# Robot Control Lab 4

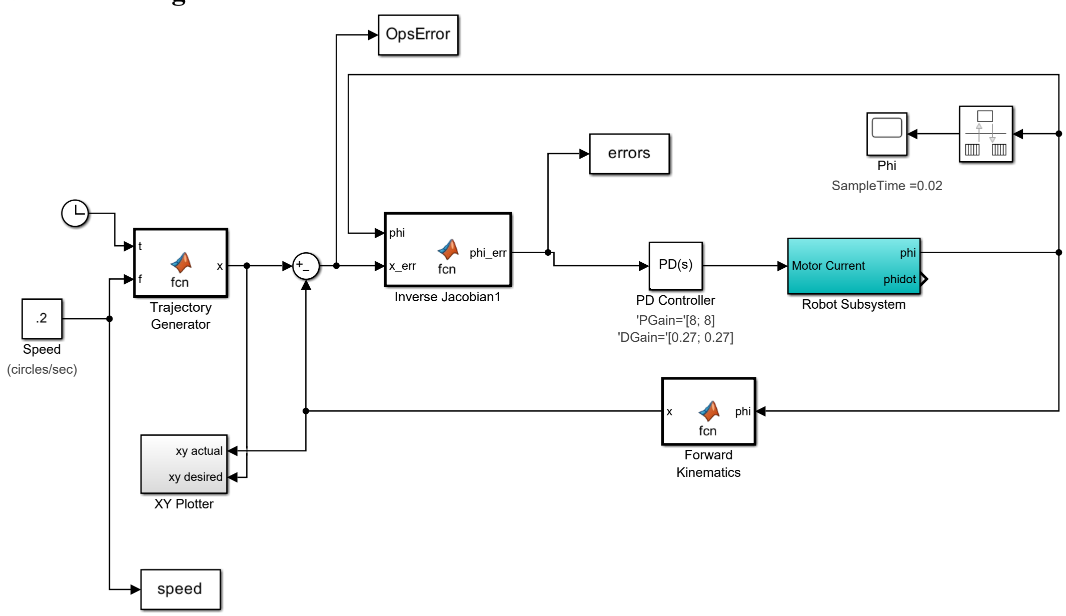
Will Graham

Simulink Functions are included in the Appendix.

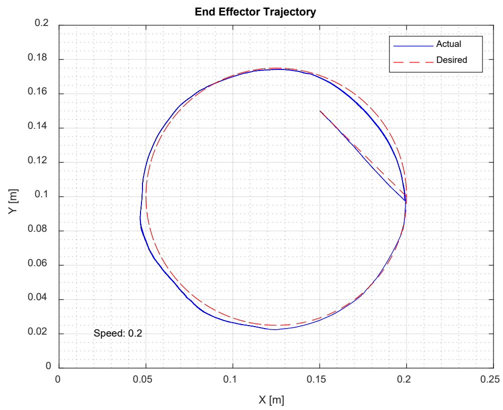
## 1 JACOBIAN INVERSE CONTROL

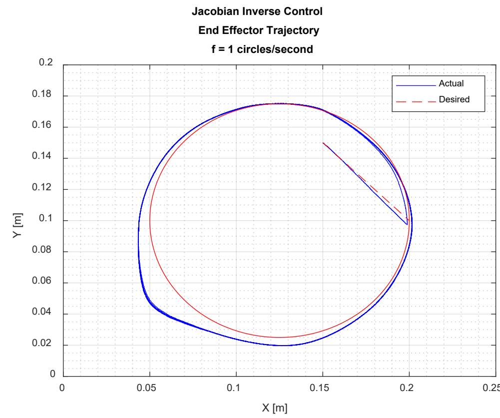
1. First implement **Jacobian Inverse Control**, in which case you can use the same the same PD gains that you used when doing decentralized control in joint space (Lab 2, problem 1). Simulate at low speed (f=0.2 circles/s) and high speed (f=1 circles/s), and compare the **trajectory** and **joint errors** from with Problem 1.1 of Lab 2.

