Data Acquisition Using LabVIEW and MATLAB for Mechanical Engineering Laboratories

Trevor Harris, Stephen Pierson, Hari Pandey, and Han Hu
Department of Mechanical Engineering, University of Arkansas, Fayetteville, AR 72701

Abstract

Data acquisition is an integral part of research and has many uses within the industry. Having a basic understanding of data acquisition systems and the capabilities or where they could be applied will create more opportunities for engineering students when they are ready for research or industry. To gain this understanding through engineering education, students could take a module in a lab class where they learn to implement a data acquisition system using LabVIEW and process the data using MATLAB. The purpose of this student paper is to show the importance of data acquisition in engineering education, illustrate where data acquisition can be implemented into the current engineering curriculum, and display some of its applications by demonstrating the process of collecting and post-processing temperature, flow rate, and pressure data in an example of heat/mass transfer experiment.

Keywords

Data Acquisition, LabVIEW, MATLAB, Undergraduate Student Paper

Introduction

Data acquisition (DAQ) is an integral part of many different research projects and has many uses throughout the industry which is why there is a need in the engineering curriculum at the University of Arkansas and many other universities for students to have a thorough understanding of data acquisition processes and its applications. LabJack and LJLogUD are the main hardware and software, respectively, used for undergraduate mechanical engineering students at the University of Arkansas [1]. To record data, students must write the values they are seeing down or take a picture with a phone or other device. This is not practical for large amounts of data and there are many better options, most of which are capable of recording data and higher rates of data acquisition. LabVIEW is briefly touched on in Lab 1 where students learn about the noise cancellation possibilities of LabVIEW, the interaction with other devices such as LabJack, with LabVIEW, and the basics of how to use the software. This is a foundation for students to build upon, but with LabVIEW being the standard in the industry for data acquisition, it is essential to better prepare students by giving them a more thorough education on how to perform data acquisition with LabVIEW [2].

To implement data acquisition into engineering education, a module could be added to Lab 1 or a similar class, where students get an introduction to data acquisition with LabVIEW, how to process the data using MATLAB or another language, and where it can be applied. This module will be placed after students have already been introduced to LabVIEW in a previous module. After students learn the basics of LabVIEW in their first module with the program, they will be

ready to move forward by learning data acquisition. To start the module students should first be introduced to the DAQ cards. First, the specs and capabilities of different DAQ cards can be taught. Once students learn the specs, they can move on to learning about other core data acquisition principles such as sampling rate, aliasing, frequency vs. amplitude signals, etc. Then, students will be prepared to learn how to connect sensors to the DAQ cards. Students can start with simple sensors like thermocouples, which take a voltage reading and convert the reading to temperature. Then they can learn to record the measurements using LabVIEW. After students learn some basic sensors, they will be able to move on to more complicated sensors such as pressure transducers which will require external wiring with power supplies and ground connections. After gaining a basic understanding of connecting sensors and recording data in LabVIEW, students will be able to move on to the next part of the module which will be interpreting the data. Students will learn to use a programming platform, such as MATLAB, to write a program that will take the recorded data and convert it into synchronized figures. Students at the University of Arkansas get an introduction to MATLAB in Computer Methods, which comes before or is concurrently taught with Lab 1, so they should be able to quickly move into processing data. They will first learn the steps to take to code the program and then be tasked with writing the program to interpret the data and display the final synchronized figures from the experiment.

There are many practical examples of data acquisition using LabVIEW in both research and industry applications [2]. In this paper, an example of acquiring temperature, flow rate, and pressure data for a pumped liquid heater, shown in **Figure 1**, will be discussed. Furthermore, the process of connecting the sensors, coding the sensors for data acquisition in LabVIEW, processing the data in MATLAB, and displaying the data in synchronized figures will also be covered.

