DAQ 2: Introduction to Data Acquisition: Analog I/O

2.1 Introduction

Now that you have seen how to use the DAQ cards to set and measure digital (on and off) voltages, we will look at the card's ability to set and measure a greater variety of voltages. Though this will be called "analog input" and "analog output," it is important to note that the computer still cannot read a true continuum of values. These cards, in particular, are capable of splitting up a range of voltages into 2^{16} (or 65536) different "levels," and the card can read which of those levels is nearest to the actual voltage at the terminals. The input range can be as small as ± 0.05 V or as large as ± 10 V, meaning the resolution on the input can be as small as about 1.5 μV or as "large" as about 300 μV. These cards can also set any of 65536 different voltages within a range of ± 10 V - giving the device a resolution of about 300 μV. For our experiments, this resolution will be more than adequate; however, some applications require greater refinement, and DAQ cards are made that are capable of higher resolutions as well as different ranges.

During this lab, you will learn how to use MATLAB and the DAQ cards to set and measure voltages, including using a loop to produce or read several different voltages over time. Please note that as always when working with the DAQ cards, your safety is paramount, followed by the safety of the equipment. The safety notice for all DAQ labs still applies. Please revisit the rules that come into play when doing one of the DAQ labs in the handout for DAQ 1.

After becoming familiar with analog voltages, you will create a network listing that will allow you to collect data to determine how an LED's color, size, and/or shape may determine how the LED works. In particular, you will be graphically determining when each of the LEDs turns on and comparing those values.

2.2 Resources

The additional resources required for this assignment include:

- Books: None
- Pratt Pundit Pages: `MATLAB:CB-68LP Pinout`,
  `http://pundit.pratt.duke.edu/wiki/EGR 53/DAQ 2`
- Lab Manual Appendices: None

Note - the second Pundit page listed contains pictures of the circuits under development as well as a couple graphs MATLAB should show you during the lab. You may therefore want to have Pundit open on the PC *not* being used for data acquisition.

2.3 Getting Started

1. Each lab group should have one person sit at the PC that has the patch of tape (blue, green, or red) on it. This is **not** the same as the strip with the computer's name on it - it will literally be a small patch of tape all by itself.

2. Set the machine to "Log on to" `...(this computer)` and use the "User name" `mrglocal` and the "Password" `p1p2dell` (that is p-ONE-p-TWO-d-e-l-l). The other lab partner can log into the other computer either using the `mrglocal` account or any other valid NET ID (as log as the "Log on to" is set to `Kerberos`). You do *not* need to run X-Win or PuTTY at all on either computer.

3. Start MATLAB on the PC with the tape on it. The first time MATLAB is run, it may take some time to start up.

4. Once MATLAB starts, make sure the `Current Directory` listed at the top of the MATLAB window is

   ```
   C:\Documents and Settings\labuser\My Documents\MATLAB
   ```

5. In the `Current Directory` window on the left, select and delete any files in the `MATLAB` directory.

2.4 Equipment

The DAQ lab this week will re-use several of the devices and elements used previously. Specifically, the DAQ board, breadboard, LEDs, resistors, and wires will again be used. Be sure that you have the following inventory at your lab station:

- *DAQ (Data Acquisition) Board:* Again, be sure to use the table of pinouts for the CB-68LP connection block, which are shown on the Pratt Pundit page `MATLAB:CB-68LP Pinout`, if you are unsure of where to make a connection.

- *Breadboard*

- *LEDs:* You should have six LEDs - both red and green examples of small, large, and rectangular LEDs.

- *Resistors:* You will need either two 470 Ω resistors or two 680 Ω resistors. There will most likely be a total of four of these in the plastic box you took from the front of the room.

- *Wires:* This week, you will need a total of six red wires, six black wires, and four green wires.

2.5 Basic Analog Output

The DAQ cards used in this lab have two analog output channels, labeled DAC0OUT and DAC1OUT on `MATLAB:CB-68LP Pinout` page. MATLAB can control the output voltage of these channels independently, and each is allowed to have a range from -10 V to +10 V. The voltage is measured relative to the AOGND (Analog Output GrouND) channel.

There are two fundamentally different ways to use the analog output channels. The first, which will be covered in a later lab, allows you to give MATLAB a vector of data and timing information for how fast the values in the vector should be applied to the output channel - with this method you can control the exact rate at which MATLAB will deliver voltages. This is useful if, for example, you are trying to control a device or make a sound by connecting a speaker to the output of your card.

The second, which is used in this lab, is to tell MATLAB a voltage value for each channel you have added. This method is far simpler and can still be used to produce a stream of voltages by using the command in a loop. The drawback for this method is there is no way to set *exactly* the speed at which the voltages will be delivered. This asynchronous method of delivering voltages could be disastrous for applications where timing is important, such as music.

To see how analog output works, you will wire the two analog output channels such that each powers a series combination of a resistor and an LED as in the first data acquisition lab. For this part of the lab, you will need the following in addition to the CB-68LP and the breadboard:

- Two LEDs (small green and red)
- Two resistors
- Two red wires and one black wire

Once you have the items above, make the connections in Table 2.1 on page DAQ 2 – 14 to make the circuit shown in Figure 2.1. Remember that anything called a "Pin" or referred to as the letter A-J, X, or Y followed by a number indicates a socket on the breadboard – the letter is the row and the number is the column. Anything called a "Line" or referred to as the letter L followed by a number indicates a gate on the CB-68LP connection block.

