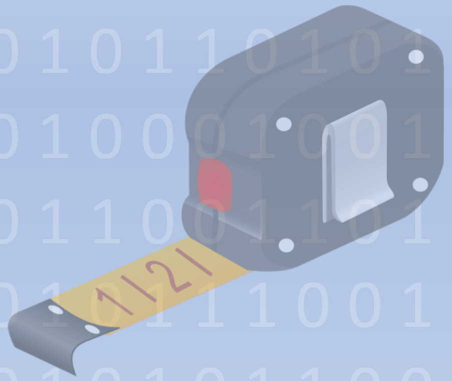
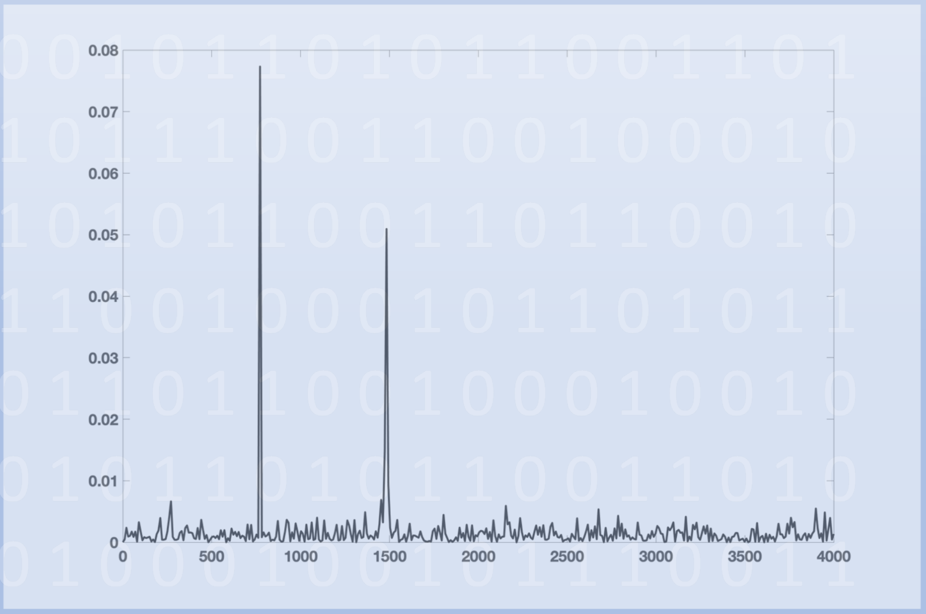
MEASUREMENT AND INSTRUMENTATION

An Introduction to Concepts and Methods





J.W. Dyer
C.E. Davis

Measurement and Instrumentation: An Introduction to Concepts and Methods

By

JOHN W. DYER, PH.D. & CHAD E. DAVIS, PH.D.

Norman, OK
2020

PREFACE

This textbook was developed with the goal of introducing the concepts of measurement and instrumentation to engineering students who have learned the underlying theory and need to develop the hands-on tools needed to make useful measurements in their engineering careers. The book is an outgrowth of the Measurement and Automation course taught at the University of Oklahoma. While the course is geared towards students majoring in Electrical or Computer Engineering, it is broadly applicable to all engineering majors. Engineers in research, industry, and manufacturing all deal with data acquisition and measurement at some point, but often times they do not have a solid understanding of what it means to measure something; what does the measurement really tell us about the thing being measured? How accurate is the measurement? How do we find out whether it is accurate? In a computer-based measurement system, how do we get the real-world phenomenon being measured into the computer? How does this process affect the reliability and accuracy of the measurements? This text attempts to answer these questions and give the student some insight into the concepts of making measurements and just what we can do or cannot do with the measurement information.

This textbook is aimed at senior-level students who have taken most of their math and engineering courses. Therefore, it will not go into too much detail on the numerous background concepts such as circuits, statics, and differential equations. Rather, it will employ these concepts in measurement scenarios. The text should also be very useful for graduate students who need to do data acquisition for their research.

A note on references: Most of the information in this book was developed from the authors' personal knowledge, but references are used to help cover some of the topics. The textbooks that are referenced are typically ones used in undergraduate and graduate courses. Hand written course notes that have been organized and kept over the years have also been a valuable source of information in creating this book. Some information from Internet sources have also been incorporated and noted. There are some fair-usage images from the Internet (that are specifically noted), but most figures in the text have been created *de novo* by the authors.

Contents

List Of Tables

List Of Figures

Chapter 1

1.1 Introduction

As the title of this text is *Measurement and Instrumentation*, it is appropriate that we begin with a brief discussion on concepts pertaining to taking *measurements*. It is broadly accepted that all engineering disciplines find the need, at times, to measure things. But *why* do we measure things? We measure things to gain information about that thing. What we do with that information varies widely, and our goals in acquiring the information play a role in how we make our measurements. For the moment we will focus on our desire to gain information about a *thing* (i.e. the device, process, or physical quantity that is being measured).

When we measure a thing, we are generally interested in some physical aspect such as length, weight, pressure, vibration, or any other of the many physical properties that something can have. If that thing is a process, then we might be interested in something less concrete; perhaps the number of people entering a store during a certain period of time. Regardless of the thing being measured, each measurement consists of two fundamental components that are both necessary to achieve the status of being a measurement. These elements are: *magnitude* and *units*. Both components are equally important, and neither of them provides enough information about the measurement on its own. For example, if I weigh a book and say,“it weighs two” this does not convey a satisfactory amount of information. If I say, “the book weighs pounds,” this also is not useful. It is only a useful measurement if the description includes both the magnitude and units, such as “the book weighs two pounds.” This simple example ignores the critical issue of how good the measurement is. We will get to the topics of measurement accuracy and error in section 1.3, but for now, the reader should be comfortable with the fundamental idea of what comprises a measurement.

When we specify a number for the magnitude of a measurement, we all understand what the enumeration used to represent the magnitude means. However, the units present a different historical tale. While the concept of units has existed as long as the concept of enumeration, the *meaning* of a unit has much more recently been codified into a uniform concept that scientists, engineers, and everyday people accept. Centuries ago, everyone understood what was meant when using descriptive phrases with enumeration, such as 3 cows, 2 horses, 5 buckets, etc. However, the specific engineering units that we rely on, such as length, weight, pressure, etc., were not well developed. Biblical references to length are often in cubits, for example, while the Greeks in ancient times used the *stade* to denote units of length. We owe the modern concept of fixed, verifiable units to the Age of Enlightenment. It was the French philosophers of the late 18th and early 19th century who understood the need to *standardize* units so that scientific experiments could be replicated for verification, bridges could be more accurately built, and civil infrastructure could be made more uniform (ensuring more efficient use of materials!). The end result is that in the 21st century we have a clear definition of specifically what one second is, what one foot is, what one kilogram is, etc. These units are defined by standards and the ability of all measuring devices to measure is defined with respect to these standards. The main concept to understand is that a measurement system must provide both a magnitude and unit of the physical quantity being measured for it to be meaningful.

1.2 Measurement Systems

We make measurements with a *measurement system*. The thing being measured is called the *measurand*. A measurement system can be as simple as a ruler, or as complex as a scanning electron-microscope. In general, we can divide measurement systems into two broad categories: *analog* and *digital*. An analog measurement does not require a computer or processor to perform the measurement, whereas a digital measurement requires some type of computing power that can *digitize* the physical representation of the measurand. In either case, the system has some *sensor* that physically interacts with the measurand. For example, a ruler is placed next to the object whose length we wish to measure. In this case the measurement system incorporates the human who reads and interprets the scale on the ruler. Sometimes the *signal* (the representation of the feature we are measuring) is modified, or conditioned. This occurs in a spring-based scale, for example. The spring that is displaced when one steps on a scale is only displaced a short distance, but through an internal mechanism this produces a much larger deflection in the scale readout dial.

A digital measuring system has many of the same elements as an analog measuring system, but with two significant differences. First, the sensor that interfaces with the environment almost always *transduces* (converts from one form to another) the physical feature into an electrical signal. Secondly, after any signal conditioning is performed, the digital measurement system also incorporates a device that converts the electrical signal into a discrete set of digital values. This is called *analog-to-digital* (A/D) conversion. The A/D converter is itself a measuring device that assigns a binary value to the voltage or current level being sensed. This will be discussed in more depth in Chapter 2. Figure 1.1 shows block diagrams of both an analog and a digital measuring system. In both systems some analog signal conditioning is often present. There are many types of analog signal conditioning that can be utilized to improve the measurement system. Some of these are amplification, analog filtering, and conversion from one type of electrical output to another. A Wheatstone Bridge (discussed in Section 4.1.3) is a popular method to convert a sensor that has an electrical output of resistance to a voltage so that it can be more easily displayed or digitized. Digital filtering is another form of signal conditioning that is used after the signal is digitized. Digital filters are very flexible and can be used to create very specific and unique filters (for example to eliminate 60 Hz line noise).

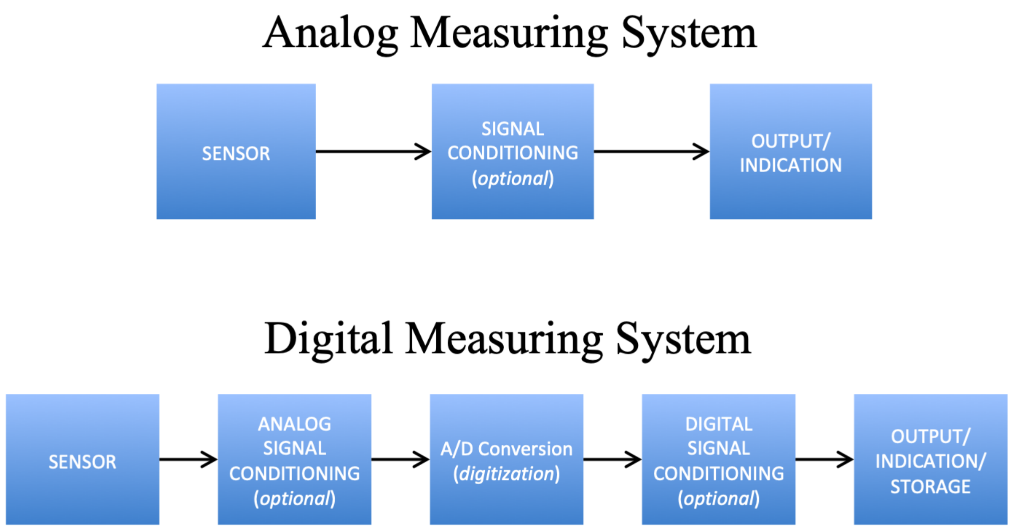

Analog Measuring System
SENSOR
SIGNAL CONDITIONING (optional)
OUTPUT/ INDICATION
Digital Measuring System
SENSOR
ANALOG SIGNAL CONDITIONING (optional)
A/D Conversion (digitization)
DIGITAL SIGNAL CONDITIONING (optional)
OUTPUT/ INDICATION/ STORAGE

Figure 1.1. ***Analog and Digital Measuring Systems:*** *The digital system also includes an analog-to-digital conversion block (called digitization in the diagram). The signal conditioning block is not present in some situations.*

1.2.1 Measurement Error

The goal of a measurement system is to find the *cosmic truth* about the measurand. However, it must be clearly understood that cosmic truth about a measurand is only a concept–there is no realizable system that measures cosmic truth. Indeed, we cannot *know* the truth; if we had a way of knowing the truth, we would not need to make measurements! However, all measuring systems made by humans incur measurement error in the system. This applies to both analog and digital measuring systems. However, a good measuring system can measure something closely enough that the difference between the result and cosmic truth has no practical effect on our design or decision-making. For example, if we have a manufacturing process that produces a designed shaft tolerance of +/- 0.001 inches (one mil), and our quality control measuring device is accurate to 0.000001 inches (one micron), then our measuring device can achieve a result that is as good as cosmic truth from our design specification perspective. The point is that all measuring systems (or devices) have some measurement error associated with them. Therefore, it must be understood that measurement error is an inescapable part of measuring something.

Measurement error is defined as the difference between the measured value and the *true* value:

$$\varepsilon(i) = x_{measured}(i) - x_{truth}(i)$$

Assuming the measurand has reached a steady state condition (see section 1.2.6 for more details on steady state) there are two broad categories of error: *deterministic* and *random*.

- Deterministic errors (also referred to as systematic errors) are those that have a static or known time-dependent behavior. These often show up as a bias in the data set. The term bias is often used interchangeably with the term "DC offset" because measuring systems often use transducers that produce electrical signals that include a DC offset voltage.

- Random errors are those whose value at any given time cannot be known but will fall within some range of possible values. Random errors typically follow some type of statistical distribution (the Gaussian distribution is the most commonly assumed form). Random errors are not repeatable from one trial to the next, while deterministic errors (especially DC offsets) are repeatable. In general, we can define random errors mathematically as the difference between the measured value and the average of all measured values.

$$\varepsilon_n[t_i] \quad = \quad x_n[t_i] \; - \; x_{\text{ave}}[t_i]$$

The average (or mean) value is obtained by taking repeated measurements from the same measurand. If we only take one measurement we can make no assessment about the veracity of that single measurement. It is only with repeated measurements that we can statistically assess the behavior of the system. Taking repeated measurements of a static measurand at a fixed time interval (e.g. measuring a 3.3V reference voltage once each second for an hour) is a straightforward process, but taking repeated measurements of a dynamic measurand such as an Electrocardiogram (ECG) signal that represents the electrical activity of the heart is much more complicated. The ECG signal through one heart beat cycle is repeated thousands of times each day. For this dynamic, but repeatable, signal, the statistical assessment of error is made across the different trials, but at the same time point within the cycle of the signal. This is shown for ten trials in Figure 1.2. The reader can see that the random error at a set time point (t_i) for any of the time trials (x_n) can be found by subtracting the mean value of all trials at that time point according to Equation 1.1.

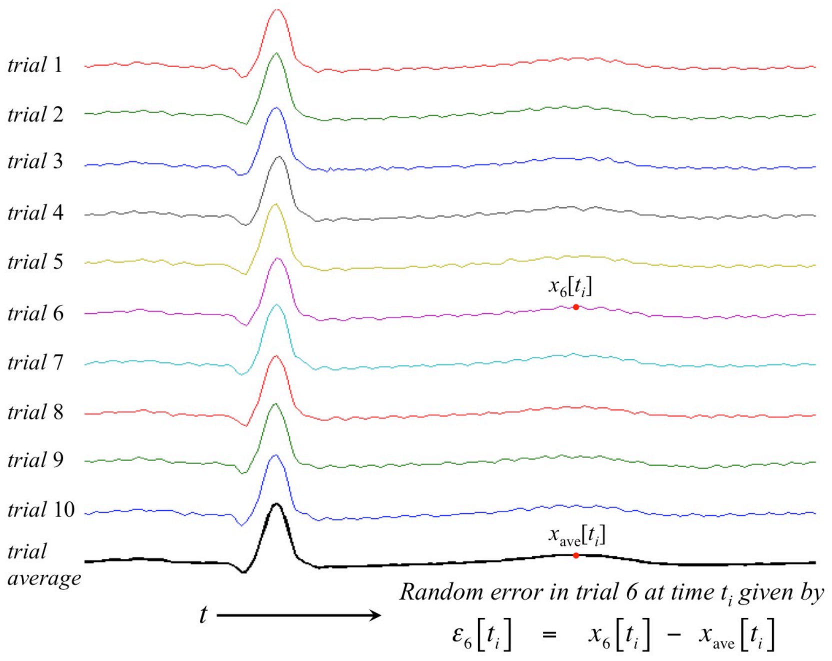


Figure 1.2. ***Time versus Trial Averaging:*** *The bottom waveform shows multiple instances of an ECG beat averaged together. The error for any given trial is taken as the difference between the trial value at some time t_i and the average at that same time point.*

It is important to recognize that the random errors are not affected by deterministic error. In order to assess the random error the mean value must be removed from the recorded data. On the other hand, deterministic error is found by taking the difference between the mean value of the trials and the true value. This creates a conundrum given that we have already asserted that we cannot ever really know the true value. Indeed, any fixed bias must be inferred by testing our measurement system against a known measurand. This is accomplished by calibration. For example, if we measure a certified 3.3V source and it consistently measures 3.8V, then we know that our measurement system has a deterministic error of 0.5V and that bias (or DC offset) can be easily removed. Furthermore, in many statistical digital signal processing applications the behavior of the *channel* between the source and the measurement system can be assessed based on the random component, so the bias error is generally removed whenever possible. Sources of random error arise from the electrical noise that is coupled into the signal path and also the electronics involved in measuring devices (thermal noise in the capacitors, resistors, circuit board traces, etc.). Sources of deterministic (or systematic) error are generally system-related and can be addressed through calibration procedures.

1.2.2 Uncertainty

In engineering, the concepts of *uncertainty* and *error* are considered separately, though they are related. Generally, statisticians think of error as the same thing as uncertainty, but in engineering, we define error specifically as being the difference between a measured value and the true value. As discussed above, the true value is a theoretical concept and cannot be exactly known. Uncertainty is considered to be the range of values, or the envelope of values, which our measurement system may return; it should be apparent that the *true* value is taken to exist somewhere in the space of uncertainty. The term *space of uncertainty* is used to encourage the reader to think in multiple dimensions. For example, in a three-dimensional position measurement (e.g. Global Positioning System, or GPS) the space of uncertainty is a three-dimensional volume. There is some uncertainty associated with the latitude measurement, another uncertainty associated with the longitude measurement, and yet a third uncertainty associated with the altitude measurement. The altitude (or vertical) uncertainty is usually larger for this system, so the volume is more ellipsoidal than spherical. To summarize: the error in any given measurement is the difference between the measurement and the truth, and the uncertainty is the range about the truth in which we expect the error to exist.

1.2.3 Precision and Accuracy

Random and deterministic errors cause us to question the validity of a measurement, or series of measurements. Furthermore, we must decide how to assess the validity of a group of measurements that all vary somewhat in value. To do this we use two concepts: *precision* and *accuracy*.

When determining how accurate a measurement is, the percent error (% Error) is usually used as the performance metric. The % Error is defined in Equation 1.3 as

$$\%Error = 100 \cdot \frac{(\text{Measured Value} - \text{True Value})}{\text{True Value}} \tag{1.1}$$

Precision is defined as the *repeatability* of a measurement. For example, if we use a caliper to measure a steel shaft with a nominal diameter of three inches, and we make four measurements of: 3.121″, 3.122″, 3.122″, 3.123″; we say that the caliper has good precision because the variance between the measurements is very small (i.e. the difference between each individual measurement and the mean is small). However, if we know that the steel shaft's diameter has a true value of 3.000″ (perhaps this shaft is a standard for calibration), then the caliper that we used has a significant systematic bias in its results that is an average value of 0.122″. This leads to the second concept, accuracy, which describes how close the average of the readings is to the true value. If we used another caliper that measured: 2.938″, 3.019″, 3.048″, and 2.991″; we get an average of 2.999″, which is only one mil (0.001″) from the nominal value. The difference between the mean value and each of the individual measurements is larger than the first group, so our second caliper is obviously much less precise, but the mean value of the measurements ended up being much more accurate. Figure 1.3 demonstrates the concepts of precision and accuracy by representing the nominal measurand value as the center of a target.

Example 1.1 *The true value of temperature is known to be* 27.1°C. *The system is at steady state and five measurements are made and the results are:* 26.93°C, 26.89°C, 27.23°C, 27.31°C, and 27.05°C. *What is likely the primary type of error that is occurring? Estimate the measured value and the % Error from these readings.*

Solution: Since the system is at steady state and the readings are above and below the true value instead of consistently high or low, **random errors** are likely the primary cause of the error. To minimize the error the average of the 5 readings can be taken as

$$\frac{26.93 + 26.89 + 27.23 + 27.31 + 27.05}{5} = 27.082$$

Estimated measured value = **27.082°C**

$$\%\text{ Error} = 100 \cdot \frac{\text{Measured Value} - \text{True Value}}{\text{True Value}} = 100 \cdot \frac{27.082 - 27.1}{27.1} = -0.06642\%$$

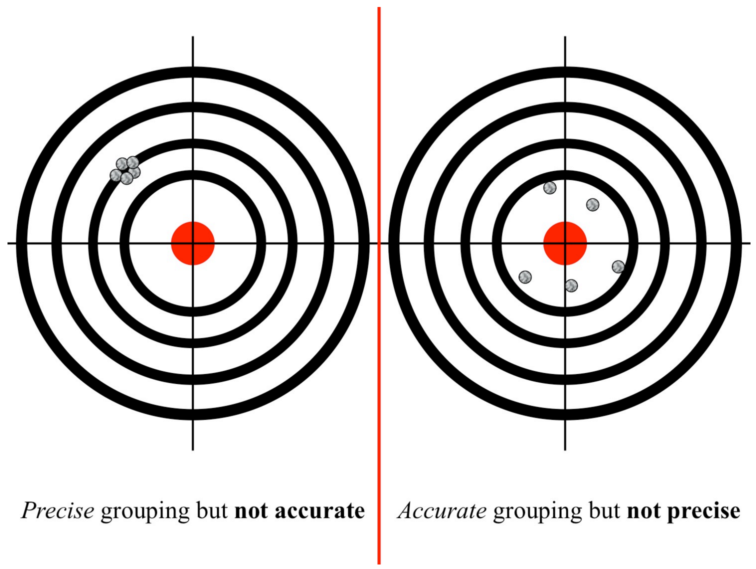

Precise grouping but **not accurate**
Accurate grouping but **not precise**

Figure 1.3. ***Precision versus Accuracy:*** *The left panel shows a very precise grouping of hits on the target, but it is not accurate if we take the center as the desired target point. The right panel shows a much larger spread, but the spread is centered on the central point of the target, meaning the grouping is less precise, but more accurate.*

Example 1.2 *The true value of temperature is known to be* 27.1 °C. *Several measurements are made and they are all very nearly equal to* 27.5 °C. *What is likely the primary type of error that is occurring? Find the % Error.*

Solution: Since the readings are consistently above the true value, bias errors are likely the primary cause of the error.

$$\%\text{ Error} = 100 \cdot \frac{\text{Measured Value} - \text{True Value}}{\text{True Value}} = 100 \cdot \frac{27.5 - 27.1}{27.1} = +1.476015\%$$

It seems evident that we would like to achieve both precision and accuracy in any measurement system, but the reality is that we cannot always achieve this. As with many other things in practice, there are often tradeoffs between the two. Inaccuracy can be improved if the bias causing the inaccuracy can be determined and removed. On the other hand, poor precision can be overcome with multiple trials. The number of trials required depends on how imprecise the measuring system is. In either case (precision or accuracy), the measurement system or device can only be assessed against some known standard. This leads us to the idea of *calibration*.

1.2.4 Calibration

How do we know whether our measurement system or device is precise, accurate, both, or neither? And does it really matter how precise or accurate the system is? The answer to the second question is equivocal! If you are trying to determine if an outlet is live or not, using a multimeter that shows 117 Vac is good enough. It may actually be 115 Vac, or 120 Vac, but since the goal is simply to determine whether the outlet is live, the exact value is not critical. On the other hand, the intensity of a pixel in a grayscale medical image could indicate whether a tumor is malignant or benign. In that case, it is imperative that the grayscale intensity of the pixels be assessed by some method that ensures some level of correctness. To determine the correctness of a measuring system, it must be used to measure a standard value that has a known pedigree and value. Scales are calibrated by known weight standards. Devices for measuring length are calibrated to *known* length standards. In the case of the medical imaging monitors, the output is what is questioned, so the pixels are set to output various levels and a second device that is known to provide an accurate measurement is used to assess the luminance. Many scientific instruments must be calibrated prior to using them in experiments so that true physical laws can be established. While the physicist is concerned with the calibration of equipment for an experiment, a machinist is likewise concerned with the calibration of a micrometer when turning a shaft down to a desired diameter.

Whatever the measurement system and measurand, even after calibration a measurement can only be guaranteed to a certain level. Clearly a ruler that has sub-markings at 1/8″ cannot measure differences in the 1/32″ range. But often the thing being measured will fall between two of the sub-markings. In this case, we say the measurement can be accurate to within ±1/2 of the sub-markings. For a ruler marked at intervals of 1/32″ the accuracy is ±1/64″. In other words, if the length of a measurand falls somewhere in between two 1/32″ marks, our measurement of the length will be in error by, at most, 1/64″.

Unit standards vary by industry. Many industries are moving toward the MKS system of measurements (meter-kilogram-second), but in the United States we still frequently encounter units in the FPS system (foot-pound-second). The choice of units is critical and must be observed by all parties involved in any large project. A glaring example of this issue occurred in the software for the Mars Climate Observer which famously crashed on the surface of Mars in 1999 during the attempt to land. The Jet Propulsion Labs (JPL) software controlling the thrusters during landing was expecting acceleration measurements in MKS units. Lockheed Martin was tasked with the software for the acceleration measurements and was providing those measurement values in FPS units. The specification called for MKS units, but the error was missed on both sides during the entire process prior to launch [LA Times, Oct 1, 1999]. Students get frustrated with professors who dock exam or homework points for missing units, but the preceding example is precisely why we constantly propound **units**, **units**, **units**!

Calibration is usually accomplished by using the measurement system to measure a *known standard*. Many measurements are taken against multiple known standards. Often, the resulting calibration employs some sort of *best fit* line because the actual measurements all have some error relative to the expected value based on the use of a standard. For example, to calibrate a measurement system for concentration of a chemical substance, one might use tightly controlled solutions of known concentration. The resulting plot of increasing concentrations might look like that shown in Figure 1.4. Because the concentrations being used are known to increase linearly, the measurements are likewise expected to increase linearly. Indeed, they generally do, as a *trend*, but from one measurement point to the next they might not follow the linear trend of the data set very closely. Often, we use a best-fit line for the calibration if the measurement is expected to increase or decrease linearly. How good is our *best-fit* estimate? This can be assessed using the R^2 value. This value is a measure of the *goodness of fit* between an estimate of the linear relationship, and the actual data points. R^2 ranges between 0 and 1 (or 0% and 100%), where a value of 0 indicates no fit, and a value of 1 indicates a perfect fit.

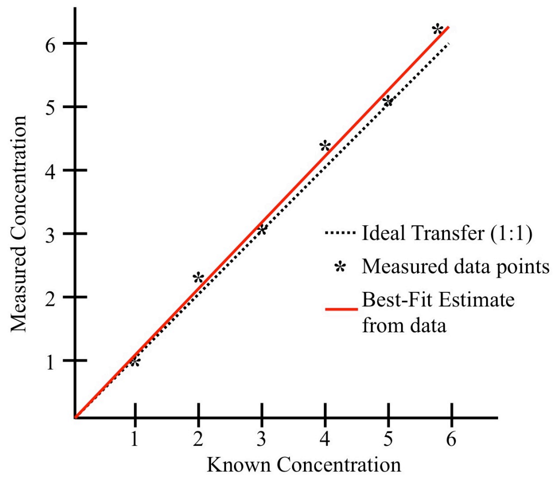


Figure 1.4. ***Calibration from data points:*** *Measurement data is taken for known values over a range of values. Then a best-fit line is estimated to provide calibration for the measuring system. The data are created for this example.*

The preceding example assumes that the trend line of a data set can be reasonably fit with a line. Sometimes this is not the case. There is generally some spread (variance) in the measured data set, relative to the standard being used, and there may be times when the best-fit line cannot pass through the data such that the line stays within the variance of every point. This can occur when the measuring device has some nonlinearities. The design goal of a measuring system is to minimize or eliminate nonlinearity if possible.

1.2.5 Linearity

Most measurement devices and systems make measurements over a clearly defined range of values. For example, a meter stick can measure something that is a few centimeters in length, to something that is one meter in length. One concern that must be addressed is whether or not the accuracy of the measurement is linear. For example, if we weigh things with a scale, is it just as accurate when weighing a five-pound item as it is when weighing a 100-pound item? Another example of the issue of linearity occurred in most cars built before the latter part of the 20^{th} century. All cable-driven speedometers were only linear over a certain range. To compensate for this, many manufacturers designed the speedometer to make measurements well into the hundreds of miles per hour (mph) even though the cars themselves were not capable of reaching the top indicated speed on the speedometer. This type of speedometer design was used to increase the indicator accuracy at the typical speeds at which the cars were driven because at this measurement range the speeds were much more linear, as demonstrated in Figure 1.5.

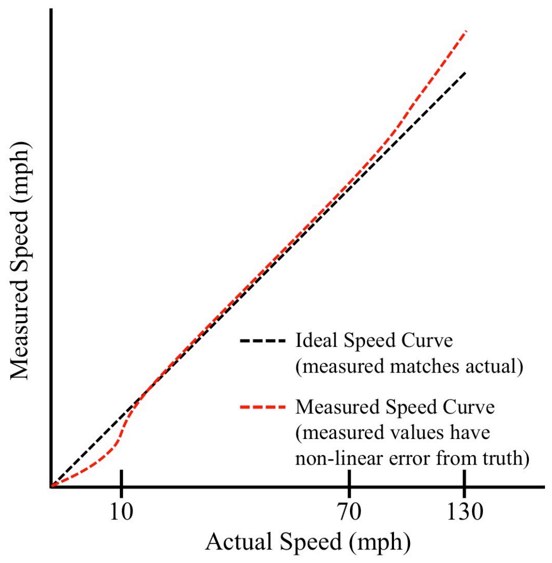


Figure 1.5. ***Nonlinearity effects:*** *Actual speedometer measurement is in error, but in a non-linear way. However, the significant non-linearities occur at the extremes of the scale (just a few miles per hour, or in excess of 70 miles per hour; both conditions of minor importance for typical driving speeds).*

1.2.6 Static versus Dynamic

Finally, let us discuss the concept of static versus dynamic measurements. In a static measurement system, we expect any instantaneous measurement to be representative of the measurand. On the other hand, in a dynamic measurement system, we expect the measurand to be changing with time, and the system response to be adjusting in time as well. Sometimes the system response in a dynamic setting may *lag* the actual physical change. For example, a thermocouple's voltage changes take some finite amount of time to adjust to the current ambient temperature; the metals both have temperature related characteristics that keep them from instantaneously changing. Thus, the measurement of temperature using a thermocouple may lag by some time from the actual temperature change. The time it takes for a measurement system to respond to a changing measurand is the *response time*. In general, we want to choose a sensor or measurement system with a response time faster than the expected dynamic changes in the measurand.

We can also consider the response behavior of a measurement system in terms of frequency, recalling that frequency and time are inversely related. In this case, a measurand that changes more often (at a *higher* frequency) calls for a measurement system that has a faster response time. This information is often presented in terms of *bandwidth*. A sensor or measurement system is specified as having a frequency response between some low frequency and some high frequency. For example, a particular accelerometer on the market has a specified bandwidth between DC and 100 Hz. One can readily guess that if the measurand is a vibration above 100 Hz, the accelerometer will not be able to accurately represent the vibration signal. This is because its bandwidth limitations don't allow it to *respond* quickly enough to the dynamic measurand.

The output signal of electronic components generally contains decaying exponential functions ($e^{-\alpha t}$) that approach zero as the signal reaches steady state. The signal often oscillates around the final value a few cycles before it settles to its final steady state value. These effects are demonstrated graphically in Figure 1.6. In this figure the input sensor detects a near-instantaneous change in the measurand. However, the measurement system response time is limited by its front-end electronics and responds more slowly. Two responses are shown: the exponential response time, and an exponential response with a small oscillation. Either way, it is evident that the first few milliseconds of readings from the measurement system should be disregarded as invalid because steady state has not been reached.

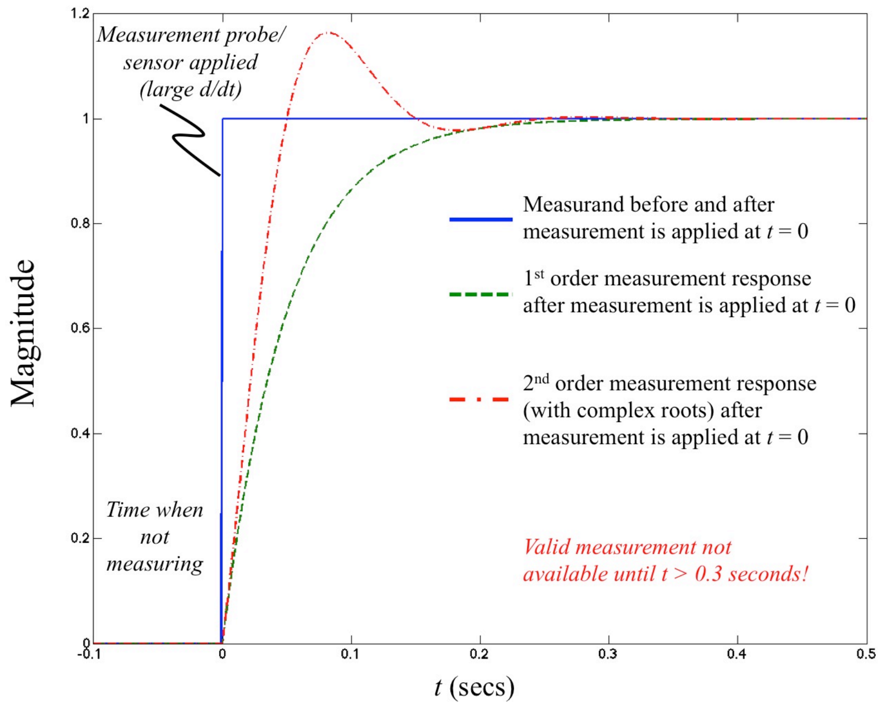


Figure 1.6. ***Response Time:*** *Measurement system response time compared to step change in system input. Note that whether the response is a first-order or second-order response, there is a finite amount of time before the output is representative of the input.*

A more specific example can be found in a common signal conditioning electronic component called an Operational Amplifier (Op Amp). The Op Amp has a feature called *slew-rate* that prevents its output from being able to instantaneously respond to a sudden change in input voltage. The slew rate is typically approximated as a linear function and typically defined as the change in voltage divided by the change in time ($SR = \Delta V/\Delta t$). A parameter called *full power bandwidth* (FPB) is used to describe whether a sinusoidal input will result in slew rate distortion. FPB is defined as:

$$FPB = \frac{SR}{2\pi \cdot V_{o,max}}$$

General-purpose Op Amps typically have slew rates less than 1 μV/s. This means that if the voltage signal going into an Op Amp changes at a faster rate than the slew rate for that Op Amp (as published in its datasheet) then the signal coming out of the Op Amp will be distorted because it is incapable of changing voltage fast enough. Example 1.3 shows more details of how slew rate distortion occurs when using Op Amps.

Example 1.3 *Will slew rate distortion occur if a 10 kHz sinusoidal signal with an amplitude of 4 Volts is input into a LM324 Op Amp configured for unity gain (i.e. the output voltage should equal the input voltage)? What is the maximum amplitude of the 10 kHz sinusoid before slew rate distortion occurs? Assume the 10 kHz sinusoid has the following equation*: $A \cdot \sin(2\pi \cdot f \cdot t)$

Solution: The LM324 is a general-purpose Op Amp that is produced and sold by many different companies. For example, the datasheet for the LM324 sold by Texas Instruments is shown at the following link: `http://www.ti.com/lit/ds/symlink/lm324-n.pdf`. Figure 7 of the datasheet in the link shows a graph that can be used to estimate the slew rate for the LM324. It shows that if a square wave goes into the LM324, then the front edge of the square wave input is converted to a ramp function output that changes by approximately 2.4 Volts in 5 μs. From this figure the slew rate is approximated as:

$$SR = \frac{2.4V}{5\mu s} = 0.48V/\mu s$$

The fastest changing part of a sine wave occurs at the zero crossing. If the time difference between each sample is 1 μs the most extreme change in voltage per time can be found from:

- @ time = 0 $\rightarrow v(0) = 4 \cdot \sin(2\pi \cdot 10,000 \cdot t) = 4 \cdot \sin(2\pi \cdot 10,000 \cdot 0) = 0$ Volts
- @ time = $1\mu s \rightarrow v(1\mu s) = 4 \cdot \sin(2\pi \cdot 10,000 \cdot 0.000001) = 0.2512$ Volts

Since the voltage at $t = 1\mu s$ is less than 0.48 Volts, then slew rate distortion will not occur. The sinusoidal signal (dashed blue line) and the slew rate limit (solid red line) are shown in the figure below (left).

To determine the maximum amplitude of the 10 kHz sinusoid before slew rate distortion occurs the following equation needs to be used to solve for the maximum amplitude, A_{max}. $A_{max} \cdot \sin(2\pi \cdot 10,000 \cdot 0.000001) = 0.48\text{Volts} \rightarrow A_{max} = 7.644\text{Volts}$.

We can use the maximum amplitude in the full power bandwidth equation to show that the maximum frequency at this amplitude is, in fact, 10 KHz.

$$FPB = \frac{SR}{2\pi \cdot V_{o,max}} \quad = \quad \frac{0.48}{2\pi \cdot 7.644} \quad \approx \quad 10,000 \text{ Hz.}$$

Figure 1.7 below (right) shows that the max amplitude before slew rate distortion occurs is 7.644.

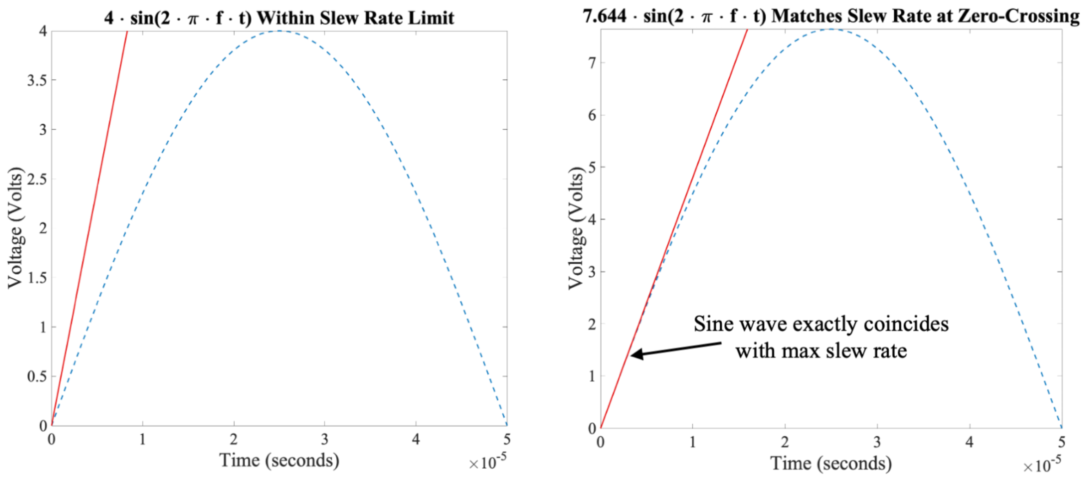


Figure 1.7. ***Slew Rate Example:*** *In the left plot, the slew rate slope is steeper than any part of the sine wave signal, so no distortion will occur. In the plot on the right, the slew rate slope exactly coincides with the sine wave signal. No distortion should occur, because the signal does not* exceed *the slew rate. However, if the amplitude of the sine wave increased, then slew-rate distortion would occur.*

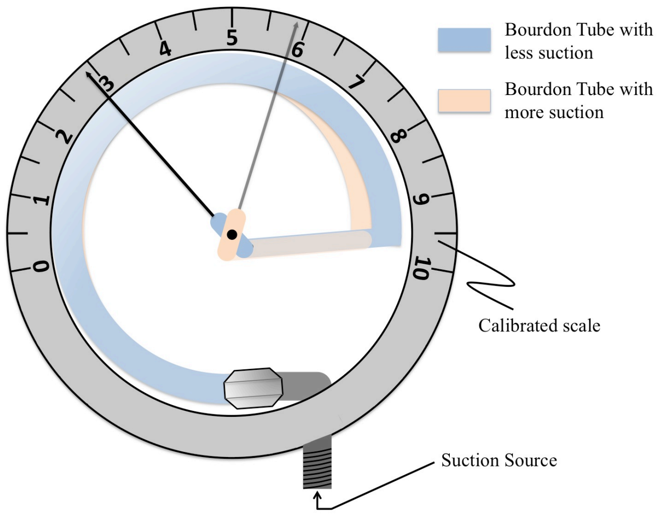
1.3 Conclusion

The remainder of this book will go deeper into some of the concepts mentioned in this introduction, as well as go into many more advanced topics related to measurement and instrumentation. Chapter 2 presents the fundamental elements of data acquisition. The concepts of measurement noise and signal conditioning are thoroughly covered in Chapter 3. Sensor types and mechanisms are discussed in Chapter 4. An overview of statistics and distributions as they relate to measurements are covered in Chapter 5. Summaries of some of the projects used in the Measurement and Automation course at OU are shown in chapters 6 though 8. These projects apply many of the concepts covered in chapters 1 through 5 and will be useful in gaining a deeper understanding of measurement and instrumentation even if you aren't taking the course at OU. Finally, since LabVIEW is used for the Measurement and Automation course and is a highly recommended software package for measurement systems, a detailed tutorial for LabVIEW is provided in the Appendix. It is recommended that the Appendix be reviewed before going through the projects in chapters 6 through 8. Additionally, it is important to note that numerous practical examples are included throughout this book to help students internalize and apply the theory they are learning.

Chapter 2

2.1 Data Acquisition

Data acquisition as a concept can be applied in both the analog and digital domains. The first step in data acquisition is to identify a physical phenomenon that is of interest. Next, a transducer must be used to transform that physical phenomenon into a physical quantity that can be measured. Finally, data is acquired (i.e. measurements are made) that represents the physical phenomenon. This process can be demonstrated by a simple analog measurement system called a Bourdon tube, as shown in Figure 2.1. In most small airplanes the instrument panel contains a *vacuum gauge* that measures the amount of vacuum produced by the engine and in older aircraft a Bourdon tube is used for the vacuum measurement. In this example, the physical phenomenon is the suction (negative pressure) in the engine's pipes, the transducer is the Bourdon tube (it converts negative pressure to a mechanical movement), and the measurement of vacuum is shown on the dial. The Bourdon tube is a soft metal tube that can bend slightly under pressure. The end of the tube is connected to a lever (and sometimes gearing) that moves the needle on its pivot. In this case, the data acquisition system is strictly analog and the data representing vacuum is only displayed instead of acquired by a computer.



0
1
2
3
4
5
6
7
8
9
10
Bourdon Tube with less suction
Bourdon Tube with more suction
Calibrated scale
Suction Source

Figure 2.1. ***Analog Measurement System:*** *The vacuum gauge from a small airplane typically uses a Bourdon tube to transduce the amount of vacuum (suction) produced by the engine to drive instruments that require vacuum. The scale is calibrated by using a known vacuum source.*

However, in this text we are much more interested in digital measurement systems because nearly all in-depth engineering analysis is based on data that is acquired by a computer. There are several reasons for this. First, computerized systems are generally much more consistent than manual analysis methods. Computerized system can also hold lots of data and can process the data easily, but manual analysis methods are intractable on large volumes of data. Finally, digital signal processing of the data as they are collected is very powerful and can be used to improve the measurement system in many ways (e.g. filtering out 60 Hz noise). If we were to measure vacuum with a computer-based system, we would still need a transducer, but it would convert vacuum (negative pressure) into an electrical signal. Voltage, current, and resistance are the most common electrical signals produced by transducers. Data acquisition (DAQ) hardware units are typically configured with voltage inputs, so we will focus on transducers with voltage outputs in this chapter. However, if a transducer converts a physical quantity to current or resistance, instead of voltage, an electrical conversion can be performed using various methods. For example, current can be converted to voltage using a Bipolar Junction Transistor (BJT) and resistance can be converted to voltage using a Wheatstone Bridge signal conditioning circuit (as described in Section 4.1.3). Chapters 3 and 4 will explain many additional measurement system details, but for the purpose of describing data acquisition in this chapter let us assume that we have a transducer that produces a change in voltage that is proportional to any change in the physical phenomenon. We will also use the term sensor instead of transducer because sensors are physical components used in measurement systems that serve the function of the transducer.

2.2 Computer-based Data Acquisition

2.2.1 Analog-to-Digital Conversion

Typically, a voltage-based sensor will have a defined range of voltage over which it operates in a predictably linear fashion. For example, a pressure sensor may be calibrated to produce an output of 0 to 5 volts for a pressure range of 0 inches of mercury down to a pressure of -10 inches of mercury (i.e. vacuum, or suction). At positive pressures, the voltage output would remain at 0 volts, while at negative pressures below -10 inches of mercury the voltage would be limited to 5 volts. The range of voltages over which the sensor operates is referred to as its *dynamic range* or *span*. The mathematically-minded reader might note that the output voltage from the sensor in this example could be considered a *real-valued scalar*. In other words, the output voltage is an element of the continuous set of real (as in the set of reals, $\mathbb{R}$) values between 0 and 5. This set of continuous values is, of course, infinite in number. The question that arises from this example is *how can we identify an infinite number of voltages between 0 and 5 volts inside a computer with a finite number of binary digits with which to represent numbers*? The answer to that questions is that *we cannot do it*! The solution is to digitize the voltage with what is called an *analog-to-digital converter* or A/D converter (ADC). An ADC employs a set number of binary digits (n bits) to represent the values it is converting. Therefore, the infinite number of different values in the dynamic range is limited by the number of distinct values that can be produced by n bits. The number of distinct values (or steps) is given by:

$$\#\ values = 2^n \tag{2.1}$$

This leads us to our first significant element in data acquisition. Specifically, we can determine the *resolution* of our ADC by dividing the dynamic range of the sensor by the number of steps produced by using n bits:

$$\text{ADC resolution} = \frac{\text{Dynamic Range}}{2^n} \tag{2.2}$$

When the dynamic range is in Volts then ADC resolution has units of Volts/step. If the dynamic range is in different units than the resolution units will change accordingly.

Example 2.1 *Using a current range from 4 to 20 mA (which is a common standard in measurement and control systems and described in this link:* `https://en.wikipedia.org/wiki/Current_loop`*), calculate the resolution when using four different A to D converters with the following numbers of bits: a) 3, b) 12, c) 16, d) 24.*

Solution:

The dynamic range is equal to the max – min $\Rightarrow$ Dynamic Range = 20 – 4 = 16 mA

a) $n = 3 \Rightarrow \text{Resolution} = \dfrac{\text{Range}}{(2^n)} = \dfrac{16\text{mA}}{(2^3)} = 2 \text{ mA/step}$

b) $n = 12 \Rightarrow \text{Resolution} = \dfrac{\text{Range}}{(2^n)} = \dfrac{16\text{mA}}{(2^{12})} = 0.003906 \text{ mA/step or } 3.906\ \mu\text{A/step}$

c) $n = 16 \Rightarrow \text{Resolution} = \dfrac{\text{Range}}{(2^n)} = \dfrac{16\text{mA}}{(2^{16})} = 0.000244 \text{ mA/step or } 0.244\ \mu\text{A/step}$

d) $n = 24 \Rightarrow \text{Resolution} = \dfrac{\text{Range}}{(2^n)} = \dfrac{16\text{mA}}{(2^{24})} = 9.54\text{E} - 7 \text{ mA/step or } 0.954 \text{ nA/step}$

This example shows that if a resolution no greater than 1 nA/step is desired for a 4 to 20 mA measurement range, a 24-bit A to D converter is a good choice because it meets the specifications, but doesn't include extra bits that results in wasted memory and increased cost of the system. Likewise, if a resolution no greater than 4 μA/step is desired a 12-bit A to D converter is the ideal choice. Even though the 16-bit and 24-bit converters would also have sufficient resolution (i.e. less than 4 μA/step), they have extra bits that are not needed and wasteful.

For example, if you needed "no greater than" 1.6 μA/step, which (a, b, c, or d) is the best option?