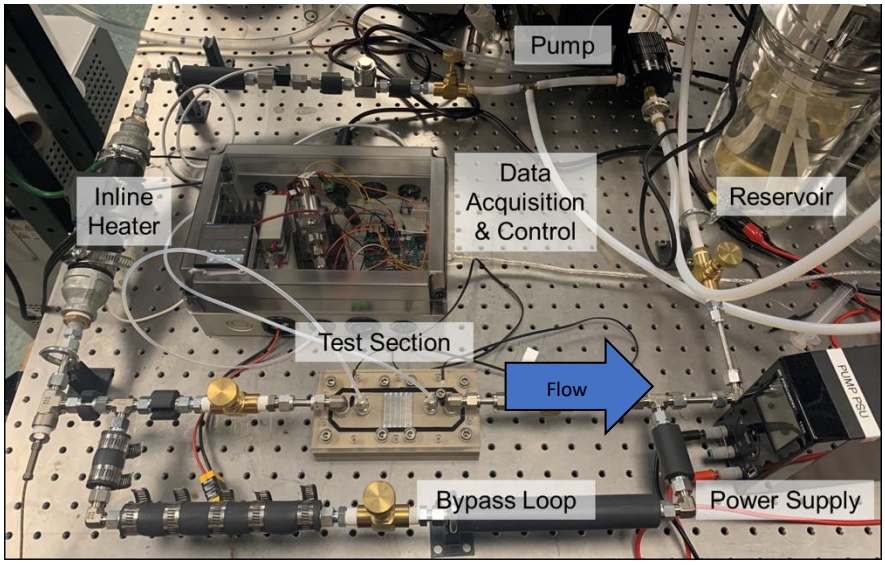


Figure 1. Image of the pumped liquid heater showing the test section and data acquisition box.

Setting up and Wiring the Sensors to the DAQ Modules

This example will utilize a NI 9219 DAQ card and a NI 9239 DAQ card. The different sensors used in this example are two type T thermocouples, two thermistors (10K Precision Epoxy Thermistor – 3950 NTC), one flow rate sensor (BV1000TRN025B), and one differential pressure transducer (PX2300-1BDI).

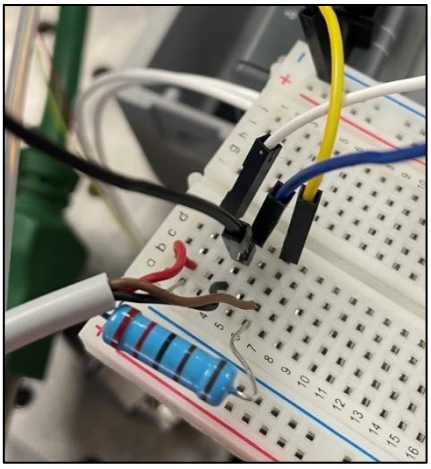
The NI 9219 card can read a wide variety of signals including voltage, current, and resistance, and most importantly, for thermistors, can produce an excitation voltage [3]. Thermistors require an excitation voltage to use because the sensor must be capable of reading the resistance from the thermistor and there can be no resistance measurement without a voltage differential. For this example, the thermistors are the only sensors that are connected to the 9219, and the remaining four, 2 type K thermocouples, flow rate sensor, and differential pressure transducer are connected to the NI 9239 card. The NI 9219 card has four different channels with each channel having six different pins to connect the sensors to [3]. The thermistor needs to be connected to the two pins where the NI 9219 card can produce the excitation and then read the change in resistance across the thermistor.

The thermistor will need to be connected to the third and fifth pin on one of the channels of the NI 9219 card and the end connections of the thermistor can be placed in either the third or fifth pin because they are interchangeable. Now the thermistor is connected to the excitation-capable pins of the 9219 and is ready to be coded in LabVIEW.

Next, the NI 9239 card will need to be connected to the four other sensors which are two thermocouples, one flow rate sensor, and one differential pressure transducer. Thermocouples will be the simplest to connect. Each thermocouple will have two different colored wires with one being the positive end and one being the negative end. The 9239 also has 4 channels but each channel only has two pins with one being a positive connection and one side being a negative connection [4]. The positive and negative analog input signal connection use screw terminals for the connection. Thermocouples will be connected by loosening the screw terminal and inserting the thermocouple wire in its correct connection pin, then tightening the terminal. Now the thermocouples are ready to be coded in LabVIEW.

Next, the flow rate sensor will need to be connected. The pulse-type flow rate sensor works by sending a square wave signal in the form of voltage to the 9239. Once the signal is received the frequency can be read and converted to the desired flow rate units such as liters per minute. The flow rate sensor will need a power source, ground connection, and a resistor before the connection process can start. A 5V power supply from an Arduino, a ground connection on the Arduino, and a 2.2K Ohm resistor were used for the flow rate sensor listed above. **Figure 2** shows the wire connections of the flow rate sensor [5]. After the sensor is connected to the power supply, ground, and resistor, it is ready to be connected to LabVIEW to be coded.

a.



b.

