SE420 Laboratory Assignment 7

Positioning Control of a Motor Using PD, PID, and Hybrid Control

Goals for this Lab Assignment:

1. Design a PD discrete control algorithm to allow the closed-loop combination of a controller and a previously identified plant to meet given specifications.
2. Learn methods to deal with noise and amplifier saturation by modification of the control design.
3. Discover the pros and cons of PD and PID controllers and learn how different design specifications may conflict.
4. Learn how hybrid control methods may alleviate the conflicts between different control methods.

Matlab Functions Used:

c2d, d2c

Prelab:

Read this entire lab before coming to class so you understand the different controllers to be designed.

Simulation (Must be Done with your Lab Partner):

The intent of this lab is to introduce the student to the design of a discrete control algorithm for the purpose of motor positioning. We will design a discrete Proportional-Derivative (PD) controller that will be implemented on the DC motor. This PD controller will then be modified into a PID and a hybrid control law.

Because we will be doing position control in this lab, we will need the **discrete** transfer function for the angular position of the motor given a voltage input: $\frac{Pos(z)}{U(z)}$. In Lab 5, we presented the discrete motor transfer function for velocity given a voltage input:

$$\frac{Vel(z)}{U(z)} = \frac{K[1-e^{-T/\tau_m}]}{z-e^{-T/\tau_m}} = \frac{c_2}{z-c_1}$$

To be mathematically correct, we should not directly add a z-transformed approximation of an integrator in line with our speed discrete transfer function because this creates two sampling points in our block diagram, more explanation on this in lab lecture. Instead, we must transform the $\frac{Vel(z)}{U(z)}$ z-transfer function into the s-domain, and then multiply by a continuous integrator in the s-domain. This gives us the correct control-to-position representation in the s-domain. We then transform back into the z-domain to obtain the position z-transfer function $\frac{Pos(z)}{U(z)}$ for the motor.

Problem 1: *Using the values for c_1 and c_2 that you obtained from Lab 5 for a sample-period of 0.001 second, find the z-transform function, $\frac{Pos(z)}{U(z)}$, for your motor. Use MATLAB to help you, using the commands 'c2d' and 'd2c'.*

This lab is the first of the semester where derivative feedback is used. A problem with this type of feedback is that the derivative amplifies signal noise. This is not a problem for your simulation since there is no noise in your simulated signals, but in any actual system you will be controlling (like the motor in lab) there may be a large amount of signal noise. A simple solution is to utilize a filtered form of the derivative, represented in the Laplace domain by the transfer function:

$$\frac{V(s)}{P(s)} = \frac{T_d \cdot s}{s + T_d}$$

With T_d large so the derivative does not roll-off until high frequencies. We will be using a value of $T_d = 500$ rad/sec. This approach adds a high-frequency roll-off to the derivative action so that instantaneous spikes are attenuated. This should look familiar to you because this transfer function is very similar to the first order filter we experimented with in lab 4 filtering the ADC channels. There is a difference though. Look at the s in the numerator. The job of this transfer function is to find the derivative of the signal applied to the filter when in addition filtering the signal.

Problem 2: *Using the "bode" command, compare the bode plots of the true differentiator "s" to this approximate derivative "500s/(s+500)". At what frequency (don't forget units) does this approximation stop comparing well with the true differentiator? For this continuous derivative approximation,* $\frac{V(s)}{P(s)} = \frac{500 \cdot s}{s + 500}$ *,to be used in your discrete controller it will need to be discretized. Use the Tustin approximation to find V(z)/P(z).*

In a previous lab, we found that our motor system exhibited nonlinear characteristics due to saturation. Typical controller design tools for linear systems might therefore predict a different system performance than what is measured. This can be a problem for this lab, but we can alleviate the saturation problems by choosing small enough amplitudes of step changes and by using a control approach that avoids large control inputs. In the classical implementation of PID control, the controller is completely in the forward loop as shown in Figure 1 below:

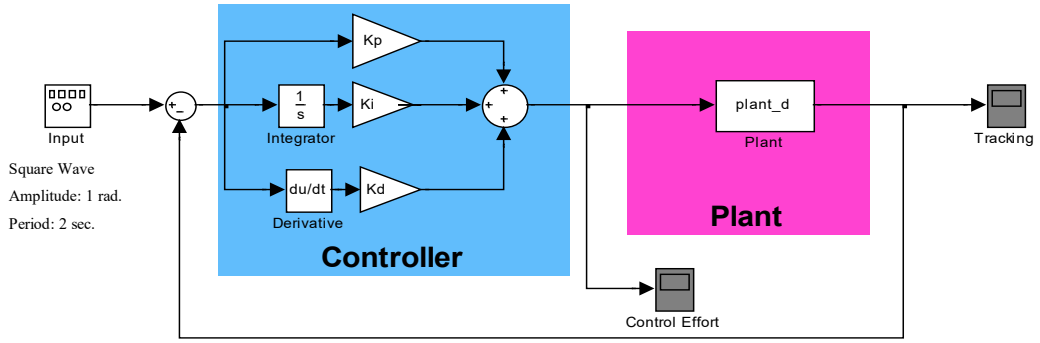

Figure 1: Typical PID control structure

There are several problems with PID control, some of which we have already seen. In Lab 6, we encountered problems with the integral term, and some of the goals of Lab 6 were to find solutions specifically to the integral-windup issue. When we consider adding a derivative term, an additional problem arises apart from noise. To see this problem, let us assume that we wish to analyze the controller performance using step inputs or square waves. For these signals, the derivative term becomes infinite for an infinitesimally short time. Even with a filtered derivative described above, the magnitude of the jump will cause a very large control signal to be sent to the plant for a short period.

The signal discontinuity of a step change generally isn't a problem in the continuous domain, but in the discrete domain the smallest time interval is determined by the sampling period. If the derivative term is very large for one sample period, and if one sample period is a large portion of the rise time, then the derivative term may contain a large portion of the control effort needed to move the motor from one position to another. With saturation, much of this control effort may be removed. If we design a controller with classical control techniques, the design will produce the expected result on the linear

plant, but not on the nonlinear system with the saturated input. The amount of control signal 'lost' during the saturation of the derivative is simply too high to adequately design the controller in our case.

If we think about the derivative term, it is present to prevent excessive velocity of the motor's motion. It is common practice to feedback the velocity of the motor directly, rather than use the derivative of the error. This produces a control structure with control elements present in both the forward and in the back portions of the feedback loop, shown in Figure 2 below:

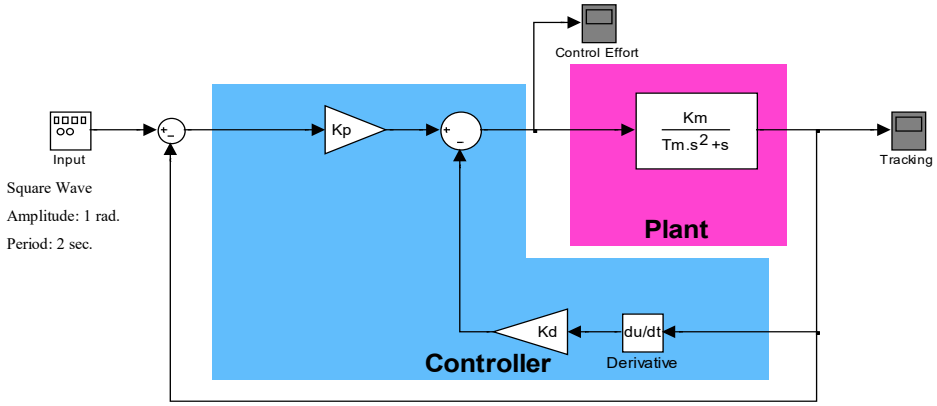

Figure 2: A PD controller with control components in the forward and back loops.

Compare this to the classical PD loop in Figure 3.

