Introducing LabVIEW and Arduino as Data Acquisition System Alternatives

Jackson Marsh, Christy Dunlap, Stephen Pierson, and Han Hu
Department of Mechanical Engineering, University of Arkansas, Fayetteville, AR 72701

Abstract

In the Department of Mechanical Engineering at the University of Arkansas, LabJack is used throughout the undergraduate curriculum for data acquisition (DAQ) in labs and projects. However, data acquisition techniques are not taught, and other DAQ systems are not used. When doing research or starting a capstone project, students are presented with the struggle of trying to cater LabJack to their needs or learning a new program, taking away from the time working on solving the problem at hand. LabVIEW and Arduino are two programs that allow the creation of DAQ systems and utilize similar techniques. However, the creation of DAQ systems using LabVIEW and the storage and analysis of data with Arduino is not covered. Critical to the creation of DAQ systems is the knowledge of the basics of data acquisition. Introducing these things alongside LabJack would allow students to make the best decision for their projects. Using these ideas, the benefits and drawbacks of LabVIEW and Arduino as DAQ systems are explored as well as the basics of data acquisition in this Student Paper.

Keywords

Data Acquisition, LabVIEW, Arduino, Student Paper

Introduction

Gathering data is an integral part of any project and at any level of research. Limited exposure to types of data acquisition (DAQ) systems can hinder students’ ability to acquire accurate and necessary data. Because of this, more DAQ systems should be introduced to students in the early stages of the curriculum. For example, mechanical engineering labs at the University of Arkansas use LabJack regularly. The skills learned in LabJack translate to other DAQ systems but the connection isn’t very apparent. In many capstone projects, it is easier to use Arduino for simple data collection. National Instrument’s LabVIEW is the industry standard for data acquisition, making it perfect for using when doing research. Because of this, it would be beneficial for students to perform work with Arduino and LabVIEW to help draw the connection between LabJack and other DAQ systems.

Arduino and LabVIEW provide great alternatives for data acquisition. Both allow flexibility and have their benefits and drawbacks. Before moving further, three things are of note. First, the usage of “Arduino” will refer to the Arduino code and hardware as a system. Second, LabVIEW is a coding environment where the user creates a “VI” or virtual instrument. However, many times “LabVIEW” will be used as the name of the code. LabVIEW is based on a language called G [1]. Finally, while these DAQ systems don’t require much programming knowledge, a basic understanding is needed. The main reason these systems are not integrated into the curriculum is the mechanical engineering program’s lack of coding instruction. The students are introduced to
the basics of MATLAB in Computer Methods for Mechanical Engineering and Arduino in General Engineering and Introduction to Mechatronics, but only some take the programing foundations elective that elevates coding abilities. LabJack requires no programming knowledge; students can just connect components and open the associated software, typically logUD. This works well for a lab setting and getting the necessary points of lab across. However, there should be a way to prepare students for setting up their systems when LabJack may not be an option. Thus, Arduino and LabVIEW will be discussed, including the benefits of the hardware and software along with the issues that may arise. Along with this all, the basics of data acquisition will be discussed so that data acquisition tools may be used to their greatest potential.

Data Acquisition Basics

All DAQ systems are broken down into three components as seen in Figure 1. This very basic Arduino systems shows the sensor, DAQ hardware, and DAQ software. The sensor’s role is to register that an event is taking place. Many times, this will be achieved using a voltage that is sent to the DAQ hardware, more of which is discussed in Arduino. The DAQ hardware usually consists of multiple parts like Figure 1. The chip on the red board boosts the signal that is then transferred via the yellow wire to an analog pin. The Arduino board can then read the signal on this pin. All data collected by DAQ hardware is then sent to the DAQ software. For Figure 1 this comes in the form of the Arduino IDE. The Arduino IDE uses a USB-B to create a connection to the Arduino board (part of the DAQ hardware).

Figure 1. All DAQ systems are based on three components: the sensor, the DAQ hardware, and the DAQ software. The example shown is a data acquisition system using an Arduino and electret condenser microphone.