Answer: 16-bit ADC

Example 2.2 *If a 3-bit ADC is used in a measurement system that has a range of 0 to 5V, correlate each of the binary numbers that make up the 8 steps to a specific voltage value. If a sensor with a near 100% accuracy measured a value of 2.36 V with an ADC that had an infinite number of bits and there were no other errors in the measurement system, what bias error results from the 3-bit ADC.*

Solution:

$n = 3 \Rightarrow \text{Resolution} = \dfrac{\text{Range}}{2^n} = \dfrac{5}{2^3} = 0.625 \text{ V/step}$

The 8 states resulting from using 3 bits could be set as follows:

- 000 ⇒ 0 V
- 001 ⇒ 0.625 V
- 010 ⇒ 1.25 V
- 011 ⇒ 1.875 V
- 100 ⇒ 2.5 V
- 101 ⇒ 3.125 V
- 110 ⇒ 3.75 V
- 111 ⇒ **4.375 V**

Notice that the highest possible value (**4.375 V**) is one step less than the maximum of the 5 Volt range. Cutting off a large portion of the dynamic range is a significant problem in this example, but in a practical ADC the number of bits is much larger than 3 so there is not much of a loss in range. For example, the lowest number of bits that is typically used is 8 bits and if an 8-bit ADC was used in this example the resolution would be 0.0195 V/step and the highest voltage at 11111111 would equal 4.98 Volts.

In this hypothetical example with perfect accuracy and no errors, the measurement of 2.36 V is considered the *True Value*. When using the 3-bit ADC, 2.36 V would be mapped to the closest binary number of 2.5 V (from the binary 100 state). The resulting bias error is 2.5 – 2.36 = 0.14 V and the % Error could be calculated as:

$$\% \text{ Error} = 100 \cdot \frac{\text{Measured Value} - \text{True Value}}{\text{True Value}} = 100 \cdot \frac{2.5 - 2.36}{2.36} = 5.9322\ \%$$

Another way to understand the effect of ADC resolution is to look at the *quantization error*, which is the difference between the value from the sensor sent to the ADC and the value that it is mapped to in the ADC. The quantization error is found by multiplying the resolution from equation 2.2 by ±0.5. The ±0.5 is used because the ADC system must create an integer number, so an ambiguous voltage that falls in between two integer values will be either rounded down or up. Therefore, the error is never greater than one half of the resolution. This can be explained by looking at the simple 3-bit ADC in example 2.2. The resolution was calculated to be 0.625 V/step so the maximum quantization error is ±0.3125 Volts. This means that the worst possible error between the actual value and the value that was digitized by the ADC is ±0.3125 V. The following example explains this in more detail.

Example 2.3 *If a 4-bit ADC is used in a measurement system that has a range of -12V to +12V, determine the worst possible error magnitude that will occur due to the measurement being digitized (i.e. quantization error). Also, determine which measured values would produce the worst possible error magnitude.*

Solution:

$$n = 4 \Rightarrow \text{Resolution} = \frac{\text{Range}}{2^n} = \frac{24 \text{ V}}{16 \text{ steps}} = \mathbf{1.5\ V/step}$$

The worst possible error would be equal to $\pm 0.5 \cdot 1.5\text{V} = \pm 0.75\text{V}$

The measured values that would have a 0.75 V error magnitude are the 15 voltages that fall at the midpoints between the 16 steps of the 4-bit ADC. The values of the 16 steps that correspond with bit values of 0000 to 1111 are as follows: {-12, -10.5, -9, -7.5, -6, -4.5, -3, -1.5, 0, 1.5, 3, 4.5, 6, 7.5, 9, 10.5} and the 15 midpoints between these 16 values that represent the different 4-bit combinations are: {-11.25, -9.75, -8.25, -6.75, -5.25, -3.75, -2.25, -0.75, 0.75, 2.25, 3.75, 5.25, 6.75, 8.25, 9.75}. All 15 of these measurements would have an error magnitude of 0.75 Volts.

The quantization error is illustrated in Figure 2.2. It shows that a sensor's output voltage signal (in black) is changing over time. The steps in red denote the quantized binary values that the ADC system is capable of computing. As the voltage rises, the quantized value takes on the integer below the actual voltage value until the actual voltage gets halfway between the current quantized integer and the next higher quantized integer. At this point the ADC then rounds up to the higher quantized integer value. Thus, the ambiguity in the quantized voltage is half of the quantization step size.

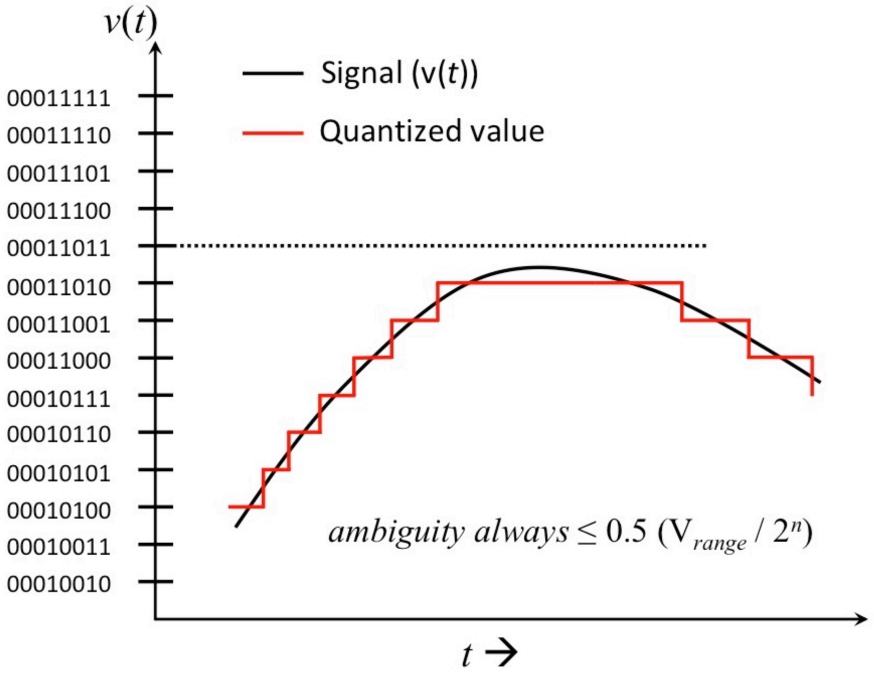


Figure 2.2. ***Quantization Ambiguity:*** *Because binary values come in one-bit quanta, an ADC system cannot resolve voltage changes less than half of the step size represented by a one-bit change. The red line represents the quantized voltage value, while the black line represents the voltage being measured. It can be seen that the quantized value stays the same until the change in voltage exceeds one-half of the step size.*

In examples 2.1, 2.2, and 2.3, the limits of the ADC were set to values of 4 to 20 mA, 0 to 5V, and -12 to +12 volts respectively. It should be noted that these limits are almost always user-adjustable in data acquisition systems. However, if the ADC limits can be set by the user it should be immediately obvious that the user could set the limits improperly and produce an undesirable result. If the range is set too wide, then the resolution is much larger than it needs to be and if the range is set too narrow the signal will exceed the user-specified limits and will be capped at the max or min levels. Neither case is desirable but setting the limits too narrow is far worse because it will produce false results. For example, if the sensor voltage ranges from 0 to 5 volts, and the ADC limits are set to ±2 volts, whenever the sensor voltage exceeds 2 volts, the ADC output will remain at 2 volts. The sensor's voltage signal is being *clipped* by the system and this produces *non-linear* behavior, which is undesirable. On the other hand, setting the ADC limits much larger than the sensor voltage range will always keep the result linear, but the resolution will be reduced. The following example shows how this happens.

Example 2.4 *If a 12-bit ADC is used with a sensor that should fall between ±2 volts, but the range was over-conservatively set to ±10 volts, what happens?*

<u>Solution</u>

Even though the sensor has an actual range of 4 volts, the ADC is set to handle a range of 20 volts. In this case the maximum quantization error is:

$$\pm 0.5\frac{20}{2^{12}} = \pm 0.5\frac{20}{4096} \;\approx\; \pm 2.441 \text{ (mV)}$$

If the range was set to the proper value of ±2 volts this error is drastically improved.

$$\pm 0.5\frac{4}{2^{12}} = \pm 0.5\frac{4}{4096} \;\approx\; \pm 0.488 \text{ (mV)}$$

By inflating the range to ±10 V, the maximum quantization error that can occur in the ADC process is 1.953 millivolts (or 400 %) higher than it needs to be.

Example 2.5 *A sensor has an output voltage range from 0 to 5 volts. Knowing that the cost of an ADC device is directly proportional to the number of conversion bits, you are tasked with determining the minimum number of bits that can provide a resolution of no larger than 0.5 millivolts/step in order to choose an ADC unit.*

<u>Solution:</u> By rearranging Equation 2.2, we have:

$$\begin{aligned} 2^n &= \frac{5-0}{2.25 \times 10^{-6}} \;=\; 10000 \\ &\Downarrow \\ n &\approx 13.288 \end{aligned}$$

Obviously, we cannot have a fractional number of bits, and since 13 bits isn't enough ($2^{13} = 8192 < 10000$), we must choose at least 14 bits. Practically speaking, most ADC units come with bits counted in factors of 2, so one might find a 12-bit ADC, a 14-bit ADC, or a 16-bit ADC. When looking at those choices the 14-bit ADC is the best choice and the resolution will be about 0.3 millivolts/step which is 40% better than the 0.5 millivolts/step resolution limit that was specified.

The ambiguity in measurement that is illustrated in Figure 2.2 is called *quantization noise*. The effect of this quantization noise is measured in terms of the root-mean-square (RMS) of the ambiguity. RMS is a standard tool used to assess the power in a zero-mean process. The quantity of interest is squared, and its mean is taken. Then, the square root of the mean is computed. Letting the error be represented by ϵ, then we compute the quantization noise power as

$$\begin{aligned} V_{q(RMS)} &= \sqrt{\frac{1}{Q}\int_{-\frac{Q}{2}}^{+\frac{Q}{2}} \epsilon^2 d\epsilon} \\ &= \sqrt{\frac{1}{Q}\left[\frac{\epsilon^3}{3}\right]_{-\frac{Q}{2}}^{+\frac{Q}{2}}} \\ V_{q(RMS)} &= \frac{Q}{\sqrt{12}} \qquad (2.3) \end{aligned}$$

where Q is the single-bit quantum step (i.e. the voltage range divided by 2^n). An important value in digital systems is the signal-to-noise ratio (SNR) based on the number of bits being used. SNR is found by taking the ratio of the signal's power to the noise power associated with the quantization noise. Signal power is estimated by assuming a zero-mean, sinusoidal signal that ranges between the maximum and minimum values of the ADC. Recall that a sinusoid represented by $A\sin(\omega t)$ has an RMS value given by

$$V_{RMS} = \left[\frac{1}{2\pi}\int_0^{2\pi} A\sin^2(\omega t)\,d\omega\right]^{\frac{1}{2}}$$
$$\Downarrow$$
$$V_{RMS} = \frac{A}{\sqrt{2}} \tag{2.4}$$

Calling the peak-to-peak value (i.e. the maximum voltage range of the ADC) V_{range}, then the presumed sinusoid is represented by $\frac{V_{range}}{2}\sin(\omega t)$, so the RMS of this signal is

$$V_{RMS} = \frac{\frac{V_{range}}{2}}{\sqrt{2}} \tag{2.5}$$

We note that V_{range} can also be represented as a function of the number of bits the system uses, and the quantization step, Q, namely

$$V_{range} = 2^N Q \tag{2.6}$$

Combining equations 2.5 and 2.6, we get

$$V_{RMS} = \frac{2^N Q}{2\sqrt{2}} \tag{2.7}$$

SNR, as stated above, is given by taking the ratio of the signal power (Eq 2.7) to the quantization noise power (Eq 2.3). This is typically computed in dB to achieve the following:

$$
\begin{aligned}
SNR &= 20\log\left(\frac{V_{RMS}}{V_{q(RMS)}}\right) \\
&= 20\log\left(\frac{\frac{2^N Q}{2\sqrt{2}}}{\frac{Q}{\sqrt{12}}}\right) \\
&= 20\log\left(\frac{2^N\sqrt{3}}{\sqrt{2}}\right) \\
&= 20\log 2^N + 20\log\frac{\sqrt{3}}{\sqrt{2}} \\
SNR &= 6.02N + 1.76 \;\; (dB) \qquad (2.8)
\end{aligned}
$$

This is an important finding because it shows that we can improve the SNR due to quantization error by about 6 dB for every bit we add to the conversion. At the time of writing, many smart phones use 24-bit ADCs. In the future as miniaturization continues, the SNR inside *smartphones* might need to be increased and Equation 2.8 can be used to inform that decision. Of course, the engineer should be prepared to assess the requirements of any given project and determine the best (in terms of cost, ease of implementation, ability to meet specification, reliability, manufacturability, etc.) ADC system to implement. In the measurement and instrumentation environment this decision is primarily driven by the nature of the measurand and the sensor you are using to measure it.

2.2.2 Sampling Rate

Once we have an analog-to-digital converter (ADC) that can generate a digital representation of the measured voltage signal, another question to be answered is *how often should we make a measurement*? Recalling the nature of our real-world voltage source (from some sensor), we know that the sensor output exists for all time, $t \in \mathbb{R}$. But it should be obvious that we cannot convert a voltage to a digital value at every instant in time, as this would require the computer system to hold an infinite number of data points. However, we do not want our system to miss any important information in the incoming sensor signal, so deciding how often to store a value of the sensor signal for use in the computer-based acquisition system is critical. This area is typically more difficult for students to understand, because it requires a thorough understanding of the concept that a time-varying signal can be represented as an infinite summation of weighted sinusoids. This section will discuss the Fourier Transform, the Nyquist criterion, and how these relate to data acquisition.

2.2.2.1 Fourier Series and Fourier Transform

A thorough examination of the underlying math for the Fourier Transform is outside the scope of this book. However, this section will attempt to present a layman's version of the Fourier Transform to help the reader better understand sampling theory. Fourier asserted that any signal could be represented as an infinite sum of weighted sinusoids. However, if we start with a simpler signal model called the *Fourier Series*, the concept may be easier to comprehend. Using the Fourier Series, a periodic signal with a fundamental frequency f_0 can be represented by an infinite sum of sinusoids, where each sinusoidal term has a frequency that is an integer multiple (i.e. harmonic) of f_0. For example, if we combine the odd harmonics of a sine wave of form $A \cdot \sin(2\pi f_0 t)$ with the amplitude (A) of each harmonic scaled with the correct weight, we can generate a square wave!

The following equation shows the Fourier series for a square wave with amplitude (A), and a fundamental frequency (f_0), that is created by taking an infinite sum of odd harmonic sine waves. The **bolded numbers** are the odd harmonics. All of the even harmonics in a square wave have *Fourier coefficients* of zero so they are not included.

$$v(t) = \frac{4 \cdot A}{\pi} \left[\frac{\sin(\mathbf{1} \cdot 2\pi f_0 \cdot t)}{1} + \frac{\sin(\mathbf{3} \cdot 2\pi f_0 \cdot t)}{3} + \frac{\sin(\mathbf{5} \cdot 2\pi f_0 \cdot t)}{5} + \cdots \frac{\sin(\mathbf{N} \cdot 2\pi f_0 \cdot t)}{N} \right]$$

For more information on generating square waves, see: `http://www.allaboutcircuits.com/textbook/alternating-current/chpt-7/square-wave-signals/`

This equation is plotted in Figure 2.3, where a sine wave with a fundamental frequency of f_0 has been added to the first four odd harmonics (k = 3, 5, 7, and 9), each with a successively smaller weighting coefficient. Fourier's theory states that the summation of harmonics is infinite, and it is clear in the figure that the resulting square wave has significant distortion when only summing a few harmonics. However, if we sum the first 200 odd harmonics (as shown in Figure 2.4), the transitions become much sharper and the only distortion is at the discontinuity.

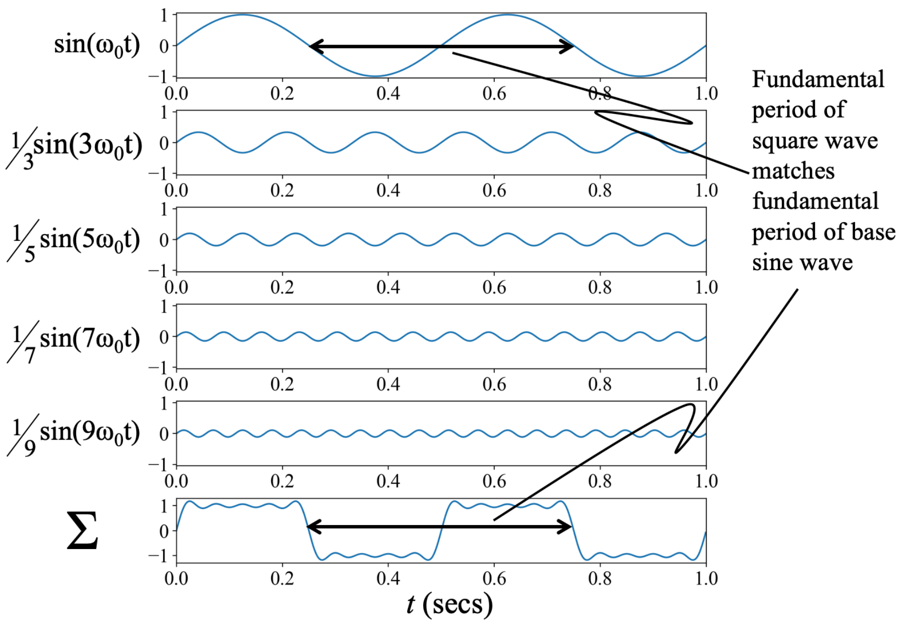


Figure 2.3. ***Square Wave from Four Odd Harmonics:*** *A periodic square wave can be produced, using the theory developed by Fourier, by adding the odd harmonics of a sine wave, and giving each odd harmonic a smaller weighting coefficient. Note that we can still imagine that all the even harmonics are summed as well, but their weighting coefficients are all zero.*

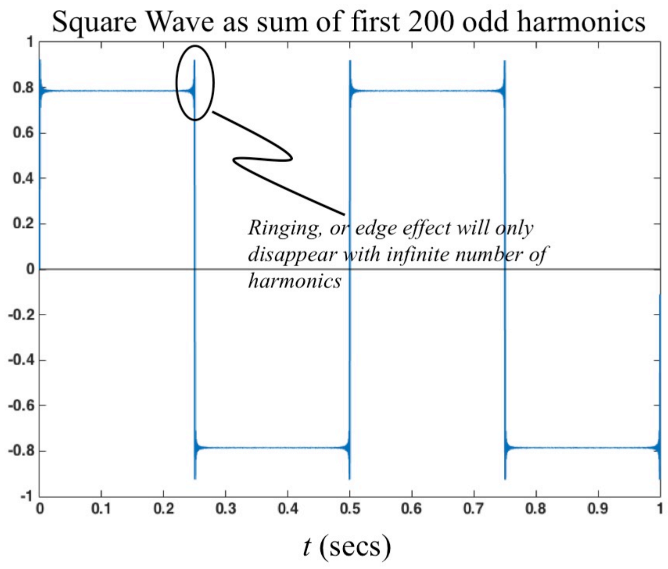


Figure 2.4. ***Square Wave from many Odd Harmonics:*** *The square wave in this graph was produced by summing 200 odd harmonics of the fundamental sine wave. In this case the waveform looks very much like a square wave, except for the ringing at the edge of each transition.*

Fourier series analysis will next be performed on a triangle wave with amplitude (A) and fundamental frequency (f_0). In similar fashion to the square wave shown in the previous figures, the triangle wave can be created by taking an infinite sum of odd harmonic sine waves, but with different Fourier coefficients and alternating signs of each term:

$$v(t) = \frac{8 \cdot A}{\pi^2}\left[\frac{\sin(\mathbf{1} \cdot 2\pi f_0 \cdot t)}{1} - \frac{\sin(\mathbf{3} \cdot 2\pi f_0 \cdot t)}{9} + \frac{\sin(\mathbf{5} \cdot 2\pi f_0 \cdot t)}{25} - \cdots + \frac{\sin(\mathbf{N} \cdot 2\pi f_0 \cdot t)}{N^2}\right]$$

The following Wikipedia links show the Fourier Series equations for the square wave and triangle wave and also have the option to play the sound of the waveforms with different numbers of terms:

https://en.wikipedia.org/wiki/Square_wave
https://en.wikipedia.org/wiki/Triangle_wave

Being able to create plots like those shown in Figure 2.3 and 2.4 in MATLAB®(or another software package) is a very important skill for engineers. The student is exposed to MATLAB®in many Sophomore, Junior, and Senior level courses and is not explicitly explained here, beyond the references in examples, as shown. Reviewing this Appendix would be beneficial before proceeding to Example 2.6.

Example 2.6 *In a similar fashion to how Figures 2.3 and 2.4 were created, write a MATLAB®program that displays square wave and triangle wave approximations as different numbers of terms of the Fourier series are used. Try to do this on your own before looking at the solution below. The results are shown in Figure 2.5*

Solution

```
% Solution - Plotting a square wave and triangle wave from sinusoids
N = 10; % Number of sinusoidal terms
fo = 60; %Fundamental Frequency
= 1/fo; %Period
A = 1; % Amplitude
Num_T = 3; %Number of periods to plot
Num_pts_per_period = 500; % # of points per period
tfinal = Num_T*T;
dt = T/Num_pts_per_period;
t = 0:dt:tfinal;
sq = 0*t; % This is the square wave signal
tr=sq; % This is the triangle wave signal
for j=1:N-1
k = 2*j-1; %This is the harmonic for the square wave
k_t = 2*(j-1)+1; %This is the harmonic for the triangle wave
```

```
sq = sq+(A*4/pi)*sin(2*pi*fo*t*k)/k;
tr = tr+(-1)ĵ*(A*8/pi2̂)*sin(2*pi*fo*t*k_t)/(k_t2̂);
end
% If you want to display multiple trials you can add
% a subplot command here.
plot(t,sq,'k')
hold % This allows the triangle plot to overlay the square wave plot
plot(t,tr,'r')
hold off
% Manually change the 10 to match the N that is used
grid on, title ('10 sinusoidal terms')
% The axis command is used to display the waveforms properly.
axis([0, tfinal, -A-0.2*A, A+0.2*A])
% The following figure shows four trials of this program and the
% graphs displayed using a 2 x 2 subplot
```

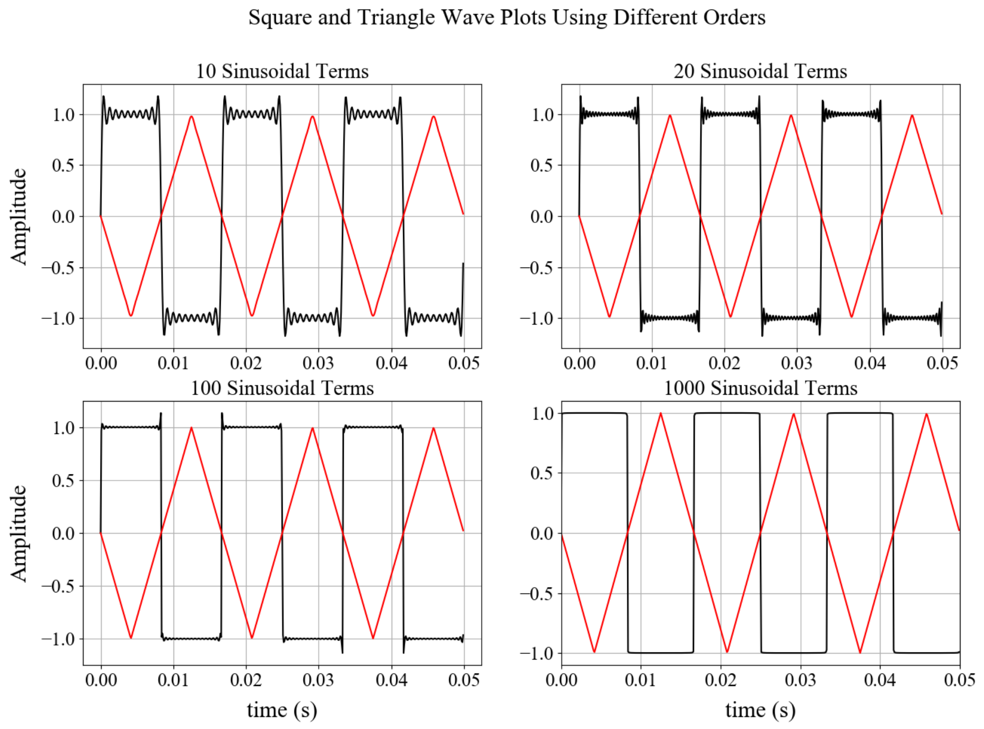


Figure 2.5. ***Example 2.6 Square and Triangle Waves:*** *By adding more sinusoids, the square wave approximation is improved. The triangle wave is approximated very well even with a low number of harmonics.*

While the Fourier Series has been shown to create periodic signals from a sum of sinusoids, the Fourier Transform can be used in a more complicated fashion to create either periodic or aperiodic signals. It can be shown that the set of complex exponentials of form $e^{j\omega_k t}$ creates an orthogonal basis set for $\omega_k \in \mathbb{R}$ (i.e. the dot product of $< e^{j\omega_k t}, e^{j\omega_l t} >= 0 \forall k, l \in \mathbb{Z}, k \neq l$). When generating a time-domain signal from a sum of weighted sinusoids, the weight of each sinusoid is determined by taking the dot product of the original time-domain signal, $x(t)$, with each basis vector. The specific definition is given by:

$$X(\omega) \quad = \quad \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$

This is the well-known Fourier Transform that determines the magnitude of each contributing sinusoid to the signal $x(t)$. It should be noted that $X(\omega) \in \mathbb{C}$, so the *magnitude* is found by:

$$|X(\omega)| \quad = \quad [X(\omega) \cdot X^*(\omega)]^{\frac{1}{2}}$$

The Fourier Transform allows us to represent a signal in the frequency domain. This is an important finding, because when we want to filter signals, we design filters based on their *frequency response* so it is essential to have a mathematical tool that allows us to characterize a signal (or system, for that matter) in the frequency domain.

So, how does all this Fourier mumbo-jumbo tie into sampling theory (prepare yourself for a mathematical journey in the following pages)? Sampling rate and frequency are related, so we wish to assess the Fourier Transform of a discretized (sampled) signal. When we take the Fourier Transform of a discrete signal, we must frame the dot product in a discrete format. Specifically, we write the Discrete Time Fourier Transform as:

$$X\left(e^{j\omega}\right) \quad = \quad \sum_{k=-\infty}^{+\infty} x[n]e^{-j\omega k} \tag{2.9}$$

Consider the periodicity of the term $e^{j\omega k}$. For any given frequency, ω_0, we know that $e^{j(\omega_0+2\pi k)n} = e^{j\omega_0 n}$. This is simply a mathematical notation that states that the complex exponential is a sinusoidal signal that is 2π-periodic. If we apply this to Equation 2.9 above, we arrive at:

$$\begin{aligned}
X\left(e^{j(\omega+m2\pi)}\right) &= \sum_{k=-\infty}^{+\infty} x[n]e^{-j(\omega+m2\pi)k} \qquad m \in \mathbb{Z} \\
&= \sum_{k=-\infty}^{+\infty} x[n]e^{-j\omega k}e^{-jm2\pi k} \\
&= \sum_{k=-\infty}^{+\infty} x[n]e^{-j\omega k}\left(e^{-jm2\pi}\right)^k \\
&= \sum_{k=-\infty}^{+\infty} x[n]e^{-j\omega k}\,(1)^k \\
&= \sum_{k=-\infty}^{+\infty} x[n]e^{-j\omega k}
\end{aligned}$$

This is a powerful result, because it tells us that the Discrete Time Fourier Transform (DTFT) is completely defined on an interval of 2π. In fact, it is common in a *Signals and Systems* textbook to show that the DTFT repeats itself every 2π, though the basic interval is taken to be $-\pi$ to π (rather than 0 to 2π). This is shown for some notional DTFT in Figure 2.6.

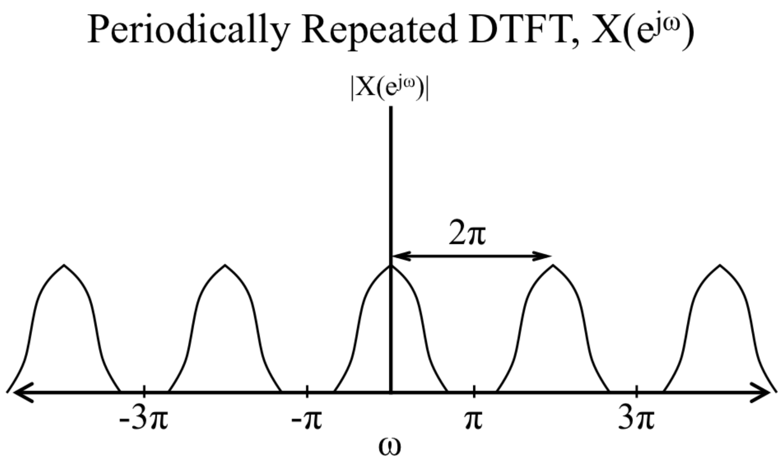


Figure 2.6. ***Periodicity of the DTFT:*** *The basic frequency spectrum of a discrete signal is completely defined on the interval from $-\pi$ to π. Since the basis vectors are periodic themselves, the base spectrum repeats itself on every 2π interval.*

We can view this in another way to understand why the repeated spectra might overlap and cause ambiguity. If we create a pulse train on some fixed interval, we can write this as:

$$p(t) = \sum_{k=-\infty}^{+\infty} \delta\,(t - kT)$$

where T is the *sampling period*. If we sample some signal, $x(t)$ at sampling interval of T, we can represent the sampled signal as:

$$\begin{aligned} x_s(t) &= x(t) \cdot p(t) \\ &= x(t) \cdot \sum_{k=-\infty}^{+\infty} \delta\,(t - kT) \\ &= \sum_{k=-\infty}^{+\infty} x(kT)\delta\,(t - kT) \end{aligned}$$

We relate this to frequency content by looking at the result in the Fourier domain. However, the careful reader will note that the process is a multiplication in the time domain, which is represented by a *convolution* in the frequency domain (which is a more difficult mathematical calculation)! Indeed, using the duality property, it can be seen that:

$$\begin{aligned} x_s(t) &= x(t) \cdot p(t) \\ &\Downarrow \quad \mathcal{F} \qquad \textit{compute Fourier Transform of both sides} \\ X_s(j\omega) &= \frac{1}{2\pi} X(j\omega) * P(j\omega) \\ &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(j\theta) P\,(j(\omega - \theta))\, d\theta \end{aligned}$$

We do not know the specifics of our notional signal, $x(t)$, but we do know that the Fourier Transform of a pulse train is given by:

$$\begin{aligned} P(j\omega) &= \frac{2\pi}{T_s} \sum_{k=-\infty}^{+\infty} \delta\,(\omega - k\omega_s) \\ &= \frac{2\pi}{T_s} \sum_{k=-\infty}^{+\infty} \delta\left(\omega - \frac{2\pi k}{T_s}\right) \end{aligned}$$

Then rewriting the convolution integral, we have:

$$
\begin{aligned}
X_s(j\omega) &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(j\theta)\ \left[\frac{2\pi}{T_s}\sum_k \delta(\omega-\theta_s)\right]\ d\theta \\
&= \frac{1}{T_s}\sum_k X\left(j(\omega - k\omega_s)\right)
\end{aligned}
$$

The student should recognize that this is simply the base Fourier Transform of $x(t)$, namely $X(j\omega)$ shifted by $k\omega_s$ for all $k \in \mathbb{Z}$. Applying this result to the periodic set of spectra shown in Figure 2.6, the periods can be couched in terms of the sampling frequency, $\omega_s = \frac{1}{T_s}$. This is shown in Figure 2.7 below. This figure also indicates the maximum and minimum frequency components of the actual signal, $x(t)$.

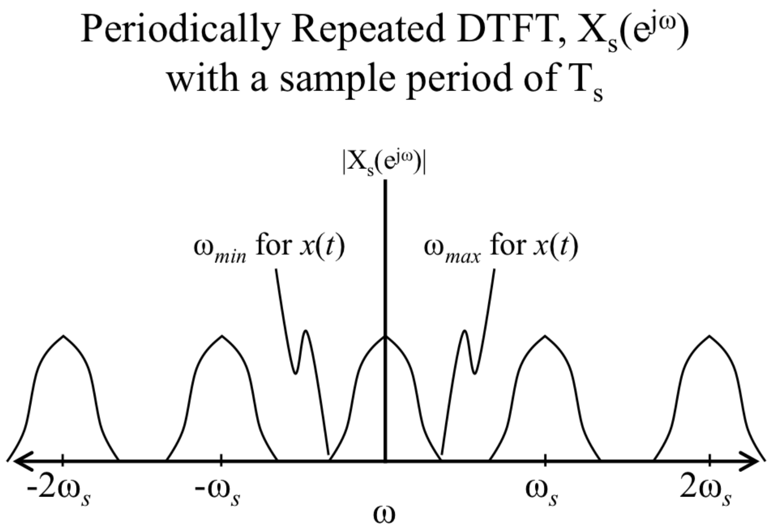

Periodically Repeated DTFT, $X_s(e^{j\omega})$
with a sample period of T_s
$|X_s(e^{j\omega})|$
ω_{min} for $x(t)$
ω_{max} for $x(t)$
$-2\omega_s$
$-\omega_s$
ω_s
$2\omega_s$
ω

Figure 2.7. ***DTFT of a Sampled Signal:*** *For a sampled signal, the centers of the periodically repeated spectra occur at intervals of ω_s. This is the rad/s sampling rate associated with the sample interval T_s The maximum and minimum frequencies associated with the original signal (that is sampled) are shown. Note that the minimum of the next repeated version of the spectrum is greater than the maximum of the previous spectrum, so that there is no overlap.*

What happens if the actual frequency content in the signal, $x(t)$, exceeds that of the sampling rate? If this occurs, then there is overlap between versions of the spectrum, creating a zone of ambiguity. This is shown in Figure 2.8. In this case, the higher frequencies of the base spectrum overlap with the lower negative frequencies of the following repeated spectrum. The ambiguity arises in that we do not know if the represented frequency is the negative reflection of a higher frequency, or the positive version of lower frequency. When this occurs, it is referred to as *aliasing*. Essentially, we *think* that the frequency is one value, but is it actually a different value. Hopefully it is clear that, because the difference between spectrum centers is ω_s, the sampling rate must ensure that $\omega_s > 2\omega_{max}$ to ensure that there is no overlap between the periodic representations of $X(e^{j\omega})$. This leads us directly to the Nyquist Sampling criterion, which states that, in order to accurately represent a signal via sampling, we must sample the signal at a sampling rate greater than twice the maximum frequency content of the signal. In other words, if f_{max} is the highest frequency component in the signal $x(t)$, then the sampling rate must be greater than twice f_{max}. In practical, and succinct, terms (accounting for the real world and converting from rad/s):

$$\omega_s > 2\omega_{max} \qquad or \qquad f_s > 2f_{max}$$

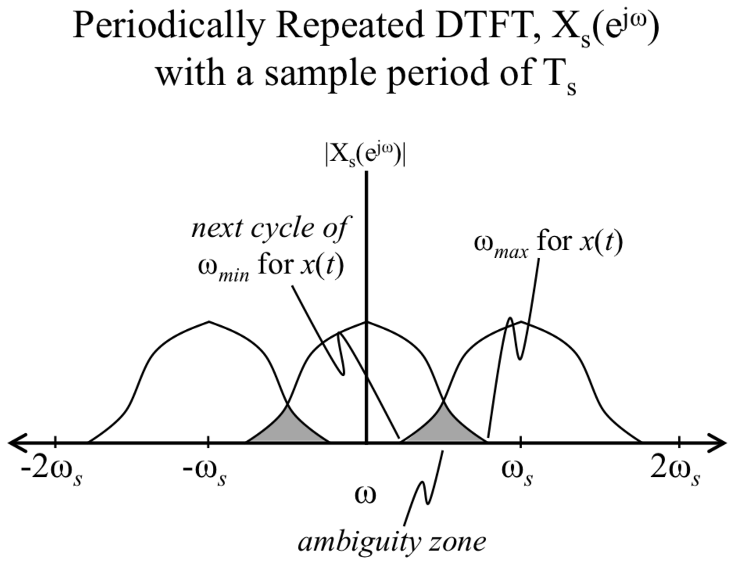

Periodically Repeated DTFT, $X_s(e^{j\omega})$
with a sample period of T_s
$|X_s(e^{j\omega})|$
next cycle of
ω_{min} for $x(t)$
ω_{max} for $x(t)$
$-2\omega_s$
$-\omega_s$
ω
ω_s
$2\omega_s$
ambiguity zone

Figure 2.8. ***DTFT of a Under-Sampled Signal:*** *When the sampling interval is too long (i.e. the sampling frequency is too low), there is an overlap zone between the periodic versions of the spectrum of $x(t)$. This is called* aliasing.

One can determine what the aliased frequency will be for a given sinusoidal input that is under sampled. This is given by the following algorithm:

$$N = mod\left(int\left(\frac{f_{signal}}{f_s/2}\right), 2\right)$$

$$\Downarrow$$

$$f_{alias} = N\frac{f_s}{2} + (-1)^N\, mod\left(f_{signal}, f_s/2\right)$$

where *mod*() is the modular function (taken with 2 here), *int* fixes the division result to be an integer (truncates the fractional portion), f_{signal} is the input sinusoid, and f_s is the sampling frequency.

Example 2.6 *Assume that a data acquisition system is sampling the input at 1000 samples per second (1000 Hz). For the following input frequencies, determine the perceived alias frequency: 700 Hz, 1200 Hz, 3400 Hz. (Note: These frequencies are all greater than 500 Hz (i.e. $f_s/2$), so aliasing will occur).*

Solution

Computing each step for the 1200 Hz input, we get:

$$
\begin{aligned}
int\left(\frac{1200}{1000/2}\right) &= 2 \\
&\Downarrow \\
N &= mod\,(2,2) \\
&= 0 \\
&\Downarrow \\
f_{alias} &= 0 \cdot 500 + (-1)^0 mod\,(1200, 1000/2) \\
&= 200
\end{aligned}
$$

Applying the same process to the 700 Hz and 3400 Hz inputs, we can determine that

Input Frequency	*Alias (Observed) Frequency*
700	300
1200	200
3400	400

The reader may note that the *alias frequency* in the table, in the presence of under-sampling, is the frequency the user observes out of the DAQ device, and is not the true frequency (thus the term *alias*).

When we discuss signal conditioning in Chapter 3, we will specifically cover the *anti-aliasing* filter. However, this concept should not be mysterious. To prevent aliasing, we simply need to make sure that no frequency content greater than $f_s/2$ is present in the signal. In other words, the anti-aliasing filter is simply a lowpass filter that ensures that the signal frequency content is less than 1/2 of the sampling frequency.

2.3 Digital-to-Analog Output

Analog-to-digital conversion for data acquisition is a primary component of computer-based instrumentation and measurement. However, a closely related process is creating analog output voltages from discrete data stored or generated in the computer. By definition, we assume that the Nyquist sampling requirement ($f_s > 2f_{max}$) is met. But the digital system can only output a discrete set of voltages, and the output voltage is generally held constant until the next sampling cycle. This quantization produces a stair-step signal that would not be pleasing in an audio or visual application. This is solved by passing the digital output signal through a lowpass filter to smooth out the stairsteps. Indeed, the lowpass filter is chosen based on the sampling criterion. For a given digital sample rate at the output, we know that the analog signal being represented can contain no frequency greater than one half of the sampling rate. Therefore, the lowpass filter is designed to block frequencies above $f_s/2$ and it should be evident that the frequency content in the step transitions we want to remove is much higher than $f_s/2$. Figure 2.9 shows the sampled voltage value in steps, as generated by the first step of the digital-to-analog conversion process. The figure also shows the lowpass-filtered output that can be considered the final output of the conversion process.

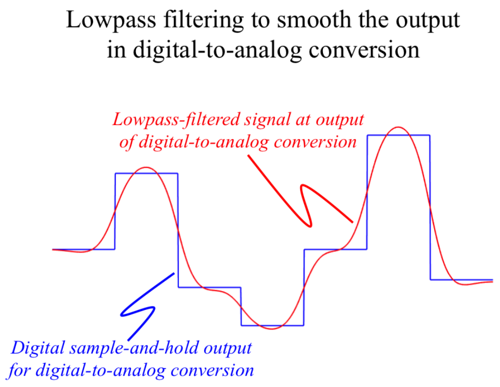

Lowpass filtering to smooth the output
in digital-to-analog conversion
Lowpass-filtered signal at output
of digital-to-analog conversion
Digital sample-and-hold output
for digital-to-analog conversion

Figure 2.9. ***Digital-to-analog conversion with lowpass filter:*** *During digital-to-analog conversion, the computer outputs a discrete voltage based on the binary digits (n-bit conversion) and holds that value until the next sample cycle. This is shown in blue. The sampled discrete signal is then filtered with a lowpass filter before final output (shown in red).*

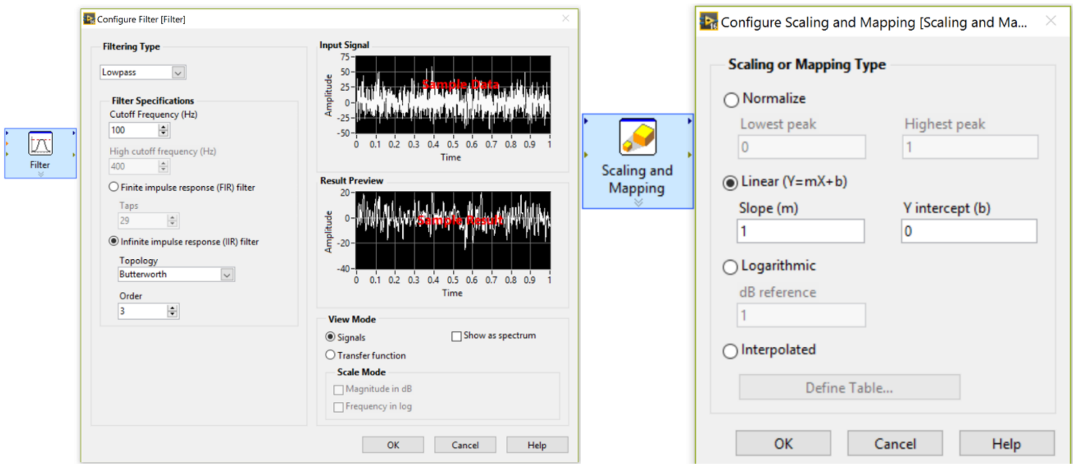
2.4 Conclusion

The two fundamental parameters to be addressed in data acquisition are *dynamic range* and *sampling rate*. These should be considered a minimum set of parameters to verify prior to data acquisition. If these two foundational concepts have not been addressed, the data should be considered hopelessly useless. Indeed, once these two items have been adequately accounted for, we have merely reduced the ambiguity associated with data acquisition to its minimum. But we must realize that ambiguity (*aka* uncertainty) still exists. Another key to minimizing the uncertainty regards signal conditioning. Signal conditioning can occur in the analog world, as well as the digital world. Because this text is concerned with measurement and instrumentation, we will confine ourselves to a discussion of the analog world. Specifically, we often need to condition (e.g. filter, amplify, etc.) the native analog signal in order to adequately digitize it. The next chapter discusses the concepts of signal conditioning.

Chapter 3

3.1 Signal Conditioning

To connect a measurand to a data acquisition system the signal usually needs to be *conditioned*. By this, we mean that we need to alter it in ways that ensure that the *information* conveyed by the signal remains present, but the signal itself is more amenable to detection and digitization. The most commonly needed types of signal conditioning are amplification and filtering. Recall that we showed in Figure 1.1 that signal conditioning can either be done in the analog or digital domain. There are many software-based solutions to accomplish amplification and filtering in the digital domain. One option is to use the Express Virtual Instruments (VIs) inside of LabVIEW as shown in Figure 3.1. When performing data acquisition in LabVIEW, as described in section A.2.8, using the Express VIs shown below are frequently used for digital signal conditioning purposes.


Configure Filter [Filter]

Filtering Type: Lowpass

Filter Specifications
Cutoff Frequency (Hz): 100
High cutoff frequency (Hz): 400
○ Finite impulse response (FIR) filter
Taps: 29
● Infinite impulse response (IIR) filter
Topology: Butterworth
Order: 3

View Mode
● Signals
○ Transfer function
☐ Show as spectrum
Scale Mode
☐ Magnitude in dB
☐ Frequency in log

Configure Scaling and Mapping [Scaling and Ma...

Scaling or Mapping Type
○ Normalize
Lowest peak: 0
Highest peak: 1
● Linear (Y=mX+b)
Slope (m): 1
Y intercept (b): 0
○ Logarithmic
dB reference: 1
○ Interpolated

Figure 3.1. ***Digital Signal Conditioning Examples in LabVIEW:*** *The left screen shot is from the Filter Express VI and the right screen shot is from the Scaling and Mapping Express VI.*

Alternatively, analog signal conditioning occurs prior to the signal being digitized, and the amplification and filtering functions are implemented with electronic hardware. The following section shows a real-world system that relies heavily on hardware-based amplification and filtering to work. In addition to amplification and filtering, another form of analog signal conditioning involves converting a sensor's electrical output from one form to another. The two most common electrical conversions are resistance-to-voltage and current-to-voltage. These two types of signal conditioning will be discussed in Chapter 4 when sensors are covered.

3.2 ECG - A Practical Signal Conditioning Example

The heart generates an electrical signal that evinces itself on the body surface. We call the recording of this electrical activity the electrocardiogram (ECG). Hopefully, the reader recognizes that the ECG traces of heart activity that are commonly seen on the monitors of Hollywood shows have undergone significant signal conditioning. To begin with, amplification is a necessity because the ECG signal is produced by processing the signals coming from multiple electrodes that are positioned around the body and these signals have amplitudes on the order of microvolts. Filtering is also needed to remove the unwanted signals that make it harder to interpret the information received from the electrodes. The ECG signal conditioning system typically consists of multiple sub-systems, as shown in Figure 3.2. The first stage is generally used when the target signal source is extremely small and the instrumentation amplifier (discussed in section 3.5.2) is the preferred method to amplify this small signal. The next two stages are filters, which will be covered in section 3.5.4. The last stage is where most of the amplification (i.e. voltage gain) occurs and it can be accomplished using many different types of amplifier configurations. Section 3.5 will begin with providing an overview of these amplifier configurations before moving on to the more complicated topics of instrumentation amplifiers and filters. The primary goal of this chapter is to explain the various sub-systems in Figure 3.2 in more detail and provide a better understanding of how amplification and filtering can be applied in any measurement system. First, however, the issue of noise will be discussed more thoroughly.

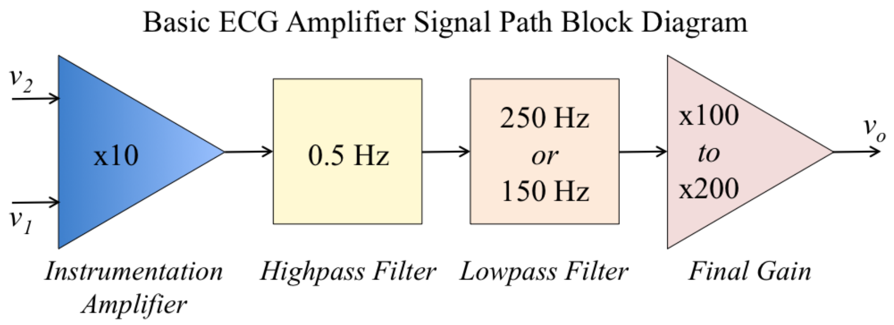


Figure 3.2. ***ECG amplifier block diagram:*** *In the ECG amplifier, the first stage is typically an instrumentation amplifier. This is followed by a highpass filter and then a lowpass filter. The lowpass block indicates a choice between 150 Hz cutoff or 250 Hz cutoff, depending on sampling and desired resolution. Large gain stages are often saved until the end so that no unwanted frequencies are amplified, making them harder to filter out.*

3.3 Noise and the Common Mode Rejection Ratio

Before introducing amplifiers, the concept of *noise* needs to be addressed. By the term noise, we mean in this text *any signal value that does not represent the actual signal-of-interest*. As mentioned in Chapter 1, measurements never represent cosmic truth. The deviance between the sampled value and the actual true value is the error, and we generally consider the error to be the noise component of a signal. The errors that contribute to noise can be in the environment, as well as the measurement system itself. For mathematical modeling purposes it is beneficial to break up the incoming signal, $x(t)$, into different terms, as follows:

$$x(t) \quad = \quad s(t) + \eta(t)$$

where $s(t)$ is the desired signal (the signal-of-interest), and $\eta(t)$ is the time-changing noise. In Figure 3.3 the blue plot shows a recorded ECG signal with noise, and the red plot shows what we *think* is the *true* ECG.