Once you have made sure the circuit is connected correctly, you can begin writing the program to control the two lights. For this part, the green LED and its resistor are connected to `DAC0OUT` and will be on the left while the red LED and its resistor are connected to `DAC1OUT` on the right. We will now go through the code required to access the data acquisition card, how to set up the two output channels, and how to set the voltages of each channel.

Open a new script file that you will save as `BasicAOutput.m`. First, you will need to write code that sets up the card and prepares it to act as an output device. The following code will clear the workspace, stop and clear any connections MATLAB currently has to the DAQ card, and create a variable called `AO` (the capital letters ay and oh) that we can use to talk with the card:

```
% Clear out workspace
clear

% Clear out DAQ objects
delete(daqfind)

% Create Analog Output Object
AO = analogoutput('nidaq', 1)
```

If you save and run this code, you will see the following message:

```
Display Summary of Analog Output (AO) Object Using 'PCI-6014'.

       Output Parameters:  1000 samples per second on each channel.

      Trigger Parameters:  1 'Immediate' trigger on START.

           Engine status:  Waiting for START.
                           0 total sec. of data currently queued for START.
                           0 samples currently queued by PUTDATA.
                           0 samples sent to output device since START.

AO object contains no channels.
```

This means MATLAB has connected to the analog output portion of the DAQ card and is awaiting further instructions. Your next task is to tell MATLAB which of the two analog output channels you would like to use. For this program, you will be using both (channels 0 and 1), and you can tell MATLAB that by adding the following code:

```
% Add channels to Analog Output Object
addchannel(AO, [0 1])
```

Now if you run the code, you will get an additional message,

```
  Index:  ChannelName:  HwChannel:  OutputRange:  UnitsRange:  Units:
   1        ''            0           [-10 10]      [-10 10]     'Volts'
   2        ''            1           [-10 10]      [-10 10]     'Volts'
```

that shows you now have access to hardware channels 0 and 1, that the DAQ has configured these outputs to range from -10 V to 10 V, and that DAC0OUT is the first channel and DAC1OUT is the second (based on their index). Once a channel is added, MATLAB always refers to it by its index number, not its hardware channel number. In this case, the "first channel" is DAC0OUT and the "second channel" is DAC1OUT.

Now that you have access to the particular channels, you can send voltages to each using the `putsample` command. For example, to test your circuit by turning both of the lights on, then pausing until the user hits a key, then turning both the lights off, you can add the code (note - the numbering of lines of code is discontinuous because lines have been skipped which will be added in later):

```
19 % Write values to output channels
20 putsample(AO, [5 5])
21 pause
22 putsample(AO, [0 0])
```

and run your program. Make sure that both of your lights come on properly. If not, the most common mistake is to have put the LED in backwards. Fix any wiring mistakes you may have, and run the code again. If you cannot find a problem with your wiring, let a TA or the instructor know.

It is important to note that the data in the first entry of the vector is assigned to the first indexed channel. For example, putting in the vector `[5 0]` will turn the green LED on since it is connected to the `DAC0OUT` hardware channel, which is indexed first. Conversely, putting in `[0 5]` will turn the red LED on since it is connected to the `DAC1OUT` hardware channel, which is indexed second. You can also set the voltages to two different non-zero values - try `[3 5]` or various other combinations of non-negative voltages and see what happens to your lights. When you are done with that, return the vector on line 20 of your code to `[5 5]` as a way of testing the lights, and run your code again.

You can also use the `putsample` method to set the lights to a stream of values by putting the command in some kind of loop. Add the following code to the bottom of your code, then run your program:

```
24 % Use loop to set several different voltages
25 for k=1:300
26     % Calculate voltages for each channel
27     Vout0 = 2.5+2.5*sin(2*pi*k/100);
28     Vout1 = 2.5+2.5*cos(2*pi*k/100);
29     % Put voltages to each channel
30     putsample(AO, [Vout0 Vout1])
31     % leave line 31 blank for now
32     % leave line 32 blank for now
33     pause(0.02)
34 end
------------------------------------------------
43 % Turn all outputs off
44 putsample(AO, [0 0])
```

Notice that the lights will become brighter, then go dim, then will be off for a bit, then will repeat that process a couple of times. This is because the LEDs will remain off for positive voltages below some threshold. Above that threshold, they will become brighter as more and more current passes through them, up to some physical tolerance at which point the LED will break. The resistor is in place in this circuit to make sure the current level never reaches that point. Also, most LEDs can handle *some* negative voltage but generally, that should be avoided.

If you want to change speeds, you can do so by changing the argument of the `pause` command or the frequency in the `sin` function. If you need more exact timing for the output, however, you will need to use a different set of commands for setting output voltages - these commands will be discussed in a future DAQ lab. For example, a pause of 0.02 sec does *not* mean there are 50 samples being sent to the card per second, it only means that there is a 20 msec delay before MATLAB continues to process the code.

