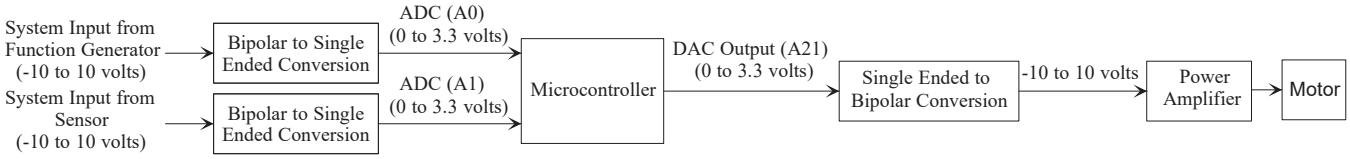
Lab 7: Digital Control

We will use the Teensy 3.6 microcontroller, the Single Ended to Bipolar (SE2B) circuit and the two Bipolar to Single Ended (B2SE) circuits we developed in the last lab to implement a digital controller first for velocity control of the motor we have been studying and then for position control of a DVD system.

Make sure the B2SE and SE2B circuits from the last lab are thoroughly tested, are organized neatly, and the input and output of each of the B2SE and SE2B circuits are clearly labeled. This will be very helpful in this week's lab.

Motor Velocity Control

1. Double check your Bipolar to Single Ended circuits (you need two for this lab) and your Single Ended to Bipolar circuit before making any connections to the microcontroller. Make the necessary connections to the microcontroller and the other components outlined below. The sensor for this lab is the digital tachometer and should be run through an op amp buffer before connection to the Bipolar to Single ended circuit. See Appendix 1 and 2.



   Note the feedback is implemented within the microcontroller code in the Microcontroller block.

2. The program template that we will use to implement the controllers on the Teensy microcontroller is "Control_Loop_Motor_Velocity_Control_P.ino" (Appendix 3). This program is setup for a simple proportional controller but the Controller can be changed by replacing the proportional controller difference equation with the difference equation for a different controller as illustrated in "Control_Loop_Motor_Velocity_Control_PI.ino" (Appendix 4) which implements the difference equation for a PI controller. Note the only thing that needs to be changed when changing to a different controller is the difference equation and the control parameters for the different controller. These lines are highlighted in the code.

3. In the steps below use a step input to characterize the system response. The theoretical results below refer to the digital model of the system including the effects of the sampling not the continuous system.

4. Investigate the proportional controller first. Start by setting the gain K to a reasonable value (from your previous continuous controller lab). Compare the experimental results from your previous continuous controller lab with the experimental determined digital controller results determined here. Note any differences.

   Continuing with the proportional controller, investigate various gain and sampling time combinations and compare these experimental results to the theoretically determined digital results. The sampling time can be changed by adding a delay to the control loop using the **delayMicroseconds()** function. Also investigate different values of the maximum voltage limit and its effect on the system response **(the maximum voltage limit is 5.0 volts)**. Discuss any differences between the experimental and theoretical results.

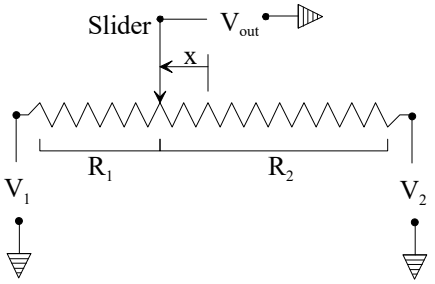
5. Implement a digital version of one of the other controllers that was built for your continuous system (Lead, Lag, PI, PD, etc.) and compare these experimental results with the theoretical digital results and with the experimental results for the continuous system determined previously. Discuss any differences. Note that this is not yet a fully digital design but rather the conversion of your previously designed continuous controller to a digital controller (Appendix 5). We will do a fully digital design in the next lab section.

   In the next lab section, we will investigate position control which is not open loop stable (an input voltage will not yield an output position without feedback).

Digital Position Control of a DVD System using a Linear Potentiometer

In this lab section the position control is investigated using a DVD drive system. Since the position feedback of the DVD system comes from encoded marks on the DVD another way of measuring position is necessary. This is accomplished by adding a linear potentiometer to measure the position of the moving laser head in the DVD system.

The figure illustrates a linear potentiometer. A potentiometer consists of a moving slider that contacts and divides a resistance into two portions R_1 and R_2. The output of the potentiometer is usually measured at the slider as shown. The position of the slider is measured from the center of the potentiometer.



Linear Potentiometer

The resistances R_1 and R_2 are related to the displacement x by the following equations.

$$R_1 = \frac{R}{2}\left(1 - \frac{x}{L}\right) \qquad R_2 = \frac{R}{2}\left(1 + \frac{x}{L}\right)$$

Where R is the total resistance of the potentiometer and L is the length of the potentiometer.

The output voltage V_{out} is equal to:

$$V_{out} = \frac{R_2}{R_1 + R_2}\left(V_1 - V_2\right) + V_2$$

This can be written in terms of the potentiometer displacement x as follows:

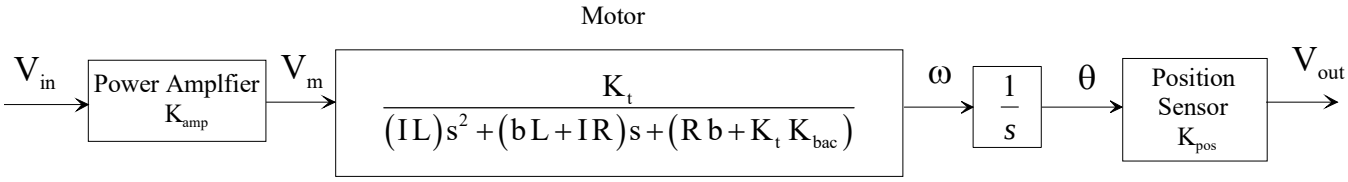
$$V_{out} = \left(\frac{1 + \frac{x}{L}}{2}\right)\left(V_1 - V_2\right) + V_2$$

For our system V1=15 volts, V2 = -15 volts and L=60 mm (2.36 in.). For x measured in mm the output potentiometer voltage is:

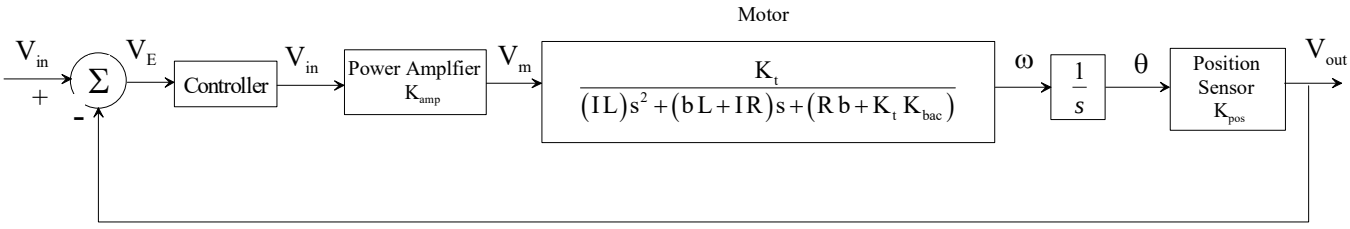
$$V_{out} = \left(\frac{1 + \frac{x}{60}}{2}\right)(30) - 15$$

For this choice of parameters V_{out} will be equal to zero when x is equal to zero at the center of the potentiometer.