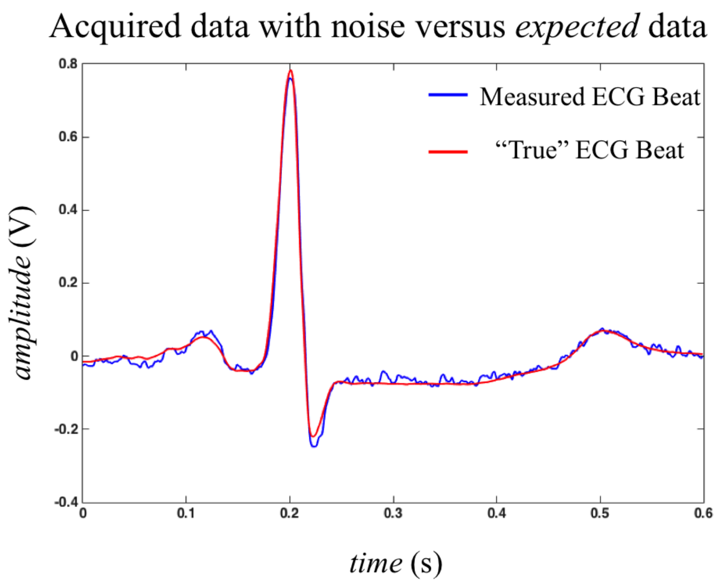


Figure 3.3. ***Measured ECG with noise:*** *Acquired (measured) ECG signal, $x(t)$, with measurement noise is shown in blue. The red trace is the* expected *ECG signal, $x(t)$.*

While one might expect that filtering is the only way to remove noise, the way the amplifier is configured and designed also plays a large role in noise reduction. There is a metric called the Common Mode Rejection Ratio (CMRR) that helps to quantify how well an amplifier rejects noise. Some amplifier configurations are better at noise reduction than others, as we will discover next.

3.4 Amplifier Concepts

In most cases, the primary purpose of an amplifier is to supply *gain*. Gain, G, is a term in electronics that is usually defined as the output voltage divided by the input voltage ($G = V_{out}/V_{in}$). While the gain term is occasionally used in reference to a current gain, the term is predominantly used in conjunction with voltage gain and that will be the case in this book. Amplifiers can be configured as either *single-ended* or *differential*. Figure 3.4 shows a simplified diagram of these two types of amplifiers. More details of these amplifiers will be described in the following sections.

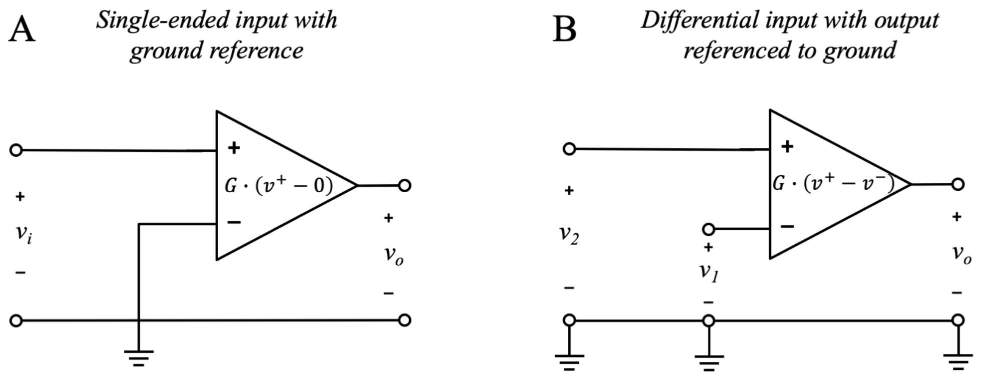

A
Single-ended input with ground reference
$G \cdot (v^+ - 0)$
v_i
v_o
B
Differential input with output referenced to ground
$G \cdot (v^+ - v^-)$
v_2
v_1
v_o

Figure 3.4. ***Single-ended versus Differential Amplifiers:*** *The single-ended amplifier assumes that the incoming signal is a single signal referenced to the same system ground as the amplifier. On the other hand, the differential amplifier assumes two separate input signals, each referenced to ground.*

In a basic Electronics course, the student learns about *inverting* and *non-inverting* amplifier configurations. Each of these configurations (see section 3.5.1 for more details) are considered *single-ended* amplifiers. In the single-ended amplifier, a signal, $s_i(t)$, is perturbed by some noise, $\eta(t)$, such that the signal is represented as $v_i(t) = s_i(t) + \eta(t)$. Since the secondary terminal (- is the inverting terminal and + is the non-inverting terminal) is connected directly to system ground, the resulting output, $v_o(t)$, is the input signal, $v_i(t)$, multiplied by the gain, G, of the amplifier, as shown in the equation below. Note that the noise, $\eta(t)$, is amplified along with the desired signal, $s_i(t)$. Therefore, the signal-to-noise ratio (SNR) of the output is no better than the SNR of the input signal. In summary, noise being amplified is a major drawback of the single-ended amplifier as denoted by the red term in the equation below.

Single-ended Amplifier:

$$\begin{aligned} v_o(t) &= G \cdot v_i(t) \\ &= G \cdot [(s_i(t) + \eta(t))] \\ &= G \cdot s_i(t) + G \cdot \eta(t) \end{aligned} \tag{3.1}$$

Conversely, in a differential amplifier, we do not ground one of the terminals, but have two separate inputs, $v_1(t)$ and $v_2(t)$, that are assumed to have the same noise, $\eta(t)$. In this case, the resulting output, $v_o(t)$, is the gain, G, of the amplifier multiplied by the difference between the two inputs. If these inputs are: $v_1(t) = s_1(t) + \eta(t)$ and $v_2(t) = s_2(t) + \eta(t)$, then the gain equation is:

Differential amplifier:

$$\begin{aligned} v_o(t) &= G \cdot [v_2(t) - v_1(t)] \\ &= G \cdot [s_2(t) + \eta(t) - s_1(t) - \eta(t)] \\ &= G \cdot [s_2(t) - s_1(t)] \end{aligned} \tag{3.2}$$

Note that the differential output voltage equation implies the complete rejection of all noise associated with the inputs. This is unrealistic in real systems and will be discussed in more detail in Section 3.5.2. While the noise is not completely removed with the differential amplifier, most of it is and the majority of the output signal is simply the gain, G, multiplied by the difference between the signals of interest, $s_2(t)$ and $s_1(t)$. The goal of the differential amplifier, then, is to maximize the signal and minimize the effect of the noise. The ECG amplifier works in precisely this manner. In the classic ECG amplifier, the ground point is taken as the right leg. At least two other voltage points are recorded; for example, the right arm and the left arm. The amplifier takes the single-ended signal of the left arm, with respect to ground, and subtracts it from the single-ended signal of the right arm, with respect to ground. Indeed, the *instrumentation amplifier* of Figure 3.2 is a differential amplifier made up of multiple op-amps to achieve the noise-reduction goal (see Section 3.5.3).

3.5 Amplifiers Circuits

An amplifier is most commonly implemented with an integrated circuit (IC) called an Operational Amplifier (or Op Amp). Op Amps have very large input resistance, small output resistance, and an internal (or Open Loop) gain that is also very large. Op Amp circuits can be designed with simple gain equations (Equations 3.1, 3.2, and 3.3) when they are considered to be *ideal*. There are several primary assumptions needed for an Op Amp to be considered ideal. These are: Input resistance = ∞, output resistance = 0, internal (or open loop) gain = ∞, the Op Amp is capable of sourcing infinite current, and the Common Mode Rejection Ratio (CMRR) is infinite. The last three of these assumptions are related to the two most common situations that can produce major inaccuracies with the ideal Op Amp model.

1. When the output of the Op Amp is connected to a load with a very small resistance the Op Amp cannot supply enough current to produce the voltage ($V_o = I \cdot R_L$) across the load resistor that is specified by the ideal gain equations. The load of the Op Amp is a term that describes the resistance connected between the output of the Op Amp and ground. The load resistance can either be a physical resistor or a component that acts like a resistor (e.g. 8Ω speaker). Example 3.5 at the end of this section demonstrates this situation.
2. When the signal going into the Op Amp has a high frequency the open loop gain decreases and causes inaccuracies in the ideal gain equations. Example 3.6 at the end of this section demonstrates this situation.
3. If the *calculated* CMRR is higher than the value in the data sheet then the ideal equations will not hold (see Example 3.7).

Other than examples 3.5 and 3.6 we will assume all Op Amps are ideal in this book. For more information about the ideal assumptions and other details of Op Amps see Module 7 of the following book: https://shareok.org/handle/11244/51946.

3.5.1 Inverting and Non-Inverting Single-Ended Amplifiers

The single-ended version of the Op Amp comes in two basic configurations: inverting and non-inverting. Figures 3.5 and 3.6 show how these amplifiers are configured and the output voltage equations. Recall that the output voltage is equal to the gain multiplied by the input voltage.

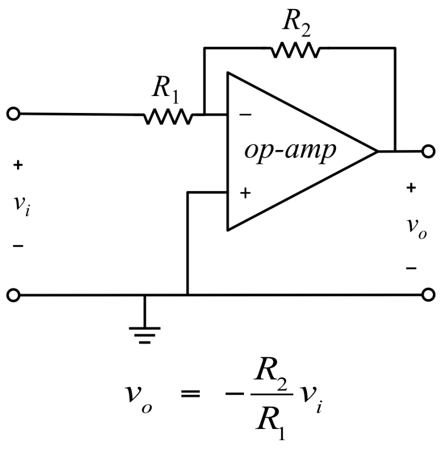


Figure 3.5. ***Inverting Op-Amp configuration:*** *In the inverting configuration the input is applied to the* negative *terminal of the op-amp, and the positive terminal is connected to ground. Thus the output is the subtraction of* $0 - V_{in}$ *and will be negative if* V_{in} *is positive and positive if* V_{in} *is negative.*

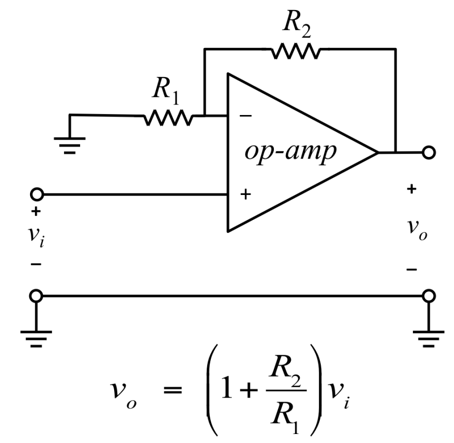


Figure 3.6. ***Non-Inverting Op-Amp configuration:*** *In the non-inverting configuration the input is connected to the positive input terminal of the Op Amp. Under the assumption of negligible current flow through* R_1*, the output is dependent on* $V_{in} - 0$.

In Figures 3.5 and 3.6, only the input and output pins of the Op Amps are shown for simplicity. In reality, Op Amps are *active* devices, so they also require positive and negative supply (or rail) voltages, ±Vcc, in order to function. Some Op Amp ICs can operate with a *single supply*, where the negative supply is connected to ground. The rail voltages are critical to how the Op Amp circuit functions because they specify the limits of the output voltage, as demonstrated in the following example.

Example 3.1 *Determine the output voltage of an Op Amp, that is assumed to be ideal, for the following cases.*

a) Non-inverting configuration with ±5V rails, 1V DC input, R_1 = 1 kΩ, and R_2 = 3.3 kΩ.

b) Non-inverting configuration with ±12V rails, 5V DC input, R_1 = 1 kΩ and R_2 = 10 kΩ.

c) Non-inverting configuration with a single 5V rail, 2V DC input, R_1 = 10 kΩ and R_2 = 1 kΩ.

d) Inverting configuration with ±12V rails, 2V DC input, R_1 = 1 kΩ and R_2 = 4.7 kΩ.

e) Inverting configuration with ±24V rails, 3V DC input, R_1 = 1 kΩ and R_2 = 10 kΩ.

f) Inverting configuration with ±5V rails, 2V DC input, R_1 = 10 kΩ and R_2 = 1 kΩ.

g) Inverting configuration with a single 5V rail, 2V DC input, R_1 = 10 kΩ and R_2 = 4.7 kΩ.

Solutions

a) Non-inverting configuration: ±5V rails, 1V DC input, R_1 = 1 kΩ, R_2 = 3.3 kΩ:

$$G = 1 + \frac{3.3}{1} = 4.3 \text{ V/V}$$
$$V_o = 1 \text{ V} \cdot 4.3 \text{ V/V} = \mathbf{4.3\ V}$$

(4.3 V is < 5 V rail, so it won't be clipped)

b) Non-inverting configuration: ±12V rails, 5V DC input, R_1 = 1 kΩ, R_2 = 10 kΩ:

$$G = 1 + \frac{10}{1} = 11 \text{ V/V}$$
$$V_o = 5 \text{ V} \cdot 11 \text{ V/V} = \mathbf{55\ V}$$

(55 V is > 12 V rail, so it will be clipped to **12 V**)

c) Non-inverting configuration: single 5V rail, 2V DC input, $R_1 = 10\ \text{k}\Omega$, $R_2 = 1\ \text{k}\Omega$: The 5V single rail means the positive supply is 5V and the negative supply is grounded.

$$G = 1 + \frac{1}{10} = 1.1\ \text{V/V}$$

(Note that the minimum gain for a non-inverting Op-amp is 1 V/V)

$$V_o = 2\ \text{V} \cdot 1.1\ \text{V/V} = \mathbf{2.2\ V}$$

(2.2 V is < 5 V rail, so it won't be clipped)

d) Inverting configuration: ±12V rails, 2V DC input, $R_1 = 1\ \text{k}\Omega$, $R_2 = 4.7\ \text{k}\Omega$:

$$G = \frac{-4.7}{1} = -4.7\ \text{V/V}$$

$$V_o = 2\ \text{V} \cdot -4.7\ \text{V/V} = \mathbf{-9.4\ V}$$

(-9.4 V is > -12 V rail, so it won't be clipped)

e) Inverting configuration: ±24V rails, 3V DC input, $R_1 = 1\ \text{k}\Omega$, $R_2 = 10\ \text{k}\Omega$:

$$G = \frac{-10}{1} = -10\ \text{V/V}$$

$$V_o = 3\ \text{V} \cdot -10\ \text{V/V} = \mathbf{-30\ V}$$

(-30 V is < -24 V rail, so it will be clipped to **-24 V**)

f) Inverting configuration: ±5V rails, 2V DC input, $R_1 = 10\ \text{k}\Omega$, $R_2 = 1\ \text{k}\Omega$:

$$G = \frac{-1}{10} = -0.1\ \text{V/V}$$

$$V_o = 2\ \text{V} \cdot -0.1\ \text{V/V} = \mathbf{-0.2\ V}$$

(-0.2 V is > -5 V rail, so it won't be clipped)

This last scenario shows that, unlike non-inverting Op Amps that have a minimum gain of 1 V/V, inverting Op Amps can be used to produce fractional gains that reduce (or attenuate) the input voltage instead of amplifying it.

g) Inverting configuration: single 5V rail, 2V DC input, R_1 = 10 kΩ, R_2 = 4.7 kΩ:

$$G = \frac{-4.7}{10} = -0.47 \text{ V/V}$$
$$V_o = 2 \text{ V} \cdot -0.47 \text{ V/V} = \mathbf{-0.94 \text{ V}}$$

(-0.94 V is < ground–the lower rail–so it will be clipped to **0 V**)

When choosing between these configurations for a specific application, the non-inverting configuration is a better choice when R_1 isn't much greater than the *source resistance*, R_s. R_s is the resistance between the input voltage source and R_1. In the inverting configuration the source resistance, R_s, is in series with R_1 so the denominator of the gain equation increases, and the overall gain is reduced (Recall: $G = -R_2/R_1$ for an inverting Op Amp). For the non-inverting configuration, the gain doesn't change when connected to a source resistance because any resistance connected between the input voltage source and the + terminal of the Op Amp has no effect on the gain. Example 3.2 demonstrates this concept further using a simulation software called Multisim.

Example 3.2 *If a 1 Volt DC source is connected to an inverting Op Amp with a 50 Ω cable determine the output voltage. For this example, the following resistors are used: R_1 = 100 Ω, R_2 = 200 Ω, and R_L = 1 kΩ. Also, determine by how much percentage the output voltage is reduced due to the 50 Ω cable.*

<u>Solution</u>

The following Multisim schematic shows the simulation of this circuit along with the calculations. If the source resistance, R_s, was reduced to 0 Ω then the gain would be: -200/100 = -2 V/V and the output voltage would be equal to: $V_{out} = G \cdot V_i = -2 \cdot 1 = -2$V. Therefore, the 50 Ω cable in the circuit below results in a V_{out} reduction of: (2 - 1.33)/2 = 33.5%.

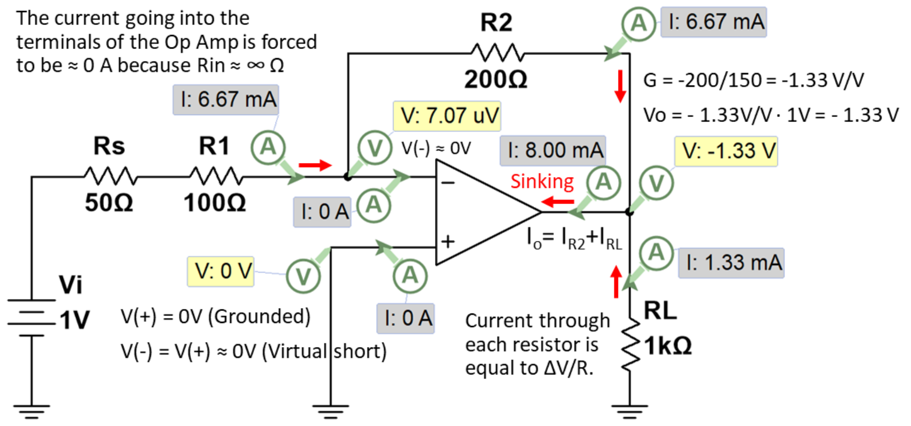

The current going into the terminals of the Op Amp is forced to be ≈ 0 A because Rin ≈ ∞ Ω
R2
200Ω
I: 6.67 mA
G = -200/150 = -1.33 V/V
Vo = - 1.33V/V · 1V = - 1.33 V
I: 6.67 mA
V: 7.07 uV
Rs
R1
V(-) ≈ 0V
I: 8.00 mA
V: -1.33 V
50Ω
100Ω
Sinking
I: 0 A
Io= IR2+IRL
I: 1.33 mA
Vi
V: 0 V
1V
V(+) = 0V (Grounded)
V(-) = V(+) ≈ 0V (Virtual short)
I: 0 A
Current through each resistor is equal to ΔV/R.
RL
1kΩ

Figure 3.7. ***Inverting Op-Amp Circuit for Example 3.2:*** *The inverting configuration has the disadvantage that the load resistance may be of a similar magnitude as the source resistance, which can significantly affect the results.*

Notes from Example 3.2: For the inverting Op Amp, the output current, I_o, flows into the Op Amp. This is called *sinking*. The magnitude of I_o is determined using Kirchhoff's Current Law (KCL), which states the current going into a node or branch (I_{R2} and I_{RL}) is equal to the current going out of a node or branch (I_o). Each individual current through a resistor can be calculated using Ohm's Law ($I_R = \Delta V/R$). For example, the current flowing through resistor R_2 is equal to (0 - 1.33)/200 = -0.00667 A = -6.67 mA.

The virtual short is another important concept for Op Amp. The virtual short simply means that the voltages on the + and - terminals of the Op Amp amplifier are forced to be approximately equal (i.e. $V(-) = V(+)$) when there is a feedback wire or resistor between the output and the V(-) pin. In Figure 3.7 the resistor $R2$ is used as feedback. If $R2$ were not included in that circuit the Op Amp would act like a comparator and the output would either go to the negative rail. When an Op Amp is used as a comparator (i.e. no feedback is added) the output goes to the negative rail when V(-) > V(+) and it goes to the positive rail when V(+) = V(-). The virtual short equation can be used to solve for the currents and voltages in the Op Amp circuit without even using the gain equation, as shown below:

- The voltage at the node above V_i is equal to 1V.
- Voltage at $V(-) = V(+) =$ 0V. The current through both R_s and R_1 equals (1-0)/150 = 6.67 mA.
- Since no current can flow into either Op Amp terminal I_{R1} is forced to be equal to I_{R2}.
- $V_o = V(-) - V_{R2} = V(-) - I_{R2} \cdot R_2 = 0 - 0.00667\text{A} \cdot 200\Omega = \mathbf{-1.33\ V}$.

For help with this type of basic circuit analysis see Module 1 of the following book: https://shareok.org/handle/11244/52245

Example 3.3 *If a 1 Volt DC source is connected to a non-inverting Op Amp with a 50 Ω cable determine the output voltage. For this example, use the same resistor values as those used in Example 3.2.*

Solution

The following Multisim schematic shows the simulation of this circuit along with the calculations.

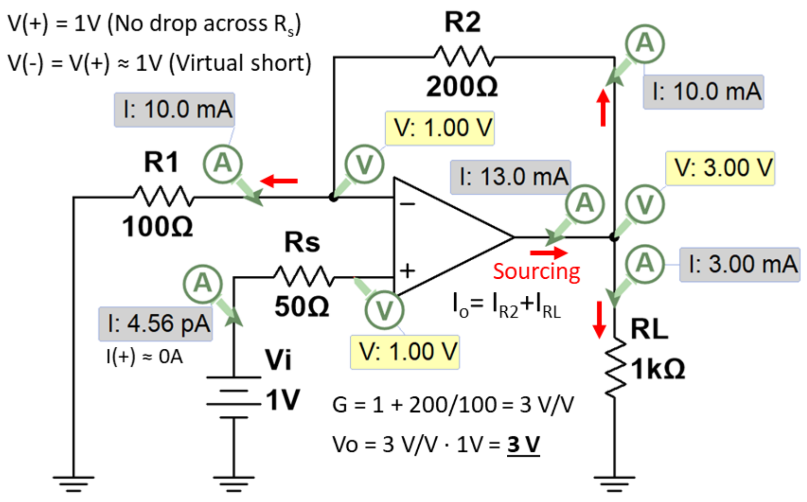


Figure 3.8. ***Inverting Op-Amp Circuit for Example 3.3:*** *In the non-inverting configuration, the input goes directly to the Op Amp positive terminal, which has a resistance on the order of Mega-Ohms, or even Giga-Ohms, so the source resistance is nearly irrelevant.*

Example 3.3 shows how the 50 Ω source resistance has no noticeable effect on the circuit, which is much different than what was seen in Example 3.2 where V_{out} was reduced by 33.5%.

Examples 3.1 through 3.3 showed how Op Amps amplify DC inputs, but AC signals are also frequently amplified with Op Amps. Example 3.4 shows an AC signal that is amplified by both an inverting and a non-inverting amplifier. With AC, the signal's peak-to-peak voltage will be used for the probes instead of the DC voltage. The peak-to-peak voltage of a symmetric sinusoid is equal to twice the amplitude. For example, a 120 V_{rms}, 60 Hz sine wave from a standard AC wall outlet has the following equation:

$$v(t) = 120\sqrt{2} \cdot \sin(2 \cdot \pi \cdot 60 \cdot t)$$

Note: The Amplitude (or peak voltage of a symmetrical sine wave) is equal to the RMS voltage multiplied by $\sqrt{2}$ (e.g. 120 V_{rms} = 169.7 $\mathrm{V_{pk}}$). The peak-to-peak voltage is equal to $2 \cdot 169.7 = 339.4$ V. This signal is plotted in Figure 3.9. This is how this signal would look if it were plotted on an oscilloscope.

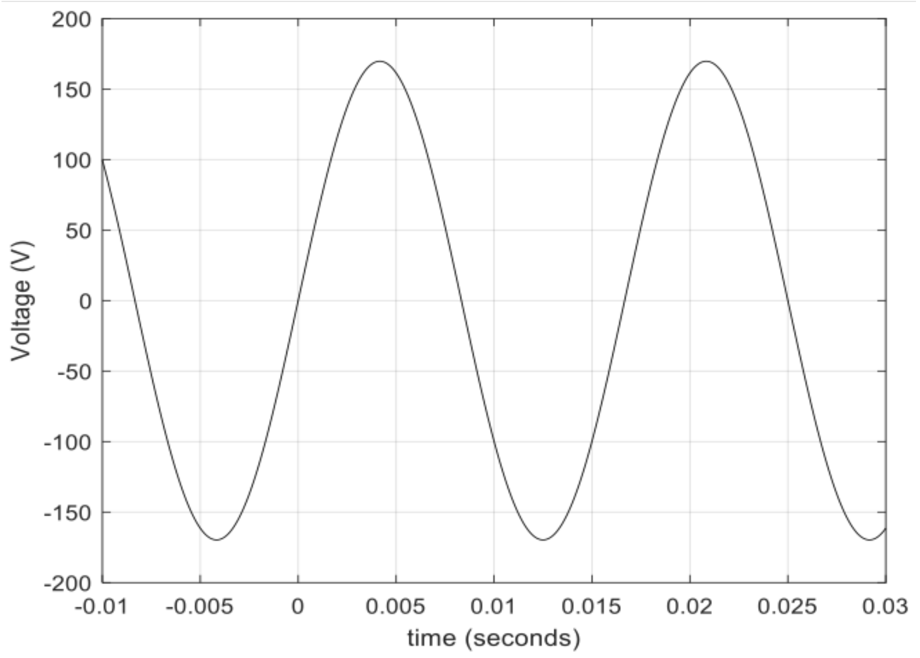


Figure 3.9. ***Standard AC outlet signal in U.S. homes (120 V_{rms}, 60 Hz sinusoid)*** *The RMS value does* not *represent peak value for an RMS representation. RMS is used with sinusoidal signals that are zero-mean as a way to indicate something related to a mean value. RMS takes the square of all signal values, computes the mean of the squared values, and then takes the square of the mean value.*

It should be noted that the 120 V_{rms} signal coming from the AC wall outlet is not typically connected directly to Op Amps due to power rating and voltage rail limitations. For this book, the LM324 Op Amp will be frequently referenced and the datasheet for the Texas Instruments version of this IC can be see at the following link: (`http://www.ti.com/lit/ds/symlink/lm324-n.pdf`). The LM324 is a general-purpose Op Amp that has a limit of ±32V for its voltage rails and around 1 Watt for its power dissipation limit. While specialized power electronic devices can be used to handle higher voltages, step down transformers are usually the preferred method of converting the 120 V_{rms} signal to lower levels that are compatible with standard electronic components, such as the LM324.

Example 3.4 *Design an Op Amp circuit that contains one inverting Op Amp and one non-inverting Op Amp that converts a standard wall AC outlet signal to two sinusoids that have peak-to-peak voltages of 15 V_{pp} and 50 V_{pp}. To allow general purpose electronic components for this design assume a 10:1 step-down transformer is used to reduce the 120 V_{rms} AC wall outlet signal to 12 V_{rms}. It is desired that only E12 series 10% tolerance standard resistor values can be used (see* www.resistorguide.com *for more details) and the output voltage needs to be within 1% of the desired output.*

Solution

After the 10:1 step-down transformer, the peak-to-peak voltage of the signal is 33.94 V_{pp}.

- Only the inverting amplifier can have a gain less than 1 V/V so it must be used to produce the 15 V_{pp} output. The gain needs to be equal to: $G = 15/33.94 = 0.442$ V/V
- The ideal gain equation of an inverting amplifier is equal to: $G = -R_2/R_1$
- Standard resistor values that would produce a ratio close to this are $R_2 = 1.2$ kΩ and $R_1 = 2.7$ kΩ. The peak-to-peak output voltage expected from this ratio is equal to: $V_{o,pp} = 33.94\text{V}_{\text{pp}} \cdot \|(-1.2/2.7)\|$ V/V = $\mathbf{15.08\ V_{pp}}$. Since the peak to peak voltage is always a positive value the absolute value of the gain was used in the output voltage calculation.
- The non-inverting amplifier cannot have a gain less than 1 V/V so it must be used to produce the 50 V_{pp} signal since the design specifies using only one inverting configuration and one non-inverting configuration. The gain needs to be equal to: $G = 50/33.94 = 1.473$ V/V
- The ideal gain equation of a non-inverting amplifier is equal to: $G = 1 + R_2/R_1$
- If $1 + R_2/R_1$ is set equal to 1.473 the resistor ratio R_2/R_1 is equal to 0.473. The closest standard resistor values that would produce a ratio close to this are: $R_2 = 470\ \Omega$ and $R_1 = 1$ kΩ. The peak to-peak-output voltage expected from this ratio is equal to: $V_{o,pp} = 339.4\text{V}_{\text{pp}} \cdot (1 + 470/1000) = \mathbf{49.89\ V_{pp}}$

The probes in the following Multisim schematic show the expected output values match closely to the calculations and are within 1% of the desired values.

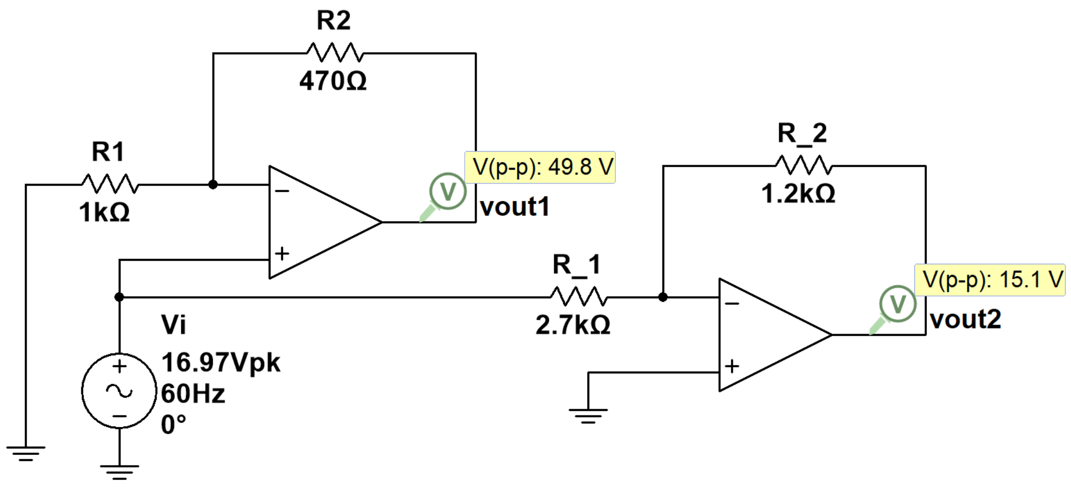


Figure 3.10. ***Circuit for Example 3.4 that uses 3 terminal virtual Op Amps in Multisim*** *Virtual Op Amps behave like ideal Op Amps in Multisim, so they can source the needed voltage or current regardless of the circuit design. Clearly, this does bot reflect reality, unless the user has exercised good circuit design.*

The following figure shows how to connect the Multisim Oscilloscopes to the circuit to show the plots. The colors can be changed on the wires that go into the (+) terminal of channels A and/or B on the oscilloscope to modify the color of the plot. The (-) terminal on the oscilloscope can be left unconnected if it is supposed to be grounded, because Multisim makes that assumption.

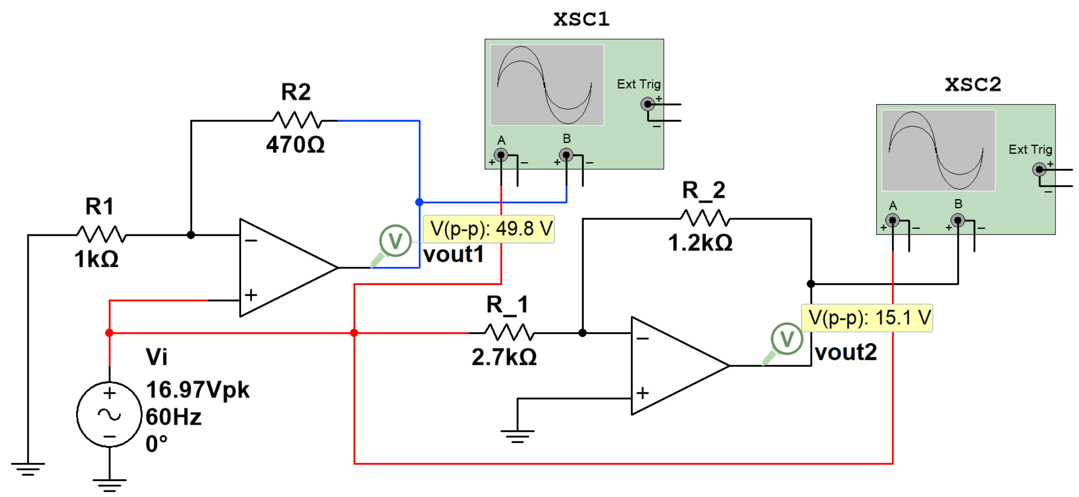
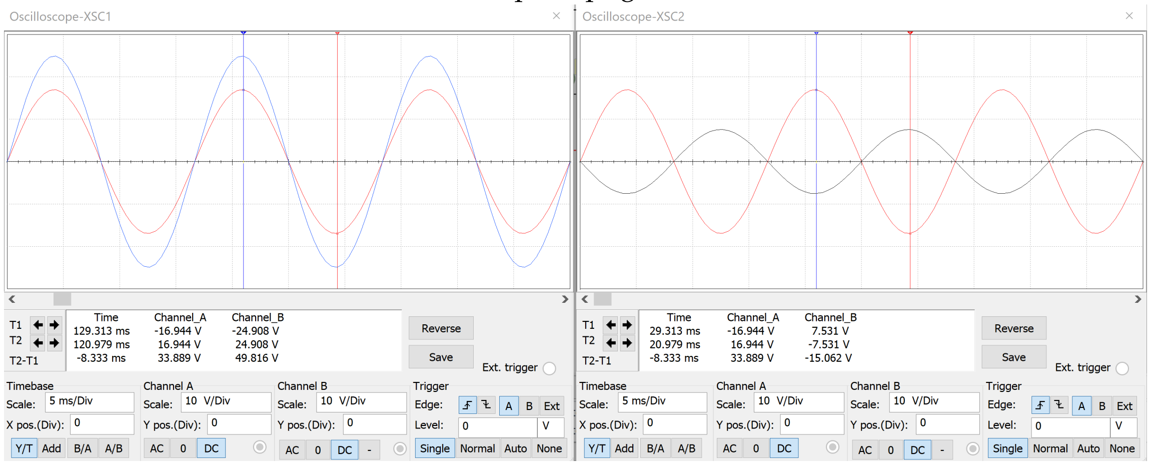


Figure 3.11. ***Circuit for Example 3.4 connected to Multisim 2-channel oscilloscopes*** *The virtual oscilloscope in Multisim is an excellent tool for predicting circuit behavior.*

The following shows the oscilloscope plots, XSC1 and XSC2, from the non-inverting and inverting Op Amp circuits, respectively. The oscilloscope cursors are used to measure the amplitude and half period. Notice that the inverting Op Amp flips the polarity of (or "inverts") the input signal. Notice in the scope plots that the absolute time values (T1 and T2) are different, but the differential time values (T2-T1) are the same for each instance. Only the differential time (i.e. difference of time between the two cursors) should be used in Multisim.

plots.png



Oscilloscope-XSC1

```
       Time         Channel_A    Channel_B
T1     129.313 ms   -16.944 V    -24.908 V
T2     120.979 ms    16.944 V     24.908 V
T2-T1  -8.333 ms     33.889 V     49.816 V
```

Timebase: Scale: 5 ms/Div; X pos.(Div): 0; Y/T
Channel A: Scale: 10 V/Div; Y pos.(Div): 0; DC
Channel B: Scale: 10 V/Div; Y pos.(Div): 0; DC
Trigger: Edge: A; Level: 0 V; Single

Oscilloscope-XSC2

```
       Time         Channel_A    Channel_B
T1     29.313 ms    -16.944 V     7.531 V
T2     20.979 ms     16.944 V    -7.531 V
T2-T1  -8.333 ms     33.889 V    -15.062 V
```

Timebase: Scale: 5 ms/Div; X pos.(Div): 0; Y/T
Channel A: Scale: 10 V/Div; Y pos.(Div): 0; DC
Channel B: Scale: 10 V/Div; Y pos.(Div): 0; DC
Trigger: Edge: A; Level: 0 V; Single

Figure 3.12. ***Multisim oscilloscope plot (Left = Non-inverting output, Right = Inverting output)*** *In the left panel the non-inverting circuit provides a small gain of the input signal, and matches the sign of the incoming signal. The right panel shows the inverting amplifier that attenuates the output, and also causes the output to be the* negative *of the input.*

All Multisim circuits in the previous example use the 3-terminal virtual Op Amp. These components are simplified Op Amp models that allow several internal parameters to be adjusted such as: Input Resistance, Output Resistance, Open Loop Gain, and the level of the voltage rails. The default voltage supply values of the 3 terminal Op Amp are ±12 V, which would only allow a 24 V_{pp} sine wave to come out of this Op Amp, so the levels had to be increased for this simulation. Multisim also has actual Op Amp ICs that can be simulated. Figure 3.13 shows the same circuit from Example 3.4 implemented with one LM324 quad Op Amp.

Note: The LM324 has 4 Op Amps inside the chip, but only two are used in this example. All 4 Op Amps inside the LM324 must use the same voltage rails so only one $\pm V_{cc}$ is set and a red x is shown for all other voltage rails. It also should be noticed that Multisim requires the negative voltage rail to be placed on the top (where the inverting (-) input is) and this is the opposite from the standard notation, where the positive voltage rail is on top and the negative voltage rail is on bottom.

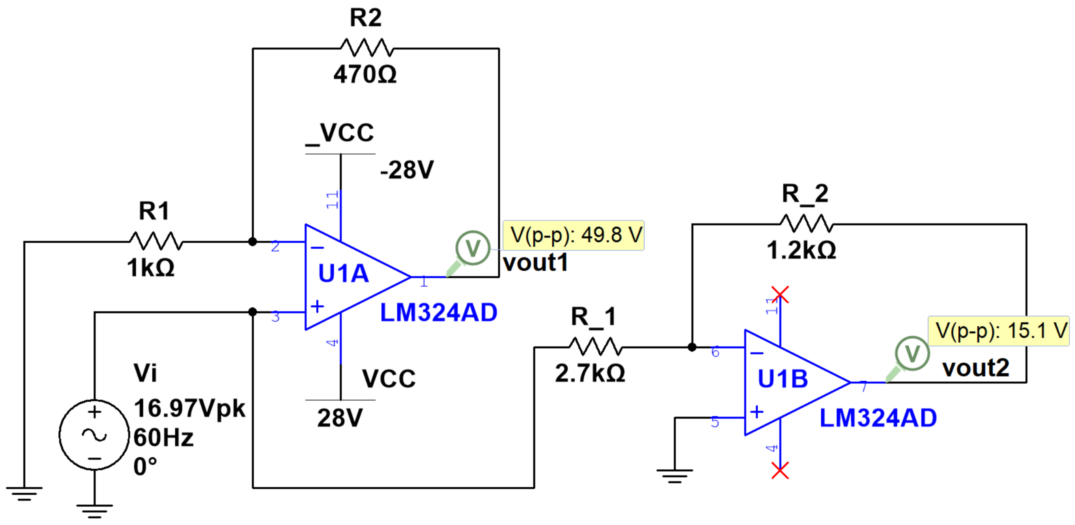


Figure 3.13. ***Circuit for Example 3.4 that uses LM324 Op Amps in Multisim*** *When we use actual integrated circuit (IC) models in Multisim, the models attempt to account for real-world issues live power supply, etc, so the output attempts to reflect real-world results if the circuit is not designed properly.*

Before moving on to differential amplifiers the two most common situations that produce major inaccuracies with the ideal Op Amp model, that were discussed at the beginning of this section, will be demonstrated. The first situation is when the output of the Op Amp is connected to a load with a very small resistance. This situation results in the Op Amp not being able to supply enough current to produce the voltage ($V_o = I \cdot R_L$) across the load resistor that is specified by the corresponding ideal gain equation ($V_o = -R_2/R_1 \cdot V_{in}$ for inverting and $V_o = 1 + R_2/R_1 \cdot V_{in}$ for non-inverting).

Example 3.5 *If a 10 Ω load resistor is added to the output of the inverting Op Amp in Example 3.4, show that the ideal gain equation (3.1) is not accurate when using an actual Op Amp like the LM324 but is accurate with the 3-terminal virtual Op Amp.*

Solution

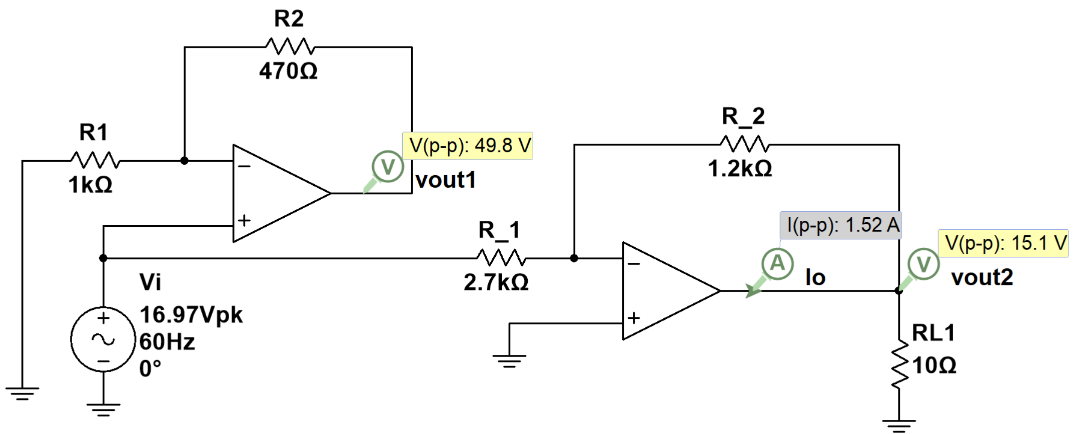


Figure 3.14. ***Ideal Op Amp Circuit for Example 3.4 with 10 Ω load:*** *With the 3-terminal virtual (ideal) Op Amps and a 10 Ω load resistor, the output voltage is unaffected by the load resistor in the ideal case. Note: The "output short circuit current" in the 3-terminal Multisim Op Amp has a default value of 25 mA. This value had to be increased for this circuit to simulate correctly to match the ideal gain equations. The short circuit current will be discussed in more detail later.*

The peak-to-peak output voltage is the same (15.1 V_{pp}) as in Example 3.4 where there was no load resistor included (i.e. $R_L = \infty$). The output of the Op Amp has a peak to peak voltage of 1.52 A. Another way of saying this is that the Op Amp needs to have a peak to peak output current of 1.52 A in order to supply R_L with enough current to have a voltage of 15.1 V_{pp} across R_L. Recall that if the ideal gain equations are satisfied, then $V_{o,pp} = V_{RL,pp} = 2 \cdot 16.97 \cdot \| - 1.2/2.7 \| = 15.1$ V_{pp}. A 1.52 peak-to-peak current is much higher than most general-purpose Op Amps can supply. The following figure shows that when the 3-terminal virtual Op Amp (that forces the ideal assumptions to be maintained) is replaced with an LM324, then the ideal gain equations are no longer accurate.

In Figure 3.15 the LM324 Op Amp is supplying as much current as possible, as determined from the internal modeling of the chip done by Multisim, but a peak-to-peak current of 51.8 mA is not sufficient to allow the output voltage to reach the 15.1 V_{pp} level specified by the ideal gain equations. In order for 15.1 V_{pp} to be across the 10 Ω load, the current through R_L would need to be equal to 1.51 A ($I_{RL} = \Delta V/R_L = 15.1/10$) and in this case the Op Amp is incapable of supplying that much current. The maximum current that an output can produce is difficult to know for certain, but the datasheet provides some clues. For the LM324, tests are shown in the datasheet with different supply voltages and loads used, and the maximum current ranges from ±20 mA to ±40 mA. The largest current the Op Amp is capable of producing is called the short circuit current, which is the output current of the Op Amp if the output pin is connected directly to ground (i.e. $R_L = 0\ \Omega$).

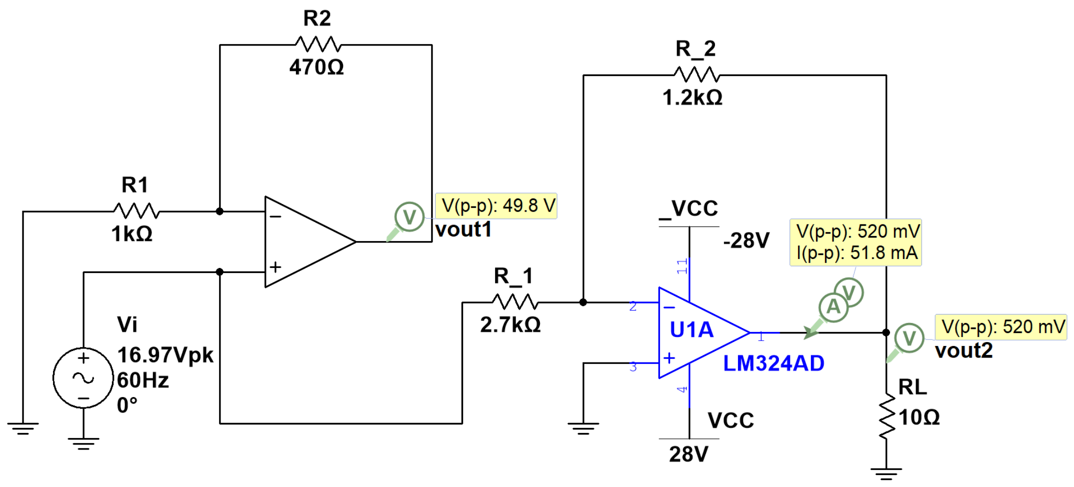


Figure 3.15. ***Modeled LM324 Op Amp Circuit for Example 3.4 with 10 Ω load:*** *The LM324 model represents an actual Op Amp and cannot source the current required to provide the theoretical voltage output across the 10 Ω load.*

In this example using the LM324, R_L was manually increased in Multisim for this circuit and it was found iteratively that R_L had to be increased to a little over 200 Ω before the peak-to-peak output voltage reached 15.1 V_{pp}. In addition to having the output voltage reduced, the Op Amp's life cycle will also be reduced when it is operated at its maximum current.

Example 3.5 demonstrated the danger of connecting a small load to an Op Amp. If high currents are needed, connecting the output of the Op Amp to one or more power transistors configured as a Class A, B, or AB output stage would solve the problem. Transistor amplifiers are outside the scope of this book, but more information can be found in [reference 7].

