MAE 106 Laboratory Exercise #7
Computer Control of a Motor

University of California, Irvine
Department of Mechanical and Aerospace Engineering

REQUIRED PARTS

QTY	PARTS
	Switching current amplifier
	Breadboard
	Small Motor
	Labjack
	Wires

1 Introduction

The purpose of this laboratory exercise is to learn how to use a computer to control a motor. In the previous labs, you used operational amplifiers to control motors. Op amps are inexpensive and respond quickly, but require changing resistors and capacitors to change the control gains. Another common way to implement a control law is to use a computer. One of the major benefits of using a computer is that you can express the control law and control gains in software. Then, changing the control law just involves typing in new software code.

The basic idea is to:
1. electronically measure the system performance (e.g. the motor shaft's position with a pot)
2. read this measurement into the computer using an analog-to-digital (AD) converter
3. calculate an appropriate control law in a program running on the computer.
4. generate a voltage output to the motor with a digital-to-analog (DA) convertor

For this lab, you will use the LabJack and a PC for the analog-to-digital conversion (reading the motor shaft position in), and for the digital-to-analog conversion (sending a voltage command to the current amplifier/motor). The LabJack is a low-cost ($90) USB-based device that contains a chip in it that does A-to-D and D-to-A conversion. The LabJack has eight analog input channels (labeled AI0 through AI7), which can read in voltages between -10 and 10 volts, with 12 bit resolution (2^{12} = 4096 discrete levels). It also has two analog output channels (labeled AO0 and AO1), which can output voltages between 0 and 5 volts, with 10 bit resolution (2^{10} = 1024 voltage levels). It also has four digital input-output (labeled IO0 through iO3) channels that can either read in or put out "high" (5 volts) or "low" (0 volts) signals.

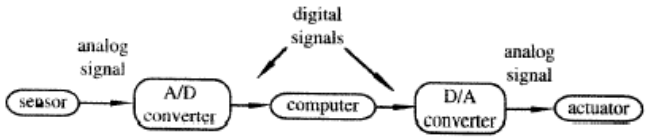


Figure 8.13 Computer control hardware.

Picture from: Ch. 8 Data Acquisition, Introduction to Mechatronics and Measurement Systems, 2nd Edition, David G. Alciatore and Michael B. Histand, McGraw-Hill 2003

2 Analog-to-Digital Conversion

In the first part of the lab, you will learn how to use the Labjack to read in a voltage waveform. Wire the function generator from the trainer kit to generate an approximately –2 to +2V, 2 Hz sinusoid into the AI0 input on the Labjack (use the osilloscope to check it – the amplitude and frequency don't need to be exact). To read in the voltage, you need to run the Matlab program readvoltage.m, which can be found in the directory C:\mae106\lab7. You are going to modify this program, so make a new directory with your name under C:\mae106 (i.e. create C:\mae106\yourname), and copy readvoltage.m into that directory. Here is a copy of the program:

```
% readvoltage.m
% MAE106 Laboratory Exercise #7
% created by Prof. David Reinkensmeyer, 2-21-2005

%LabJack Parameters
Idnum = 0;  % which Labjack you are using -- default = 0
demo = 0; % if > 0, puts software in "demo mode" -- can call functions without a Labjack
ADchannel = 0; % which analog-to-digital channel to read
channelgain = 0; % 0 = gain of 1

numsamps = 100; % how many samples to read
v = zeros(numsamps,1);  % initializes an array to hold the voltage samples
t0 = clock; % read the system clock
for i = 1:numsamps
   % read the voltage from the Labjack AD converter
    [voltage OverVoltage errorcode Idnum] = EAnalogIn(Idnum, demo, ADchannel, channelgain);
    v(i)= voltage; % save the voltage in the array
end;
tf = etime(clock,t0); % read the system clock again
samprate = numsamps/tf;  % calculate how long it took on average to read each sample
disp(['Samprate = ' num2str(samprate) ' Hz']);
```

Note that the "%" in Matlab denotes a comment. This program reads in 100 samples of the voltage on AI0 and stores the values in the array v. It also calls the PC's clock to calculate how many samples per second it read.

The program reads in data from the Labjack using the function `EanalogIn`. The documentation for the `EanalogIn` function is appended to the end of this lab so you can see what the its input and output parameters are. This function was written by the engineers at the company that makes the Labjack, and makes low level calls to the Labjack through the USB port.

To run the program, start Matlab by clicking the Matlab icon on the desktop. Change the Matlab working directory to be the directory that you just created. You can do this by entering the following command into the Matlab command prompt:

```
>> cd('c:\mae106\yourname')
```

Now, run the program by typing:

```
>> readvoltage
```

If the program is typed in correctly, it should run without an error, show you the sampling rate for the data you just took (i.e. how many samples per second the Labjack is returning). To see the data that you just collected, type:

```
>> plot(v,'.-')
```

This will show a graph of the sine wave that you read into the computer. The '.-' tells Matlab to put a dot at each sample and to connect the dots.