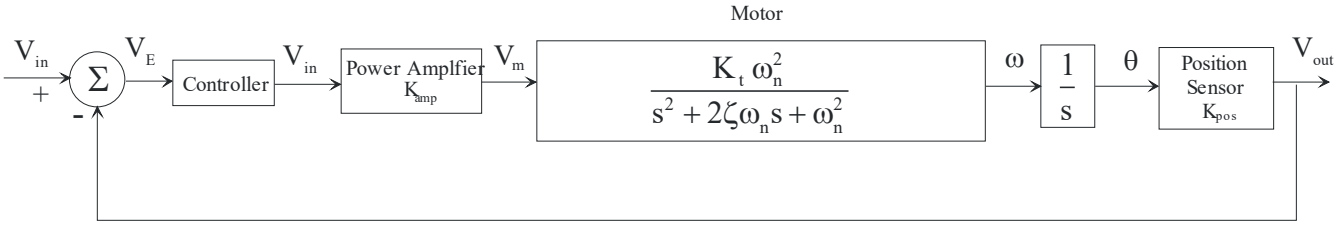
There are two significant differences with position control using a DC motor. The first is that since the input voltage is proportional to the motor velocity the open loop system is not stable for position. So the closed loop system must be used to identify the system characteristics. Usually the controller used is a simple proportional controller. Once the closed loop transfer function has been identified it can be used to calculate the open loop transfer function which can be used to design the controller.



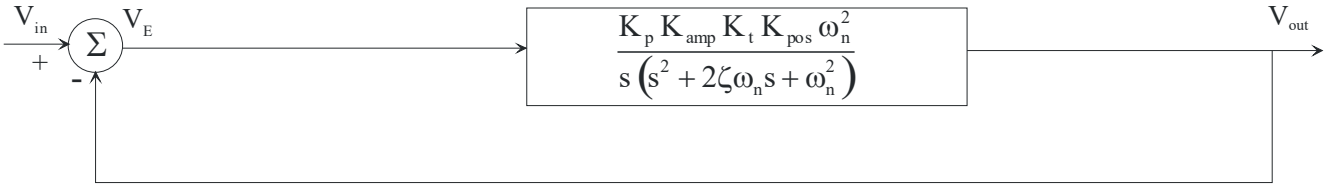
Open Loop Position Control using a DC Motor (Unstable)



Closed Loop Position Control using a DC Motor (Stable)

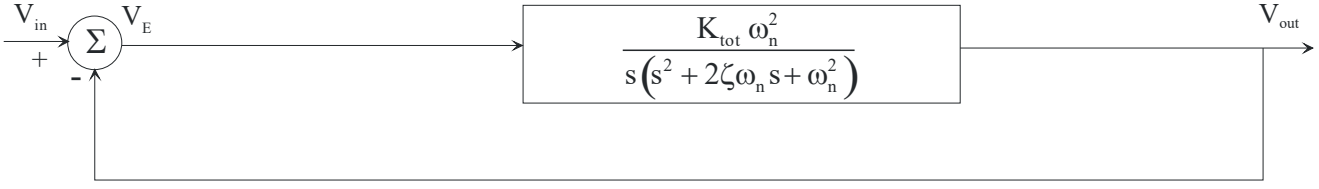


Closed Loop Position Control with Motor written in Standard Second Order Form

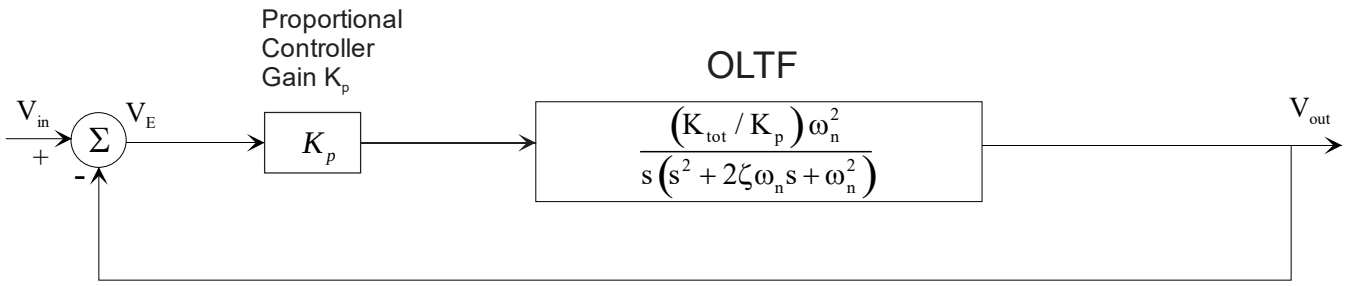


Closed Loop Position Control with a Proportional Controller (K_p)

Let $K_{tot} = K_p K_{amp} K_t K_{pos}$



Closed Loop Position Control with K_{tot}



Closed Loop Position Control with K_p factored out

So the open loop transfer function is:

$$OLTF = \frac{\left(K_{tot}/K_p\right)\omega_n^2}{s\left(s^2+2\zeta\omega_n s+\omega_n^2\right)}$$

Where the division by K_p in the equation above is necessary because the final form of the OLTF should not include the controller gain K_p.