There are two types of data acquisition: continuous and transient. Many times, data sampling is continuous, meaning that the sampling of data is constant in accordance with the sampling rate.
This is usually a result of a “while” loop in the code. This is extremely easy to set up and useful if everything about an event is to be known. However, this also uses up lots of storage space in the DAQ software. Transient data, as seen in Figure 2, only records data at certain points. For Figure 2, a limit of 1.5 was set so that everything above this limit would be recorded. That way if there is a long test going, but only certain quantities of data are needed, only the needed quantities will be collected. This saves on storage space and makes the data easier to analyze. One key to transient sampling is to always record the time. When using continuous sampling, so long as the sampling frequency and the start time are known, each data point can then be figured out at what time it occurred. This is not the case for transient data sampling, and therefore time must always be recorded.

**Figure 2.** Continuous and transient sampling of a bubble acoustics test, with transient data equal to and above 1.5 sound pressure being recorded. This transient data is a result of the bubble acoustics and the mechanism used to generate bubbles.

Another reason to record time is to check synchronization. Many times, when using multiple DAQ systems synchronization is needed to collect valuable results. There are three types of synchronization: manual, software, and hardware. Manual synchronization is when the user tries to start two DAQ systems as close together as possible. In analysis, the physical time is used to sync these. This can result in variable synchronization from test to test. Software synchronization is a result of the coupling of two software together. LabVIEW SDKs (discussed further in LabVIEW) allow for this pretty nicely. The synchronization is much better because when the software starts, signals are sent to both hardware systems. However, differences in signal strengths and cables limit its capabilities. Because of this, much like in manual synchronization, software synchronization can be variable between tests. Hardware synchronization is the best type. With hardware synchronization, the two DAQ hardware are coupled together. This way both receive the same signal at the same time. In this way, so long as the sampling rate of each is known, the samples should match up. An average synchronization of tests using high-speed imaging and acoustic sensing is shown in Figure 3.
Figure 3. Different types of synchronization were used when coupling high-speed imaging and bubble acoustics to track the release of the bubble. The results show the differences in the level of precision that different types of synchronization result in.

Figure 4. A sine wave of 2Hz is sampled at 3 different frequencies as an example of the Nyquist Frequency.

The sampling frequency is the most important data acquisition parameter due to the Nyquist Frequency. The Nyquist frequency as modeled in Figure 4 gives the minimum sampling frequency needed to accurately model a waveform, by showing a 2-hertz, 3-hertz, and 4-hertz sampling for the sine wave of 2-hertz. As shown in Figure 4 a 2-hertz sine wave when sampled at 2-hertz is a flat line. This would be an inaccurate representation of the wave. The Nyquist frequency says that the maximum frequency obtainable is half that of the sampling frequency. Although the 3-hertz line appears to work, it is not greater than or equal to the Nyquist frequency.
of the graphed wave. This can be explained because the geometric definition says that the frequency need follow the same trajectory as the wave it is trying to match. Whenever the wave is rising, the data should be rising and when the wave falls, the data should follow suit. Although for much of the graph the 3-hertz seems to do this, there are times at which the graph is falling but the data is rising. This can be seen from around two-thirds of a second to three-quarters of a second in Figure 4. Due to all of this, the 4-hertz sampling frequency is the minimum frequency needed to accurately acquire data and model a 2-hertz sine wave.

**LabVIEW**

LabVIEW is a coding environment where code as seen in Figure 5 can be created. LabVIEW is split into two parts—the front panel and the block diagram. The block diagram is where the coding takes place (Figure 5a. and 5b.). The blocks of the code are tied together using wires. The code runs left to right through the wires, transferring the information. Much like any other form of code, it is important to stay organized when working in LabVIEW. Otherwise, LabVIEW code can easily start to look like a mess of lines and dialog boxes. This can make it difficult to debug. Because of this, it is beneficial to keep everything spaced out and labeled when learning LabVIEW. If done well, LabVIEW code flows with the line of thought of the creator and is easy to manipulate. Basics of coding are still necessary when working with LabVIEW. For example, in Figure 5a. all the code is contained in a “while” loop. When the program starts running, it immediately enters this “while” loop; throughout the entire run of the program, data is being collected and sent to both the graph for real-time viewing and a file for data storage.