Q1 The labjack sampling rate is limited because it takes time to process data in the AD converter, the USB port, and the Matlab program. What sampling rate did the readvoltage.m program calculate? The sampling rate is an important parameter because it determines what the maximum frequency is that can be read accurately. For example, imagine that you are reading data from an accelerometer that is measuring ground movement due to an earthquake. If you read in the data too slowly (say at 1 Hz), you would miss some of the oscillations of the earthquake. With your Labjack setup, what is the fastest sine wave that you can read in from the function generator and still get a reasonable representation when you plot the voltage? How does this frequency compare to the Nyquist frequency?

3 Digital-to-Analog Conversion

The Labjack can also output a voltage between 0 and 5 volts on its DA channels (AO0 or AO1). You can make a call to the DA channel using the function `EanalogOut`. The documentation for this function is included at the end of the lab. Remove the function generator input to AI0, and wire the AO0 channel so that is the input to AI0. Now, you are going to use the computer to generate a sine wave voltage on DA0, and read it into AI0. Modify the "for" loop in your program so that it contains a call to the DA converter by inserting the following lines in your program. The required three NEW LINES are marked; you don't need to type in the comments.

```
t0 = clock; % read the system clock
f = 2; % NEW LINE 1 determines how many oscillations of the sinusoid to put out
for i = 1:numsamps
   analogOut0 = 1+sin(2*pi*f*i/numsamps); % NEW LINE 2 create a sine wave on AD0
   analogOut1 = 0; % NEW LINE 3 put 0 volts out on AD1
   [Errorcode Idnum] = EAnalogOut(Idnum, demo, analogOut0, analogOut1); % NEW LINE
   % read the voltage from the Labjack AD converter
    [voltage OverVoltage errorcode Idnum] = EAnalogIn(Idnum, demo, ADchannel, channelgain);
    v(i)= voltage; % save the voltage in the array
end;
```

Save the program, run it, and plot the voltage that you measured.

Q2: Why did the sampling rate change when you included a call to the DA convertor?

PRACTICAL EXAM 1: Demonstrate to the TA that you can read in a sinusoid with five oscillations (you will have to modify the code slightly to read exactly five oscillations)

4 Computer Control of the Motor

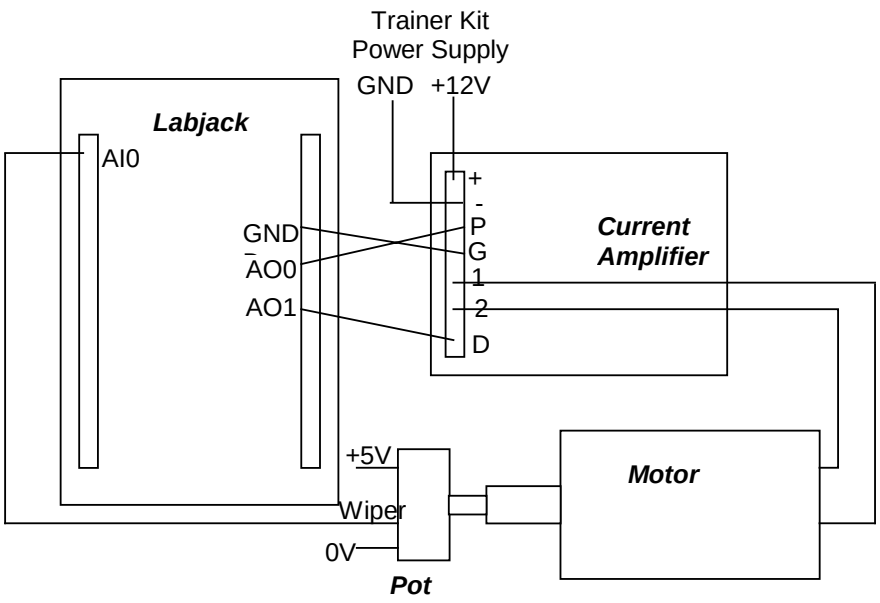
For the final part of this lab, you will use a Matlab program to implement a feedback controller for the motor. To provide power to the motor, you will use a low-cost current amplifier. This current amplifier is conceptually very similar to the one you used previously. You apply a low-power voltage (-5 to 5 volts) to the amplifier, and it provides a high-power current (0-3 amps, 12-55 Volts depending on the supply voltage) for the motor that is proportional to the voltage.

