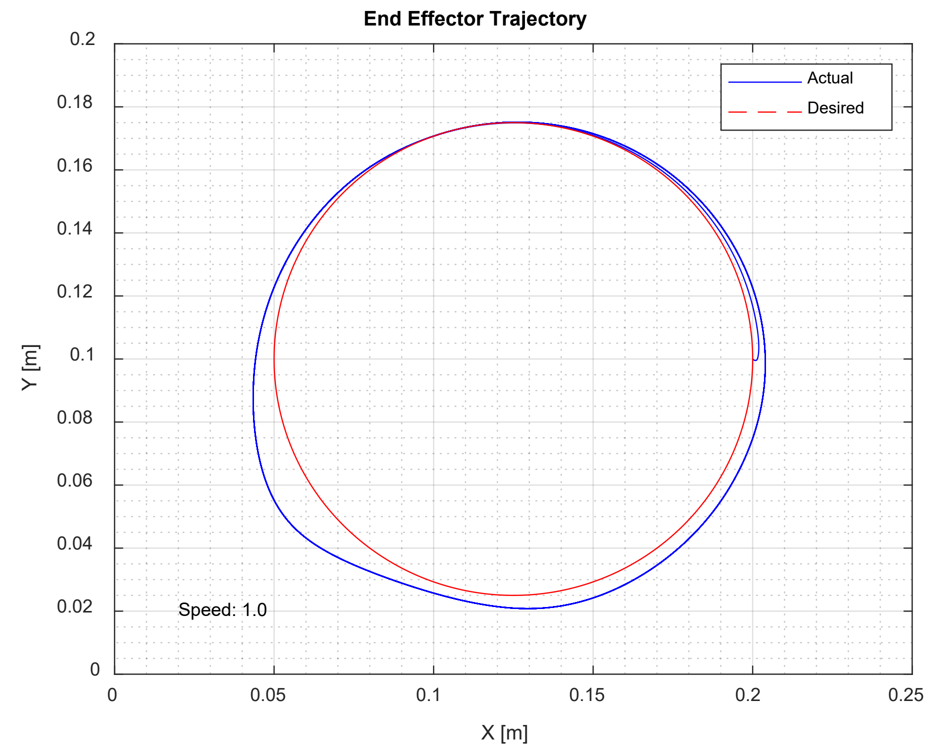
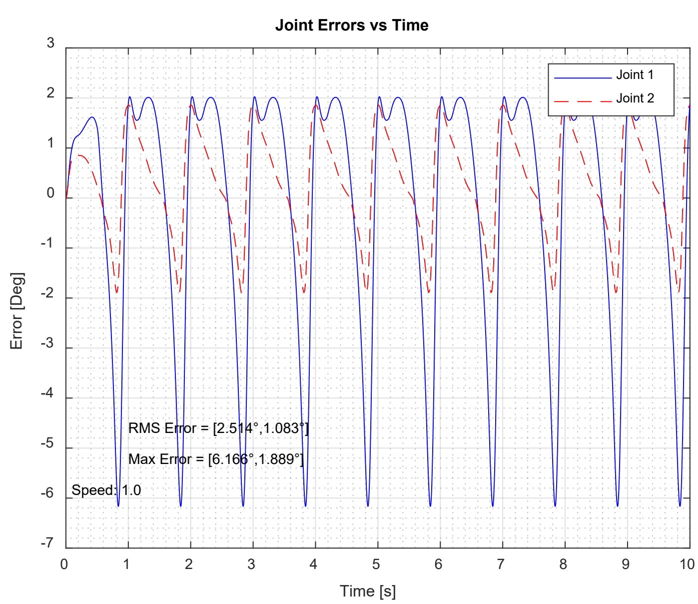
Zachary Orton

