Lab 2: Analog and Digital I/O

1. Introduction

Analog and digital signals are an important part of engineering experiments and mechatronic design. Recording data for experimental investigation requires an understanding of how data signals are read from inputs. Likewise, controlling a mechatronic or robotic device requires an understanding of how command signals are written to outputs. This lab will walk you through hands on procedures to read, manipulate, and output data in intelligent ways. This will be done using both Arduino microcontrollers and PC-based data acquisition (DAQ) devices from National Instruments. This lab will also teach you the skills to develop Graphical User Interfaces (GUIs) for visualization and control. Specifically, you will develop a MATLAB GUI for reading/writing data to/from the National Instruments DAQ. Remember, MATLAB is free through software.utah.edu if you don't already have a copy on your computer.

1.1 Lab Objectives

- Learn the fundamentals of discrete signals
- Learn how analog signals are converted to digital signals
- Learn how to interface with data acquisition devices
- Practice methods of Analog and Digital input using Arduinos
- Practice methods of digital and analog output (PWM) using Arduinos
- Learn how to develop MATLAB GUIs

1.2 Project Objectives

- Review team working agreement
- Review refined cad models, design metrics, and concept selection.

1.3 Lab Hardware

- 1 – Arduino
- 1 – DC Motor Shield
- 1 – LED
- 1 – 100 Ohm Resistor
- 1 – DC Motor kit
- 1 – Jumper wire kit
- 2 – BNC-to-BNC cables
- 2 – BNC-to-Test clips
- 1 – BNC-to-Banana adapter

1.4 Project hardware

- N/A

2. Laboratory Concepts

2.1 Digital and Analog signals

For a signal to be **analog** means it is continuously variable and can have an infinite amount of different values. While the range of a signal may be finite, the actual value can be any possible value in between its limits. For a signal to be **digital**, it will have a finite or **discrete** number of possible values. This could mean, for example, two possible values or 1,000 possible values. Either way the number of possible values is limited.

2.2 Data Acquisition Devices (DAQs)

Data acquisition devices, or DAQs, are devices that allow computers and microprocessors to read in voltages from sensors or other external sources. One of the most basic and important functions of a DAQ is analog input, a.k.a. Analog to Digital (A/D) conversion. The analog voltage that you connect to the DAQ is converted into a digital number and stored in computer memory.

Example functions of a DAQ

- Analog input
- Analog output
- Digital input/output
- Timing/counting

DAQs often have additional functionality, such as analog output, a.k.a. Digital to Analog (D/A) conversion. This allows you to use your computer to output voltages for powering sensors or controlling various types of actuators. The D/A channels are separate from the A/D channels and may or may not have the same range, resolution, and speed. It is also common for a DAQ to have several digital inputs and outputs for communicating with other digital devices or for turning things off and on. Digital inputs and outputs often share the same channels, where a given channel can be configured as either an input or an output via software. A digital channel is always low or high, usually corresponding to 0 V or 5 V, like shown in Fig.1 below.

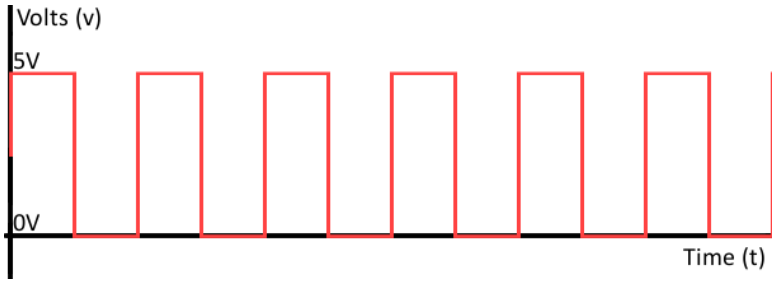


Figure 1. Values of a digital channel are only high or low (5V or 0V).

Finally, DAQs usually have their own internal clocks and counters. Therefore, all the inputs and outputs can take place at regular time intervals, and external events can be counted.

2.3 Analog to Digital Conversion (A/D)

In order to use computers or data acquisition to measure analog signals A/D conversion must take place. This is the act of breaking up a signal into a discrete number of possible values. A visual example of this process is shown in the analog (real signal) and the digital representation of that signal in Fig. 2. The following section will explain the process of A/D in more detail.

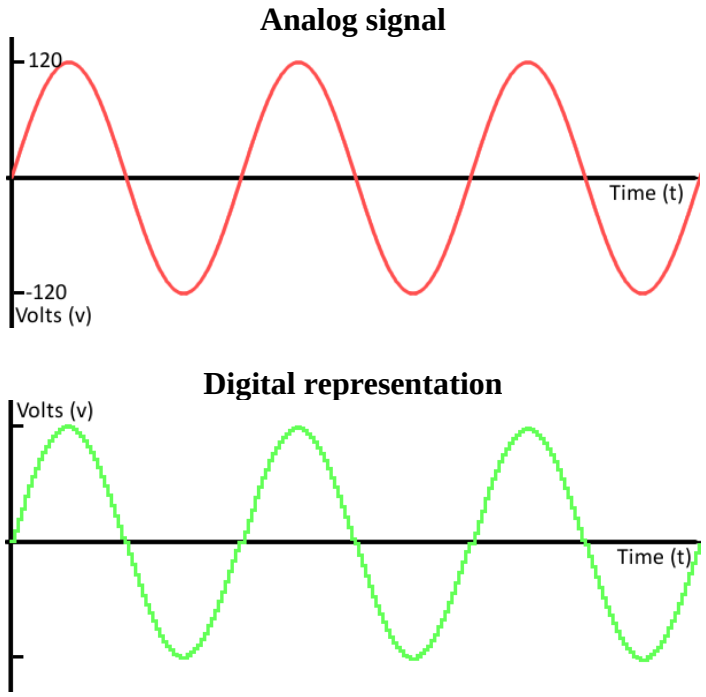


Figure 2. Smooth analog signal (top) and stair case digital signal (bottom).

Important characteristics of a DAQ's A/D are given below. By now you should be familiar with most of them. The number of channels of A/D usually depends on whether we choose to use differential mode (measure difference between two voltages) or single-ended mode (measure single voltages with respect to ground).

