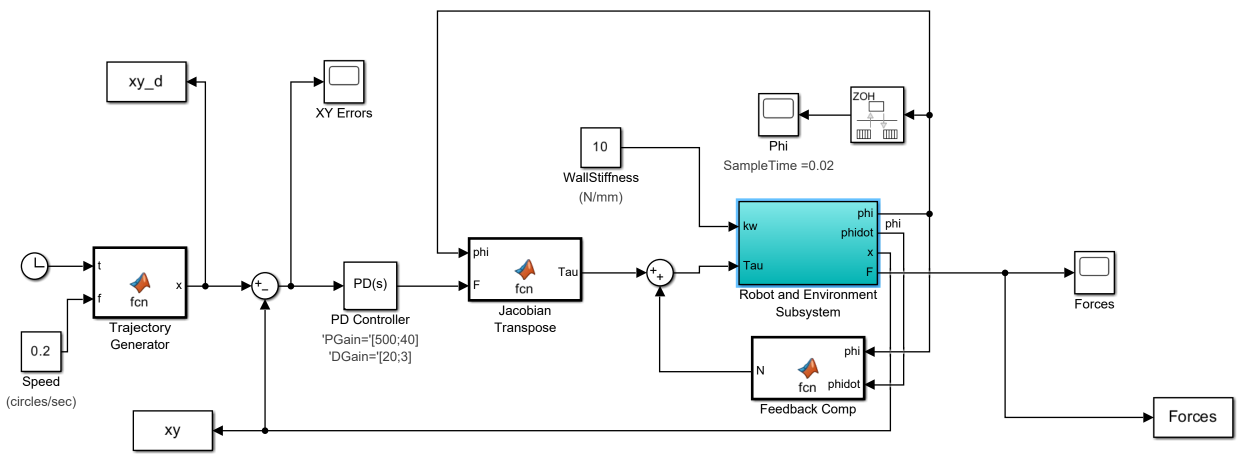
# PS8 Indirect Force Control

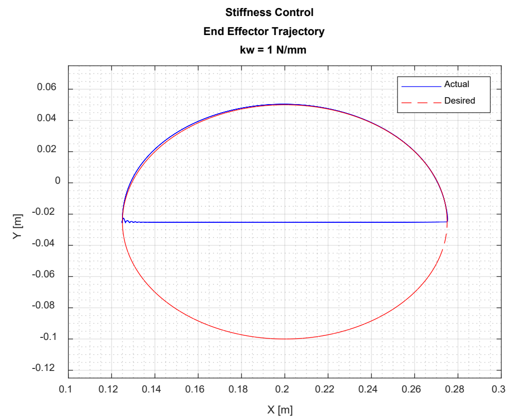
Will Graham

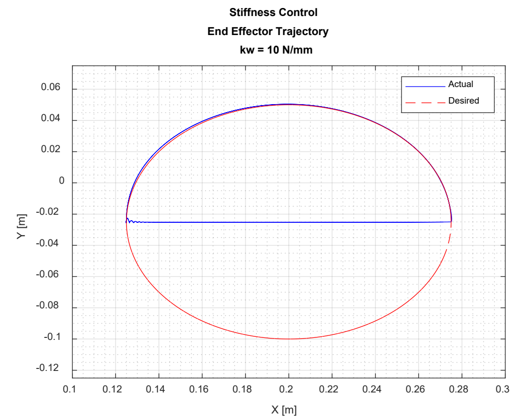
## 1 Stiffness/Damping Control

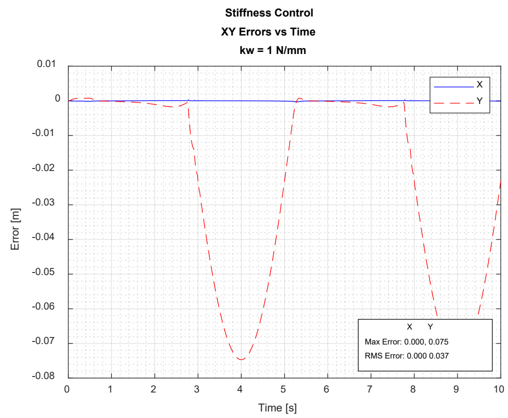
### 1.1 Simulink

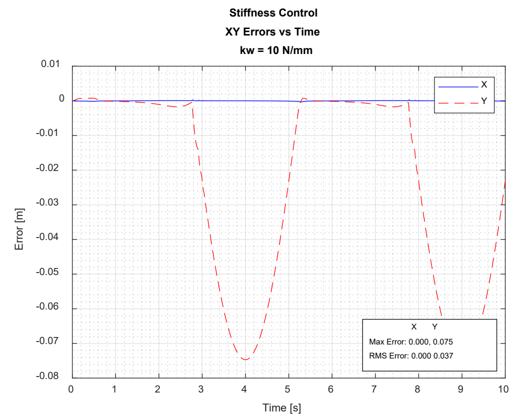


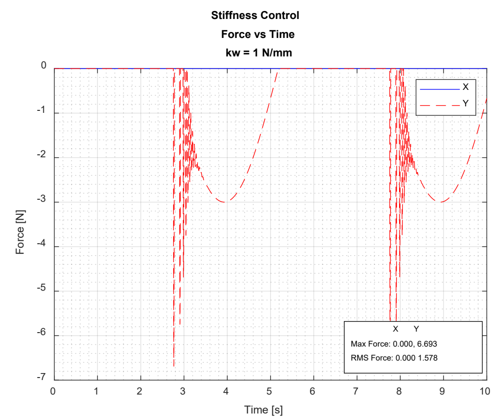
## 1.2 Results

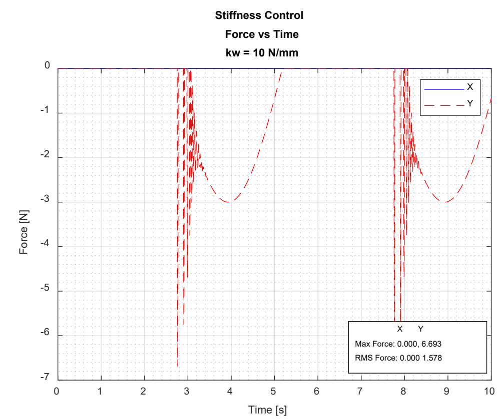


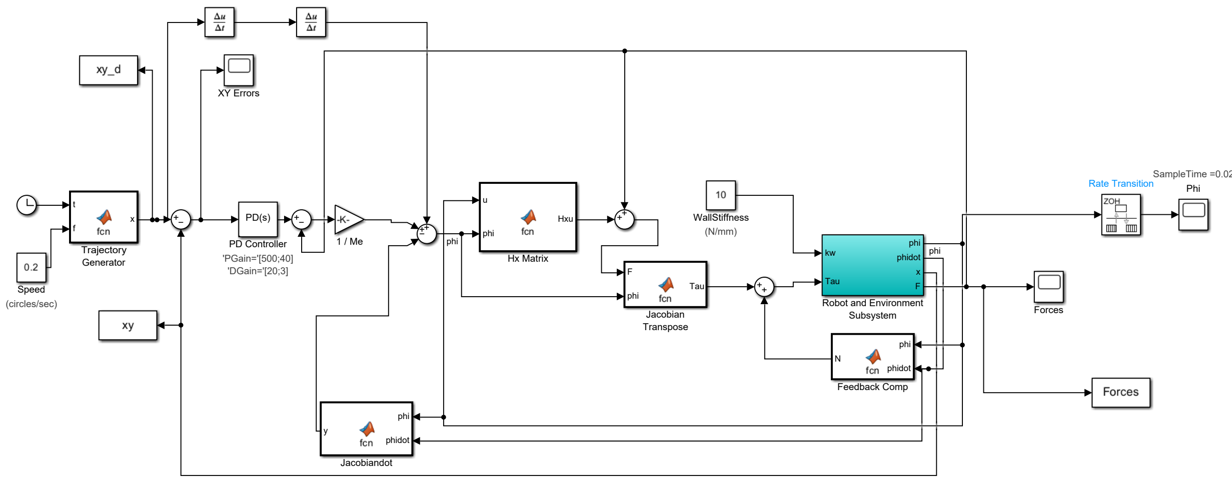




## 1.3 Controller Analysis

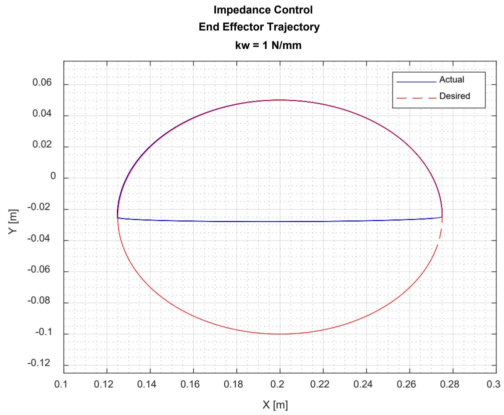
This tracker did exceptionally well in tracking the motion, but was prone to large jumps within the end effector forces. This unfortunately led us to have Forces larger than our desired 3 N maximum. If motors are ideal (no saturation) the initial force vibrations disappear. This is not realistic to physical systems, so this is not a great controller option when dealing with precise and force-sensitive operations.