### 1.1.1 Simulink Diagram

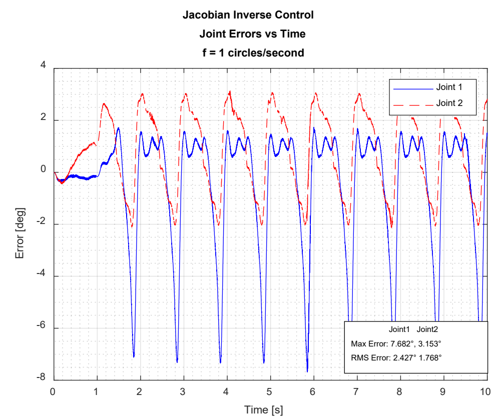


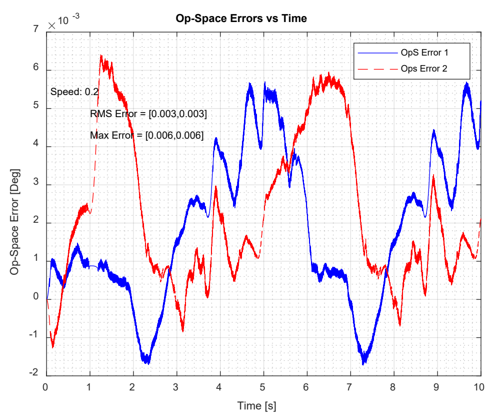
### 1.1.2 Plot Results

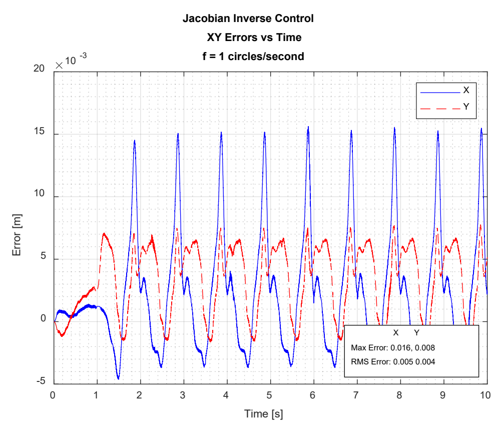


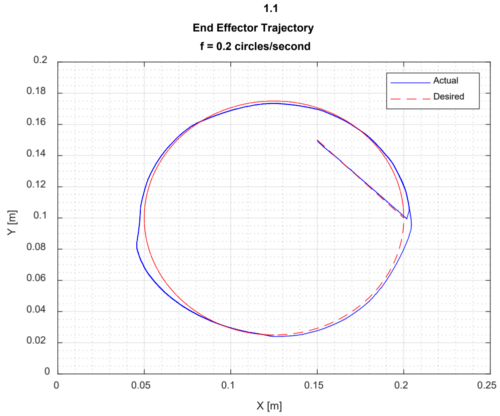


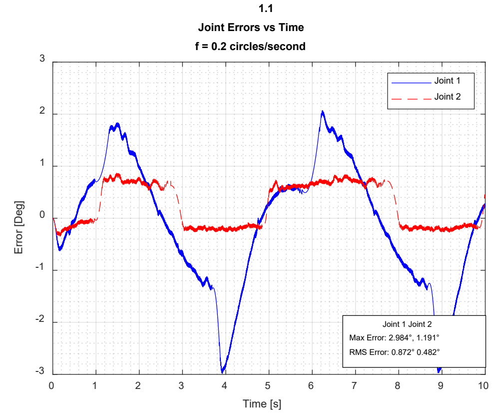


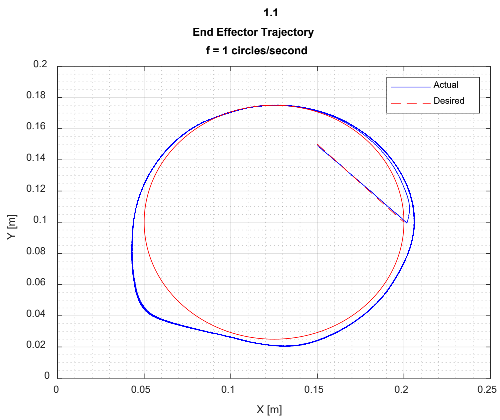



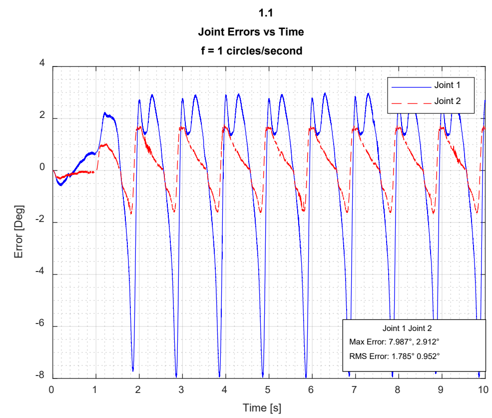



## 1.1.3 Lab 2 Controllers




## 1.1.4 Comparison Tables and Analysis

Table of Joint Error Results from Jacobian Inverse Control

| | | f = 0.2 circles/sec | f = 1 circles/sec | % Difference between fast/slow |
|---|---|---|---|---|
| **Joint 1** | Max [deg] | 2.375 | 7.682 | 69% |
| | RMS [deg] | 0.886 | 2.427 | 63% |
| **Joint 2** | Max [deg] | 2.433 | 3.153 | 23% |
| | RMS [deg] | 1.242 | 1.738 | 29% |

Table of Joint Error Results from PD Controller

| | | f = 0.2 circles/sec | f = 1 circles/sec | % Difference between fast/slow |
