max Joint Errors
max_joint_error = max(abs(joint_errors)); % deg

% RMS Operational Space Errors
RMS_op_error = rms(op_errors); % deg
% Max Operational Space Errors
max_op_error = max(abs(op_errors)); % deg

% Plot Joint Errors vs Time
if strcmp(model_title, 'jacobian_inverse_control')
    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_joint_error,3),'
deg'), 'Units', 'normalized');
    hold on;
    text(0.01,0.05,append('Max Error = ', mat2str(max_joint_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');
end

% Plot Operational Space Errors vs Time
```

```
figure;
plot(time, op_errors(:,1), 'b-');
hold on;
plot(time, op_errors(:,2), 'r--');
hold on;
text(0.01,0.10,append('RMS Error = ', mat2str(RMS_op_error,3),' m'),
'Units', 'normalized');
hold on;
text(0.01,0.05,append('Max Error = ', mat2str(max_op_error,3),' m'),
'Units', 'normalized');
title(append('Operational Space Errors vs. Time for ', model_title));
xlabel('time (s)');
ylabel('error (m)');
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');
```