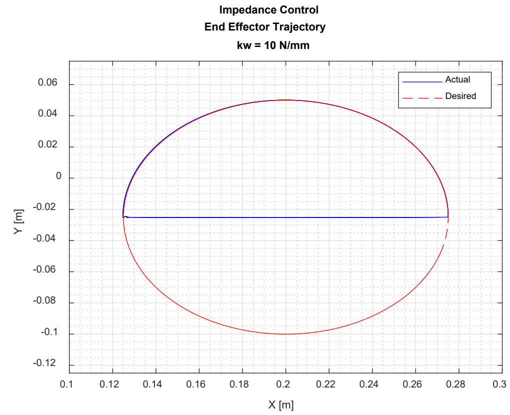
# 2 Impedance Control

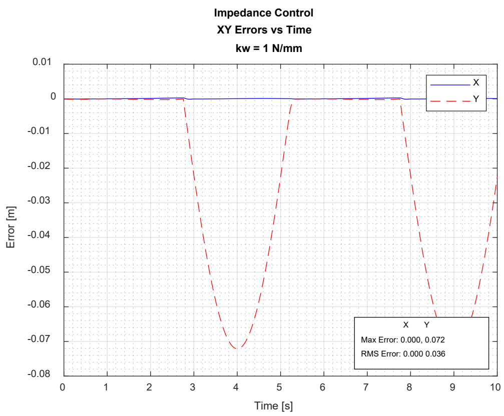
## 2.1 Simulink

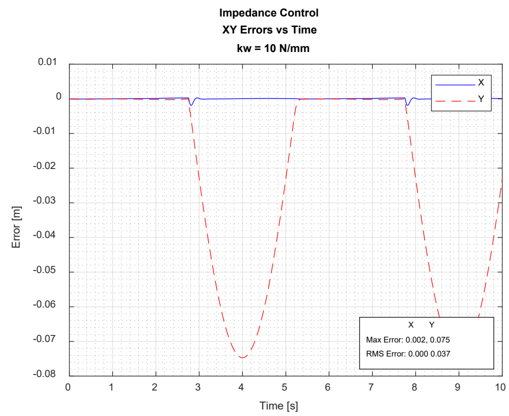


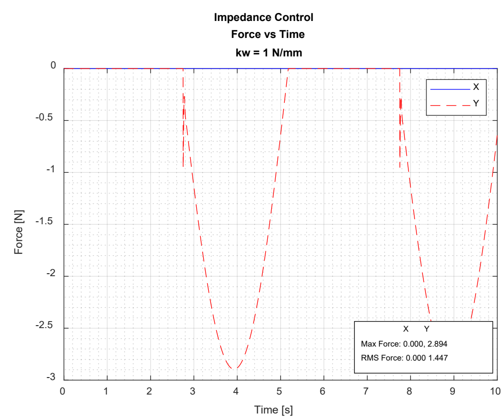
## 2.2 Plots

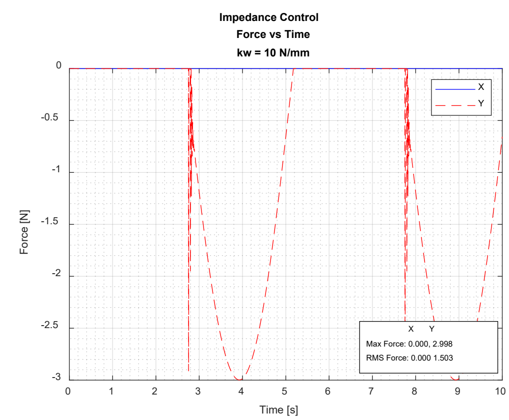


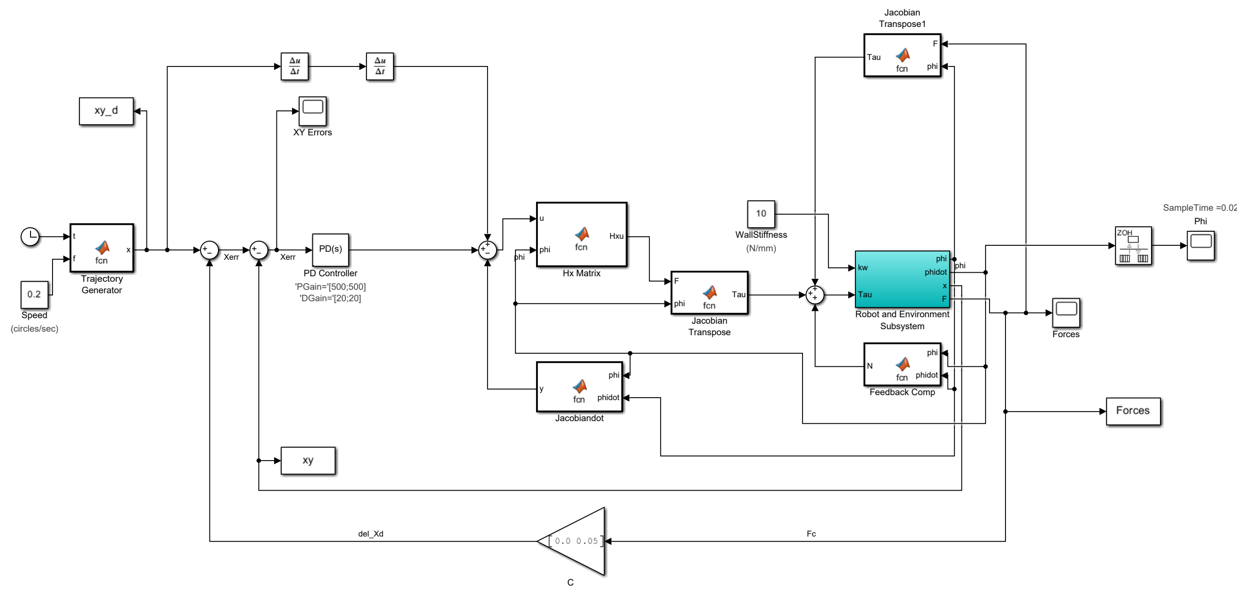




## 2.3 Controller Analysis

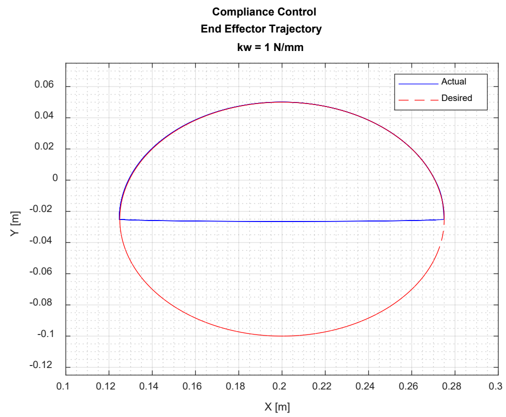
Impedance control did remarkably well. This controller stayed below the target force maximum, and was stable in stiffer environments. It was relatively simple to implement, and is a solid option for force control.