|---|---|---|---|---|
| **Joint 1** | Max [deg] | 2.984 | 7.987 | 63% |
| | RMS [deg] | 0.872 | 1.785 | 51% |
| **Joint 2** | Max [deg] | 1.91 | 2.912 | 34% |
| | RMS [deg] | 0.482 | 0.952 | 49% |

Jacobian Improvement by Percentages on Both Slow and fast controllers

| | | **f = 0.2 circles/sec** | **f = 1 circles/sec** |
|---|---|---|---|
| **Joint 1** | Max [deg] | -26% | -4% |
| | RMS [deg] | 2% | 26% |
| **Joint 2** | Max [deg] | 21% | 8% |
| | RMS [deg] | 61% | 45% |

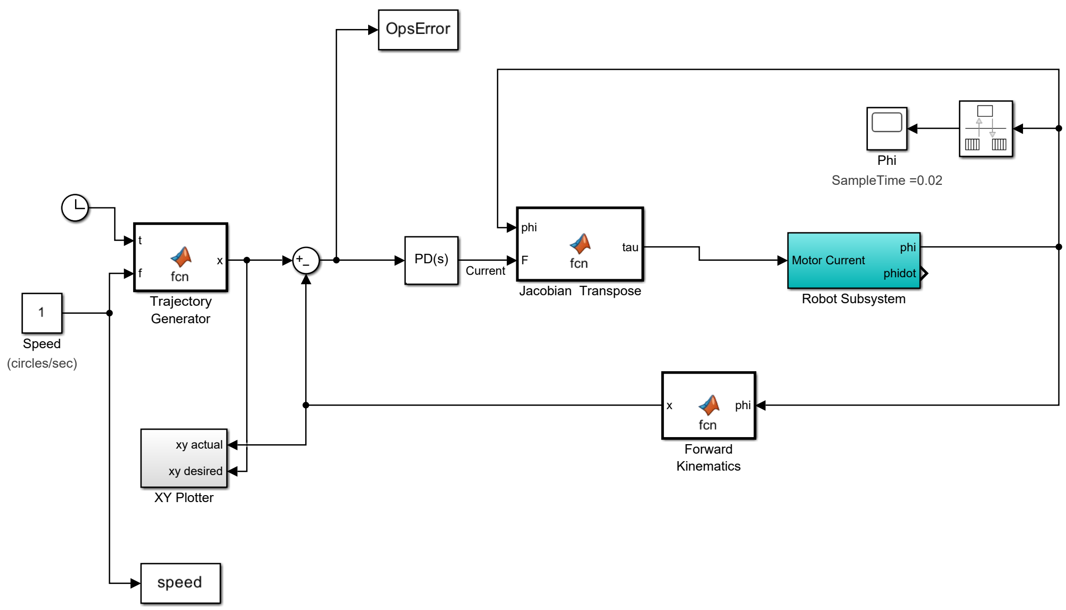
There was a much larger error in joint 2 than joint 1 with inverse Jacobian control. However, there seemed to be a similar trajectory with the lower speed. When trajectory speed was increased there was a noticeable difference with Jacobian inverse control performing much worse with clear inaccuracy at bottom left side of circle trajectory.

