ASEN 2003: Lab 6
Rotary Position Control

Samuel Felice*, Abdulla Alameri†, Samuel Firth‡, Nicholas Boender§
University of Colorado Boulder, Boulder, CO, 80303

Date Submitted: 04/22/2019

This report gives a brief overview of the principles of control response theory in application. A foundational understanding of control systems and the differential equations of motion to which they are applied is developed first. Then, two experiments are developed to show how control systems are actually implemented, and the error that arises when dealing with them. These experiments and the data taken from them are then compared to their respective theoretical models. It is shown that the second experiment in question, though nearly identical to the first save for one distinct feature, is vastly more complicated in its analysis. The control systems presented in this paper can be solved with relatively low level analysis, as the focus of this report is on the analysis of control system behavior operating with differing gain values, as opposed to the most efficient way of solving one.

Contents

I Theory **2**

I.A Control Block Diagrams 4

II Experiment **4**

III Results & Analysis **7**

III.A Rigid Bar 7

III.B Flexible arm 12

III.C Rigid arm Vs. Flexible arm 19

IV Conclusions and Recommendations **20**

V Acknowledgements **21**

VI Appendix **22**

VI.A Team Member Participation Table 22

VI.B MATLAB 22

Nomenclature

θ_D	=	desired position (angle) of arm $[rad]$
Θ_D	=	desired position (angle) of arm in Laplace domain $[rad]$
θ_L	=	measured (actual) position (angle) of arm in time domain $[rad]$
Θ_L	=	measured (actual) position (angle) of arm in Laplace domain $[rad]$
V_{in}	=	input voltage $[V]$
$\mathbf{V}_{in}$	=	input voltage in Laplace domain
K_m	=	proportional motor constant $[V/rad/s = N - m/A]$
K_g	=	gear ratio [-]
J	=	moment of inertia of the whole system $[kg - m^2]$
R_m	=	motor resistance [Ohms]
$K1/Kp$	=	Proportional control gain value for hub angle
$K2$	=	Proportional control gain value for tip displacement
$K3/Kd$	=	Derivative control gain value for hub angle
$K4$	=	Derivative control gain value for tip displacement

I. Theory

The first system in question is a simple rigid arm driven by a motor. The only forces acting on the arm are the torque supplied by a motor in the system controller, and friction. For the purposes of this lab, friction is negligible. The equation of motion for the system, then, is found in equation 1, where J, B, and M_L are, respectively, the moment of inertia of the arm about the shaft, a dampening coefficient, and any external moments that disturb the arm.

$$J\dot{\omega}_L + B\omega_L + M_L = M_0 \tag{1}$$

In equation 1, M_0 represents the moments applied to the rigid arm by the motor. The goal in this lab is to design a control law, $U(t)$, such that $M_0 = U(t)$, and the rigid arm behaves according to some restrictive parameters. In this lab, the goal is to bring the rigid arm through a desired angle θ_L as quickly as possible while overcoming the influence of inertia. The control law governing such motion, then, can be modeled as appears in equation 2.

$$V_{in} = K_{p\theta}\left(\theta_D - \theta_L\right) + K_{D\theta}\left(\dot{\theta}_D - \dot{\theta}_L\right) \tag{2}$$

This function relates the desired input voltage, which is proportional to the torque applied by the motor using the relationship shown in equation 3.

$$M_0 = K_g K_m I_m = \frac{K_g K_m}{R_m} \left(V_{in} - K_g K_m \omega_L \right) \tag{3}$$

After some algebraic manipulation, equation 1's Laplace transform is calculated, revealing the open loop transfer function for the rigid arm relating input voltage to angular position, shown in equation 4.

$$\frac{\Theta_L}{\mathbf{V}_{in}} = \frac{K_g K_m / (J R_m)}{s(s + K_g^2 K_m^2 / J R_m)} \tag{4}$$

Solving this expression for Θ_L yields:

$$\mathbf{V}_{in} = \frac{s(s + K_g^2 K_m^2 / J R_m)}{K_g K_m / (J R_m)} \Theta_L \tag{5}$$

which can be substituted into the Laplace transform of equation 2:

$$\mathcal{L}(V_{in}) = \mathbf{V}_{in} = K_{p\theta} \left(\Theta_D - \Theta_L \right) + s K_{D\theta} \Theta_D = \frac{s(s + K_g^2 K_m^2 / J R_m)}{K_g K_m / (J R_m)} \Theta_L \tag{6}$$

By expanding and combining like terms, the closed loop transfer function from Θ_D to Θ_L is:

$$\frac{\Theta_L}{\Theta_D} = \frac{K_{p\theta} K_g K_m / (J R_m)}{s^2 + \left(K_g^2 K_m^2 / (J R_m) + K_{D\theta} K_g K_m / (J R_m) \right) s + K_{p\theta} K_g K_m / (J R_m)} \tag{7}$$

In order to select the gains for both the rigid control arm and the flexible one, Matlab scripts were used.

In the case of the rigid arm, the only gains present in the closed loop transfer function are K_D and K_P - derivative and proportional gain, respectively. A range of potential values was given, where $-1.5 \leq K_D \leq 1.5$ and $-2 \leq K_P \leq 50$. ζ and ω_n, as well as settling time, overshoot, and V_{in}, were calculated over 10,000 pairs of K_D and K_P in the given ranges. Then, pairs which yielded results for each of the aforementioned values within their bounds were returned. The best of these pairs was selected for use in the final control law.

In the case of the flexible arm, the control law must account for the "wobble" of the arm after the rigid arm would have stopped. The updated control law is show below in equation 8.

$$V_{in} = K_{p\theta} \left(\theta_D - \theta_L \right) + K_{D\theta} \left(\dot{\theta}_D - \dot{\theta}_L \right) + K_{pd} \left(d_D - d \right) + K_{Dd} \left(\dot{d}_D - \dot{d} \right) \tag{8}$$

Here, d represents the "tip deflection" of the flexible arm, or how much the tip of the arm is "bent" away from the equilibrium state. Given the presence of 4 gains, a process similar to the one performed for the rigid arm would be far too computationally demanding, requiring 100^4 calculations of several restrictive parameters (as with the rigid arm). Instead, only 6 values for each gain are looped through, giving only 6^4 combinations. This is obviously a less

accurate method than the one performed on the rigid arm, though achieving a higher degree of accuracy (without a more powerful computer) would require the use of methods in complex analysis such as pole placement. Such methods are beyond the scope of this lab.

A. Control Block Diagrams

Control block diagrams are useful for visualizing the process by which a control system drives a system from a measured state to a desired state. In the scope of this lab, an arm is driven from a measured angular position to a desired angular position (between 0.1 and 0.7 radians from starting position.) Below is the control block diagram of the rigid arm system.

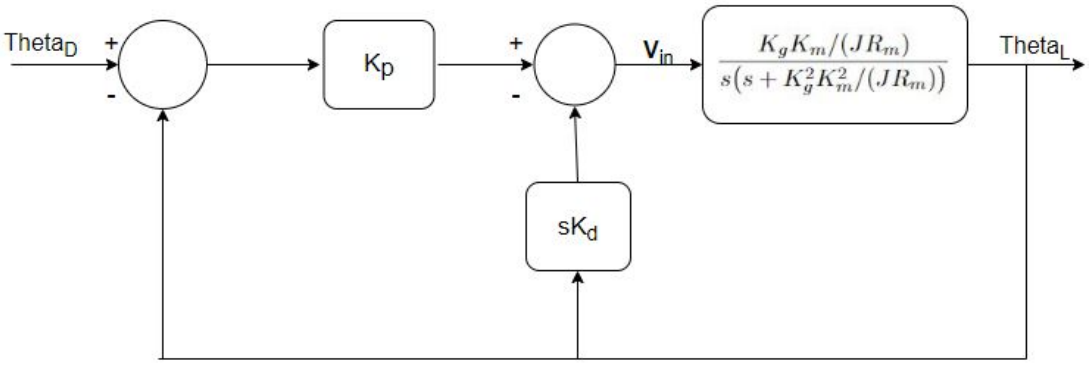

Fig. 1 Control block diagram of the rigid arm system

The term in the rightmost box is the open loop transfer function relating input voltage to position in the Laplace domain (equation 4) and was the starting point of the derivation of the closed loop transfer function from Θ_L to Θ_D (equation 4 to equation 7).

II. Experiment

The operation and data collection of both the rigid arm system and the flexible arm system are effectively identical; the flexible arm system has additional strain gauges mounted on the tip of the arm that provides tip deflection data.

A desired input angle is sent from the labVIEW vi to the system. The system has a mounted myRIO microcontroller with an inbuilt digital to analog converter (DAC), which converts the incoming command for voltage to the act of the commanded voltage being drawn from the lab station and sent to the DC motor. The motor utilizes a gearbox with gear ratio k_g to rotate the output shaft, and thereby the arm to the desired angular position. Potentiometers mounted on the output shaft (and the aforementioned strain gauges in the case of the flexible arm) sense positional data and send that data back through the myRIO microcontroller to the labVIEW vi, where it is displayed on-screen in real time. This process can be visualized with the aid of a functional block diagram.

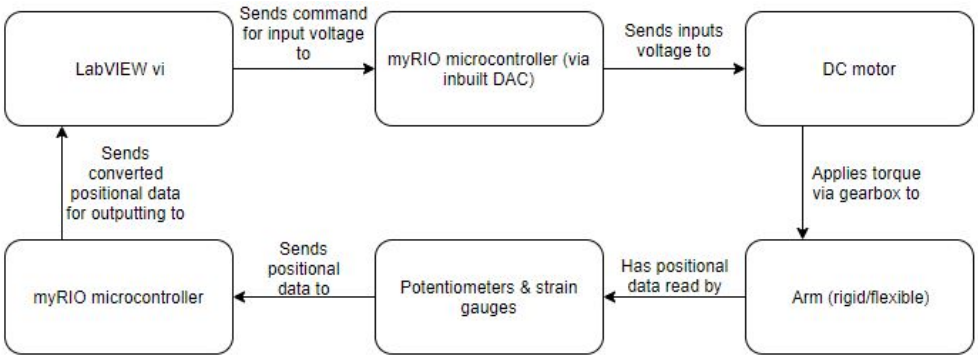

Fig. 2 Functional block diagram of the rigid/flexible arm system

Below are the two system that were used for this lab. Both of them utilize the same base, which contains a myRIO microcontroller and a DC motor. The body draws power from the lab station. The only physical difference between the two systems is the make-up of the arm. One utilizes a solid aluminum bar, which is modeled as a solid body, and the other utilizes a 12 inch metal ruler with strain gauges on the tip. The flexible arm, not being suitably modeled as a solid body, was modeled as a two DOF spring-mass-damper system. The rigid arm is much more controllable than the flexible arm; as the flexible arm comes to a halt, the arm strains and begins to wobble, resulting in residual tip vibrations. Careful tuning was needed in order to reduce these tip vibrations below the acceptable threshold.

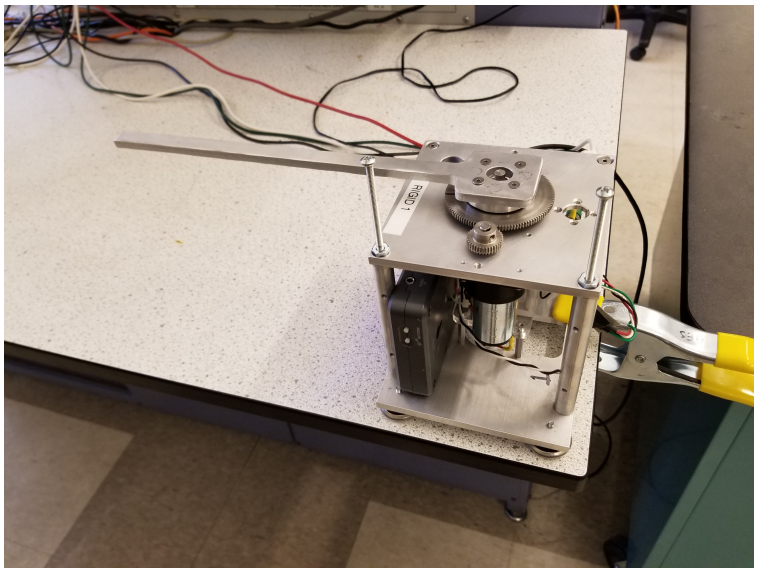

Fig. 3 Rigid arm system; the arm consists of a solid aluminum bar, which can be modeled accurately as a solid body

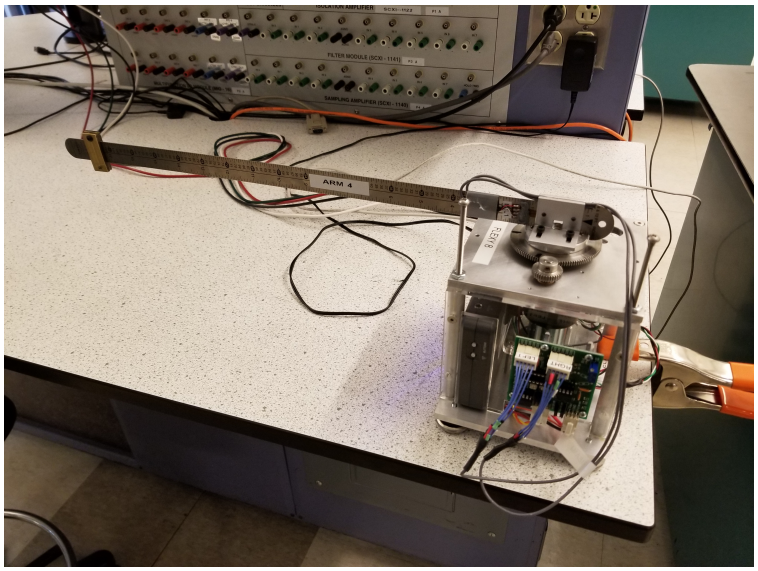

Fig. 4 Flexible arm system; the arm consists of a 12 inch, metal ruler with stain gauges mounted on the tip

Data collection began with the rigid arm system. Connections between the lab station and the unit were confirmed and the lab station power supply was turned on. The labVIEW vi was run using the arrow in the upper left corner of the screen and the correct unit was selected from the drop down menu. It was important that the correct unit was selected as each unit responded to commands slightly differently and these variations in response were normalized based on the unit. Once the vi was running, the magnitude of the step function (i.e. the half of the desired angular position) was input and the gain values selected. Only half of the desired angle was input, as the step function would go from negative to positive. Thus, inputting half of the desired angle resulted in a rotation of the full angle.