DAQ A/D Conversion Characteristics

- input range (e.g. ± 10 V)
- resolution (e.g. 16 bit)
- speed (e.g. 100,000 samples/second)
- input impedance (e.g. 100 MΩ)

A/D conversion is dependent on the factors shown above. Further labs will instruct on these issues. The resolution of a converter indicates the number of discrete values it can produce over the range of analog values. Values are usually stored electronically in binary form, so the resolution is usually expressed in bits (binary digits, discussed later). In consequence, the number of discrete values available, or "levels", is assumed to be a power of two. For example, an A/D with a resolution of 8 bits can encode an analog input to one in 256 different levels, since $2^8 = 256$. The

values can represent the ranges from 0 to 255 (i.e. unsigned integer) or from −128 to 127 (i.e. signed integer), depending on the application.

Resolution can also be defined electronically and expressed in volts. The minimum change in voltage required to guarantee a change in the output code level is called the Least Significant Bit (LSB) voltage. The resolution Q of the A/D is equal to the LSB voltage. The voltage resolution of an A/D is equal to its input voltage range divided by the number of intervals, where M is the number of bits:

$$Q = \frac{V_{HI} - V_{LOW}}{2^M}$$

2.4 Decimal and Binary Conversion

In order for A/D to remember values, the data must be saved in bits. As the name suggests, bits are binary, meaning on or off, 1 or 0, true or false. Bits are a base 2 system compared to the decimal system that is base ten. In the decimal number system, the weight of each digit increases by a factor of 10 (i.e. 10^i). In the binary number system, the weight of each digit increases by a factor of 2 (i.e. 2^i). Each binary digit will have the following values in decimal form:

Digit *i*	7	6	5	4	3	2	1	0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Value	128	64	32	16	8	4	2	1

Converting from Binary to Decimal

Converting from binary to decimal is done by computing the decimal value of each binary digit and then adding them up as shown in this example for the binary number: 10100101.

Binary digit place	7	6	5	4	3	2	1	0
Binary digit value	1	0	1	0	0	1	0	1
Decimal digit value	128	0	32	0	0	4	0	1

$$= 128 + 32 + 4 + 1 = 165$$

Converting from Decimal to Binary

One way to convert from decimal to binary is to iteratively divide the decimal number by 2, using the remainders to form the binary digits. The following example shows how to convert from decimal to binary for the decimal number 178. ***Starting on the right*** divide by 2. If the number is odd, the remainder would be 1 and the quotient would round down to the nearest digit. The remainder becomes your binary number: $178_{10} = 10110010_2$ (the subscript here denotes the number's base system).

	$\frac{1}{2} = 0$	$\frac{2}{2} = 1$	$\frac{5}{2} = 2$	$\frac{11}{2} = 5$	$\frac{22}{2} = 11$	$\frac{44}{2} = 22$	$\frac{89}{2} = 44$	$\frac{178}{2} = 89$
Remainder	1	0	1	1	0	0	1	0

2.5 Arduino Analog and Digital I/O

This section will discuss how to properly use the Arduino's pins for input and output operations for both digital and analog pins.

Digital pins

The Arduino Uno includes 14 digital pins which are clearly numbered from 0-13 that can be set either to input or output by using the command `pinMode(pin, mode)`, where `mode` can be either `INPUT` or `OUTPUT`. Digital pins can only detect on or off (nominally 5V or 0V). They can only output 0V or 5V as well. Some of these pins can be used for PWM analog output, will be discussed below. In order to read a digital input or generate a digital output you must use the following commands while the appropriate pin is set to the corresponding mode:

- `digitalRead(pin)`: Reads the voltage at a specified digital pin, either HIGH (5V) or LOW (0V).
- `digitalWrite(pin, value)`: Output a HIGH (5V) or a LOW (0V) voltage from a digital pin. Note that on most boards this can be used on pin 13 (once its mode has been set to output) to control the on-board LED.

Analog input pins

The Arduino Uno includes 6 analog pins that are clearly labeled from 0-5. In order to reference the analog pins, put an 'A' in front of whatever pin is desired (i.e. A0-A15 for the Uno). In order to read from an analog input, you must use the following command:

- `analogRead(pin)`: Reads the voltage from the specified analog pin.

PWM pins

Digital pins can provide the illusion of any voltage between 0V and 5V using Pulse Width Modulation (PWM), like shown in Fig. 3. PWM is where a pin (which can only provide 0V or 5V) is repeatedly changed between on and off. If this is done fast enough relative to the dynamics of the load (~1000 times per second is often sufficient for mechanical or mechatronic systems) the voltage effectively becomes equal to the ratio of the on time over the period (the on time plus the off time). Thus a 5 V PWM signal with 33% duty cycle (powered on for 33% of the time) will look like a ~1.67 V signal. The Arduino command to generate a PWM signal is the following:

- `analogWrite(pin, value)`: Writes an analog value (PWM wave) to a pin. The 'value' is the duty cycle: between 0 (0% duty cycle) and 255 (100% duty cycle). Note that on most boards this can be used on pin 13 (once its mode has been set to output) to control the on-board LED.

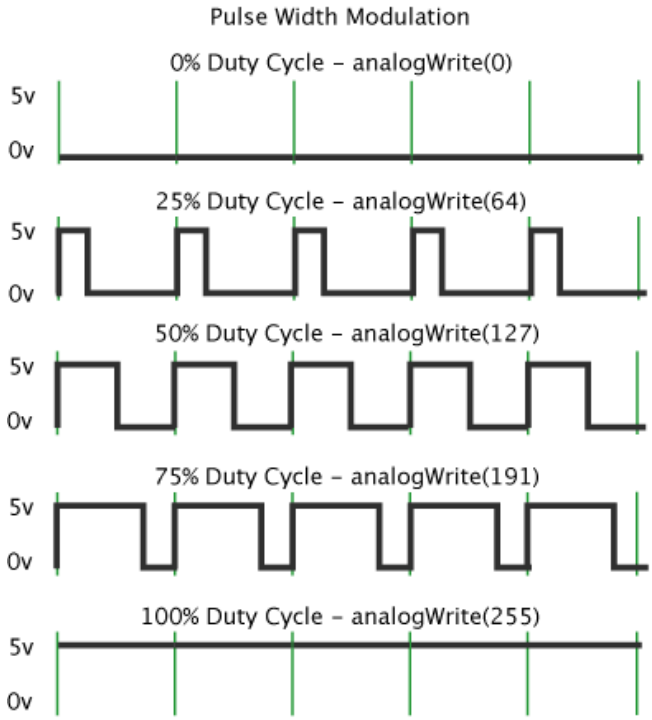


Figure 3. Signal representation of Pulse Width Modulation (PWM). The input is an 8-bit number, where 0 results in a 0% duty cycle, and 255 results in a 100% duty cycle.