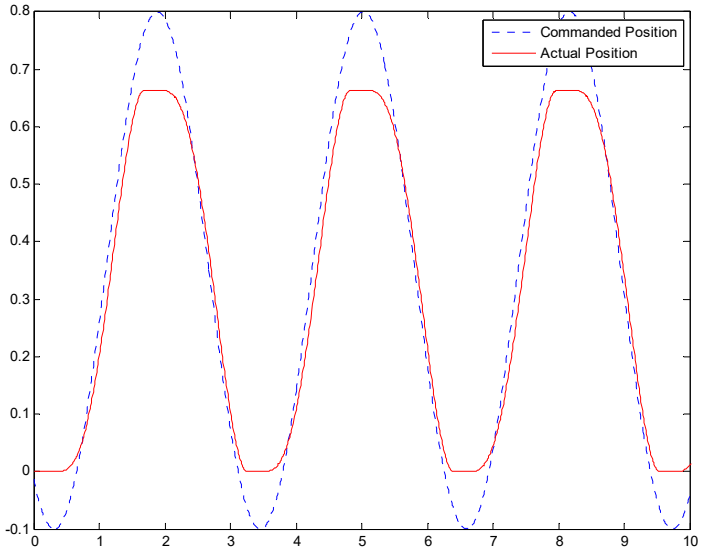
The closed loop transfer function is:

$$CLTF = \frac{V_{out}}{V_{in}} = \frac{K_{tot}\,\omega_n^2}{s^3+2\zeta\omega_n s^2+\omega_n^2 s+K_{tot}\,\omega_n^2}$$

Use our standard method to determine the frequency response of the closed loop system with Kp = 1.0. Fit a curve to this response using the closed loop transfer function to determine the three model parameters $\left(K_{tot},\zeta,\omega_n\right)$. Note that K_{tot} serves a different role in this transfer function. It does not simply shift the frequency response curve up or down but changes the shape of the curve. This occurs because K_{tot} appears in the denominator as well as the numerator. Once the model parameters have been determined using the closed loop transfer function the open loop transfer function, which is necessary to design the controller, can be found from the equation above.

This process is necessary because for position control the system is not open loop stable so the open loop transfer function cannot be determined directly as was the case with velocity control.

Another significant difference with position control is that often friction will cause the response of the system to fall short of the desired response especially at low speeds. The figure below illustrates this. Therefore, friction presents an additional challenge when designing the control system.

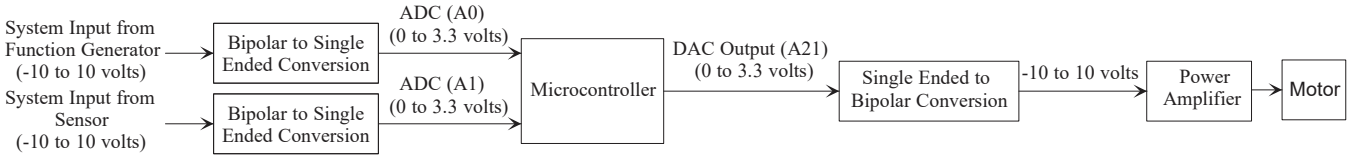


Commanded Position and Actual Position with Friction

Procedure

1. In the steps below the **maximum input amplitude should be no greater than 3 volts**. Remember the input voltage corresponds to a physical position not a velocity. For this system **limit the highest frequency in your investigation to 20 Hz**.

2. Double check your Bipolar to Single Ended circuits (you need two for this lab) and your Single Ended to Bipolar circuit before making any connections to the microcontroller. Make the necessary connections to the microcontroller and the other components outlined below. The sensor for this lab is the potentiometer output (position measurement) and should be run through an op amp buffer before connection to the Bipolar to Single ended circuit. See Appendix 2 and Appendix 6 Wiring Hookups for DVD Setup.



Note the feedback is implemented within the microcontroller code in the Microcontroller block.

3. The program template that we will use to implement the controllers on the Teensy microcontroller is "Control_Loop_Motor_Velocity_Control_P.ino" (Appendix 3). This program is setup for a simple proportional controller but the Controller can be changed by replacing the proportional controller difference equation with the difference equation for a different controller as illustrated in "Control_Loop_Motor_Velocity_Control_PI.ino" (Appendix 4) which implements the difference equation for a PI controller. Note the only thing that needs to be changed when changing to a different controller is the difference equation and the control parameters for the different controller. These lines are highlighted in the code.

4. Go through the startup and checking procedure (in the following section)

5. Using the standard method Bode method, determine the transfer function of the motor and position system. **Be careful with the DVD systems don't run them to long or to fast as this will wear them out quickly.** Note that program "Control_Loop_Motor_Velocity_Control_P.ino" implements a proportional controller with the feedback loop and the transfer function for the tested system will be the closed loop transfer function. This can be used to calculate the open loop transfer function necessary for the controller design. **Limit the highest frequency in your investigation to 20 Hz**.

6. Using the **Closed Loop Transfer Function** determined above determine the **Open Loop Transfer Function** and use this to design and implement a digital controller to generate an accurate and fast position response for the DVD system not considering friction. Consider different types of inputs including a step and ramp response. **The design method used to design this controller should be one of the design methods we learned earlier i.e. Bode, Root Locus or State Space. Do not use trial and error to design this controller. The controller design should be completely digital, do not design a continuous controller and convert it to a digital controller.** See Appendix 7 for the procedure to convert a digital controller to a difference equation**. Also the design process is important. As part of your lab check off you should present the process you used to arrive at your final controller.**

7. Use the simulink program provided to design a controller for the simulated motor with friction system to reduce the position errors while maintaining a fast response. The three necessary files are:

   Motor_Position_Modelw_friction_6m_Lab_digital.m
   Motor_Position_Modelw_friction_6_open_loop_digital.slx
   Motor_Position_Modelw_friction_6_closed_loop_digital.slx

   These files are located on the website.

8. Modify the controller designed above to improve the position accuracy considering friction while maintaining an accurate and fast position response. Consider different types of inputs including a step and ramp response. Use the experience gained in steps 6 and 7. Note that the effect of friction is amplified for slow movements. So to investigate the effect of friction study slow movements.

Potentiometer Output

Be careful with the potentiometer output. The potentiometer output is the slider of the potentiometer and cannot tolerate much current. So be careful not to connect the potentiometer output to the input voltages as this will damage or destroy the potentiometer.

Startup and Checking Procedure

Follow the startup procedure below. Do not simply hook everything up and hope the connections are correct. Both the potentiometer and the DVD drive system can be easily damaged.

1. Hookup the +15 and -15 connections to the potentiometer. Measure the voltage at the potentiometer output while gently moving the DVD head. Note the negative or ground for this measurement is the power supply ground. The voltage should be zero near the middle and vary as the DVD head is moved. If this is not the case fix the problem before moving on.

2. Turn the power off. And hook up everything else except the motor + and – connection. Connect the motor voltage to the oscilloscope and turn the power back on and run a sin wave input on the function generator (say 2 volts peak to peak at 5 Hz) and to make sure that the voltage to the motor is correct. If this is not the case fix the problem before moving on.