Proportional and derivative gain value selection is discussed in detail in Section I. Theory.

Initially, only proportional gain values were used. This resulted in quick rotation, but the system consistently undershot the desired angle. Once values for a derivative gain were entered, the system was able to get much closer to the desired angle. Slightly negative derivative gain values (Kp -0.01) resulted in faster motion than without without the unstable behavior characteristic of larger negative derivative gain values. Larger positive derivative gain values were also tested and are discussed in detail in Section III. Results Analysis.

Once the rigid arm testing had concluded, testing on the flexible arm began. A similar selection process as the rigid arm was used, and is also discussed in Section I. Theory. Initially, only K1 values were used. This resulted in tremendous overshoot and residual vibrations. Additional gains were added while maintaining the previously tested values. For in-depth results, see Section III. Results Analysis.

During one of the trials, rigid arm unit 8 began acting erratically. There was substantial noise in the angular position data, resulting in an almost unreadable screen. The resulting data was deemed unsuitable and scrapped. Bobby took away the faulty unit for maintenance and data collection continued on a different unit.

III. Results & Analysis

A. Rigid Bar

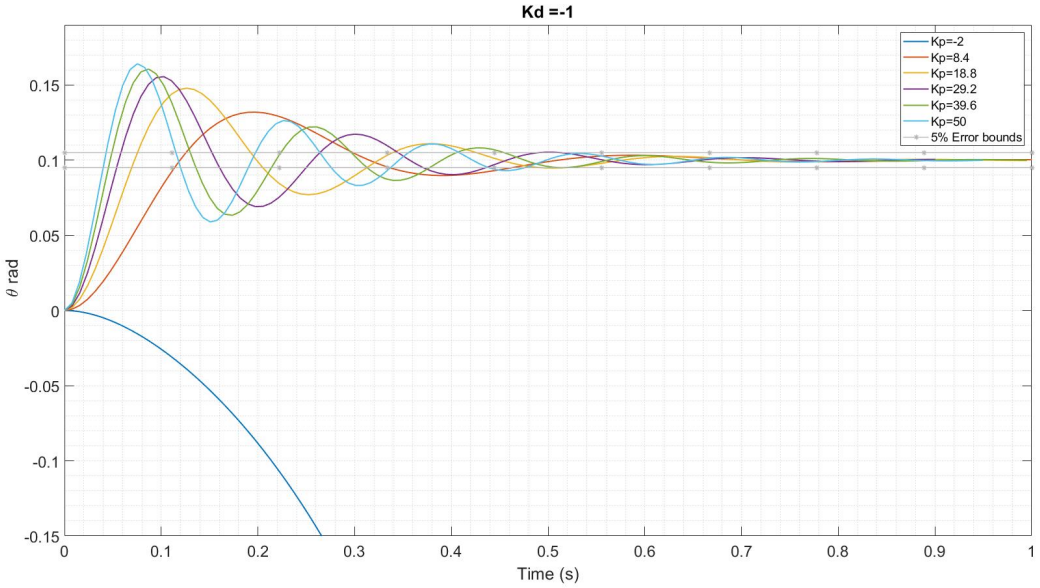

Fig. 5 Step Size 0.1, Kd=-1

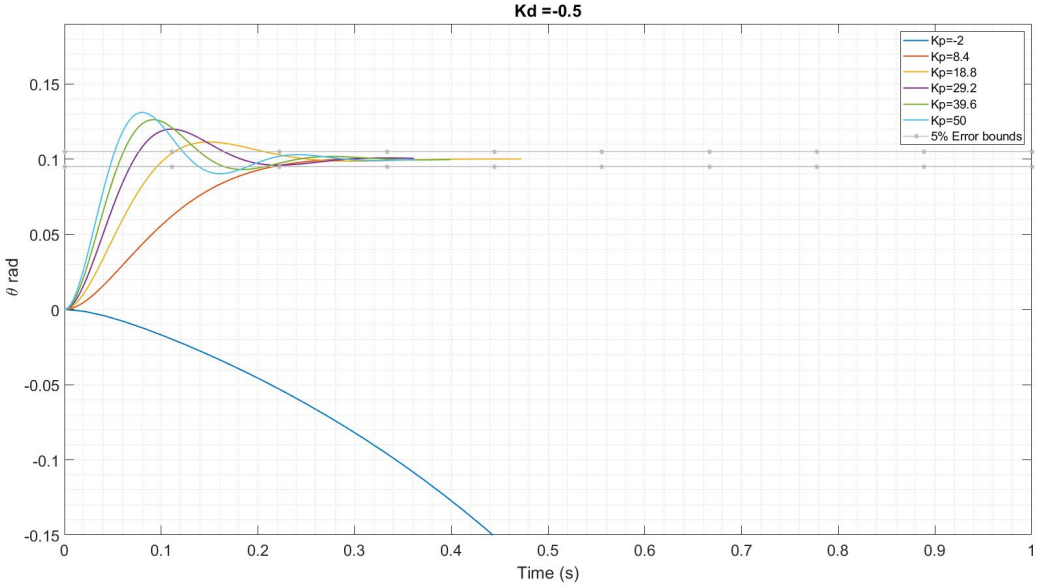

Fig. 6 Step Size 0.1, Kd=-0.5

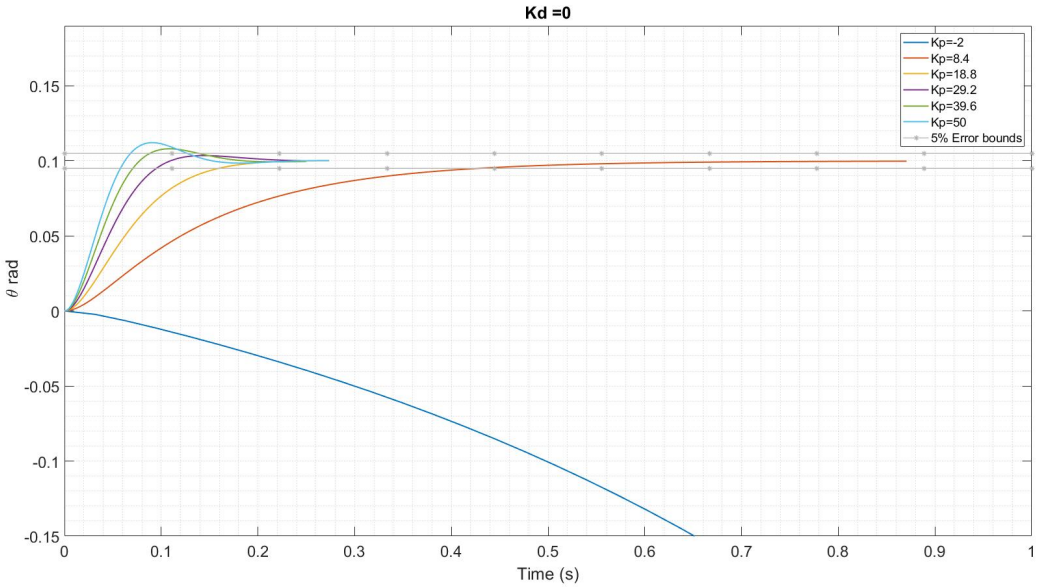

Fig. 7 Step Size 0.1, Kd=0

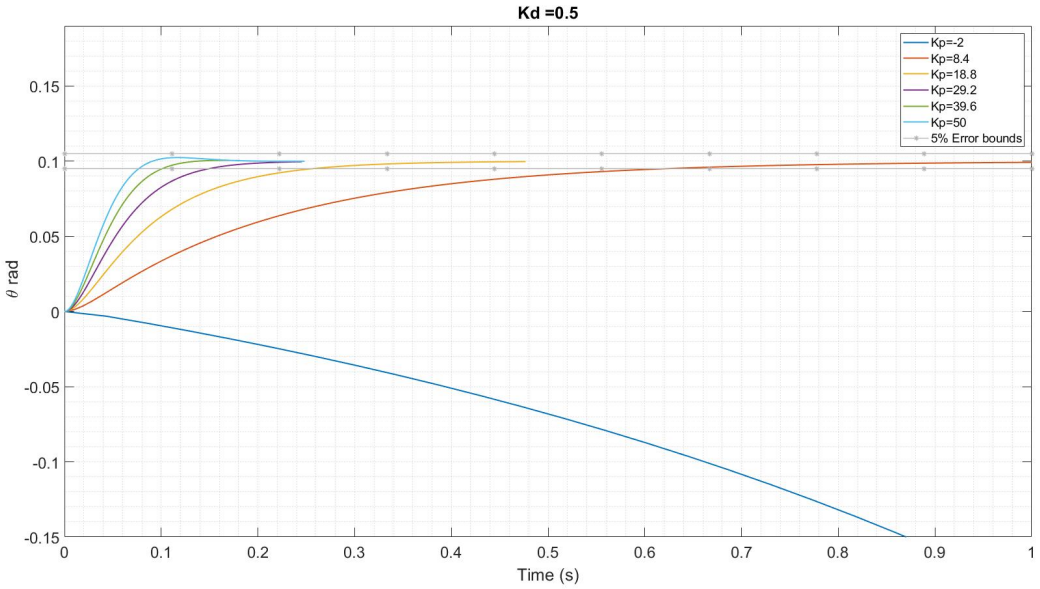

Fig. 8 Step Size 0.1, Kd=0.5

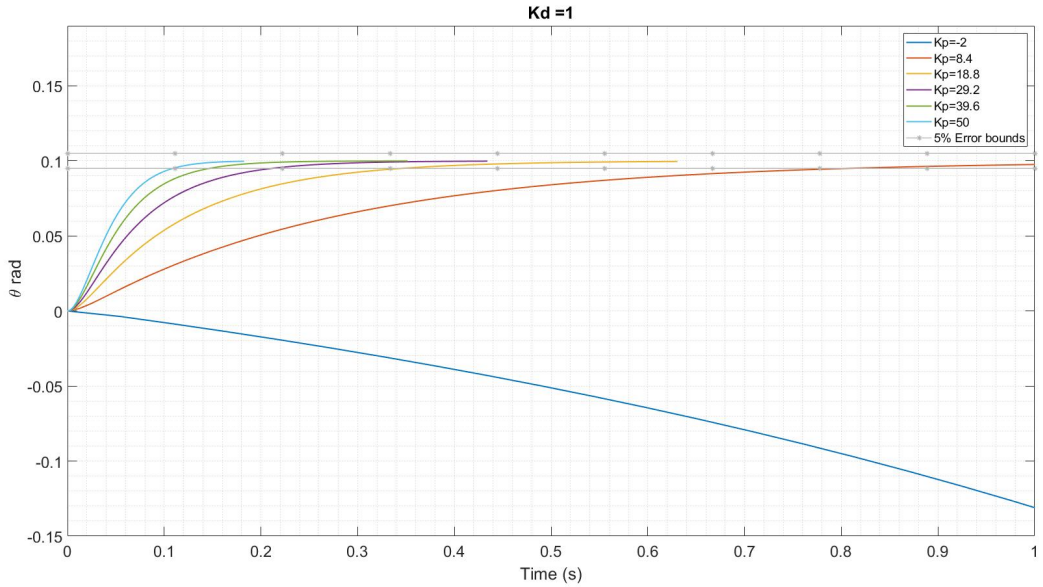

Fig. 9 Step Size 0.1, Kd=1

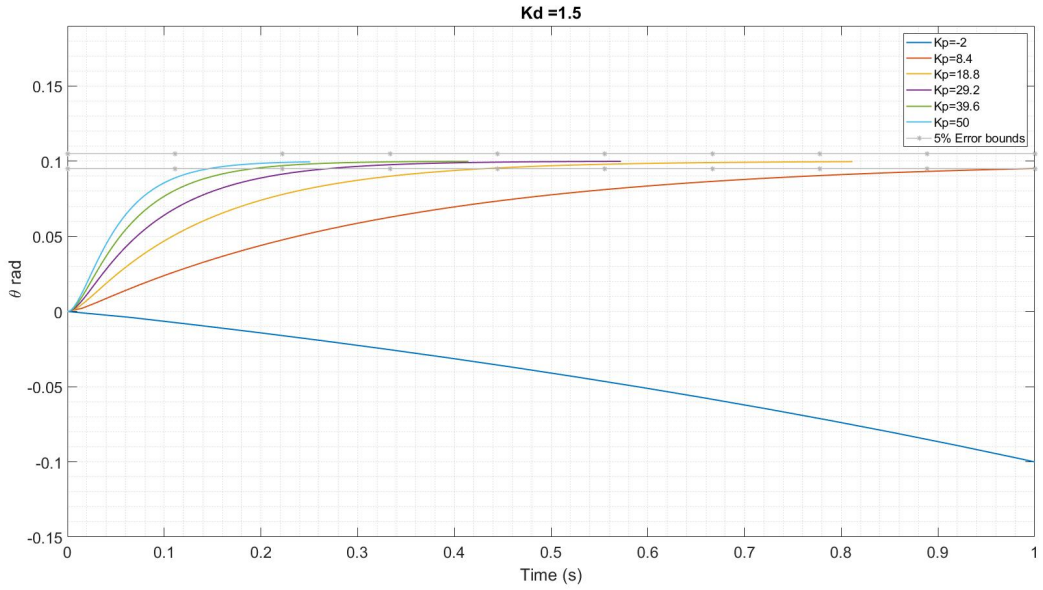

Fig. 10 Step Size 0.1, Kd=1.5

The plots above represent a simulation of potential outcomes with a 0.1 step size and varying derivative and proportional control (Kd and Kp) values. From the simulation it was determined that Kd would need to fall between -0.5 and 0.5 in order to minimize overshoot and underdamping of the system. From the simulation it can be seen that in general larger Kp values caused the 5% settling time to decrease but could cause some significant overshoot if paired with low Kd values. Using this information the three tests seen below were run. For each of these test cases the vertical

red line represents the 5% settling time of the simulation and the vertical blue line represents the 5% settling time of the experimental data.