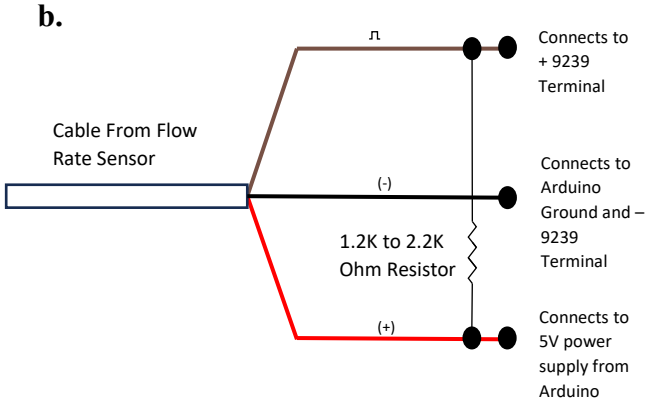


Figure 2. The wiring setup for the flow rate sensor. **a.** The white wire is connected to the 5V Arduino power supply. The black wire is connected to the Arduino ground. The blue wire is going to the negative port on a NI 9239 channel. The yellow wire is connected to the positive port on the NI 9239. **b.** Wiring schematic for flow rate sensor [5].

The final sensor to be connected to the NI 9239 is the differential pressure transducer. The differential pressure transducer works by connecting to the different sides of a test section then the pressure drop is measured and sent to the 9239 in the form of voltage. Once the signal is received, the voltage reading is converted to psi, pascal, or the desired pressure unit. Before connecting the sensor to the 9239, the pressure transducer must first be connected to a 9V-30V power supply and a loop resistance depending on the voltage from the power source[6]. After the sensor is wired, the sensor will be ready to be coded in LabVIEW.

Creating Code in LabVIEW for Data Acquisition

LabVIEW presents a user-friendly environment that does not require a heavy background in programming. The software uses a drag-and-drop block diagram environment with wire connections between the block functions to present the code. After every block function is inserted into the block diagram, they are all wired together and then the block diagram code is ready to run. Currently, LabVIEW is the standard for data acquisition and is prevalent in research and many other purposes in the industry [2]. LabVIEW consists of two working windows called the Front Panel and the Block Diagram. The Front Panel is where the user can interact with the code that is created in the Block Diagram. The Block Diagram is where all the different block functions can interact with each other and connect via wires. See the Block Diagram in **Figure 3** and Front Panel in **Figure 4**.

The code in **Figure 3** relies on the preset DAQ Assistant Function UI in LabVIEW. The DAQ Assistant function allows a user to save a lot of time by having all the necessary functions to control the different NI DAQ cards in one window. Intuitive steps can be followed in the DAQ Assistant properties window and data can be recorded in a few simple steps. To start right click in the block diagram window to open the function selection tab and then search DAQ Assistant and drag the function into the block diagram window. Then the properties panel will open for the DAQ Assistant, and the different sensors can be added to the DAQ Assistant. To add different

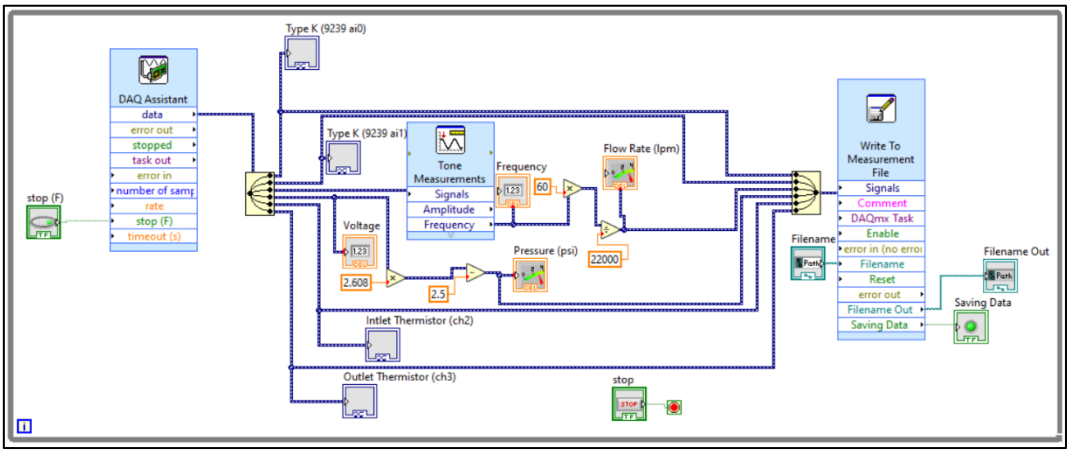


Figure 3. The block diagram LabVIEW code for six synchronous sensors, two thermocouples, two thermistors, a flow rate sensor, and a differential pressure transducer that are written to a .lvm file using the Write to Measurement File function.

sensors to the DAQ Assistant click the plus sign and then select the type of sensor that is being used. Resistance, thermocouple, and voltage sensing are used in this example.