2.6 MATLAB GUI Info

A Graphical User Interface (or GUI) is simply a window on your computer that contains buttons, menus, text boxes, check boxes, track bars, scroll bars, etc. that allow you to control some process on the computer with your mouse and receive visual feedback in graphical form about the state of the process. This is in contrast to a computer program where your main input is in the form of entering text with the keyboard, and your main feedback is text displayed on the screen.

One of the most useful tools is a GUI whose purpose is to help the user create other Graphical User Interfaces. This is the purpose of the App Designer interface in MATLAB; it is a GUI for creating GUIs (or an app for creating apps). It allows you to click and drag your mouse to create the buttons, text boxes, menus, etc. that you will use to form your GUI.

Open the App Designer interface using the APPS menu in MATLAB or by typing `appdesigner` in the MATLAB command line, and then select a Blank App. Then look for the buttons that toggle the interface between *Design View* and *Code View*. When *Design View* is selected, you can select various objects from the *Component Library* on the left and click and drag them onto your canvas to create buttons, text boxes, and other objects. If you select use *Code View*, you can view the "behind the scenes" MATLAB code that controls your GUI. Most of the code that controls the layout of your GUI is autogenerated and cannot be edited. However, you will be able to add and edit certain sections of code to control the functionality of your GUI. By working back and forth between the *Design View* and *Code View*, you can create custom MATLAB GUIs (a.k.a. apps) for simulation, data acquisition, visualization, and more.

Note that all the objects in your GUI are listed in the *Component Browser* panel on the right side of the interface. If you click on any object that you have added to the GUI (either on the canvas or in the *Component Browser* list), you can then view and edit various properties of the

object. If you select *Callbacks*, you can add a "callback function" to the object. The "callback function" executes when you run your app and interact with the object. Interactions include pressing buttons, moving sliders, editing values, etc. Once a callback function has been created, you can switch to the *Code View* and add your own code to the callback function(s) to make your GUI do what you want in response to various interactions.

The primary property of any object is its name. You can rename an object by double-clicking on it in the *Component Browser*. Any changes you make to the object's name will then auto-update in the *Code View*. It is helpful to assign meaningful names to each of your objects so that you can easily recognize and use them in the code. To retrieve or set any property of the object within your code, you can use the syntax `app.Object.Property`, where `Object` is the name of the particular object, and `Property` is the name of the particular property.

A great way to learn how to use GUIs is to develop your own and use the MATLAB documentation online as a resource. The following link describes how to run the App Designer Tutorial: https://www.mathworks.com/help/matlab/creating_guis/create-a-simple-app-or-gui-using-app-designer.html . The 5 minute "Getting Started with App Designer" video (https://www.mathworks.com/videos/app-designer-overview-1510748719083.html) may also be helpful. Note that App Designer has replaced the old GUIDE interface in MATLAB for creating GUIs. The GUIDE interface will soon be deprecated is no longer recommended.

2.7 MATLAB Data Acquisition Toolbox

In order to interface with the DAQ hardware in the PC and acquire/manipulate data, you can use a variety of software packages like MATLAB or LabVIEW. Using such software packages, you can immediately manipulate data, automate data acquisition tasks, synchronize data from multiple devices, control inputs and output properties, and save data for post processing. In this lab you will learn the basics of Analog and Digital I/O using MATLAB's Data Acquisition Toolbox. For more information and tutorials please view the MathWorks documentation at https://www.mathworks.com/help/daq/index.html.

Please view the following tutorials in order to answer questions in the prelab concerning MATLAB Data Acquisition Toolbox.

https://www.mathworks.com/help/daq/examples/acquire-data-using-ni-devices.html

https://www.mathworks.com/help/daq/generate-signals-on-both-analog-and-digital-channels.html

3. Pre-Lab Exercises

Many of the answers can be found in the lab handout, but some require you to investigate documentation indicated in the prelab.

1. A/D basics
   a. Describe (with detail) the difference between analog and digital signals.
   b. Convert the following decimal to binary. Show your work.
      37
      121
      360
   c. Convert the following binary numbers to decimal. Show your work.
      1101
      01110
      10101100

2. Arduino ADC
   a. What values can you expect from an `analogRead()` function call? What voltages do these values correspond to?
   b. What values can be expected and what voltages do they correspond to a `digitalRead()` operation?
   c. What process does the `analogWrite()` function do to output a variable signal? What pins on the Arduino Uno can do this?

3. MATLAB GUI
   Build the GUI following the steps described below in Section 3.1 MATLAB GUI SETUP. Upload the file Yourname_Lab2_GUI.mlapp to this assignment, where "Yourname" is literally your name, but without any spaces. Doing the tutorial mentioned in Section 2.6 first may help if you are feeling uncomfortable.

3.1 MATLAB GUI SETUP (Digital Oscilloscope)

In this prelab exercise you will develop your own MATLAB GUI from scratch. Using the App Designer interface and MATLAB session-based data acquisition, you will develop a basic scope to plot analog data to the screen as well as have the functionality to save that data to a file. This GUI will be used and expanded in further exercises and labs. It may also be useful for portions of your project for diagnostics and testing. When you are finished with exercise 4.1 your GUI will look like the following figure.