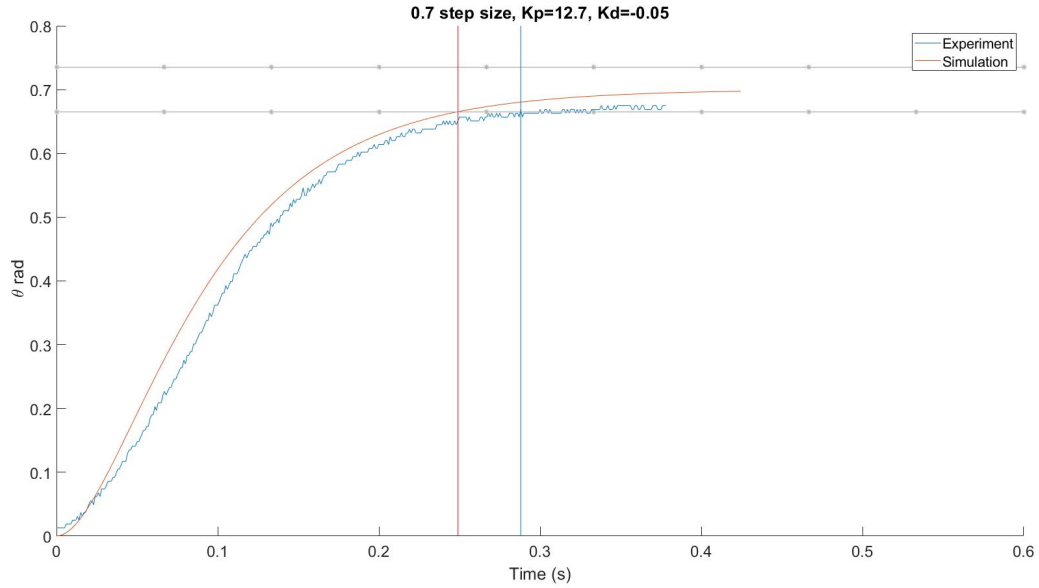

Fig. 11 Step Size 0.7

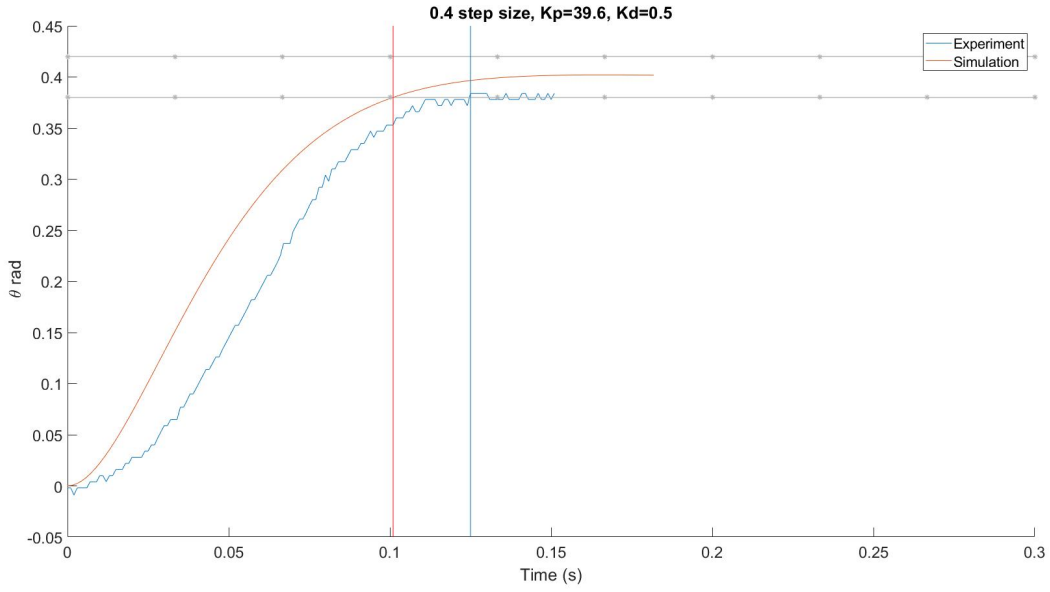

Fig. 12 Step Size 0.4

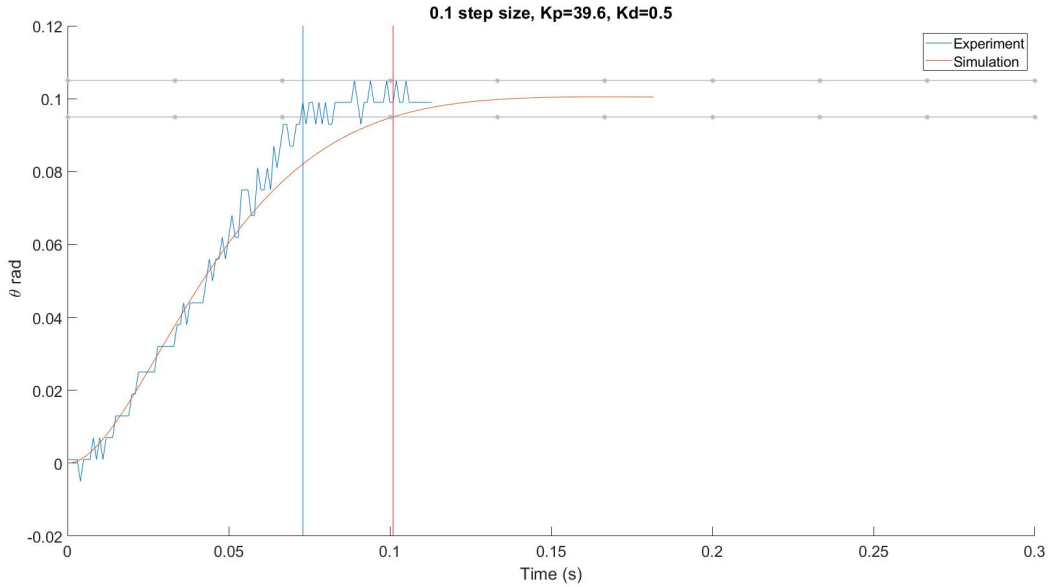

Fig. 13 Step Size 0.1

The test with a 0.7 step size, seen above, was run with a Kp value of 12.7 and a Kd value of -0.05 to determine if it was reasonably possible to get a 5% settling time of less than 0.15 seconds with such a large step size while also remaining within the voltage limit. The test seen above was not the only one run, but was the test with the best results. As can be seen the model itself predicted a roughly 0.25 second 5% settling time while the experiment was closer to 0.3 seconds.

The next test was run with a 0.4 step size, Kp value of 39.6, and Kd value of 0.5 and succeeded in getting a 5% settling time less than 0.15 seconds. The model predicted a settling time of 0.1 seconds and the experiment ended up having a settling time of roughly 0.125 seconds. The final test was run with a 0.1 step size and the same Kp and Kd values as the second test to see if an even lower settling time could be produced. Interestingly, for this test the model predicted a greater settling time of 0.1 seconds than the experiment actually produced, about 0.075 seconds.

The difference between the model and the experiment in all the cases can most likely be attributed to friction in the system and the interaction of the motor and electronic power. Friction most likely acts as unaccounted for derivative control, slowing the movement of the rigid bar as it swings and increasing the settling time. Friction however it probably negligible when compared to the error brought about by the electronics and motor of the system. The model assumes an ideal system whereas the motor responds ideally to all commands given. However, motors lose effectiveness when voltages are low and so the change in the position of the bar will not be the same as what the model is assuming. This can be seen in the 0.7 step size and 0.4 step size graphs where the slope of the experimental position line is much less than the model early on. The third experiment with a step size of 0.1 also seems most effected by this, as the experiment seems to linearly approach the step then quickly flatten out whereas the model predicts a much smoother curve.

B. Flexible arm

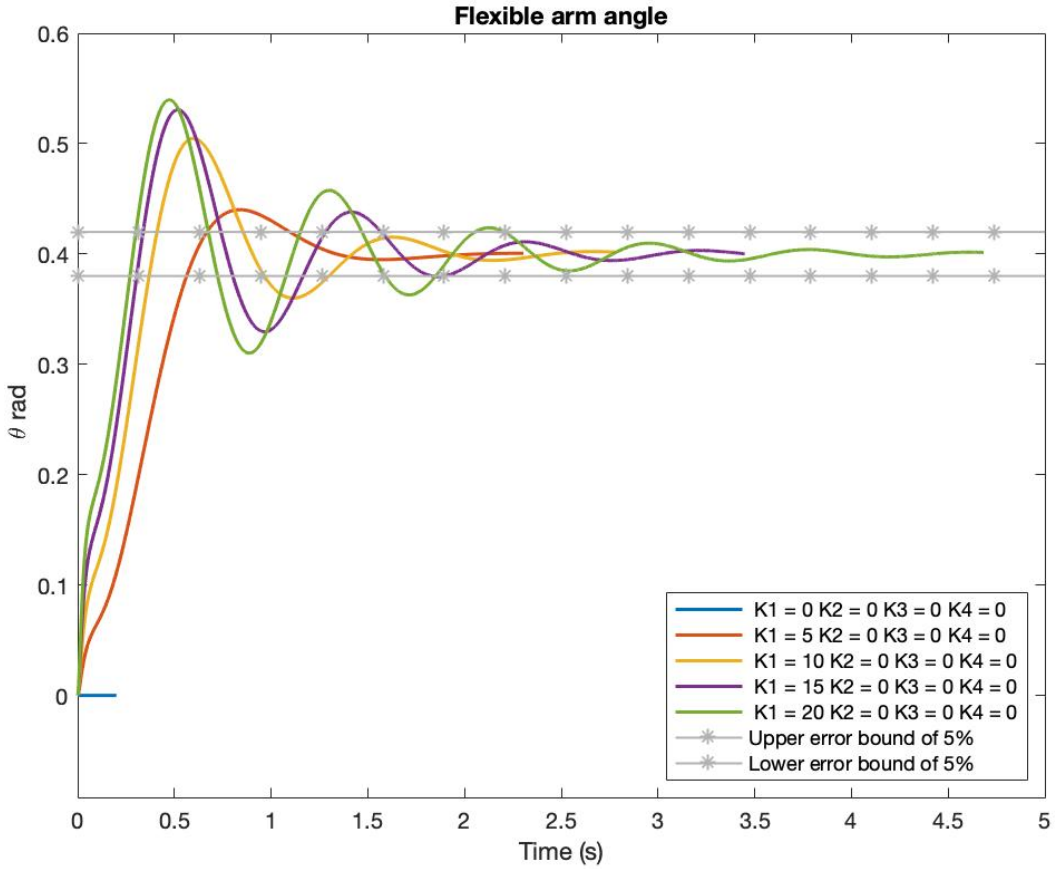

Fig. 14 Theoretical model for the flexible arm hub angle when only the proportional control (K1) (for the hub angle) is changed.

The flexible arm was a little bit more challenging to analyze. With a knowledge of dynamics, cantilever beams and arms can be modeled as a spring-mass-damper system. In other words, the flexible arm can be modeled as if it is two spring-mass-damper systems joined together. However, the flexible arm has two elements which are controlled by proportional derivative (PD) controls. To examine the effect of the K1 controller, different K1 values where examined while K2, K3, K4 were held to 0. This was done via picking random K1 values through the lab station vi. It was observed that as K1 was increased, both overshoot and settling time within the 5% error bound increased also. The same trend was observed for the tip displacement that was measured via a potentiometer. This matched the theoretical simulations that can be seen in figures 14 and 14.