To add the thermocouple, a thermocouple sensor needs to be added to the DAQ Assistant. After the thermocouple is added, the cold junction compensation or CJC value needs to be selected. Some DAQ modules will have a built-in feature to select this value but a constant value of 25 will be used for this example for both thermocouples.

A thermistors can be added as a resistance sensor. To do this, click the plus sign; then click

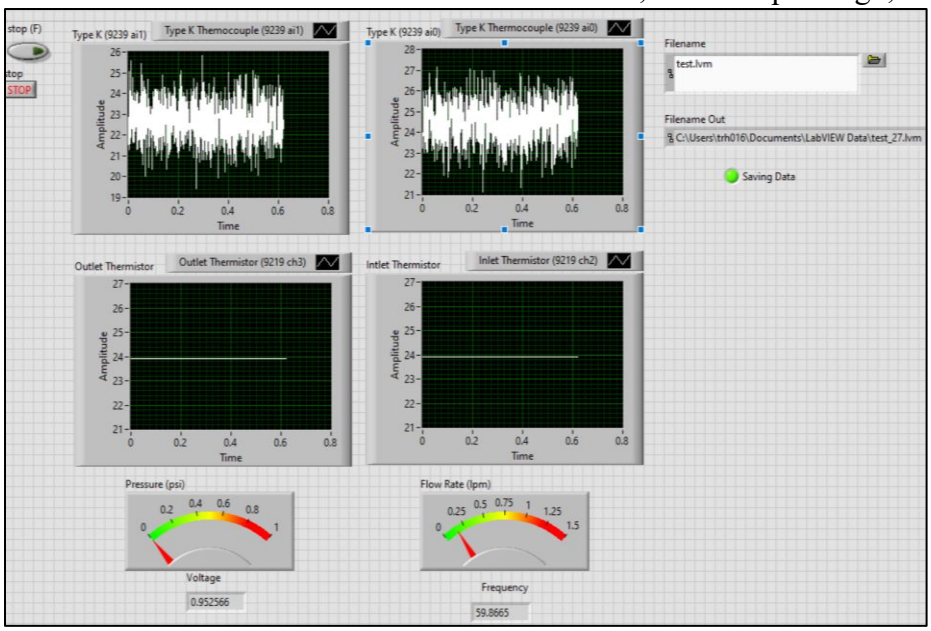


Figure 4. The LabVIEW Front Panel, where users can interact with code and interpret the data acquired in real time.

analog input and select the channel the sensor is in; and finally, select resistance. Now that the

sensor is added, the thermistor can be formatted. Each thermistor will come with a lookup table where the LabVIEW code will look up the table conversion to go from the measured unit, Ohms, to the desired unit, temperature, which is normally Celsius. To insert the custom scaling, first, save the lookup table as a .txt file. After the file is saved, click the custom scaling wrench and insert a new custom table. Once the table is inserted, the code will reference the lookup table before displaying the temperature the thermistor is reading. This process can be repeated for as many thermistors as needed.

To set up the flow meter, add an analog voltage sensor to the DAQ Assistant. Next, select the voltage range and insert the samples to read and rate (Hz) numbers. This example used 10k Hz and 1k samples to read. After that, select OK. The majority of coding for this sensor is done in the block diagram, unlike the temperature sensors where most of the work is done in the DAQ Assistant. After the DAQ Assistant is set up, the frequency of the square wave produced by the flow rate sensor needs to be read. To do this, right-click in the block diagram window, look up tone measurements, and drag it into the block diagram. Then select amplitude and frequency and click OK to add the Tone Measurements function to the block diagram. Next, connect the data tab on the DAQ Assistant to the signals tab on the Tone Measurement function. Now take the frequency and multiply it by 60 to get the pulses per minute of the sensor. Then look up the pulses per minute from the manufacturer's handbook and divide the pulses per minute by that number. See **Figure 3** for the flow meter sensor LabVIEW code. In this example, the pulses per liter were 22,000. Now that the pulses are converted, the sensor is ready.