3. Turn the power off and hookup the motor. Turn the power on and run a sin wave input on the function generator (same amplitude and frequency as in step 2). If the DVD head goes completely to the left or right turn the power off immediately and figure out what is wrong. The DVD drive gears can be easily stripped if driven to the stops.

4. Once everything checks out above do the Bode analysis of the DVD drive

Hookup Wires

Use the hookup wires with the banana plugs on one end that we used for the previous labs to connect the power to the power amplifier and to the potentiometer. You will need to use regular hookup wire for the other connections.

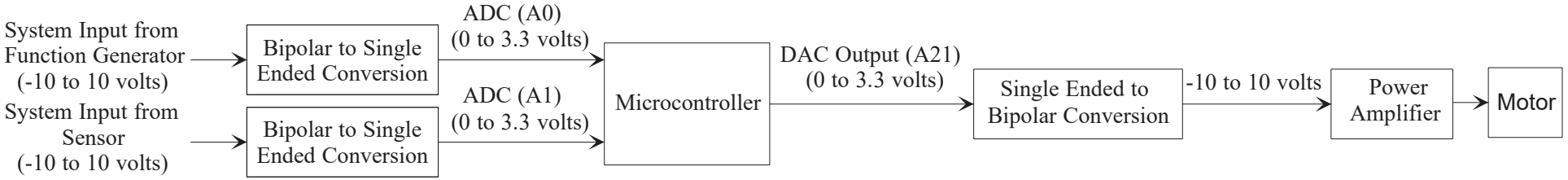
Hand in:

During your lab demonstration next week show the various plots you were asked to generate in this lab as well as any values you were asked to determine in this lab and be prepared to demonstrate any of the circuits you built in this lab. Have the plots and values organized so that you can present them efficiently. For other information that should be documented to demonstrate you completed a particular part of the lab use the screen capture capability of the oscilloscope. In the storage menu choose 8-Bitmap instead of CSV to get a bit map of the screen. Note you still need to get the data with CSV to compare with theoretical values.

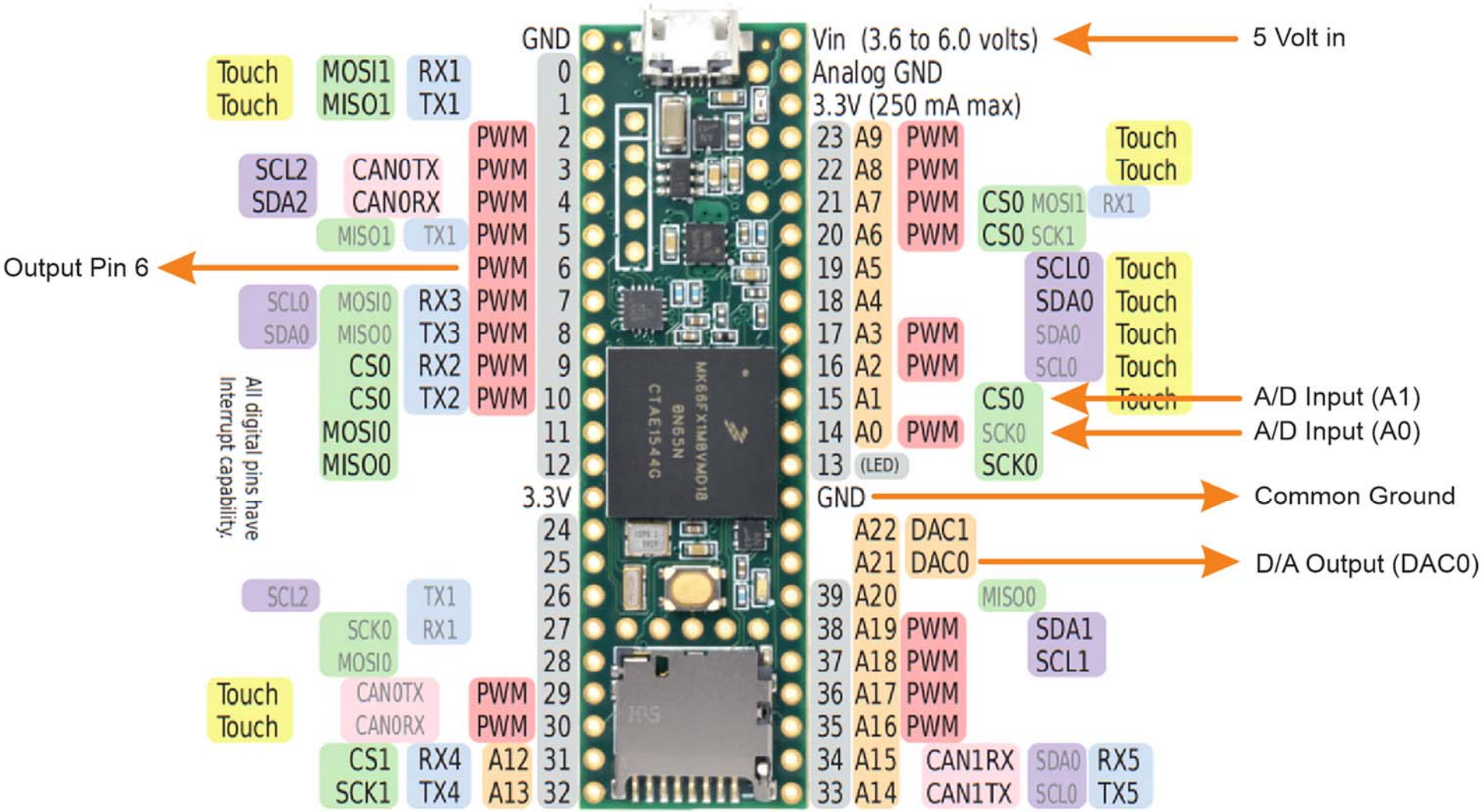
Lab Demonstration Part 1
(through the Motor Velocity Control portion of the Lab): Wednesday May 26

Lab Demonstration Part 2 (remainder of the lab): Wednesday June 2

Appendix 1: Microcontroller Connections



Appendix 2: Microcontroller Connections



Note the power connection (5 Volt in) is not necessary as long as the USB cable is plugged in.