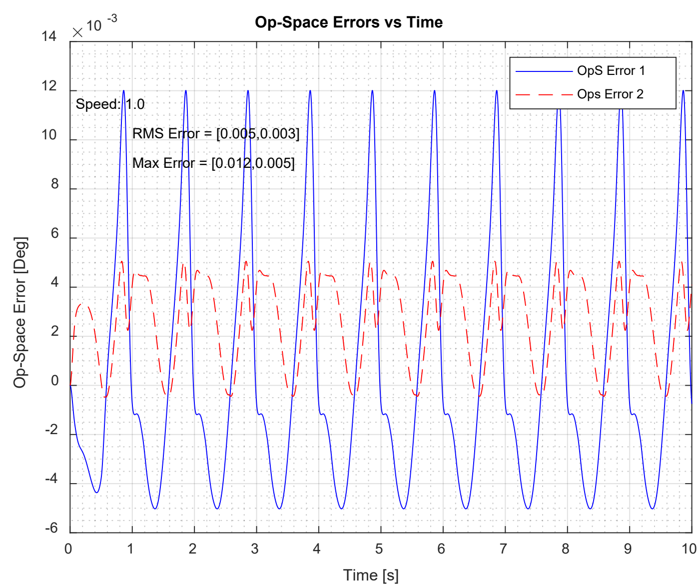
3/29/2023

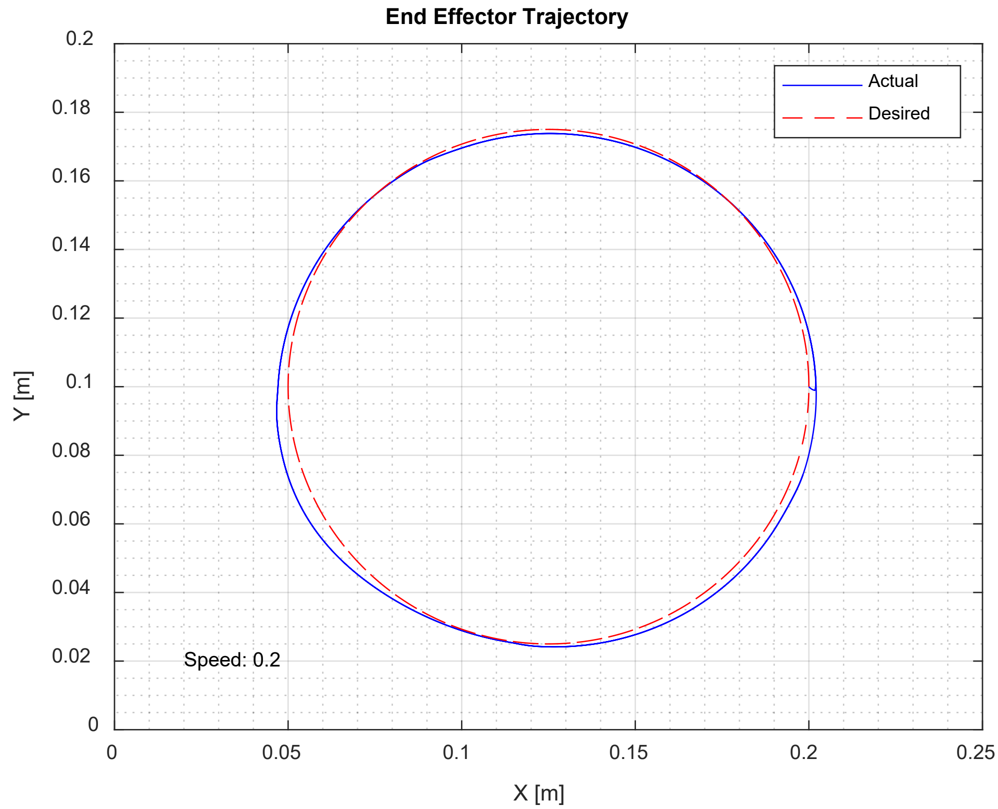
PS# 7

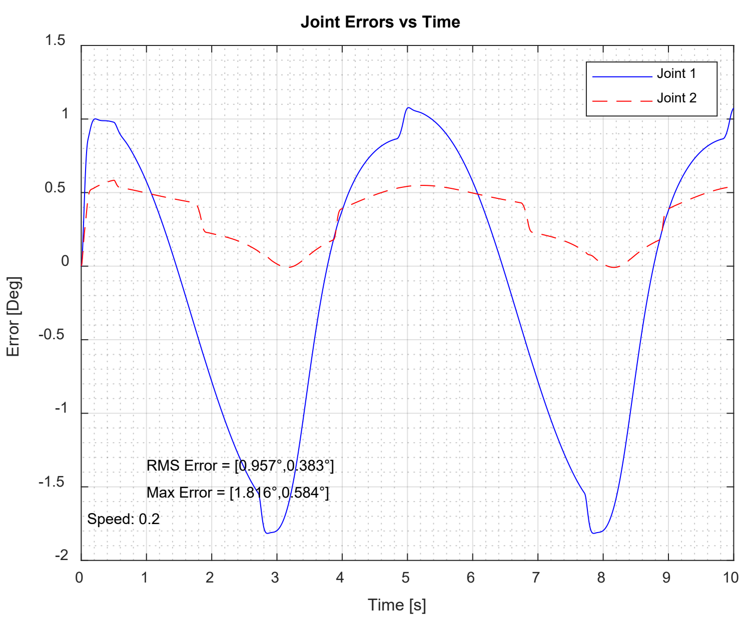
P1)

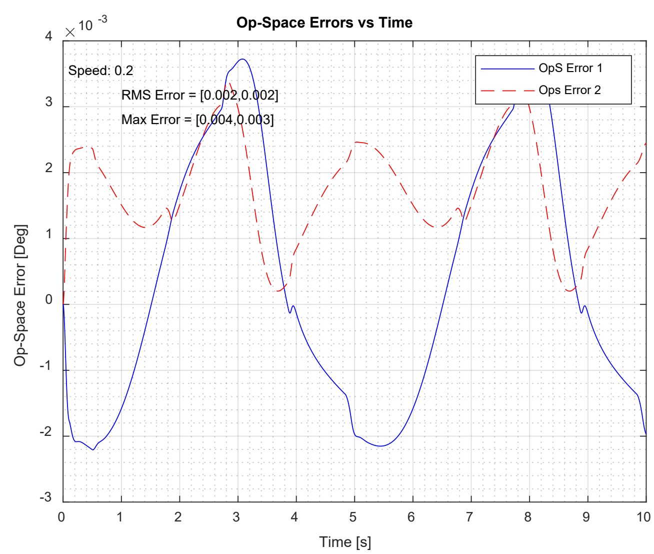


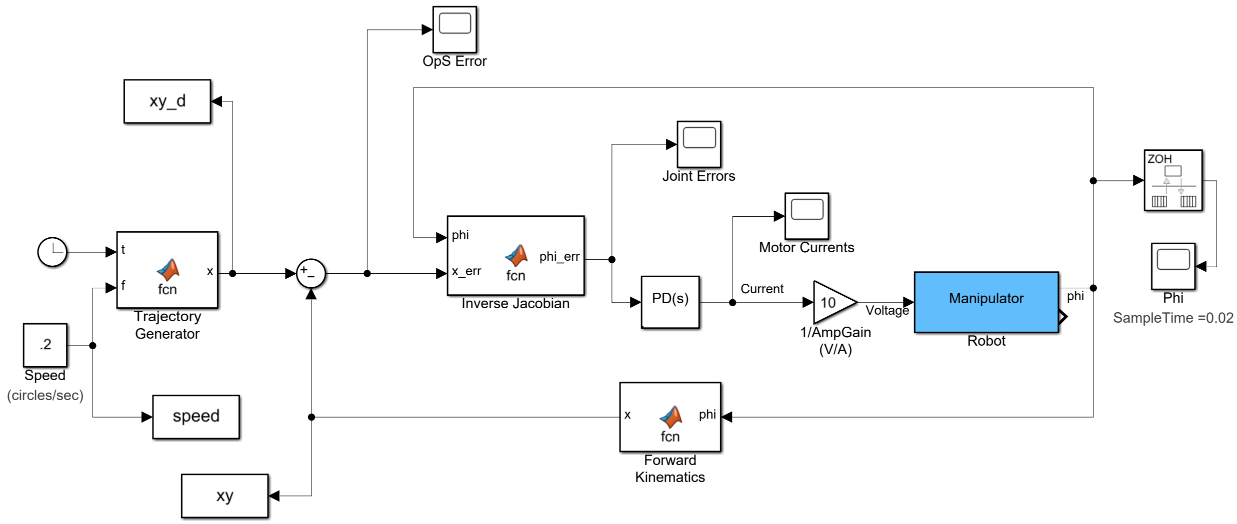





End Effector Trajectory
Actual
Desired
Speed: 0.2
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.05
0.1
0.15
0.2
0.25


Joint Errors vs Time
Joint 1
Joint 2
RMS Error = [0.957°,0.383°]
Max Error = [1.816°,0.584°]
Speed: 0.2
Error [Deg]
Time [s]


Op-Space Errors vs Time
× 10 -3
Speed: 0.2
RMS Error = [0.002,0.002]
Max Error = [0.004,0.003]
OpS Error 1
Ops Error 2
Op-Space Error [Deg]
Time [s]