The second common situation, that results in the ideal gain equations being inaccurate, is when the signal going into the Op Amp has a high frequency. Example 3.6 demonstrates this situation. For details of how the gain is changed when the frequency is increased refer to Module 7 of the following book: `https://shareok.org/handle/11244/51946`

Example 3.6 *Increase the 60 Hz signal in Example 3.4 to 60 kHz to show that the ideal gain equation (3.1) is not accurate when using an actual Op Amp like the LM324 but is accurate with* *the 3-terminal virtual Op Amp.*

Solution

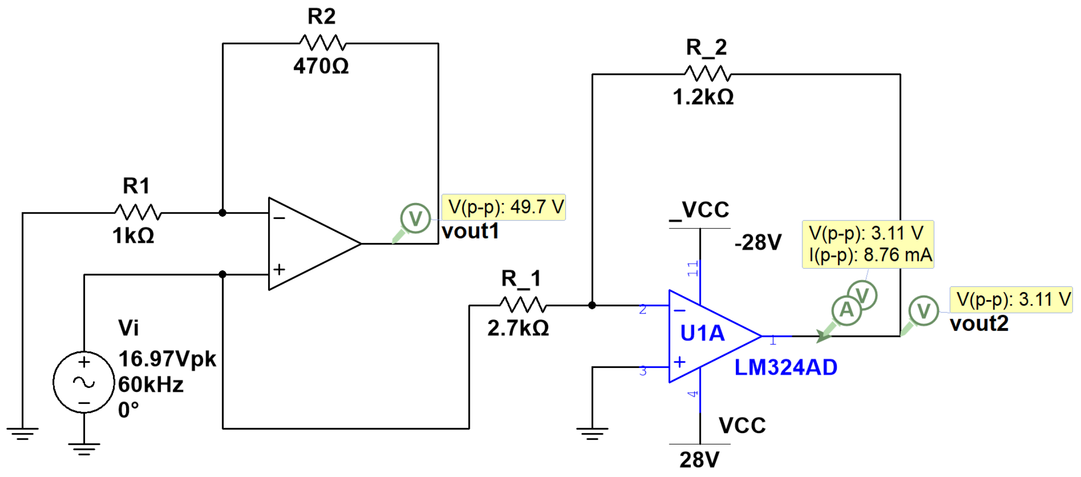


Figure 3.16. ***Ideal Op Amp versus LM324 at High Frequency:*** *The ideal Op Amp on the left of the figure behaves according to the ideal gain equation but the LM324 model shows that the real device has frequency limits, so the amplitude of the output is reduced to a value much lower than it was in Figure 3.15 (15.1 V) because the ideal gain equation for inverting Op Amps is no longer valid.*

As the frequency is increased in this example the gain in the LM324 is greatly reduced from the 15.1 V $_{pp}$ level it was at in example 3.4, but the virtual Op Amp remains at close to the same as the 49.98 V_{pp} level that is produced from the ideal gain equation for an inverting Op Amp (Recall: $V_{o,pp}$ = 339.4 V_{pp} $\cdot(1 + 470/1000)$ = 49.89 V_{pp}). Therefore, when using Op Amps for signal conditioning the frequency of the input signal is an important factor to consider for Op Amp selection. The Gain Bandwidth Product (GPB) is the most common frequency-related parameter to be listed on the datasheet and selecting an Op Amp with a higher GPB will reduce the gain reduction effects that are demonstrated in Example 3.6.

3.5.2 Differential Amplifier

As previously stated, the differential (or difference) amplifier has a signal component at the input of both Op Amp input terminals. Typically, the signal of interest will be transmitted on one of the inputs and nothing will be transmitted on the other input. The extra input is used so that noise that is picked up along the two transmission lines (i.e. wires) will be common for both inputs. In general, since the output of an Op Amp is given by $\hat{G} \cdot (v^+ - v^-)$, this tends to eliminate noise. Consider again two incoming signals that consist of:

$$\begin{aligned} v^-(t) &= s_1(t) + \eta(t) \\ v^+(t) &= s_2(t) + \eta(t) \end{aligned}$$

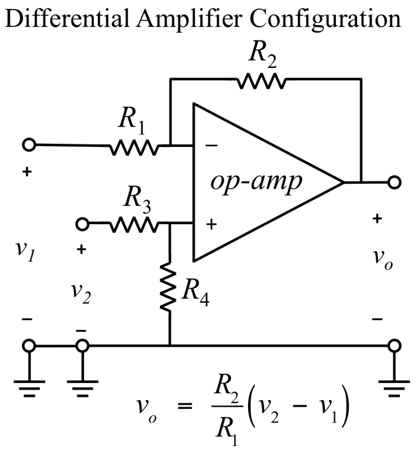
Then the output of a differential amplifier can be written as

$$\begin{aligned} v_o(t) &= \hat{G}\left(v^+(t) - v^-(t)\right) \\ &= \hat{G}\left[(s_2(t) + \eta(t)) - (s_1(t) + \eta(t))\right] \\ &= \hat{G}\left[(s_2(t) - s_1(t)) + (\eta(t) - \eta(t))\right] \\ &= \hat{G}\left(s_2(t) - s_1(t)\right) \end{aligned}$$

Clearly, this shows that mathematically noise cancels out in the differential mode. However, we know intuitively that the noise is never completely canceled out. This is another situation where reality is not the same as the ideal situations that we desire. Accounting for noise on the two inputs to vary slightly, we can re-write this as:

$$\begin{aligned} v^-(t) &= s_1(t) + \eta_1(t) \\ v^+(t) &= s_2(t) + \eta_2(t) \\ &\Downarrow \\ y(t) &= \hat{G}\left(v^+(t) - v^-(t)\right) \\ &= \hat{G}\left[(s_2(t) + \eta_2(t)) - (s_1(t) + \eta_1(t))\right] \\ &= \hat{G}\left[(s_2(t) - s_1(t)) + (\eta_2(t) - \eta_1(t))\right] \\ &= \hat{G}\left(s_2(t) - s_1(t)\right) \quad + \quad \hat{G}\left(\eta_2(t) - \eta_1(t)\right) \end{aligned} \tag{3.3}$$

In this case, we see that the additive noise is present, and has a gain of $\hat{G}$. However, if we assume that the additive noise is very similar between both the positive and negative inputs, then the difference, $\varepsilon(t) = \eta_2(t) - \eta_1(t)$ is generally much smaller than either $\eta_1(t)$ or $\eta_2(t)$. Moreover, $\varepsilon(t)$ is also much smaller than $s_2(t) - s_1(t)$. Because of the different gain magnitudes between the inverting and non-inverting configurations, the differential amplifier must balance the input signals in a realizable circuit. This can be achieved by the circuit shown in Figure 3.17, when the differential amplifier is *balanced* with: $R_2/R_1 = R_4/R_3$. Typically, balancing is achieved by setting R_2 equal to R_4 and R_1 equal to R_3, but these values can be different as long as the ratios (R_2/R_1 & R_4/R_3) are equal. When these resistor ratios are equal the amplifier is referred to as a balanced differential amplifier and has an output equation shown in Figure 3.17.



Differential Amplifier Configuration
R_2
R_1
R_3
R_4
op-amp
−
+
v_1
v_2
v_o
$v_o = \frac{R_2}{R_1}(v_2 - v_1)$

Figure 3.17. ***Differential amplifier configuration:*** *General configuration of a differential amplifier circuit. The R_4/R_3 resistor divider balances the magnitude of the non-inverting gain of $1+R_2/R_1$ with the magnitude of the inverting gain of R_2/R_1.*

The *differential input* of a differential amplifier is, then, considered to be $v_2 - v_1$ and the differential gain is equal to R_2/R_1 when the balanced resistor ratio ($R_2/R_1 = R_4/R_3$) is maintained. Notice that this is the same gain magnitude as the inverting single-ended amplifier, but it has a positive sign instead of a negative.

In addition to the differential input and gain there is also a common-mode input and gain in a differential amplifier. The common-mode input is the mean of v_2 and v_1 and the common-mode gain is shown in equation 3.7. Notice that the common-mode gain reduces to 0 when the balanced resistor ratio ($R_2/R_1 = R_4/R_3$) is maintained.

$$\text{Differential Input} = V_{i,diff} = v_2 - v_1 \tag{3.4}$$

$$\text{Differential Gain} = A_d = R_2/R_1 = R_4/R_3 \qquad \text{(If balanced: } R_2/R_1 = R_4/R_3\text{)} \tag{3.5}$$

$$\text{Common-Mode Input} = V_{i,CM} = (v_2 + v_1)/2 \tag{3.6}$$

$$\text{Common-Mode Gain} = A_{CM} = V_o/V_{i,CM} = [R_4/(R_3 + R_4)] \cdot [1 - (R_2/R_1)(R_3/R_4)] \tag{3.7}$$

While it is desired to maintain the balanced resistor ratio, it is impossible to have the ratio perfectly equal due to resistor tolerances. If the resistor ratios are not equal, then the differential gain will need to be solved using superposition, as shown in Example 3.7.

Example 3.7 *If a differential Op Amp is designed with $R_1 = 3$ kΩ, $R_2 = 9$ kΩ, $R_3 = 2$ kΩ, and $R_L = 1$ kΩ, determine the value of R_4 so that it is balanced? If the input voltages are $v_1 = 7$ V and $v_2 = 4$ V, what will the output voltage be equal to? Is the Op Amp sourcing or sinking current in this design? Solve this problem using the ideal gain equation for balanced differential Op Amps and also verify you get the same answer using superposition.*

Solution

To balance the differential Op Amp R_2/R_1 must equal R_4/R_3. This occurs when $R_4 = 6$ kΩ. The rest of the calculations are shown in Figure 3.18.

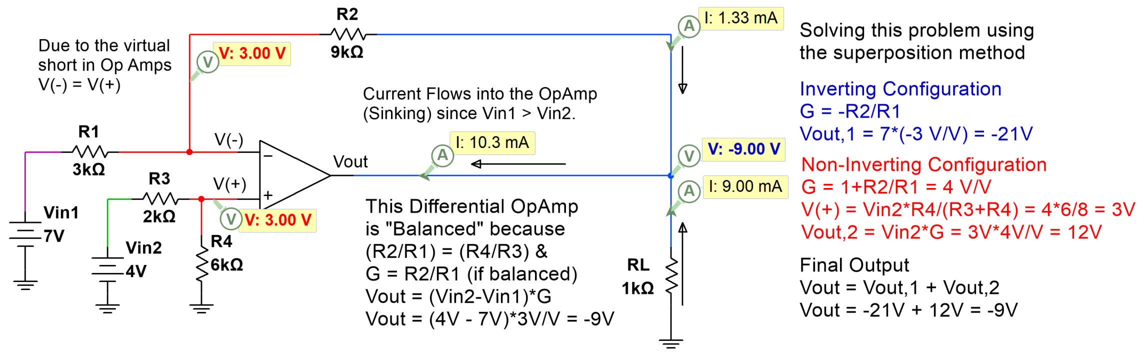


Figure 3.18. ***Differential amplifier used in Example 3.7:*** *The single Op Amp differential amplifier requires all the resistors to be precisely balanced to get the theoretical results.*

Equation 3.7 shows that the common-mode gain is no longer zero when the resistor ratio is not balanced. The ratio of the differential gain and the common-mode gain is called the Common-Mode Rejection Ratio (CMRR) and is used as a metric in determining how well an Op Amp can reject common-mode signals, while amplifying differential signals. The CMRR is defined by equation 3.8. This equation shows that as ACM approaches zero the CMRR approaches infinity, which is what we want because that means signals that are common to both input pins (i.e. noise) will be effectively removed.

$$\text{CMRR} = \frac{|A_d|}{A_{CM}} \tag{3.8}$$

The CMRR in equation 3.8 is given in absolute (or V/V) gain. Absolute gain is often converted to gain in decibels (dB) using equation 3.9. Conversely, equation 3.10 shows how to convert from gain in dB to gain in V/V.

$$\text{Gain (dB)} = 20 \cdot \log_{10}[\text{Gain(V/V)}] \tag{3.9}$$

$$\text{Gain (V/V)} = \frac{10^{\text{Gain (dB)}}}{20} \tag{3.10}$$

As previously explained, both inputs to the Op Amp have near equal noise signals, so noise is considered a common-mode signal and the common noise is mostly removed by the balanced differential Op Amp. However, when the noise is different on the two inputs, the difference of the noise, $\epsilon(t)$, will be amplified by the much larger differential gain instead of the common mode gain which is near zero. For this reason, extreme care needs to be taken to make sure the two transmission lines are close together and travel through the same environment. The length of the two lines is an equally important consideration because longer lines increase the probability that noise can be induced into the circuit.

Even if the noise on both lines were perfectly equal the common noise would still not be completely eliminated for three reasons:

1. **External resistor imbalance** – The four resistors used in the differential Op Amp configurations have tolerances, so it is impossible to achieve perfect balance ($R_2/R_1 = R_4/R_3$). As the ratios differ, equation 3.7 shows that A_{CM} becomes non-zero and therefore the CMRR becomes finite. Example 3.8 shows how resistor tolerances reduce the CMRR.

2. **Internal differences in the Op Amp** – Ideally, the Op Amp would have an infinite CMRR if the external resistors are balanced. But, the Op Amp has internal resistance and transconductance differences that result in a non-zero ACM, which results in a finite CMRR. For example, the CMRR of the LM324 is specified as 85 dB in the datasheet for DC signals (see link in previous section). This means that even if R_2/R_1 and R_4/R_3 are equal the best possible CMRR for this Op Amp is 85 dB. Additionally, the CMRR for Op Amps gets smaller at higher frequencies. Most datasheets will provide a plot that shows the expected CMRR versus frequency. Figure 6 of the datasheet for the LM324 shows that the CMRR begins dropping when the frequency exceeds ~10 kHz and drops approximately 20 dB/dec (i.e. ~65dB @ 100 kHz).

3. **Exceeding the common-mode input limits** – Obviously you can't perfectly eliminate a common noise signal that has an amplitude of 1 million volts. There are limits to the common-mode inputs that are published in the datasheet and if these limits are exceeded the published CMRR values will no longer be valid. The limit for the common mode input published in the datasheet for the LM324 is 2 Volts lower than the magnitude of the rail voltage that is used.

Example 3.8 *Assuming that there are no internal differences in the Op Amp that limit the CMRR and that the common-mode input is kept below the limits specified in the datasheet, what is the worst possible CMRR for Example 3.7 if the resistors have 10% tolerances? Also, what is the minimum value of the rail voltages that should be specified for this design if it was implemented with the LM324?*

Solution

According to equation 3.8 , the CMRR gets smaller (or worse) as A_{CM} gets larger. From equation 3.7, A_{CM} will get larger when R_2/R_1 gets smaller, so the worst-case scenario would be R_2 = 8.1 kΩ and R_1 = 3.3 kΩ. It is not as easy to see, but from analysis it can be determined that A_{CM} increases as R_3 decreases and R_4 increases. So, to obtain the largest A_{CM} (and the most imbalanced differential amplifier possible), R_3 would be 1.8 kΩ and R_4 would be 6.6 kΩ. Plugging these resistors into equation 3.7 yields:

$$\begin{aligned} A_{CM} &= [R_4/(R_3 + R_4)] \cdot [1 - (R_2/R_1)(R_3/R_4)] \\ &= [6.6/(1.8 + 6.6] \cdot [1 - (8.1/3.3)(1.8/6.6))] \\ &= 0.26 \text{ V/V} \end{aligned}$$

The differential gain in this case would be solved using superposition as follows:

From the inverting configuration:

$$\begin{aligned} G_{inv} &= -R_2/R_1 = -8.1/3.3 = -2.4545 \text{ V/V} \\ V_{o,1} &= V_{in,1} \cdot G_{inv} = 7V \cdot 2.4545 \text{ V/V} = -17.18\text{V} \end{aligned}$$

From the non-inverting configuration

$$G_{non-inv} = 1 + R_2/R_1 = 1 + 8.1/3.3 = 3.4545 \text{ V/V}$$
$$V(+) = V_{in,2} \cdot R_4/(R_3 + R_4) = 4 \cdot 6.6/(1.8 + 6.6) = 3.14 \text{ V}$$
$$V_{o,2} = V(+) \cdot G_{non-inv} = 3.14V \cdot 3.4545 \text{ V/V} = 10.86 \text{ V}$$

Combining both outputs yields:

$$V_o = V_{o,1} + V_{o,2} = -17.18 + 10.86 = -6.32 \text{ V}$$

(verified in Figure 3.19)

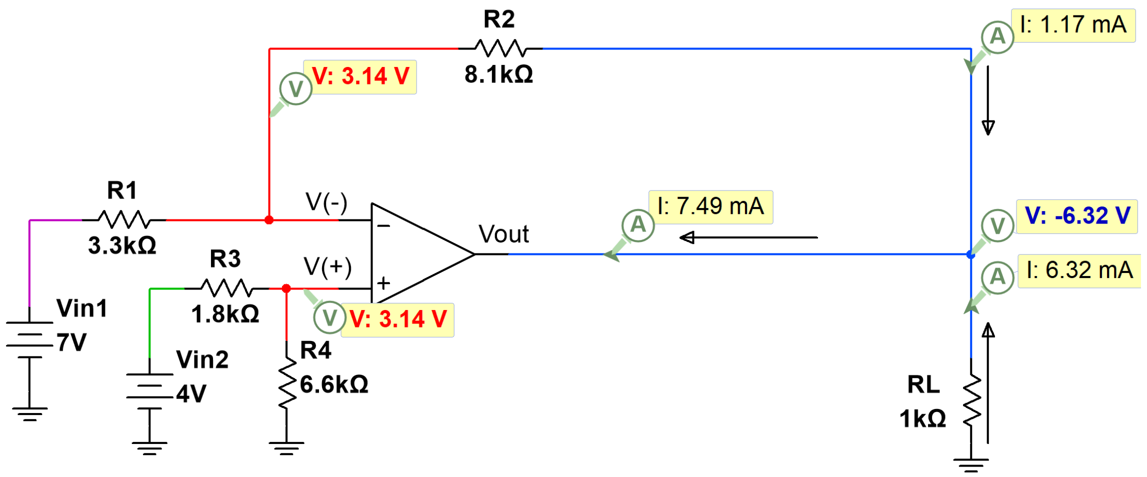


Figure 3.19. ***Differential amplifier used in Example 3.7 with resistors changed:*** *In this re-computation of Example 3.7, the resistors have been changed to show the worst possible effects of using 10% resistors.*

The differential input, v_{id} = 4 - 7 = -3 V

The differential gain, A_d = -6.32/-3 = 2.108 V/V

CMRR = $\frac{|A_d|}{|A_{CM}|}$ = 2.108/0.26 = 8.12 V/V or 18.2 dB

First, the minimum rail voltages need to be selected to avoid the common-mode input limits from being exceeded. From Equation 3.6:

$$V_{i,CM} = \frac{(V1 + V2)}{2} = \frac{(7 + 4)}{2} = 5.5 \text{ V}.$$

If we use a LM324 Op Amp we would need the rail voltages to be 2 Volts higher than the common-mode input, so we would select ±7.5 V rails. It should be noted that this Op

Amp technically only uses the negative rail since the output voltage is negative. However, the positive supply is also needed for it to work properly and symmetrical rails are almost always used for Op Amp designs.

The rail voltages should also be checked to make sure clipping won't occur. Since the output voltage (-6.32V) is greater than the negative rail voltage (-7.5V), the rail voltage would typically be considered acceptable. However, most Op Amps aren't capable of having output voltages that reach the rail voltages, so often times the rails need to be increased a volt or two higher than the largest expected output voltage magnitude. For this reason, it would be a good design choice to set the rails to at least ±8V. Another pitfall in selecting the rail voltages is selecting rails that are overly large to protect against potential clipping. However, the rail voltages should try to be kept only slightly larger than the expected output voltage to minimize power dissipation. If the maximum allowable rails of ±32V were used, then a lot of power would be dissipated by the Op Amp and this is a waste of energy and also could result in the power dissipation limit being exceeded for the Op Amp. The power dissipation limit for an IC will depend on the package type. For the LM324 the limits according to the datasheet are: 1130 mW for the Plastic Dual Inline Package (PDIP) through-hole chip and 800 mW for the Small Outline Integrated Circuit (SOIC) surface-mount chip.

Before moving on to instrumentation amplifiers, a differential amplifier circuit with added noise will be simulated to provide a deeper understanding of what is taking place when it is used for signal conditioning purposes.

Example 3.9 *Consider an accelerometer that is measuring a 10 Hz vibration signal that has a peak to peak voltage of 10 mV. The signal of interest is in the presence of 60 Hz noise and that noise produces a peak to peak voltage of 2 mV on the transmission line. Design a circuit to amplify the vibration signal so that its peak to peak voltage is 1V. Assume the output of the Op Amp connects directly to a DAQ that has a 100 kΩ pull down resistor. For comparison purposes try two different designs: 1) A non-inverting single-ended Op Amp and 2) A balanced differential Op Amp. Use* *the LM324 to implement both designs.*

Solution

The desired gain can be calculated by dividing the output by the input:

$$G = 1.1\text{V}/10\text{mV} = 110\text{V/V}$$

For the non-inverting Op Amp:

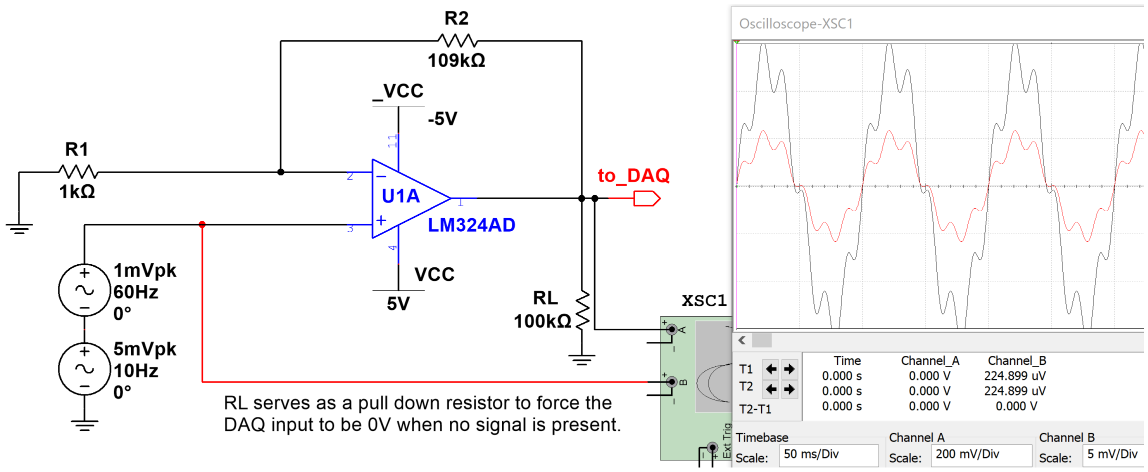
$$G = 1 + R_2/R_1 = 110, \text{ so } R_2/R_1 = 109$$

We will set $R_1 = 1$ kΩ and $R_2 = 109$ kΩ.
For the differential Op Amp

$$G = R_2/R_1 = R_4/R_3 = 110$$

We will set $R_1 = R_3 = 1$ kΩ and $R_2 = R_4 = 110$ kΩ.

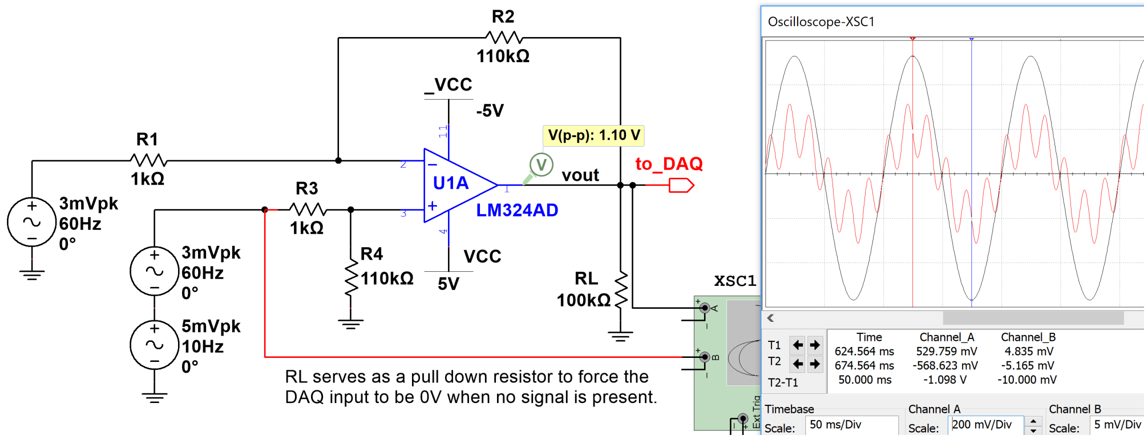
Figures 3.20 and 3.21 show the results from the non-inverting and differential designs respectively. The non-inverting design amplifies the noise and the signal at the same level, so the output voltage has the same signal to noise ratio as the input. However, the balanced differential amplifier amplifies the 10 Hz signal to a peak to peak voltage of 1.1V, but since the noise is common to both inputs it is removed, and no trace of the 60 Hz noise signal can be visibly noticed on the oscilloscope plot for v_{out}.



```
        Time       Channel_A   Channel_B
T1      0.000 s    0.000 V     224.899 uV
T2      0.000 s    0.000 V     224.899 uV
T2-T1   0.000 s    0.000 V     0.000 V

Timebase             Channel A              Channel B
Scale: 50 ms/Div     Scale: 200 mV/Div      Scale: 5 mV/Div
```

Figure 3.20. ***Circuit design for Example 3.9 implemented with a non-inverting single-ended amplifier.*** *When the input is not differential (it is single-ended), then noise in either side of the input (positive or negative terminal) is passed through to the output with amplification.*



```
        Time        Channel_A    Channel_B
T1      624.564 ms  529.759 mV   4.835 mV
T2      674.564 ms  -568.623 mV  -5.165 mV
T2-T1   50.000 ms   -1.098 V     -10.000 mV

Timebase             Channel A              Channel B
Scale: 50 ms/Div     Scale: 200 mV/Div      Scale: 5 mV/Div
```

Figure 3.21. ***Circuit design for Example 3.9 implemented with a balanced differential amplifier.*** *In a differential configuration noise on both inputs is subtracted and eliminated in the ideal situation.*

3.5.3 Instrumentation Amplifier

The instrumentation amplifier is a more complicated version of the differential amplifier that includes multiple Op Amps to improve performance. Like the basic differential amplifier previously discussed, the ability of this scheme to reject noise that is common between both inputs relies heavily on certain resistors being precisely equal (i.e. both resistors labeled R_4 must be precisely equal, both resistors labeled R_3 must be precisely equal, etc)). Any variance will cause gain imbalances and reduce the effectiveness of eliminating common noise. For this reason, and given modern manufacturing techniques, instrumentation amplifiers are typically manufactured as monolithic integrated circuits so that the resistance values can be laser-trimmed on the chip substrate during the manufacturing process. It should be noted that both inputs go through the non-inverting pin of the respective input buffer Op Amps, ensuring that the input source signals both see a high input resistance into the instrumentation amplifier. This is particularly useful for very small, weak signals like heart signals (ECG) or brain signals (EEG).

The most common instrumentation amplifier design has two additional Op Amps that serve as input buffers. The general three Op Amp instrumentation amplifier design is shown in Figure 3.22.

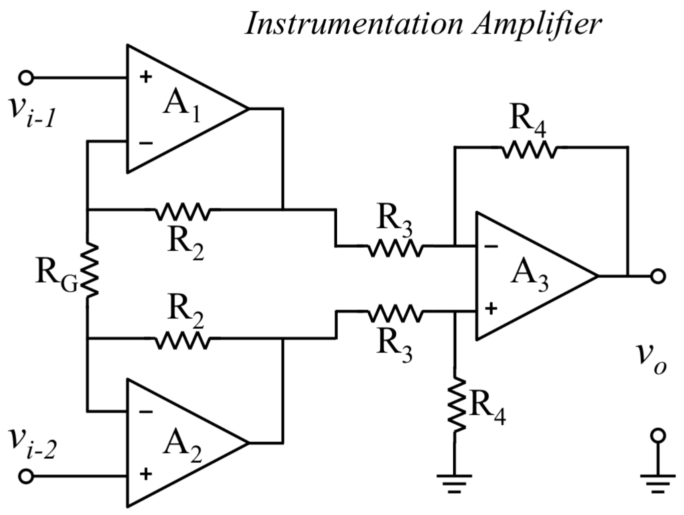


Figure 3.22. ***Three Op Amp Instrumentation amplifier configuration:*** *General configuration of an instrumentation amplifier circuit. R_G is the only external, user-selectable value, and the actual gain equation from a manufacturer is specified in the data sheet.*

For most instrumentation amplifiers only the resistor labeled R_G is supplied by the user. This resistor is common to both of the input buffer Op Amps (A1 and A2 in Figure 3.22), so it does not affect the common mode gain. Any inaccuracies between the actual and nominal value of this resistor are simply carried into the gain equation. The gain equation can be determined for instrumentation amplifiers using superposition analysis, but it is usually not necessary because the gain equation is given in the datasheet. An example of an instrumentation amplifier that is based off the three Op Amp design shown in Figure 3.22 is the INA217. The following is the datasheet for this IC: `http://www.ti.com/lit/ds/symlink/ina217.pdf`. The IN217 has a gain equation given by the following equation and, if no R_G resistor is included, the default gain is 1 V/V.

$$G = 1 + \frac{10\ k\Omega}{R_G}$$

Another common instrumentation amplifier contains only two Op Amps that serve as input buffers, but these two Op Amps also supply all the gain instead of relying on a third Op Amp to set the final gain like the design in Figure 3.22. The general two Op Amp instrumentation amplifier design is shown in Figure 3.23.

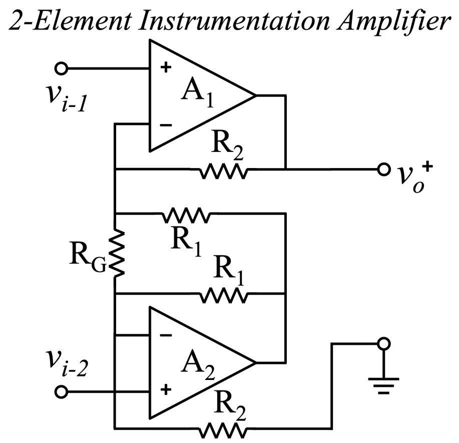


Figure 3.23. ***Two Op Amp Instrumentation amplifier configuration:*** *Alternate configuration for an instrumentation amplifier circuit using two op-amps. R_G is the only external, user-selectable value, and the actual gain equation from a manufacturer is specified in the data sheet.*

Some ICs have multiple instrumentation amplifiers in one package just like standard Op Amps (e.g. The LM324 quad Op Amp has four standard Op Amps in one 14-pin package). The INA2126 is an example of a dual instrumentation amplifier that uses the two Op Amp design shown in Figure 3.23 for each of the instrumentation Amplifiers on the IC. The INA126 is the single instrumentation amplifier version of the INA2126. The following is the datasheet for both the INA126 and INA2126: `http://www.ti.com/lit/ds/symlink/ina2126.pdf`. The INA126 and IN2126 both have a gain equation given by the following equation and if no R_G resistor is included the default gain is 5 V/V.

$$G = 5 + \frac{80\ k\Omega}{R_G}$$

All the ICs mentioned in this chapter have through-hole Dual Inline Packages (DIP) options so they can be easily integrated where the circuitry is implemented using a solderless prototype board (i.e. bread board) or soldered prototype board. Most instrumentation amplifiers are only available on surface mount IC packages so be sure to check that when specifying a chip for your application.

Example 3.10 *Use an IN2126 dual instrumentation amplifier to meet the following specifications:*

a) *Convert DC inputs of 3V and 2.23V to 10V*.

b) *Convert DC inputs of 2V and 1V to 5V*.

Solution

For part a) the required value of R_G can be calculated as follows:

$$G = 10/(3-2.23) = 13 = 5 + 80\text{ k}\Omega/R_G \Rightarrow \mathbf{R_G = 10\text{ k}\Omega}$$

(This is verified in Figure 3.24)

For part b) the required value of R_G can be calculated as follows:

$$G = 5/(2-1) = 5 = 5 + 80\text{ k}\Omega/R_G \Rightarrow \text{To get a gain of 5 V/V no } R_G \text{ is included}$$

(This is verified in Figure 3.24) **Note:** The IC selected in Multisim that is used in Figure 3.24 is the SN200501036DRFE4. The IN2126 is a replacement for this IC, as stated in the package information in Multisim.

From the beginning of this Chapter, let's look back at the 4-stage ECG signal conditioning block diagram that was shown in Figure 3.2 now that amplifiers have been covered. The first stage is usually implemented using an instrumentation amplifier, but the differential amplifier is also a possibility. The final gain stage can be implemented using a simple single-ended Op Amp configuration. The middle two stages of Figure 3.2 that

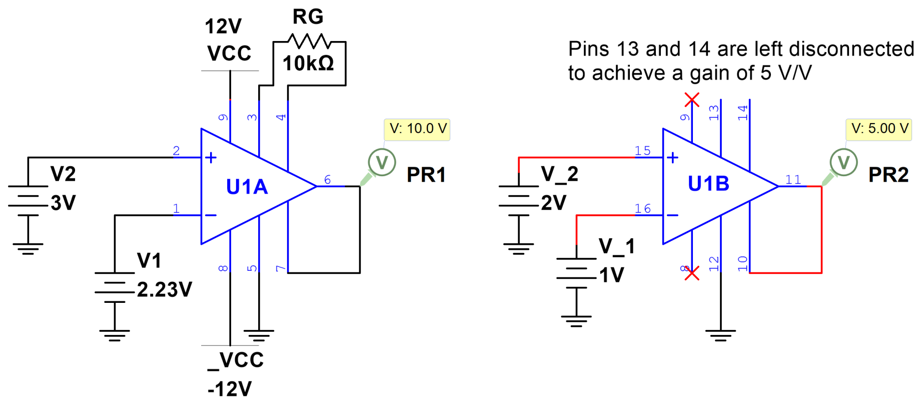


Figure 3.24. ***Implementation of an IN2126 Instrumentation amplifier for Example 3.10:*** *The instrumentation amplifier is designed to ensure the critical differential resistors are balanced. In this example, R_G is left out, so the base gain of 5 V/V is realized.*

involve filtering are discussed next.

3.5.4 Active Filter Circuits

Filter circuits are used to select, or block, certain frequency ranges within the signal. These circuits can also be thought of as *frequency selective* circuits. The reader may recall from the sample-rate discussion that any signal can be decomposed into a sum of sinusoids. The noise in an acquired signal is generally *not* wanted and can often be filtered out (though not always). Sometimes the noise is of a higher frequency than the signal-of-interest, and a filter that removes higher frequencies and keeps lower frequencies can be used to condition the incoming signal. Such a filter is called a *lowpass* filter. On the other hand, baseline wander and DC offsets can be removed with *highpass* filters (which *pass* higher frequencies and block DC and low frequencies). Before filter circuits can be understood, the reader should be familiar with the Laplace transform, as we need this mathematical tool to discuss the frequency-dependent gain equation. Briefly, the Laplace transform is given by:

$$X(s) = \int_{-\infty}^{+\infty} x(t)e^{-st}dt$$

where $s = \sigma + j\omega$. The real part of s, namely σ, determines the *region of convergence*, and when $\mathcal{Re}\{s\} < 0$, we can treat the Fourier transform as being the Laplace transform at $s = j\omega$. Thus, we can plot the frequency-dependent gain of a filter using the magnitude of the Fourier Transform, namely

$$|H(j\omega)| = |H(s)|_{s=j\omega}$$

It should also be recalled that $H(s)$ and $H(j\omega)$ are complex-valued functions so that the magnitude, $|H(j\omega)| = [H(j\omega) \cdot H(-j\omega)]^{1/2}$. The frequency gain response of a filter circuit is usually provided in the form of a graph with frequency (in Hz) on the abscissa (x-axis) and the magnitude (generally in dB) on the ordinate (y-axis).

3.5.4.1 Lowpass Filters

A lowpass filter allows *low* frequencies to pass through the system, while blocking *high* frequencies. What constitutes low and high is relative to the circuit and system application. Filter performance is often studied in the frequency domain where it is much more readily seen what frequencies are passed, and what frequencies are blocked. The frequency magnitude response plots the magnitude of each frequency based on the filter *transfer function*. The magnitude values are generally reported in units of dB, though sometimes absolute (V/V) units are used. The magnitude of the transfer function ($|H(j\omega)|$) is initially an absolute gain and if you need to convert it to dB Equation 3.9 is used: $20\log_{10}(|H(j\omega)|)$.

It is evident from Equation 3.9 that a gain of 1 V/V for a given frequency translates in decibels to 0 dB. An important value in the filter's frequency response is the *cutoff frequency*. This point is defined as the point where the attenuation of frequencies reaches 0.707 (or $1/\sqrt{2}$), which, in dB, is -3 dB. This so called "three dB point" is commonly referred to when discussing filters. The reader needs to realize that it is not the frequency at which the gain is equal to -3 dB, but rather the frequency at which it is 3 dB lower than the pass band gain. This is illustrated in Figure 3.25. The *roll off* region is the point where the passband begins to decrease in magnitude. How quickly the response rolls off is a function of the filter order.

A first-order filter is modeled mathematically with a first-order differential equation. When we look at the output-to-input ratio in the Laplace Domain, the *transfer function* can be represented as:

$$H(s) = \frac{k\omega_c}{s+\omega_c} = \frac{\frac{1}{R_1C}}{s+\frac{1}{R_2C}}$$

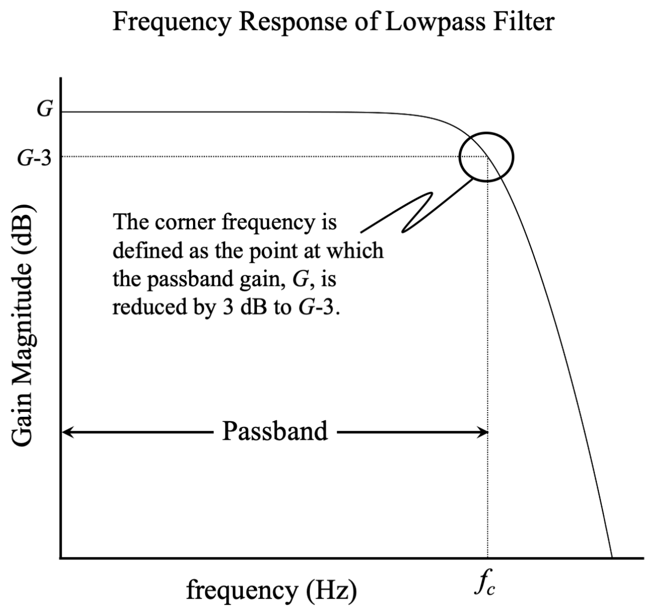


Figure 3.25. ***Frequency response of lowpass filter:*** *The lowpass filter allows frequencies lower than the cutoff frequency, f_c, to pass while frequencies above f_c are attenuated (effectively blocked).*

where k is the gain factor, and R_1, R_2, and C are shown in Figure 3.25. It is clear that when $R_1 = R_2$, the gain factor for frequencies below f_c is simply 1. Also, we can see from this expression that the corner frequency (in rad/s), ω_c, is determined by the combination of R_2 and C. Physically realizable active filters generally use lower orders, (second to fourth order) due to component count and complexity. It is noteworthy that digital filters can be easily realized for any practical order and are very powerful in the range of filter design and effect possible. This is why many modern systems are trending toward fairly simple signal conditioning filters in hardware, and post-acquisition digital filters that can provide more flexible and effective filtering (e.g. the LabVIEW Express Filtering VI shown in Figure 3.1).

It is worth noting that students should be very familiar, and comfortable, with negotiating the difference between frequency units in Hertz (Hz) and frequency units in radians per second (rad/s). All of the transfer function math learned in a Signals and Systems course is developed around the trigonometric functions that use rad/s. We often use the greek symbol ω to represent frequency in rad/s. However, industry standard is to plot the frequency response with the units of Hz for the frequency on the x-axis and units of dB for the magnitude of the gain on the y-axis. The Hertz is a *cycle per second*, and is more understandable in the physical world. But, because the math is oriented toward the use of rad/s, we will need to be able to move between these two units readily. The relationship is:

$$\omega = 2\pi f \tag{3.11}$$

This is important, because in the discussions below corner frequencies are generally designed to be normalized to 1 rad/s for didactic convenience. Most students who remember their coursework in Introductory Electronics recall taking a frequency specification in Hz, converting that to rad/s, designing the filter, and then plotting the filter response using Hz again.

The basic first-order active lowpass filter circuit is given by the schematic shown in Figure 3.26. This provides an *inverting* gain, lowpass filter. In other words, even the frequencies that are passed through the filter are inverted, so the output is a negative version of the passed frequencies. In general, the first-order filter response is fairly gradual between the passed frequencies and the blocked frequencies.

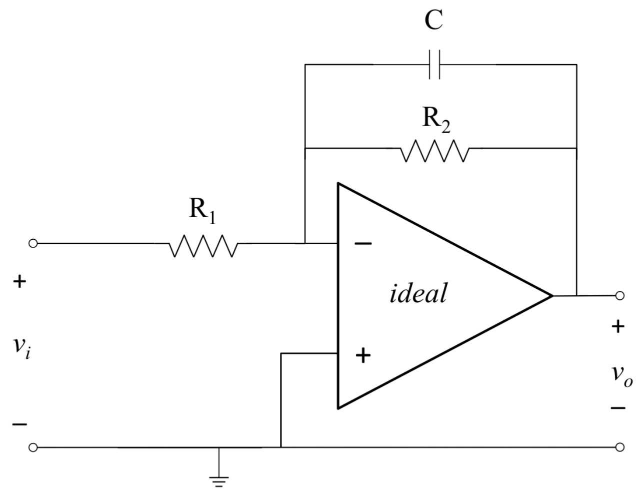


Figure 3.26. ***Basic*** 1^{st}***-order Active Lowpass Filter Configuration:*** *This first-order lowpass filter is the one generally introduced in beginning Electronics texts. First-order responses are fairly weak filters in terms of frequency selectivity, so they might be cascaded for better frequency response.*

A better, second-order, response can be achieved by cascading two identical first-order filters, though the overall cutoff frequency is altered by cascading. This is shown in 3.27, where it is seen that cascading identical first-order filters changes the cutoff frequency. Indeed, if one cascades n first-order filters with a cutoff frequency of 1 rad/s, using the configuration shown in Figure 3.26, the overall cutoff frequency changes by:

$$\omega_{c_n} = \sqrt{\sqrt[n]{2}-1} \tag{3.12}$$

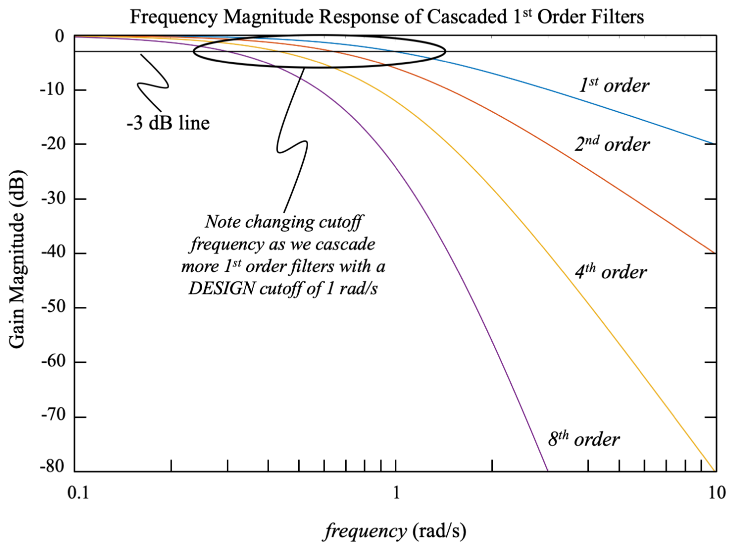


Figure 3.27. ***Cutoff Frequency Response of Cascaded Filters:*** *Taking a first-order lowpass filter with cutoff frequency (ω_c) of 1 rad/s, and cascading 2, 4, and 8 of them, changes the effective cutoff frequency, ω_{c_n}.*

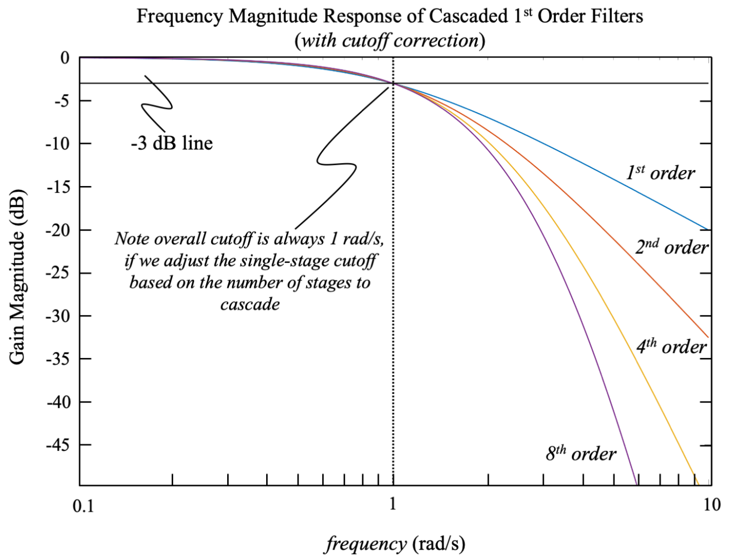


Figure 3.28. ***Cutoff Frequency Response of Adjusted Cascaded Filters:*** *Taking a first-order lowpass filter with adjusted cutoff frequency based on the number of cascaded sections ensures that the overall cascade system always provides the desired cutoff frequency (in this case, 1 rad/s). The single-stage, first-order cutoff frequencies are, respectively, 1, 1.5538, 2.2990, and 3.3240 rad/s.*

Equation 3.12 tells us that if we cascade n first-order lowpass filters, the effective cutoff frequency of the entire system will be given by ω_{c_n}. In order to get the desired cutoff frequency, we must adjust the single-stage cutoff frequency by $1/\omega_{c_n}$ For example, if we desire a cutoff frequency of 1 rad/s, but want to cascade eight first-order filters to realize an 8^{th}-order response, we see that cascading eight first-order filters gives $\omega_{c_8} = 0.300845$ rad/s, so we need to adjust the single-stage cutoff by the inverse, or $\omega_1 = 3.324$ rad/s. The results of adjusting the singe-stage cutoff frequency are shown in Figure 3.28. In each of the filter responses, the individual stage cutoff frequency was adjusted so that the overall cascaded system provides the same cutoff at 1 rad/s, but with varying steepness in the *roll-off* portion of the response.

In practice, a much better filter for signal fidelity over the passband is the Butterworth lowpass filter. The basic Butterworth filter is a second-order filter, and can be cascaded for higher-order response. The Butterworth filter maintains uniform gain over more of the passband, meaning the frequency response is maximally flat in the passband. The configuration of a unity-gain second-order Butterworth lowpass filter (using the Sallen-Key configuration) is shown in 3.29.

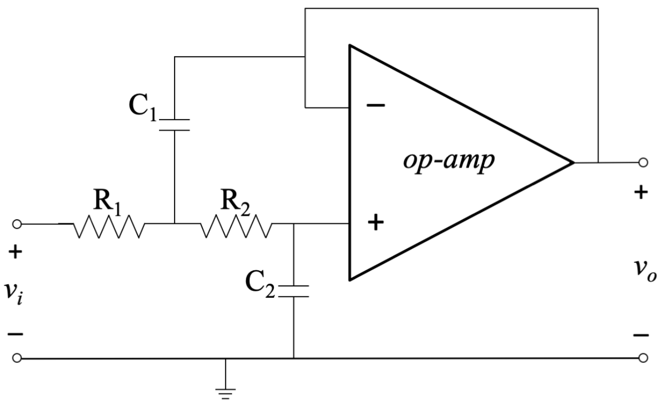


Figure 3.29. ***Second-order Butterworth Lowpass Filter:*** *The Butterworth filter guarantees a maximally flat frequency response in the passband. The configuration in this figure is based on the Sallen-Key realization of the Butterworth math model.*

In the lowpass configuration, many online calculators use a fixed resistance for both resistors and then the value of C_1 is twice C_2, namely $C_1 = 2C_2$. In practice, a fourth-order response using the Butterworth filter is achieved by cascading two second-order Butterworth filters. But, again, care must be taken to adjust the single-stage cutoff frequency so that the effective cutoff frequency of the entire system is the desired value. The unity-gain second-order transfer function (math model) of the Butterworth response is given by:

$$H(s) = \frac{1/(R_1R_2C_1C_2)}{s^2 + (1/R_2C_1 + 1/R_1C_1)s + 1/(R_1R_2C_1C_2)}$$

Given this transfer function it turns out that the corner frequency is given by:

$$\omega_c^2 = 1/R_1R_2C_1C_2$$
$$\Downarrow$$
$$\omega_c = \sqrt{1/R_1R_2C_1C_2}$$

The standard filter response can be compared with the Butterworth frequency magnitude response to demonstrate the more effective flat passband of the Butterworth filter. Figure 3.30 plots the second-order filter response created by cascading two standard first-order filters, versus a second order Butterworth filter. A fourth order Butterworth filter is also plotted to show that cascading two second order Butterworth filters improves the flatness of the passband even more. Each filter was adjusted so that the cutoff frequency is set to 1 rad/s, so they all pass through the -3 dB point together at a frequency of 1 rad/s.