Appendix 3: Control_Loop_Motor_Velocity_Control_P.ino

```
//  AD_DA_Test.ino

// ADC 16 bits 0-65535 and 0-3.3 volts
// DAC 12 bits 0-4095  and 0-3.3 volts (conversion factor between DAC and ADC = 16)

#include <ADC.h>
#include <ADC_util.h>

ADC *adc = new ADC(); // adc object

unsigned int System_Input, Sensor_Input, Control_Output;
float Max_Voltage, Kp, System_Input_v, Sensor_Input_v, Error, Control;

void setup()
{
  Serial.begin(38400);
  pinMode(6, OUTPUT); // Set Pin 6 as an Output Pin

   ///// ADC0 ////
  // reference can be ADC_REFERENCE::REF_3V3, ADC_REFERENCE::REF_1V2 or ADC_REFERENCE::REF_EXT.
  adc->adc0->setReference(ADC_REFERENCE::REF_3V3);

  adc->adc0->setResolution(16); // set bits of resolution

  // ADC_CONVERSION_SPEED  =  VERY_LOW_SPEED, LOW_SPEED, MED_SPEED, HIGH_SPEED_16BITS, HIGH_SPEED or VERY_HIGH_SPEED
  // ADC_SAMPLING_SPEED    =  VERY_LOW_SPEED, LOW_SPEED, MED_SPEED, HIGH_SPEED or VERY_HIGH_SPEED

  adc->adc0->setConversionSpeed(ADC_CONVERSION_SPEED::HIGH_SPEED); // Set the conversion speed
  adc->adc0->setSamplingSpeed(ADC_SAMPLING_SPEED::HIGH_SPEED);     // Set the sampling speed

  adc->adc0->recalibrate(); // Recalibrate adc0

  pinMode(A0,  INPUT);  // Set pin A0 as an input (AD)
  pinMode(A1,  INPUT);  // Set pin A1 as an input (AD)
  pinMode(A21, OUTPUT); // Set pin A21 as output (DA)
  pinMode(A22, OUTPUT); // Set pin A22 as output (DA)

  analogWriteResolution(12); // Set DA resolution (12 bits)

  Max_Voltage = 5.0;   // Maximum Voltage = 5 volts because of the gain of 2 of the power amplifier
  Kp          = 1.0;   // Proportional control constant
}
```

```
void loop()
{

    System_Input = adc->adc0->analogRead(A0);                   // Read Analog value channel A0 (AD) - Function Generator Input
                                                                // (0-3.3 volts) after the B2SE circuit

    System_Input_v = (float) System_Input*(20./65535.)- 10.0;   // Convert the System_Input digital value (0-65535) to a voltage
                                                                // Note the input (0-65535) digital range is mapped to a bipolar
                                                                // +- 10 volts range
                                                                // Note that the (20./65535.) term needs the decimal point
                                                                // or else it is interrupted as an integer and the result is zero
                                                                // The underscore v (_v) notation denotes +- 10 volt units

    Sensor_Input = adc->adc0->analogRead(A1);                   // Read Analog value channel A1 (AD) - Sensor Input (0-3.3 volts)
                                                                // after the B2SE circuit
    Sensor_Input_v = (float) Sensor_Input*(20./65535.)- 10.0;   // Convert the Sensor_Input digital value (0-65535) to a voltage
                                                                // Note the input (0-65535) digital range is mapped to a bipolar
                                                                // +- 10 volts range
                                                                // Note that the (20./65535.) term needs the decimal point
                                                                // or else it is interrupted as an integer and the result is zero
                                                                // The underscore v (_v) notation denotes +- 10 volt units


    Error   = (System_Input_v - Sensor_Input_v);                // Error (units are voltage +- 10 volts)     This equation
                                                                // implements the Feedback
    Control = Kp * Error;                                       // Control (units are voltage  +- 10 volts)  This equation
                                                                // implements the control equation


    if(fabs(Control) >= Max_Voltage)                            // Check Maximum voltage
    Control = copysign(Max_Voltage,Control);

    Control_Output = floor((Control + 10.)*4095./20.);          // Convert control voltage to a digital number for output
                                                                // Note the output bipolar range +- 10 Volts is mapped to the
                                                                // digital range 0 to 4095


    analogWrite(A21,Control_Output);                            // Write digital value (DA)

    digitalWriteFast(6, !digitalReadFast(6));                   // Toggle output pin 6 for cycle timing

}
```

The highlighted lines above are the only lines in the code that need to be changed to implement a different controller.