2.6 Basic Analog Input

Analog input works in much the same way as analog output - you need to access the card, set up the channels you want to measure, then use `getsample` to have the card take measurements from the analog input lines you added. For the NI 6014e card, there are 16 analog input channels (ACH0 through ACH15). With these, there are two ways of measuring voltages - *single-ended* and *differential.* Single-ended means you read a single voltage and compare it to a common ground, so you can take up to 16 measurements at a time. The specific line to connect to the common ground is `AIGND`. One limitation of this method is the fact that you cannot measure across just any element in the circuit - you can only measure a voltage relative to the ground voltage. Imagine someone standing on a step-stool, and assume you want to know their height. If one end of the tape measure must stay on the floor, you will need to take two measurements - one from the ground to the top of the person's head, and one from the group to the top of the stool. To get the height, you will need to take the difference of these two measurements. The same is true for single-ended measurements.

Differential measurements, on the other hand, take two voltage measurements at once, and the voltage measurement given is the difference between the two. Note, however, that what is really happening is that both leads are being measured against that same common ground - you will still need to have `AIGND` connected to take voltage measurements. Electronically, then, the same measurements are happening, only the program will automatically calculate the voltage difference for you. One drawback of this is every voltage difference you want to measure requires two leads - even if two of the measurements happen to share a common location, you must have a complete pair of wires for each.

The way the card is set up in MATLAB, there are eight available differential voltage channels - 0 through 7. The voltage reported for a particular channel, however, is actually the difference between the voltage measured on the hardware channel of the same number and the voltage measured on the hardware channel eight higher. For example if you are using channel 4 in differential mode, you are really asking for the difference in voltage between `ACH4` and `ACH12`. Differential channel 6 is the difference between `ACH6` and `ACH14`. This numbering system explains in part the particular placement of the hardware channels on the CB-68LP board - the differential pairs are generally positioned with one directly under the other in a column. The only differential input pair that is split across columns is the `ACH4-ACH12` pair.

To begin analyzing analog input, save your `BasicAOutput.m` script from above, then use Save As... to save it as `BasicAIO.m`. You will be using much of the same code as before. First you will need to add some input channels. Just beneath the code that added the output channels, type in:

```
% Create Analog Input Object
AI = analoginput('nidaq', 1)

% Add channels to Analog Input Object
addchannel(AI, [0 4])
```

and when you run the code, MATLAB will print

```
Display Summary of Analog Input (AI) Object Using 'PCI-6014'.

  Acquisition Parameters:  1000 samples per second on each channel.
                           1000 samples per trigger on each channel.
                           1 sec. of data to be logged upon START.
                           Log data to 'Memory' on trigger.

      Trigger Parameters:  1 'Immediate' trigger(s) on START.

           Engine status:  Waiting for START.
                           0 samples acquired since starting.
                           0 samples available for GETDATA.

AI object contains no channels.
```

to the screen in response to creating `AI` and will print

```
Index:  ChannelName:  HwChannel:  InputRange:  SensorRange:  UnitsRange:  Units:
1       ''            0           [-5 5]       [-5 5]        [-5 5]       'Volts'
2       ''            4           [-5 5]       [-5 5]        [-5 5]       'Volts'
```

in response to adding the two channels. Note that only the *first* hardware channel of the differential pair is shown. Still, the first indexed channel `HwChannel 0` will monitor the difference in voltages between `ACH0` and `ACH8` and the second indexed channel will monitor `ACH4` and `ACH12`. As an aside, it should be clear that you cannot add hardware channels 8 or above when using differential inputs - those are already in use as the second part of a differential pair!

There are several defaults to notice for the analog input object and channels. First of all, the input object is set up to receive 1000 samples per second on each channel and to take 1 sec. of data once triggered. The cards are actually capable of taking 200000 samples per second, distributed across any open channels. Second, the card is reporting that no data has been acquired and no data is available for the `getdata` command. The latter part will only be relevant when we start using the timed (synchronous) mode for the cards.

When the channels are added, note that the default case is to measure between -5 V and +5 V. For this card, the values can be set to ranges of ±50 mV, ±500 mV, ±5 V, and ±10 V, depending on which range will give you the best precision while simultaneously being large enough to capture all your voltages. For this lab, all voltages will be within the range of ±5 V.

Notice that the index values are 1 and 2 again since you are opening up channels on a new *input* object. Unlike digital I/O, where all the lines were on a single object and you had to specify which were inputs and which were outputs, the input and output objects for *analog* voltages are separate. For this circuit, channel 0 - which is the voltage difference between ACH0 and ACH8 - will be measuring the voltage across the left-hand circuit (the sum of the voltage drops across the resistor and the green LED) while channel 4 - which is the voltage difference between ACH4 and ACH12 - will be measuring the voltage across the right-hand circuit.