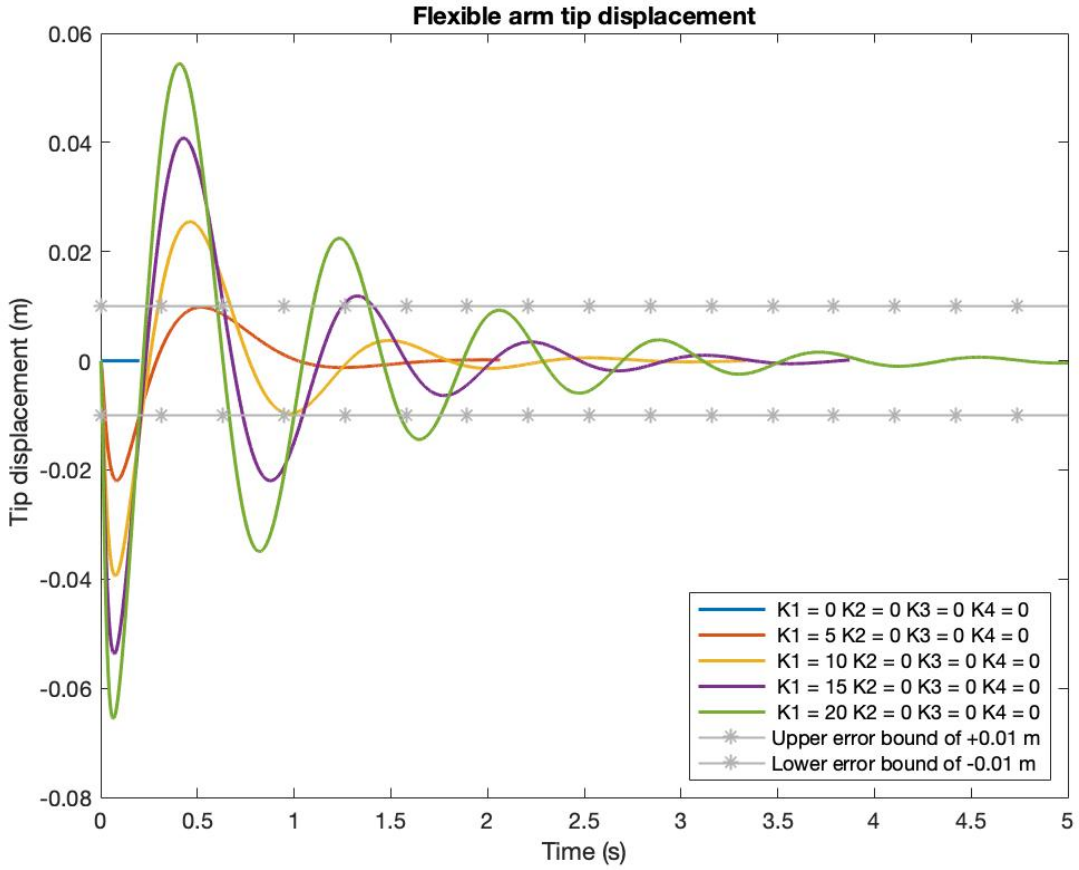

Fig. 15 Theoretical model for the flexible arm tip displacement when only the proportional control (K1) (for the tip displacement) is changed.

Next, gain values were considered for K2 (proportional control gain value for tip displacement) while maintaining the K1 gain values. For the hub angle measurements, it was observed that the settling time increased relative to when K2 was set to 0, as did the overshoot. This also matched the theoretical predictions. For instance, if one considered the red line in figure 14 for when K1 = 5, one can clearly see that when a K2 = -37.5 as shown in figure 16 then the overshoot increases from 0.434 rad to 0.467 rad, and the settling time increases from 1.2 seconds to 1.4 seconds.

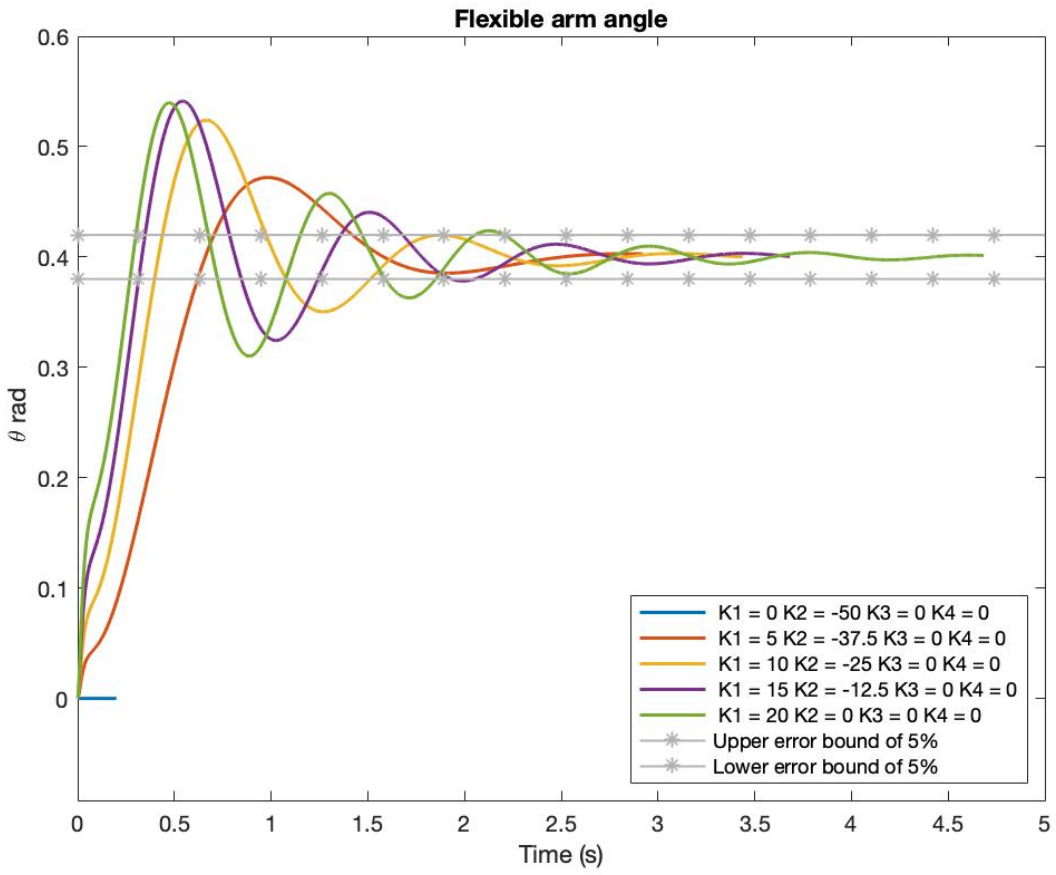

Fig. 16 Theoretical model for the flexible arm hub angle both K1 and K2 are changed.

On the other hand, as the K2 values increased more in the negative direction (i.e. became more negative) the closed loop applied more control on the tip displacement, and hence the deflection decreased significantly relative to when there was no control applied. This trend was clearly observed both on the VI readings and on the theoretical model shown in figure 17. For instance when considering the change for an unchanged K1 value but varying K2 values one can see clearly that the tip deflection decreased as in the case of the two red lines in figures 17 and 15.

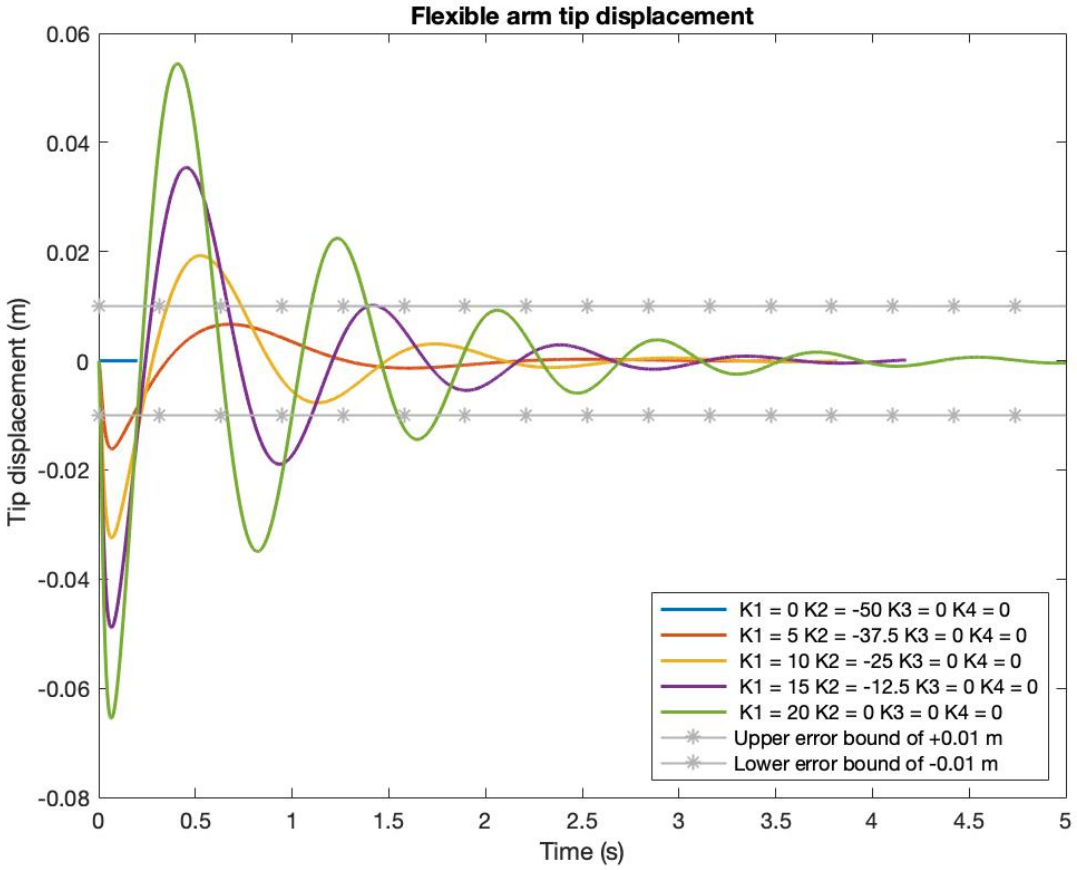

Fig. 17 Theoretical model for the flexible arm tip displacement both K1 and K2 are changed.

When K3, and K4 values are also applied (derivative control gain values), different trends were observed. Firstly, the addition of K3 and K4 caused less overshooting to happen to the hub angle. However, settling time increased. On the other hand, the tip displacement had less overshoot and a lower settling time. Again, this matched the theoretical model as seen in figures 18 and 19.

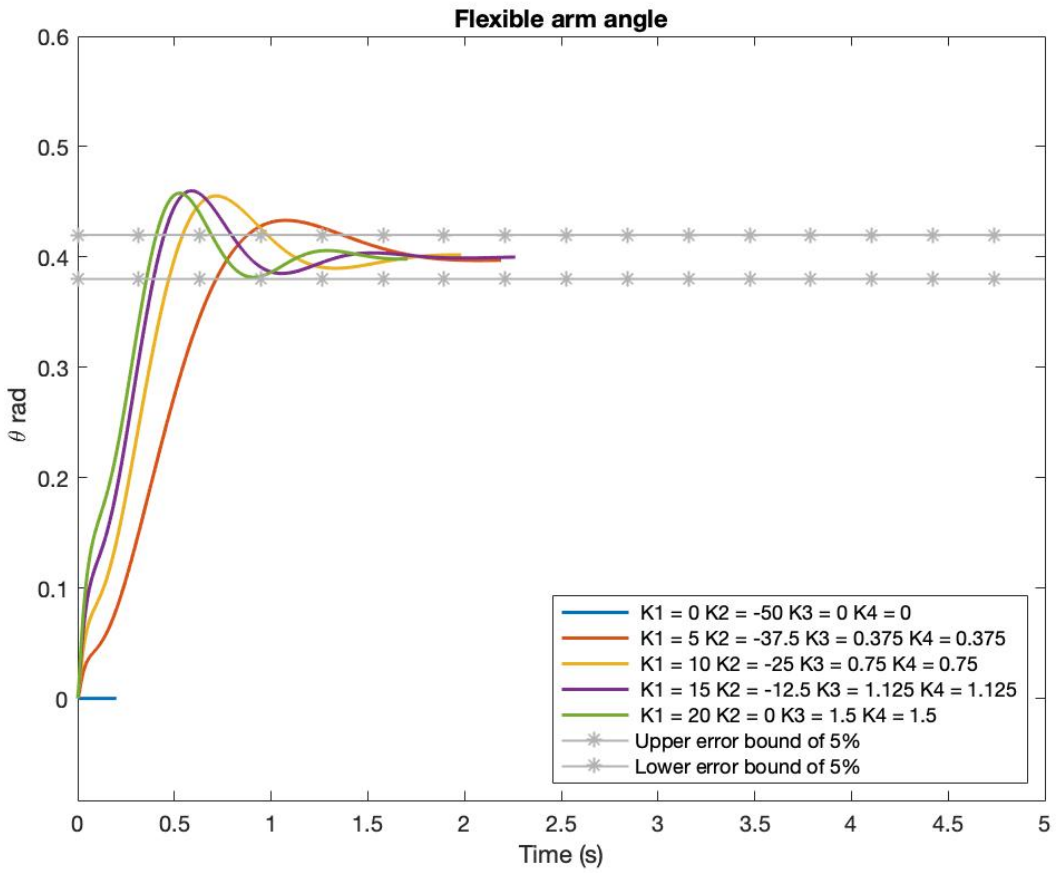

Fig. 18 Theoretical model for the flexible arm hub angle using all four gains (K1,K2,K3,K4).

It is worth noting that all of these tests were done with a step size of 0.4 rad (i.e. +/- 0.2 from the zero point) while keeping tip displacement below 0.01 m.

With that being said, multiple gain values were tested in order to pick gain values such that the overshoot did not exceed 10%, settling time was within 5% of the final value (0.4 rad), and that tip deflection remained lower than 0.01 m. Two sets of gain values were chosen to achieve this goal. The first set consisted of gain values K1 = 6.2070, K2 = -32.7590, K3 = 0.9830, and K4 = 0.1690. The second set use value of K1 = 8.2760, K2 = -6.8970, K3 = 1.2930, K4 = 0.2439. The theoretical simulations and experimental data were plotted on the same plot as shown in figures 20 and 21.