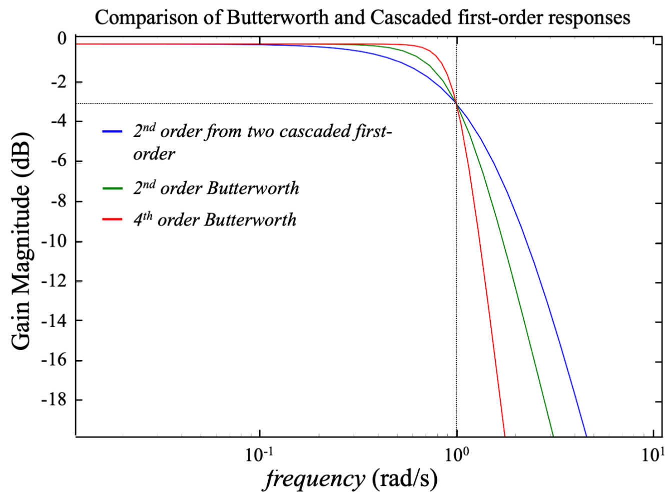


Figure 3.30. ***Comparison of standard and Butterworth Lowpass Filters:*** *The blue trace shows the second order response of two basic first order filters cascaded together (with adjusted ω_c so that the effective corner frequency is 1 rad/s). The green trace shows the frequency response of a second order Butterwroth lowpass filter with a cutoff frequency of 1 rad/s. It is noteworthy that the blue trace begins to roll off just beyond the 0.1 rad/s point, whereas the Butterworth response remains flat until about 0.4 rad/s. Finally, the fourth order Butterworth filter (in red) is shown for comparison. Note that it remains flat in the passband all the way to 0.7 rad/s.*

Lowpass filters are a very common type of filter for two main reasons:

- They are commonly used in digital data acquisition systems to prevent a phenomenon called *aliasing* (see Section 2.2.2.1). The lowpass filter ensures that no frequencies greater than the half-sample rate ($f_s/2$) can pass to the digitizer. This is often referred to as an *anti-aliasing* filter, but is simply a lowpass filter with a corner frequency (in Hertz) just below $f_s/2$.
- Additive noise is generally broad-spectrum, and much of the noise occurs above the frequencies of interest. Lowpass filters can reduce much of the additive noise before digitizing the signal.

In measurement and instrumentation the use of analog (physically realizable) filters is generally focused on signal conditioning of the incoming electrical signal with the goal that the acquired signal is expected to be as representative of the actual measurand as possible. Measuring the electrical signal from the heart, for example, can be corrupted by electrical noise from the skeletal muscles. But because electrical muscle noise (EMG) is generally of higher frequency than the ECG, a lowpass filter can be used to mitigate the interference of unavoidable muscle movement (even a patient lying very still must breathe!). The other basic filter is a highpass filter, which is discussed in the following section.

3.5.4.2 Highpass Filters

The other basic type of active filter is a highpass filter. The highpass filter seeks to pass higher frequencies and block lower frequencies. The *cutoff* frequency defines what is a high frequency, and what is a low frequency. Figure 3.31 shows the basic frequency response of the highpass filter. Frequencies less than f_c are attenuated to -3 dB or more, while frequencies greater than f_c are passed at the design gain. The fundamental first-order highpass active filter (inverting, as presented in many introductory electronics textbooks) is shown in 3.32.

In this case, the transfer function describing the frequency response is based on a first order equation with the following Laplace Domain gain expression (see Figure 3.32):

$$H(s) = \frac{ks}{s + \omega_c} \quad = \quad \frac{\frac{R_2}{R_1}s}{s + \frac{1}{R_1 C}}$$

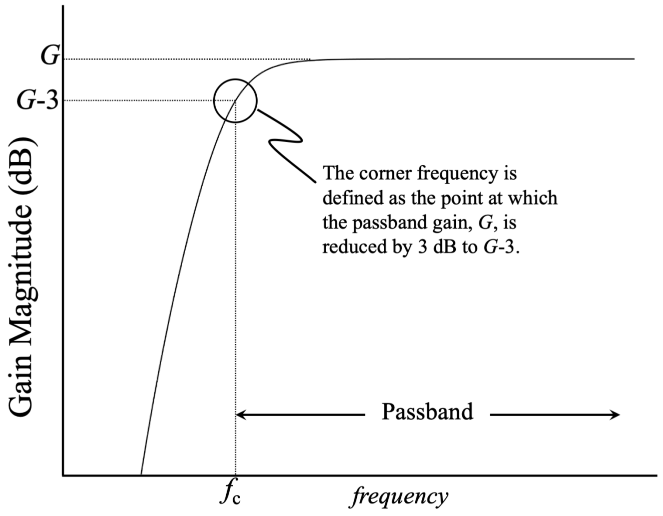


Figure 3.31. ***Frequency response of highpass filter:*** *The highpass filter allows frequencies higher than the cutoff frequency, f_c, to pass while frequencies below f_c are attenuated (effectively blocked).*

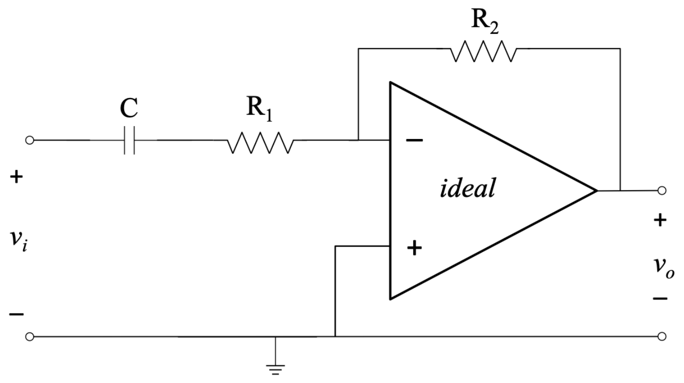


Figure 3.32. ***Basic*** *1st**-order Highpass Filter Configuration:*** *This first-order highpass filter is the one generally introduced in beginning Electronics texts. First-order responses are fairly weak filters in terms of frequency selectivity, so they might be cascaded for better frequency response.*

Like we saw in the lowpass filter, setting $R_1 = R_2$ provides a unity gain. The corner frequency, in rad/s, is controlled by the combination of R_1 and C.

As with the basic first-order lowpass filter, the configuration in Figure 3.32 is inverting. Furthermore, it has a weak roll-off as it attenuates so that the passband is not as flat as we might desire. A more effective version of the highpass filter that better preserves the incoming signal morphology is the second-order Butterworth highpass filter, realized with the Sallen-Key configuration shown in Figure 3.33.

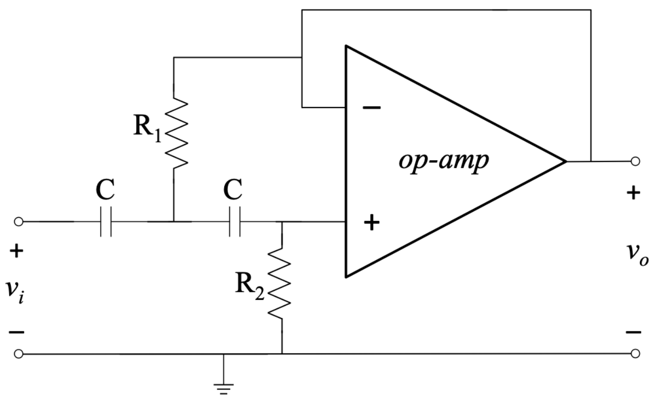


Figure 3.33. ***Second-order Butterworth Highpass Filter:*** *The Butterworth filter guarantees a maximally flat frequency response in the passband. The configuration in this figure is based on the Sallen-Key realization of the Butterworth math model.*

In this case, the transfer function is given by:

$$H(s) = \frac{s^2}{s^2 + (1/R_2C_1 + 1/R_2C_2)\,s + 1/R_1R_2C_1C_2}$$

Again, the corner frequency of this filter is given by the root of the last term in the denominator, namely $\omega_c = \sqrt{1/R_1R_2C_1C_2}$. Some popular online design tools often set $C_1 = C_2 = C$, and then solve for R_1 and R_2. In this case, $R_2 \approx 2R_1$, though not exactly. As with the lowpass version, two second-order Sallen-Key configurations can be cascaded to realize a fourth-order highpass Butterworth filter with a maximally flat passband.

3.5.4.3 Bandpass and Bandstop Filters

A third class of filters consists of bandpass and bandstop filters. These filters do exactly what their names imply. The bandpass filter passes a set of frequencies in some limited range (frequency band), from f_{low} to f_{high}. On the other hand, the bandstop filter blocks frequencies over some range. From a system perspective, we can model a bandpass filter as a lowpass filter cascaded with a highpass filter. Figure 3.34 shows a block diagram of a bandpass filter and the corresponding frequency response. The reader may observe that the order of operation is not critical. Thus, the input signal $x(t)$ could be filtered by the highpass filter first, followed by the lowpass filter, and the result would be the same.

However, we cannot cascade filters to produce a bandstop filter. Instead, the input signal, $x(t)$, is sent through a highpass and a lowpass filter separately and the outputs

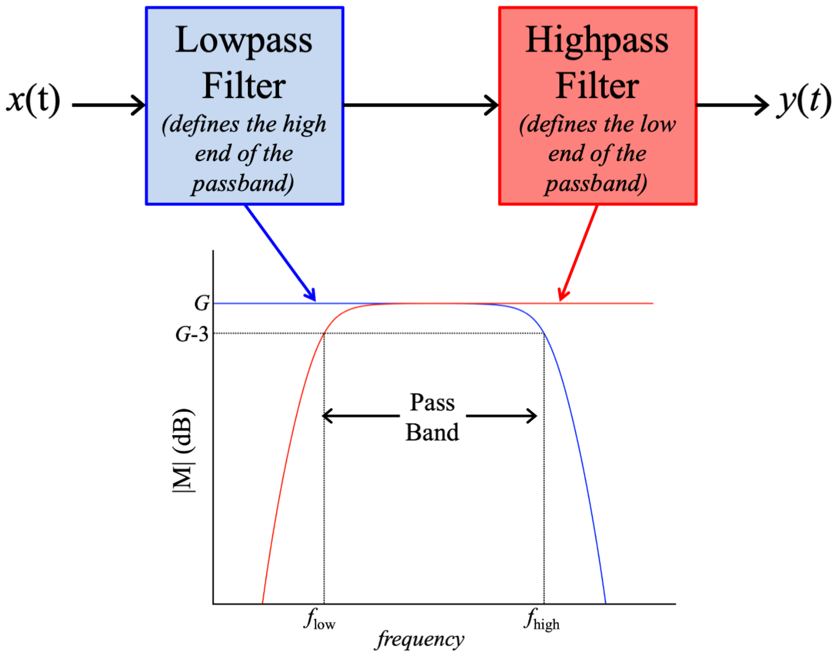


Figure 3.34. ***Bandpass Filter Block Diagram and Frequency Response:*** *The bandpass filter can be modeled as a cascade of a lowpass filter and a highpass filter. The lower cutoff frequency arises from the highpass filter specification, while the upper cutoff frequency arises from the lowpass filter specification, as seen by the two colored magnitude response lines.*

are combined. This is shown in block diagram form, along with a frequency magnitude response, in Figure 3.35. It should be readily obvious that if $x(t)$ is filtered by the lowpass filter, there are no frequencies greater than f_{low} remaining to be filtered by the highpass filter. Likewise, if $x(t)$ is passed through the highpass filter, there are no frequencies less than f_{high} remaining to be filtered. Thus, the two filters must be applied separately and the results combined to get the upper and lower passbands.

In neither case are these filters designed to create a very narrow band, whether a pass band or a stop band. More specialized filters are needed to create narrow (or *notch*) filters. The most frequent notch filter that one might encounter in data acquisition is the 60 Hz notch filter. Power line interference is a common problem in many instrumentation systems and is sometimes mitigated prior to digitization using an analog (i.e. realized with actual hardware components) 60 Hz notch filter. However, digital filters are very effective and can be made adaptive so that more effective notch filtering can often be accomplished after digitization. Many modern systems that have line-noise considerations (e.g. an ECG recording device) employ digital filters to mitigate 60 Hz interference. Of course, it is always best to properly ground and shield the data acquisition portion of an

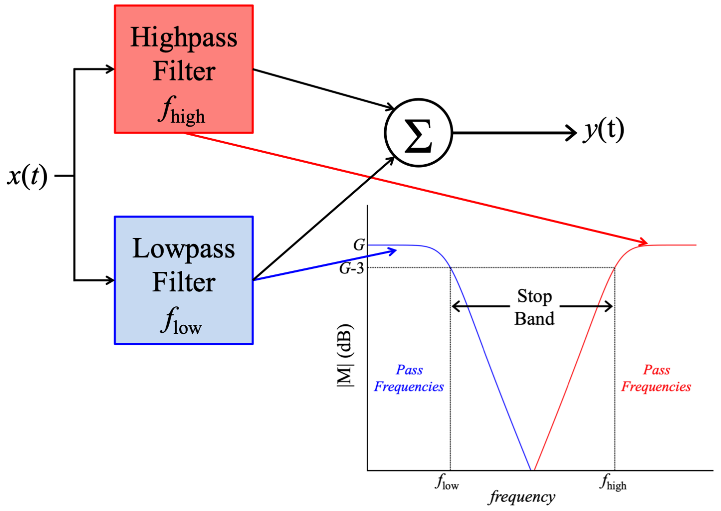


Figure 3.35. ***Bandstop Filter Block Diagram and Frequency Response:*** *The bandstop filter can be modeled as two separate filters in parallel, with a summer to combine the separate results. In this case, the lower frequency is defined by the lowpass filter, while the upper frequency is defined by the highpass filter.*

instrumentation system to minimize noise from the outset.

To better understand bandpass and bandstop filters, we observe the action of each filter represented in Figure 3.36. A base signal was created as the sum of a 10 Hz sine wave, a 100 Hz sine wave, and a 500 Hz sine wave. Subsequently this combined signal was filtered with a lowpass filter, a highpass filter, a bandpass filter, and a bandstop filter. The lowpass filter was set to a cutoff frequency of 30 Hz, so that only the 10Hz signal remained (though one can see filter transient behavior at the beginning and end). The highpass filter cutoff frequency was set to 300 Hz and the base signal was filtered. In this case, only the 500 Hz signal remained at the output of the filter. Similarly, the bandpass filter was set to include the frequency band from 30 Hz to 300 Hz. In this case, the only remaining component of the base signal was the 100 Hz sine wave. Finally, the bandstop filter was set with the same corner frequencies as the bandpass filter, but with the goal of eliminating the frequencies between 30 Hz and 300 Hz. The lower panel clearly shows the 500 Hz sine wave riding on the 10 Hz sine wave, but the 100 Hz sine wave has been filtered out. Of course, common signals have many more than just three frequencies, but this example shows how signals are affected by the various filters.

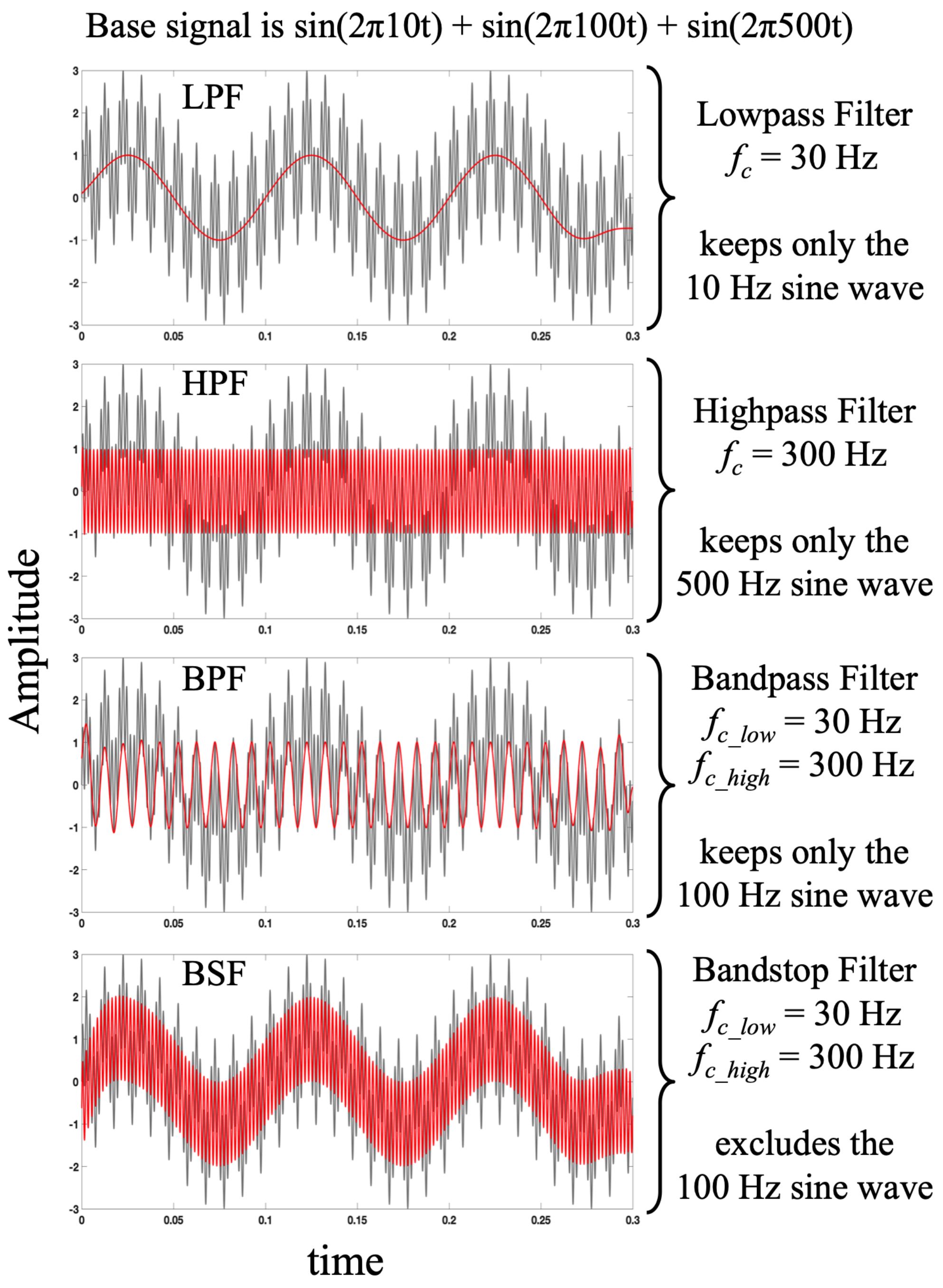


Figure 3.36. ***Comparison of Filter Effects for Various Filter Types:*** *A single base signal consisting of three separate frequencies was created (the gray signal in each panel) and filtered by lowpass, highpass, bandpass, and bandstop filters. In each case, the designed filter selected some frequencies to eliminate and one or more frequencies to pass through, based on the designed corner frequencies. The output of each filter is plotted in red.*

3.6 Conclusion

The two most needed types of signal conditioning are amplification and filtering. Highpass filters are often used to block any DC offsets that might appear in a signal. This is not uncommon in biomedical signal acquisition, for example, because each electrode placed on the skin has its own impedance that is a result of the skill of the technician applying the electrodes. When slight impedance mismatches occur, this can create a DC offset in the resulting signal. While we do not consider a constant DC voltage to have a *frequency*, the Fourier transform shows us that the DC element of a signal can be thought of as 0 Hz, so a highpass filter of 0.5 Hz would be more than adequate in blocking the DC offset. On the other hand, the lowpass filter is usually implemented to reduce white noise interference. These noise sources can be environmental (such as the electromyographic activity in an ECG), or intrinsic to the system (such as thermal noise in the electronics). These noises generally have a fairly wide frequency band, but often toward the higher frequency end of the acquisition system. Indeed, the fact that the noise may exceed the frequency bandwidth of the data acquisition system is a fundamental reason to employ lowpass filters. A properly designed lowpass filter can guarantee that we avoid aliasing by limiting the incoming frequency content to be less than one half of the sampling rate (in order to meet the Nyquist sampling criterion). In this case, the lowpass filter is often referred to as an *anti-aliasing filter*.

Digital signal processing (DSP) is a well-developed field that has produced many techniques for modifying discrete signal content. Digital filters have significant flexibility in that their frequency response can be easily controlled and they are generally preferable for intense filtering and signal analysis needs. Analog filter and amplification circuits are mainly used to condition the signal in such a way that the signal content can best be digitized without losing signal fidelity with respect to the originating physical phenomenon being represented by the signal. Analog signal conditioning is a fundamental step in accurately measuring real-world phenomena with a computer-based data acquisition system, and a reasonable understanding of filters–and an ability to implement those filters–is critical to good measurement and instrumentation practices.

Chapter 4

4.1 Sensors

Computer-based measurement systems typically begin wtih the conversion of the physical phenomenon being measured into an electrical signal. The process of converting one physical phenomenon to another physical phenomenon is called *transduction*. A device that can perform this task is called a *transducer*. However, a transducer is a general type of device that simply converts one physical phenomenon to another. This does not necessarily imply conversion to an electrical form. For example the Atmos clock, that was invented by Jaeger-LeCoultre, uses a bellows that is filled with a gas that is sensitive to room temperatures. As the ambient temperature increases, the gas bellows expands, and as the temperature decreases, the sealed gas bellows contracts. One side of the bellows is fixed to the frame of the clock, allowing the other side to move back and forth. A small spring is attached to the moving side and connected to a very low-resistance ratcheting mechanism that winds the spring. In this case, the ideal gas law (PV = nRT) is invoked to translate temperature changes into mechanical movement (see the Atmos clock in Figure 4.1).

The point of the Atmos clock example is that not all transducers convert to an electrical form. On the other hand, when the term sensor is used it is referring to a device that can convert one physical form specifically into an electrical signal. From this perspective, sensors are a subset of transducers. However, in practice the terms *sensor* and *transducer* are often used interchangeably. There are sensors for many different physical phenomena. Some of the common ones that will be discussed in Section 4.2 are:

- Strain Gauge (4.2.1)
- Linear Variable Differential Transformer (4.2.2)
- Potentiometric Sensing (4.2.3)
- Ultrasound Sensor (4.2.4)
- Accelerometer (4.2.5)
- Pressure sensor (4.2.6)

- Encoders (4.2.7)
- Temperature Sensor (4.2.8-4.2.9)

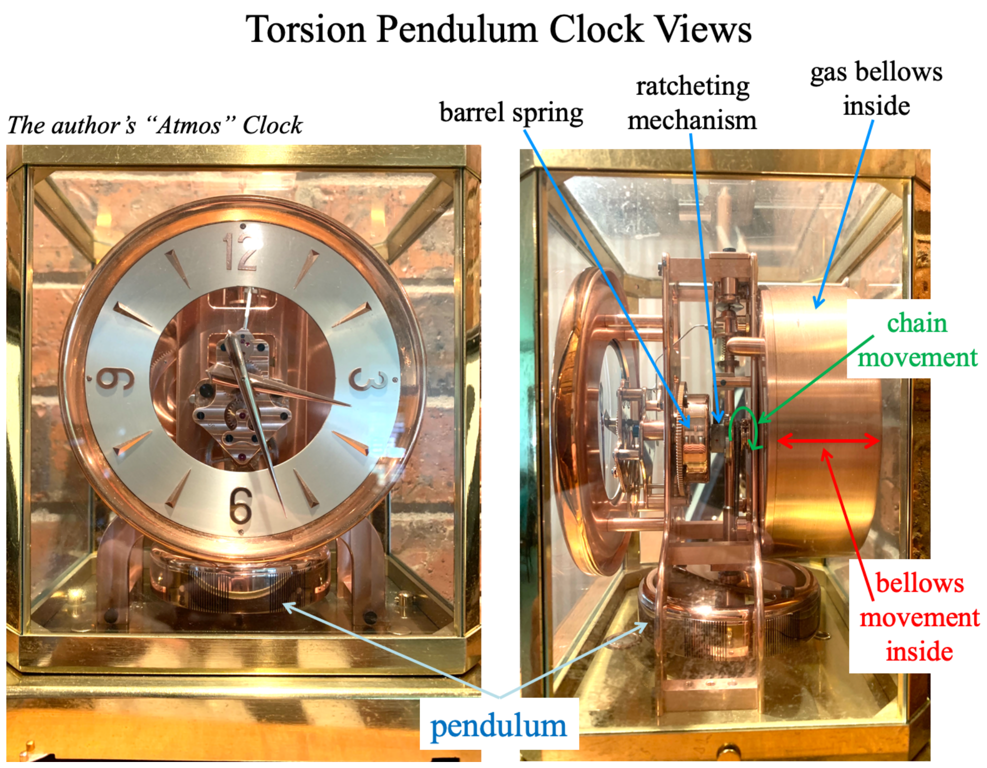


Figure 4.1. ***Atmos Clock as a Transducer:*** *The gas bellows inside the brass case at the back of the clock expands and contracts with temperature changes, which pulls and releases the small chain attached to the ratcheting mechanism. This winds the spring one way, and the ratchet ensures that it does not loosen when the chain direction reverses. The pendulum regulates the clock. Atmos Clock photo by John Dyer*

There are sensors that are capable of acquiring an electrical representation of nearly any phenomenon one can imagine due to the ubiquitous nature of computer-based data acquisition and measurement systems. This book will focus on some of the more widely used sensors and discuss the basic features that apply to all sensors and have a direct effect on one's ability to accurately measure something.

4.1.1 Sensitivity

Sensors have a very important parameter that is associated with them called *sensitivity*. The sensitivity of a sensor is the *ratio of the electrical output to the physical quantity input*. Sensitivity is often referred to as the scale factor or the conversion factor because it is the factor that can be used to scale or convert the measured output to the input. If there is only a small change in electrical quantity due to a large change in the physical quantity the sensor would be said to have a poor sensitivity. For example, a J-Type thermocouple has a published sensitivity of only 50 μV/°C. This means if you want to distinguish a temperature change of 0.1 °C you would only have a corresponding voltage change of 5 μV, which would likely be a smaller amplitude than noise in the environment. For sensors like thermocouples with poor sensitivity, signal conditioning becomes vitally important in order to accurately make a measurement. The sensitivity of a sensor, in essence, defines a linear relationship that can be established to programmatically convert the electrical reading into a value in the desired units of the physical measurand. Thermocouples are a good example of a sensor where a sensitivity value is used to give a close approximation, but more accuracy can be obtained using published thermocouple charts. Figure 4.2 shows a K-Type Thermocouple plot that has a published sensitivity of 41 μV/°C.

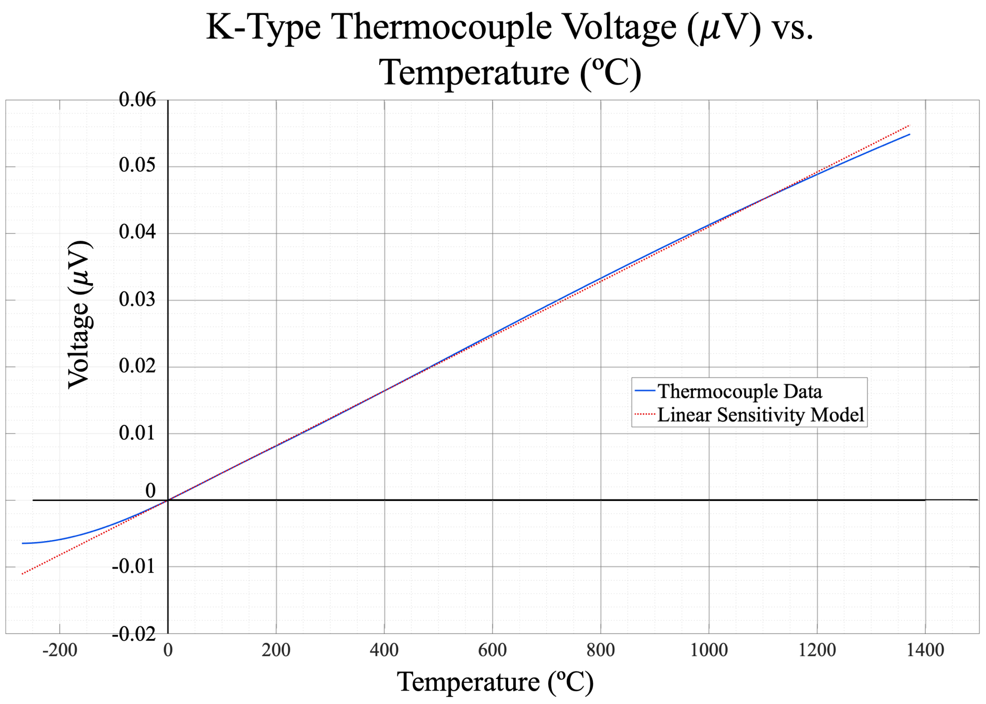


Figure 4.2. ***K-Type Thermocouple Voltage vs Temperature plot:*** *The data come from the following link:* `https://www.thermometricscorp.com/PDFs/Thermocouple-Charts/Type-K-Thermocouple-Chart-C.pdf`.

The red dashed line comes from the linear sensitivity model (Voltage = 41 μV/°C · Temperature) and the blue line shows the data from the K-Type Thermocouple chart. The K-Type Thermocouple chart ranges from -270 °C to 1372 °C. The linear sensitivity model appears to be accurate at low positive temperatures on this curve, but nonlinearities exist at the extremes which makes the Thermocouple chart much more accurate than the linear model. Many sensor types have both a linear sensitivity model and a more complicated model that tries to model nonlinearities more accurately.

While the K-Type Thermocouple doesn't include on offset (i.e. $y = 0 \ @ \ x = 0$) many sensors have an offset so the linear equation $y = mx + b$ must be used to create the linear sensitivity model. Figure 1.5 shows an example of a speed sensor that exhibits non-linearity, but does NOT include an offset of the y-intercept (i.e. $y = 0 \ @ \ x = 0$). Figure 4.3 shows an example of a sensor that has a linear sensitivity, but has a DC offset.

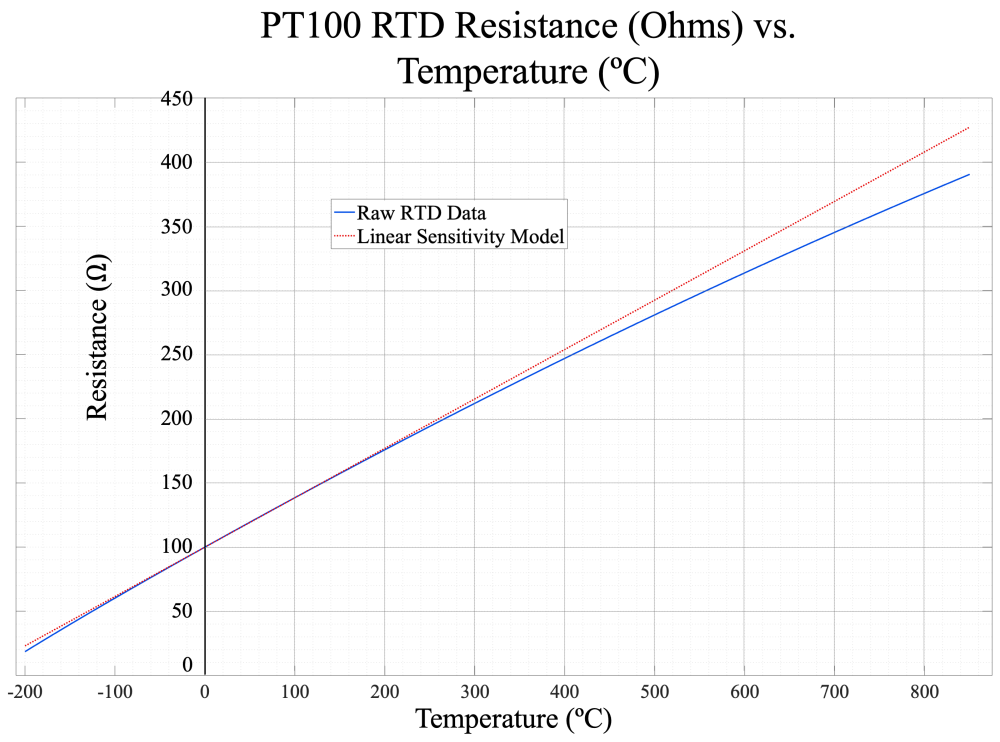


Figure 4.3. ***Resistance versus Temperature plot of a PT 100 Resistance Temperature Detector:*** *The data come from the following link:* `https://www.pyromation.com/Downloads/Data/385_c.pdf`*.*

This example is from a Resistance Temperature Detector (RTD) that was made according to the PT100 standard, which means it is a platinum RTD and has a resistance of 100 Ohms at 0 °C and has a published sensitivity of 0.385 Ω/°C. The linear sensitivity model matches the RTD chart model closely until the temperature gets above around 300 °C. When comparing the PT100 RTD to the K-Type thermocouple it can be seen that the RTD is much more linear at negative temperatures, but it becomes much more nonlinear than the thermocouple at high temperatures. The RTD also has a much smaller temperature range.

The red dashed lines in Figures 4.2 and 4.3 can be described by an equation of the form $y = mx + b$. In this case, the sensitivity of the sensor is described by the slope term, m. Figure 4.3 shows an example where the sensor has a non-zero resistance in the linear sensitivity model when the physical quantity input (temperature) is zero. Some sensors have a non-zero current and/or voltage when the physical quantity input is zero. Thus, for many sensors there is a y-axis offset that needs to be accounted for to accurately represent the physical quantity. An example of this can be seen in the 4 to 20 mA Omega EWS-RH sensor (see user manual at `https://www.omega.com/Manuals/manualpdf/M3501.pdf`), which measures relative humidity and temperature. This device uses yet another type of temperature sensor called a thermistor. The thermistor has an electrical output of resistance, but signal conditioning circuitry is included in the Omega device to convert the resistance to current and voltage. The 4 to 20 mA current loop standard has been used in measurement systems for a long time. This protocol, that has a span of 4 mA to 20 mA as the input changes from 0 to 100% of its range, is popular for many reasons. One benefit is that a current near zero would indicate a fault condition instead of the low range of the physical quantity being measured. Another benefit is that noise is less of an issue because the current is high enough that the noise floor is usually avoided. The current output can be easily converted to a voltage so that it is compatible with most DAQ devices by simply using a resistor. For the Omega EWS-RH sensor a 250 Ω resistor is used to convert the 4 to 20 mA current to 1 to 5 Volts. Regardless of whether current or voltage are used as the output, there is a non-zero y-intercept in the linear equation. Figure 4.4 shows the current in mA versus the temperature in degrees Celsius published in the Omega EWS-RH datasheet. This device also includes a humidity sensor that has 0 to 100% relative humidity that is scaled to 4 to 20 mA. This Omega thermistor-based temperature sensor behaves in somewhat linear fashion, but only for a much smaller temperature range than the thermocouple and RTD. Using the data from the Omega manual for this device, the average slope is equal to: (20 mA- 4 mA)/(60 °C - -15 °C) = 0.21333 mA/°C. The y-intercept is the current value (7.2 mA) when the temperature is 0 °C. Given this, the linear sensitivity model is $y = 0.21333x + 7.2$.

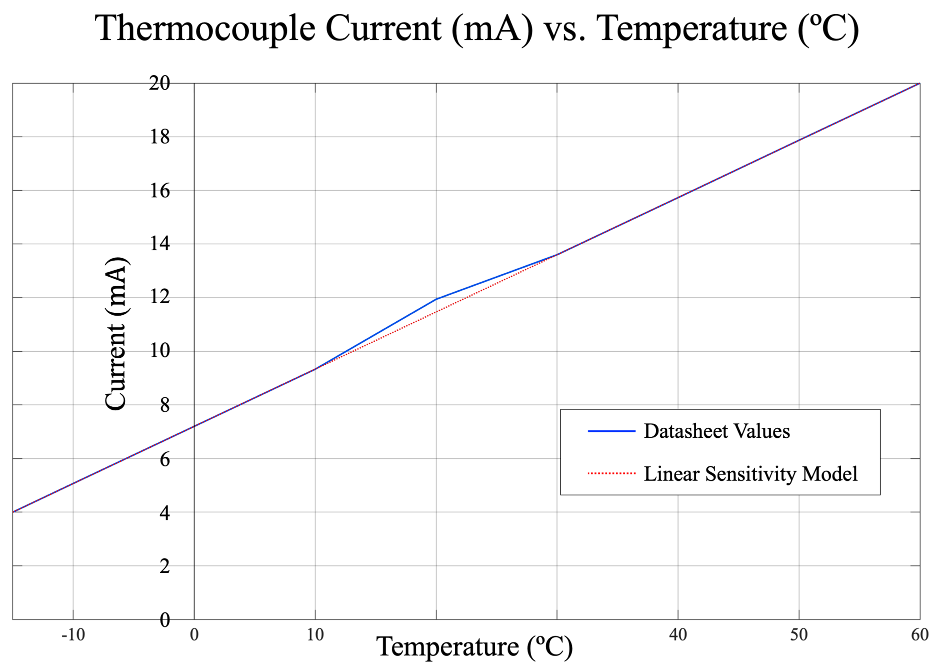


Figure 4.4. ***Current versus Temperature plot of an Omega Thermistor Probe:*** *The data come from* `https://www.omega.com/Manuals/manualpdf/M3501.pdf`.

All sensors have some sort of sensitivity and offset that must be accounted for in order to relate the electrical output to the physical measurand. When the 4 to 20 mA current loop protocol isn't used, the offset is usually near zero. But, it is not uncommon in sensors powered from a positive voltage supply to have some offset. Thus, during acquisition each incoming digitized value must be converted based on the specified sensitivity (i.e. slope) and offset. This is generally a programmatic conversion within the acquisition process but must be defined by the user based on calibration and manufacturer information. A robust measurement system would allow the user to adjust the parameters of the sensitivity conversion based on calibration testing against a standard. In the case of the Omega EWS-RH device, pages 6 and 7 of the manual (https://www.omega.com/Manuals/manualpdf/M3501.pdf) define the calibration procedure for the humidity and temperature sensors respectively. While these three temperature sensors served as good examples for the topic of sensitivity, more information will be given in the temperature sensor section.

4.1.2 Bandwidth

A second feature for which the user must account when choosing a sensor is the *bandwidth*. Bandwidth is directly related to frequency, which in turn relates to how quickly the physical phenomenon is changing. Usually the term bandwidth, in the context of sensors, refers to the highest frequency that is contained in the signal coming from the sensor. In Chapter 2 we learned that the Nyquist sampling criterion requires that the sampling frequency must be at least twice the highest frequency contained in the signal that is being measured. However, if high frequency noise is unexpectantly present in the sensor signal that causes the signal to violate the Nyquist sampling criterion, then aliasing will occur. In order to reduce the likelihood of aliasing, an anti-aliasing filter is often added in the measurement system. It can either be designed into the sensor or added externally. As discussed in Chapter 3, an anti-aliasing filter is simply a low pass filter that removes (or at least greatly attenuates) signals that are beyond the expected maximum frequency of the sensor. By removing these unwanted high frequency components prior to sampling, it greatly reduces the likelihood of aliasing affecting the measurement.

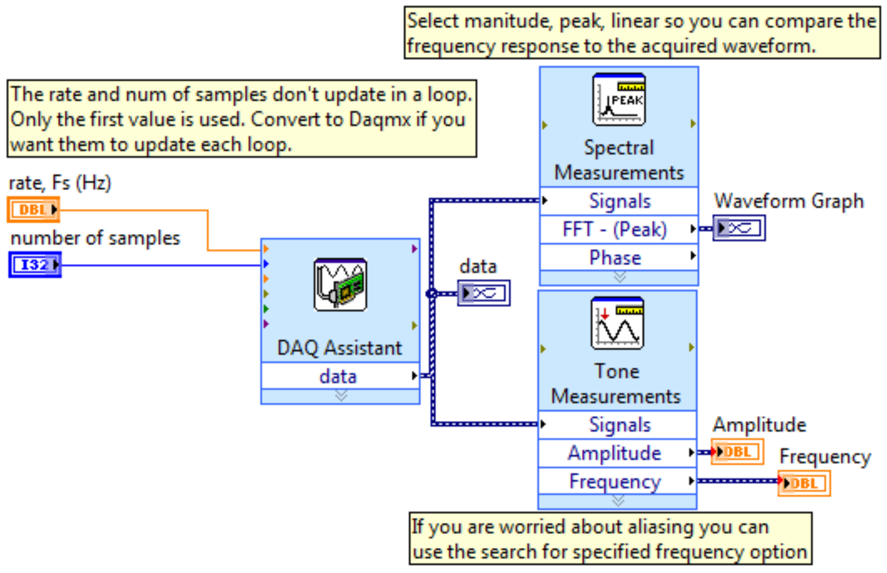
Bandwidth can be described by looking at the example of vibration, which is measured using a sensor called an accelerometer. This type of device will be covered in detail in a later section, but one may note that, when a structure is vibrating, it is undergoing some amount of displacement over time (i.e. velocity). The velocity also changes over time, producing acceleration. The vibration of a guitar string is typically in the hundreds of Hertz (cycles per second), while a bridge truss may only vibrate at a few Hertz. The expected frequency of vibration defines the bandwidth of the sensor that is needed when measuring the vibration. Clearly, an accelerometer that has a bandwidth of 10 Hz would be acceptable for the low-frequency vibration of the bridge truss, but would be utterly inadequate for measuring the vibration of a guitar string. The key facet of bandwidth is to ensure that the sensor bandwidth is capable of responding to the *maximum* frequency expected in the physical system.

In summary, once the proper sensor is selected for a task, one must ensure two things: 1) the sampling frequency of the data acquisition system is chosen to meet the Nyquist sampling criterion ($F_s > 2 \cdot F_{max}$); and 2) the device or user-supplied signal-conditioning circuitry prevents any signal corruption above the expected maximum frequency point. The first item can be addressed by selecting a DAQ unit that has a high enough sampling rate to handle the full bandwidth of the sensor. The second item can be addressed by adding an anti-aliasing filter, as previously discussed.

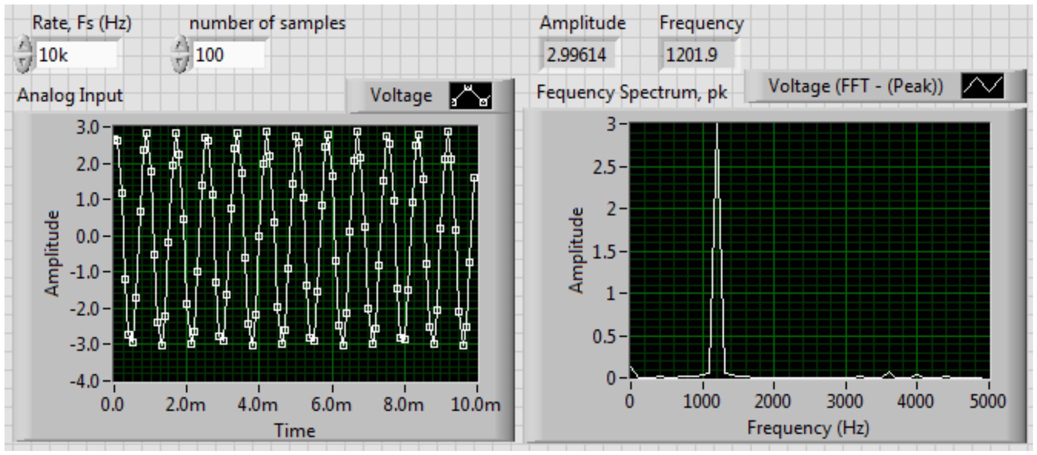
Example 4.1 *Design a measurement system using NI Hardware and LabVIEW software to measure a 1202 Hz sinusoidal signal at the following sampling frequencies: 10 kHz, 3 kHz, 2 kHz, 1.4 kHz, 1 kHz. Which of these will result in the correct signal being measured and which will result in aliases? For each of the cases that result in aliasing, calculate the frequency of the aliases using the equations from Example 2.7. If the sensor that was being used had a bandwidth of 300 Hz and the 1202 Hz sinusoidal signal was a result of noise what could be done to remove it? If a sampling frequency was set at 1000 Hz so that it is safely above twice the maximum frequency of the sensor* (2·300)*, what would happen if nothing was done to address the noise signal?*

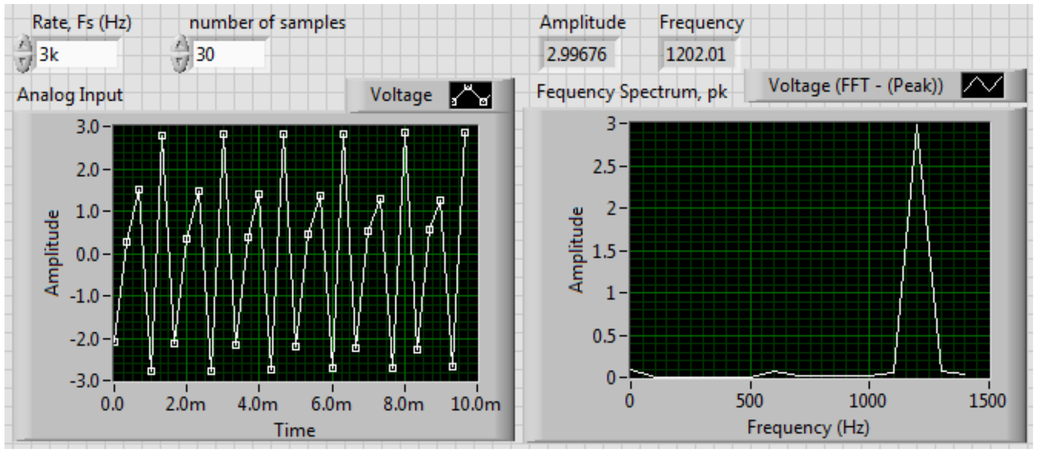
Solution:

The following LabVIEW code can be used for this measurement system:


Select manitude, peak, linear so you can compare the frequency response to the acquired waveform.
The rate and num of samples don't update in a loop. Only the first value is used. Convert to Daqmx if you want them to update each loop.
rate, Fs (Hz)
DBL
number of samples
I32
DAQ Assistant
data
data
PEAK
Spectral Measurements
Signals
FFT - (Peak)
Phase
Waveform Graph
Tone Measurements
Signals
Amplitude
Frequency
Amplitude
DBL
Frequency
DBL
If you are worried about aliasing you can use the search for specified frequency option

The sampling frequency must be at least twice the maximum frequency $(2 \cdot 1202)$ to avoid aliasing so no aliasing will occur for the F_s = 10 kHz and 3 kHz cases. The following screenshots show an actual signal of approximately 1202 Hz being measured using a DAQ with the sampling frequency set at 10 kHz and 3 kHz. Notice how the 10 kHz sampled signal looks much more like a sinusoid than the 3 kHz sampled signal, but both produce a nearly identical frequency response plot. This demonstrates that as long as the Nyquist sampling criterion is maintained, and the signal is sampled at a sampling frequency of no lower than 2404 Hz the frequency of the signal will be able to be detected. However, if you want the signal to appear more like a true sinusoid when you connect the samples with straight lines (as shown in the left plots below) then it is best to sample moderately higher than the Nyquist rate.