There are a couple of details about the current amplifier that are different, however. First, the amplifier is a switching type amplifier. This means that it creates the desired current by rapidly switching on and off at an appropriate rate. The switching rate for switching amplifiers is typically more than 10 KHz. The advantage of a switching type amplifier is that it consumes less power because the power transistors inside it don't heat up much when they are fully off (so I = 0 and Power = IV = 0), or when they are fully

on (so R = 0 and $P=I^2R=0$). The other common type of amplifier that you used previously, a linear amplifier, biases power transistors between fully "off" and fully "on" (0<R<inifinity), and thus the transistors heat up. The disadvantage of a switching amplifier is that the rapid switching adds electrical noise (and sometimes audible noise – try to listen to see if you can hear the amplifier switching frequency).

Second, the amplifier has the option to provide a unipolar input between 0 and 5 volts (instead of a bipolar one between –5 and 5 volts). If you do this, then you must provide a second input – the Direction input (which is either 0 or 5 Volts). The Direction input tells the motor which way to spin (i.e. it changes the direction of the current provided by the amplifier). This feature allows the amplifier to be controlled with two unipolar voltages, instead of one bipolar voltage. This is useful for this laboratory exercise because the Labjack DA output can only vary between 0 and 5 volts. We will use the second Labjack DA output to provide the Direction input to tell the motor which way to spin (you could also use a digital output from the Labjack).

Wire up your motor, amplifier, Labjack, and power supply as follows:



The current amplifier terminals are as follows:
+ = Positive power supply (connect to +12 V on trainer kit)
- = Negative power supply (connect to Ground on trainer kit)
P = Input into current amplifier (connect to Labjack AO0)
G = Ground (connect to any Labjack ground terminal)
1,2 = leads through which current is generated (connect to motor)
D = Direction input (connect to Labjack AO1)

The following Matlab program is called controlmotor.m and can be found in C:\mae106\lab7***. Copy it over to your directory,*** because you will need to modify it. This program implements a PD position feedback controller. It reads in the pot signal, calculates the control law, and then sends out the control signal to the motor amplifier. Notice that the derivative of the error is calculated by taking the difference between the current error and the previous error.

```
% controlmotor.m
% MAE106 Laboratory Exercise #7
% This program implements a PD controller using the Labjack
% Written by Prof. David Reinkensmeyer, 2/21/2005

%LabJack Parameters
Idnum = 0;  % which Labjack you are using -- default = 0
demo = 0; % if > 0, puts software in "demo mode" -- can call functions without a Labjack
ADchannel = 0; % which analog-to-digital channel to read
channelgain = 0; % 0 = gain of 1
analogOut0 = 1; %//        analogOut0 - Voltage from 0.0 to 5.0 for AO0
analogOut1 = 1; %//        analogOut1 - Voltage from 0.0 to 5.0 for AO1

% Control parameters
desiredangle0 = 2.5; % can range from 0 to 5 volts, for 0 to angular range of potentiometer
desiredfrequency = 0; % frequency of desired sinusoid to track
desiredamplitude = 1; % amplitude of sinusoid to track

pgain = .4; % proportional gain
dgain = .2; % derivative gain
numsamps = 60; % number of samples to read
samplingrate = 30; % nominal sampling rate in Hz.

% initialize data arrays and other variables to zero
cv = zeros(numsamps,1);
pv = zeros(numsamps,1);
dv = zeros(numsamps,1);
da = zeros(numsamps,1);
poserror= 0;
lastposerror = 0;
poserrordot = 0;
t0 = clock; % read the system clock

for i =1:numsamps
    % calculate the desired angle to move the motor
    desiredangle= desiredangle0+desiredamplitude*sin(2*pi*desiredfrequency*i/samplingrate);
    % read in the pot voltage
    [voltage OverVoltage errorcode Idnum] = EAnalogIn(Idnum, demo, ADchannel, channelgain);
    % update control variables
    lastposerror = poserror;
    poserror = voltage-desiredangle;
    poserrordot = poserror-lastposerror;
    % calculate PD control law
    controlvoltage = -pgain*poserror-dgain*poserrordot;
    % save data in arrays
    cv(i)=controlvoltage;
    pv(i)=voltage;
    dv(i)=dgain*poserrordot;
    da(i) = desiredangle;
    % Calculate direction input for current amplifier
    if controlvoltage > 0
        analogOut0 = controlvoltage;
        analogOut1 = 5;
    else
        analogOut0 = -controlvoltage;
        analogOut1 = 0;
    end
    % create the desired output voltages on the DA channels
    [Errorcode Idnum] = EAnalogOut(Idnum, demo, analogOut0, analogOut1);
end

% calculate sampling rate
tf = etime(clock,t0);
samprate = numsamps/tf;
disp(['Samprate = ' num2str(samprate) ' Hz']);

% zero motor ouput
analogOut0 = 0;
analogOut1 = 0;
[Errorcode Idnum] = EAnalogOut(Idnum, demo, analogOut0, analogOut1);

% plot results
```

```
figure(1); clf;
subplot(221);
plot(pv);
hold on;
plot(da,'r');
ylabel('Pot Voltage');
a = axis;
legend('actual','desired');
%line(a(1:2), [desiredangle desiredangle]);
ylim([0 5]);
xlim([0 numsamps]);
xlabel('Sample Number');

subplot(222);
ylabel('Control Voltage');
plot(cv)
hold on;
plot(dv,'r');
legend('total voltage','voltage due to d-term');
ylim([-5 5]);
xlim([0 numsamps]);
xlabel('Sample Number');
ylabel('Control voltages')
hold on;
shg; % show the graph
```