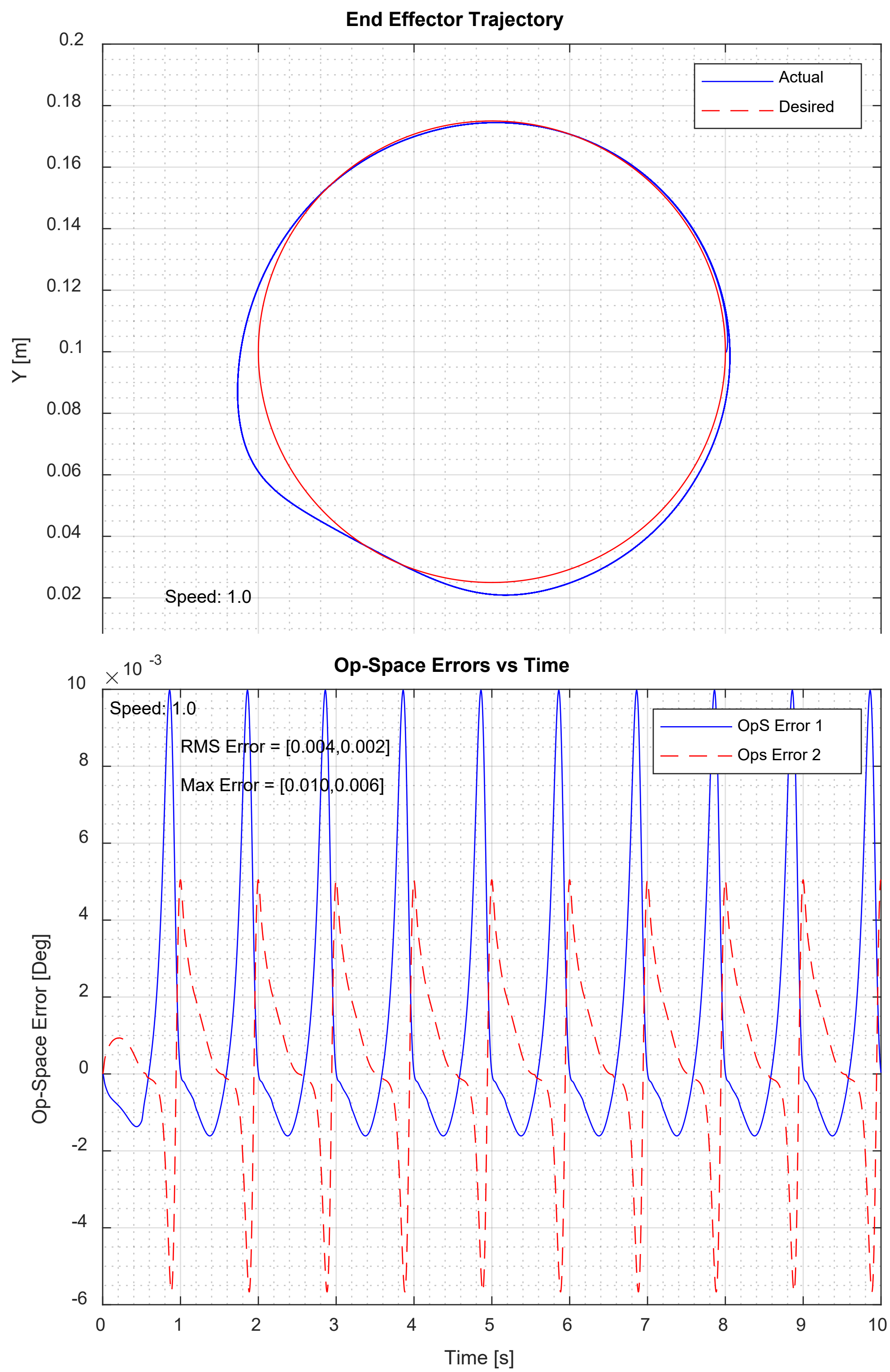
Jacobian Inverse Control

```
function phi_err = fcn(phi, x_err)
%Inverse jacobian
a1=0.15;
a2=0.15;

J =[-a1*sin(phi(1)) - a2*sin(phi(2)), -a2*sin(phi(2)) ; a1*cos(phi(1)) + a2*cos(phi(2)), a2*cos(phi(2))] *[1, 0 ; -1, 1];

Jinv = inv(J);

phi_err = Jinv * x_err;
```

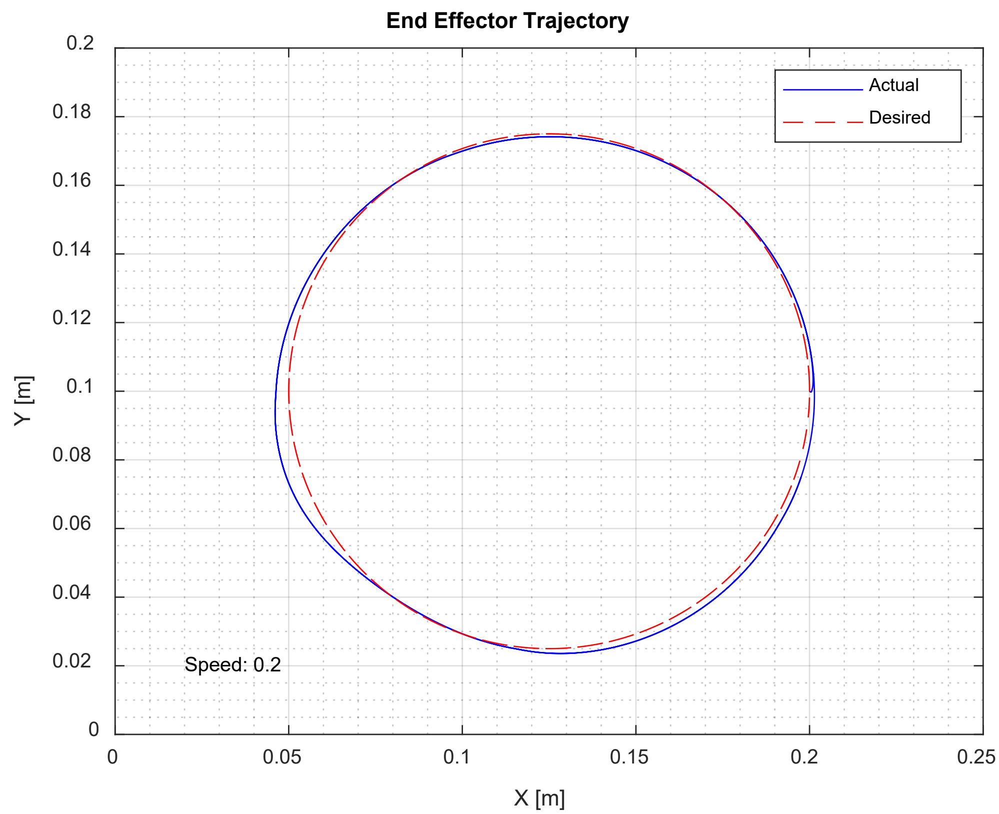
Compare:

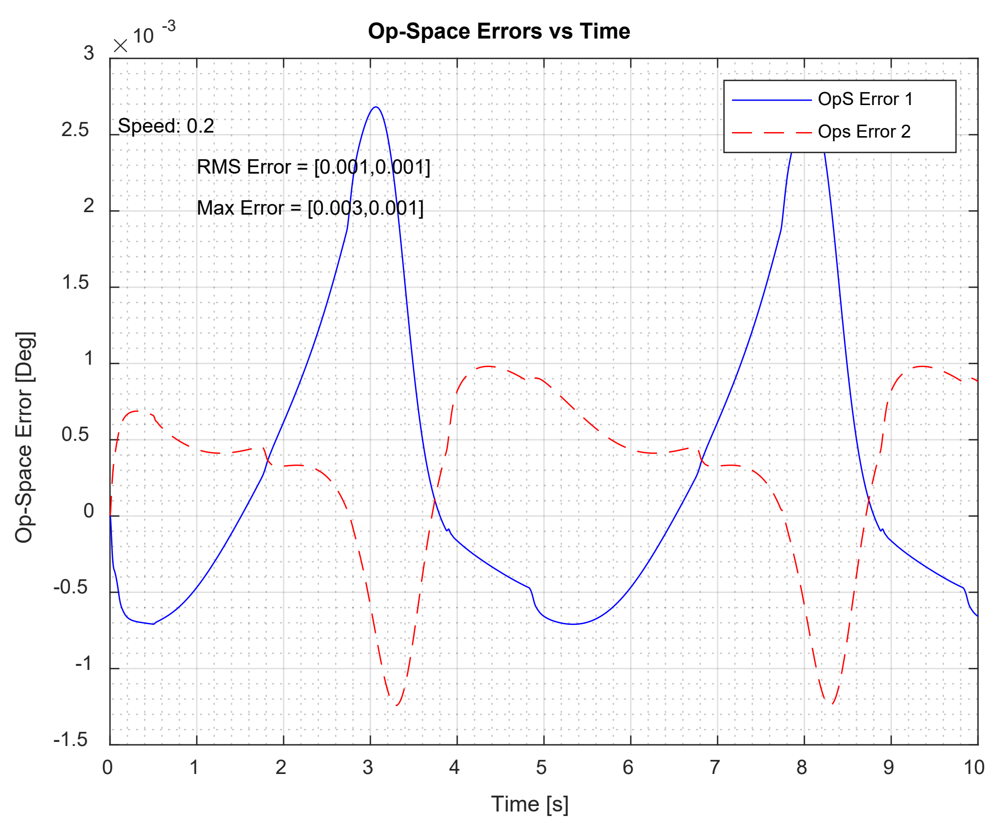
We see really close results to the ones we saw in PS5 with the normal PD control and feedforward in joint space, operational space may be slightly worse when directly comparing RMSE.

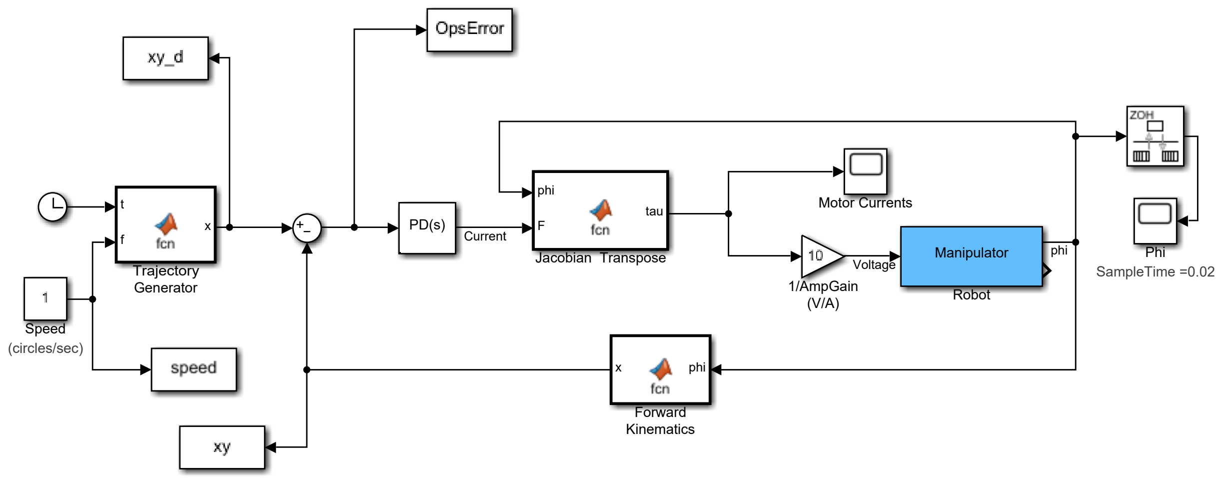
P2.1)



End Effector Trajectory
Actual
Desired
Speed: 1.0
Y [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
Op-Space Errors vs Time
× 10^-3
Speed: 1.0
RMS Error = [0.004,0.002]
Max Error = [0.010,0.006]
OpS Error 1
Ops Error 2
Op-Space Error [Deg]
10
8
6
4
2
0
-2
-4
-6
0
1
2
3
4
5
6
7
8
9
10
Time [s]


End Effector Trajectory
Actual
Desired
Speed: 0.2
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.05
0.1
0.15
0.2
0.25

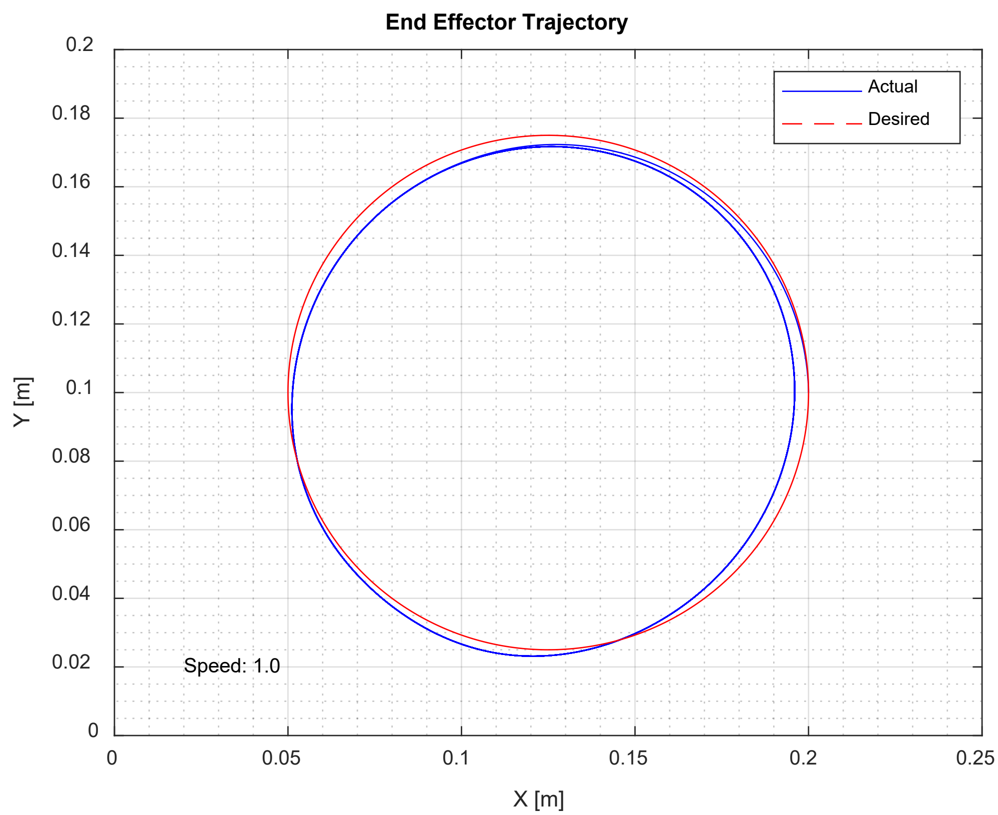

Op-Space Errors vs Time
× 10 -3
OpS Error 1
Ops Error 2
Speed: 0.2
RMS Error = [0.001,0.001]
Max Error = [0.003,0.001]
Op-Space Error [Deg]
Time [s]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
1
2
3
4
5
6
7
8
9
10