The final sensor in this example is the differential pressure transducer. Add an analog voltage sensor to the DAQ Assistant. Set the voltage range, rate (Hz), and samples to read to the desired values. After the values are set, the voltage needs to be scaled to the desired unit of pressure which for this example was psi. To go from the initial reading of voltage to pressure requires a linear transformation. For this example, when the flow loop was off and therefore the pressure drop was zero the voltage hovered around 0.96 volts which also corresponded with a 0 psi drop and 4 mA. Then the range of amperage was known from the manufacturer specs which was 4 mA – 20 mA which meant that the voltage needed to be multiplied by five to reach the desired range. So that meant that 4.8 volts corresponded with a 10 psi drop (psi range is 0psi-10psi from manufacturer specs) and 20 mA. Now that the endpoints of pressure were found, (0.96V, 0 psi) and (4.8V, 10 psi), they could be used to create the point-slope equation which will transform the voltage to pressure. See **Figure 3** for the LabVIEW wiring of the conversion.

Now that all of the sensors are coded with LabVIEW, the data from the sensors needs to be recorded. To record the data, a Write to Measurement Function needs to be inserted. To insert, right-click in the block diagram window and look up Write to Measurement, then drag the function block into the block diagram. Format the function to the desired needs. This example writes the data to a .lvm file. Once the function is set up, wire all of the sensors to the Write to Measurement function and data will be ready to be recorded. Adding controls and indicators is recommended to see when data is being recorded, the location of the file, and the name of the file. See **Figure 3** for the wired connections with the sensors and the Write to Measurement function.

Data Processing Using MATLAB

MATLAB will be the software used to display the data initially recorded in a .lvm file to the final synchronized figures.

```
%get .lvm file from File Explorer and change into a .txt without headers that MatLab can work with

filename = uigetfile({'*.lvm'; '*.txt'});                %retrives file from UI
fidi=fopen(filename,'r');                        %opens file
filename = erase(filename,'_noHeaders');         %erase file suffixes
filename = erase(filename,'.lvm');
filename = erase(filename,'.txt');
newfile = strcat(filename,'_noHeaders.txt');     %rename the the new file to filename_noheaders.txt
fido=fopen(newfile,'w');                         %open the new file
headerlines=23;                                  %specify the number of header lines
for i=1:headerlines,fgetl(fidi);end              %copy the original file minus the header to the new file
buf=fread(fidi,'*char');
fwrite(fido,buf);
fclose('all');
```

Figure 5. MATLAB code for converting the file to .txt file with no headers.

The first step is to convert the data from the original .lvm file to a .txt file and to remove the headers so the data can be transferred into an array (**Figure 5**). To convert the .lvm file, the file first needs to be retrieved from a folder where the data is stored with the same program that is converting the data. Once the file is retrieved, the file will be opened, and the file type will be converted to a .txt file. The .lvm file comes with headers that include information such as the time when the data started to be recorded, sample rate, number of channels, etc. For the data to be put into an array these headers must be removed so only the data recorded is left behind. Next, the amount of header lines needs to be figured out by looking at the original .lvm file and then erasing those header lines from the file. After the file is formatted correctly to where the data can be interpreted, the data is ready to be imported into an array.

The first column is time for most .lvm files. The rest of the columns will be each of the sensors used in the program in the order they are initialized in DAQ Assistant. This example uses six different sensors meaning there will be seven columns of data. Import each sensor to the array as seen in **Figure 6** and assign each column with the sensors that are being utilized.

```

experiment_file = importdata(newfile);          %import data into array

time = experiment_file(:,1);                    %import the columns from the file into corresponding arrays
t_inlineHeater = experiment_file(:,2);
t_testSectionHeater = experiment_file(:,3);
dP = experiment_file(:,4);
t_i = experiment_file(:,5);
t_o = experiment_file(:,6);
flow_signal = experiment_file(:,7);
```

Figure 6. MATLAB code for converting the raw data into an array that MATLAB can interpret.

As noted above, the flow rate sensor works by sending a square wave signal to the 9239 DAQ card, and then this signal is interpreted by LabVIEW. The original LabVIEW code utilized the Tone Measurement function block to measure the frequency of the square wave, and this took the average frequency over a tenth of a second and then recorded the data every tenth of a second meaning there was a sample rate of 10 Hz which was slower than needed. The Tone

Measurement also had a problem with measuring the frequency of the noise when the flow rate was zero meaning that the sensor was saying that there was always fluid flowing through the loop even when the pump for the loop was off. Both problems can be solved with MATLAB. Using MATLAB, the sample rate can be increased to 300 Hz and the sensor will read 0 lpm while the flow loop is off.