Type “controlmotor” at the Matlab command prompt (or press F5 from the Matlab editor window to run the program), and the motor should move so that the pot is in its middle position (corresponding to an output voltage of 2.5 volts). The program should then plot the response of the motor. If the motor moves all the way to one extreme, then you probably have implemented positive feedback. You can correct this by changing the sign of the gains in the program, or by switching the 1 and 2 leads from the current amplifier to the motor.

To measure the step response of the motor, manually turn the shaft away from the desired position, then start the program.

PRACTICAL EXAM 2: Demonstrate to the TA that your motor is under control.

Q3: Experiment with different proportional and derivative gains and see how they affect the step response of the controller. Can you make the system behave like an overdamped system? What happens when you decrease the P gain? What happens when you increase the P gain? What happens when you decrease the D gain? What happens when you increase the D gain?

Modify the code so that the desired pot angle is a 2 Volt peak-to-peak sine wave at 1.0 Hz. (Hint: you can just change the `desiredfrequency` parameter in the program).

Q4: Tune the proportional and derivative gains of the controller to get the best tracking you can for the 1.0 Hz sine wave. Record data showing how well your controller performs. You can use the Matlab command “save” to save your data to a file (type help save in the Matlab prompt for documentation).

WRITE-UP

- due at your next laboratory session
- each student must complete his or her own write-up
- make sure to use your own words and to type the write-up!!
- include your name and laboratory time on the write-up

1. Plot the best tracking performance of the 1 Hz sinusoid that you achieved. The plot should show the desired angle of the motor, and the actual angle for several cycles of the sinusoid. State the proportional and derivative gains that achieved this performance.
2. If you increased your gains too much, the motor went unstable. This result is not predicted by the continuous time model of the system (i.e. increasing Kp and Kd should not make the second order system go unstable). Provide an explanation. Hint: Model the effect of the Labjack as adding an e^{-sT} in the forward loop of the controller (i.e. a pure delay). Approximate the e^{-St} as $1-e^{-St}$ (1st order Taylor's expansion), and calculate the closed loop transfer function. Show that the system goes unstable if the gains are too large.

Documentation for EAnalogIn and EAnalogOut

```
%/*****************************************************************************
**************/
%// EAnalogIn
%//
%// Easy function.  This is a simplified version of AISample. Reads the voltage from
1 analog input.
%// Calling this function turns/leaves the status LED on.  Execution time for this
function is 50 ms
%// or less.
%//
%// MATLAB Syntax:
%//         [voltage OverVoltage errorcode Idnum] = EAnalogIn(Idnum, demo, channel,
gain)
%//
% // Inputs:
%//         idnum - Local Id, serial number, or -1 for first LJ found
%//         demo - 0 for normal operation, >0 for demo mode.  Demo mode allows this
function to be called
%//                  without a LabJack.
%//         channel  -  Channel  command  is  0-7  for  single-ended,  or  8-11  for
differential
%//         gain - Gain command is 0=1, 1=2, ..., 7=20.
%//
%// Outputs:
%//         voltage - Returns the voltage reading.
%//         OverVoltage - If >0 over voltage has been detected on one of the selected
analog inputs
%//         errorcode - LabJack error codes or 0 for no error.
%//         idnum - Local ID number of Labjack, or -1 if no LabJack is found.


/*****************************************************************************
*************/
// EAnalogOut
//
// Easy function.  This is a simplified version of AOUpdate. Sets the voltage of both
analog outputs.
// Execution time for this function is 50 milliseconds or less.
//
// MATLAB Syntax:
//          [Errorcode Idnum] = EAnlaogOut(Idnum, demo, analogOut0, analogOut1)
//
// Inputs:
//          Idnum - Local Id, serial number, or -1 for first LJ found
//          demo - 0 for normal operation, >0 for demo mode.  Demo mode allows this
function to be
//          called without a LabJack.
//          analogOut0 - Voltage from 0.0 to 5.0 for AO0
//          analogOut1 - Voltage from 0.0 to 5.0 for AO1
//
// Outputs:
//          idnum - Local ID number of Labjack, or -1 if no LabJack is found.
//          errorcode - LabJack error codes or 0 for no error.
```