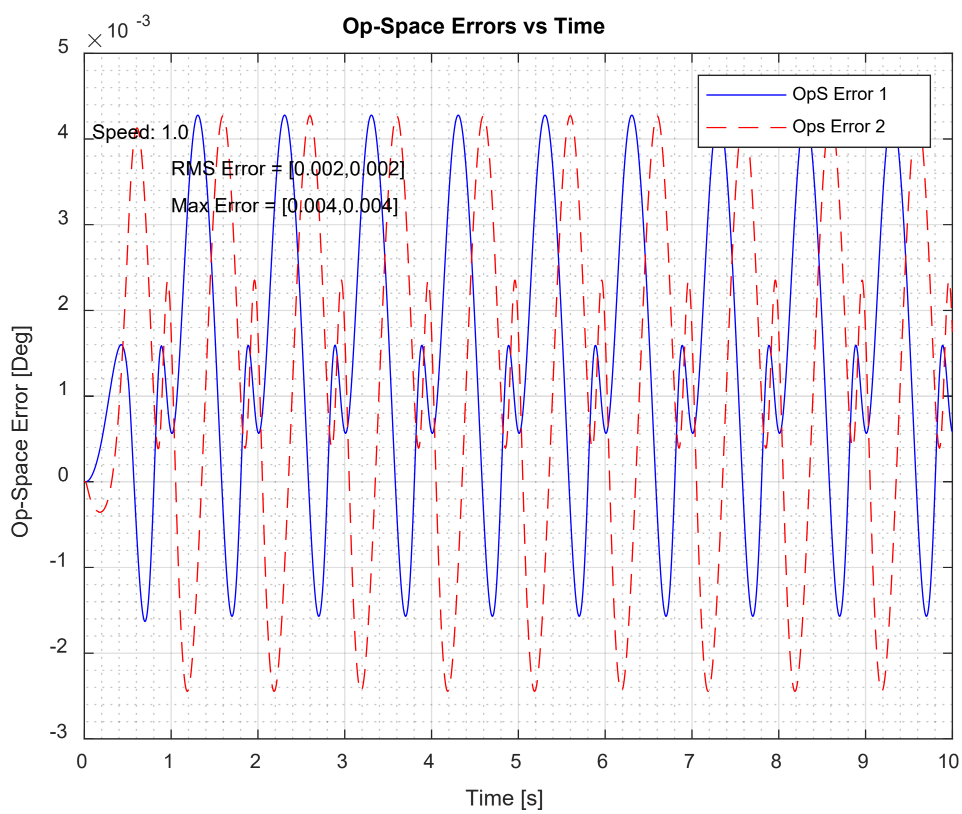
Jacobian Transpose Control

```
function tau = fcn(phi, F)
%Inverse jacobian
a1=0.15;
a2=0.15;

J =[-a1*sin(phi(1)) - a2*sin(phi(2)), -a2*sin(phi(2)) ; a1*cos(phi(1)) + a2*cos(phi(2)), a2*cos(phi(2))] *[1, 0 ; -1, 1];

Jtrans = J';

tau = Jtrans * F;
```

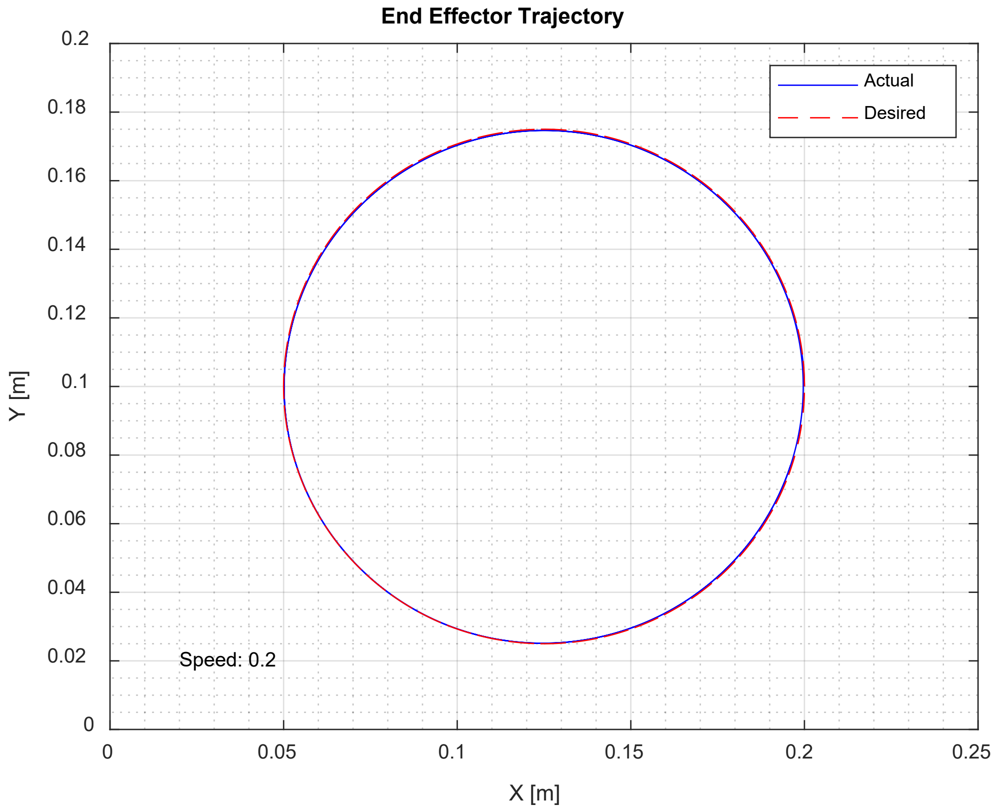
P2.2)