# 2 JACOBIAN TRANSPOSE CONTROL

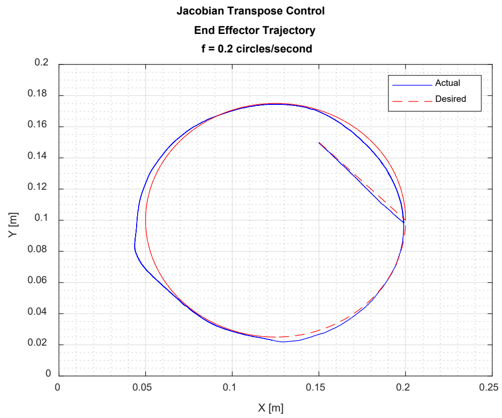
2. Next, you will implement a series of operational space controllers based on the **Jacobian Transpose**. For each of the following control schemes, simulate at low speed (f=0.2 circles/s) and high speed (f=1 circles/s) and compare the tracking performance with the other control schemes. For these controllers, you will not be computing the joint errors at all, so just compare the errors in operational space. Since the PD gains for these controllers will now be penalizing errors in operational space, you will have to find a different set of PD gains than you used in problem 1. For part 2.1, find a set of PD gains that results in comparable RMS errors to problem 1. Then use the same PD gains in parts 2.1-2.3 for a fair comparison.
   **2.1** Jacobian Transpose Control
   **2.2** Inverse Dynamics Control in Operational Space
   **2.3** Robust Control in Operational Space