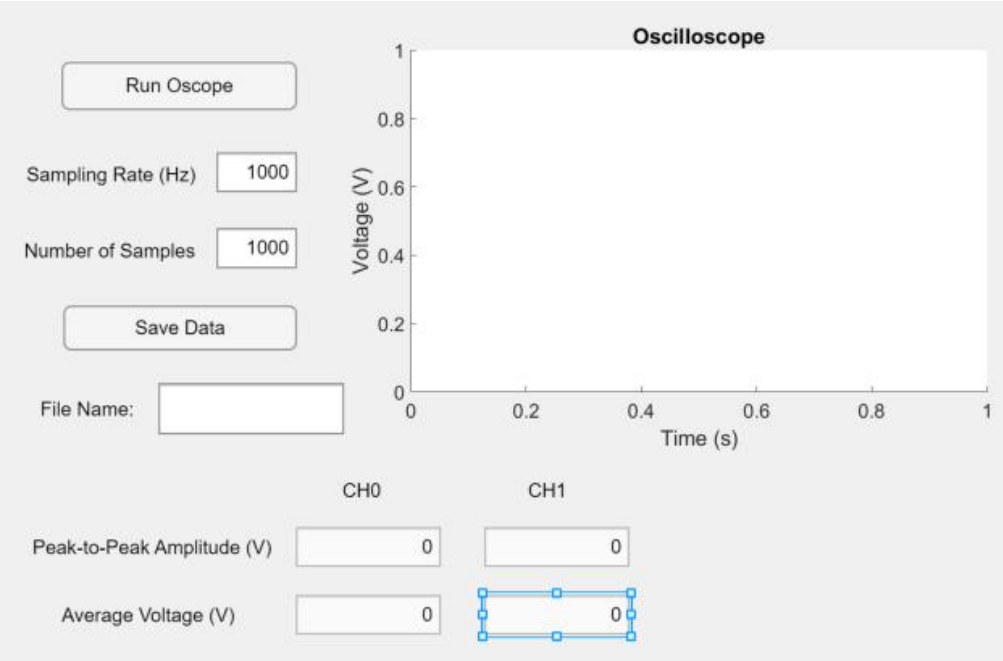


Figure 4. GUI layout at the end of section 4.1.

3.1.1 New GUI

To begin, open MATLAB and set your current folder to a safe place (e.g. on a flash drive or dedicated folder on the desktop). **Be sure to save all files (e.g. on your flash drive or upload to your cloud storage when finished) for future labs.** Follow the proceeding steps to create your own GUI:

- Type **`appdesigner`** into the MATLAB command window
- In the App Designer Start Page select a new Blank App
- When new app appears, find the Save icon in the menu above, select *Save as*, and name your file Yourname_Lab2_GUI.mlapp **(where "Yourname" is your name)** and make sure the file is being saved in the correct folder destination.

3.1.2 Create Plotting axes

In this step, you will create the axes that will be used to display the signal that is inputted to the DAQ.

- In the *Design View*:
  - In the *Component Library* on the left side of the interface, find and click on the Axes object. Then click and drag on your canvas to make an Axes object the size you want.

If you want, you can specify the exact size and position by finding and editing the POSITION property in the *Component Browser* on the right side of the interface.

- Next, we will edit the name of the axes object. If you look in the list of objects at the top of the *Component Browser*, you will see the default name of your axes object is UIAxes. Double-click on the name and change it to `OscopeAxes`
- Next find the LABELS property group and edit the properties as follows:
  - Title.String: Oscilloscope
  - XLabel.String: Time (s)
  - YLabel.String: Voltage (V)
- Verify that the labels have changed on your axes object in the canvas

3.1.3 Create Run button

The function of this button will be to start and stop the data acquisition process.

- In the *Design View*:
  - In *Component Library*, find and click on the *State Button* object. Then click and drag on your canvas to make a *State Button* object the size you want.
  - Edit the name of the button and name it `RunButton`
  - Change the button label to `Run Oscope`. You can do this by double clicking on the button itself, or looking under the BUTTON property group and editing the *Text* property. Note that the label does not have to match the name of the object, even though they start out the same. The label is what the user of the app will see, but the name is what will be used in the code.
  - You can change the text size and style of the label using the FONT properties if you want.
  - While the RunButton is still selected, click on *Callbacks* and add a ValueChangedFcn. Note that if you switch to *Code View*, there is now a new function called RunButtonValueChanged, and the code inside this function is editable. Later we will add some code here, but first we need to create some more objects in our GUI.

3.1.4 Create Sample rate and Duration dialog boxes

The function of these two dialogs will be to set the rate of sampling you want the DAQ to use, and the duration of the data recording.

- In the *Design View*:
  - Add two Edit Fields (Numeric) and size them to your preference
  - For one of the Edit Fields change the name to `RateEditField` and the Label to `Sampling Rate (Hz)`. Change the Value to 1000.
  - For the other Edit Field, change the name to `DurationEditField` and the Label to `Duration (s)`. Change the Value to 1.

- In the *Code View*:
  - Add the following code to the `RunButtonValueChanged` function:

```
while(app.RunButton.Value)
    rate = app.RateEditField.Value;
    duration = app.DurationEditField.Value;

    % These next lines are for the pre-lab only.
    % Comment them out after you are in lab and
    % replace with commands to real DAQ hardware.
    time = 0:1/rate:duration;
    data = sin(10*time);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    plot(app.OscopeAxes,time,data);
    drawnow;
end
```

- Save and run the app. Click the Run Oscope Button and verify that your plot displays a sine wave. Technically the plot is repeatedly redrawing itself, but you can't tell since our simulated data is always the same. You can click the button again to stop the loop. Try interacting with the edit fields and run the oscope with different sample rates and durations.

4. Laboratory Exercises

In this lab exercise you will continue editing the MATLAB GUI developed in your prelab. You can download it from your Canvas prelab submission, bring it with you on a jump drive, or download it from your favorite cloud storage. Open MATLAB on your lab PC and then open your GUI in the App Designer. Make sure the DAQ device is plugged in before opening MATLAB.

4.1 Data Acquisition, Analysis, and Display

4.1.1 Data Acquisition. We first need to add some code to the GUI to interface with the data acquisition card in your lab PC.