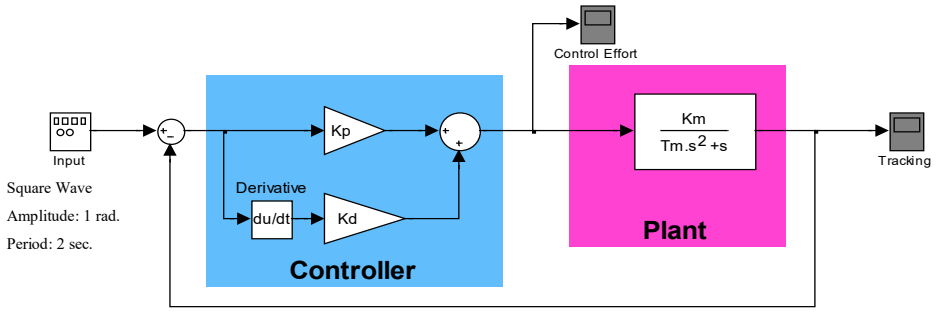

Figure 3: A PD controller with control components only in the forward loop (classical form).

Problem 3: *Using continuous-domain representation of the components (transfer functions in the Laplace operator 's' and set 'du/dt' to just 's'), derive the closed-loop transfer functions for the two PD controlled systems of Figure 2 and Figure 3. Use variables to represent the P-gain, D-gain, motor gain, and motor time constant (Kp, Kd, Km, and Tm respectively in above Figs.). Use just "s" for du/dt. Do they have the same poles? Do they have the same zeros?*

As an initial exercise we will design a PD controller of the type shown in Figure 2. Our design method we will be to use root locus techniques to adjust the Kp and Kd gains until a good set of gains are found. We will be using the function "controlSystemDesigner" to perform this design. Run the following commands, and note the comments, to launch **controlSystemDesigner:**

T = sisoinit(4); % The parameter 4 combines an error path controller C1 along with a feedback controller C2

T.G.Value = your discrete open loop transfer function Pos(z)/U(z) found earlier in problem 1.

T.OL1.View = {'rlocus'}; % Root Locus plot for error path

T.OL2.View = {'rlocus'} % Root Locus plot for feedback path

controlSystemDesigner(T) % Launch controlSystemDesigner with the above initializations.

A graphical user interface will load showing a window that looks like Figure 4. First change a preference in **controlSystemDesigner** telling it how to display the transfer functions you add to your controller. Select the "Preferences" button. Under the "Options" tab select the radio button "Zero/pole/gain" in the "Compensator Format" section and click "OK". Next you will not make a change but I want you to see what block diagram **controlSystemDesigner** is using and where your controllers C1 and C2 are located in the block diagram. *Note here that the C1 and C2 I am talking about are Control 1 and Control 2 in controlSystemDesigner, not the c1 and c2 parameters we identified in Lab 5.* Click the "Edit Architecture" button and another window will appear. Notice that controlSystemDesigner is using the fourth option. C1 has the error as its input and C2 has the motor's position as its input. (See Figure 4). In our case C1 is just our Kp gain. C2 is the portion of the controller's transfer function in the feedback path. In our case this is the discrete representation of the transfer function Kd*500s/(s+500) which you found in problem 2. The input to G (the plant) is C1 – C2. C1 can just start out at 1. Double click on C2 in the "Controllers and Fixed Blocks" section and enter C2's poles, zeros and gain by right clicking in the "Dynamics" white space. Then in the Open Loop 1 root locus plot, error path, you can click on your pole locations (pink squares) and drag them to a desired location. C1's gain changes when you drag the poles to a new location. In Open Loop 2 root locus plot, feedback path, you can click on the pole locations (pink squares) and drag them to a desired location. C2's gain changes as you drag these poles in the Open Loop 2 root locus plot. As you are tuning it is nice to plot the step response of the system. Find on the left under Responses find T_r2y. Right click on T_r2y and select Plot and then Step. This will produce the step response. Reposition this plot so that it is under your Open Loop 1 root locus plot. Another useful plot is the control effort step response T_r2u so you can see how large of control effort is needed. Find on the left under Responses find T_r2u. Right click on T_r2u and select Plot and then Step. This will produce the step response. Reposition this plot so that it is under your Open Loop 2 root locus plot.

For the design, start with a Kp=1 and Kd=1. Next use **controlSystemDesigner** to adjust Kp in the Open Loop 1 root locus plot (C1's gain) and Kd in the Open Loop 2 root locus plot (C2's gain). As you adjust the pole locations / gains watch the system response T_r2y until you achieve the design specifications given below.