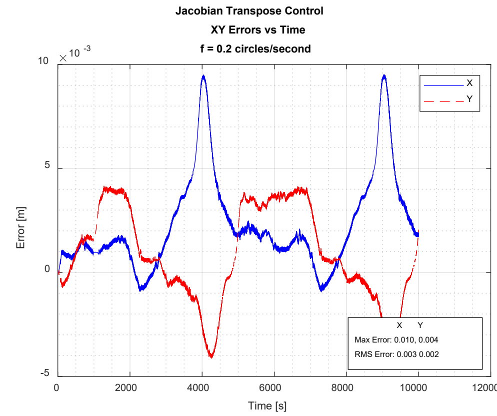
## 2.1 JACOBIAN TRANSPOSE CONTROL

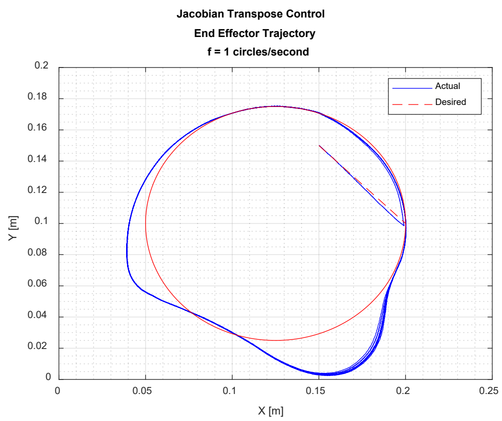
### 2.1.1 Simulink Model

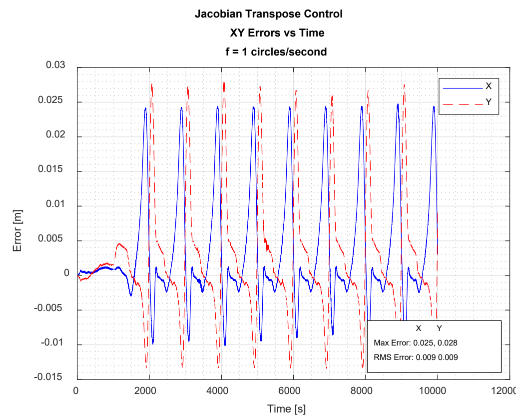


### 2.1.2 Plots



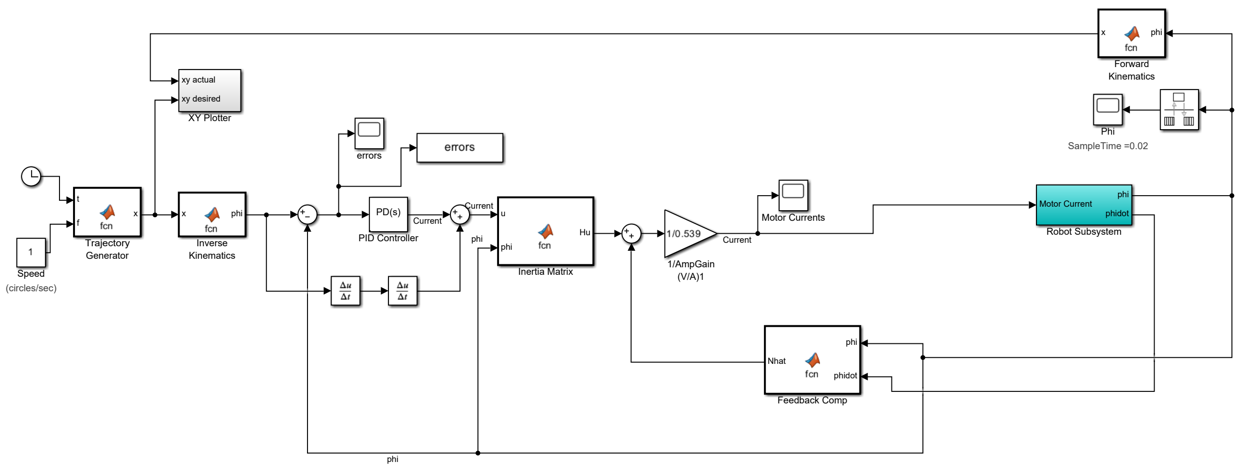




### 2.1.3 Tabular Results

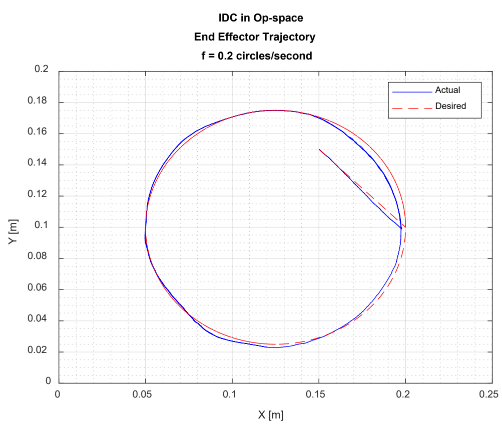
| | | f = 0.2 circles/sec | f = 1 circles/sec |
|---|---|---|---|
| **X** | Max [m] | 0.01 | 0.025 |
| | RMS [m] | 0.003 | 0.009 |
| **Y** | Max [m] | 0.004 | 0.028 |
| | RMS [m] | 0.002 | 0.009 |