As seen in Figure 5, LabVIEW has different ways of coding to get to the same output. The code shown is used to sample hydrophones to detect bubble dynamics. The time and hydrophone signal are both recorded. Both Figures 5a. and 5b. achieve the same output. Both output to a
These DAQmx functions are built into LabVIEW and work with NI’s data acquisition modules. This is one of the drawbacks of LabVIEW. While it is extremely useful, it is of industry standard and the prices reflect this [2]. Between the software costs and the data acquisition modules, it can be expensive to get started with LabVIEW. Because of this, the uses for LabVIEW tend to lend themselves toward research where grants cover the cost. However, LabVIEW does not have to be used exclusively with NI products [1]. LabVIEW has a great community that creates and publishes VIs for many different uses. Alongside this, many companies that have DAQ hardware also have LabVIEW SDKs or provide VIs. With all of this, the Academic License for LabVIEW comes with many add-ons [3] and code integration of Python, C/C++, .NET, and MATLAB as is standard with LabVIEW [2].

**Arduino**

```c
//Create variables
float rawSound; //Will store the sound as a voltage
int t; //Will store the raw sound value
int t_old; //Create a time variable
int t_id; //Store time variable
int dt; //Change in time

void setup() {
  Serial.begin(9600); //Serial.println("Hi, Volts");
}

void loop() {
  //Compute Time
  t = millis(); //Milliseconds Since Start
  dt = t - t_old; //Find dt

  //Compute Sound
  rawSound = analogRead(A0); //readSignal[analog reads the data from an analog input]
  floatSound = rawSound * 0.0049; //Multiply by number of volts per unit

  //Output Data
  Serial.print(t);
  Serial.print(",");
  Serial.println(floatSound);
  //Change time
  t_old = t; //Make Time Old the t
  //Delay Next Read
  delay(10);
}
```

*Figure 6.* Example of Arduino code that collects time and sound. It then writes this data to the serial monitor.

Arduinos are great ways to start with data acquisition. The ability to make every Arduino unique to the problem at hand, at an affordable price, makes them great for prototyping. Arduino’s programming interface is free to use, and the hardware used for data acquisition is cheap. This
also means that troubleshooting with Arduino is quick and easy. **Figure 6** shows an example piece of Arduino code. Arduino uses C++ as its base coding language. This scripting language allows the transition from MATLAB and other coding languages easier. However, the simplicity of Arduino leads to some problems quickly. Arduinos are limited to 5-volt pins, meaning there is a threshold at which data cannot be read. This needs to be kept in mind when building the circuits and coding the programs. Also, having multiple data acquisition instruments set up requires having multiple Arduinos set up as well. While this is not an immediate challenge, it may present itself when logging data.

![Figure 6](image)

**Figure 6** shows an example piece of Arduino code. Arduino uses C++ as its base coding language. This scripting language allows the transition from MATLAB and other coding languages easier. However, the simplicity of Arduino leads to some problems quickly. Arduinos are limited to 5-volt pins, meaning there is a threshold at which data cannot be read. This needs to be kept in mind when building the circuits and coding the programs. Also, having multiple data acquisition instruments set up requires having multiple Arduinos set up as well. While this is not an immediate challenge, it may present itself when logging data.

![CoolTerm](image)

**Figure 7.** a. CoolTerm program that allows recording of data from Arduino. b. CoolTerm’s serial monitor shows exactly what would appear on the serial monitor within the Arduino program. c. 14 data points from the sample shown. d. Graphical representation of the data collected using the Arduino setup seen in **Figure 1**.

There are two options when using Arduino to view data: the serial monitor and the serial plotter. The serial plotter takes the data and plots it in real-time, much like the graphs within LabVIEW. However, the data cannot be written anywhere if the serial plotter is being used. Because of this, the serial plotter makes a useful tool for calibration, but the serial monitor is much better for recording acquired data. The serial monitor is not without problems of its own. Seen in **Figure 7** is CoolTerm. CoolTerm is a free program that connects to the port of the Arduino and records the collected data [4]. Recording that data is shown in **Figure 7a**, and allows easy saving of and storage of data in a text file.

**Figure 7c.** shows another problem that comes with using Arduino, a timing issue. The left column of the data in **Figure 7c.** is time data in milliseconds, since the start of data acquisition. As shown in the code of **Figure 6**, there is supposed to be a 10-millisecond delay. Using the
Nyquist Frequency would limit the data processing ability to 50-hertz. Yet, as time progresses throughout the test, the delay does not hold, and therefore, neither does the Nyquist frequency. Throughout the entire test, as shown in Figure 7d, there are about 6 seconds worth of data missing. This is because the Arduino must record the data and transmit it back to the computer. This results in timing issues that mess with sampling rates [5]. However, this does not mean the data is inaccurate. The time is still being accurately recorded, but because of the irregularity between points, it is more difficult to do analysis with the data.