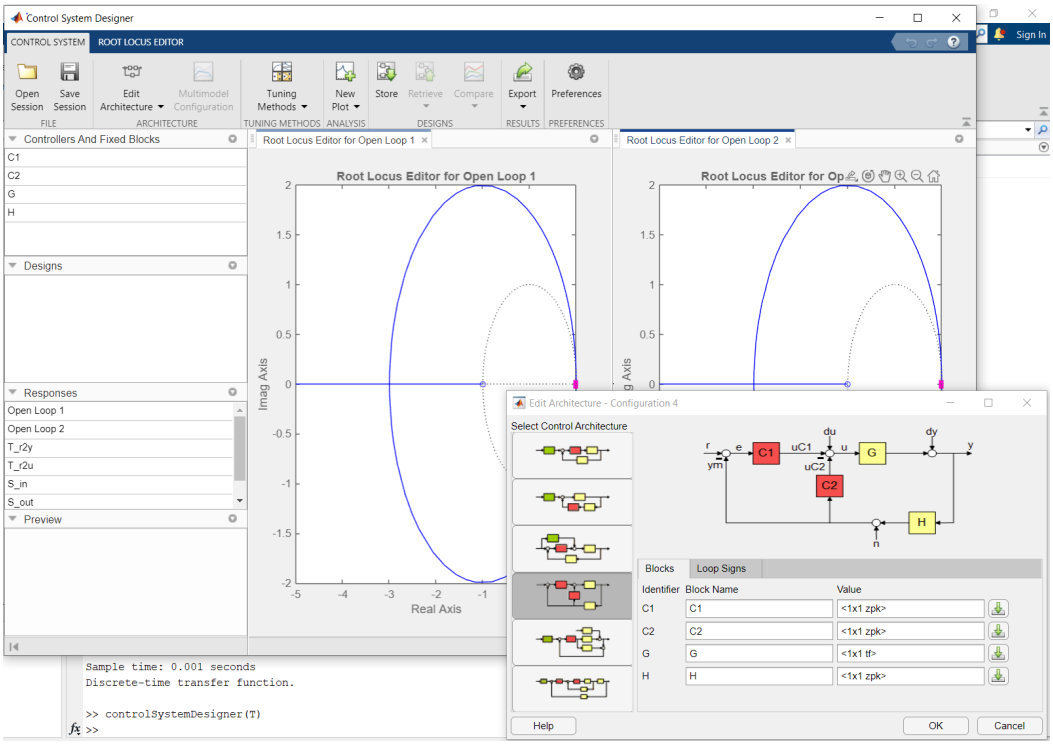

Figure 4: The SISO Design Tool Matlab Command "controlSystemDesigner"

Problem 4: *Using the discrete plant representation for motor position control from problem 1 and the Tustin derivative approximation from Problem 2, use "*controlSystemDesigner*" to design your controller. Initially pick a starting derivative gain, Kd, =1, and use "*controlSystemDesigner*" to find values for Kp and Kd such that the following design specifications are met:*

1) *Less than 1% overshoot for a step or square-wave input*
2) *Settling time greater than 200 ms and less than 350 ms*
3) *Note: In controlSystemDesigner if you right click in white space of the Root Locus plot you can specify these design requirements and the plot will show you, in white, where your closed loop poles should be located to meet the requirements.*

Once you have a set of Kp and Kd gains, simulate your controller in SIMULINK in the discrete domain, and verify that the design specifications for tracking of 1 radian amplitude square wave are met. Are the specifications still met if you limit the control effort with a saturation term to be within +/- 10 units?

Laboratory Exercise

Note: For this lab we will not be using the friction compensation we found in Lab 6.

PD Control

1. Implement the controller you just designed in controlSystemDesigner and simulated in Simulink on the actual motor and compare simulated data to actual data. Use "matlab/simulink5ms_plotAndGains.slx" to plot the motor's position and control effort applied to the motor. Use a reference square-wave of amplitude 1 radian with a

period of 2 seconds. Experiment with the Kp and Kd gains to get a feel of how each gain changes the systems response. Show your controller to your TA and answer question 2 of the lab check-off.