The network listing, including the new wires you need to connect, is in Table 2.2 on page DAQ 2 – 15. Go ahead and connect the wires as specified in the table, making sure you have connected them to the proper lines of the CB-68LP. The code you need to add to take measurements goes in the `for` loop, just under the `putsample` command, on line 32:

```
24 % Use loop to set several different voltages
25 for k=1:300
26     % Calculate voltages for each channel
27     Vout0 = 2.5+2.5*sin(2*pi*k/100);
28     Vout1 = 2.5+2.5*cos(2*pi*k/100);
29     % Put voltages to each output channel
30     putsample(AO, [Vout0 Vout1])
31     % Read voltages from all input channels
32     Voltages(k,:) = getsample(AI);
33     pause(0.02)
34 end
```

The reason for the `Voltages(k,:)` is that the `getsample` command will have, as an output, a row vector containing the measured voltages for each of the differential channels you are looking at - in this case, a 1x2 vector each time `getsample` is called since there are two open channels. At the end of the run, the `Voltages` variable will thus be a 300x2 matrix of the voltages measured across the combination of the resistor and the LED. If you add the commands

```
36 % Plot voltages versus index
37 n = 1:k;
38 Vleft  = Voltages(:,1);
39 Vright = Voltages(:,2);
40 plot(n, Vleft, 'g', n, Vright, 'r')
41 legend('Green LED', 'Red LED', 0)
```

after the `end` of the `for` loop, you will see the voltages measured as a function of the index of the matrix. Again, the timing is not precise here so despite the channels measuring the sinusoid as generated by the output channels, the graph may not look exactly like a sinusoid. Generally, however, the loop will take about the same amount of time to complete each run, so the graph should be close. A sample is provided on the supplemental Pratt Pundit page for this lab.

Go ahead and experiment with some different functions for the outputs. For example, you might try the following for lines 27 and 28 in your code:

- Two sinusoids at the same frequency and phase:

```
Vout0 = 2.5+2.5*sin(2*pi*k/100);
Vout1 = 2.5+2.5*sin(2*pi*k/100);
```

- Two sinusoids at different frequencies:

```
Vout0 = 2.5+2.5*sin(2*pi*k/100);
Vout1 = 2.5+2.5*sin(4*pi*k/100);
```

- Two lines - one with a positive slope and one with a negative slope:

```
Vout0 = 5 * (k/300);
Vout1 = 5 * (1 - (k/300));
```

- Two sinusoids at increasing frequencies:

```
Vout0 = (2.5+2.5*sin(2*pi*k.^1.8/1200));
Vout1 = Vout0;
```

- Two random signals:

```
Vout0 = 5*rand(1);
Vout1 = 5*rand(1);
```

If you put in your own functions, just make sure the values assigned to the voltages remain greater than -1 V and less than 5 V. The -1 V limit is set to make sure you do not break an LED by applying too much reverse voltage; the 5 V limit comes into play because the DAQ card is not set to measure anything higher - voltages greater than 5 V will be read as 5 V - a process called *clipping*.

2.7 Assignment

The assignment for this lab includes two major parts. First, you will add measurement lines to get more information about the voltages in your circuit. Specifically, in addition to the total voltage across the resistors and LEDs together, you will measure the voltage across each resistor and the voltage across each LED.

Next, you will use this circuit to obtain data about the six LEDs - small green, small red, large green, large red, rectangular green, and rectangular red. You and your partner(s) will then graph the data in a plot broken up into six subplots, where each will contain three lines - the total voltage across a resistor-LED combination, the voltage across just the resistor, and the voltage across just the LED. From these graphs, you will be able to determine when each LED turns on and whether LED shape or color matter. You will then submit that information as well as the answers to some questions on a handout provided in class. Finally, you will copy all your files and graphs to your OIT account.

It is important to understand how LEDs work, and to do that better, you will be examining the electrical response of the LED to an external voltage as it increases from -1 V to 5 V. For this part of the assignment, you will actually be taking *six* voltage measurements simultaneously. You will be measuring the total voltages just as you are now, but you will be adding a measurement for the voltage across each resistor and a measurement across each LED.

2.7.1 Accessing Voltage Information on Multiple Channels

The critical part about adding more measurements is understanding the differences among what is called a *channel*, what the number attached to an `ACH` is, and how MATLAB access the data. In differential mode, the channel - or `HwChannel` - refers to which pair of voltages are going to be measured. Channel 3, for example, refers to *difference* between voltages measured on `ACH3` and `ACH 11`. The individual `ACH` wires themselves will be measuring voltages relative to ground. In differential mode, then, each *channel* is comprised of two `ACH` lines. Those lines are *always* 8 apart for this particular DAQ card.

The *index* is how MATLAB refers to that channel. A channel's index is determined by the order in which channels were added to an object. For example, if the line:

```
addchannel(AI, [3 1 4 2 6 7])
```

were to show up in your code, you would be telling MATLAB to add channels 3, 1, 4, 2, 6, and 7 - in that order - to the object called `AI`. The voltage measured as channel 3 - which is the voltage difference between `ACH3` and `ACH11` - would be considered the first voltage since channel 3 was added first. It would thus have an index of 1. Channel 6 - the difference between `ACH6` and `ACH14` - would have an index of 5 since it was opened fifth.

When using the `getsample` command, the voltages returned will be in the same order as the channels were added. For example, if the `addchannel` command above were followed with

```
MyVals = getsample(AI)
```

`MyVals` matrix would be a 1x6 matrix with voltages measured on channels 3, 1, 4, 2, 6, 7, respectively. If you were to want to pull out a specific voltage - say the voltage on channel 2, you would *first* have to note that channel 2 was the fourth channel opened. Then you could write:

```
Channel2Voltage = MyVals(4)
```

For the assignment at hand, you need to take six different measurements. In this case, it may be easiest to use channels 0, 1, and 2 for the left (green) side and channels 4, 5, and 6 for the right (red) side. Note that channels 3 and 7 are not used - MATLAB is fine with skipping channels.