As shown in **Figure 7**, the first step is to define the variables used in the for loop. After everything is defined the for loop must run for the length of the signal that was recorded. This is done by referencing the column of the flow rate data from the array and making the for loop run for the length of the column. After the length of the for loop is defined the next step is to count the number of pulses and to find the frequency of the pulses. Because the voltage of the square wave is going from 0V to 5V the code counts when there is a pulse where the voltage jumps from 0V to 5V. Once this pulse is recorded, the code will record the pulses until the counter reaches counter max which is the number of pulses that are averaged together. Then the frequency is found by taking the reciprocal of the period. Once the frequency is found, it is then divided by the pulses per liter which will give the result in liters per minute. The for loop also recognizes if there has been no pulse for a set amount of time and then will record the flow rate as 0 lpm. After the flow rate data has been recorded, it is then put back into the array where the rest of the data is so it can be used in the final plots of synchronized data.

```
%reconstruct the flow rate data from the flow signal data stream

Q = flow_signal;                                        %define the flow rate array
sampleRate = time(2)                                    %get the sample rate from the time column
lastPulse = 1;                                          %stores the position of the last "pulse" in flow_signal
lastFlowRate = 0;                                       %stores the previous flow rate
counter = 0;
counterMax = 2;                                        %sets the number of turbine pulses to average together
firstFlowRate = true;
noFlowThreshold = .2;                                   %set the amount of time (s) without a pulse that a flow rate can be considered 0

for i = 2:length(flow_signal)                           %iterate over the length of flow_signal
    if (flow_signal(i) >= 2.5) && (flow_signal(i-1) < 2.5)      %pulse detected?
        counter = counter + 1;
        if (counter >= counterMax)
            avg_period = sampleRate*(i-lastPulse)/counter;      %gets average period between pulses over the span of counter in (s)
            lastFlowRate = 1/(avg_period)*60/22000;             %converts to flow rate (LPM) and stores it
            lastPulse = i;                                      %reset lastPulse and counter
            counter = 0;
            if firstFlowRate                                    %write over the first Q(j) values that come before the first pulse is detected
                firstFlowRate = false;
                for j = 1:i
                    Q(j) = lastFlowRate;
                end
            end
        end
    end
    if (i-lastPulse)*sampleRate > noFlowThreshold
        lastFlowRate = 0;                                   %sets flow rate to 0 if no pulse has been detected in time threshold
    end
    Q(i) = lastFlowRate;                                    %writes latest flow rate value to Q(i)
end

```

Figure 7. MATLAB code for reconstructing the flow rate signal and finding the pulses of the meter so the final measurement in liters per minute can be found.

The final step with MATLAB is to plot the synchronized data. Define the parts of the plot that are needed such as labels, legends, etc., and pull the data from the column of the array where the data is stored to plot the final figure. See **Figure 8** for an example of how to code the plots.

```
%Temperature Plot
figure
plot(time,t_inlineHeater,'Color','black','LineWidth',1);
hold on
plot(time,t_testSectionHeater,'Color','red','LineWidth',1);
hold on
plot(time,t_i,'Color','green','LineWidth',1);
hold on
plot(time,t_o,'Color','blue','LineWidth',1);
grid on

% xlim([0 2400])
ylim([90 105])
xlabel('Time, \itt\rm (s)')
ylabel('Temperature, \itT\rm (^\circC)')
set(gca,'FontName','Arial')
set(gca,'FontSize',16)
legend('Inline Heater','Test Section Heater','Inlet Temp', 'Outlet Temp', 'Location', 'northeast','Fontsize',14);
% legend('Inlet Temp', 'Outlet Temp', 'Location', 'southeast','Fontsize',14);
```

```
%Flow Plot

figure
% yyaxis left
plot(time,Q,'Color','red','LineWidth',1);
ylim([-1 3])
ylabel('Flow Rate, \itQ\rm (L/min)')
xlabel('Time, \itt\rm (s)')
set(gca,'FontName','Arial')
set(gca,'FontSize',16)
legend('Flow Rate', 'Location', 'northeast', 'Fontsize',14);

%Pressure Drop Plot

figure
% yyaxis right
plot(time,dP,'Color','black','LineWidth',1);
ylim([-2 10])
ylabel('Pressure Drop, \itdP\rm (kPa)')
xlabel('Time, \itt\rm (s)')
set(gca,'FontName','Arial')
set(gca,'FontSize',16)
legend('Pressure Drop', 'Location', 'northeast', 'Fontsize',14);
```

Figure 8. MATLAB code for plotting the synchronized figures. This code can be configured to format the figures and arrange the data in whichever way is needed.