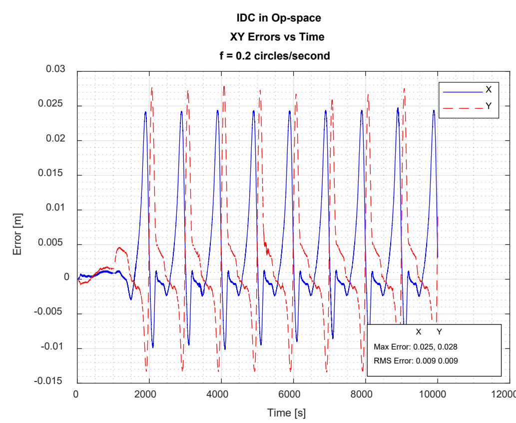
## 2.2 INVERSE DYNAMICS CONTROL IN OPERATIONAL SPACE

### 2.2.1 Simulink Model

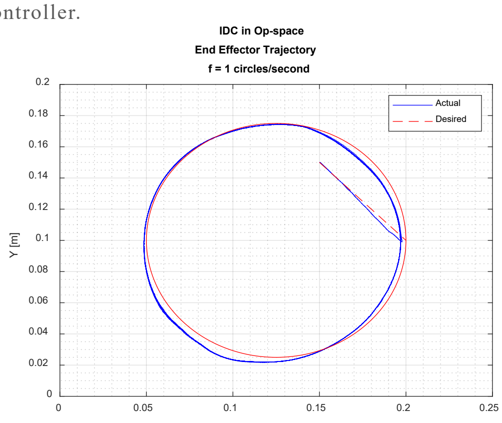


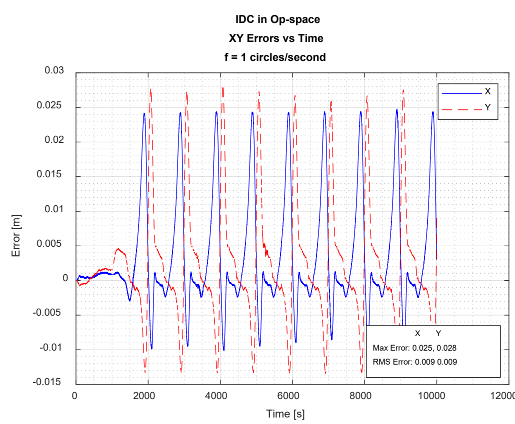

### 2.2.2 Plots




Note: I believe we accidentally saved the fast errors within the slow errors data file. We don't have time to run back and redo the lab, but our errors were consistent with our x-y trajectory, and lower in maximum magnitude and RMS values than the fast controller.



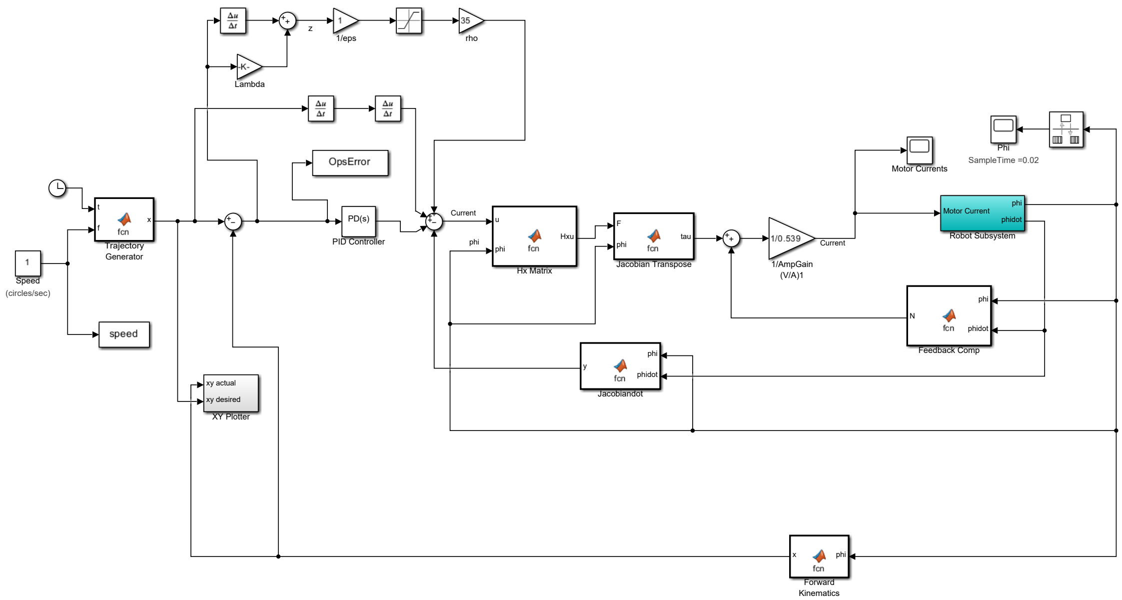



### 2.2.3 Tabular Results

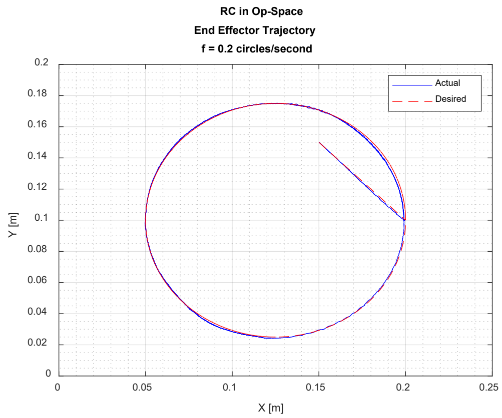
| | | **f = 0.2 circles/sec** | **f = 1 circles/sec** |
|---|---|---|---|
| **X** | Max [m] | 0.025 | 0.025 |
| | RMS [m] | 0.009 | 0.009 |
| **Y** | Max [m] | 0.028 | 0.028 |
| | RMS [m] | 0.009 | 0.009 |