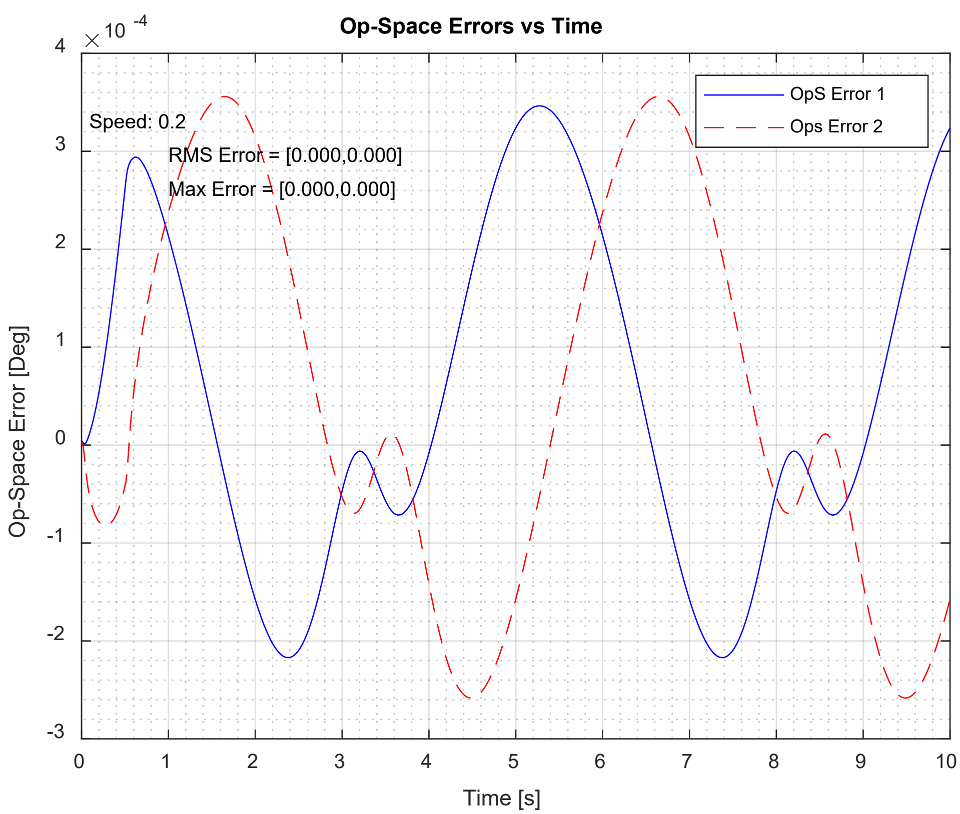


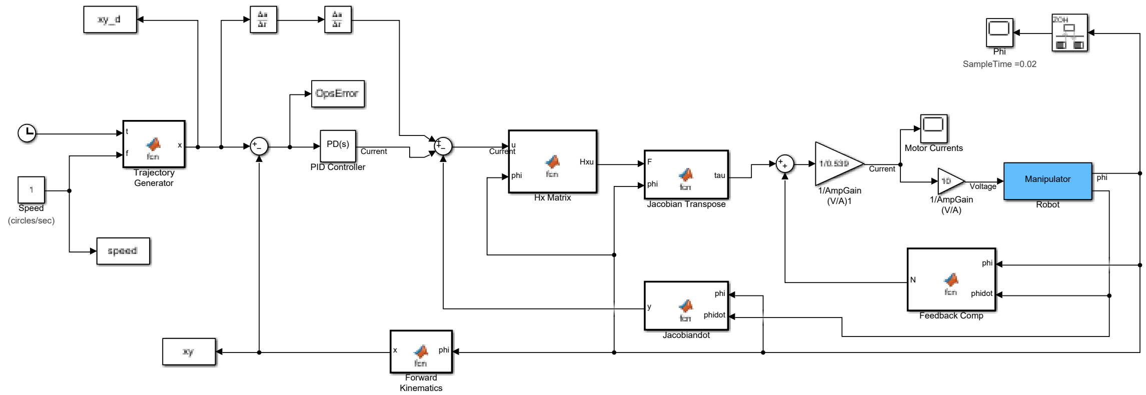
End Effector Trajectory
Actual
Desired
Speed: 1.0
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.05
0.1
0.15
0.2
0.25



Op-Space Errors vs Time
×10 -3
OpS Error 1
Ops Error 2
Speed: 1.0
RMS Error = [0.002,0.002]
Max Error = [0.004,0.004]
Op-Space Error [Deg]
Time [s]
5
4
3
2
1
0
-1
-2
-3
0
1
2
3
4
5
6
7
8
9
10


End Effector Trajectory
Actual
Desired
Speed: 0.2
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.05
0.1
0.15
0.2
0.25


Op-Space Errors vs Time
× 10 -4
OpS Error 1
Ops Error 2
Speed: 0.2
RMS Error = [0.000,0.000]
Max Error = [0.000,0.000]
Op-Space Error [Deg]
Time [s]
4
3
2
1
0
-1
-2
-3
0
1
2
3
4
5
6
7
8
9
10



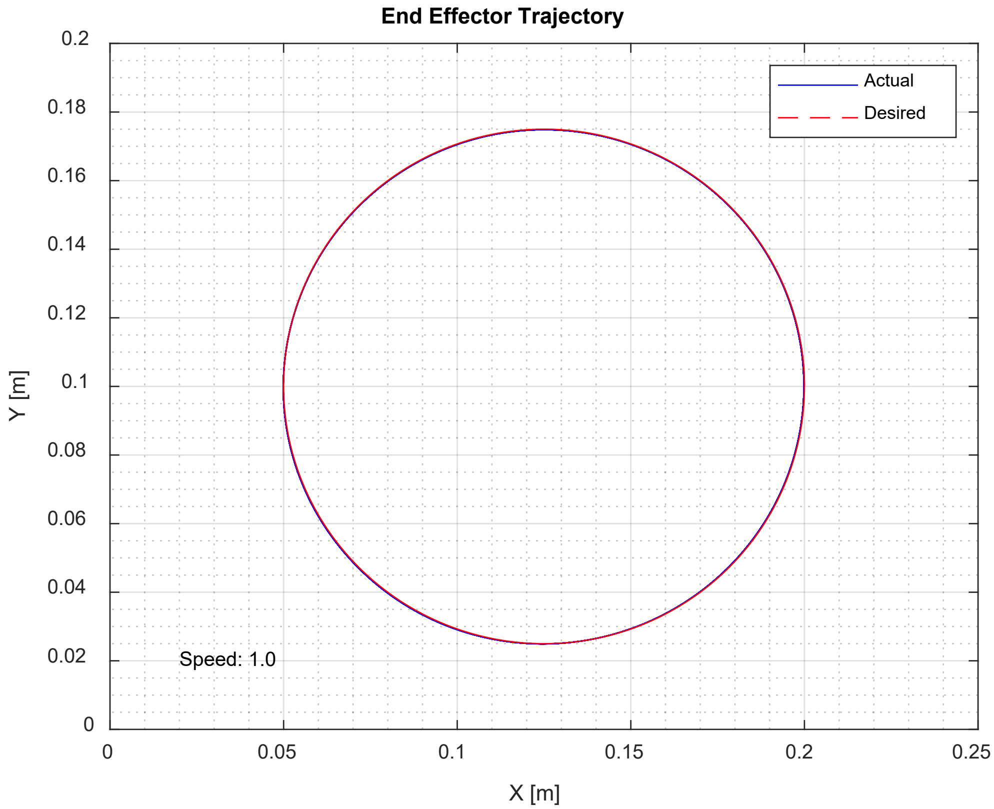
Inverse Dynamics Control in Operational Space