- In the *Code View*:
  - Look in the *Code Browser* panel on the left of your app designer window and click on *Properties*. Then click on the green plus symbol. Note that a new section of editable code now appears where you can declare your own variables. These variables belong to the app and are accessible from any of the functions in your app using the syntax *app.variablename*. Add the following variables to your properties:

```
properties (Access = private)
    Ain   % Analog Input Session
    Dout  % Digital Output Session
    CH0   % Analog input channel 0
    CH1   % Analog input channel 1
    Time  % Array of time samples
    Data  % array of data samples
    Aout  % Analog output session
end
```

  - Next look under the *Component Browser* panel on the right side of your app designer window and select the GUI itself by clicking on Yourname_Lab2_GUI at the top of the list of objects in your app. Then select *Callbacks* below and add a StartupFcn. Note that a new editable function called `startupFcn` now appears in your code. Code that you add in this function will execute once when you first run your app. This is a great place to initialize your data acquisition hardware.
  - Add the following code to your `startupFcn`:

```
% Code that executes after component creation
function startupFcn(app)
    app.Ain = daq("ni");
    app.CH0 = addinput(app.Ain,"Dev2","ai0","Voltage");
    app.CH1 = addinput(app.Ain,"Dev2","ai1","Voltage");
    app.CH0.TerminalConfig = "SingleEnded";
    app.CH1.TerminalConfig = "SingleEnded";
```

  - Now find your `RunButtonValueChanged` function. Comment out the section of simulated data and edit your function to the following:

```
function RunButtonValueChanged(app, event)
    while(app.RunButton.Value)
        rate = app.RateEditField.Value;
        duration = app.DurationEditField.Value;

        % These next lines are for the pre-lab only.
        % Comment them out after you are in lab and
        % replace with commands to real DAQ hardware.
        % time = 0:1/rate:duration;
        % data = sin(10*time);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Acquire real data from the DAQ hardware
        app.Ain.Rate = rate;
        result = read(app.Ain,seconds(duration));
        app.Time = result.Time;
        app.Data = result.Variables;

        plot(app.OscopeAxes,app.Time,app.Data);
        legend(app.OscopeAxes,'CH0','CH1');
        drawnow;
    end
end
```

- Save your app and run it.

4.1.2 Test current progress

- Using a BNC-to-BNC cable, connect the AI0 on the NI DAQ to the function generator.
- Set the function generator to a sine wave with a frequency of 10 Hz
- Verify that when you activate the Run Oscope button, you see the real 10 Hz signal from the function generator displayed in your oscilloscope axes. Now you should see the plot update every time the sampling loop repeats. Change the frequency and amplitude on your function generator and see what happens. You can also connect the function generator to AI1 on the NI DAQ and verify that you can acquire a signal on Channel 1 as well.

Have a TA check your progress and sign you off. Continue working if your TA is helping another student and cannot check you off within a minute or two. Be sure that your TA signs you off before you leave.

4.1.3 Create saving function

This function will allow you to save your acquired data to a .mat file.

- In the *Design View*:
  - Add a *Button* with the name `SaveButton` and the label `Save Data`
  - Add an *Edit Field (Text)* with the name `FileNameEditField` and label `File Name`
  - Size and format their appearance as you prefer

- Add a *ButtonPushedFcn* callback function to your `SaveButton`

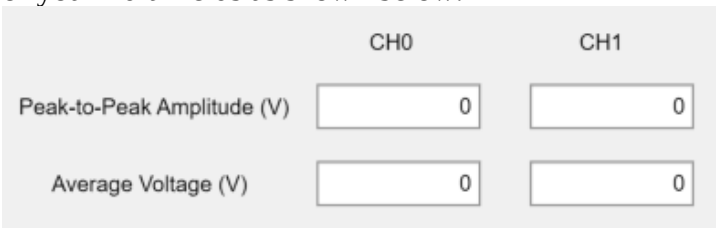
- In the *Code View*:
  - In the `SaveButtonPushed()` function, add the following lines of code

```
filename = app.FileNameEditField.Value;
rawData = app.Data;
timeData = app.Time;
save(filename,'rawData','timeData');
```

  - Save and run your app. Use the Run Oscope button to acquire some data and then stop. Then enter a filename in the edit field and hit the Save Button.
  - Look in your MATLAB window to make sure the file got saved in your working directory. You can type `load filename` at the command prompt to load the data to your workspace. Two variables should load into your workspace: *rawData* and *timeData*. *rawData* should be an *n*x2 array containing the data from Channels 0 and 1.

4.1.4 Detect and Display the Peak-to-Peak Amplitude and Average Voltage of Signals

- In the *Design View*:
  - Add an *Edit Field (Numeric)* and delete the label (click on the label so that only the label is highlighted and hit the delete button). Find the *Editable* property under the INTERACTIVITY property group, and uncheck the box.
  - Then use copy/paste to make 3 more identical Edit Fields. Arrange these four Fields in a 2x2 array. Then name them `CH0AmpField`, `CH1AmpField`, `CH0AveField`, and `CH1AveField.`
  - Add four *Labels* from the *Component Library*. Use them to label the rows and columns of your Edit Fields as shown below:



- In the *Code View*:
  - Add the following code to the while loop in the `RunButtonValueChanged()` function after the plotting step

```
min0 = min(app.Data(:,1));
max0 = max(app.Data(:,1));
ave0 = mean(app.Data(:,1));
min1 = min(app.Data(:,2));
max1 = max(app.Data(:,2));
ave1 = mean(app.Data(:,2));
app.CH0AmpField.Value = max0-min0;
app.CH1AmpField.Value = max1-min1;
app.CH0AveField.Value = ave0;
app.CH1AveField.Value = ave1;
```

- Save and run your app and see if the GUI displays the correct peak-to-peak and average voltages for your signals.

4.1.5 Create a closing function

This new function will make sure the data acquisition is not still running when you close the app, and prevent associated error messages from occurring.