In Figure 7d, the y-axis is in volts, while the normal measurement of sound is either in pressure or decibels. When doing data acquisition, especially using Arduino, it is important to keep in mind that the instruments transmit data using voltages. While programs like LabVIEW incorporate the conversions to output the correct units, if using Arduino, the user must code the conversions. In the code as seen in Figure 6, there is a conversion to get the data to volts. That is because Arduino outputs a resistance value between 0 and 1023 [6]. Because of this, to achieve numerically useful data from Arduino, code must be written to convert it to the correct units; usually, this first step is converting to volts.

**Conclusion**

<table>
<thead>
<tr>
<th>Arduino</th>
<th>LabVIEW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong>: Cheap and easy to learn</td>
<td><strong>Pro</strong>: Multiple ways to code same thing</td>
</tr>
<tr>
<td><strong>Pro</strong>: Maneuverable and easily adaptable</td>
<td><strong>Pro</strong>: Industry Standard</td>
</tr>
<tr>
<td><strong>Con</strong>: Synchronization and time keeping</td>
<td><strong>Con</strong>: Expensive hardware and software</td>
</tr>
<tr>
<td><strong>Con</strong>: Limited range of data acquisition</td>
<td><strong>Con</strong>: Exponential ramp up of code difficulty</td>
</tr>
</tbody>
</table>

Table 1. A summary of the pros and cons of both Arduino and LabVIEW are shown.

Exploring the options of Arduino and LabVIEW with students, as summarized in Table 1, allows them to make better choices when planning projects. In the Mechanical Engineering Department of the University of Arkansas, LabVIEW is only seen by students during Lab 1. The experience uses LabVIEW as a control; the students use code given to them by the instructor. An addition to this experience showing data collection and allowing students to employ either the simple VIs or the DAQmx VIs would give students a launching point for using LabVIEW. This would set up students to utilize LabVIEW’s range of customizable code and accuracy in data acquisition. Arduino is currently used in the Intro to Mechatronics course, but, like LabVIEW, Arduino is used for control. Arduino is used to acquire real-time data to actuate a motor. Using a program alongside Arduino to save this acquired data and harnessing Arduino’s maneuverability and adaptability would allow students to create better systems. Most importantly, incorporating this within a lab while also teaching data acquisition basics would help solidify all of the necessary skills. Together all of this should allow for any data acquisition needs within research or a capstone project to be easier to obtain.

**Acknowledgments**

This work was supported by the Arkansas EPSCoR Data Analytics that are Robust & Trusted (DART) through seed grant number 22-EPS4-0028, under NSF grant number OIA-1946391, Chancellor’s Funds for Commercialization, and Honors College Research Team Grant at the University of Arkansas.
References


Jackson Marsh

Jackson Marsh is a mechanical engineering undergraduate concentrating in aerospace and minoring in mathematics at the University of Arkansas. From Fall 2020 to Spring 2022 he has been named to the Chancellor’s List. Jackson has received the Arkansas Space Grant Consortium Student Intensive Training Grant and twice received the Honors College Research Grant. He plans to pursue a master’s degree after graduation in May 2024.

Christy Dunlap

Christy Dunlap is a Ph.D. student in the Department of Mechanical Engineering at the University of Arkansas. Christy obtained her B.S. in Mechanical Engineering and B.S. in Mathematics with Applied Concentration from the University of Arkansas in 2021. Her research covers system design, DNA sequencing, thermal data analytics, and multimodal fusion. Christy is proficient in programming using Python, MATLAB, C++, and Arduino, machine learning packages including TensorFlow and scikit-learn, operating system and software maintenance on Linux systems.

Stephen Pierson

Stephen Pierson is a University of Arkansas Honors College Fellow, Arkansas Governor’s Distinguished Scholar, and undergraduate mechanical engineering student at the University of Arkansas. His current research interests lie in the applications of materials science and advanced manufacturing techniques.
Han Hu

Han Hu is an Assistant Professor in the Department of Mechanical Engineering at the University of Arkansas. He leads the Nano Energy and Data-Driven Discovery (NED₃) Laboratory and his research interests cover experimental characterization and multi-scale modeling of two-phase heat transfer enhancement on micro-/nano-structured surfaces, immersion cooling of power electronics, diffusion kinetics in high-entropy alloys, and multimodal data fusion.