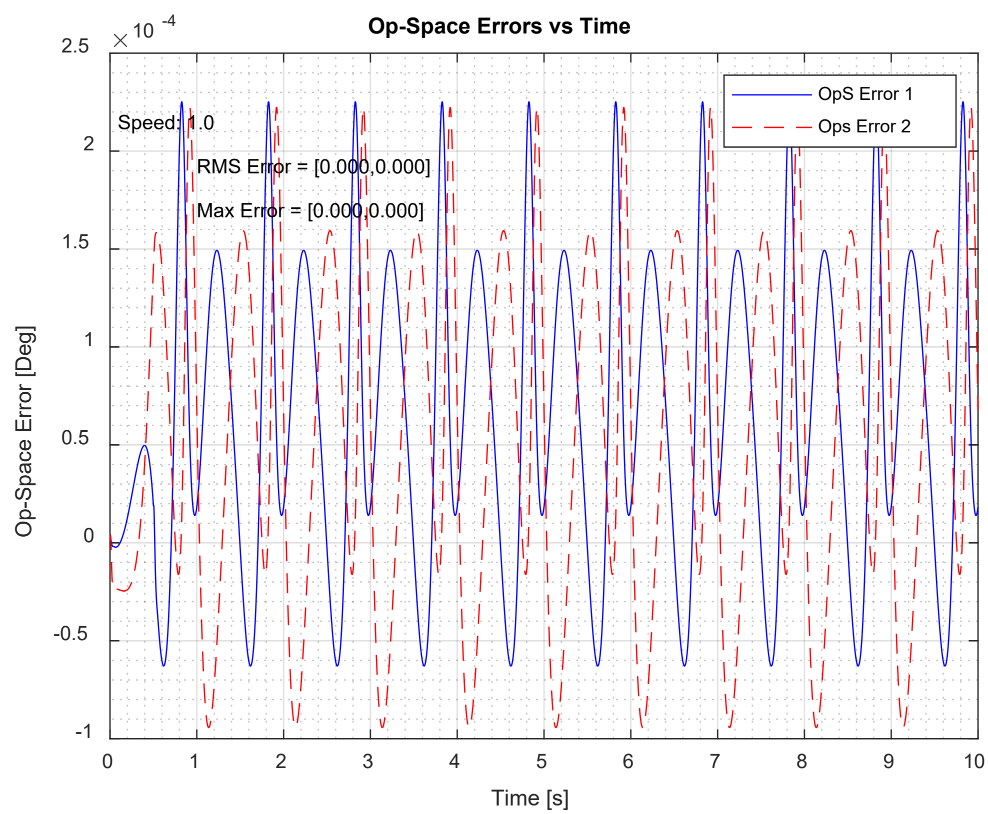
```
function Hxu = fcn(u, phi)
% Inertia Matrix.

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


J =[-a1*sin(phi(1)) - a2*sin(phi(2)), -a2*sin(phi(2)) ; a1*cos(phi(1)) + a2*cos(phi(2)), a2*cos(phi(2))] *[1, 0 ; -1, 1];

Jtrans = J';
Jinv = inv(J);



Hxu = inv(Jtrans)*H*Jinv*u;
```

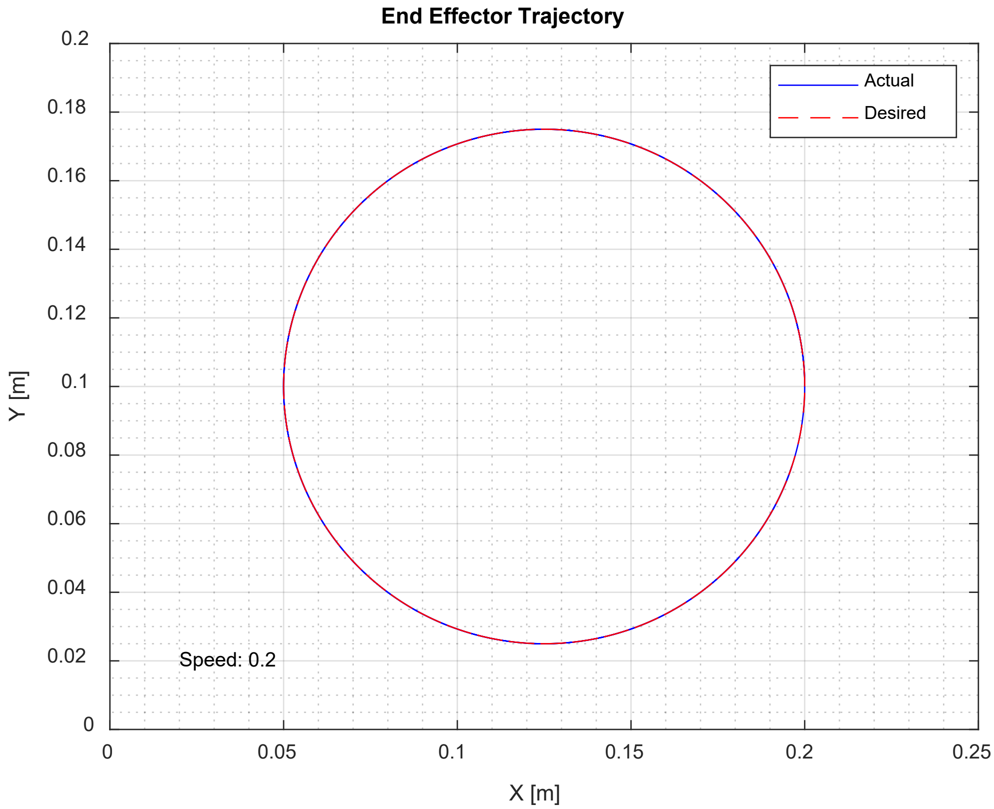
```
function y = fcn(phi, phidot)
%Inverse jacobian
a1=0.15;
a2=0.15;

Jdot =[-a1*cos(phi(1)), -a2*cos(phi(2)) ; -a1*sin(phi(1)), -a2*sin(phi(2))];

%take derrivate of J

y = Jdot * phidot;
```

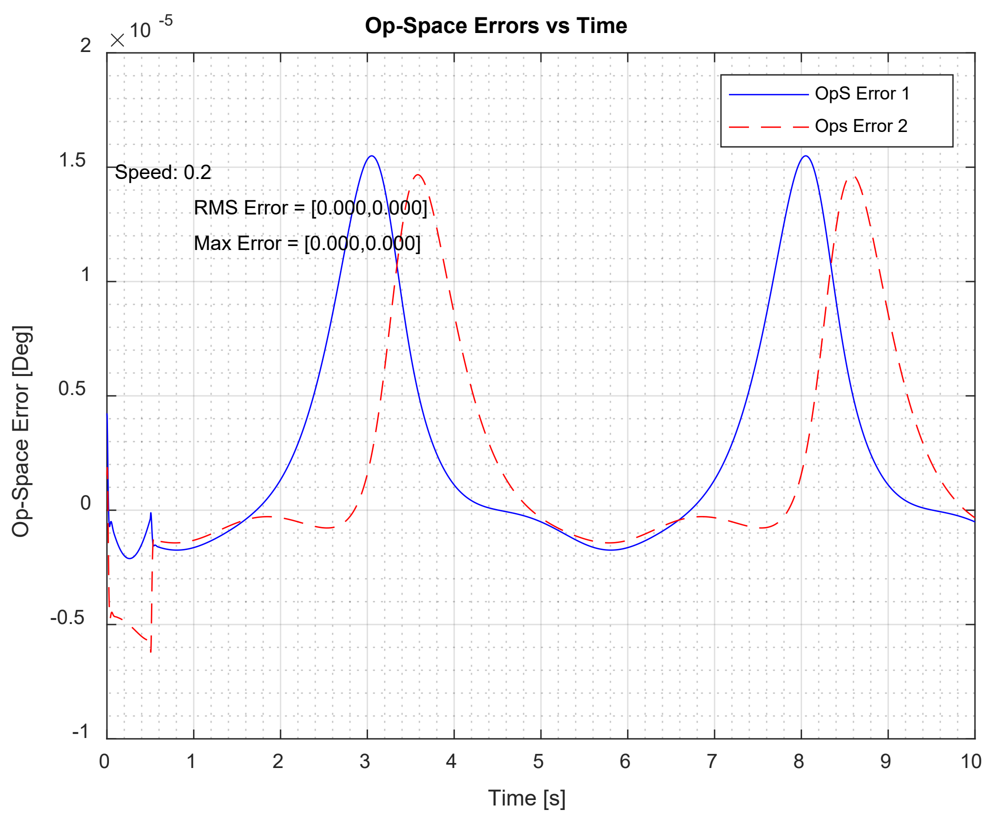
```
function tau = fcn(F, phi)
%Inverse jacobian
a1=0.15;
a2=0.15;

J =[-a1*sin(phi(1)) - a2*sin(phi(2)), -a2*sin(phi(2)) ; a1*cos(phi(1)) + a2*cos(phi(2)), a2*cos(phi(2))] *[1, 0 ; -1, 1];

Jtrans = J';

tau = Jtrans * F;
```