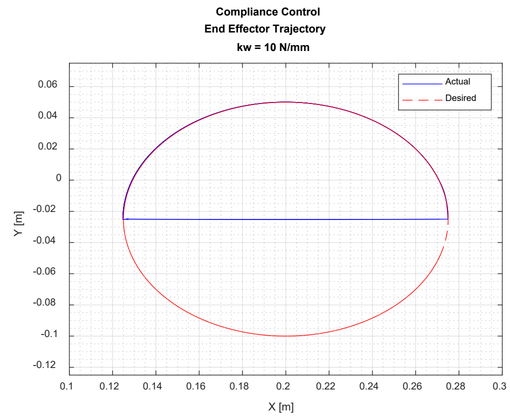
# 3 Compliance Controller

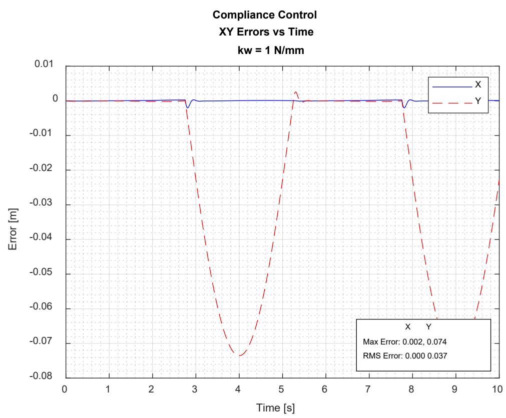
## 3.1 Simulink Model

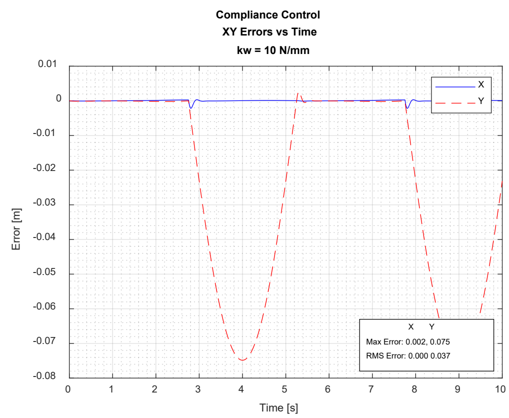


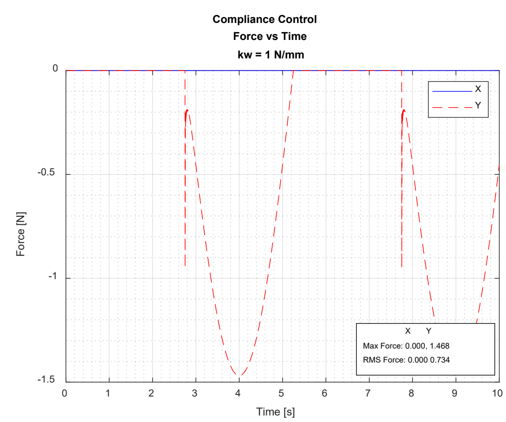
## 3.2 Plots

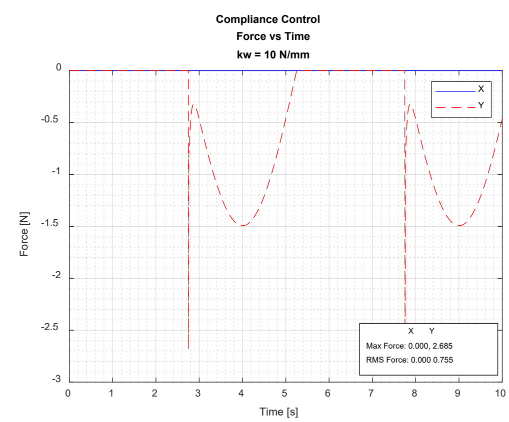


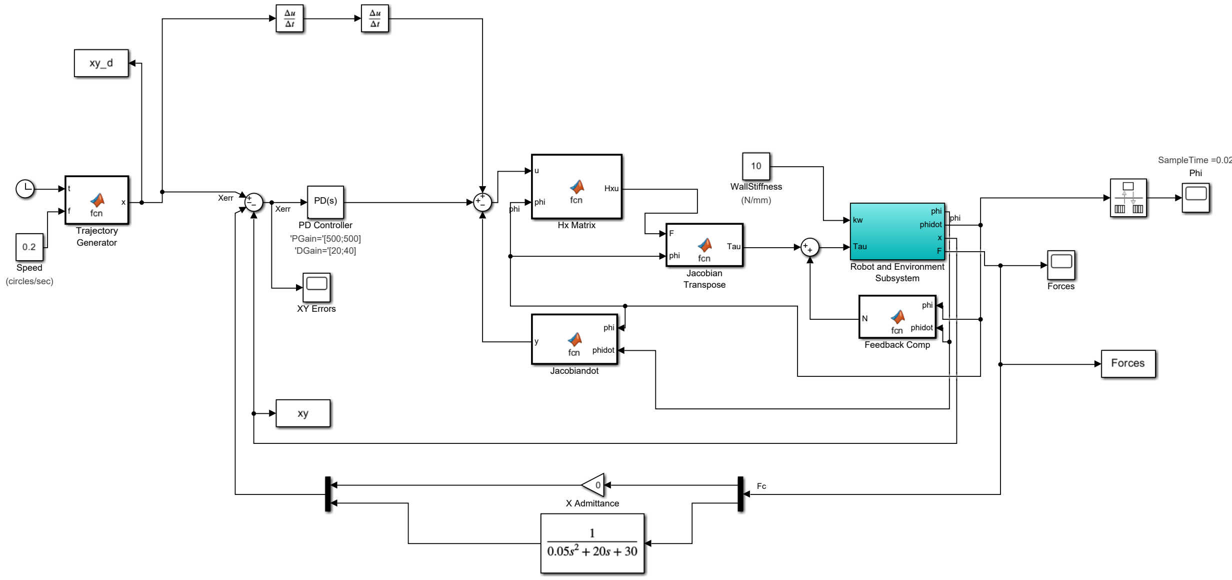




## 3.3 Controller Analysis

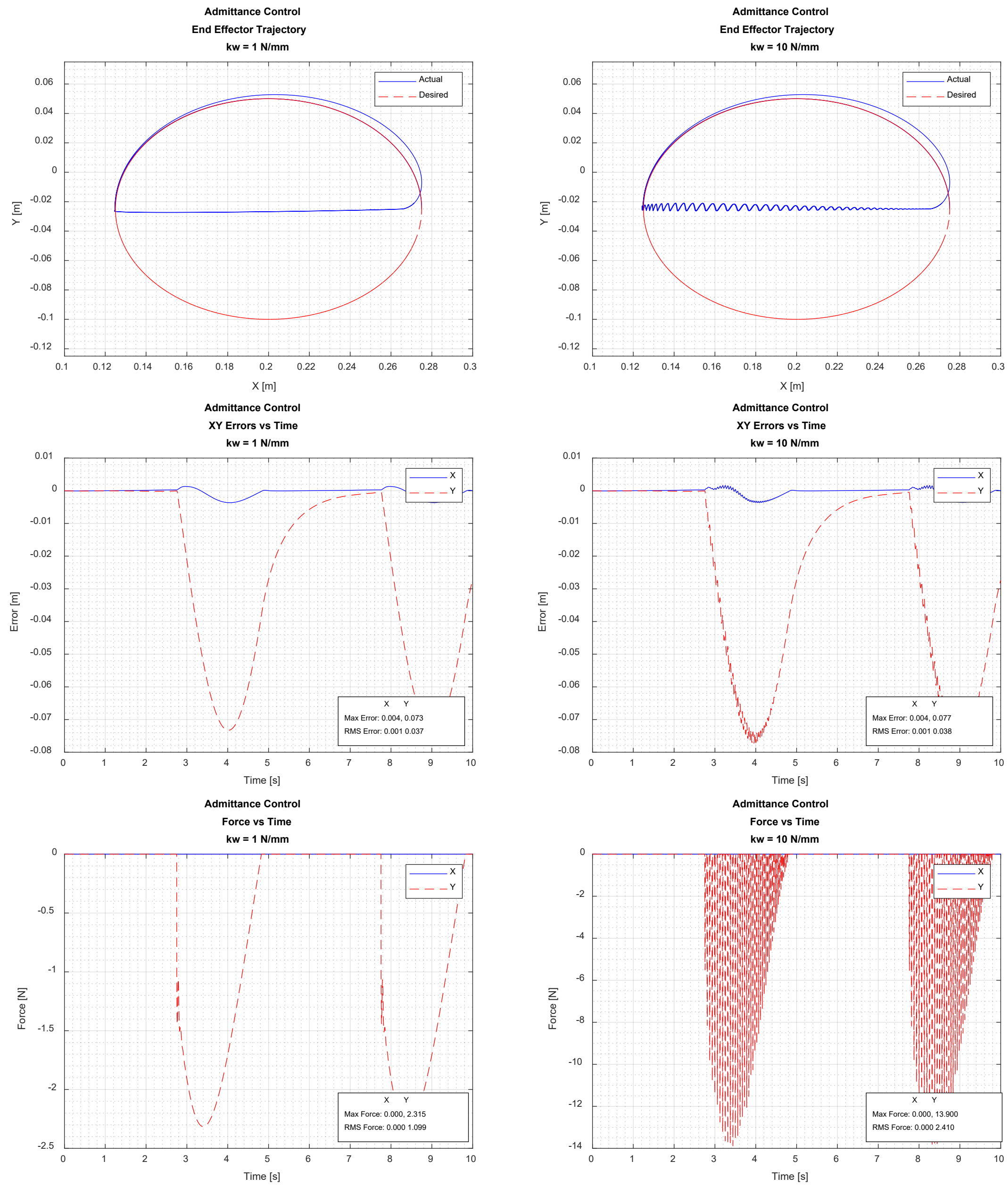
Compliance Control was even more accurate than impedance control, but still had high initial impact forces that seem to get much worse with stiffer environments. This controller is more complicated than impedance control but had better results. I'd recommend this for high-precision work in stiff environments.

# 4 Admittance Control

## 4.1 Simulink Model



## 4.2 PLOTS




## 4.3 CONTROLLER ANALYSIS

Admittance control had the best force control by far, with hardly any initial impact force spikes. However, it struggled with trajectory tracking, with lower accuracy than of the other controllers. Additionally, admittance control went unstable very quickly with a stiffer environment. This is because the stiffness of the environment is built within the transfer function. This would be perfect for a known environment, where the end effector is in constant contact with its surroundings, and the surroundings are fragile/sensitive to force.

## Table of Contents



# Inertia Matrix

function Hxu = 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


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) This block supports an embeddable subset of the MATLAB language. See the help menu for details.

```
a1=0.15;
a2=0.15;

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 = J'*F;
```

# Feedback Compensation

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

N = V + G + F;
```

# Jacobiandot

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;
```

*Published with MATLAB® R2022b*