2.7.2 Altering Your Script and Circuit

First, save the `BasicAIO.m` script as

```
AIO_NETID1_NETID2.m
```

where NETID1 and NETID2 are the NET IDs of the people in the group. Append the file name as necessary if there are more people in the group. For example, `mrg`, `wns`, and `rap3` would create a script named:

```
AIO_mrg_wns_rap3.m
```

Add comments to the top as follows:

```
% Group member 1: NAME (ID)
% Group member 2: NAME (ID)
% Group member 3: NAME (ID) (if needed)
% We have all followed the Community Standard (ID1, ID2, ID3, etc)
```

so again, in the above case, the file used by `mrg`, `wns`, and `rap3` would begin:

```
% Group member 1: Michael Gustafson (mrg)
% Group member 2: Rebecca Simmons   (rap3)
% Group member 3: W. Neal Simmons   (wns)
% We have all followed the Community Standard (mrg, rap3, wns)
```

Next, change the line that added channels to the input object to:

```
% Add channels to Analog Input Object
addchannel(AI, [0 1 2 4 5 6])
```

and add the wires as specified in Table 2.3 on page DAQ 2 – 16. This will set up the card and input object as follows:

Index	Channel	ACH	Voltage
1	0	ACH0 and ACH8	Total, left (green)
2	1	ACH1 and ACH9	Resistor, left (green)
3	2	ACH2 and ACH10	LED, left (green)
4	4	ACH4 and ACH12	Total, right (red)
5	5	ACH5 and ACH13	Resistor, right (red)
6	6	ACH6 and ACH14	LED, right (red)

Now, change the functions in the `for` loop such that the output voltages will go from -1 V to 5 V for both output channels as `k` goes from 1 to 300. That is, change the `Vout0` and `Vout1` lines to:

```
Vout0 = -1+6*(k-1)/299;
Vout1 = Vout0;
```

Next, you will want to quickly produce a figure by plotting all six voltages at once. This is so you can verify that the six measurements were properly taken. You will post-process the data in another file to actually create your final plot. Replacing the lines

```
% Plot voltages versus index
n = 1:k;
Vleft = Voltages(:,1);
Vright = Voltages(:,2);
plot(n, Vleft, 'g', n, Vright, 'r')
legend('Green LED', 'Red LED', 0)
```

with a simple

```
% Plot Voltages matrix
plot(Voltages)
```

will ask MATLAB to plot the 300x6 matrix obtained from the looped `getsample` command. Recall that plotting a matrix in MATLAB means plotting each column in its own line using MATLAB's color wheel. The default colors are, in order, blue, green, red, cyan, magenta, yellow, black, after which MATLAB repeats the colors. In this case, the blue and cyan lines are the total voltages, the green and magenta lines are the resistor voltages, and the red and yellow voltages are the LED voltages. Some of the lines may be "hidden" behind other lines if the measurements are nearly the same.

At this point, you should make sure that your six lines make sense. The two lines representing total voltage should be straight, starting at -1 V when `k=1` and growing to 5 V when `k=300`. The two lines representing the LED voltages will track along that same path, then at some point take a fairly hard right, as if the voltage across the LEDs simply could not keep up any more. The "break point" may be different for the two LEDs, but not vastly so. Finally, the voltage across the resistors will start at 0 V and remain there until the same time as the LEDs make their break - at that point, the resistor voltages will begin to rise, making up for the voltage difference between the total voltage and the now-lagging LED voltage. An example of this graph is presented at the Pratt Pundit page for this lab.

Note that your lines may have very small amounts of noise (jaggies) in them, but should not have wild oscillations. Also, make sure none of your voltages goes well below -1 V or above 5 V. If any of that happens, you have likely miswired something. Check your wiring again and run the program. When you believe you have successfully measured the six voltages for the small LEDs, have a TA check your graph.

2.7.3 LED Curves - Small LEDs

Once you are sure you have good data, type the following in the command window:

```
SmallVoltages = Voltages
save SmallData SmallVoltages
```

The file `SmallData.mat` now contains the values you just measured. This will be useful for your plots later.

2.7.4 LED Curves - Large LEDs

Once you obtain, verify, and save data for the small LEDs, you will want to do the same for the large LEDs. Replace the small LEDs with the large version of the same color and re-run your program. Your graphs should be quite similar to the graphs for the small LEDs, though it is possible that the break points will be different. Make sure your graph makes sense, check with a TA if you desire, then type the following in the command window:

```
LargeVoltages = Voltages
save LargeData LargeVoltages
```

2.7.5 LED Curves - Rectangular LEDs

Once you obtain, verify, and save data for the large LEDs, you will want to do the same for the rectangular LEDs. Once you get a good graph, type the following in the command window:

```
RectVoltages = Voltages
save RectData RectVoltages
```

2.7.6 LED Driver Circuit Voltage Plots

Having saved the three data files containing information about all six LEDs, you will write a script called

```
VoltagePlotter_NETID1_NETID2.m
```

where NETID1 and NETID2 are the NET IDs of the people in the group. Append the file name as necessary if there are more people in the group. For example, `mrg`, `wns`, and `rap3` would create a script named:

```
VoltagePlotter_mrg_wns_rap3.m
```