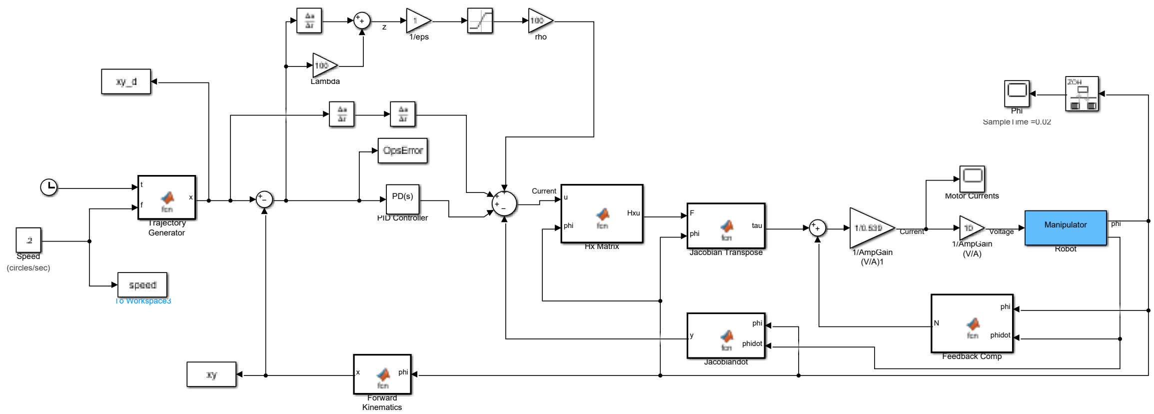
P2.3)


End Effector Trajectory
Actual
Desired
Speed: 1.0
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0.05
0.1
0.15
0.2
0.25


Op-Space Errors vs Time
$\times 10^{-4}$
OpS Error 1
Ops Error 2
Speed: 1.0
RMS Error = [0.000,0.000]
Max Error = [0.000,0.000]
Op-Space Error [Deg]
Time [s]
2.5
2
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
9
10


End Effector Trajectory
Actual
Desired
Speed: 0.2
Y [m]
X [m]
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.05
0.1
0.15
0.2
0.25


Op-Space Errors vs Time
×10^-5
OpS Error 1
Ops Error 2
Speed: 0.2
RMS Error = [0.000,0.000]
Max Error = [0.000,0.000]
Op-Space Error [Deg]
Time [s]
2
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
9
10



Robust Control in Operational Space

Compare: Just like we have seen before IDC is better than PD and sliding control with IDC gives us the best tracking, we see the same results here in operational space.

Appendix:

Code:

```
%Zachary Orton Plotting Script
close all
%Plot actual end effector trajectory
figure(1)
plot(xy(:,1) , xy(:,2), 'b')
grid on
grid minor
hold on

%Plot desired end effector trajectory
plot(xy_d(:,1) , xy_d(:,2), '--r')
ylabel("Y [m]")
xlabel("X [m]")
title("End Effector Trajectory")
legend("Actual", "Desired")
axis([0 0.25 0 0.2])

%Add the circle speed to the plot
text(.02,0.02, sprintf("Speed: %.1f", speed))

%%
% %Get joint errors in degrees
% jerrDeg = rad2deg(errors.signals.values);
% Jointerror1 = jerrDeg(:,1);
% Jointerror2 = jerrDeg(:,2);
%
% % Root Mean Squared Errors & Max Errors
```

```
% RMSE1 = sqrt(mean((Jointerror1).^2));
% RMSE2 = sqrt(mean((Jointerror2).^2));
% RMSE = [RMSE1 , RMSE2];
% MaxError1 = max(abs(Jointerror1));
% MaxError2 = max(abs(Jointerror2));
% MaxError = [MaxError1, MaxError2];
%
% figure(2)
% %plot joint errors vs time
% plot(errors.time , Jointerror1, 'b')
% grid on
% grid minor
% hold on
% plot(errors.time , Jointerror2, '--r')
% ylabel("Error [Deg]")
% xlabel("Time [s]")
% title("Joint Errors vs Time")
% legend("Joint 1", "Joint 2")
% %add text about speed
% text(0.09, -MaxError(1)*.95, sprintf("Speed: %.1f", speed))
% text(1, -MaxError(1)*.75 , sprintf("RMS Error = [%.3f°,%.3f°]", RMSE(1), RMSE(2)))
% text(1, -MaxError(1)*.85, sprintf("Max Error = [%.3f°,%.3f°]", MaxError(1), MaxError(2)))

%%
%OpS Error

Opserror1 = OpsError.signals.values(:,1);
Opserror2 = OpsError.signals.values(:,2);

% Root Mean Squared Errors & Max Errors
ORMSE1 = sqrt(mean((Opserror1).^2));
ORMSE2 = sqrt(mean((Opserror2).^2));
ORMSE = [ORMSE1 , ORMSE2];
MaxOpsError1 = max(abs(Opserror1));
MaxOpsError2 = max(abs(Opserror2));
MaxOpsError = [MaxOpsError1, MaxOpsError2];

figure(3)
plot(OpsError.time, Opserror1, 'b')
grid on
grid minor
hold on
plot(OpsError.time, Opserror2, '--r')
ylabel("Op-Space Error [Deg]")
xlabel("Time [s]")
title("Op-Space Errors vs Time")
legend("OpS Error 1", "Ops Error 2")

text(0.09, MaxOpsError(1)*.95, sprintf("Speed: %.1f", speed))
text(1, MaxOpsError(1)*.85 , sprintf("RMS Error = [%.3f,%.3f]", ORMSE(1), ORMSE(2)))
text(1, MaxOpsError(1)*.75, sprintf("Max Error = [%.3f,%.3f]", MaxOpsError(1),
MaxOpsError(2)))
```