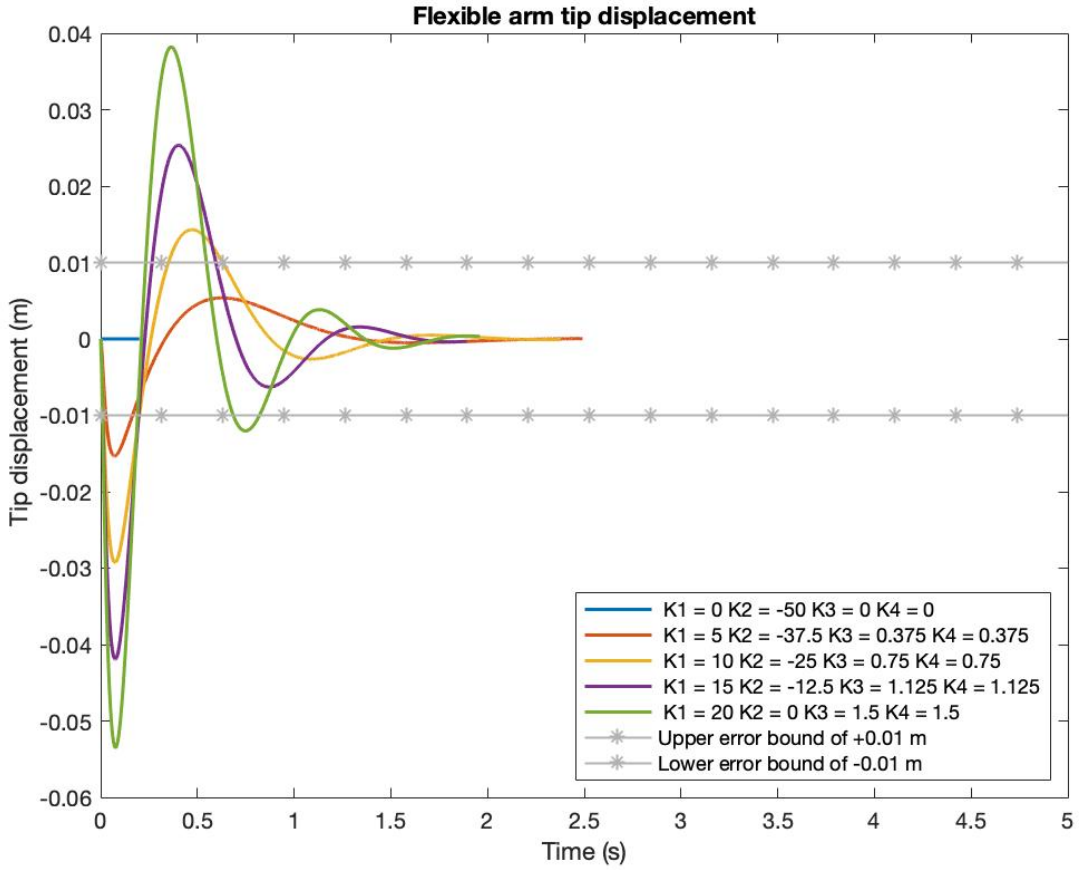

Fig. 19 Theoretical model for the flexible arm tip displacement using all four gains (K1,K2,K3,K4).

When considering figures 20 and 21, it is obvious that the trend and shape of graphs lines up perfectly, which validates the model. When looking at figure 20 that describes the change in the flexible arm hub angle, we notice that the experimental data does not reach the final value desired of 0.4 rad, it rather levels out at a value slightly below at 0.38 rad. This might be due to the natural damping that the system contains that the models do not account for. This natural dampening can be from the friction between the hub gear and the interaction with the flexible arm connections to the hub. As one can also observe on figure 20, the first set of gain values represented by the blue and purple lines for theoretical and experimental results respectively has slightly higher value for the hub angle compared to the second set. In theory, both gain sets should have met the requirement of 10-5 % over shoot allowed. Set 1 where K1 = 6.2070, K2 = -32.7590, K3 = 0.9830, and K4 = 0.1690 had a settling time of 0.76 seconds and an overshoot of 1.7 % from the desired angle. The second set where K1 = 8.2760, K2 = -6.8970, K3 = 1.2930, and K4 = 0.2439 had a settling time of 0.65 seconds, and did not overshoot at all. The experimental results did not overshoot nor did it have a meaningful settling time simply because it did not reach to the desired input value.

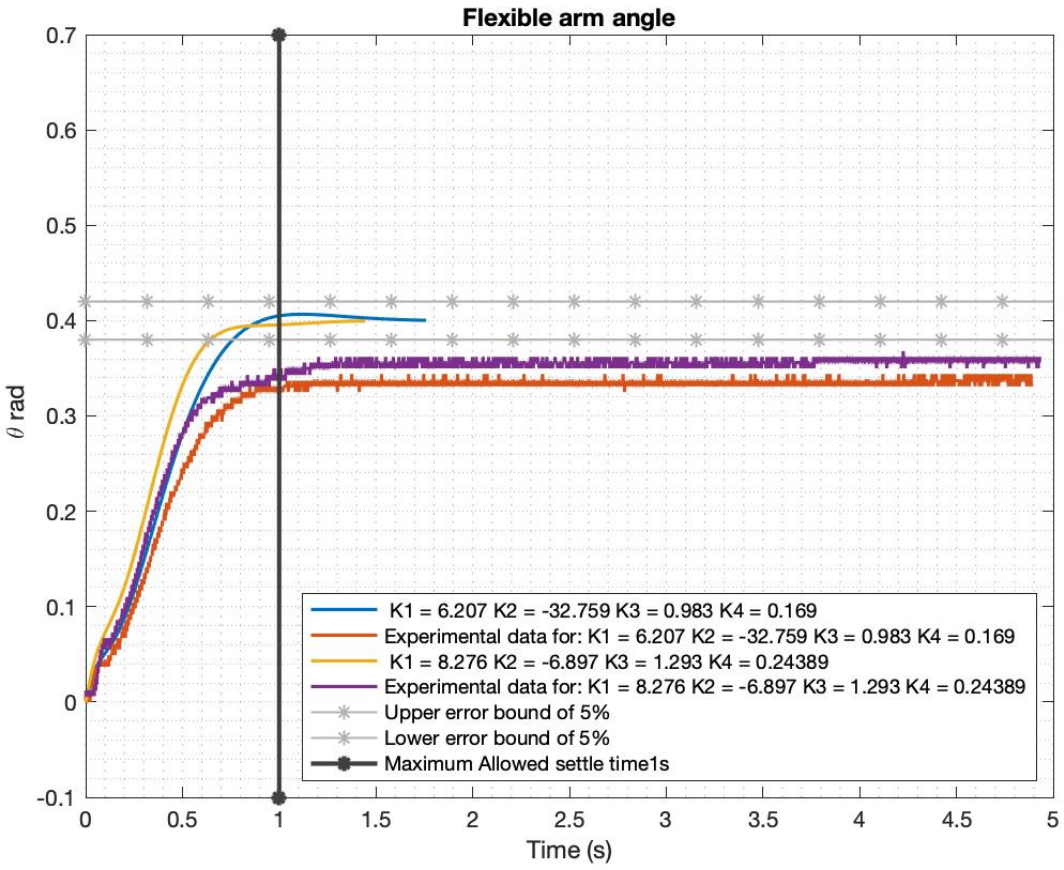

Fig. 20 Theoretical model and experimental data for flexible arm hub angle.

On the other hand, the tip displacement shown in figure 21 had to be contained within 0.01 m. For the first set of K1 = 6.2070, K2 = -32.7590, K3 = 0.9830, and K4 = 0.1690 (same as the ones shown in figure 20), we observe a big overshoot that exceeded the allowable overshoot in tip displacement. As a result, the second set of K1 = 8.2760, K2 = -6.8970, K3 = 1.2930, K4 = 0.2439 were considered, which in return caused the tip deflections not to exceed the allowable limit. In both cases, we observe a perfect match between the theoretical model and the collected data, which increases the confidence in the closed loop system that was developed to model the tip displacement.

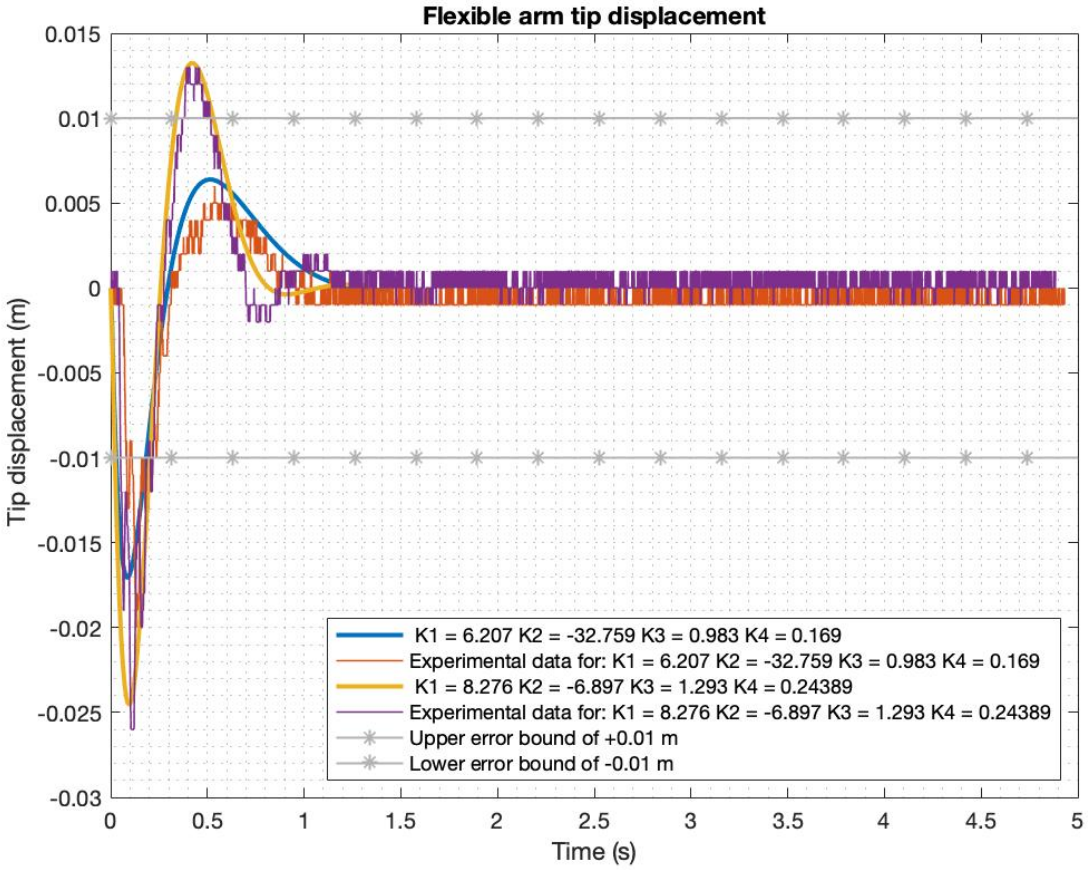

Fig. 21 Theoretical model and experimental data for flexible tip displacement.

One might notice that the tip displacement as the arm starts to move initially does exceed the limit of +/- 0.01 m. This initial dip that exceeds - 0.01 m on figures 15 , 17, 19 and figure 21 can be ignored because it happens instantaneously before the system starts to apply the control mechanism to the tip displacement. Once the tip displacement is being applied, a quick rise in the tip displacement happens represented by the steep upward slope after the dip.

Overall, the models seemed to match the experimental results pretty well, yet the natural dampening that is happening in the system is one of the big sources of error that the model does not account for.

C. Rigid arm Vs. Flexible arm

As can be seen from the figures above, a flexible bar is much more difficult to control compared to a rigid bar. The rigid bar, in general followed the model created relatively accurately and did not have any lingering oscillations that could be seen in the collected data. At the same time the rigid bar did not overshoot nearly as easily as the flexible bar did and was able to reach its 5% settling much more quickly than the flexible bar. It was comparatively difficult to get the flexible bar to reach 5% settling in a timely manner. In general, this means that controlling a flexible structure would be

much more difficult than controlling a rigid one. The flexible structure would be more vulnerable to oscillations/wobble and so would be difficult to precisely control.

Coming up with gain values is a harder process for the flexible bar, because the gain values picked must satisfy more criteria (hub angle and tip displacement) at the same time compared to the rigid bar where you will have to control one output only (the hub angle). Indeed implementing multiple type of inputs help.

One factor that might affected the experimental results for both rigid and flexible arm is the dead band of the motor. The gears inside the motor have their own friction, and to overcome this friction more voltage needs to be applied. As the voltage decreases, the friction forces dominates the motion, and the torque is to small to get the arm moving. This is one of the main reasons why the flexible arm saturates below the desired angle, because as the arm reaches the desired angle, the voltage decreases, and more and more friction occurs. This is also can be looked at in terms of the applied gains. For the flexible arm, the gain values were lower than the rigid arm. As a result, the rigid arm gained much more angular acceleration compared to the flexible arm, and hence overshot more than the flexible arm. In a sense, the higher your gain values are, you are more likely to escape this dead band effect, however you are more likely to overshoot too.

IV. Conclusions and Recommendations

Control and response theory is incredibly versatile and useful in a wide variety of applications, and though the scope presented in this experiment was narrow, that much should be obvious. By only looking at two relatively simple systems such as those presented here, it is clear that understanding how to manipulate control laws and find gains for those functions is fundamentally important to any system that an engineer wishes to manipulate. As with most advanced engineering problems, the development of these control laws is not a simple task. Complicating a system, even with seemingly small changes, can massively increase the number of gains to find, and how to define control laws for that system. As well, solving differential equations of motion is similarly affected by small complications to a system. What is fundamentally important to understand is that control theory allows us to command the behavior of a system by measuring its current state and comparing it to some desired state, then actuating the system so as to move it toward the desired state. The error between the current and desired states of a system can be measured for more than one state, and by using the error in the desired rates of change of the state. Each of these errors will affect the behavior of the controller in some way, and often the most difficult part of controlling a system is finding out how much to actuate it based upon those errors. In this report, we consider various ways of calculating the proper gains for a control law, though they are primarily brute-force or guess-and-check algorithms. In the future, other methods should be explored, at least at a high level.