Appendix 4: Control_Loop_Motor_Velocity_Control_PI.ino

```
//  AD_DA_Test.ino

// ADC 16 bits 0-65535 and 0-3.3 volts
// DAC 12 bits 0-4095  and 0-3.3 volts (conversion factor between DAC and ADC = 16)

#include <ADC.h>
#include <ADC_util.h>

ADC *adc = new ADC(); // adc object

unsigned int System_Input, Sensor_Input, Control_Output;
float Max_Voltage, System_Input_v, Sensor_Input_v, Error, Control, Kp, Ki, Ts;
float Error_m1, Control_m1, Error_m2, Control_m2, Error_m3, Control_m3, Error_m4, Control_m4;

void setup()
{
  Serial.begin(38400);
  pinMode(6, OUTPUT); // Set Pin 6 as an Output Pin

   ///// ADC0 ////
  // reference can be ADC_REFERENCE::REF_3V3, ADC_REFERENCE::REF_1V2 or ADC_REFERENCE::REF_EXT.
  adc->adc0->setReference(ADC_REFERENCE::REF_3V3);

  adc->adc0->setResolution(16); // set bits of resolution

  // ADC_CONVERSION_SPEED  =  VERY_LOW_SPEED, LOW_SPEED, MED_SPEED, HIGH_SPEED_16BITS, HIGH_SPEED or VERY_HIGH_SPEED
  // ADC_SAMPLING_SPEED    =  VERY_LOW_SPEED, LOW_SPEED, MED_SPEED, HIGH_SPEED or VERY_HIGH_SPEED

  adc->adc0->setConversionSpeed(ADC_CONVERSION_SPEED::HIGH_SPEED); // Set the conversion speed
  adc->adc0->setSamplingSpeed(ADC_SAMPLING_SPEED::HIGH_SPEED);     // Set the sampling speed

  adc->adc0->recalibrate(); // Recalibrate adc0

  pinMode(A0,  INPUT);  // Set pin A0 as an input (AD)
  pinMode(A1,  INPUT);  // Set pin A1 as an input (AD)
  pinMode(A21, OUTPUT); // Set pin A21 as output (DA)
  pinMode(A22, OUTPUT); // Set pin A22 as output (DA)

  analogWriteResolution(12); // Set DA resolution (12 bits)

  Max_Voltage = 5.0;      // Maximum Voltage = 5 volts because of the gain of 2 of the power amplifier
```

```
  Error_m1    = 0.0;      // Error minus 1 (Initial Error value one time step back)
  Error_m2    = 0.0;      // Error minus 2 (Initial Error value two time steps back)
  Error_m3    = 0.0;      // Error minus 3 (Initial Error value three time steps back)
  Error_m4    = 0.0;      // Error minus 4 (Initial Error value four time steps back)

  Control_m1  = 0.0;      // Control minus 1 (Initial Control value one time step back)
  Control_m2  = 0.0;      // Control minus 2 (Initial Control value two time steps back)
  Control_m3  = 0.0;      // Control minus 3 (Initial Control value three time steps back)
  Control_m4  = 0.0;      // Control minus 4 (Initial Control value four time steps back)

  Kp          = 1.0;      // Proportional control constant - must be calculated from your controller design and set properly here
  Ki          = 1.0;      // Integral control constant - must be calculated from your controller design and set properly here
  Ts          = .006;     // Sampling time (seconds) - must be measured and set properly here
}


void loop()
{

    System_Input = adc->adc0->analogRead(A0);                   // Read Analog value channel A0 (AD) - Function Generator Input
                                                                // (0-3.3 volts) after the B2SE circuit

    System_Input_v = (float) System_Input*(20./65535.)- 10.0;   // Convert the System_Input digital value (0-65535) to a voltage
                                                                // Note the input (0-65535) digital range is mapped to a bipolar
                                                                // +- 10 volts range
                                                                // Note that the (20./65535.) term needs the decimal point
                                                                // or else it is interrupted as an integer and the result is zero
                                                                // The underscore v (_v) notation denotes +- 10 volt units

    Sensor_Input = adc->adc0->analogRead(A1);                   // Read Analog value channel A1 (AD) - Sensor Input (0-3.3 volts)
                                                                // after the B2SE circuit
    Sensor_Input_v = (float) Sensor_Input*(20./65535.)- 10.0;   // Convert the Sensor_Input digital value (0-65535) to a voltage
                                                                // Note the input (0-65535) digital range is mapped to a bipolar
                                                                // +- 10 volts range
                                                                // Note that the (20./65535.) term needs the decimal point
                                                                // or else it is interrupted as an integer and the result is zero
                                                                // The underscore v (_v) notation denotes +- 10 volt units


    Error   = (System_Input_v - Sensor_Input_v);                // Error (units are voltage +- 10 volts)     This equation
                                                                // implements the Feedback

    // Control = Kp * Error;                                    // Control (units are voltage  +- 10 volts)  This equation
                                                                // implements the control equation
```

```
Control = (-Kp + 1./2.*Ki*Ts)*Error_m1 + (Kp + 1./2.*Ki*Ts)*Error + Control_m1;      // Control (units are voltage  +- 10
                                                                                     // volts)  This equation implements the
                                                                                     // control equation (PI Control)

// Save the previous values of the Error and Control signal values for potential use in the Control Difference Equation above
// Note that only Error_m1 and Control_m1 are needed for the control equation above but a more complicated control difference
// equation may require more previous values

    Error_m4   = Error_m3;
    Error_m3   = Error_m2;
    Error_m2   = Error_m1;
    Error_m1   = Error;


    Control_m4 = Control_m3;
    Control_m3 = Control_m2;
    Control_m2 = Control_m1;
    Control_m1 = Control;


    if(fabs(Control) >= Max_Voltage)                        // Check Maximum voltage
    Control = copysign(Max_Voltage,Control);

    Control_Output = floor((Control + 10.)*4095./20.);      // Convert control voltage to a digital number for output
                                                            // Note the output bipolar range +- 10 Volts is mapped to the digital
                                                            // range 0 to 4095

    analogWrite(A21,Control_Output);                        // Write digital value (DA)

    digitalWriteFast(6, !digitalReadFast(6));               // Toggle output pin 6 for cycle timing

}
```

The highlighted lines above are the only lines in the code that need to be changed to implement a different controller.

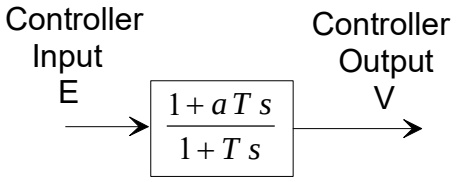
Appendix 5: Conversion of a Continuous Controller to a Digital Controller Lead or Lag Control

A continuous controller can be converted to a digital controller using any of the approximate methods (forward difference, backward difference, etc.) The most accurate of the approximate methods corresponds to the trapezoidal rule and is also called Tustin's transformation.

$$s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right) \tag{1}$$

Where Ts is the sample time.

For a lead or lag controller:



$$\frac{V}{E} = \frac{1+aTs}{1+Ts} \tag{2}$$

Equation 1 is used to substitute for s in equation 2.

$$\frac{V}{E} = \frac{1+aT\dfrac{2}{T_s}\left(\dfrac{z-1}{z+1}\right)}{1+T\dfrac{2}{T_s}\left(\dfrac{z-1}{z+1}\right)} = \frac{(T_s+2aT)z+T_s-2aT}{(T_s+2T)z+T_s-2T} \tag{3}$$

$$V\left[(T_s+2T)z+T_s-2T\right] = E\left[(T_s+2aT)z+T_s-2aT\right] \tag{3a}$$

Note from the definition of the z operator $V(k)\,z = V(k+1)$ *and* $E(k)\,z = E(k+1)$

$$(T_s+2T)\,V(k+1)+(T_s-2T)V(k) = (T_s+2aT)E(k+1)+(T_s-2aT)E(k) \tag{3b}$$

Solving for V(k+1)