Example 2.7 shows equations to calculate the alias frequency and those equations could be used to determine the alias frequencies for the three alias cases. However, those equations are rather complicated so a quick method will be used as an alternative in this example. The first thing that is needed is to compute the normalized radian frequency, ω_{norm}. If ω_{norm} is greater than π then aliasing occurs. Frequently, the ω_{norm} will be greater than π, but less than 3π. If this is the case, then the alias frequency can be calculated simply as $|F_s - F_{actual}|$. If ω_{norm} is greater than 3π this method doesn't work, and it would be best to use the equations shown in Example 2.7 that work for all possible alias conditions.

$\omega_{norm} = 2\pi F_{actual}/F_s$

- **Case 1**:

$$\omega_{norm} = 2\pi 1200/2000 = 1.2\pi$$

$$\text{since } \pi \leq \omega_{norm} \leq 3\pi, \text{ then } F_{alias} = |F_s - F_{actual}| = |2000 - 1202| = \mathbf{798\ Hz}$$

- **Case 2**:

$$\omega_{norm} = 2\pi 1200/1400 = 1.714\pi$$

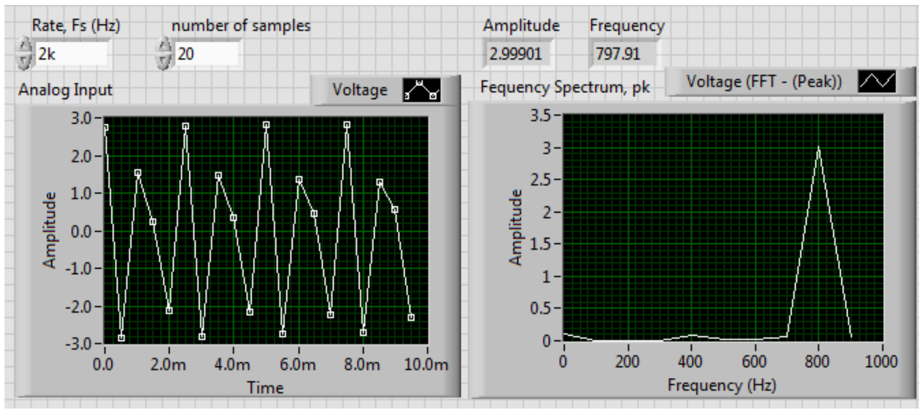
$$\text{since } \pi \leq \omega_{norm} \leq 3\pi, \text{ then } F_{alias} = |F_s - F_{actual}| = |1400 - 1202| = \mathbf{198\ Hz}$$

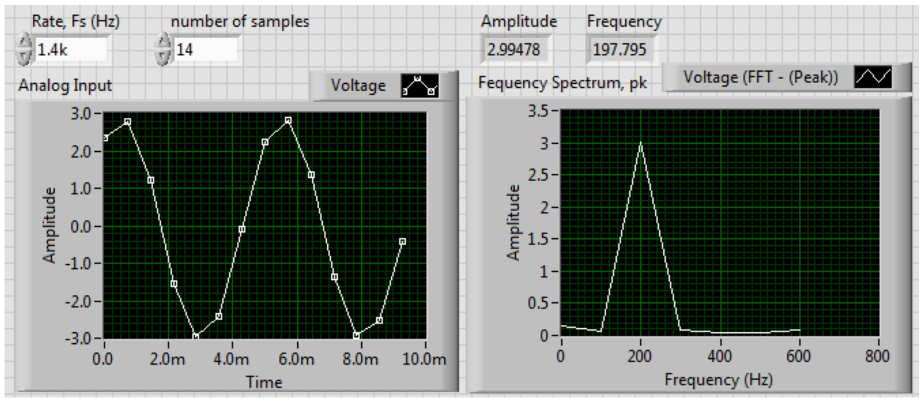
- **Case 3**:

$$\omega_{norm} = 2\pi 1000/2000 = 2.4\pi$$

$$\text{since } \pi \leq \omega_{norm} \leq 3\pi, \text{ then } F_{alias} = |F_s - F_{actual}| = |1000 - 1202| = \mathbf{202\ Hz}$$

These results are verified below:

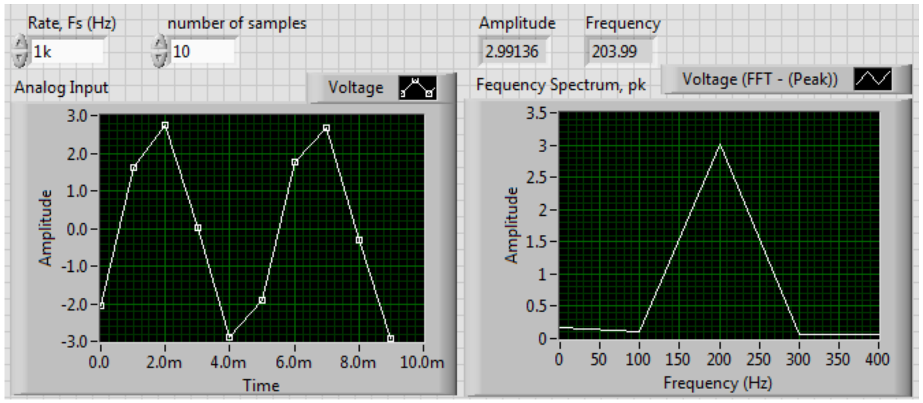

Rate, Fs (Hz): 2k
number of samples: 20
Amplitude: 2.99901
Frequency: 797.91


Rate, Fs (Hz) 1.4k

number of samples 14

Amplitude 2.99478

Frequency 197.795


Rate, Fs (Hz) 1k

number of samples 10

Amplitude 2.99136

Frequency 203.99

If the 1202 Hz was a noise signal and the sensor signal had a bandwidth of 300 Hz then an anti-aliasing filter could be used to eliminate the 1202 Hz noise signal. Since the sampling frequency was selected to be 1000 Hz in this example the anti-aliasing filter could be designed as a Low Pass Filter with a corner frequency of approximately 500 Hz.

If an anti-aliasing filter wasn't used then an alias of the 1202 Hz signal would appear in the bandwidth of the sensor as a 202 Hz frequency component. This additional frequency component would likely corrupt the sensor measurement and make it unusable.

4.1.3 Signal Conditioning with the Wheatstone Bridge

Some sensor types, such as the RTD in Figure 4.3, use small resistive changes as the method of transducing the physical phenomenon into an electrical signal. A strain gauge is another sensor that works in this manner. The user knows that Ohm's Law relates voltage, resistance, and current, but the changes in resistance in a strain gauge can be very small so that the commensurate linear change in voltage or current is also small. However, the bridge arrangement can be used to better assess the resistive change in a strain gauge. This is shown in Figure 4.5. In this figure, the resistor shown as R_x represents the strain gauge. It should be recognized from basic circuit analysis that the current supplied by the voltage source, V_{BAT}, is divided between two branches. The first branch consists of R_1 and R_2 in series, so that the these two resistors form a voltage divider. The other current path is through the two resistors R_3 and R_x. These two resistors also form a voltage divider. The midpoint of each resistor pair has some voltage associated with it, and the difference between these two voltage points is representative of the state of the bridge. When the bridge is *balanced*, the voltage at v_a is the same as the voltage at v_b, so that the difference is $v_a - v_b = 0$. Obviously, in this condition, we expect $R_1/R_2 = R_3/R_x$.

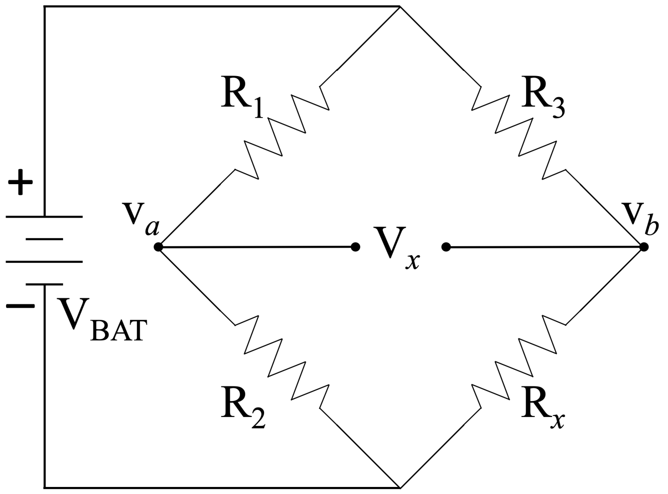


Figure 4.5. ***Classic Resistive Bridge Circuit:*** *The four-resistor bridge can be balanced by ensuring that the ratio R_1/R_2 is the same as the ratio of R_3/R_x. When the strain gauge, represented by R_x is placed in compression or tension, the ratio of R_3/R_x changes so that $v_a - v_b \neq 0$.*

However, when the strain gauge is placed in compression or tension, its internal resistance changes, so that the ratios are no longer equal, (i.e. $R_1/R_2 \neq R_3/R_x$). At the same time, this implies that $v_a \neq v_b$, so that in the imbalanced configuration $v_a - v_b \neq 0$. Mathematically, we can write

$$
\begin{aligned}
v_a &= V_{BAT}\frac{R_2}{R_1+R_2} \\
v_b &= V_{BAT}\frac{R_x}{R_3+R_x} \\
&\Downarrow \\
v_a - v_b &= V_{BAT}\frac{R_2}{R_1+R_2} - V_{BAT}\frac{R_x}{R_3+R_x} \\
&= V_{BAT}\left[\frac{R_3}{R_1+R_2} - \frac{R_x}{R_3+R_x}\right] \\
v_a - v_b &= V_{BAT}\left[\frac{R_3R_2 - R_1R_x}{(R_1+R_2)(R_3+R_x)}\right] \qquad (4.1)
\end{aligned}
$$

Because the balanced condition requires the two ratios to be equal, we have

$$
\begin{gathered}
\frac{R_1}{R_2} = \frac{R_3}{R_x} \\
\Downarrow \\
R_3R_2 = R_1R_x
\end{gathered}
$$

Clearly, from Equation 4.1, when $R_3R_2 = R_1R_x$ then $v_a - v_b = 0$. However, when the R_x value changes, then there is a non-zero difference between v_a and v_b, which we we call V_{diff}. Then substituting V_{diff} into the left hand side of Equation 4.1, and solving for R_x, we get

$$V_{diff} = V_{BAT}\left[\frac{R_2R_3 - R_1R_x}{(R_1+R_2)(R_3+R_x)}\right]$$

$$\frac{V_{diff}}{V_{BAT}} = \frac{R_2R_3 - R_1R_x}{(R_1+R_2)(R_3+R_x)}$$

$$\frac{V_{diff}\left[(R_1+R_2)(R_3+R_x)\right]}{V_{BAT}} = R_2R_3 - R_1R_x$$

$$\frac{V_{diff}(R_1R_3+R_2R_3+R_1R_x+R_2R_x)}{V_{BAT}} = R_2R_3 - R_1R_x$$

$$\frac{V_{diff}}{V_{BAT}}(R_1R_3+R_2R_3) + \frac{V_{diff}}{V_{BAT}}(R_1+R_2)R_x - R_2R_3 = -R_1R_x$$

$$\frac{V_{diff}}{V_{BAT}}(R_1R_3+R_2R_3) - R_2R_3 = -R_1R_x - \frac{V_{diff}}{V_{BAT}}(R_1+R_2)R_x$$

$$\frac{V_{diff}}{V_{BAT}}(R_1R_3+R_2R_3) - R_2R_3 = \left[-R_1 - \frac{V_{diff}}{V_{BAT}}(R_1+R_2)\right]R_x$$

$$R_2R_3 - \frac{V_{diff}}{V_{BAT}}(R_1R_3+R_2R_3) = \left[R_1 + \frac{V_{diff}}{V_{BAT}}(R_1+R_2)\right]R_x$$

$$\frac{R_2R_3 - \frac{V_{diff}}{V_{BAT}}(R_1R_3+R_2R_3)}{R_1 + \frac{V_{diff}}{V_{BAT}}(R_1+R_2)} = R_x \tag{4.2}$$

What we can glean from this derivation is that the change in the sensor's resistance, R_x, is proportional to the change in the physical phenomenon being measured. If the resistance, R_x can be determined from the bridge voltage difference, then the ultimate physical phenomenon can be computed based on the sensitivity value of the sensor. Many data acquisition systems provide the partial bridge, with known values for R_1 and R_3. In more complex measurement systems, R_2 can be adjusted under the zero-input condition so that the two sections of the bridge are balanced and $V_{diff} = 0$. After this is accomplished, any non-zero input will result in a value for V_{diff} that can be converted first to a value for R_x, and then to a value that estimates the physical phenomenon. The values of R_1 and R_3 are internally known to the system, and R_2 is known once the initial zero-input calibration is accomplished.

A Wheatstone bridge can also be used to measure Rx using the manual method. This is where R_1 and R_3 are set to identical values, R_2 is specified as a precision variable resistor that is adjusted until the bridge is balanced at V_{Diff} = 0 V. Since the bridge is balanced and R_1 is equal to R_2, then R_x is equal to the known value of the variable resistor. The variable resistor must have a good enough resistance resolution to obtain the desired R_x measurement. If cost is not an issue, the manual method is best accomplished by purchasing a piece of equipment that has the desired specifications.

Example 4.2 *A Wheatstone bridge circuit is built with V_{BAT} = 4.5 V, $R_1 = R_2$ = 10 kΩ, and both R_3 and R_x are thermistors that have a 10 kΩ resistance at a room temperature of 72 °F (see ohmmeter measurement in Figure 4.6).*

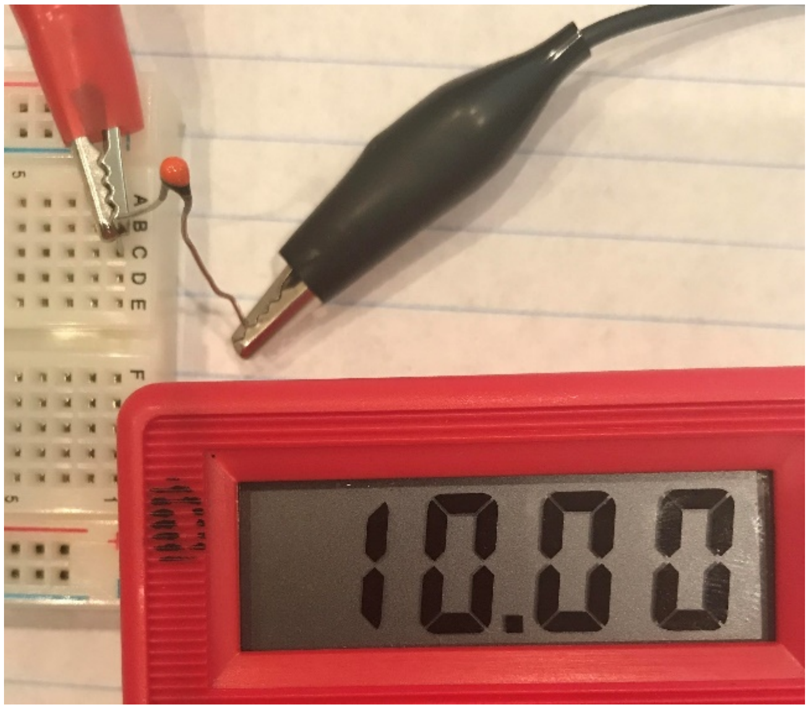


Figure 4.6. ***Ohmmeter Measurement:*** *Placing the two multimeter leads across the thermistor, the meter shows that the thermistor has a resistance of 10.00 kΩ, which corresponds to 72°F. Ohmmeter Measurement photo by Chad Davis.*

If R_x is heated up to ~ 93 °F by holding the device with your fingers and the bridge output voltage (V_{DB}) is equal to 0.27 V (shown in Figure 4.7), calculate the value of R_x.

Solution:

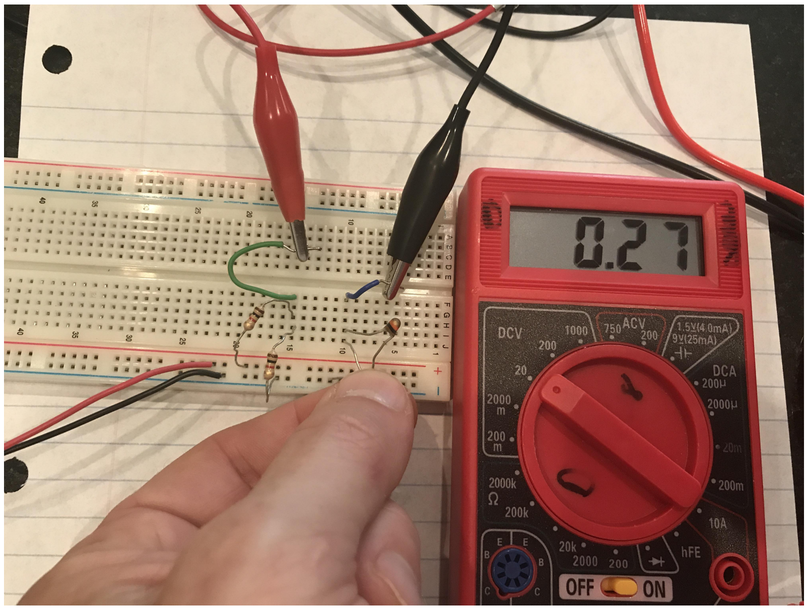


Figure 4.7. ***Voltage Measurement with Meter*** *The red and black leads from the multimeter are placed across the bridge to measure V_{diff}. The differential voltage is shown to be 0.27 volts. Measuring the bridge voltage photo by Chad Davis.*

Plugging in values to solve for R_x:

$$R_x = \frac{R_2R_3 - \frac{V_{diff}}{V_{BAT}}(R_1R_3 + R_2R_3)}{R_1 + \frac{V_{diff}}{V_{BAT}}(R_1 + R_2)}$$

$$R_x = \frac{10000 \cdot 10000 - \frac{0.27}{4.5}(10000 \cdot 10000 + 10000 \cdot 10000)}{10000 + \frac{0.27}{4.5}(10000 + 10000)}$$

$$R_x = 7,857\Omega \quad \text{(Figure 4.8 shows the measured resistance at} \sim 93\ {}^\circ\text{F is 7.76 k}\Omega\text{)}$$

Measure the resistance with the ohmmeter when holding the thermistor with your fingers (as shown in Figure 4.8) and calculate the % error between the calculated resistance and the measured resistance.

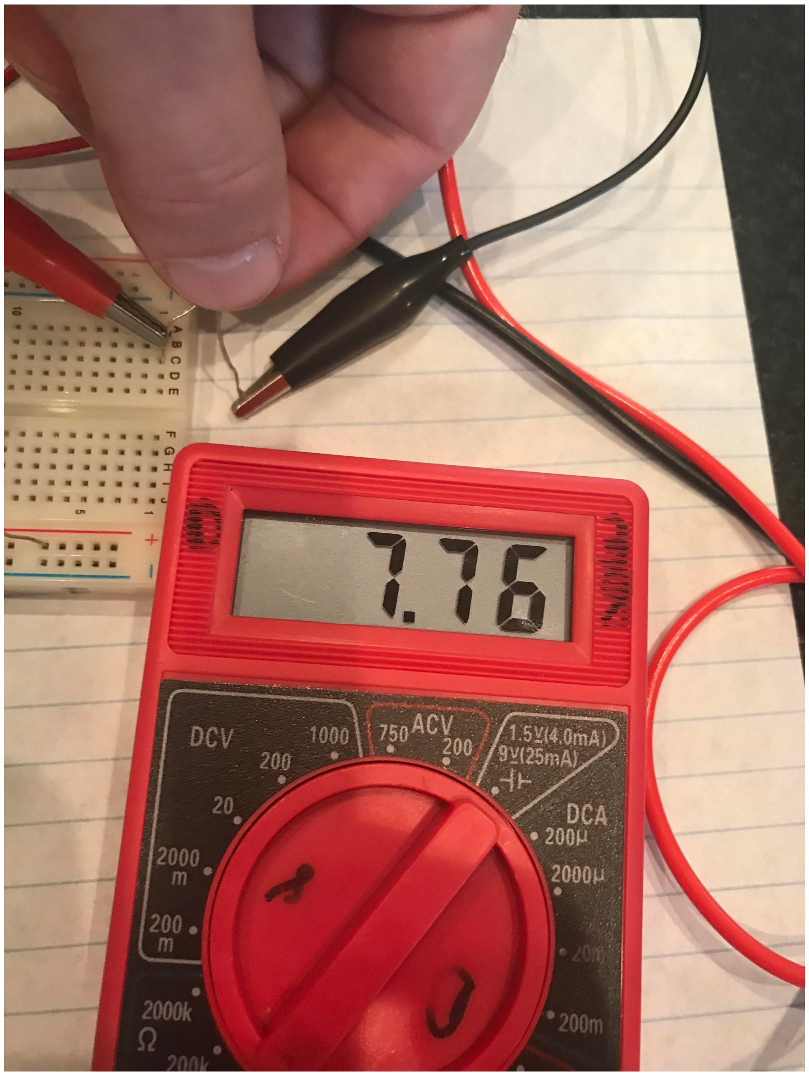


Figure 4.8. ***Measuring Bridge Thermistor*** *After holding the thermistor between two fingers to simulate warmer temperatures, the resistance of the thermistor is measured. Here the resistance is 7.76 kΩ, which correspodns to about 93 °F. Measuring Bridge Thermistor photo by Chad Davis.*

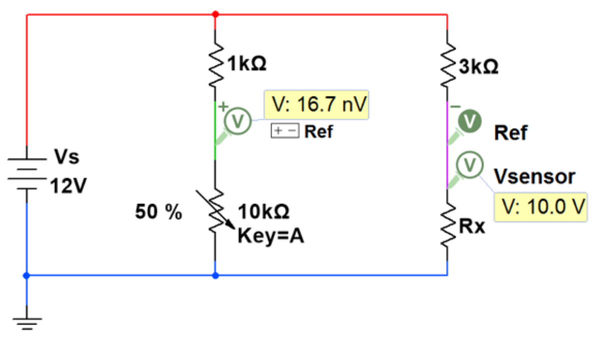
$$\% \text{ Error} = 100 \cdot \frac{\text{Measure Value - True Value}}{\text{True Value}}$$

$$\% \text{ Error} = 100 \cdot \frac{7.76\text{k}\Omega - 7.857\text{k}\Omega}{7.857\text{k}\Omega} = -1.24\% \quad \text{(Between measured and calculated } R_x\text{)}$$

Example 4.3 *Use the manual method described previously to determine the unknown resistance, R_x, in the Wheatstone bridge circuit below. Use R_x to solve for the sensor voltage. Assume 16.7 nV is ~ 0V.*

Solution:

The solution is shown in Figure 4.9 below:



Solution

Balanced since VDB = ~ 0V
R1 = 1k, R2 = 0.5*10k = 5k
R1/R2 = 1/5 = 0.2
Balanced if R3/Rx = R1/R2 = 0.2
Rx = R3/0.2 = 3k/0.2 = 15k

The votlage across Rx can be solved using VDR
VRx = 12*(Rx/(Rx+3k)) = 12*15/(15+3) = 10 V

Figure 4.9. ***Solution for Example 4.3:*** *The circuit is shown with measurements added. It should be noted that the voltage probe with the "+" sign is part of a differential probe. The other part of the differential probe is the V circle filled with green, and labeled "Ref."*

Example 4.4 *Using the circuit simulation shown here, determine the unknown resistance, R_x, in the Wheatstone bridge circuit below. Determine both R_x and the sensor voltage.*

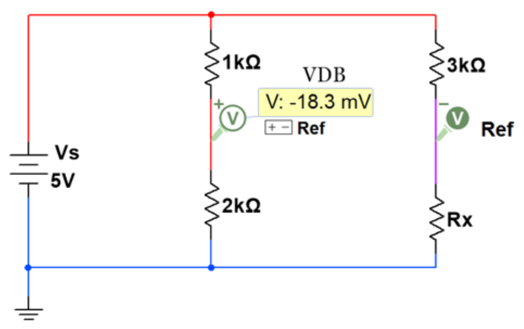


Figure 4.10. ***Solution for Example 4.4:*** *In the solution for Example 4.4, Multisim is used to place a differential probe across the bridge. It can be seen that the differential voltage, V_{diff}, for the bridge is -18.3 mV.*

Solution:

The bridge is not balanced, since V_{DB} is not 0 V. We know that: $R_1 = 1\text{k}\Omega$, $R_2 = 2\text{k}\Omega$, $R_3 = 3\text{k}\Omega$, adn $V_{DB} = -18.3$ mV. Using Equation 4.2 we have:

$$\begin{aligned} R_x &= \frac{R_2 R_3 - \frac{V_{diff}}{V_{BAT}}(R_1 R_2 + R_2 R_3)}{R_1 + \frac{V_{diff}}{V_{BAT}}(R_1 + R_3)} \\ &= \frac{2\text{k}\Omega \cdot 3\text{k}\Omega - \frac{-0.0183}{5}(1\text{k}\Omega \cdot 2\text{k}\Omega + 2\text{k}\Omega \cdot 3\text{k}\Omega)}{1\text{k}\Omega + \frac{-0.0183}{5}(1\text{k}\Omega + 3\text{k}\Omega)} \\ &= \mathbf{6.1k\Omega} \end{aligned}$$

Now we can solve for the sensor voltage (V_{Ref} in the figure) using the standard voltage divider:

$$\begin{aligned} V_{Ref} &= \frac{R_x}{R_3 + R_x} \\ &= \frac{6.1\text{k}\Omega}{3\text{k}\Omega + 6.1\text{k}\Omega} \\ &= \mathbf{3.35Vdc} \end{aligned}$$

4.2 Sensor Types

The following sections describe a number of common sensors. Those covered do not constitute an exhaustive list of sensors, but are commonly found in engineering measurements.

4.2.1 Strain Gauges

The strain gauge is a popular sensor for detecting the bending or flexing of a structure or specimen. The detection of micrometer-scale deflections in steel beams is one example where these sensors are frequently used. Most strain gauges are made of a metallic resistive element that changes resistance when it is stretched or compressed according to the equation $R = \rho L/A$. The effects of strain on the geometrical terms (A and L) are illustrated in Figure 4.11. The top left image in Figure 4.11 shows a strain gauge element in its unloaded state. The top right image demonstrates the compressed state, where the resistive element thickens ($A \uparrow$) and gets shorter ($L \downarrow$) under compression resulting in a reduced resistance ($R \downarrow$). The lower image demonstrates the tension state, where the resistive element thins ($A \downarrow$) and gets longer ($L \uparrow$) under tension resulting in an increased resistance ($R \uparrow$).

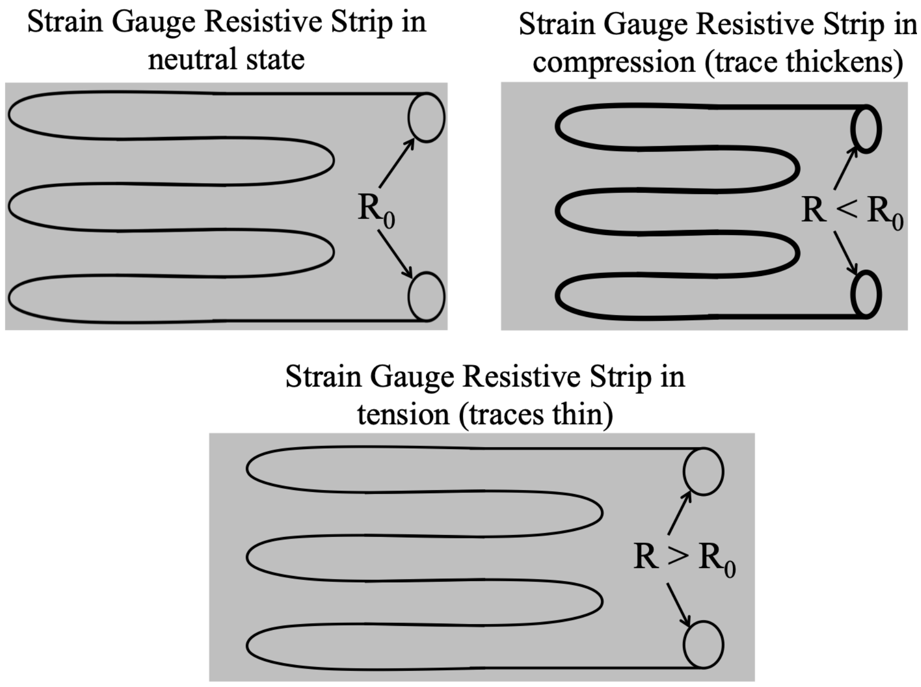

Strain Gauge Resistive Strip in neutral state
R_0
Strain Gauge Resistive Strip in compression (trace thickens)
$R < R_0$
Strain Gauge Resistive Strip in tension (traces thin)
$R > R_0$

Figure 4.11. ***Strain Gauge Element:*** *The strain gauge consists of a small resistive network that has some nominal resistance, R_0, when not under compression or tension. When the gauge is compressed, the traces shorten and thicken, reducing the resistance of the element so that $R < R_0$. When the gauge is in tension, the traces thin out and elongate so that resistance increases, making $R > R_0$.*

In the equation $R = \rho L/A$, strain's effect on the material property called resistivity (ρ) is not something that can be illustrated. Due to the *piezoresistive* effect, the resistivity (ρ) increases when strain is applied and depending on the material this change can be very significant in the overall resistance change. The effect of strain on these parameters factors into a constant of the strain gauge called *gauge factor*, which will be discussed later.

The strain gauge resistive element is what makes up the fourth element of a resistive bridge, as shown in the previous section. The resistor R_x represents the strain gauge. Generally, R_1 and R_3 are fixed inside the data acquisition signal conditioning hardware. R_2 is sometimes specified to be a potentiometer that can be manually adjusted under no-load conditions so that the bridge can be manually balanced (i.e. $V_x = 0$) before the load is applied. Effectively, this means setting R_2 to the no-load R_0 value of the strain gauge so that the bridge is balanced. This is shown schematically in Figure 4.12.

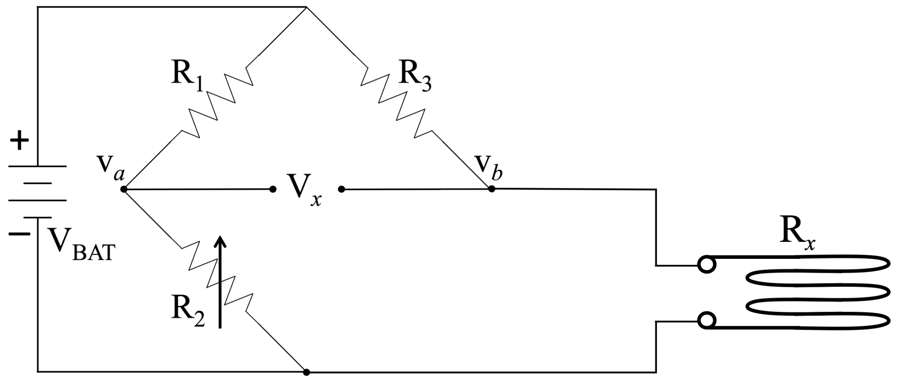


Figure 4.12. ***Strain Gauge as Part of a Resistor Bridge:*** *The strain gauge makes up the fourth leg of a full resistor bridge. In some systems, R_2 can be adjusted under no-load conditions so that the bridge is balanced (i.e. $V_x = 0$). Then when the strain gauge is exposed to compression or tension, the bridge becomes unbalanced and V_x can be used to find R_x from Equation 4.2.*

Once a load is applied to the balanced bridge the strain produces a voltage change and that voltage change can be used to calculate R_x using the equation in section 4.1.3. Another method to determine R_x is to manually adjust R_2 once the load is applied until the bridge is returned to a balanced state with V_x equal to zero. When the bridge is balanced R_x can be computed as $R_2 \cdot R_3 / R_1$. This manual method relies on using a precision potentiometer that can be turned to set V_x equal to 0.

Once the resistance R_x is determined, the strain can be computed based on the *gauge factor* of the strain gauge. For metal strain gauges the gauge factor usually has a value of 2. This parameter describes the sensitivity of the strain gauge to deformation. The equation for the gauge factor (GF) is

$$GF = \frac{\frac{\Delta R}{R_0}}{\frac{\Delta L}{L}} \tag{4.3}$$

where

ΔR = *the change in strain gauge resistance*

R_0 = *the unloaded resistance of the strain gauge*

ΔL = *the change in length of the gauge* ($\Delta L > 0$ *in tension;* $\Delta L < 0$ *in compression)*

L = *the unloaded length of the gauge*

In summary, the process is to balance the bridge with R_2 under no-load conditions. Then, since we know the value of R_2 that balances the bridge, we know the initial resistance of the strain gauge (R_0) is $R_3 \cdot R_2/R_1$. Strain itself is typically taken as $\Delta L/L$ so that once the strain gauge is loaded, V_x is measured and used to solve for R_x, then $\Delta R = R_x - R$ and strain is found by

$$\frac{\Delta L}{L} = \frac{\frac{R_x - R_0}{R_0}}{GF}$$

The reader should note that, when $R_x < R_0$, namely, the loaded gauge resistance is smaller than the unloaded resistance, and ΔR takes on a negative value, so that strain is negative. This corresponds to the compression state as shown in Figure 4.11. Conversely, when R_x is greater than the initial resistance of the gauge, the equation above makes it clear that strain will be positive, and this will correspond to tension.

Example 4.5 *A metal strain gauge with an unloaded resistance of 1 kΩ was put into a Wheatstone bridge circuit along with two 1 kΩ fixed resistors and a precision potentiometer. For the circuit in Figure 4.12, R_1 and R_3 were set as 1 kΩ fixed resistors and R_2 was set as the precision potentiometer and when the potentiometer was adjusted to 1.02 kΩ, the V_{DB} output voltage (the differential voltage* *between V_a and V_b) in the bridge was equal to 0 V. If R_x is the loaded strain gauge, calculate the resistance and strain of the strain gauge and determine whether it was a tension or a compression load.*

Solution:

Based on the description of the circuit the manual method is being used to determine the sensor resistance, R_x. Since $R_x = R_2$ when $R_1 = R_3$ for a balanced bridge, the result is $\mathbf{R_x = 1.02k\Omega}$.

To determine the strain, a GF approximation of 2 will be used since it is a metal strain gauge. From equation 4.3, $GF = 2 = (\Delta R/R)/(\Delta L/L) = (\Delta R/R)/$ strain $\Rightarrow Strain = (\Delta R/R)/GF = ((1.02 - 1)/1)/2 =$ **0.01 m/m**.

Since the ΔR was > 0, ΔL must also be > 0. Therefore, the strain gauge was under **tension**.

Example 4.6 *A strain gauge with a gauge factor of 2 and an unloaded resistance of 90 Ω was place in the R_x location in the Wheatstone bridge circuit in Figure 4.12 with a 5 V power supply. If an identical strain gauge was placed in the R_2 position and two 50 Ω fixed resistors were used for R_1 and R_3, what would be the value of R_x if the V_{DB} output voltage in the bridge was equal to 1 mV after a load was applied on strain gauge R_x? Also, calculate the strain of the strain gauge in units of µm/m and determine whether it was a tension or a compression load.*

Solution:

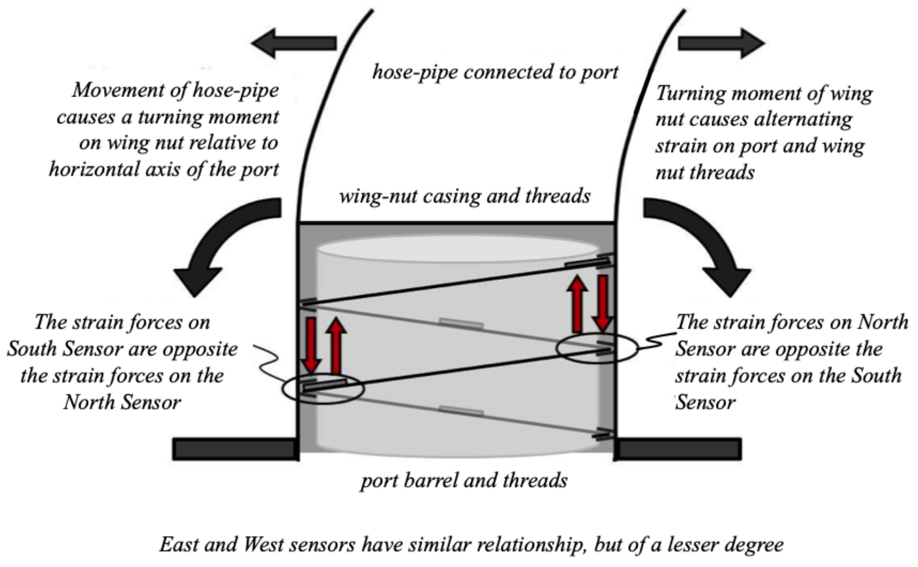
From Equation 4.2, we have:

$$\begin{aligned} R_x &= \frac{R_2R_3 - \frac{V_{diff}}{V_{BAT}}(R_1R_2 + R_2R_3)}{R_1 + \frac{V_{diff}}{V_{BAT}}(R_1 + R_2)} \\ &= \frac{50 \cdot 90 - \frac{0.001}{5}(50 \cdot 50 + 50 \cdot 90)}{50 + \frac{0.001}{5}(50 + 50)} \\ &= \mathbf{89.936\ \Omega} \end{aligned}$$

From Equation 4.3, we have:

$$\begin{aligned}
\text{strain} &= \frac{\frac{\Delta R}{R}}{GF} \\
&= \frac{\frac{\mathbf{89.936}-90}{90}}{2} \\
&= -0.0003554\text{m/m} \\
\textbf{strain} &= \mathbf{-355.4}\ \boldsymbol{\mu}\textbf{/m}
\end{aligned}$$

One example where strain gauges were used in personal experience was the analysis of metal fatigue in equipment that had a large coupling thread that was breaking under duress. Small sections of thread were removed and strain gauges were placed at four cardinal points (North, South, East, West, meaning 90° apart). Strain data were collected during the operation of the equipment and then analyzed offline. Figure 4.13 shows the threaded port and the location of the strain sensors. Offline analysis that shows a portion of the strain data over a one-second period is shown in Figure 4.14. The reader can see that opposite strain gauges are plotted and that the two strains are 180° out of phase with each other. In other words, when one side goes under compression, the other side is in tension. This showed that the hose-pipe and the main section were flexing at the threaded coupling and creating a turning moment along the diameter of the threads.



hose-pipe connected to port
Movement of hose-pipe causes a turning moment on wing nut relative to horizontal axis of the port
Turning moment of wing nut causes alternating strain on port and wing nut threads
wing-nut casing and threads
The strain forces on South Sensor are opposite the strain forces on the North Sensor
The strain forces on North Sensor are opposite the strain forces on the South Sensor
port barrel and threads
East and West sensors have similar relationship, but of a lesser degree

Figure 4.13. ***Placement of Strain Gauges to Analyze Thread Coupling:*** *The threaded coupling region has four strain gauges mounted at 90° intervals. During operation, the pipe and threaded body tend to vibrate and flex around the thread bore causing compression on one side and tension on the other repeatedly.*

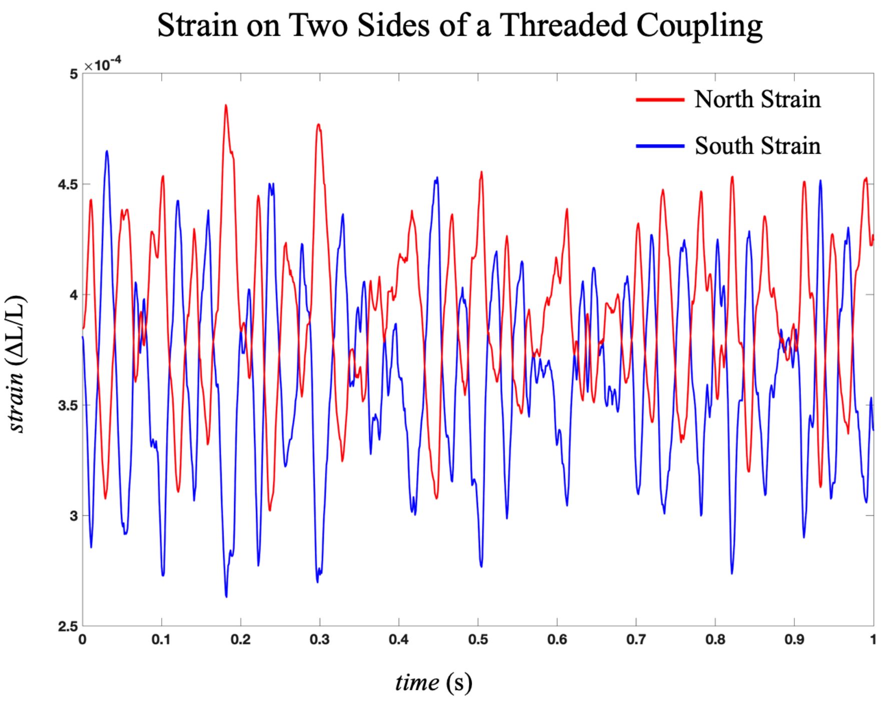


Figure 4.14. ***Strain Measurements for Opposite Strain Gauges on the Threaded Coupling:*** *The two traces show the strain is in opposite directions for the North and South sensors.*

4.2.2 Linear Variable Differential Transformer (LVDT)

While strain involves the change in length of an object due to tension or compression, displacement is the change in the position of an object. Displacement is a vector that includes the magnitude of the position change and the direction of the change. If the object moves in a straight line it is said to have rectilinear motion. The most popular type of sensor for measuring rectilinear motion is the Linear Variable Differential Transformer (LVDT) because this type of snesor can be used to measure linear movement with a range of up to several inches with an accuracy on the order of micrometers. The device uses the principle of induction in an exquisite application of that physical phenomenon. Figure 4.15 shows a cutaway image of an LVDT containing three separate coils wound around a movable ferrite core. The central coil is energized with AC, which induces an alternating magnetic field in the ferrite core. This alternating magnetic field, in turn, induces alternating current in each of the secondary coils on either side of the primary coil. The induced current (and, therefore voltage) in the secondary coils is determined by the amount of overlap between the ferrite core and each coil. If the core is moved more toward the back of the housing, the secondary coil on that end induces more voltage than the secondary coil on the other end. As the core moves out of the housing the induced voltage in the right coil decreases and the induced voltage in the coil near the opening of the housing increases. The differential voltages can be used to determine the position of the core from the following linear equation.

$$\Delta x = k \cdot \Delta V_{out} \tag{4.4}$$

where Δx is displacement and k is the *sensitivity* of the LVDT based on the excitation voltage of the primary coil. For a given excitation voltage, the sensitivity is a line with a slope that describes the relation between the output voltage (the difference between the induced voltages in the two secondary coils) and the displacement. National Instruments is a company that focuses significantly on instrumentation and sensors, and they suggest (https://www.ni.com/en-us/innovations/white-papers/06/measuring-position-and-displacement-with-lvdts.html):

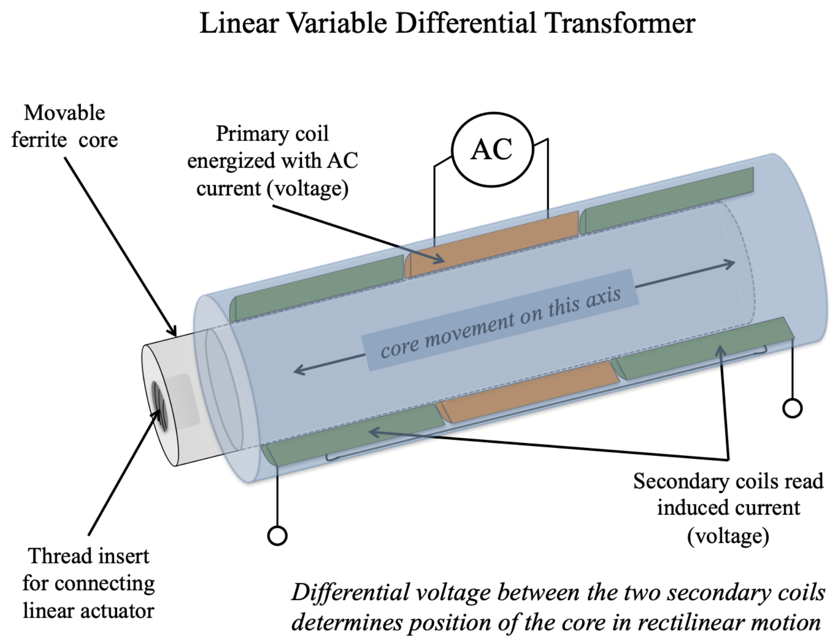
> "Broad ranges of LVDTs are available with linear ranges from at least ±50 cm down to ±1 mm. The time response is dependent on the equipment to which the core is connected. The units of an LVDT measurement are typically in mV/V/mm or mV/V/in. This indicates that for every volt of stimulation applied to the LVDT there is a definite feedback in mV per unit distance. A

> carefully manufactured LVDT can provide an output linear within ±0.25% over a range of core motion, with very fine resolution. The resolution is limited primarily by the ability of signal conditioning hardware to measure voltage changes."

The Volt term in the units mentioned in the quote above indicates the excitation voltage applied to the primary coil. From a transducer perspective, the LVDT treats the displacement as the input, and the differential voltage between the two secondary coils as the output. Thus, for a given excitation voltage, the slope of the sensitivity curve is volts per displacement ($V_{out}/\Delta x$). Other sensors sometimes provide sensitivity in the opposite sense so that the numerator of the slope is the change in the physical phenomenon, relative to some change in the electrical signal in the denominator (e.g. °C/mV, or PSI/mV). It is also important to recognize that a core position precisely in the middle produces a voltage difference of 0 volts between the two secondary coils.

Because any movement of the core must physically change the induced voltage in each of the secondary coils, the LVDT is essentially capable of infinite resolution. However, as stated in the quote above, the ability of the signal conditioning circuit to detect small voltage differences is limited and becomes the critical factor in determining LVDT resolution. With good circuit design, LVDTs are available that can measure displacements in the mirco-inch range.

LVDTs are found in a number of industrial applications where accuracy and environment are of importance. For example, in the aviation industry the LVDT can be used to determine the precise position of a control surface with great accuracy. Because commercial airliners fly at tens of thousands of feet, temperature is a significant issue, so a sensor that is robust to sub-zero temperatures is useful. LVDTs also find use in balancing rotating machinery. If the LVDT is spring-loaded and placed in the housing of the rotating device so that the tip is in contact with the rotating element, then any eccentricity of the rotating element (which adversely affects balance) can be detected. Indeed, a time-based trace of the measurement during rotation can describe the entire profile of the (presumably) circular component.



Linear Variable Differential Transformer
Movable ferrite core
Primary coil energized with AC current (voltage)
AC
core movement on this axis
Secondary coils read induced current (voltage)
Thread insert for connecting linear actuator
Differential voltage between the two secondary coils determines position of the core in rectilinear motion

Figure 4.15. ***Linear Variable Differential Transformer:*** *The Linear Variable Differential Transformer uses three coils and the principle of induction to measure linear displacement (movement) of an external item. There is no electrical contact between the item whose linear motion is being measured and the electrical circuit of the sensor. An alternating current energizes the primary coil, which induces a magnetic field in the ferrite core. The ferrite core is free to move in the barrel of the sensor, and is attached to the moving item. The magnetic field in the core induces current in the secondary coils depending on the amount of overlap between the core and each coil. These coils are electrically connected and the difference between the two determines the position of the core. Clearly, if the core is centered the current (or voltage) differential is zero.*

4.2.3 Potentiometric Sensing

Another method of measuring position can be accomplished with a potentiometer. A potentiometer is simply a variable resistor. The device generally consists of three terminals. Two of the terminals make up the ends of the internal resistive element so that the resistance measured between these two terminals is always the same, and corresponds to the potentiometers nominal resistance marking. A third terminal is connected to the *wiper*, which is the variable element. As the wiper moves across the resistive element the resistance between the wiper terminal and the end terminals changes. When the wiper is moved all the way to one end, the resistance between the wiper and that terminal is near zero and the resistance between the wiper and the other terminal is nearly the nominal resistance of the potentiometer. A circuit can easily be constructed that supplies a known voltage across the end terminals of the potentiometer, and then the voltage at the wiper terminal can be measured in order to determine the position change of the wiper. Because the three-terminal configuration is essentially a voltage divider, the voltage at the wiper varies as the wiper position varies. Figure 4.16 shows a schematic of a potentiometric measurement system.