- In the *Code View*:
  - Select the *app.UIFigure* object at the top of the list under the *Component Browser* and then click on *Callbacks*.
  - Underneath, add a *CloseRequestFcn* callback. Note that a new editable function called `UIFigureCloseRequest()` now appears in the code.
  - If CloseRequestFcn is not available in the droplist of options, you likely have an object selected. Click onto the background of the figure so no object is highlighted, and your list of callback functions should now include CloseRequestFcn. This function will execute when the user attempts to close your app.
  - Edit your UIFigureCloseRequest() function to look like the following:

```
function UIFigureCloseRequest(app, event)
    if(app.RunButton.Value == 0)
        delete(app)
    else
        errordlg('Please stop data aquisition before closing the app');
    end
end
```

- Save and run your app. Test the functionality of your closing function by trying to close your app while the scope is still running.

Have a TA check your progress and sign you off. Continue working if your TA is helping another student and cannot check you off within a minute or two. Be sure that your TA signs you off before you leave.

4.2 Digital I/O

In this exercise, you will add a Toggle Switch to the GUI and will output a digital signal when the switch in On. This digital signal will be read by the Arduino and will light an LED when the digital signal is 5V (e.g., True, 1, on, etc). This button will also be used in the following exercise. When you are finished with your GUI in exercise 4.2, it will look like the following figure.

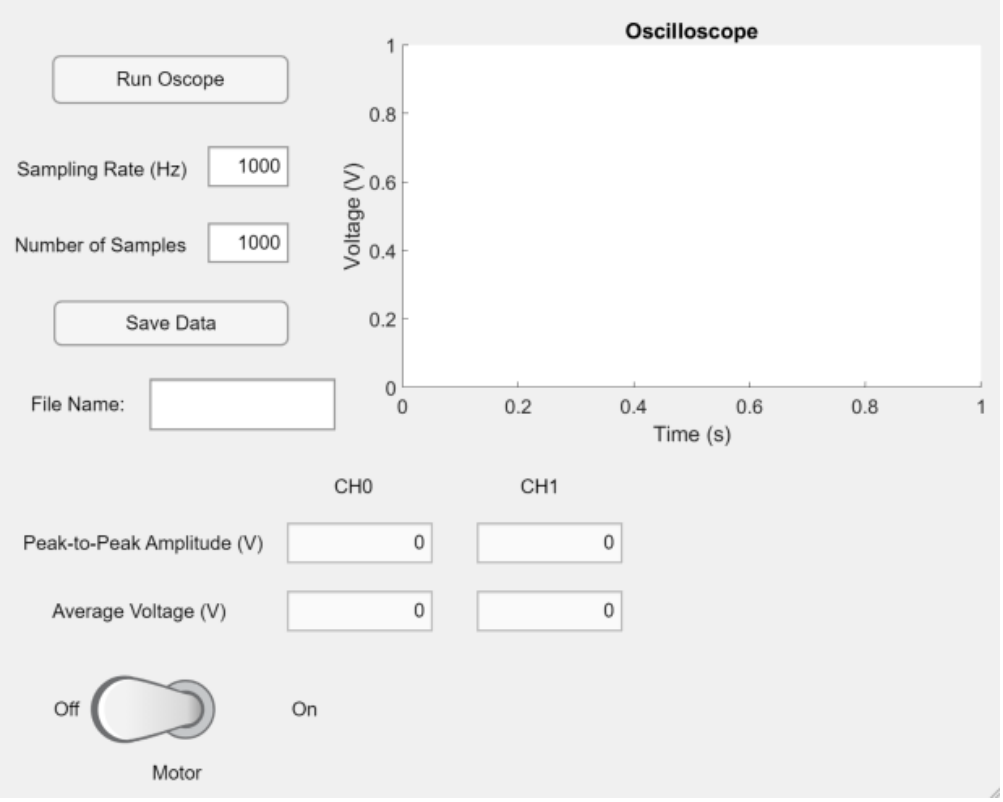


Figure 5. GUI layout at the end of section 4.2.

4.2.1 Add a Toggle Switch to the GUI

- In the *Design View*:
  - Add a Toggle Switch to the canvas
  - Change the Orientation property to horizontal
  - Change the label to Motor
  - Change the name to MotorSwitch
  - Add a ValueChangedFcn to the Toggle Switch

4.2.2 Digital output through DAQ

- In the *Code View*:
  - In the `startupFcn()` add the following lines of code to initialize the digital output channel.

```
app.Dout = daq("ni");
addoutput(app.Dout,"Dev2","port0/line0","Digital");
```

- Modify the `MotorSwitchValueChanged()` function to the following to set the digital output high or low when the switch is toggled. Note that for the Toggle Switch, the Value is not a number, but rather a string that is "On" or "Off". We will use the strcmp() function in MATLAB to compare strings.

```
function MotorSwitchValueChanged(app, event)
    value = app.MotorSwitch.Value;
    if(strcmp(value,'Off'))
        write(app.Dout,0);
    else
        write(app.Dout,1);
    end
end
```

4.2.3 Go to Canvas for this class and go to Files\Lab Handouts\Lab 02 and download the Arduino sketch `Lab2_arduino.ino`.

- Start the Arduino IDE, connect the Arduino to the computer using the USB cable, and open the `Lab2_arduino.ino` sketch.
- The sketch contains the following Arduino code, plus code for the next section, which initializes digital pins, reads digital pins, and lights up an LED. Take a moment to review the code below. Read the comments indicating what each line does.
- Be sure the proper board is selected and upload the software to the Arduino.

```
#include "DualTB9051FTGMotorShield.h"
DualTB9051FTGMotorShield md;

const int Apin = 0; // Analog input pin for motor voltage
const int Dpin = 6; // GDigital input pin from DAQ
const int LEDpin = 13; // Digital output pin for LED

void setup() {
  Serial.begin(9600); // Initialize serial monitor
  pinMode(Dpin,INPUT); // Set Dpin as input
  pinMode(LEDpin,OUTPUT); // Set LEDpin as output
  md.init(); // Initialize motor controller
  md.enableDrivers(); // Necessary initialization function
}

void loop() {
  int button = digitalRead(Dpin);// Read DAQ digital state
  int Mval = analogRead(Apin); // Motor voltage 0-1023
  int M = 0;

  if(button) { //if true
    digitalWrite(LEDpin,HIGH); // then light the LED
    M = map(Mval,0,1024,-400,400); // motor control value
  }
  else{ //if false
    digitalWrite(LEDpin,LOW); // then turn off LED
    M = 0; // then don't power motor
  }
  //md.setM1Speed(M); // command voltage to motor
}
```

4.2.4 Connect Arduino/motor controller to NI DAQ

- With a jumper wire connect the P0.0 pin on the NI DAQ to the Arduino digital pin 6
- With a jumper wire connect the DGND on the NI DAQ to the Arduino GND

4.2.5 Connect an external LED to Arduino pin 13 in series with a 100 Ohm resistor

4.2.6 Test the GUI with the Arduino, the light should turn on and off when you toggle the Motor Switch.

Have a TA check your progress and sign you off. Continue working if your TA is helping another student and cannot check you off within a minute or two. Be sure that your TA signs you off before you leave.