Again, add comments to the top as follows:

```
% Group member 1: NAME (ID)
% Group member 2: NAME (ID)
% Group member 3: NAME (ID) (if needed)
% We have all followed the Community Standard (ID1, ID2, ID3, etc)
```

so, as in the above case, the file used by `mrg`, `wns`, and `rap3` would begin:

```
% Group member 1: Michael Gustafson (mrg)
% Group member 2: Rebecca Simmons   (rap3)
% Group member 3: W. Neal Simmons   (wns)
% We have all followed the Community Standard (mrg, rap3, wns)
```

You will use this script to produce the graphical information you need to determine how color and shape may impact when the LED turns on. You script should first clear the workspace and any graphs that already exist. You will then load the three data files. To make reading easier, you should put each voltage measurement into its own matrix. For example, you might use the following to split the data obtained when using the small LEDs:

```
load SmallData
SmallGreenTotal = SmallVoltages(:,1);
SmallGreenRes   = SmallVoltages(:,2);
SmallGreenLED   = SmallVoltages(:,3);
SmallRedTotal   = SmallVoltages(:,4);
SmallRedRes     = SmallVoltages(:,5);
SmallRedLED     = SmallVoltages(:,6);
```

Copy this code to make whatever changes you need for the large and rectangular LED data. Next, for each LED produce a plot of the total voltage, the resistor voltage, and the LED voltage as a function of the *total voltage*. While just having the numbers 1 through 300 as the x axis was fine above for making sure the data was saved properly, such an axis provides no easily-obtained useful information. In the subplots, put the green LEDs on the left and the red LEDs on the right; put the small LEDs on the top, the large LEDs in the middle, and the rectangular LEDs on the bottom. That is:

```
subplot(3,2,1)
% small green plotting commands
subplot(3,2,2)
% small red plotting commands
subplot(3,2,3)
% large green plotting commands
subplot(3,2,4)
% large red plotting commands
subplot(3,2,5)
% rect green plotting commands
subplot(3,2,6)
% rect red plotting commands
```

Turn the grid on for each subplot (`grid on`) and be sure to include a title for each of the subplots indicating which LED is being shown. Do *not* include a legend for each subplot; just be consistent about having the total voltage as a solid line, the LED voltage as a dashed line, and the resistor voltage as a dotted line. Save the plot as a PNG file using the command:

```
print -dpng DAQ2_NETID1_NETID2
```

where NETID1 and NETID2 are the NET IDs of the people in the group. Append the file name as necessary if there are more people in the group. For example, `mrg`, `wns`, and `rap3` would create a graph using:

```
print -dpng DAQ2_mrg_wns_rap3
```

To have MATLAB help you determine the cutoff voltages, you can use the zoom feature and the `ginput` command. Once the plot has been created, expand the figure window to full size and zoom in on the particular subfigure containing the LED you want to characterize. Keep zooming until you clearly see the point at which the resistor voltage starts to turn away from 0 - in this case, when the resistor voltage is above 0.001 V. Next, type:

```
[VX, VY] = ginput(1)
```

in the command window. This will bring up crosshairs on the figure window. Click at the point on the top-left graph where the resistor voltage line intersects 0.001 V. When you are done clicking, MATLAB will report the x and y values of where you clicked by storing them in the `VX` and `VY` vectors, respectively. In this case, the `VY` value should be very close to 0.001 V and `VX` value should be the cutoff voltage of the LED. Repeat this process for the other six subplots and be sure to enter the data on the worksheet you are to fill out.

2.8 Lab Report

You will be submitting your answers on a handout distributed at the beginning of lab. You must also make sure to store your code and graph in your OIT directory - instructions for that will come later. Given the information you have collected for the six LEDs, you will answer the following:

(1) Based on your graphs, what is the approximate cutoff voltage for each of the different LEDs?

(2) Based on your answer above, discuss how you think the shape and/or color of an LED changes these values and whether the differences in these values are significant relative to the averages of the values. That is, if the values were 10.1, 10.2, 10.0, 10.0, 10.1, and 10.2, the overall differences (at most 0.2) would be small relative to the average of 10.1; on the other hand, if the values were 0.4, 0.5, 0.6, 0.4, 0.5, and 0.6, the overall differences (at most 0.2) are significant relative to the average of 0.5.

While grading, the TAs will go into the directories of the people in each group to verify the graph and also that files have been transferred over. Make sure you have copied everything into the correct directory. Go to the Start Button, then All Programs, then SSH Secure Shell, then Secure File Transfer Client. Log into your favorite OIT machine. The left side of the screen will be the file structure for the PC you are on and the right side will be the file structure of your OIT account. Note that on the PC, your files are stored in

```
C:\Documents and Settings\mrglocal\My Documents\MATLAB
```

On the OIT account, go into your EGR53 directory and then create a new folder called `daq2`. You can then drag and drop all your scripts (`.m`), data (`.mat`), and graphics (`.png`) files into your `daq2` folder. You do *not* need to copy any other files, such as `.asv` files. **Make sure** that *each person in the group* has all the required data and script files for this lab. There should be four scripts (`BasicAOutput`, `BasicAIO`, `AIO_`, and `VoltagePlotter_`), three data files (one each for the small, large, and rectangular LEDs), and one picture.