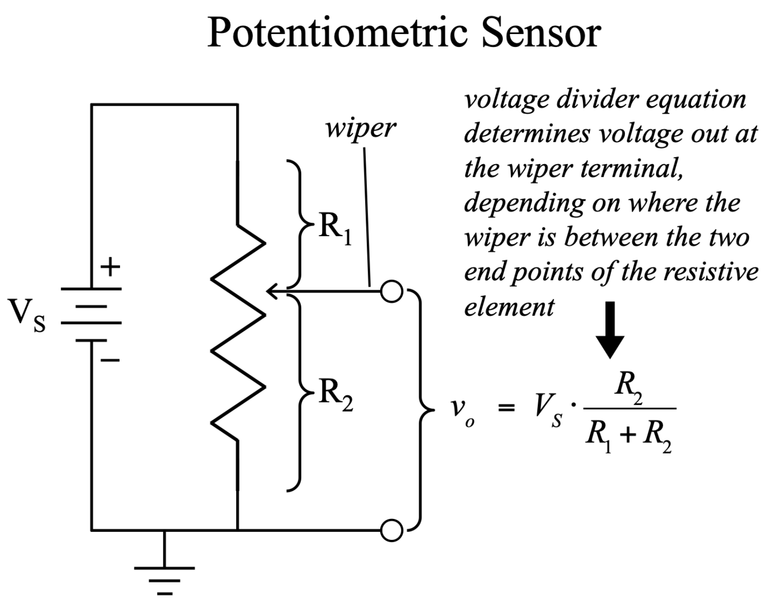

Potentiometric Sensor
wiper
voltage divider equation determines voltage out at the wiper terminal, depending on where the wiper is between the two end points of the resistive element
V_S
+
−
R_1
R_2
$v_o = V_S \cdot \frac{R_2}{R_1 + R_2}$

Figure 4.16. ***Potentiometric Sensor:*** *The potentiometric sensor is based on a variable resistor and the classic voltage divider equation. The resistance between the wiper terminal and either end of the resistive element changes as the wiper is moved. If a voltage is placed across the resistive element, a voltage proportional to the wiper position can be measured at the wiper terminal. The maximum voltage at the wiper will be just smaller than the supply voltage, and the minimum wiper voltage will be just a bit greater than zero.*

These devices are frequently circular (the resistive element is wound in a circular fashion) so that the wiper is controlled by a rotary arm. Indeed, the throttle position sensor on many cars is simply a potentiometric sensor. A disassembled throttle position sensor is shown in Figure 4.17.

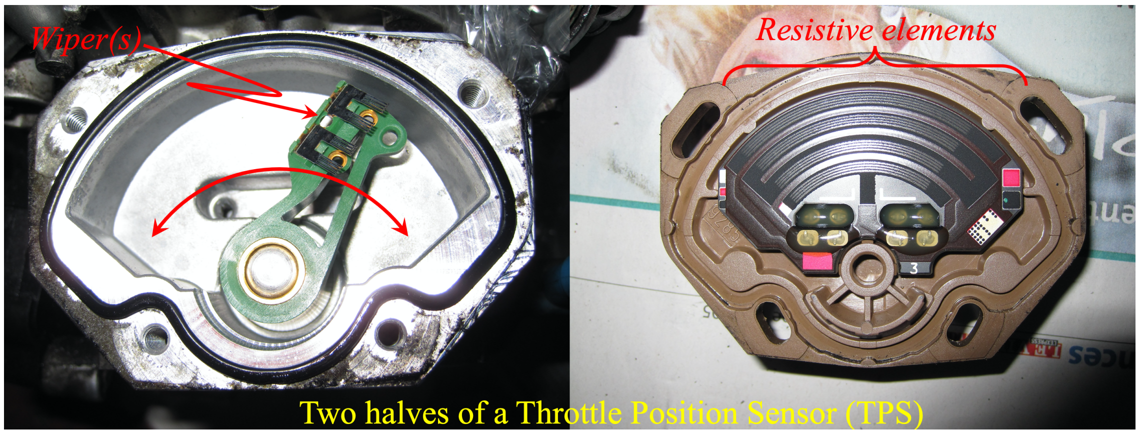


Figure 4.17. ***Throttle Position Sensor:*** *A typical throttle position sensor on a car has a housing with a central section that rotates. The rotating section attaches to the throttle shaft. Internally, this rotating section is the* wiper *of the variable resistor (left side of figure). The resistive element is shown on the right. The wipers move across the resistive element in an arc. One terminal of the resistive element is connected to ground and the other terminal is connected to a tightly regulated voltage. The wiper forms a voltage divider and the wiper voltage represents the throttle position. Left Image: ©The RedBurn CC-BY-SA-4.0,* `https://commons.wikimedia.org/wiki/File:Throttle_position_sensor_-_1.jpg`*. Right Image: ©The RedBurn CC-BY-SA-4.0,* `https://commons.wikimedia.org/wiki/File:Throttle_position_sensor_-_2.jpg`

Example 4.7 *The following two types of 1000 Ohm potentiometers were used as position sensors to adjust volume control on a car radio.*

a A single turn rotary potentiometer

b A 10-turn rotary potentiometer that had a helical internal resistive element.

A 12 Volt source was connected to the ends of each of the potentiometers. If it was experimentally determined that a voltage range between 6 and 9 volts corresponds to the audio level range that is most desirable in a car, determine the amount of corresponding angular displacement of each of the potentiometers.

Solution

The following shows the resistance calculations for any 1000 Ω potentiometer that is connected a with 12 Volt source.

$$V_0 = V_s \frac{R_2}{R_1 + R_2} \quad = \quad V_s \frac{x \cdot R_{pot}}{R_{pot}} \quad = \quad V_s \cdot x$$

R_{pot} is the nominal resistance of the potentiometer, which is the total resistance between the two terminal ends. x represents the percentage change of the wiper. When x equals 0%, $R_2 = 0$ and $V_0 = 0$. When x equals 100%, $R_2 = R_{pot}$ and $V_0 = V_s$. In this example $R_{pot} = 1000\Omega$ and $V_s = 12$ V. Solving for x at the two voltage levels of interest yields:

$$\text{Case 1:} \quad V_0 = 6 \qquad x = \frac{V_0}{V_s} = \frac{6}{12} = 0.5$$

$$\text{Case 1:} \quad V_0 = 9 \qquad x = \frac{V_0}{V_s} = \frac{9}{12} = 0.75$$

For the single turn potentiometers, the total angular rotation is 360° so if x is 0.5 to 0.75 in the desired audio range the range of angular displacement would be from 180° to 270°, which would give a change of 90°.

For the 10-turn potentiometers, the total angular rotation is $10 \cdot 360°$ so if x is 0.5 to 0.75 in the desired audio range the range of angular displacement would be from 1800° to 2700°, which would give a change of 900°.

It was decided that the 10-turn potentiometer would allow for better fine tuning of the audio level for this application.

4.2.4 Ranging Sensors

Some displacement sensors are commonly referred to as ranging sensors. There are many ways to implement a ranging sensor, with the three most common types being infrared (IR), ultrasonic, and laser. For IR ranging sensors, an IR LED sends a signal that reflects off of a surface and is received by a phototransistor that converts IR light to voltage. IR range-finding is the most economical form and by far the easiest to implement (you just read the voltage pin that corresponds to distance), but it has a limited range and is not very accurate due to a nonlinear relationship between range and voltage. For example, the Sharp GP2Y0A21YK0F (`https://www.pololu.com/product/136`) has a range of only

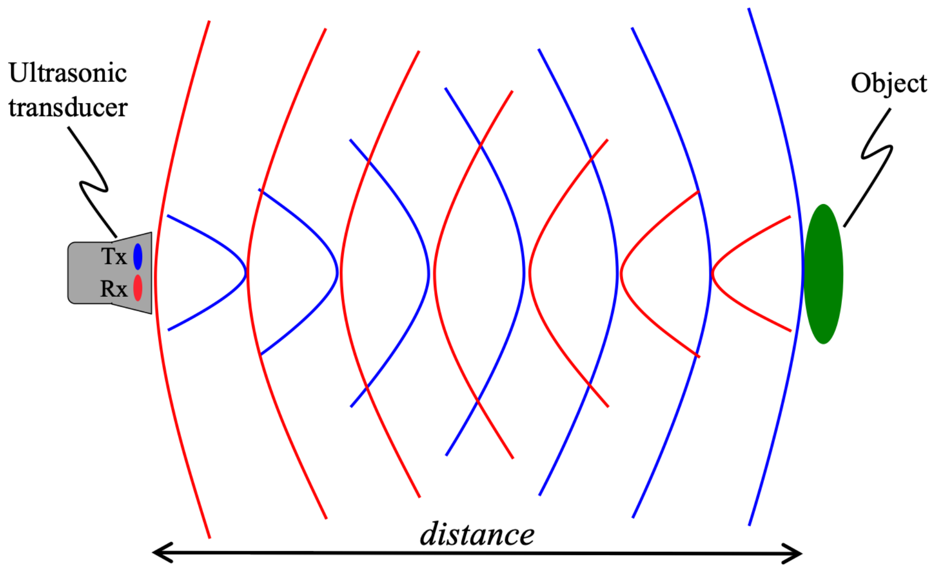
10 to 80 cm. As the previous link shows, short range versions of this Sharp sensor can be used to get the minimum range down to 2 cm (https://www.pololu.com/product/2450).

Ultrasonic ranging sensors are also commonly used to make distance measurements. They have the advantage of eliminating the need for physical contact with the surface whose distance is being measured. The ultrasonic sensor is a relatively simple device that relies on ultrasonic sound transmission. These sensors operate in the tens of kilohertz (small range finding applications) into the tens and hundreds of megahertz for medical applications. The underlying principle for IR, ultrasonic, and laser range finders is similar to radar; namely, a transmit signal is broadcast, reflects off of a surface, and is returned to the sensor. For ultrasonic ranging, the time delay between transmission and reception, accounting for the speed of sound and dividing by two (for the round trip), determines the distance. This process is shown in Figure 4.18.

$$\text{Range} = \frac{1}{2} \cdot \textit{speed of sound} \cdot \Delta t \tag{4.5}$$

$$\begin{aligned} &\textit{where } \Delta t \textit{ is the travel time} \\ &\textit{speed of sound } \approx 340.29\text{m/s} \end{aligned}$$

The transmitter and receiver are co-located in the sensor so that angled deflections from an object minimally affect the distance measurement. Clearly, as the baseline between the transmit and receive elements increases, the geometry of the reflection path and the transmit path begin to diverge.



$$\begin{aligned} distance &= s \bullet \frac{t_{Rx} - t_{Tx}}{2} \\ \text{where } t_{Rx} &= \text{the receive time} \\ t_{Tx} &= \text{the transmit time} \\ s &= \text{the speed of sound } (\approx 340.3\ m/s) \end{aligned}$$

Figure 4.18. ***Ultrasonic Range Finder:*** *The ultrasonic range-finder measures distance from the sensor to an object. The principle is based on the idea that the ultrasonic transmission proceeds out from the sensor, bounces off an object, and returns to the sensor. The sensor tracks the time elapsed between the transmission and reception of the ultrasonic waveform, and uses this to compute the distance.*

The speed of sound in earthbound environments (say a factory assembly process, for example) is mostly a function of temperature. For typical altitudes in most cities, we consider the speed of sound to be determined by $s\ (m/s) = 331.5 + 0.6 \cdot T(°C)$. So, at 20°C the approximate speed of sound is 343.5 m/s. At 25°C the speed of sound becomes 346.5 m/s. To put this in perspective, if a robotic system needs to ensure that an object remains about 20 cm from some operating mechanism this translates to a possible time differential of 1.01 μsecs depending on the temperature being somewhere between 20°C and 25°C. The lesson to be learned is that in many environments the temperature may vary slightly, but that variation results in a small time differential in most ultrasonic ranging applications. Clearly radar itself is a much more effective ranging tool at larger distances. Indeed, most ultrasonic ranging devices are not feasible if the range being measured exceeds a few meters. However, for smaller ranges they are a very cost-effective way to monitor distance.

Like ultrasonic ranging, the equation to determine range in a laser ranging device is simply the travel time multiplied by the speed of the wave and dividing the result by 2 to account for the send and receive time. Equation 4.6 shows that the only difference for the laser range finding equation is that the speed of light is much faster than the speed of sound. This faster speed allows for the measurement of much longer ranges, but it prevents achieving precise, sub-millimeter accuracies. For example, some military grade laser range finders can measure ranges over 20 km, which is at a completely different level than IR or ultrasonic range finders. You can reference the Electromechanical Systems book by Chad Davis here (`https://shareok.org/handle/11244/316816`).

$$\text{Range} = \frac{1}{2} \cdot c \cdot \Delta t \tag{4.6}$$

where c is the speed of light, $c = 299,792,458$ m/s, and Δt is the difference between the send and receive time.

4.2.5 Accelerometers

Using a displacement or ranging sensor and a timing source is a method that is often used to determine the velocity of an object, as velocity is equal to the displacement divided by time. However, when acceleration is to be measured a sensor called an accelerometer is usually used to measure acceleration directly. There are at least two relevant ways to determine the acceleration in this setting. First, we can simply recall Newton's third law, which we typically express as F = ma, meaning that acceleration can be understood as $a = F/m$. It should be noted that acceleration is also frequently defined by the g-force, where 1g is 9.8 m/sec^2, 2g is 19.6 m/sec^2, etc. Noting the relationship between force, mass, and acceleration it is not surprising that many accelerometers use a small, delicate mass suspended inside a casing to measure acceleration. A second way of approaching the accelerometer is to note that acceleration is the derivative of velocity (i.e. the change in velocity over time). The reader may recall that velocity (not speed) is a vector that describes both a magnitude (speed) and a direction. A change in either of these facets constitutes a change in velocity. As a side note, this is why an airplane that flies at the same airspeed through a turning arc is considered to be in accelerated flight. In this case, a suspended mass is moving with the same speed and direction as the enclosing case, and if the enclosing case is slowed down, the inertia of the suspended mass causes it to briefly change its position in the case before returning to its stable resting position. Figure 4.19 shows a depiction of a small mass suspended by a delicate arm inside an enclosure.

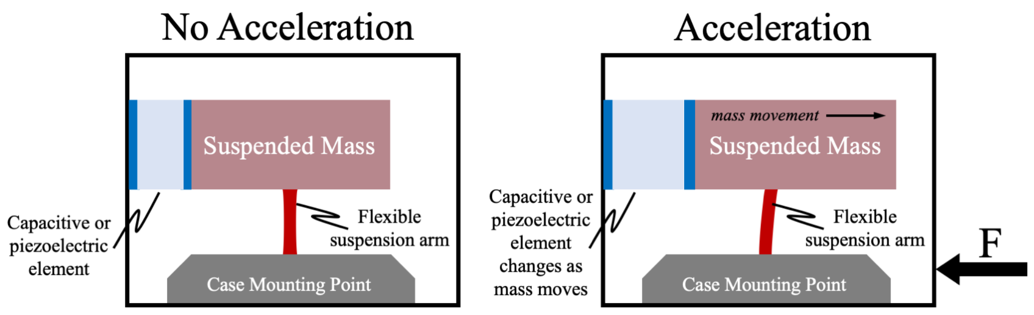


Figure 4.19. ***Suspended Mass Accelerometer:*** *The suspended mass inside the accelerometer is suspended with a flexible arm. When the case changes velocity (is exposed to a force) the mass shifts and the capacitive or piezoelectric element changes value.*

Electrical connections to the device allow the transduction of the displacement of the mass to an electrical signal. These displacements are very small and have sudden changes so methods other than the displacement sensors previously discussed are usually used. The two most common ways to measure displacement of the mass in an accelerometer are piezoelectric sensors or capacitive displacement sensors. For piezoelectric sensing elements, the deformation of a piezoelectric crystal induces a voltage. Compression can cause a negative voltage, while expansion causes a positive voltage change. On the other hand, a capacitive sensing element must have a method of assessing the capacitance and converting that to a displacement of the mass. The reader may recall from a past physics course that the capacitance of a parallel-plate capacitor is defined as $C = \epsilon A/d$ where A is the plate area, ϵ is the permittivity of the dielectric separating the plates (relative to vacuum), and d is the plate separation. Clearly this can be solved for $d = \epsilon A/C$. Ultimately the accelerometer is converting the internal mass movement, in reaction to a force applied to the external case, into an electrical signal. In an accelerometer with a bipolar power supply (+ and - V), the acceleration values can take on both positive and negative voltages representing both positive and negative acceleration. On the other hand, many simplified devices are only powered from a single, positive power supply. In this case, the no-acceleration voltage output is some positive value between 0 and the power supply voltage. Because both positive and negative acceleration are converted to positive voltages, the user must know the DC offset for the no-acceleration case and remove this value from that data in order to properly measure the acceleration of the subject.

One significant use of accelerometers is to measure vibration. Consider a point on a beam in some structure that is undergoing an earthquake. The beam is flexing back and forth, so that we say it is vibrating. Therefore, the subject point on the beam is moving back and forth. Thus, we have displacement over time, which is velocity, and the displacement changes directions regularly, causing accelerated movement. If we mount an accelerometer to the beam, we can determine the frequency (or frequencies) of vibration. Figure 4.20 shows how an accelerometer can be placed on a beam to measure vibration.

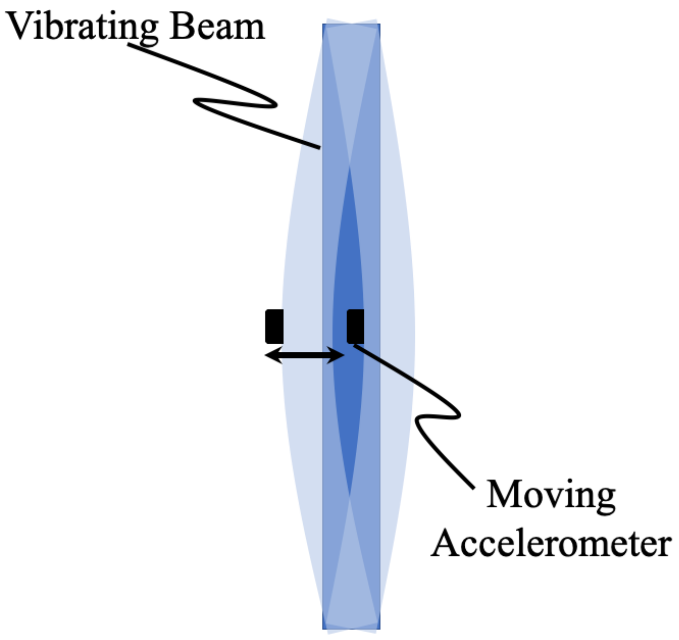


Figure 4.20. ***Accelerometer to Measure Vibration:*** *As the beam flexes (movement exaggerated to demonstrate the principal) the accelerometer moves back and forth. This creates an electrical signal proportional to the G's to which the beam is being exposed.*

Accelerometers, like most sensors, have constraints associated with their performance. The ability of the accelerometer to detect vibration, for example, is closely associated with the stiffness of the suspension arm, and is reflected in the *bandwidth* of the device. If an accelerometer has a bandwidth of 10 Hz, clearly one cannot measure a vibration with a frequency of 20 Hz. Therefore the user must have some understanding of the physical system being measured when choosing an accelerometer in order to choose a device that can adequately measure the system's acceleration (vibration).

Another parameter that must be identified is the sensitivity of the device. In this case, the sensitivity is generally specified in volts/G or mVolts/G. It should be noted, though, that a wider sensitivity range reduces the ability to discriminate finer gradations of acceleration. For example, a device that is powered by a single-sided +5 volts and is capable of measuring ±5 G's is less sensitive to different levels of acceleration than a device powered by the same +5 volts, but only capable of measuring ±1 G. This is partly because the data acquisition systems are limited to finite step resolutions, as covered in Chapter 2. It is noteworthy that, once again, we see why it is critical to select a dynamic range in the data acquisition system that just encompasses the possible incoming voltage values so that the step resolution can be as finely tuned to the desired sensitivity as possible.

Another specification generally found with most accelerometers relates to the instantaneous G's that the device can handle without being damaged. This is generally related to accidental dropping, or other things that can generate very large acceleration (in the case of dropping the sensor, the acceleration is negative–or deceleration–when the device impacts the ground; this can be thousands of G's). Large accelerations can damage or completely destroy the suspension arm and result in a faulty device. Even if an output is achieved from the damaged device, it can not be considered reliable.

Acceleration (vibration) comes in many forms. One example is the classic mass-spring-damper system, shown in Figure 4.21.

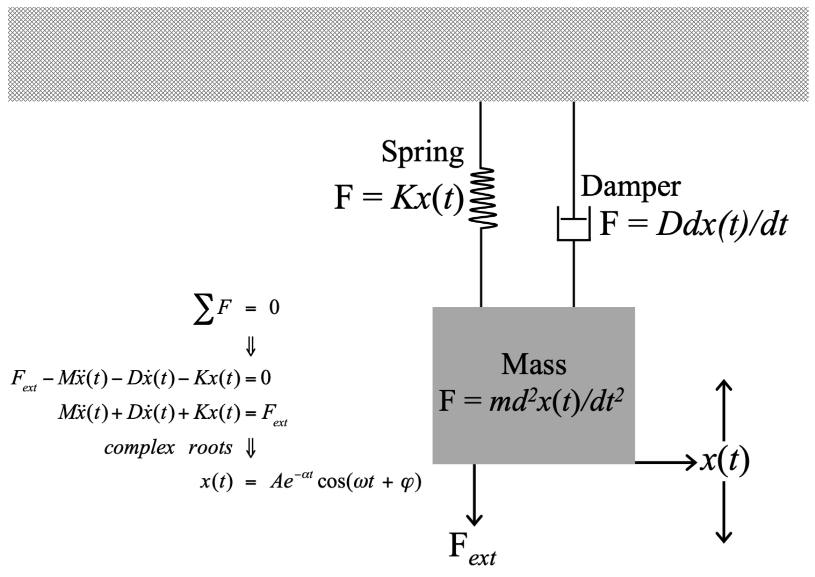

Spring
$F = Kx(t)$
Damper
$F = Ddx(t)/dt$
$\sum F = 0$
$\Downarrow$
$F_{ext} - M\ddot{x}(t) - D\dot{x}(t) - Kx(t) = 0$
$M\ddot{x}(t) + D\dot{x}(t) + Kx(t) = F_{ext}$
complex roots $\Downarrow$
$x(t) = Ae^{-\alpha t}\cos(\omega t + \varphi)$
Mass
$F = md^2x(t)/dt^2$
$x(t)$
F_{ext}

Figure 4.21. ***Mass-Spring-Damper System:*** *In the mass-spring-damper system a mass is suspended from a fixed surface by a spring and damper. Using Newton's Second Law we let the sum of the forces acting on the mass be zero, so that we can form a linear, constant-coefficient, differential equation as shown. The homogeneous solution of this equation is found from the roots when we assume an unforced system (in the Laplace Domain, this polynomial on s forms the* characteristic equation*).*

This system is described by a second-order differential equation

$$M\frac{d^2x(t)}{dt^2} + D\frac{dx(t)}{dt} + Kx(t) = F_{ext}(t)$$

In the Laplace Domain, assuming an impulse input $\delta(t)$ (i.e. $F_{ext}(t) = \delta(t)$), we can find the transfer function of the system to be

$$H(s) = \frac{\frac{1}{M}}{s^2 + \frac{D}{M}s + \frac{K}{M}}$$

We call the denominator of this transfer function the *characteristic equation*, and the roots of this polynomial tell us the unforced behavior of the system. It is noteworthy that a second-order differential equation of this sort clearly has two roots, and these roots can be distinct and real, repeated and real, or a complex conjugate pair (like magnetic poles that only come in a dipole pair, complex roots must always come in a complex conjugate pair). If the roots are real and distinct, $x(t)$ takes the form $Ae^{-\alpha t} + Be^{-\beta t}$ for $t \geq 0$. On the other hand, if the roots are repeated and real, the solution is of the form $x(t) = Ae^{-\alpha t} + Bte^{-\alpha t}$ for $t \geq 0$. Finally, if the roots form a complex conjugate pair, the solution has the form of a decaying sinusoid, mathematically described as $x(t) = 2|K|e^{-\alpha t}\cos\left(\omega t + \phi\right)$.

The data shown in Figure 4.22 are taken from a mass-spring-damper experiment using an accelerometer. The accelerometer is powered by a single +5 volt power supply, but is capable of measuring both positive and negative acceleration. This means the zero-acceleration point is represented by the DC offset in the signal. Furthermore, the lower panel shows that there is a small amount of noise in the signal.

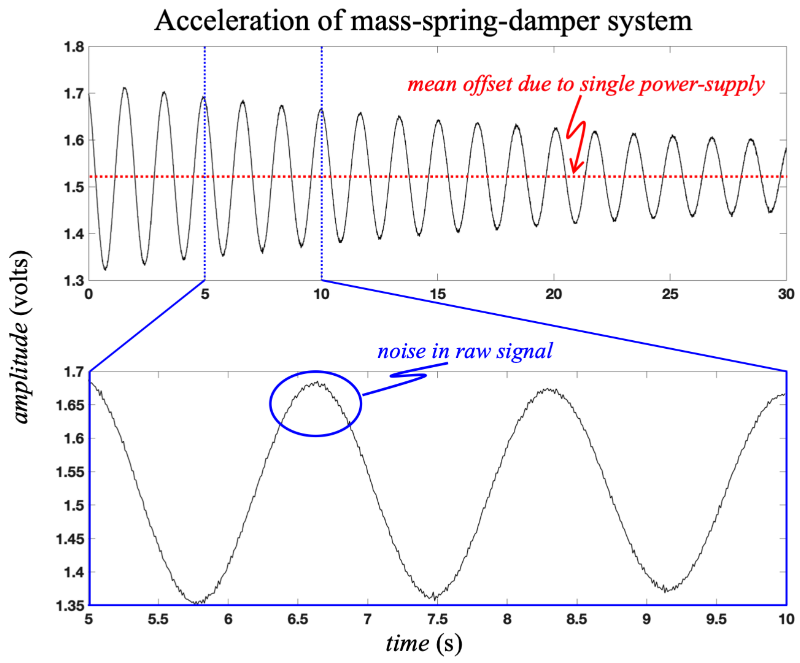


Figure 4.22. ***Accelerometer Data from a Mass-Spring-Damper System:*** *In the upper panel, the oscillations are sinusoidal and decaying. These data represent acceleration of the Mass being observed with a DC offset and noise. The lower panel zooms in on the 5-10 second time epoch to more clearly show the noise in the data. If one uses the sinusoidal peaks to detect the decay rate, or the oscillatory period, the noise can prevent accurate identification of the exact peak point.*

If one is to glean information from this system, knowledge of the accelerometer sensitivity is important. However, because the homogeneous solution to the equation tells us the unforced behavior, we can find the resonance and decay rate of the system simply from the voltage data. Of course, the estimated decay process will not be correct unless the mean value (DC offset) is first removed from the data. By removing the mean and also filtering the data to smooth it, the waveform can then be used for analysis. The fundamental frequency can easily be found from a spectral analysis (e.g. the Power Spectrum, which is the Fourier Transform times its complex conjugate), but also by identifying the interval between successive peaks (e.g. estimate the period and then apply $f = 1/T$). The decay rate (also known as the time constant in decaying exponential equations) can be found by identifying the sinusoidal peaks (both time and magnitude) and putting these ordered pairs into an exponential fit algorithm. The modified data is shown in Figure 4.23, as well as the identified peaks.

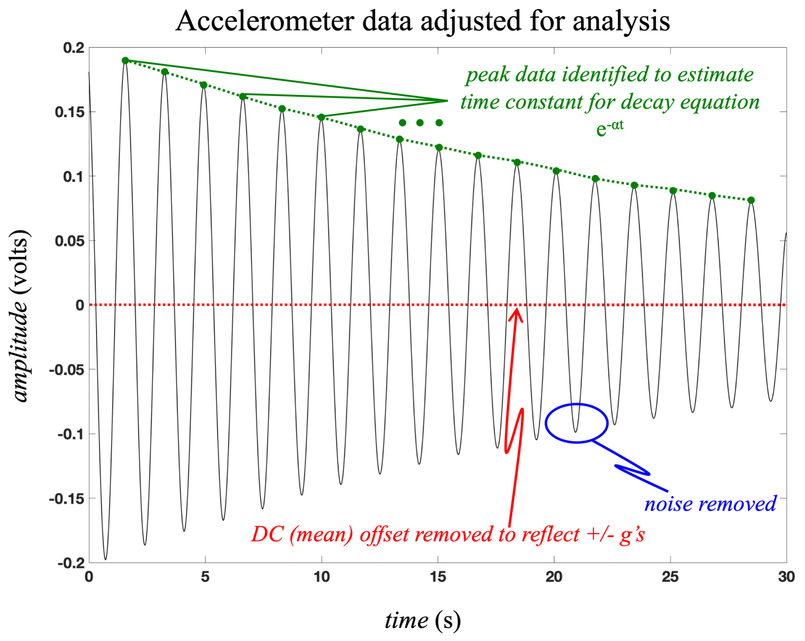


Figure 4.23. ***Accelerometer Data After Adjustment:*** *After mean removal and lowpass filtering, the peaks are easily identified and the mean period can easily be determined (and, therefore, the frequency). The peaks can also be used to determine the time constant for the exponential decay*

Indeed, it is noteworthy that, if we measure the initial displacement of the mass, we can construct the actual $x(t)$ without knowing the accelerometer parameters. By simply displacing the mass to some extent and then releasing it at $t = 0$, the sinusoid is a cosine with a phase of 0. Since ω (oscillation frequency) and α (decay rate) can be assessed with the voltage data from the accelerometer, and the initial displacement (A or $2|K|$ in Figure 4.21, or the solution equation for complex roots, respectively) can be measured, the position equation can be determined experimentally. Chapter 6 shows a case study that provides a practical application of accelerometers.

4.2.6 Pressure Sensor

In some engineering applications pressure must be measured. A common example is oil pressure in a car engine. Because lubrication of the moving components is critical to engine life, oil pressure in the system is an important parameter to track. Another example of pressure measurement is atmospheric pressure, which is often reported along with temperature and humidity. When measuring pressure, a *pressure sensor* is used to convert the pressure under consideration to an electrical signal. There are several different technologies employed to transduce the pressure into an electrical signal, but the reader should also be aware that there are also a number of different types of pressure to measure. For example, *gauge pressure* is the pressure relative to atmospheric pressure. On the other hand *absolute pressure* is measured relative to a perfect vacuum. Further, *vacuum pressure* is typically the pressure *below* atmospheric pressure (i.e. if we consider local atmospheric pressure to be a baseline, the vacuum pressure is that pressure below the baseline–this is what we measure when checking the vacuum system in a car. In each case, it can be observed that the measurement is between a pressure of interest, and some other pressure baseline (which might be zero in the case of vacuum).

There are a number of different technologies to accomplish pressure transduction, but all involve the physical deflection of a sensor surface. This occurs on a micro-scale and is not necessarily a deflection that is readily obvious to the eye. For example, a capacitive pressure sensor operates on the principle of a parallel plate capacitor. In this instance, given a fixed dielectric material and plate area, the capacitance is a function of the distance separating the two plates. If one of those plates is subjected to pressure different from the other plate then the plate distance changes and, therefore, the capacitance changes. The piezoelectric-based sensors take advantage of the electric properties of piezoelectric material under deformation. When a piezoelectric crystal is deformed it generates an electric potential. However, this potential is generated *only during the deformation*. Once the deformation is stable, no more electrical potential is generated. In other words, the piezoelectric effect is a *dynamic* process. Long term pressure measurement with this type of device would require some integration process to track pressure during stable times in the system. As the reader may have surmised a pressure sensor can also be based on strain gauge technology. This technology (described in section 4.2.1) is very well suited for small deflections of a pressure-sensing diaphragm. Not surprisingly, the LVDT technology of section 4.2.3 can also be used to measure pressure. In this case the movable ferrite core of the linear variable differential transformer (LVDT) is coupled to the pressure sensing surface so that any small deflections in the surface are tracked by the LVDT.

Pressure transducers have a number of applications. Pressure is often used to determine tank fluid level in large industrial applications. Obviously fluid has some weight per unit volume, so that a pressure sensor mounted in the bottom of a tank can detect the weight of the fluid and, therefore, the volume. This, of course, assumes that one knows the density of the fluid being tracked, which is a reasonable assumption. Pressure transducers are also used in the automotive industry. Most of the fuel economy measures gained since the 1970s are due to computerized control of the combustion process. One of the sensors used in this computer control is the Manifold Absolute Pressure (MAP) sensor. Air pressure and density allow a computation of mass air flow, which gives the computer key information for determining the amount of fuel to inject for optimum fuel burn. As the throttle is opened, the MAP increases, calling for greater fuel quantities to maintain the proper air-fuel mixture. Tire pressure sensor systems are a good example of gauge pressure measurements being taken at each wheel and transmitted into the cabin computer. Yet another example is the oil pressure measurement sensor. Another transportation example is the vacuum gauge in a general aviation aircraft, which uses a bourdon tube as discussed in section 2.1. Of course, there are many other examples across a number of application areas.

4.2.7 Encoders

Many sensors have been previously discussed that are used primarily to measure linear displacement, but angular displacement is usually measured with a sensor called an encoder. Encoders are devices that convert angular position or motion of a shaft or axle to an analog or digital code. There are two types: optical and magnetic. The optical type operates using the same core components as the IR range sensors (an IR LED and a photodiode). The photodiode is a 2-terminal sensor that converts IR light to current. The photodiode is frequently combined with a Bipolar Junction Transistor (BJT) to convert the current to a voltage and amplify the signal. With an IR range finder, the IR light reflects off a surface and then returns to the phototransistor to cause a voltage change, but with the optical encoder the IR light is aimed directly at a phototransistor. When the beam is blocked the output voltage of the phototransistor changes. A gear, or an encoder wheel, is mounted on a motor shaft so that the beam is blocked and passed through as the shaft spins. In this way, the phototransistor in the encoder produces a square wave voltage. The square wave can be used to measure the angular velocity in radians per second (rad/sec) or revolutions per minute (rpm) or the number of pulses in the square wave can be counted to determine the amount of angular displacement. Figure 4.24 shows a Lego NXT motor that has been taken apart to show the internal encoder. The black gear is used for the encoder. It has 36 openings (or slits), so the resolution is 360°/36 = 10°/step.

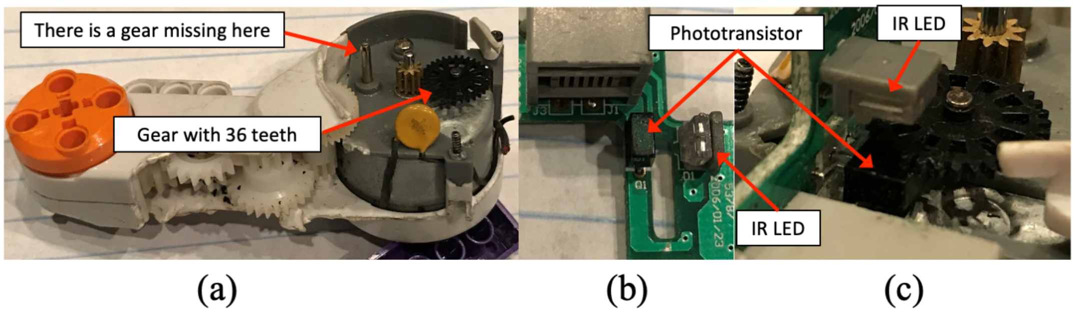


Figure 4.24. ***Lego NXT Motor:*** *(a) Internal view without the drive gear; (b) Encoder PCB that contains the circuitry that controls the IR LED and photodiode; (c) Encoder sending IR light through a gear with 36 slits cut out. Photos (a), (b), and (c) by Chad Davis*

If a magnetic encoder is used, a Hall Effect sensor is usually employed as a “magnetic pickup” to detect metal gear teeth or one or more magnets that rotate with the shaft. Fundamentally, the hall effect sensor measures the strength of a local magnetic field. Their primary use is in monitoring angular velocity or displacement, but they are also used in small current sensing applications where a changing current through a wire generates a small, local magnetic field. These sensors are designed to have a voltage, or current, output that is proportional to the magnetic field. In modern cars, the speed of the car is determined using hall effect sensors. Figure 4.25 shows a wheel (a car flywheel, for example) with four magnetic elements on it. The hall effect sensor is placed in close proximity to the rotating wheel. Each time the magnet passes by a voltage pulse is generated by the sensor. If we start at the sensor on the eastern face of the wheel as shown in the figure below (so there is a pulse right at the beginning of revolution), one full revolution will occur as the fifth pulse is counted. However, there are four magnets, so dividing by four determines the number of revolutions, while the remainder provides information about any partial revolution. Counting the revolutions over a given time period can be converted to revolutions-per-minute, or whatever unit base we desire.

A “latching” digital Hall Effect sensor is shown below. This device produces a square wave output voltage that changes between 0 and V_{CC} each time it detects the polarity change of a magnet. This is shown in Figure 4.26.

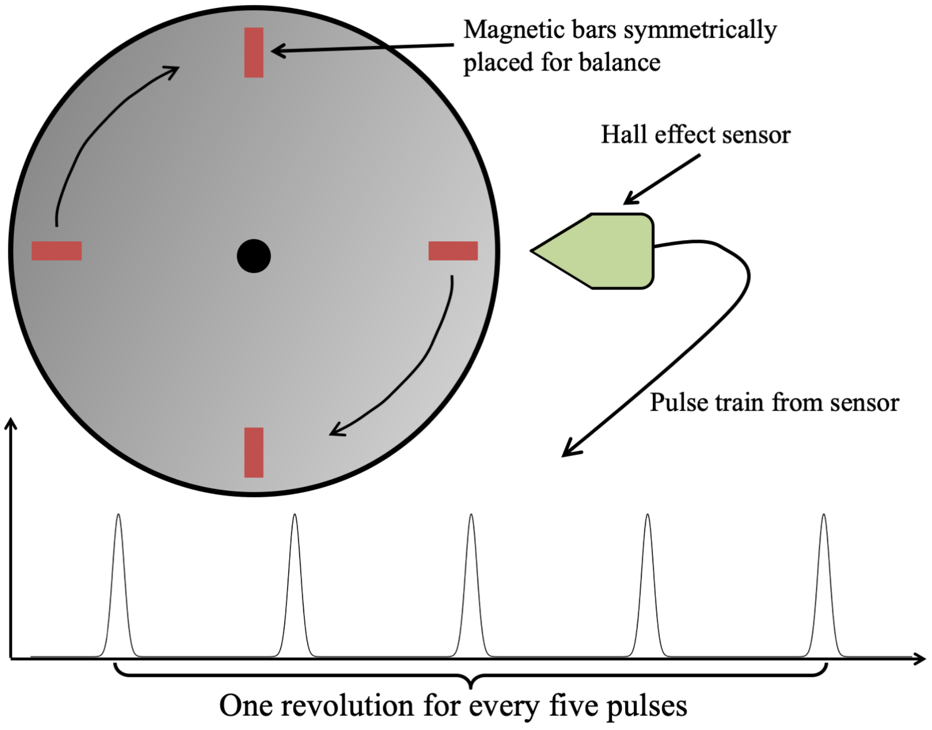

Magnetic bars symmetrically placed for balance
Hall effect sensor
Pulse train from sensor
One revolution for every five pulses

Figure 4.25. *Hall Effect Sensor Measuring Rotational Speed:* *As the wheel rotates, each magnetic element generates a voltage pulse from the hall effect sensor. One rotation requires us to get back to the original magnetic element, so it takes five pulses to constitute one revolution. Computers are very adept at counting pulses and marking every n^th^ pulse.*

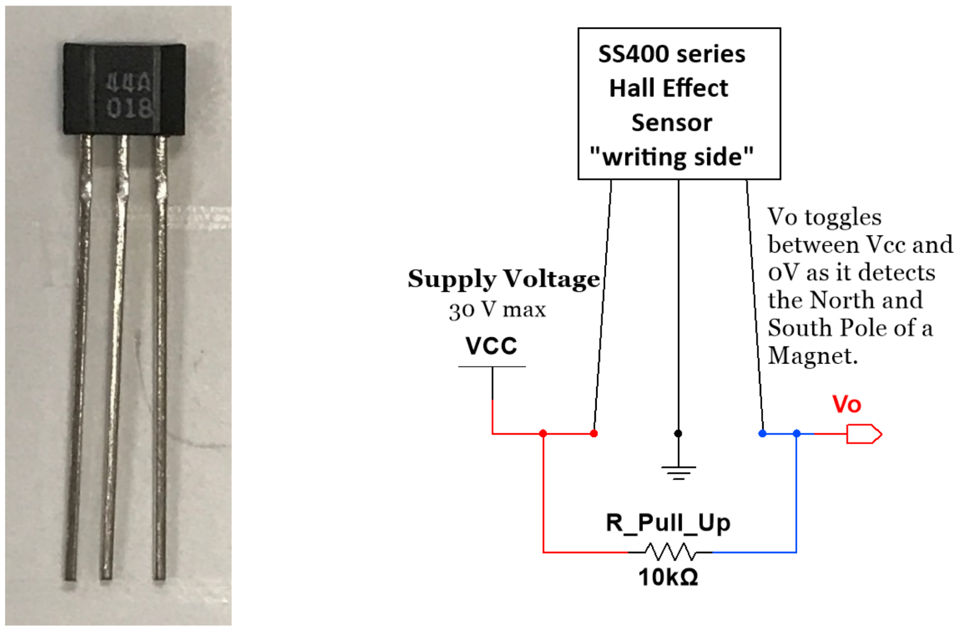


Figure 4.26. *SS4400 Series Latching Hall Effect Sensor: On the left side is an image of the actual sensor. On the right of the figure is a diagram that shows how to use the sensor in a circuit. Note that a pull-up resistor must be added for it to work. Sensor photo by Chad Davis.*

Since this is a digital sensor, the output can be sent directly into a digital input line of a DAQ and the ADC process can be skipped, as long as V_{CC} is set to the proper voltage level. The digital V_{CC} voltage level is 5V (also called TTL voltage level) for many DAQs. This sensor acts as follows:

- When the south pole of a magnet is in range of the sensor, its output is 0V
- When the north pole of the magnet is in range of the sensor, its output is 5V.

In this way, as a magnet spins a square wave is produced that can be used to determine angular position or velocity just like was previously described with the optical encoder. The hall effect sensor used in the figure above requires a pull up resistor so that the output is never floating and is forced to V_{CC} when nothing is connected to the output pin. This is shown schematically to the right of the sensor in Figure 4.26.

4.2.8 Thermocouple

The thermocouple is a commonly used temperature sensor that consists of two separate metals. The metals undergo a *thermo-electric effect*, which was a phenomenon that was first discovered by the German physicist Thomas Seebeck. Because each metal has a different thermo-electric behavior, a voltage difference is formed between the two metals and the voltage difference is proportional to the temperature. In practice, one needs a calibration measurement of a *cold junction*, or some method of finding the zero point. Further, the usable range of a thermocouple is frequently estimated to be linear, so that the equation that relates voltage and temperature is given by $V_x = k_T\left(T_x - T_{ref}\right)$, where the term k_T is the thermocouple *sensitivity*, and the T_{ref} term is provided by the manufacturer, or can be determined by a calibration process. The thermocouple sensitivity is generally in the μV/°C range. The sensitivity constant is often determined by placing the reference node in an ice bath so that the T_{ref} value is 0°C.

Thermocouples can be placed in relatively harsh environments, such as extremely high or low temperatures. However, their sensitivity is not perfectly linear, nor are they accurate to sub-degree precision. If more accuracy is needed than using the linear sensitivity equation, then a thermocouple reference chart can be used. The following example shows how to convert the voltage measurement to temperature using both the linear sensitivity equation and the thermocouple reference chart. This link contains more information about thermocouples and has a link to temperature charts for different types of thermocouples: https://www.thermocoupleinfo.com).

Example 4.8 *A K-type thermocouple has a reading of 5.183 mV. If the reference temperature is at 0 °C, what is the temperature, T_x, that corresponds to the thermocouple voltage reading?*

Solution:

$$V_x = k_T \cdot (T_x - T_{ref}) \Rightarrow 5.183\text{mV} - 0\text{mV} \quad = \quad 41\mu\text{V}/°C \cdot (T_x - 0)$$

Converting mV to μV and solving for T_x, we have

$$\begin{aligned} 5.183\text{mV} \cdot 1000 &= 5183\mu\text{V} \\ &\Downarrow \\ T_x &= \frac{5183\mu\text{V}}{41\mu\text{V}/°\text{C}} \\ &= \mathbf{126.4146°C} \end{aligned}$$

If the K-type thermocouple reference chart (http://www.thermocoupleinfo.com/pdf/type-k-thermocouple-reference-table.pdf) is used, it shows the following values:

- $V_{Low} = 5.165$ mV @ 126 °C
- $V_{High} = 5.206$ mV @ 127 °C

Using linear interpolation to solve for T_x between T_{Low} and T_{High}:

$$T_x = T_{Low} + \left(T_{High} - T_{Low}\right) \frac{V_x - V_{Low}}{V_{High} - V_{Low}}$$

$$= 126 + (127 - 126) \frac{5.183 - 5.165}{5.206 - 5.165}$$

$$T_x = \mathbf{126.439°C}$$

Assuming the temperature reference chart contains the "true values," the calculation in this example using the linear sensitivity equation has the following error:

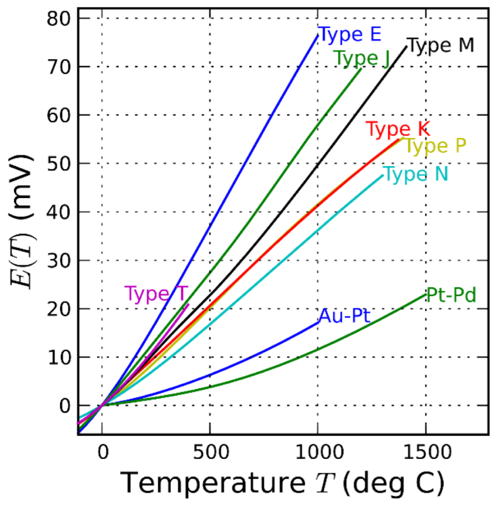
$$\begin{aligned}\%\text{Error} &= 100\,\frac{\text{Measured} - \text{True}}{\text{True}} \\ &= 100\,\frac{126.4146 - 126.439}{126.439} \\ &= \mathbf{-0.0193}\%\end{aligned}$$

An important aspect of thermocouples is that external signal conditioning circuitry is often needed to amplify the signal and to provide noise reduction because the sensitivity constant is typically in the μV/°C range. Thermocouples are categorized by the type of metals used and the temperature range associated with the metals. Table 4.1 shows the most common types, along with their range and accuracy (compiled from information at https://www.thermocoupleinfo.com).

Table 4.1. *Thermocouple Types with temperature ranges, accuracy, and material composition*

Thermocouple Type	Temperature Range (°C)	Accuracy (°C)	Materials
J	-210 to 760 (thermocouple grade wire)	± 2.2 or ± 0.75%	Iron/Constantan
K	-270 to 1260 (thermocouple grade wire)	± 2.2 or ± 0.75%	Nickel-Chromium/Nickel-Alumel
T	-270 to 370 (thermocouple grade wire)	± 1.0 or ± 0.75%	Copper/Constantan
N	-270 to 1260 (thermocouple grade wire)	± 2.2 or ± 0.75%	Nicrosil/Nisil
E	-270 to 870 (thermocouple grade wire)	± 1.7 or ± 0.5%	Nickel-Chromium/Constantan
B	0 to 1700 (thermocouple grade wire)	± 0.5%	Platinum-Rhodium(30)/Platinum-Rhodium(6)
R	-50 to 1480 (thermocouple grade wire)	± 1.5 or ± 0.25%	Platinum-Rhodium(13)/Platinum
S	-50 to 1480 (thermocouple grade wire)	± 1.5 or ± 0.25%	Platinum-Rhodium(10)/Platinum

The reader may note there are temperature overlaps between various thermocouple types, but the materials are different and have better, or worse, properties in certain environments (oxidizing versus inert, extreme temperatures, etc.). Figure 4.27 shows representative sensitivity curves for common thermocouples. It should be observed that some of the thermocouples are not linear over their full (physically) possible temperature range. This must either be accounted for, or the device should only be used in an application that guarantees it remains in the linear performance region.