V. Acknowledgements

We would like to thank Professors Axelrad and Frew for a fun and challenging semester. We'd also like to thank our lab coordinator Bobby for all his efforts in keeping the lab equipment in good repair. Finally, we'd like to thank all of the TA's and LA's for their help, both in class and out.

VI. Appendix

A. Team Member Participation Table

Name	Plan	Model	Experiment	Results	Report	Code
Samuel Felice	1	2	1	1	1	1
Samuel Firth	1	1	1	1	2	1
Abdullah AlAmeri	1	1	1	2	1	2
Nicholas Boender	2	1	2	1	1	1

B. MATLAB

Rigid Body Gain Calculation

```
close all
clear all
K_g = 33.3; %[-]
K_m = 0.0401; %V/(rad/s)
R_m = 19.2; %Ohms
J_hub = 0.0005; %kg*m^2
J_load = 0.0015; %kg*m^2
J = J_hub+J_load; %kg*m^2
K_D = linspace(-1.5,1.5,100);
K_P = linspace(-2,50,100);
for i = 1:100
    for j = 1:100
        omega_n(i,j) = sqrt((K_P(i)*K_g*K_m)/(J*R_m));
        zeta(i,j) = ((K_g^2*K_m^2)+(K_D(j)*K_g*K_m))/(2*sqrt(K_P(i)*K_g*K_m*J*
            R_m));
        M_p(i,j) = exp(pi*((-zeta(i,j)/sqrt(1-zeta(i,j)^2))));
        t_s(i,j) = 3/(zeta(i,j)*omega_n(i,j));
    end
end

```

```
max_voltage = zeros(size(t_s));
for i = 1:numel(K_P)
    for j = 1:numel(K_D)
        max_voltage(i,j) = K_P(i)*(0.7)+K_D(j)*(0.7/t_s(i,j));
    end
end
count = 0;
for i = 1:100
    for j = 1:100
        if zeta(i,j)<1.1 && zeta(i,j)>0.9 && abs(M_p(i,j))<0.05 && abs(t_s(i,j
            ))<0.15 && abs(max_voltage(i,j))<9
            count = count+1;
            nice(:,count) = [K_P(i) K_D(j)];
            zeta_nice(count) = zeta(i,j);
            omega_n_nice(count) = omega_n(i,j);
            max_voltage_nice(count) = max_voltage(i,j);
            t_s_nice(count) = t_s(i,j);
        end
    end
end
```

Rigid Bar Simulation/Model

```

%% Constants

clear
clc
close all;


thetad = input('Wanted Angle(Radians): ');
Kg = 33.3;
```

```
11 Km = 0.0401; %V/(rad/sec)
12 Rm = 19.2; %ohms
13 %Rigid Arm
14 J_hub = 0.0005; %Kgm^2
15 J_load = 0.0015; %Kgm^2
16 J = J_hub + J_load;
17
18 %% Get Matrix of Kp and Kd values
19 Kp = linspace(-2,50,6);
20 Kd = linspace(-1.0,1.5,6);
21
22 deg = zeros(49,644);
23 Time = zeros(49,644);
24 index = 1;
25 for i=1:length(Kd)
26     for j = 1:length(Kp)
27         %% Closed Loop Values
28         n1 = Kp(j)*Kg*Km/(J*Rm);
29         d2 = 1;
30         d1 = (Kg^2*Km^2/(J*Rm) + Kd(i)*Kg*Km/(J*Rm));
31         d0 = Kp(j)*Kg*Km/(J*Rm);
32
33         %% Simulation
34         %%% Closed Loop System
35         num = n1;
36         den = [d2 d1 d0];
37         sysTF = tf(num,den);
38         %%% Step Response
39         [x,t] = step(sysTF);
40         x = (thetad)*x;
41 %          figure(1);clf;
42         Time(index,1:length(t)) = t';
```

```
43          deg(index,1:length(x)) = x';
44 %           plot(t,x);
45 %           xlabel("Time");
46 %           ylabel("Angle \deg ");
47 %           hold on
48
49          % The goal is to have less than 5% overshoot, and reach to 5% of
50          % the steady state value within 0.15 seconds, so the values would
51          % be stored in the following matrix:
52
53          % what this's doing that it will define the threshold of settling
54          % time to be when the value is 5% of the stead state value, now
55          % you'll have to look manually into when that time is
56
57          SystemInfo = stepinfo(sysTF,'SettlingTimeThreshold',0.05);
58
59 % store settling time and maximum over shoot, settling time again is
60 % already defined to be within 0.05 seconds!
61
62 SettleTime(i,j) = SystemInfo.SettlingTime; % each row represent 1 Kd value,
       each column Kp
63 OvershootValue(i,j) = SystemInfo.Overshoot;  % each row represent 1 Kd value,
       each column Kp
64
65
66
67 index = index + 1;
68
69      end
70 end
71
72 %% Plot data
```

```
73
74 %{
75 figure;clf;
76 subplot(1,7,1)
77 hold on;
78 for k = 1:6
79     plot(Time(k,:),deg(k,:));
80 end
81 hold off;
82 title("Kd = -1.5");
83 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
84 xlabel("Time(sec)");
85 ylabel("Position(theta)");
86 ylim([-10,10]);
87 subplot(1,7,2)
88 hold on;
89 for k = 7:12
90     plot(Time(k,:),deg(k,:));
91 end
92 hold off;
93 title("Kd = -1.0");
94 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
95 xlabel("Time(sec)");
96 ylabel("Position(theta)");
97 ylim([-10,10]);
98 subplot(1,7,3)
99 hold on;
100 for k = 13:18
101     plot(Time(k,:),deg(k,:));
102 end
103 hold off;
104 title("Kd = -0.5");
```

```
105 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
106 xlabel("Time(sec)");
107 ylabel("Position(theta)");
108 ylim([-10,10]);
109 subplot(1,7,4)
110 hold on;
111 for k = 19:24
112     plot(Time(k,:),deg(k,:));
113 end
114 hold off;
115 title("Kd = 0");
116 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
117 xlabel("Time(sec)");
118 ylabel("Position(theta)");
119 ylim([-10,10]);
120 subplot(1,7,5)
121 hold on;
122 for k = 25:30
123     plot(Time(k,:),deg(k,:));
124 end
125 hold off;
126 title("Kd = 0.5");
127 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
128 xlabel("Time(sec)");
129 ylabel("Position(theta)");
130 ylim([-10,10]);
131 subplot(1,7,6)
132 hold on;
133 for k = 31:36
134     plot(Time(k,:),deg(k,:));
135 end
136 hold off;
```

```
137 title("Kd = 1.0");
138 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
139 xlabel("Time(sec)");
140 ylabel("Position(theta)");
141 ylim([-10,10]);
142 subplot(1,7,7)
143 hold on;
144 for k = 37:42
145     plot(Time(k,:),deg(k,:));
146 end
147 hold off;
148 title("Kd = 1.5");
149 legend("Kp=-2","6.67","15.33","24","32.67","41.33","50");
150 xlabel("Time(sec)");
151 ylabel("Position(theta)");
152 ylim([-10,10]);
153 %}
154
155
156 %% plot 2:
157
158 %{
159 figure;
160 for i = 1:length(Kd)
161
162     for j = 1:length(Kp)
163         subplot(1,length(Kd),i);
164         Time_plot = Time(((i-1)*length(Kp))+j,:);
165         deg_plot = deg(((i-1)*length(Kp))+j,:);
166
167         zero_index = find(Time_plot==0);
168         Time_plot(zero_index(2:end)) = [];
```

```
169
170             deg_plot(zero_index(2:end)) = [];
171
172             plot(Time_plot,deg_plot) ;
173             hold on;
174             ylim([-1,1]);
175             xlim([0,1]);
176
177
178
179         end
180
181     title(['Kd =' num2str(Kd(i))])
182     legend([ num2str(Kp(1)) ],[ num2str(Kp(2)) ],[ num2str(Kp(3)) ],[ num2str(Kp
            (4)) ],[ num2str(Kp(5)) ],[ num2str(Kp(6)) ] );
183
184 end
185 %}
186 %% plot 3:
187
188 figure;
189 for i = 1:length(Kd)
190
191     for j = 1:length(Kp)
192
193         figure(i)
194
195         Time_plot = Time(((i-1)*length(Kp))+j,:);
196         deg_plot = deg(((i-1)*length(Kp))+j,:);
197
198         % clean the zeros:
199         % clean everything but first index
```

```
200         zero_index = find(Time_plot==0);
201         Time_plot(zero_index(2:end)) = [];
202
203         deg_plot(zero_index(2:end)) = [];
204
205         plot(Time_plot,deg_plot,'LineWidth',1.5) ;
206         hold on;
207         ylim([-thetad-thetad*0.5 ,thetad+thetad*0.9]);
208         xlim([0,1]);
209
210
211
212     end
213
214 % plot overshoot bounds
215
216 Errorbound = 0.05*thetad*(ones(1,10));
217 ErrorTime = linspace(0,1,10);
218
219 plot(ErrorTime,Errorbound+thetad,'*-','Color',[0.7 0.7 0.7],'LineWidth',1)
220 plot(ErrorTime,thetad-Errorbound,'*-','Color',[0.7 0.7 0.7],'LineWidth',1)
221
222 title(['Kd =' num2str(Kd(i))],'fontsize',24)
223 grid minor
224 xlabel('Time (s)','fontsize',22)
225 ylabel('\theta rad','fontsize',22)
226 legend([ 'Kp=' num2str(Kp(1)) ],[ 'Kp=' num2str(Kp(2)) ],[ 'Kp=' num2str(Kp
        (3)) ],[ 'Kp=' num2str(Kp(4)) ],[ 'Kp=' num2str(Kp(5)) ],[ 'Kp=' num2str
        (Kp(6)) ] ,'5% Error bounds','fontsize',14);
227 set(gca,'FontSize',18)
228 hold on
229
```

```
230 end
231
232
233 %% printout tbale with results
234
235 % prepeare inputs to table
236
237 Kd_table = [ ones(1,length(Kp))*Kd(1)...
238     ones(1,length(Kp))*Kd(2)...
239     ones(1,length(Kp))*Kd(3)...
240     ones(1,length(Kp))*Kd(4)...
241     ones(1,length(Kp))*Kd(5)...
242     ones(1,length(Kp))*Kd(6)...
243     ];
244
245 Kp_table = [ Kp...
246     Kp...
247     Kp...
248     Kp...
249     Kp...
250     Kp...
251     ];
252
253 Settle_table = [ SettleTime(:,1)'...
254     SettleTime(:,2)'...
255     SettleTime(:,3)'...
256     SettleTime(:,4)'...
257     SettleTime(:,5)'...
258     SettleTime(:,6)'...
259     ];
260
261 Overshoot_table = [ OvershootValue(:,1)'...
```

```
262     OvershootValue(:,2) '...
263     OvershootValue(:,3) '...
264     OvershootValue(:,4) '...
265     OvershootValue(:,5) '...
266     OvershootValue(:,6) '...
267     ];
268
269
270 Table_results = table(Kd_table',Kp_table',Settle_table',Overshoot_table',...
271     'VariableNames',{'Kd','Kp','Settle_time','Overshoot'})
272
273
274
275 fprintf('NOTE :')
276
277 fprintf( '\n' )
278 fprintf( '\n' )
279
280
281 fprintf('with 5%% overshoot allowed you are limited to overshoot values of: \n
    ')
282
283 fprintf( [num2str(thetad) char(177) num2str(thetad.*0.05) ] )
284
285 fprintf( '\n' )
286
287
288 fprintf('Error bound on settling time is 5%% ')
289
290 fprintf( '\n' )
```

Rigid Bar Model V. Experiment

```
%% Housekeeping
clear;
clc;
close all;

%% Constants
Kg = 33.3;
Km = 0.0401; %V/(rad/sec)
Rm = 19.2; %ohms
%Rigid Arm
J_hub = 0.0005; %Kgm^2
J_load = 0.0015; %Kgm^2
J = J_hub + J_load;

%% Read in data
data = load("kp12.7kdneg0.05.txt");

Time = data(:,1);
Time = Time/1000;
hubangle = data(:,2);
Kp = data(1,8);
Kd = data(1,10);
angle_correction = abs(data(1,6));
thetad = 2*angle_correction;

%% Correct/Manipulate Data
% ensure plot starts at bottom
if hubangle(1) > 0
    topcut = find(hubangle < -angle_correction);
    Time = Time(topcut(10):end);
    hubangle = hubangle(topcut(10):end);
end
```

```
33
34 % cut beginning off
35 i = 1;
36 while abs(hubangle(i) - hubangle(1)) < 0.2*angle_correction
37     i = i + 1;
38 end
39 Time = Time(i-30:end);
40 hubangle = hubangle(i-30:end);
41
42 % manipulate data
43 Time = Time - Time(1);
44 hubangle = hubangle + angle_correction;
45 % cut end off
46 k = 1;
47 while abs(hubangle(k) - hubangle(1)) < (2*0.92)*angle_correction
48     k = k + 1;
49 end
50 Time = Time(1:k+50);
51 hubangle = hubangle(1:k+50);
52
53 %% Model
54 % Closed Loop Values
55 n1 = Kp*Kg*Km/(J*Rm);
56 d2 = 1;
57 d1 = (Kg^2*Km^2/(J*Rm) + Kd*Kg*Km/(J*Rm));
58 d0 = Kp*Kg*Km/(J*Rm);
59
60 % Simulation
61 %%% Closed Loop System
62 num = n1;
63 den = [d2 d1 d0];
64 sysTF = tf(num,den);
```

```
%%% Step Response
[x,t] = step(sysTF);
x = (thetad)*x;

Time2 = t;
deg = x;

%% Error
Errorbound = 0.05*thetad*(ones(1,10));
ErrorTime = linspace(0,0.3,10);

%% Get 5% settling time for data
S95_D = find(hubangle > (thetad-Errorbound));

%% Get 5% settling time for model
S95_M = find(deg > (thetad-Errorbound));

%% Plot
figure;
hold on;
plot(Time,hubangle);
plot(Time2,deg);

plot(ErrorTime,Errorbound+thetad,'*-','Color',[0.7 0.7 0.7],'LineWidth',1)
plot(ErrorTime,thetad-Errorbound,'*-','Color',[0.7 0.7 0.7],'LineWidth',1)

line([Time(S95_D(1)) Time(S95_D(1))], ylim);
line([Time2(S95_M(1)) Time2(S95_M(1))], ylim,'Color','red');

xlabel('Time (s)','fontsize',22);
ylabel(' \theta rad','fontsize',22);
set(gca,'FontSize',18);
```

```
title([num2str(thetad) ' step size, Kp=' num2str(Kp) ', Kd=' num2str(Kd)]);
legend("Experiment","Simulation");


disp('Experimental Settling Time(sec) = ');
disp(num2str(Time(S95_D(1))));
disp('Theoretical Settling Time(sec) = ');
disp(num2str(Time2(S95_M(1))));
```