2.9 Cleaning Up

Once you are sure your files have been copied over, disassemble your circuit and put all the parts into the bucket. Delete all the files in MATLAB's `Current Directory` (either by selecting them all in the `Current Directory` window in MATLAB and deleting them or by going to the directory listed above and deleting the files). Next, empty the recycling bin on the desktop.

When you think your lab station is cleaned up and the files are cleared off, ask a TA to verify the contents of the bucket, the contents of your `MATLAB` directory, and the contents of the recycling bin - this will serve as attendance for the lab and you cannot receive a grade higher than 0 for this work without getting checked off for having properly stowed your gear and dumped your files.

For this week, leave MATLAB up and running and leave the Recycling Bin folder open on the screen. Do *not* log off the PC. Put the parts bin, breadboard, screwdriver, and wires back to the front of the room.

2.10 Circuit Diagrams and Network Listings

2.10.1 Basic Analog Output

Item	First	Second	Note
LEDG	Pin J7	Pin Y7	Put longer lead of LED in row J
LEDR	Pin J19	Pin Y19	Put longer lead of LED in row J
RES1	Pin D7	Pin G7	
RES2	Pin D19	Pin G19	
RED1	Pin A7	Line L22	DAC0OUT
RED2	Pin A19	Line L21	DAC1OUT
BLK1	Pin Y23	Line L54	AOGND

Table 2.1: Network Listing for Basic Analog Output

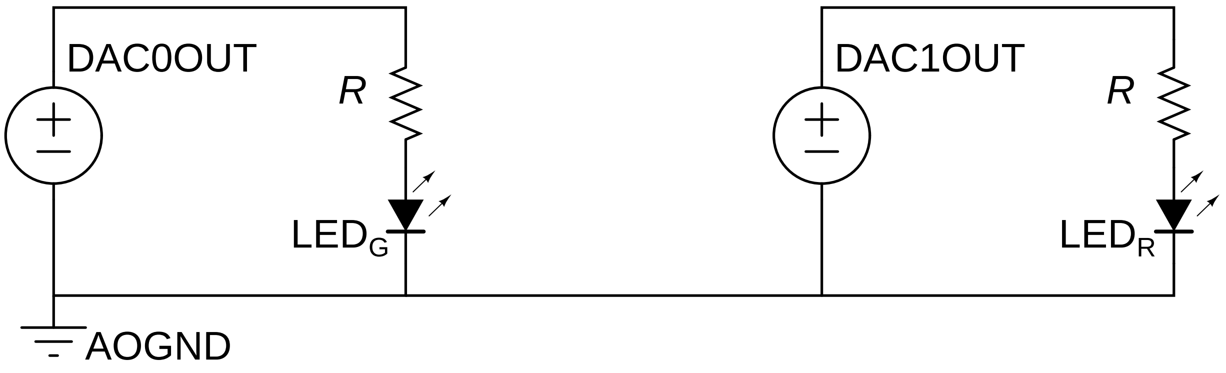

Figure 2.1: Circuit Diagram for Basic Analog Output

2.10.2 Basic Analog Input/Output

Addition	Item	First	Second	Note
	LEDG	Pin J7	Pin Y7	Put longer lead of LED in row J
	LEDR	Pin J19	Pin Y19	Put longer lead of LED in row J
	RES1	Pin D7	Pin G7	
	RES2	Pin D19	Pin G19	
	RED1	Pin A7	Line L22	DAC0OUT
	RED2	Pin A19	Line L21	DAC1OUT
	BLK1	Pin Y23	Line L54	AOGND
*	BLK2	Pin Y22	Line L56	AIGND
*	RED3	Pin B7	Line L68	ACH0: $v^{+}_{\text{total,left}}$
*	BLK3	Pin Y5	Line L34	ACH8: $v^{-}_{\text{total,left}}$
*	RED4	Pin B19	Line L28	ACH4: $v^{+}_{\text{total,right}}$
*	BLK4	Pin Y17	Line L61	ACH12: $v^{-}_{\text{total,right}}$