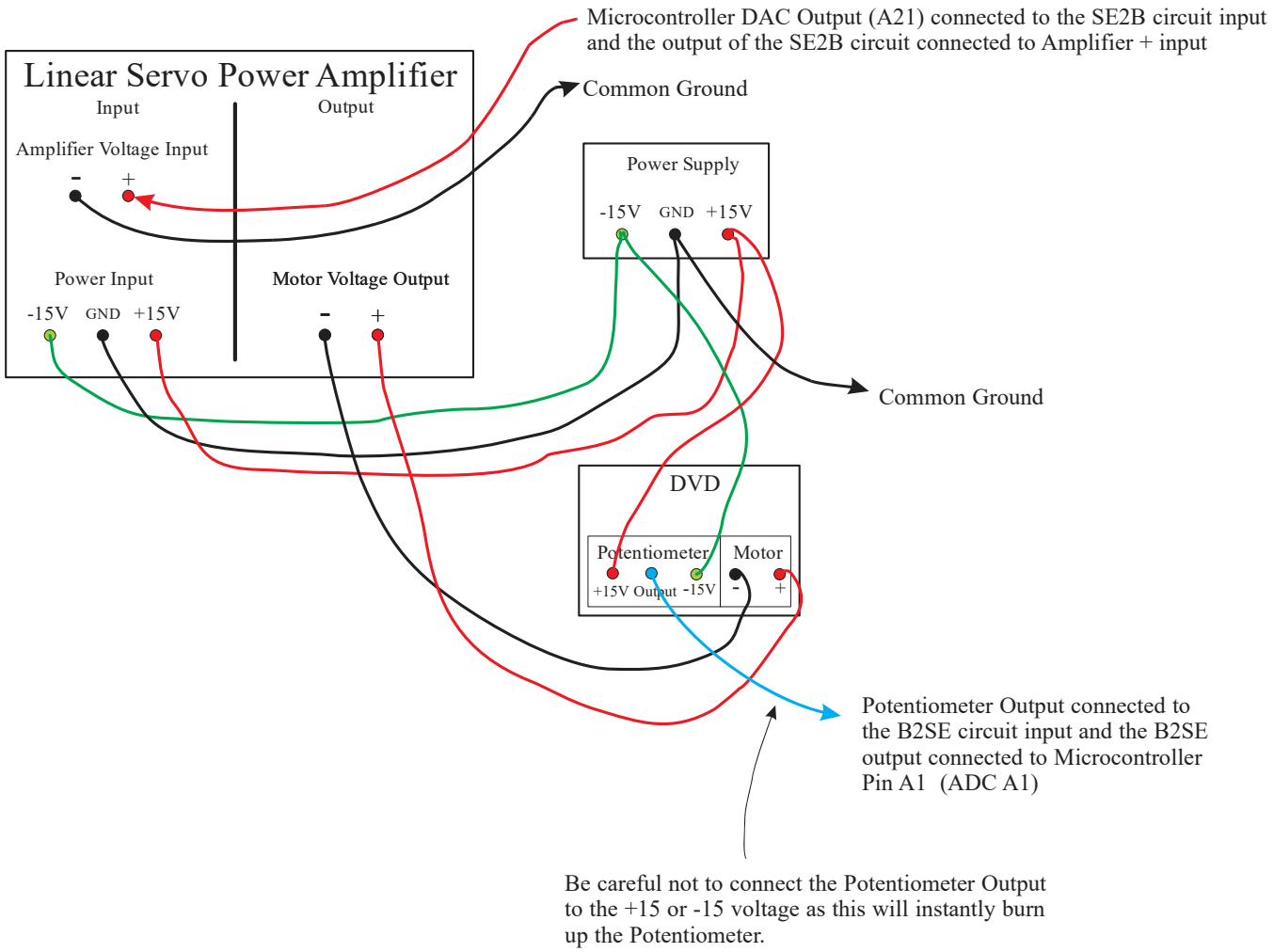
$$V(k+1) = \frac{-(T_s-2T)V(k)+(T_s+2aT)E(k+1)+(T_s-2aT)E(k)}{(T_s+2T)} \tag{3c}$$

Equation 3c gives the Controller output V at time (k+1). What is required however is the Controller output at time (k). This can be easily achieved by simply shifting each time k value by –1.

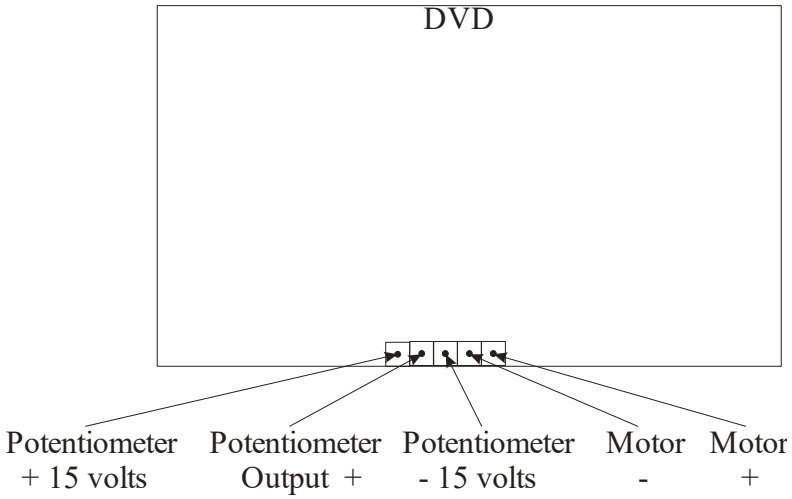
$$V(k) = \frac{-(T_s-2T)V(k-1)+(T_s+2aT)E(k)+(T_s-2aT)E(k-1)}{(T_s+2T)} \tag{3d}$$

Equation 3d gives the Controller output V(k) as a function of the control parameters (a and T), the sampling time T_s and V(k-1), E(k) and E(k-1).

Appendix 6: Wiring Hookups for DVD Setup

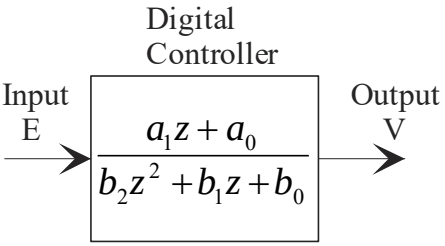


DVD Connections



Be careful not to connect the Potentiometer Output to the +15 or -15 voltage as this will instantly burn up the Potentiometer. The Potentiometer Output is the wiper of the potentiometer and cannot tolerate a high current.

Appendix 7: Conversion of a Digital Controller to a Difference Equation



$$\frac{V}{E} = \frac{a_1 z + a_0}{b_2 z^2 + b_1 z + b_0} \tag{1}$$

$$V\left(b_2 z^2 + b_1 z + b_0\right) = E\left(a_1 z + a_0\right) \tag{1a}$$

Note from the definition of the z operator $V(k)\,z = V(k+1)$ *and* $E(k)\,z = E(k+1)$

$$b_2 V_{k+2} + b_1 V_{k+1} + b_0 V_k = a_1 E_{k+1} + a_0 E_k \tag{1b}$$

Solving for V(k+2)

$$V_{k+2} = \frac{a_1 E_{k+1} + a_0 E_k - b_1 V_{k+1} - b_0 V_k}{b_2} \tag{1c}$$

Equation 3c gives the Controller output V at time (k+2). What is required however is the Controller output at time (k). This can be easily achieved by simply shifting each time k value by –2.

$$V_k = \frac{a_1 E_{k-1} + a_0 E_{k-2} - b_1 V_{k-1} - b_0 V_{k-2}}{b_2} \tag{1d}$$

Equation 1d gives the Controller output V(k) as a function of the controller parameters (a_0, a_1, b_0, b_1, b_2), and V(k-2), V(k-1), E(k-1) and E(k-2).