Figure 9 shows the final synchronized MATLAB plots. The data comes from a test experiment where the LabVIEW program above was used, and data was recorded to a .lvm file while the pumped liquid heater was flowing. Different parameters in the pumped liquid heater were manipulated to change different variables such as flow rate, pressure, and temperature which are the reasons for the fluctuations in the figures. The noise in the pressure drop figure most likely came from bubbles accidentally entering the pressure tubes connected to the test section. Temperature ranges vary for different tests, but normally max out around 100 degrees Celsius. If temperatures are lower, the axis range can be adjusted in MATLAB and the same goes for any of the other sensor figures. **Figure 9** utilizes all the sensors listed above. The thermistors are used in the inlet and outlet temp figure and the other two temperature figures use the type K thermocouples.

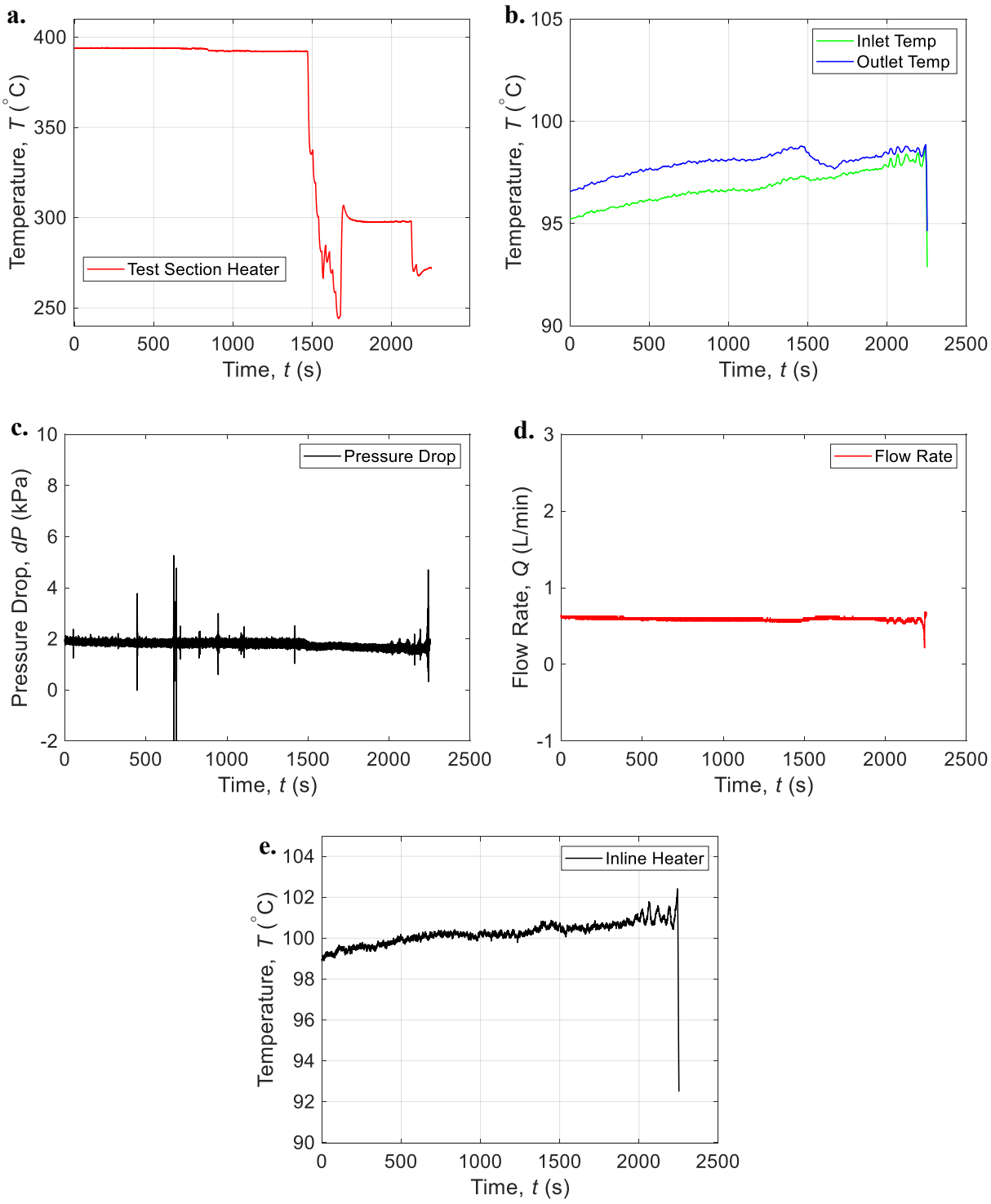


Figure 9. Displays the synchronous figures showing temperature, flow rate, and pressure drop. **a.** Test section heater. **b.** Inlet and outlet test section temperature comparison. **c.** Pressure drop across test section **d.** Flow rate across flow loop. **e.** Inline heater temperature.

Conclusion

Data acquisition should be a part of all engineering students' education. Through a short module in Lab 1 or a similar class, students will learn the basics of many data acquisition techniques which they will then be able to apply in their future research opportunities and jobs in industry. The example shown in this paper offers a small glimpse of the many possibilities that LabVIEW skills make students capable of. With a proper education in data acquisition, future engineering students will be better prepared to enter the working world and meet the challenges that they will face in testing systems for their purposes and needs.

Acknowledgments

This work was supported in part by NSF (Grant No. OIA-1946391) through the Arkansas NSF EPSCoR DART SURE Program (Grant No. 23-EPS4-0041) and University of Arkansas Chancellor's GAP Fund and Chancellor's Fund for Collaboration. S. Pierson acknowledges support from the Arkansas Department of Higher Education through a 2023 Student Undergraduate Research Fellowship and the University of Arkansas Honors College through an Honors College Research Grant.

References