Flexible arm random gains calculations

```
%% info:

        %  flexible arm dynamics can be modeled as the rigid arm
        %  (a spring-mass-damper system) plus an additional spring-mass-
            damper



        %% housekeepign:

         clear
         clc
         close all


        %% ask for input

        % desired vlues
         thetad = input('Wanted Angle(Radians): ');
         dispd = input('Wanted displacement(meter): ');

        %% closed loop transfer constants:
```

```
22
23          Kg = 33.3 ; % no units
24          Km = 0.0401 ; %V/(rad/sec);
25          Rm = 19.2 ; % Ohms
26
27          Jhub = 0.0005;
28          Jload = 0.0015;
29          J = Jhub+Jload ;
30          LinkLength = 0.45;
31          Marm = 0.06 ; %kg
32          Jarm = 0.004 ; %kgm^2
33          Mtip = 0.05 ; %kg
34          JM = 0.01;
35          fc = 1.8 ; %HZ
36
37          JL = Jarm + JM;
38
39          Karm = (2*pi*fc)^2 *(JL) ;
40
41          % note:
42          % K1 = Kp?
43          % K2 = Kpd
44          % K3 = KD?
45          % K4 = KDd
46
47          K1 = linspace(0,20,5);
48          K2 = linspace(-50,0,5);
49          K3 = linspace(0,1.5,5);
50          K4 = linspace(0,1.5,5);
51
52      % definitions in lab docuement:
53
```

```
54
55          p1 = - (Kg^2 * Km^2) / (Jhub*Rm) ;
56          p2 = (Kg^2 * Km^2 * LinkLength ) / (Jhub*Rm) ;
57
58          q1 = Karm / ( LinkLength*Jhub) ;
59          q2 = - ( (Karm*( Jhub + JL )) / (JL*Jhub)) ;
60
61          r1 = (Kg*Km)/(Jhub*Rm) ;
62          r2 = - (Kg*Km*LinkLength)/(Jhub*Rm);
63
64          % l = lambda;
65
66
67          % The angle of the arm (or output shaft) is denoted by, theta_L
68
69
70
71          % Closed Loop System
72
73          index = 1;
74
75          for j=1:length(K1);
76
77                  %
78
79          l3 = -p1 + K3(j).*r1 + K4(j).*r2 ;
80          l2 = -q2 + K1(j).*r1 + K2(j).*r2 + K4(j).*(p2*r1 - r2*p1) ;
81          l1 = p1*q2 - q1*p2 + K3(j).*(q1*r2 - r1*q2) + K2(j).*(p2*r1 - r2*p1) ;
82          l0 = K1(j).*(q1*r2 - r1*q2);
83
84          num_angle = K1(j) .* [ r1 0 (q1*r2 - r1*q2) ] ;
85          den_angle = [1 l3 l2 l1 l0];
```

```
86
87          % disp = displacement of the tip.
88
89          num_disp = K1(j) .* [ r2 (r1*p2 - p1*r2)  0];
90          den_disp = [1 13 12 11 10];
91
92          System_angle = tf(num_angle,den_angle);
93
94          System_disp = tf(num_disp,den_disp);
95
96          [ TipeAngle t_angle ] = step(System_angle); % apply unit step, the
                response is the same for magnitude
97          % that's bigger than 1 so you just scale it.
98
99          [ TipDisplacement t_disp ] = step(System_disp);
100
101
102         % repeat the unit input
103         TipeAngle = (thetad)*(TipeAngle);
104         TipDisplacement = thetad*TipDisplacement;
105
106
107         Time_angle(index,1:length(t_angle)) = t_angle';
108         deg(index,1:length(TipeAngle)) = TipeAngle';
109
110         Time_disp(index,1:length(t_disp)) = t_disp';
111         disp(index,1:length(TipDisplacement)) = TipDisplacement';
112
113
114         % The goal is to have less than 5% overshoot, and reach to 5% of
115         % the steady state value within 0.15 seconds, so the values would
116         % be stored in the following matrix:
```

```
117
118         % what this's doing that it will define the threshold of settling
119         % time to be when the value is 5% of the stead state value, now
120         % you'll have to look manually into when that time is
121
122
123         % store settling time and maximum over shoot, settling time again is
124         % already defined to be within 0.05 seconds!
125
126         %SettleTime(i,j) = SystemInfo.SettlingTime; % each row represent 1 Kd
                value, each column Kp
127         %OvershootValue(i,j) = SystemInfo.Overshoot;  % each row represent 1
                Kd value, each column Kp
128
129         SystemInfo_angle = stepinfo(System_angle,'SettlingTimeThreshold',0.05)
                ;
130         SettleTime_angle(index,1) = SystemInfo_angle.SettlingTime; % each row
                represent 1 Kd value, each column Kp
131         OvershootValue_angle(index,1) = SystemInfo_angle.Overshoot;  % each
                row represent 1 Kd value, each column Kp
132
133         SystemInfo_disp = stepinfo(System_disp,'SettlingTimeThreshold',0.05);
134         SettleTime_disp(index,1) = SystemInfo_disp.SettlingTime; % each row
                represent 1 Kd value, each column Kp
135         OvershootValue_disp(index,1) = SystemInfo_disp.Overshoot;  % each row
                represent 1 Kd value, each column Kp
136
137         index = index + 1;
138
139         end
140
141         %% plot
```

```
142
143         figure;
144
145         for j = 1
146
147         for i = 1:length(K1)
148
149                 figure(j)
150
151                 Time_plot = Time_angle(i,:);
152                 deg_plot = deg(i,:);
153
154                 % clean the zeros:
155                 % clean everything but first index
156                 zero_index = find(Time_plot==0);
157                 Time_plot(zero_index(2:end)) = [];
158
159                 deg_plot(zero_index(2:end)) = [];
160
161                 plot(Time_plot,deg_plot,'LineWidth',1.5,...
162             'DisplayName',[' K1 = ' num2str(K1(i)) ' K2 = ' num2str(K2(i))...
163             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]);
164                 hold on;
165
166
167
168
169         end
170
171                 ylim([-thetad-thetad*0.5 ,thetad+thetad*0.9]);
172                 xlim([0,5]);
173                 title('Flexible arm angle')
```

```
174                 Errorbound = 0.05*thetad*(ones(1,20));
175                 ErrorTime = linspace(0,6,20);
176
177             plot(ErrorTime,Errorbound+thetad,'*-','Color',[0.7 0.7 0.7],'
                    LineWidth',1,'DisplayName','Upper error bound of 5%')
178             plot(ErrorTime,thetad-Errorbound,'*-','Color',[0.7 0.7 0.7],'
                    LineWidth',1,'DisplayName','Lower error bound of 5%')
179
180             xlabel('Time (s)')
181             ylabel('\theta rad')
182             %legend(['K1=' num2str(K1(1))],['K1=' num2str(K1(2))])%,['K1='
                    num2str(K1(3))],['K1=' num2str(K1(4))],['K1=' num2str(K1(5))
                    ],'5% Error bounds');
183         legend('show','Location','SouthEast');
184     grid minor
185     ylim([-thetad+(thetad/1.3) thetad+(thetad/2)])
186
187         end
188
189         % do the plot for tip displacement
190
191         for j = 1
192
193         for i = 1:length(K1)
194
195                 figure(j+1)
196
197                 Time_plot = Time_disp(i,:);
198                 disp_plot = disp(i,:);
199
200                 % clean the zeros:
201                 % clean everything but first index
```

```
202                 zero_index = find(Time_plot==0);
203                 Time_plot(zero_index(2:end)) = [];
204
205                 disp_plot(zero_index(2:end)) = [];
206
207                 plot(Time_plot,disp_plot,'LineWidth',1.5...
208             ,'DisplayName',[' K1 = ' num2str(K1(i)) ' K2 = ' num2str(K2(i))...
209             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]) ;
210                 hold on;
211
212
213
214
215         end
216
217                 %ylim([-dispd-dispd*0.5 ,dispd+dispd*0.9]);
218                 xlim([0,5]);
219                 title('Flexible arm tip displacement')
220                 Errorbound = 0.1*dispd*(ones(1,20));
221                 ErrorTime = linspace(0,6,20);
222
223             plot(ErrorTime,Errorbound+dispd,'*-','Color',[0.7 0.7 0.7],'
                    LineWidth',1,'DisplayName','Upper error bound of 10%')
224             plot(ErrorTime,dispd-Errorbound,'*-','Color',[0.7 0.7 0.7],'
                    LineWidth',1,'DisplayName','Lower error bound of 10%')
225
226             xlabel('Time (s)')
227             ylabel(' Tip displacement (m)')
228            %legend([ 'K1=' num2str(K1(1)) ],[ 'K1=' num2str(K1(2)) ])%,[ 'K1='
                    num2str(K1(3)) ],[ 'K1=' num2str(K1(4)) ],[ 'K1=' num2str(K1(5))
                     ],'5% Error bounds');
229         legend('show','Location','SouthEast')
```

```
230
231         end
232
233
234
235         %% printout tbale with results
236
237         % prepeare inputs to table
238
239         K1_table = [ K1...
240             %ones(1,length(K1))*K1(2)...
241             %ones(1,length(K1))*K1(3)...
242             %ones(1,length(K1))*K1(4)...
243             %ones(1,length(K1))*K1(5)...
244              ];
245
246         K2_table = [ K2...
247             %ones(1,length(K2))*K2(2)...
248             %ones(1,length(K2))*K2(3)...
249             %ones(1,length(K2))*K2(4)...
250             %ones(1,length(K2))*K2(5)...
251              ];
252
253         K3_table = [ K3...
254             %ones(1,length(K3))*K3(2)...
255             %ones(1,length(K3))*K3(3)...
256             %ones(1,length(K3))*K3(4)...
257             %ones(1,length(K3))*K3(5)...
258              ];
259
260         K4_table = [ K4...
261             %ones(1,length(K4))*K4(2)...
```

```
262             %ones(1,length(K4))*K4(3)...
263             %ones(1,length(K4))*K4(4)...
264             %ones(1,length(K4))*K4(5)...
265              ];
266
267
268         Settle_table_angle =  [ SettleTime_angle' ];
269
270         Overshoot_table_angle =  [ OvershootValue_angle' ];
271
272         Settle_table_disp =  [ SettleTime_disp' ];
273
274         Overshoot_table_disp =  [ OvershootValue_disp' ];
275
276
277
278
279         Table_results = table(K1_table',K2_table',K3_table',K4_table',
                Settle_table_angle',Overshoot_table_angle',...
280              Settle_table_disp',Overshoot_table_disp',...
281              'VariableNames',{'K1','K2','K3','K4','Settle_time_angle','
                    Overshoot_angle','Settle_time_disp','Overshoot_disp'})
282
283
284
285 %       fprintf('NOTE :')
286 %
287 %       fprintf( '\n' )
288 %       fprintf( '\n' )
289 %
290 %
291 %       fprintf('with 10%% overshoot allowed you are limited to overshoot
```

```
    values of: \n')
%
%       fprintf( [num2str(thetad) char(177) num2str(thetad.*0.1) ] )
%
%       fprintf( '\n' )
%
%
%       fprintf('Error bound on settling time is 5%% ')
%
%       fprintf( '\n' )
%
```