PID Control

2. Add an integral error term as you did with your velocity controller in Lab 6 (use the Tustin approximation to implement the integral). Also add code to prevent integral windup. Use the Expressions Window in Code Composer Studio to modify and tune Kp, Ki, and Kd gains and additionally your reference square-wave amplitude.
3. Again experiment with the Kp, Kd and Ki gains to get a feel on how each effects the system response. With a Ki gain of 100, try to tune the Kp, Kd gains to eliminate steady-state error within 0.5 seconds of the step change. Do question 3 of the lab check-off.
4. How to design PI plus velocity feedback controller in controlSystemDesigner. First show that the PI controller transfer function emulated with the Tustin rule can be written in the form $K_{Loop} * \frac{z-zero_c}{z-1}$, with $K_{Loop} = \frac{2*\mathrm{Kp}+\mathrm{Ki}*\mathrm{T}}{2}$ and $zero_c = -(\frac{Ki*T-2*Kp}{2*Kp+Ki*T})$. KLoop and zeroc are the design values you can tune in controlSystemDesigner to design the PI portion of the controller. (You will use this in your controller design in Lab 8 so make sure to save this derivation.) For this lab you will not perform any design in controlSystemDesigner but just take the values you found for Kp and Ki and add the PID control to your controlSystemDesigner design of your PD (Proportional plus velocity feedback) controller and show that it has a similar response to your motor's actual response.

Hybrid Control

The previous PID controller implementation shows that the PID and the PD have conflicting effects on the tracking performance. One method to gain the advantages of each control method is to use a 'hybrid' or 'switching' controller. A hybrid controller has a very specific mathematical definition, but for purposes of this lab we may think of hybrid control as simply switching between different control methods to gain the advantages of both. We will demonstrate this by switching between PD and PID control of the motor. Obviously, hybrid control is especially concerned with the switching conditions. In this lab we will measure how fast the tracking error is changing, or the time-derivative of our error, and use this to define a switching condition.

5. Modify your code to run two controllers: one PD with lower gains and one PID with higher gains.
    a. The PD lower gain controller should be switched on when the magnitude of the derivative of the tracking error is higher than or equal to some switch point. We will make this 'switch-point' a value you will tune. The derivative term in the PD controller will still be the derivative of the motor position as you did in simulation, but the switching condition depends on the derivative of the tracking error. Find the derivative of the tracking error with the same derivative approximation of 500s/(s+500) emulated with the Tustin rule.

b. The "PID high gain" controller should be switched on when the magnitude of the error-dot term is less than the switch point. You will find the controller runs best if the integral-sum is zeroed each time the PD controller is switched on. DO NOT integrate the error when the PD lower gain controller is running.

6. While attempting to track the square-wave from before, tune your Kp, Ki, Kd, square-wave amplitude, and switch-point values to satisfy the following conditions for tracking a 1 radian amplitude square wave:
    a. No steady-state error after 400 ms of any change in reference position.
    b. Rise time in the range 150ms < tr < 200ms.
    c. Ki gain greater than 500. This will make the system robust to disturbances. Of course the disturbances can't be too large because we are dealing with a small motor.
    d. Overshoot less than 1.5%
7. Make a plot of your controller for check-off.
8. Do question 4 of the lab check-off.

Lab Check Off:

1. Demo your simulation.
2. Demo your PD controller implemented on the DSP/motor system. How did your simulation results compare to implementation results (see questions from Exercise 1)? You should have two plots (or better, one with both data sets plotted) to show to your TA to compare: one simulation, one implementation. How do they compare? Do you see overshoot? The same rise time? Do you see significant saturation in the control effort? Do you observe any steady-state error?
3. Demo your PID controller to your TA. Is it possible to have an aggressive Ki gain (greater than 100) yet still achieve zero overshoot? What is the percent overshoot when you apply a 6 radian amplitude square wave? 12 radian amplitude? 18 radian amplitude?
4. Demo your Hybrid controller. What is the Hybrid controller's percent overshoot when you apply a 6 radian amplitude square wave? 12 radian amplitude? 18 radian amplitude? How is the hybrid control better than PD? To answer this question, consider the following: How is the Hybrid controller better than the PID control when tracking different amplitude square waves (look at overshoot, settle time, rise time, etc)? Are the gains of your Hybrid controller higher or lower than the PID control? What happens if you tune your Ki gain too high? What happens if you choose your switch-point too high? Too low?