[1] J. Marsh, C. Dunlap, Stephen Pierson, and Han Hu. (2022). Introducing LabVIEW and Arduino as Data Acquisition System Alternatives. Presented at 2022 ASEE Midwest Section Conference. [Online]. Available: https://bpb-us-e1.wpmucdn.com/wordpressua.uark.edu/dist/a/708/files/2022/09/Marsh-ASEE-2022-Midwest-Section-Conference-Paper3279.pdf

[2] Aida Skamo and Dejan Jokic. (2023). Advantages of early adoption of LabVIEW as industry-standard software in academia. Presented at 2023 12th Mediterranean Conference on Embedded Computing (MECO). [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10155017

[3] National Instruments, "NI-9219 Getting Started," NI, https://www.ni.com/docs/en-US/bundle/ni-9219-getting-started/page/overview.html (accessed Jun. 28, 2023).

[4] National Instruments, "NI-9239 Getting Started," NI, https://www.ni.com/docs/en-US/bundle/ni-9239-getting-started/page/overview.html (accessed Jun. 28, 2023).

[5] Omega Engineering, "Omega Engineering, Vision Turbine Flow Meters | Omega Engineering," https://www.omega.com/en-us/flow-instruments/flow-meters/turbine-flow-meters/bv1000-2000-3000-series/p/BV1000TRN025B (accessed Jul. 18, 2023).

[6] Omega. *User's Guide MODEL PX2300 Differential Pressure Transmitters*. (1999). [Online]. Available: https://assets.omega.com/manuals/test-and-measurement-equipment/pressure/pressure-transducers/M3244.pdf

Trevor Harris

Trevor Harris is an honors student pursuing a degree in Mechanical Engineering at the University of Arkansas. He is passionate about technology and learning how things work. His current research is focused on multimodal sensing and signal processing of liquid cooling loops for electronic devices.

Stephen Pierson

Stephen Pierson is an Honors College Fellow, mechanical engineering student, and undergraduate research assistant at the University of Arkansas. Stephen is a 2023 Goldwater Scholar and 2023 Arkansas SURF fellow. His current research interests lie in the two-phase cooling and advanced manufacturing techniques.

Hari Pandey

Hari is a Ph.D. Candidate in the Department of Mechanical Engineering at the University of Arkansas. His research focus includes prediction and mitigation of boiling crisis using acoustic emission and high-speed imaging, improving boiling heat transfer using wicking structures, and immersion cooling of power electronics.

Han Hu

Han Hu is an Assistant Professor in the Department of Mechanical Engineering at the University of Arkansas. He received his B.S. from the University of Science and Technology of China in 2011 and Ph.D. from Drexel University in 2016. His research is focused on electronics cooling, thermal diagnostics, advanced manufacturing, signal processing, and engineering education.