4.3 Analog I/O

In this exercise, you will be adding a slider to the GUI. This slider will then be used to send an Analog signal through the NI DAQ to the Arduino. Using this signal, you will control a DC motor to turn both directions (i.e. -5V to 5V output) if the toggle button from the previous exercise is press ON. At the end of this exercise your GUI will look like the following figure.

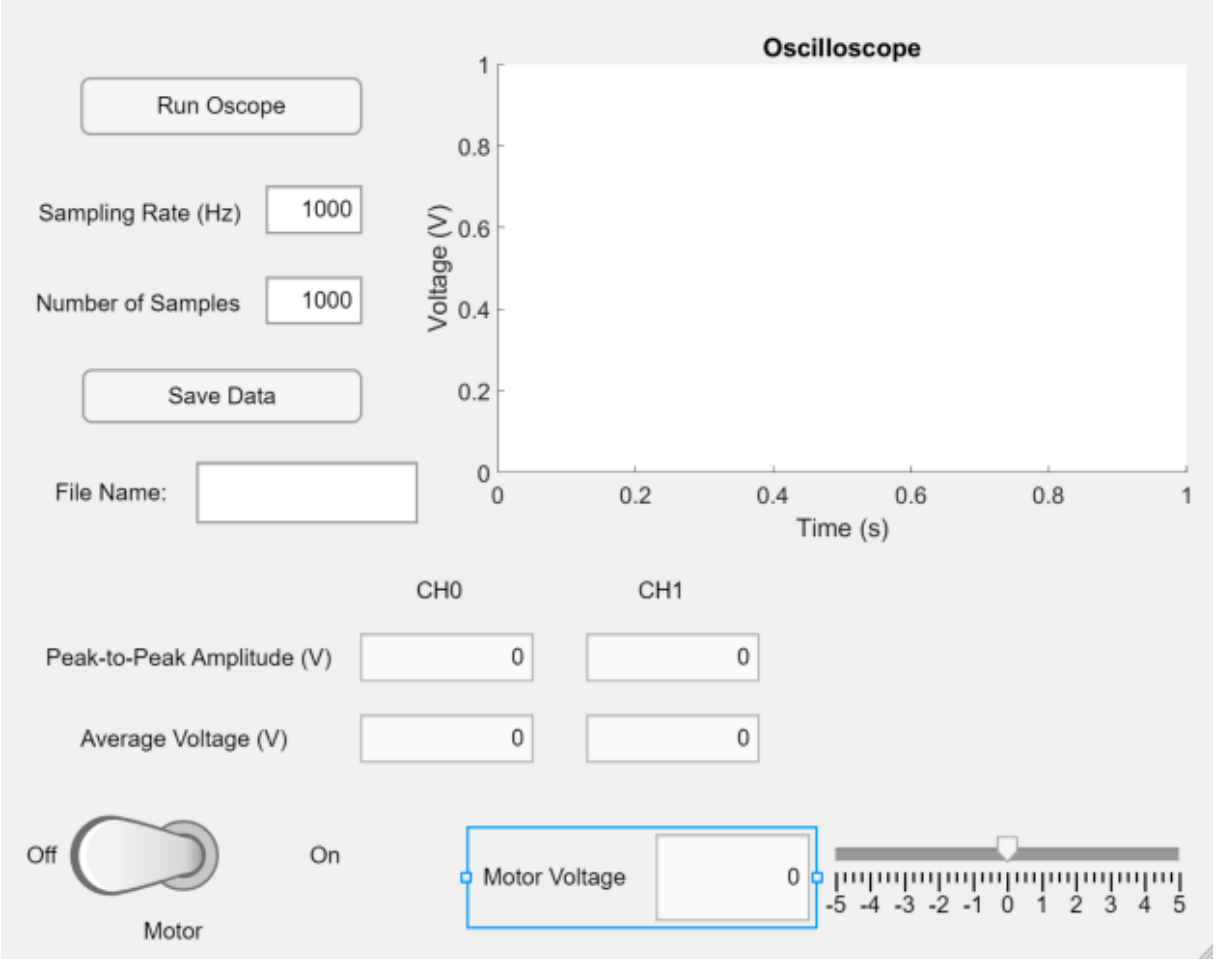


Figure 6. GUI layout at the end of section 4.3.

4.3.1 Add slider in GUI

- In the *Design View*:
  - Add an *Edit Field (Numeric)* to the GUI. Change the name to MotorVoltageEditField and the label to Motor Voltage (V). Find and uncheck the Editable property.
  - Add a *Slider* to the GUI and size it to your preference. Delete the label.
  - Change the slider name to MotorVoltageSlider
  - Change the Limits property of your slider to -5,5
  - Add a ValueChangingFcn callback to your slider

- In the *Code View*:
  - Modify your `MotorVoltageSliderValueChanging()` callback function to look like this:

```
function MotorVoltageSliderValueChanging(app, event)
    MotorVolts = event.Value;

    % display the motor voltage from the slider in the edit field
    app.MotorVoltageEditField.Value = MotorVolts;
end
```

- Run your app and test the slider. As you drag the slider back and forth, the Motor Voltage edit field should display the value from the slider

4.3.2 Add analog output from DAQ. Note that we will use the analog output of the DAQ to send the desired voltage command to an Arduino, and the Arduino will then control the motor. Therefore we will need to map the desired voltage command (-5 to 5V) to the range of the Arduino analog input (0 to 5V). Once the Arduino receives this command, it will then decode and remap it to control the motor.