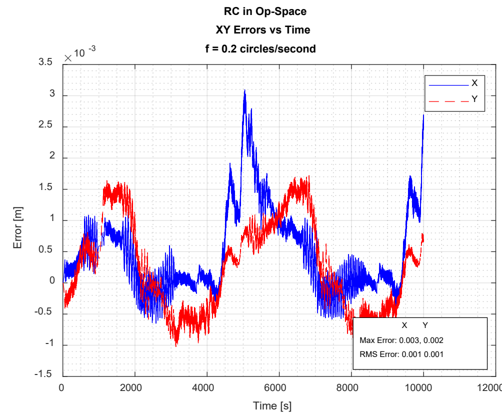
## 2.3 ROBUST CONTROL IN OPERATIONAL SPACE

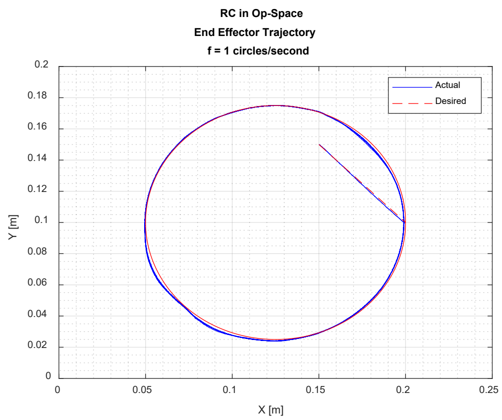
### 2.3.1 Simulink Model

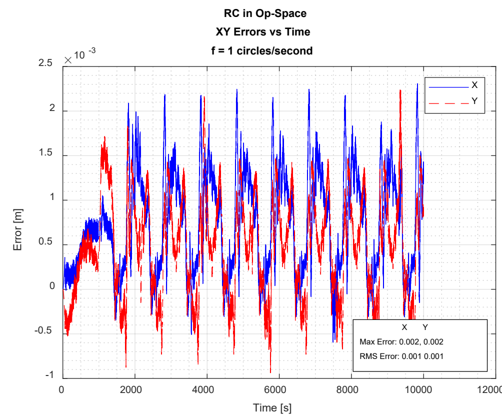


### 2.3.2 Plots




### 2.3.3 Tabular Results

| | | **f = 0.2 circles/sec** | **f = 1 circles/sec** |
|---|---|---|---|
| **X** | Max [m] | 0.003 | 0.002 |
| | RMS [m] | 0.001 | 0.001 |
| **Y** | Max [m] | 0.002 | 0.002 |
| | RMS [m] | 0.001 | 0.001 |

## 2.4 Comparison between Controllers in Section 2

The best controller in all scenarios was the robust controller in op-space. The errors at all speed were very low, and hardly differed between the fast and slow frequencies. The inverse dynamics controller also performed well and had a relatively smooth trajectory. Results indicated that there was no difference between fast and slow trajectories, but this may be due to a mistake in saving data. Either way, the inverse dynamics controller also outperformed the Jacobian transpose controller.

# IDC OS Functions

function Hu = fcn(u, phi)

```
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

Hu = H*u;
```

function Nhat = fcn(phi, phidot) Feedback Comp

```
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 = [0 -h ;h 0]*[phidot(1)^2;phidot(2)^2]; % centr torques
G = [G1;G2]; % gravity torques
F = [F1;F2]; % frictional torques

Nhat = V + G + F;
```

*Published with MATLAB® R2022b*

# Jacobian Transpose and Inverse Function Write-out

## Table of Contents



# Jacobian Transpose

function tau = fcn(phi, F)

```
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;
```

# Jacobian Inverse

function phi_err = fcn(phi, x_err)

```
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;
```

*Published with MATLAB® R2022b*

## Table of Contents



```
% IDC_OS_RC Function List
```

# Hx Matrix

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;
```

# Jacobian Transpose

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;
```

# Jacobian Dot

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;
```

# Feedback Comp

function N = fcn(phi, phidot) Feedback Comp

```
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 = [0 -h ;h 0]*[phidot(1)^2;phidot(2)^2]; % centr torques
G = [G1;G2]; % gravity torques
F = [F1;F2]; % frictional torques

N = V + G + F;
```

*Published with MATLAB® R2022b*