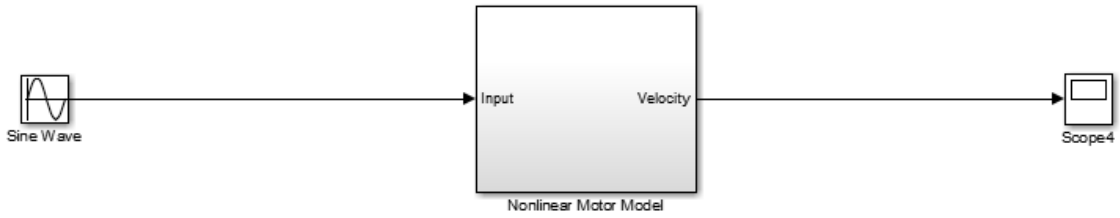
Appendix 8: Friction Models

Motor_Velocity_Modelw_friction_6m_Lab.m %Matlab Program

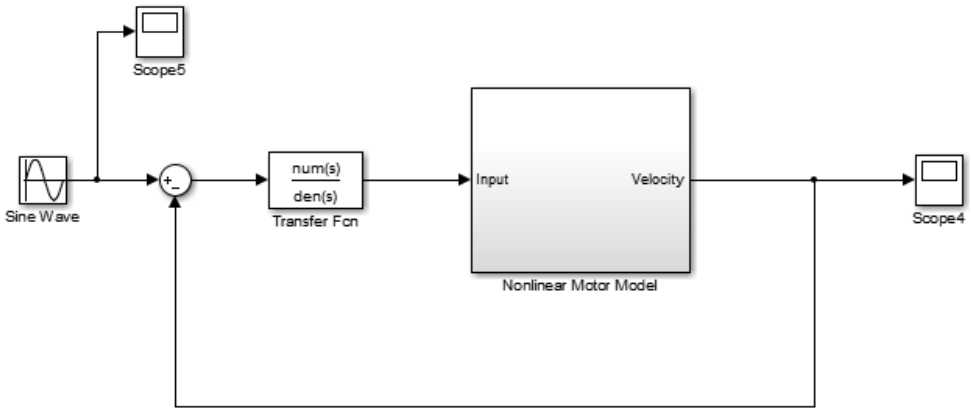
```
% Use this Program simulate the system with friction and then design a
% controller to improve the response

clear;
close all;

% Motor_Inertia = 4.2e-6;
% Fo = 1/Motor_Inertia;

Fo = 1;

% Motor Model parameters determined from the Bode analysis
Zeta         = ;
frequency_n  = ; % Hz
Gain         =
Omegan      = 2*pi*frequency_n; % rad/sec

Input_Amplitude = .1;

Kinetic_friction_Force = 1000;              % This is the normalized friction force i.e.
Friction_force/Inertia
                                            % Adjust this value until the theoretical
frictional response
                                            % matches the experimental response

Kinetic_friction_transition_rate = 1000;    % Do not change this value


qd = -.1:.0001:.1;
c4 = Kinetic_friction_Force;
c5 = Kinetic_friction_transition_rate;

friction_force_total = c4*tanh(c5*qd);

figure
plot(qd,friction_force_total)
xlabel('Velocity')
title('Friction Force (Coulomb)')


Ncycles = 5;
Npoints = 1000;

Period = 1;
tend = Ncycles*Period;
dt   = Period/Npoints;

% Calculate the open loop response
Input_Frequency  = 2*pi/Period; %rad/sec for sin input
sim('Motor_Velocity_Modelw_friction_6_open_loop',[0:dt:tend]);

figure
plot(ScopeData3(:,1),ScopeData3(:,2),ScopeData4(:,1),ScopeData4(:,2))
legend('Input','Velocity')
title('Open Loop Response')

% Closed Loop response
% Proportional Control Illustration

Control_gain = 1;

% The two lines below set up a proportional controller
% but any type of controller can be specified here
Control_num = [Control_gain];
Control_den = [1];
```

```
% Calculate the closed loop response
Input_Frequency  = 2*pi/Period; %rad/sec for sin input
sim('Motor_Velocity_Modelw_friction_6_closed_loop',[0:dt:tend]);

figure
plot(ScopeData5(:,1),ScopeData5(:,2),ScopeData4(:,1),ScopeData4(:,2))
legend('Input','Velocity')
title('Closed Loop Response')
```

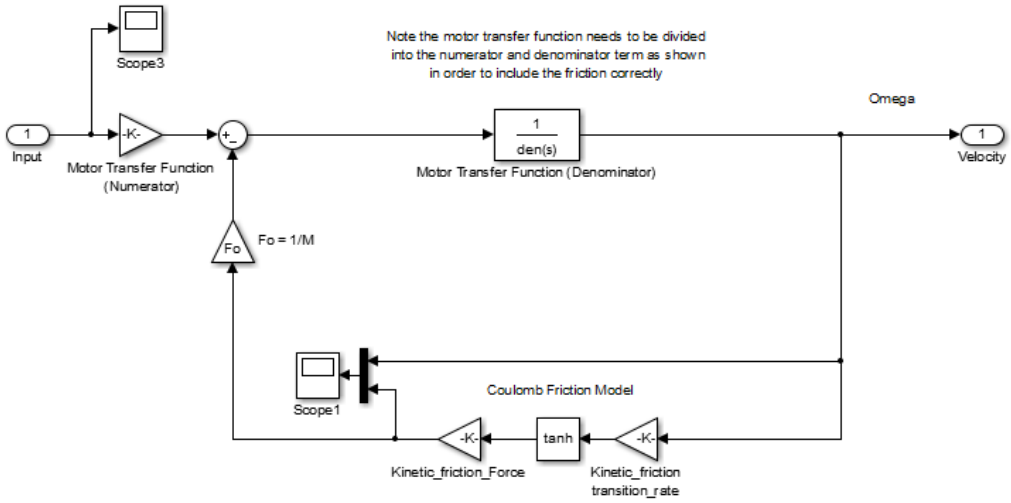
Motor_Velocity_Modelw_friction_6_open_loop.slx %Simulink Program



Motor_Velocity_Modelw_friction_6_closed_loop.slx %Simulink Program



Nonlinear Motor Model



Note the Nonlinear Motor model is in a different form that is necessary in order to have the velocity accesable for the columb friction model.