- In the *Code View*:
  - In the `startupFcn()` function, add the following lines of code to initialize the Analog output channel:

```
app.Aout = daq("ni");
addoutput(app.Aout,"Dev2","ao0","Voltage");
```

  - In the `MotorVoltageSliderValueChanging()` callback function add the following lines of code.

```
% Map motor voltage from range of [-5,5]
% to range of [0,5] for Arduino to
% be able to receive it.
ArduinoVolts = (MotorVolts+5)/2;
% Update analog output
write(app.Aout,ArduinoVolts);
```

  - Next modify the `MotorSwitchValueChanged()` function so that it resets the voltage command to zero if the switch is toggled:

```
function MotorSwitchValueChanged(app, event)
    % Reset the Motor Voltage command to 0
    app.MotorVoltageSlider.Value = 0;
    app.MotorVoltageEditField.Value = 0;
    % Motor voltage of 0 maps to Arduino voltage of 2.5
    ArduinoVolts = 2.5;
    % Update digital and Analog outputs
    if(strcmp(app.MotorSwitch.Value,'Off'))
        write(app.Dout,0);
    else
        write(app.Dout,1);
    end
    write(app.Aout,ArduinoVolts);
end
```

- Save and run your GUI and verify that everything works. You can connect the AO0 pin on the NI DAQ to the AI0 pin on the NI DAQ and use your GUI's Oscope to measure the voltage being outputted.

4.3.3 Note that the Lab2_arduino.ino sketch includes the following to code to read an analog signal and control a motor. Review the sketch and note the following lines of code in the sketch.

- The motor driver library is included and creates the object as shown below. You may need to download the library from the Library Manager (i.e., Sketch menu, Include Libraries item, then Library Manager, and finally search for DualTB9051FTGMotorShield)

```
#include "DualTB9051FTGMotorShield.h"
DualTB9051FTGMotorShield md;
```

- This line initializes the analog pin as a global variable

```
const int Apin = 0;
```

- This line initializes the motor driver object in the `setup()` function

```
md.init();                          //initialize motor controller
```

- These lines read the analog value in the `loop()` function (should be before the `if` statement).

```
int Mval = analogRead(Apin);    //Motor voltage 0-1023
int M = 0;
```

- This line in the `if(true)` branch convert the motor voltage to command in the range of -400 to 400

```
M = map(Mval,0,1024,-400,400);//motor control value
```

- This line in the `else` branch changes the command to 0

```
M = 0;                          //then dont power motor
```

- This next line after the logic statements powers the motor. THIS LINE IS COMMENTED OUT IN THE Lab2_arduino.ino FILE. REMOVE THE DOUBLE FORWARD SLASHES AT THE START OF THE LINE SO IT LOOKS LIKE THE FOLLOWING, OTHERWISE THE MOTOR WILL NOT WORK:

```
md.setM1Speed(M);               //command voltage to motor
```

4.3.4 Connect Arduino to NI DAQ

- Using a BNC-to-Test clips, connect the A0 pin and GND of the Arduino to the AO0 on the NI DAQ.

4.3.5 Connect Motor to Arduino

- Using jumper wires connect the black and red leads of the motor to the M1A and M1B terminal on the motor shield
- Connect 5V and GND from the benchtop power supply to the blue VIN and GND screw terminals on the motor shield. On the benchtop power supplies, the rightmost red/black pair of connections are a dedicated 5V/GND supply. Use this pair of connections rather than either of the red/green/black groupings which are connected to variable-voltage channels.
- Connect all GNDs together to make a common GND

4.3.6 Test the GUI and Arduino. The motor should turn as you move the slider back and forth. It should change speed and direction.

Have a TA check your progress and sign you off. Be sure that your TA has signed you off on all parts of the lab.

Clean up your area and have the TA sign you off.

5. Post-Lab Exercises

1. Give examples where using this MATLAB GUI could be useful for your project.

2. What do the following two lines of code from the lab do? Be specific about each line.

```
rate = app.RateEditField.Value;
app.myDAQ.Rate = rate;
```

3. Give 3 examples of true analog instruments and explain how they are analog (e.g., a mercury thermometer is an analog instrument to measure temperature because the mercury level is a continuous variable). Note: temperature is a continuous variable, but that does not necessarily mean the instrument measuring it is analog, as there are digital thermometers!

4. Why does the Arduino analogRead() return a value between 0-1023 while the analogWrite() takes a value from 0-255?

5. Assuming a 5V max output, what is the Arduino C function and value (between 0 – 255) to output a 1.5V signal using an Arduino Uno. Show what the PWM signal would look like (e.g. voltage vs time). Provide proper labels and values in the plot.

6. Project Milestone 2

PM 2 is due during Lab 3, teams will prepare a Team Working Agreement and Preliminary Design Evaluations. Submit both on Canvas.

Team Working Agreement

Using the "Team Working Agreement.pdf" each team will create a team contract. Each member **must sign** and accept the terms of the agreement. Each team will present their agreement to the TA in Lab 3. Submit the agreement in PDF form.

Preliminary Design Evaluations

Each team must continue developing their concepts. Each team should have at least 3 preliminary robot designs to accomplish the project objectives. Teams will create a short 5-10-minute presentation in PowerPoint format evaluating each of these designs. You will submit the presentation with your team number in the filename on Canvas. Your presentation should discuss the following:

For each design

- Provide preliminary CAD assembly of design
- How the design accomplishes each task
- Discuss what hardware will be used
  - actuators, sensors, microcontrollers and boards, and mechanisms
- Discuss autonomous strategy
  - How will you transmit wireless info to robot?
  - break up of your autonomous tasks, brief walk through control for each task
- How does this design compare to others?
  - Create design metrics to evaluate each design

Conclusions

- From your design metrics make a decision on what design to move forward with. You will refine this concept and give a detailed description in PM 3.

Your TA will give you feedback on your designs. This feedback should be addressed in further Milestones.