Flexible arm tests gain, model, and expermintal data

```
%% info:

%  flexible arm dynamics can be modeled as the rigid arm
%  (a spring-mass-damper system) plus an additional spring-mass- damper



%% housekeepign:

clear
clc
close all


%% ask for input

% desired vlues
thetad = input('Wanted Angle(Radians): ');
dispd = input('Wanted displacement(meter): ');
```

```
20
21 SetTime = 1;
22 %% closed loop transfer constants:
23
24 Kg = 33.3 ; % no units
25 Km = 0.0401 ; %V/(rad/sec);
26 Rm = 19.2 ; % Ohms
27
28 Jhub = 0.0005;
29 Jload = 0.0015;
30 J = Jhub+Jload ;
31 LinkLength = 0.45;
32 Marm = 0.06 ; %kg
33 Jarm = 0.004 ; %kgm^2
34 Mtip = 0.05 ; %kg
35 JM = 0.01;
36 fc = 1.8 ; %dHZ
37
38 JL = Jarm + JM;
39
40 Karm = (2*pi*fc)^2 *(JL) ;
41
42 % note:
43 % K1 = Kp?
44 % K2 = Kpd
45 % K3 = KD?
46 % K4 = KDd
47
48 % use actual gains applied using data collected for error analysis
49
50 trial1 = load('Data/Group18_trail1_flex_TG') ;
51 trial2 = load('Data/Group18_trail2_flex_TG') ;
```

```
52
53 K1 = [ mean(trial1(:,8)) mean(trial2(:,8)) ];
54 K2 = [ mean(trial1(:,9)) mean(trial2(:,9)) ];
55 K3 = [ mean(trial1(:,10)) mean(trial2(:,10)) ];
56 K4 = [ mean(trial1(:,11)) mean(trial2(:,11)) ];
57
58 % definitions in lab docuement:
59
60
61 p1 = -(Kg^2 * Km^2) / (Jhub*Rm) ;
62 p2 = (Kg^2 * Km^2 * LinkLength ) / (Jhub*Rm) ;
63
64 q1 = Karm / ( LinkLength*Jhub) ;
65 q2 = -( (Karm*( Jhub + JL )) / (JL*Jhub)) ;
66
67 r1 = (Kg*Km)/(Jhub*Rm) ;
68 r2 = -(Kg*Km*LinkLength)/(Jhub*Rm);
69
70 % l = lambda;
71
72
73 % The angle of the arm (or output shaft) is denoted by, theta_L
74
75
76
77 % Closed Loop System
78
79 index = 1;
80
81 for j=1:length(K1);
82
83         %
```

```
84
85  l3 = -p1 + K3(j).*r1 + K4(j).*r2 ;
86  l2 = -q2 + K1(j).*r1 + K2(j).*r2 + K4(j).*(p2*r1 - r2*p1) ;
87  l1 = p1*q2 - q1*p2 + K3(j).*(q1*r2 - r1*q2) + K2(j).*(p2*r1 - r2*p1) ;
88  l0 = K1(j).*(q1*r2 - r1*q2);
89
90  num_angle = K1(j) .* [ r1 0 (q1*r2 - r1*q2) ] ;
91  den_angle = [1 l3 l2 l1 l0];
92
93 % disp = displacement of the tip.
94
95  num_disp = K1(j) .* [ r2 (r1*p2 - p1*r2)  0];
96  den_disp = [1 l3 l2 l1 l0];
97
98  System_angle = tf(num_angle,den_angle);
99
100 System_disp = tf(num_disp,den_disp);
101
102 [ TipeAngle t_angle ] = step(System_angle); % apply unit step, the response is
        the same for magnitude
103 % that's bigger than 1 so you just scale it.
104
105 [ TipDisplacement t_disp ] = step(System_disp);
106
107
108 % repeat the unit input
109 TipeAngle = (thetad)*(TipeAngle);
110 TipDisplacement = thetad*TipDisplacement;
111
112
113 Time_angle(index,1:length(t_angle)) = t_angle';
114 deg(index,1:length(TipeAngle)) = TipeAngle';
```

```
115
116 Time_disp(index,1:length(t_disp)) = t_disp';
117 disp(index,1:length(TipDisplacement)) = TipDisplacement';
118
119
120 % The goal is to have less than 5% overshoot, and reach to 5% of
121 % the steady state value within 0.15 seconds, so the values would
122 % be stored in the following matrix:
123
124 % what this's doing that it will define the threshold of settling
125 % time to be when the value is 5% of the stead state value, now
126 % you'll have to look manually into when that time is
127
128
129 % store settling time and maximum over shoot, settling time again is
130 % already defined to be within 0.05 seconds!
131
132 %SettleTime(i,j) = SystemInfo.SettlingTime; % each row represent 1 Kd value,
      each column Kp
133 %OvershootValue(i,j) = SystemInfo.Overshoot;  % each row represent 1 Kd value,
       each column Kp
134
135 SystemInfo_angle = stepinfo(System_angle,'SettlingTimeThreshold',0.05);
136 SettleTime_angle(index,1) = SystemInfo_angle.SettlingTime; % each row
      represent 1 Kd value, each column Kp
137 OvershootValue_angle(index,1) = SystemInfo_angle.Overshoot;  % each row
      represent 1 Kd value, each column Kp
138
139 SystemInfo_disp = stepinfo(System_disp,'SettlingTimeThreshold',0.05);
140 SettleTime_disp(index,1) = SystemInfo_disp.SettlingTime; % each row represent
      1 Kd value, each column Kp
141 OvershootValue_disp(index,1) = SystemInfo_disp.Overshoot;  % each row
```

```
    represent 1 Kd value, each column Kp
142
143 index = index + 1;
144
145 end
146
147
148 %% Experimental data:
149
150
151 % adjust the data
152 %t1 = trial 1
153
154 % get time
155 trial1_time = trial1(:,1)*10^(-3);
156 trial2_time = trial2(:,1)*10^(-3);
157
158 % zero time based on the first index
159
160 trial1_time = trial1_time(:) - trial1_time(1);
161 trial2_time = trial2_time(:) - trial2_time(1);
162
163
164 % shift all the measured tip angle from -input/2 to +input/2
165 % so it goes from 0 to input/2;
166
167 % the + thetad/2 is to shift the data up by the input.
168 % the hub goes from negative input/2 to +input/2
169
170 trial1_angle = trial1(:,2) + abs(min(trial1(:,2)));
171 trial2_angle = trial2(:,2)+ abs(min(trial2(:,2)));
172
```

```
173 % git tip displacement
174 trial1_tip = trial1(:,3);
175 trial2_tip = trial2(:,3);
176
177
178 %% dissect data
179
180
181
182 % - - - - - - - - - ( Trial 1 ) - - - - - - - - - - - - - - - -
183
184 figure(3)
185
186 plot(trial1_time,trial1_angle);
187 pts = ginput(2);
188 %info = getrect(figure(3));
189
190 % find closest time so you can cut from there
191
192 idx = knnsearch([trial1_time trial1_angle],pts);
193
194 trial1_time = trial1_time(idx(1):idx(2));
195 trial1_angle = trial1_angle(idx(1):idx(2)); % measured change in angle
196 measureDisp_t1 = trial1_tip(idx(1):idx(2)); % measured change in tip
      displacement
197 % zero time
198 trial1_time = trial1_time - trial1_time(1);
199
200 % get releated tip angle
201 trial1_tip = trial1_tip(idx(1):idx(2));
202
203
```

```
204 % - - - - - - - - - ( end of tiral 1 ) - - - - - - - - - - - - -
205
206
207 % - - - - - - - - - ( Trial 2 ) - - - - - - - - - - - - - - - -
208
209 figure(3)
210
211 plot(trial2_time ,trial2_angle);
212 pts = ginput(2);
213 %info = getrect(figure(3));
214 % find closest time so you can cut from there
215
216 idx = knnsearch([trial2_time trial2_angle],pts);
217
218 trial2_time = trial2_time(idx(1):idx(2));
219 trial2_angle = trial2_angle(idx(1):idx(2));
220
221 % zero time
222 trial2_time = trial2_time - trial2_time(1);
223
224 % get releated tip angle
225 trial2_tip = trial2_tip(idx(1):idx(2));
226
227
228 % - - - - - - - - - ( end of tiral 2 ) - - - - - - - - - - - - -
229
230
231 % store expermintal data again to be plotted:
232
233 time_exp = { trial1_time trial2_time };
234 angle_exp = { trial1_angle trial2_angle };
235 tip_exp = { trial1_tip trial2_tip };
```

```
236
237
238 %% plot
239
240 figure;
241
242 for j = 1
243
244 for i = 1:length(K1)
245
246         figure(j)
247
248         Time_plot_angle = Time_angle(i,:);
249         angle_plot = deg(i,:);
250
251         % clean the zeros:
252         % clean everything but first index
253         zero_index = find(Time_plot_angle==0);
254         Time_plot_angle(zero_index(2:end)) = [];
255
256         angle_plot(zero_index(2:end)) = [];
257
258                 plot(Time_plot_angle,angle_plot,'LineWidth',1.5,...
259             'DisplayName',[' K1 = ' num2str(K1(i)) ' K2 = ' num2str(K2(i))...
260             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]);
261         hold on
262                 plot(time_exp{:,i},angle_exp{:,i},'LineWidth',1.5,...
263             'DisplayName',['Experimental data for:' ' K1 = ' num2str(K1(i)) '
                K2 = ' num2str(K2(i))...
264             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]);
265
266         hold on;
```

```
267
268
269
270
271
272 end
273
274         ylim([-thetad-thetad*0.5  ,thetad+thetad*0.9]);
275         ylim([-0.1  ,0.7]);
276
277         xlim([0,5]);
278         title('Flexible arm angle')
279         Errorbound = 0.05*thetad*(ones(1,20));
280         ErrorTime = linspace(0,6,20);
281
282           plot(ErrorTime,Errorbound+thetad,'*-','Color',[0.7 0.7 0.7],'
                LineWidth',1,'DisplayName','Upper error bound of 5%')
283           plot(ErrorTime,thetad-Errorbound,'*-','Color',[0.7 0.7 0.7],'
                LineWidth',1,'DisplayName','Lower error bound of 5%')
284
285 plot([SetTime , SetTime],[-0.1 0.7],'*-','Color',[0.25, 0.25, 0.25],'
    DisplayName',['Maximum Allowed settle time' num2str(SetTime) 's' ],'
    LineWidth',2)
286
287
288   grid minor
289
290   xlabel('Time (s)')
291   ylabel(' \theta rad')
292   %legend([ 'K1=' num2str(K1(1)) ],[ 'K1=' num2str(K1(2)) ])%,[ 'K1=' num2str(
        K1(3)) ],[ 'K1=' num2str(K1(4)) ],[ 'K1=' num2str(K1(5)) ],'5% Error
        bounds');
```

```
293
294
295 legend('show','Location','SouthEast')
296
297
298 end
299
300 saveas(figure(1),'FlexableAngle.jpg')
301
302 % do the plot for tip displacement
303
304 for j = 1
305
306 for i = 1:length(K1)
307
308         figure(j+1)
309
310         Time_plot_disp = Time_disp(i,:);
311         disp_plot = disp(i,:);
312
313         % clean the zeros:
314         % clean everything but first index
315         zero_index = find(Time_plot_disp==0);
316         Time_plot_disp(zero_index(2:end)) = [];
317
318         disp_plot(zero_index(2:end)) = [];
319
320                 plot(Time_plot_disp,disp_plot,'LineWidth',2,...
321             'DisplayName',[' K1 = ' num2str(K1(i)) ' K2 = ' num2str(K2(i))...
322             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]);
323         hold on
324                 plot(time_exp{:,i},tip_exp{:,i},'LineWidth',0.8,...
```

```
325             'DisplayName',['Experimental data for:' ' K1 = ' num2str(K1(i)) '
                    K2 = ' num2str(K2(i))...
326             ' K3 = ' num2str(K3(i)) ' K4 = ' num2str(K4(i))]);
327
328         hold on;
329
330
331
332
333 end
334
335         %ylim([-dispd-dispd*0.5 ,dispd+dispd*0.9]);
336         xlim([0,5]);
337         title('Flexible arm tip displacement')
338         Errorbound = 0.1*dispd*(ones(1,20));
339         ErrorTime = linspace(0,6,20);
340
341   plot(ErrorTime,Errorbound+dispd,'*-','Color',[0.7 0.7 0.7],'LineWidth',1,'
        DisplayName','Upper error bound of 10%')
342   plot(ErrorTime,dispd-Errorbound,'*-','Color',[0.7 0.7 0.7],'LineWidth',1,'
        DisplayName','Upper error bound of 10%')
343 grid minor
344   xlabel('Time (s)')
345   ylabel(' Tip displacement (m)')
346   %legend([ 'K1=' num2str(K1(1)) ],[ 'K1=' num2str(K1(2)) ])%,[ 'K1=' num2str(
        K1(3)) ],[ 'K1=' num2str(K1(4)) ],[ 'K1=' num2str(K1(5)) ],'5% Error
        bounds');
347 legend('show','Location','SouthEast')
348
349 end
350
351
```

```
352 saveas(figure(2),'FlexableTip.jpg')
353
354 %% printout tbale with results
355
356 % prepeare inputs to table
357
358 K1_table = [ K1...
359     %ones(1,length(K1))*K1(2)...
360     %ones(1,length(K1))*K1(3)...
361     %ones(1,length(K1))*K1(4)...
362     %ones(1,length(K1))*K1(5)...
363     ];
364
365 K2_table = [ K2...
366     %ones(1,length(K2))*K2(2)...
367     %ones(1,length(K2))*K2(3)...
368     %ones(1,length(K2))*K2(4)...
369     %ones(1,length(K2))*K2(5)...
370     ];
371
372 K3_table = [ K3...
373     %ones(1,length(K3))*K3(2)...
374     %ones(1,length(K3))*K3(3)...
375     %ones(1,length(K3))*K3(4)...
376     %ones(1,length(K3))*K3(5)...
377     ];
378
379 K4_table = [ K4...
380     %ones(1,length(K4))*K4(2)...
381     %ones(1,length(K4))*K4(3)...
382     %ones(1,length(K4))*K4(4)...
383     %ones(1,length(K4))*K4(5)...
```

```
384       ];
385
386
387   Settle_table_angle =  [ SettleTime_angle' ];
388
389   Overshoot_table_angle =  [ OvershootValue_angle' ];
390
391   Settle_table_disp =  [ SettleTime_disp' ];
392
393   Overshoot_table_disp =  [ OvershootValue_disp' ];
394
395
396
397
398   Table_results = table(K1_table',K2_table',K3_table',K4_table',
          Settle_table_angle',Overshoot_table_angle',...
399        Settle_table_disp',Overshoot_table_disp',...
400       'VariableNames',{'K1','K2','K3','K4','Settle_time_angle','Overshoot_angle'
              ,'Settle_time_disp','Overshoot_disp'})
```