Table 2.2: Network Listing for Basic Analog Input/Output

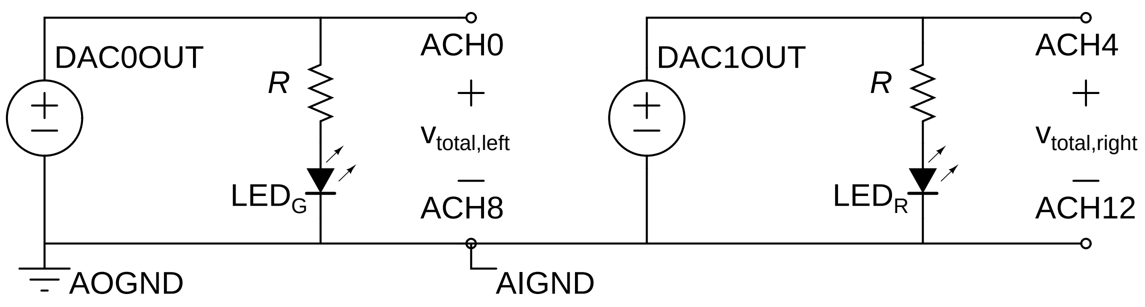

Figure 2.2: Circuit Diagram for Basic Analog Input/Output

2.10.3 LED Curves

Addition	Item	First	Second	Note
	LEDG	Pin J7	Pin Y7	Put longer lead of LED in row J
	LEDR	Pin J19	Pin Y19	Put longer lead of LED in row J
	RES1	Pin D7	Pin G7	
	RES2	Pin D19	Pin G19	
	RED1	Pin A7	Line L22	DAC0OUT
	RED2	Pin A19	Line L21	DAC1OUT
	BLK1	Pin Y23	Line L54	AOGND
	BLK2	Pin Y22	Line L56	AIGND
	RED3	Pin B7	Line L68	ACH0: $v^+_{\text{total,left}}$
	BLK3	Pin Y5	Line L34	ACH8: $v^-_{\text{total,left}}$
*	GRN1	Pin C7	Line L33	ACH1: $v^+_{\text{R,left}}$
*	BLK5	Pin H7	Line L66	ACH9: $v^-_{\text{R,left}}$
*	RED5	Pin I7	Line L65	ACH2: $v^+_{\text{LED,left}}$
*	GRN2	Pin Y3	Line L31	ACH10: $v^-_{\text{LED,left}}$
	RED4	Pin B19	Line L28	ACH4: $v^+_{\text{total,right}}$
	BLK4	Pin Y17	Line L61	ACH12: $v^-_{\text{total,right}}$
*	GRN3	Pin C19	Line L60	ACH5: $v^+_{\text{R,right}}$
*	BLK6	Pin H19	Line L26	ACH13: $v^-_{\text{R,right}}$
*	RED6	Pin I19	Line L25	ACH6: $v^+_{\text{LED,right}}$
*	GRN4	Pin Y16	Line L58	ACH14: $v^-_{\text{LED,right}}$

Table 2.3: Network Listing for LED Curves

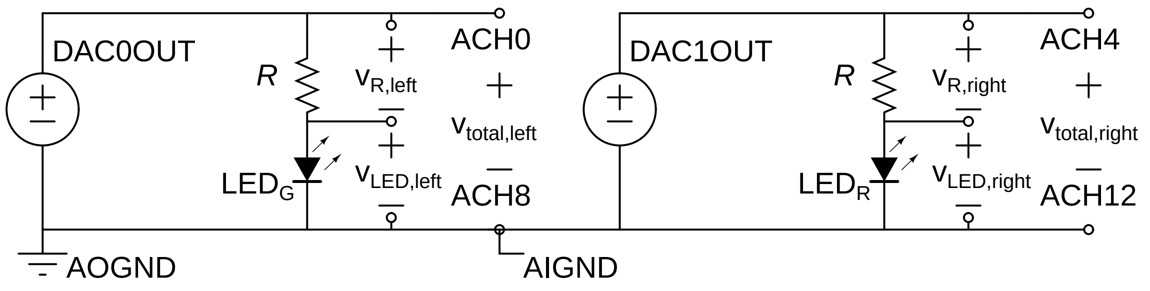

Figure 2.3: Circuit Diagram for LED Curves

2.11 Code for `BasicAOutput.m`

```
1   % Clear out workspace
2   clear
3
4   % Clear out DAQ objects
5   delete(daqfind)
6
7   % Create Analog Output Object
8   AO = analogoutput('nidaq', 1)
9
10  % Add channels to Analog Output Object
11  addchannel(AO, [0 1])
12
13
14
15
16
17
18
19  % Write values to output channels
20  putsample(AO, [5 5])
21  pause
22  putsample(AO, [0 0])
23
24  % Use loop to set several different voltages
25  for k=1:300
26      % Calculate voltages for each channel
27      Vout0 = 2.5+2.5*sin(2*pi*k/100);
28      Vout1 = 2.5+2.5*cos(2*pi*k/100);
29      % Put voltages to each output channel
30      putsample(AO, [Vout0 Vout1])
31      % leave line 31 blank for now
32      % leave line 32 blank for now
33      pause(0.02)
34  end
35
36
37
38
39
40
41
42
43  % Turn all outputs off
44  putsample(AO, [0 0])
```

2.12 Code for `BasicAIO.m`

```
% Clear out workspace
clear

% Clear out DAQ object
delete(daqfind)

% Create Analog Output Object
AO = analogoutput('nidaq', 1)

% Add channels to Analog Output Object
addchannel(AO, [0 1])

% Create Analog Input Object
AI = analoginput('nidaq', 1)

% Add channels to Analog Input Object
addchannel(AI, [0 4])

% Write values to output channels
putsample(AO, [5 5]);
pause
putsample(AO, [0 0]);

% Use loop to set several different voltages
for k=1:300
    % Calculate voltages for each channel
    Vout0 = 2.5+2.5*sin(2*pi*k/100);
    Vout1 = 2.5+2.5*cos(2*pi*k/100);
    % Put voltages to each output channel
    putsample(AO, [Vout0 Vout1])
    % Read voltages from all input channels
    Voltages(k,:) = getsample(AI);
    pause(0.02)
end

% Plot voltages versus index
n = 1:k;
Vleft = Voltages(:,1);
Vright = Voltages(:,2);
plot(n, Vleft, 'g', n, Vright, 'r')
legend('Green LED', 'Red LED', 0)

% Turn all outputs off
putsample(AO, [0 0])
```