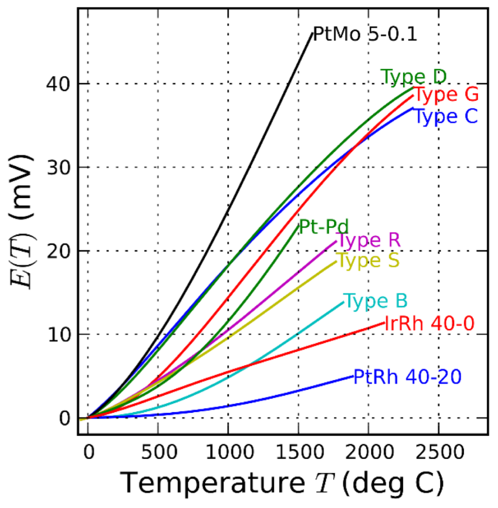


Figure 4.27. ***Graph of Thermocouple Sensitivity:*** *Graphs showing the sensitivity in mV/°C for different types of thermocouples. Some sensitivities are very linear above 0 °C (e.g. Type E) while others are less linear (e.g. Types S and B). (Left) ©Nanite and used under CCO 1.0 license.* ***https://en.wikipedia.org/wiki/Thermocouple#/media/File:Intermediate_temperature_thermocouples_reference_functions.svg*** *(Right) ©Nanite and used under CCO 1.0 license.* ***https://en.wikipedia.org/wiki/Thermocouple#/media/File:High_temperature_thermocouples_reference_functions.svg***

The temperature sensitivity of thermocouples is in the μV/°C range, so external signal conditioning circuitry is often used in conjunction with the thermocouples. Frequently the signal conditioning consists of a lowpass filter (for both noise reduction and anti-aliasing, see Chapter 3) and amplification. In a computer-based measurement system using a thermocouple the user must ensure that the calibration accounts for the precise level of amplification so that a proper overall sensitivity value can be set in the program. Ideally the user program would provide the opportunity for the user to set this value prior to data collection. In some instances the exact value is less critical than simply knowing the general temperature range a system is operating in. This can occur when the thermocouple is being used for thermal protection, so that when the temperature of a system exceeds some (conservatively chosen) temperature limit, the system is de-energized, or idled back to prevent damage related to excessive temperatures.

4.2.9 Thermistor

The thermistor is another type of temperature-measuring device. These devices are based on a *thermally sensitive resistor*, thus the name thermistor. As the full name implies, the device changes resistance with temperature. As students who have had a basic circuits course know, most metals have some thermal coefficient that causes their resistance to change with temperature. However, the thermistor has a much more pronounced relation between temperature and resistance. Thermistors come in two basic types: negative temperature coefficient (NTC) and positive temperature coefficient (PTC). The NTC thermistor *decreases* resistance with *increasing* temperature, or it can be said that temperature and resistance are inversely proportional. The PTC thermistor is directly proportional, meaning an increase in temperature causes an increase in resistance. Thermistors are considered to be extremely accurate, particularly compared to thermocouples, but they operate in lower temperature ranges and are not usually appropriate above about 300°C.

The basic thermistor equation relies on a factor called the B-value. This value relates two temperature points and the two related resistance values as follows:
(https://www.electronics-tutorials.ws/io/thermistors.html)

$$B_{(T1/T2)} = \frac{T_2 \times T_1}{T_2 - T_1} \times \ln\left(\frac{R_1}{R_2}\right)$$

where temperatures are in degrees Kelvin, and resistances are in Ohms. Typical specifications provide T_1 and T_2 in Celsius, so the user must be prepared to convert to Kelvin prior to using this equation (recall that Kelvin and Celsius are simply separated by a constant 273.15, so that 0°C = 273.15°K). It is also important to note that the B-value is associated with a temperature range, so that one might see a specification of $B_{10/100}$ = 2552, meaning that the stated B-value holds for temperatures between 10°C and 100°C. Further, the nominal resistance given for an NTC thermistor is related to the lower temperature, so that a 5 kΩ NTC thermistor with a $B_{10/100}$ = 2552 has a resistance of 5000 Ohms at 10°C. The characteristic temperature/resistance curve for this example thermistor is shown in Figure 4.28. The shaded region is the region for which the temperature measurement is valid based on the B-value.

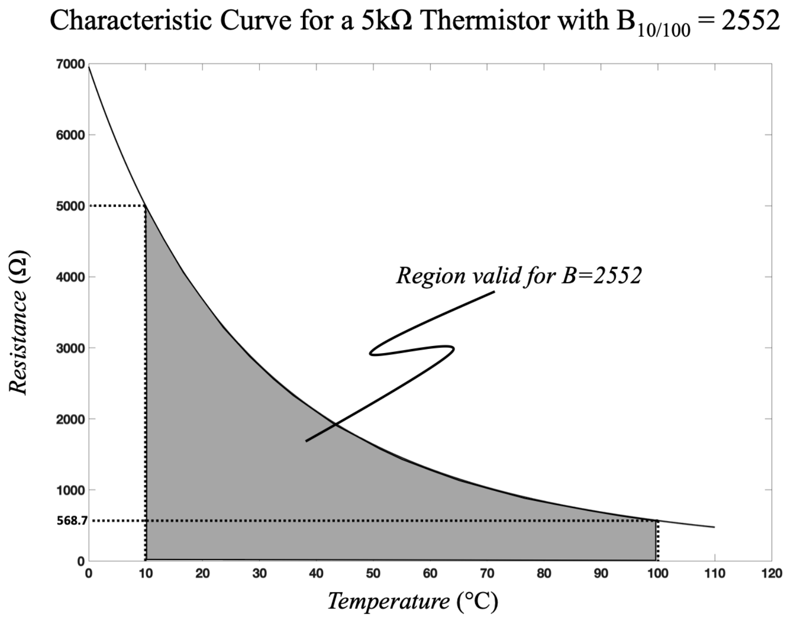


Figure 4.28. ***Characteristic Curve for a Thermistor:*** *The resistance as a function of temperature for a 5kΩ thermistor with $B_{10/100}$ =2552. The shaded gray region indicates the range for which the curve is valid based on the temperature limits provided by the B-vlaue.*

The resistance value associated with the upper temperature limit is determined algebraically by rearranging the B-value equation so that

$$B = \frac{T_2 \times T_1}{T_2 - T_1} \times \ln \frac{R_1}{R_2}$$

$$\Downarrow$$

$$e^B = e^{\left(\frac{T_2 \times T_1}{T_2 - T_1}\right)} \times \frac{R_1}{R_2}$$

$$e^B - e^{\left(\frac{T_2 \times T_1}{T_2 - T_1}\right)} = \frac{R_1}{R_2}$$

$$R_2 = \frac{R_1}{e^{\left(\frac{B(T_2 - T_1)}{T_2 \cdot T_1}\right)}}$$

$$\Downarrow$$

$$R_2 = \frac{5000}{e^{\left(\frac{2552(373.15 - 283.15)}{373.15 \cdot 283.15}\right)}}$$

$$R_2 = 568.7\Omega$$

In temperature measurement applications the thermistor is generally placed in a resistive bridge circuit as discussed in Section 4.2.1 and demonstrated in the circuit in Figure 4.7. In that section the variable resistive element was a strain gauge. The same method works with thermistors so that one can solve for resistance and, therefore, temperature accurately. Indeed, typical thermistor accuracy is on the order of 0.1°C. This is also a preferred method because much of the analog-to-digital conversion technology is geared toward voltage measurements. Figure 4.29 shows a typical circuit for PC-based temperature measurement using a thermistor. The circuit provides for noise reduction, anti-aliasing, and possibly gain via the lowpass filter, analog-to-digital conversion, and computer storage.

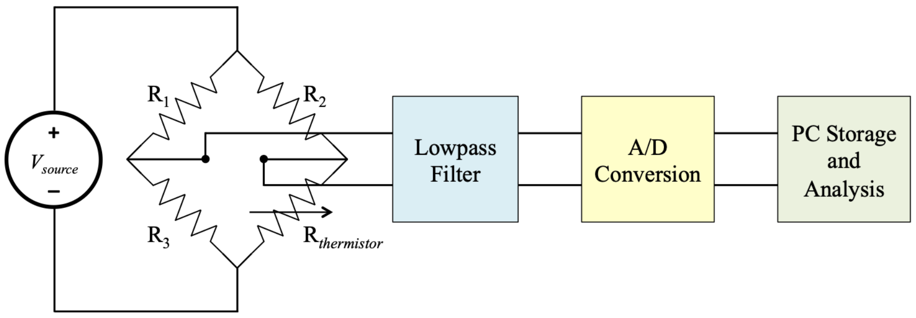


Figure 4.29. ***Thermistor Temperature Measurement in a Bridge:*** *The resistive bridge arrangement is very sensitive to small resistance changes and provides a superior method for accurate temperature measurement using a thermistor.*

A simpler method that may (or may not) eliminate some of the front-end circuitry (lowpass filter, gain) is shown in Figure 4.30. In this method a simple voltage divider is used with the thermistor comprising one of the resistors. As shown in the figure, using the thermistor as the lower of the two resistors makes it the primary element for determining the voltage, V_T (i.e. the voltage due to the resistance of the thermistor). In this configuration a negative temperature coefficient thermistor will cause $R_{thermistor}$ to reduce as temperature increases. This also causes V_T to go *down* as temperature increases. If one desires a direct relationship (as opposed to an inverse relationship), either a positive temperature coefficient (PTC) thermistor can be used, or the position of R_1 and $R_{thermistor}$ can be swapped in the circuit. In the latter case, as the temperature increases and the resistance in $R_{thermistor}$ decreases, the voltage drop across $R_{thermistor}$ decreases, meaning the voltage drop across R_1 increases. Thus, the output voltage taken across R_1 is directly proportional to the temperature.

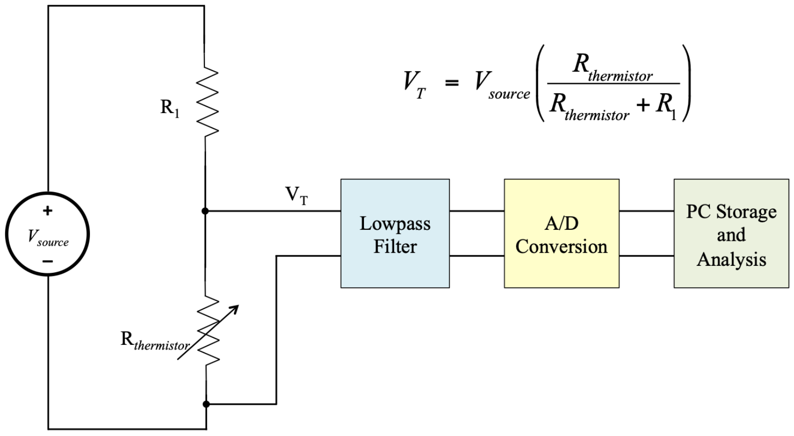


Figure 4.30. ***Thermistor Temperature Measurement in a Voltage Divider:*** *The voltage divider circuit is simpler to implement, but less sensitive than the bridge circuit. In the circuit as shown the output voltage decreases as temperature increases (inverse relationship). If R_1 and $R_{thermistor}$ are swapped, the output voltage rises as temperature rises, making a direct relationship between the two.*

Figure 4.31 shows a photo of a thermistor in a voltage divider configuration with R_1 being a fixed 10 kΩ resistor and the thermistor is a 10 kΩ NTC thermistor (the same type shown in the Wheatstone bridge example in Figure 4.7). At room temperature the thermistor resistance should be ∼ 10 kΩ and the voltage across the thermistor should be about one-half the source voltage (1/2 V_{CC}).

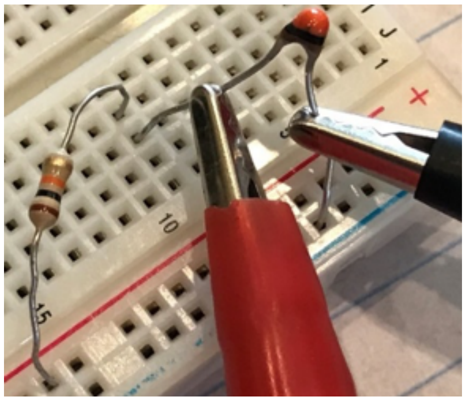
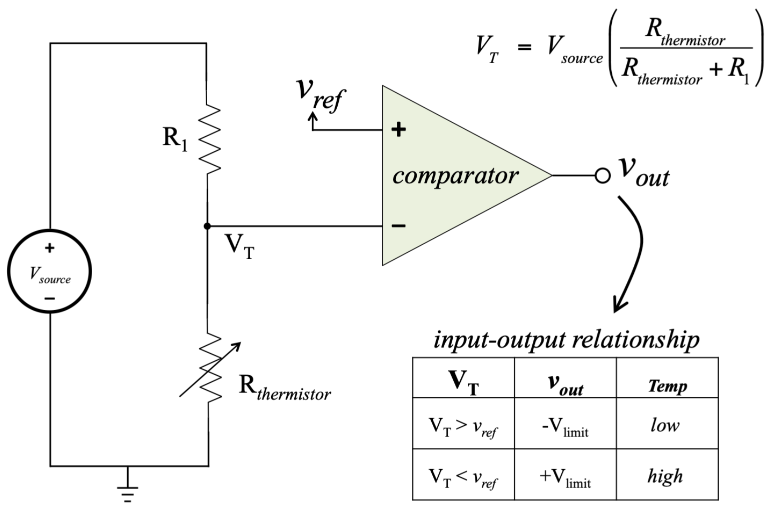


Figure 4.31. ***Photo of a Thermistor Voltage Divider:*** *The thermistor is in series with a resistor, forming a voltage divider. Voltage is measured across the thermistor (see the red and black lead clips in the photo) to estimate temperature. Photo by Chad Davis.*

Thermistors are also used in temperature protection so that the goal is to simply de-energize a circuit, or stop a process, if a temperature limit is exceeded. This can be accomplished without the data acquisition component of the previous systems as shown in Figure 4.32. In this circuit, a simple comparator circuit (essentially an op-amp with no inverting feedback) monitors the voltage divider output. If an NTC thermistor is used and located in the voltage divider as shown, then the V_T voltage can be applied to the negative (inverting) input terminal of the comparator, while a reference voltage, v_{ref} is applied to the positive (non-inverting) terminal. The operation of the comparator is such that when $v^+ > v^-$, the output goes to the maximum positive voltage output (usually just a bit less than the supply voltage to the comparator). When the opposite condition holds, namely $v^+ < v^-$, the output goes to the maximum negative voltage (this is generally the negative power supply which may be symmetric to the positive supply, or it may be ground in a single-sided power supply).


$$V_T = V_{source}\left(\frac{R_{thermistor}}{R_{thermistor}+R_1}\right)$$
v_{ref}
+
comparator
−
v_{out}
R_1
V_T
+
V_{source}
−
$R_{thermistor}$
input-output relationship

V_T	v_{out}	Temp
$V_T > v_{ref}$	$-V_{limit}$	low
$V_T < v_{ref}$	$+V_{limit}$	high

Figure 4.32. ***HI-LO Control with a Thermistor:*** *Using a simple voltage divider and a comparator, one may achieve an ON-OFF, or HI-LO control circuit that causes a binary action based on temperature. The thermistor voltage, V_T, is inversely proportional to the temperature (assuming an NTC thermistor), but when placed on the negative input of the comparator it provides a direct relationship. If the temperature rises above a certain value–associated with a specific reduced resistance $R_{thermistor}$–then $V_T < v_{ref}$ and the output goes HI. Otherwise, the output remains LO. The HI output can be used to control external cooling devices, or simply provide the signal to stop the circuit or process.*

While comparators are frequently used with Thermistors, a Metal Oxide Semiconductor Field Effect Transistor (MOSFET) switching circuit is also a useful way to implement them to turn something on or off. The following example demonstrates how to use a thermistor with a MOSFET.

Example 4.9 *Using a voltage divider configuration, design a voltage divider circuit that causes a BS170 MOSFET to turn on when an NTC 10 kΩ thermistor is heated. Assume the threshold voltage for the BS170 is ∼ 2V. Note: The threshold voltage is the gate voltage required to turn on the device and allow current to flow from the drain to the source. The gate, drain, and source are the three pins of the MOSFET.*

Solutions:

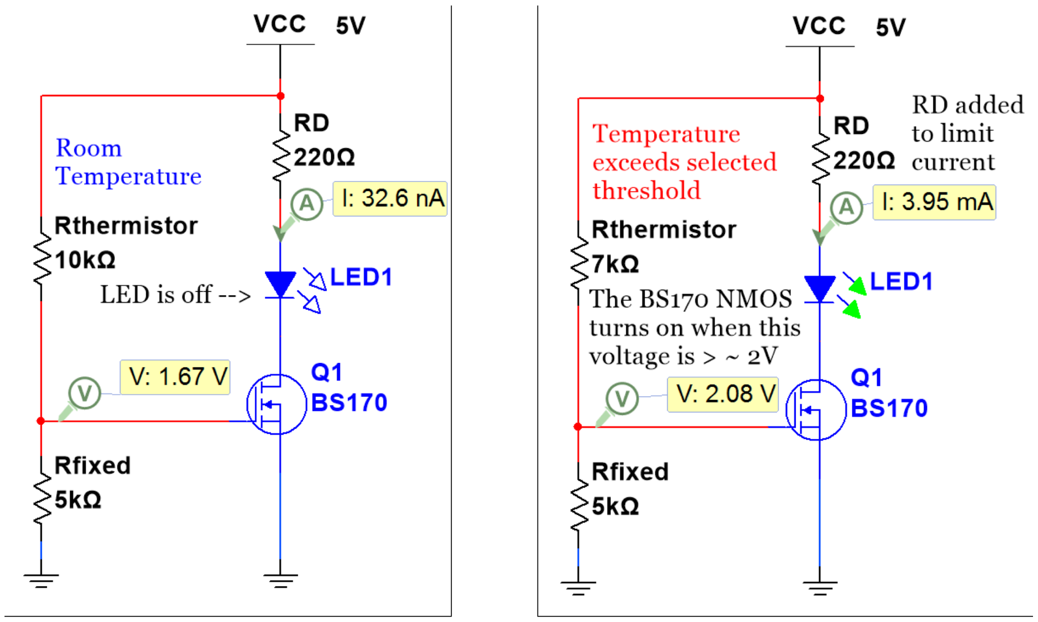


Figure 4.33. ***Example 4.9 Schematic:*** *Schematic showing how the thermistor is used to turn on an LED when the temperature increases. (a) Gate voltage is 1.67V and the MOSFET is off so no current flows through the LED and it is Off, (b) Gate voltage is 2.06V when the thermistor is heated up to the unknown level in this example and the MOSFET turns on so current flows through the LED and turns it On. The MOSFET drain current is shown to be 3.95 mA, which is less than the 20 mA rating of typical LEDs.*

4.3 Conclusion

The interface between the physical phenomenon being measured and the computer-based measuring system is a critical component of the overall measurement process. Although the preceding sections do not exhaustively cover all the possible sensor types available, they provide an understanding of the fundamentals of signal conditioning and data acquisition using sensors as an interface between the physical world and the computer. Sensors exist to transduce nearly any physical phenomenon into an electrical signal. The key features for a user are the sensitivity (how many volts or amps are generated per unit of physical dimension) and bandwidth (how quickly the sensor responds to external changes). Of course, consideration for the environment in which the sensor will be placed (temperature, pressure, vibration) must be addressed for the best sensor selection. However, given the ubiquitous nature of computer-based acquisition systems, the reader should thoroughly appreciate the importance of good sensor selection in achieving the best possible measurement results.

Chapter 5

5.1 Statistical Distributions

Before explaining statistical distributions, the difference between the *continuous* time, $x(t)$, and *discrete* time, $x[n]$, domains needs to be discussed. Consider the example of a car battery that has three primary operation phases: 1) Prior to starting the car, 2) While the car is being started, and 3) After the car is started. The following shows a 12-Volt car battery that is being measured with a Multimeter prior to it being started. It reads 12.546 Volts and remains fairly constant, so we could model it as the constant continuous time function $x_1(t) = 12.546$. A six-cell car battery will typically read approximately 12.6V when fully charged (2.1V per cell). If it reads 12V or lower it is almost completely discharged, which is an indication that either the battery is bad, or something consuming electricity was left on in the car, such as an overhead light.

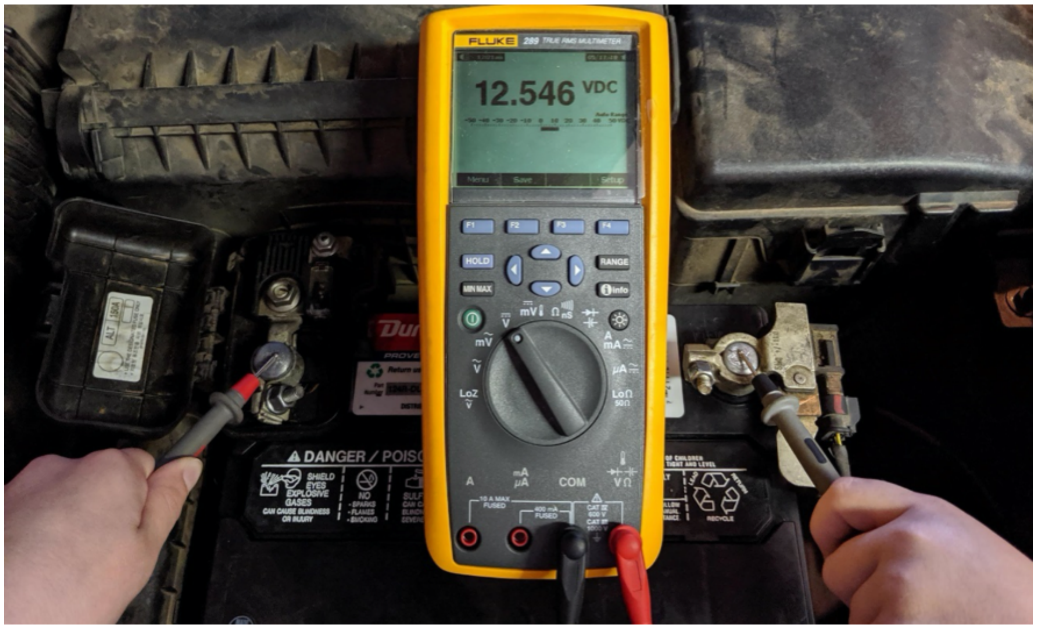


Figure 5.1. ***Car battery being measured with a multimeter:*** *The red lead of the multimeter is placed on the positive terminal and the black lead is placed the negative terminal to read the battery voltage. Photo of battery measurement by Chad Davis.*

When the key is turned and the car is started, the voltage quickly rises to a level of approximately 14.4 Volts, so we could model the final phase as $x_3(t) = 14.4$. The voltage of a car battery will be higher when the car is running because the alternator is continuously charging it. A rise of approximately 2V is to be expected with a good battery and an alternator that is functioning properly. If you try to start the car and the battery drops to a lower voltage, that is a sign that the battery is bad because it is unable to operate correctly when a load is applied. A more sophisticated load test is done at local automotive parts stores that provide a detailed quality report of the battery. In this example, we observed that the transition from phase 1 to 3 took approximately 3 seconds and if we assumed it was a linear transition from 12.546 V to 14.4 V, then we could model phase 2 as $x_2(t) = [(14.4 - 12.546)/3] \cdot t + 12.546$. If we set each of these phases at three seconds in length for simplicity, we could model the battery voltage in the continuous time domain as:

$$x(t) = \begin{cases} 12.546 & \text{(Phase 1)} \quad 0 < t < 3 \\ 0.618 \cdot (t - 3) + 12.546 & \text{(Phase 2)} \quad 3 \leq t \leq 6 \\ 14.4 & \text{(Phase 3)} \quad 6 < t \leq 9 \end{cases}$$

If we wanted to more accurately measure the car battery voltage during these 9 seconds, we could use a data acquisition system. However, when acquiring the data points (i.e. samples) from the DAQ we would be moving from the continuous time domain, $x(t)$, to the discrete time domain, $x[n]$. If we use a sample rate of 10 Hz (10 samples per second), then 10 samples are acquired every second and the discrete time mathematical model would be

$$x[n] = \begin{cases} 12.546 & \text{(Phase 1)} \quad 0 < n < 30 \\ 0.618 \cdot (n - 30) + 12.546 & \text{(Phase 2)} \quad 30 \leq n \leq 60 \\ 14.4 & \text{(Phase 3)} \quad 60 < n \leq 90 \end{cases}$$

When collecting the data, it is evident that the actual data do not fit the mathematical model perfectly. The following plot shows the mathematical model and the measured data overlaid on the same plot.

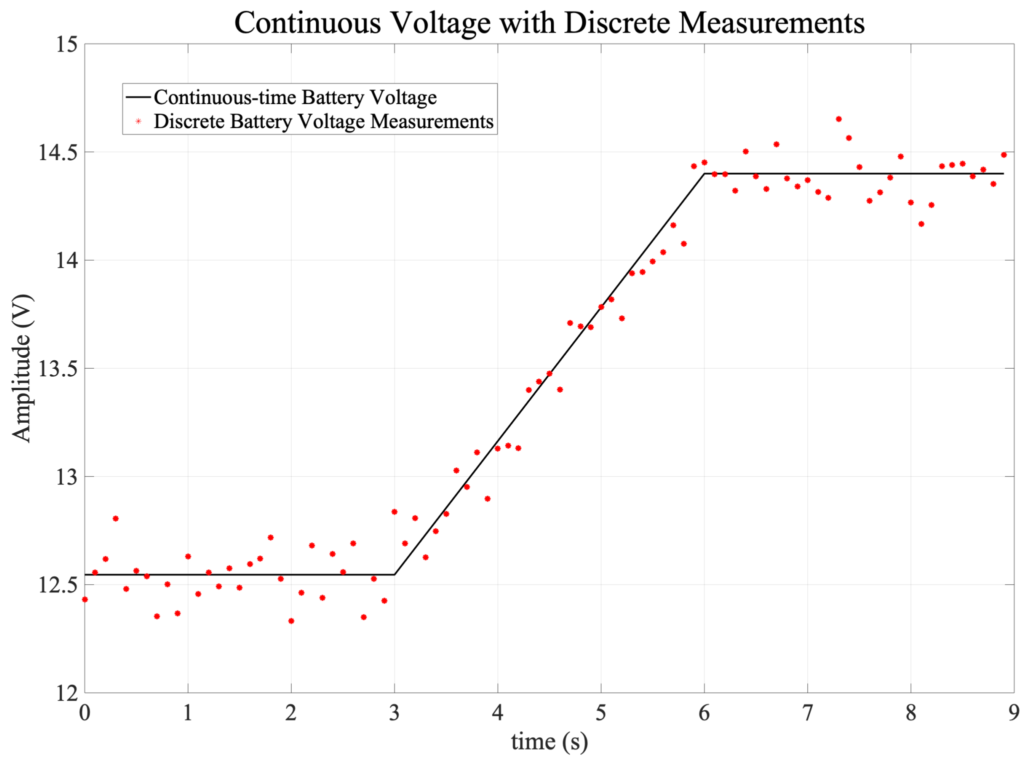


Figure 5.2. ***Car battery voltage model and discrete measurements:*** *The continuous time math model cannot be realized in a real computer system, which is not infinite. If we take discrete measurements, each measurement has some error (noise) associated with it. If we subtract the modeled value, the residual "noise" will have some statistical distribution.*

Given that each measurement made by the measurement system in Figure 5.2 has some error that is a combination of both random (noise) and deterministic (systemic) elements, the overall collection of measurements is considered a random process. If we look at only a short section of the phase 1 data very few of the samples–if any–will be precisely the value of the model (12.546 Volts). The measurement uncertainty makes the measurements look random. This is shown in Figure 5.3. In the lower panel, the voltage scale makes the value look like it is very close to a constant value of approximately 12.5 Volts. In the upper panel, where it is zoomed in around the 12.5-volt range, we see that each sample is slightly different. To help make sense of the data, we can compute several different parameters associated with the distribution of the measurements. Also, the reader should take notice that the battery voltage data are plotted as individual points, x[n], and not as a continuous signal, x(t). We must recognize that, in fact, all computer-based measurement systems collect discrete data points. This leads to a need for certain definitions regarding statistics over some set of discrete data points.

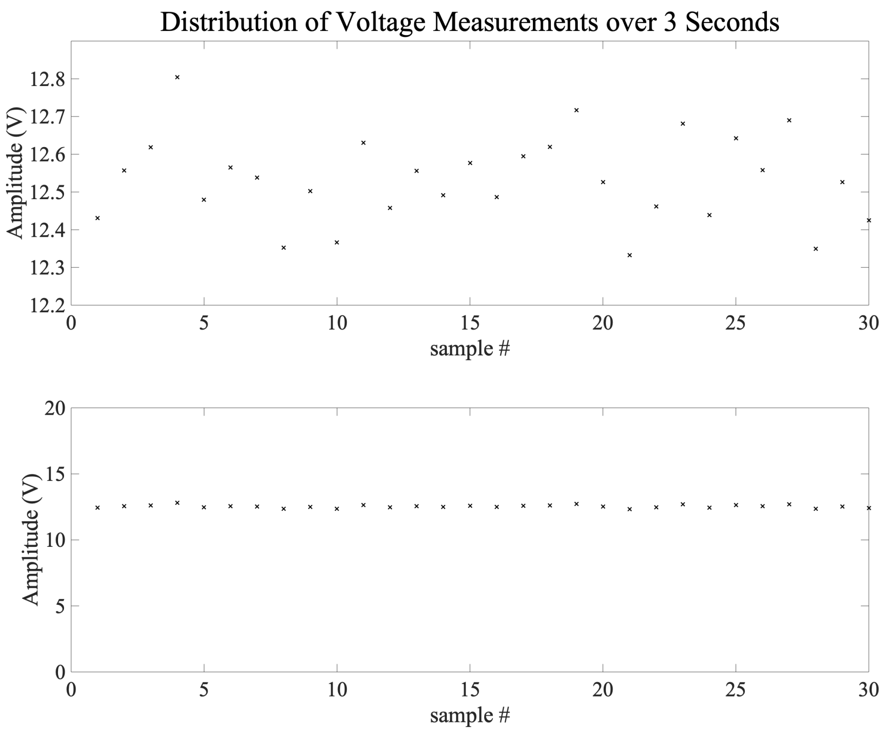


Figure 5.3. ***Samples of Battery Measurement Over Three Seconds:*** *Car battery voltage was measured at a 10 Hz sample rate with a DAQ. The upper panel shows each of the 300 values measured on a scale that allows the user to see the difference between each measurement of the 12-volt battery. The lower panel is scaled out so that the value appears to be consistently at approximately 12.5 Volts. Note that the measurement points are discrete even though the battery itself is producing a voltage at every time, $t \in \mathbb{R}$, during the measurement process.*

5.1.1 Sample

In the battery voltage measurement example we know that the battery will hold a voltage at every moment in time during our sampling over one hour. The voltage continues to exist for the other 59 seconds of every minute when we are not making a measurement. Because we only store one measurement each minute for one hour, the collection of those measurements represents a *sample* of battery voltages. In computer-based measurement systems we can compute statistical parameters over the sample, but this may–or may not–be directly related to the statistics of the entire *population*.

5.1.2 Population

The *population* is the entire set of possible measurements. In a continuous system, the population is infinite because time, t, is an element of the real set, $\mathbb{R}$. Clearly, we cannot store a measurement value for every possible time, t, because that would require infinite storage and infinite time to collect the data. So, by definition, any computer-based measurement system is a discrete system. However, the Nyquist Sampling Criterion (see section 2.2.2) stated that as long as we sample at a rate that is at least twice the maximum frequency of the continuous signal then the discrete data allows us to infer properties of the continuous signal. Indeed, in many applications we use a sample set to infer statistical properties of a population; not only in engineering applications but also in biological, epidemiological, and many other applications.

5.1.3 Sample Space

The *sample space* is the set of all possible values in the outcome of a measurement. Let's use the Fluke 289 Multimeter shown in Figure 5.1 to help explain this concept. Figure 5.1 shows that the device automatically sets the voltage range magnitude to 50V when making the 12.546V battery measurement and at the 50V range it has a resolution of ±0.001 Volts. If a voltage that has a magnitude less than 5V is measured with this device, it automatically reduces the range to 5V and has a resolution of ±0.0001 Volts. This means that when measuring a 12 Volt car battery with the red meter lead on the positive battery terminal and the black meter lead on the negative battery terminal to guarantee we get a positive voltage reading, the possible values of the sample space would be: 5.001, 5.002, 5.003...49.998, 49.999, 50.000. However, the accuracy of the meter is 0.025%, so if we measured a battery that we expected to have a voltage of 12.6V we would be limited by the accuracy of the meter to have a resultant measurement range that is between 12.560V and 12.640V ($12.6 \pm .025\% \cdot 12.6$). Therefore, our sample space in the Fluke 289 Multimeter contains many more data value possibilities than we would ever expect to see with the measurement of the 12V car battery. In any measurement system, a sample space contains all possible outcomes of the measurement, but we never expect to see every single one of the sample space values in the actual data sample.

5.1.4 Distribution

The *distribution* of a random variable describes how the random variable is spread out over the sample space. There are many different distributions that occur in engineering and scientific applications. The two most common are the *uniform* distribution and the *normal* distribution (the normal distribution is also commonly referred to as *Gaussian*). The

Poisson distribution is another type that is an inherently discrete distribution that describes the probability of some number of events occurring in a fixed time interval. It is frequently seen in industrial engineering studies of queuing to help companies determine service levels during various times of the business day. Another distribution used in the engineering community is the *Rayleigh* distribution. In RF communication systems, *Rayleigh Fading* describes the probability of the signal strength of a transmitted electromagnetic waveform along a non-uniform path through a medium, such as the troposphere or ionosphere. This occurs in High Frequency radio transmissions over the horizon, for example. There are hundreds of other distributions and the following link shows most of the ones that are commonly used: https://en.wikipedia.org/wiki/List_of_probability_distributions. This text will mainly focus on the normal and uniform distributions but familiarizing yourself with other distributions is beneficial because you will undoubtedly come across some data sets that are better represented by one of the other distributions. For example, if you give an exam in a class during flu season and have many students who missed multiple class periods and their understanding of the material hasn't caught up yet with the top students in the class you might get a bimodal distribution that looks something like the following figure. Bimodal distributions with two distinct peaks are common in academia because there tend to be many situations where a large group of students "get it" and a large group of students do not. The ultimate challenge of teaching in any setting is to get everyone into the "get it" category and avoid distributions like the one shown below in Figure 5.4.

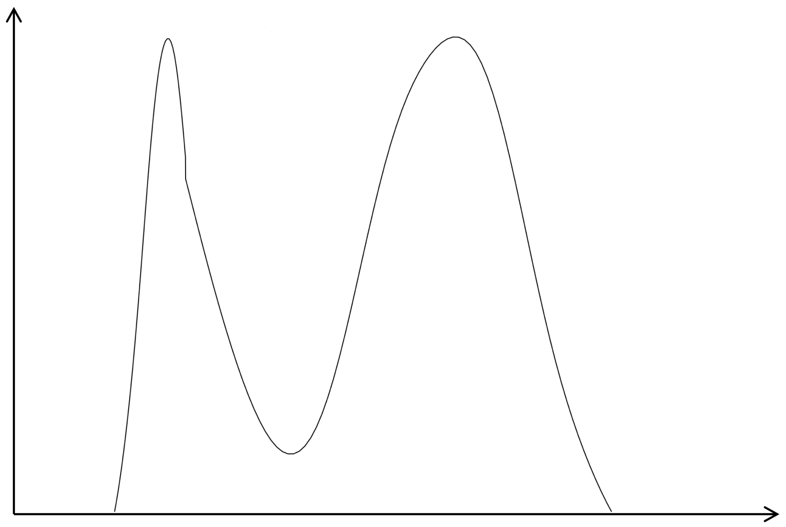


Figure 5.4. ***Example of a Bimodal Distribution:*** *The bimodal distribution looks like two separate normal distributions combined.*

The example in Figure 5.5 below shows how to use LabVIEW to create examples of uniform and normal distributions that might occur on an exam in a class. The upper FOR loop includes the random number VI that has a dice icon. This VI produces a random number between 0 and 1 that has a uniform distribution. The bottom loop includes a VI that produces a random number between 0 and 1 that has a normal distribution. The samples produced by these FOR loops are shown in Figure 5.7 along with some of the statistical parameters that will be discussed in more detail in section 5.2, and histogram plots that are covered in section 5.4.

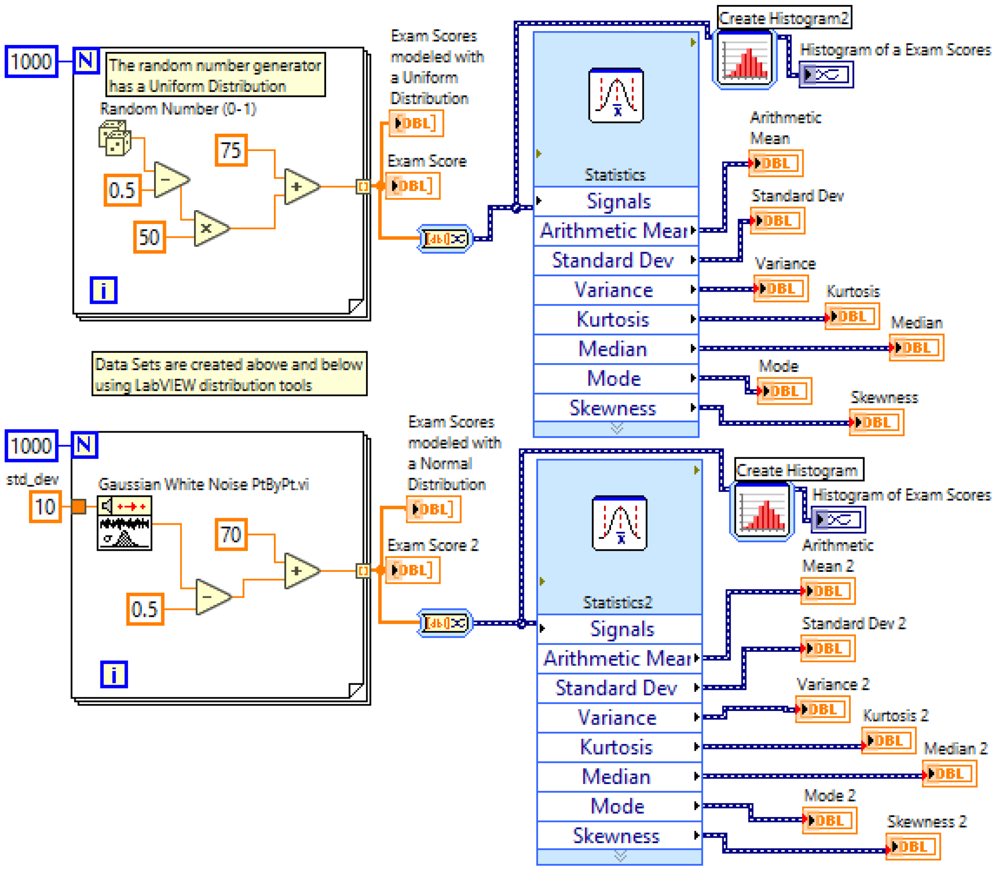


Figure 5.5. ***LabVIEW code to Create Uniform (Top) and Normal (Bottom) Distributions of Exam Scores:*** *LabVIEW can produce different random distributions. The basic random number generator (the dice in the upper loop) always produces a uniform distribution–each value has equal probability. However, a normal distribution can be achieved with the Gaussian White Noise VI. Note that it requires an input for standard deviation, and the mean can be added by the user.*

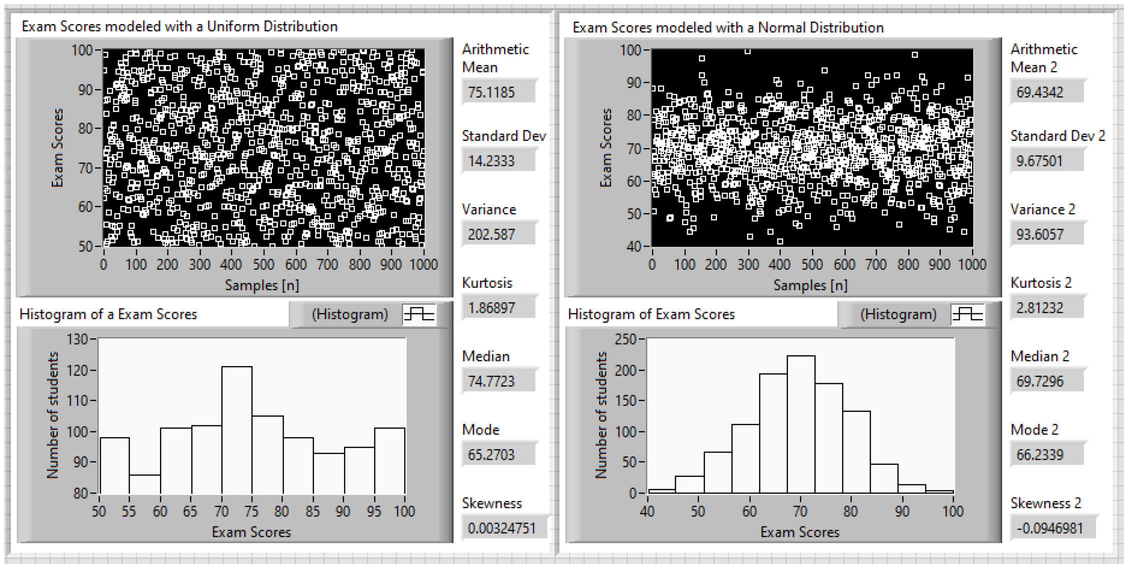


Figure 5.6. ***Results of the LabVIEW code from Figure 5.5:*** *A histogram can show how often each value occurs within the data set. The histogram uses bins (ranges) of values. On the left, the uniform distribution places sample points all across the space in the upper plot, and the histogram shows relatively equal values (the small number of samples keeps it from being more uniform). On the right side, it is clear that the samples fall in a narrower space (defined by the standard deviation), and the histogram outline has the character of a normal (Gaussian) distribution.*

5.1.5 Probability

The term *probability* describes how likely an *event* is to occur. A *probability density function*, or pdf, uses the x-axis (the abscissa) as the coordinate for each possible outcome, and the y-axis (the ordinate) to show the likelihood of each outcome for a given experiment (or measurement). Because the pdf describes the probability for all possible outcomes, the area under the pdf curve (i.e the integral of the pdf) must be equal to 1, since one of the possible outcomes must always occur. The figure below shows an example of a pdf and a corresponding box plot. The middle point of the boxplot and of the pdf is called the median. The interquartile region (IQR) contains 50% of the measurements that are centered about the median. The upper plot in the figure below shows the pdf with the IQR highlighted so that it is aligned with the box plot. The bottom plot shows the pdf with labels of standard deviation multiples ($\pm 1\sigma$, $\pm 2\sigma$, etc.) from the median and the percentage of the measurements that fall into the different regions. Figure 5.7 below shows an example of a pdf and a corresponding box plot.

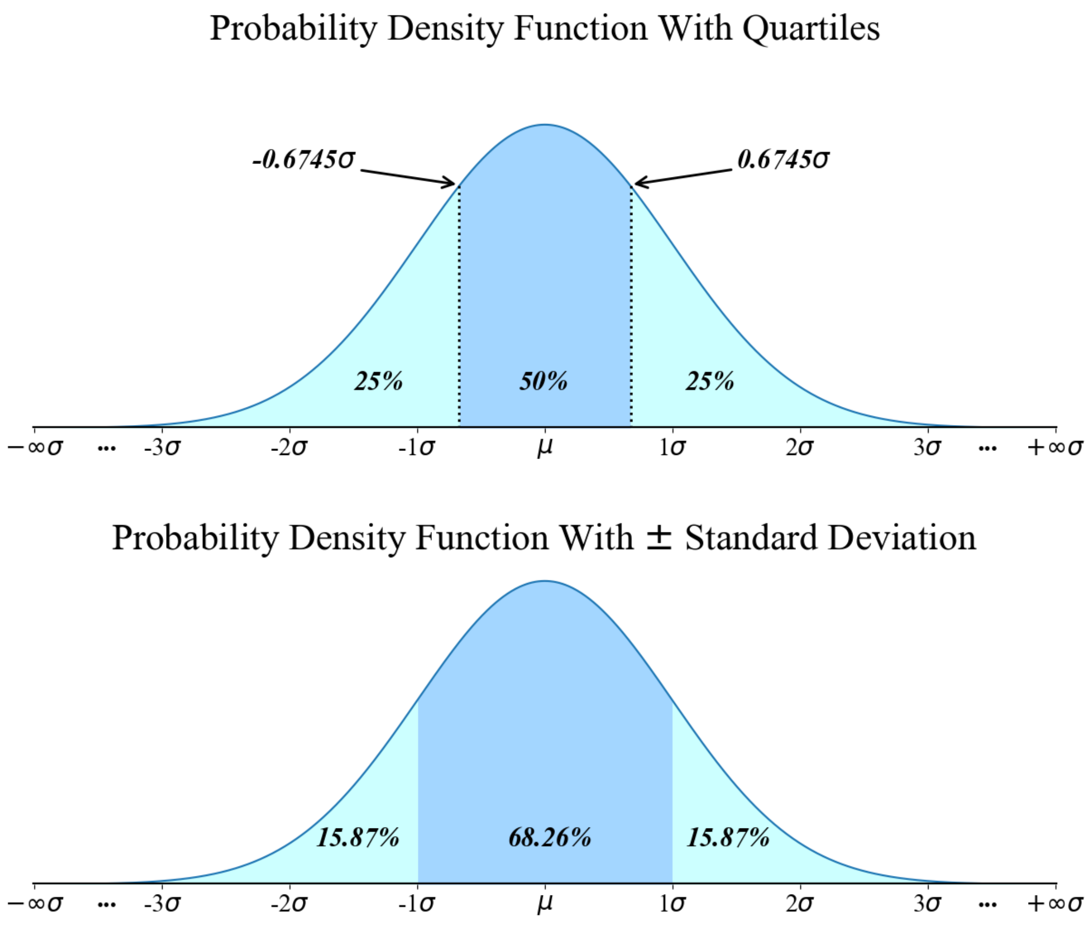


Figure 5.7. ***Example of a Probability Density Function for a Normal Distribution:*** *The probability distribution function for a Normal Distribution is symmetric about the mean. The upper plot shows how the quartile divisions compare to the standard deviation (σ). The middle 50% about the mean occurs from -0.6745σ to $+0.6745\sigma$. In the lower plot the reader can see what percent of the distribution occurs within ± 1 standard deviation.*

5.2 Statistical Measures

Each type of statistical attribute is called a *parameter*. Common parameters of the normal distribution are the *mean*, *standard deviation*, and *variance*, though these are not the only parameters associated with the normal distribution. Although it is beyond the scope of this text any number of parameters can be generated for a continuous distribution using *moment generation*. However, for most applications the mean and variance are the two main parameters, and they constitute the first two *moments* of the normal distribution. Standard deviation is simply the square root of the variance, as we shall see below.

5.2.1 Mean Value

The first measure we will assess is the mean. The mean is a measure of the tendency of repeated measures. In our battery measurement example above the central tendency, or mean, is 12.546 Volts. We refer to this as the *sample mean* because we can only compute the mean for the set of data points in our sample. When we have a discrete set of sample data points the mean is computed by:

$$\overline{x} = \sum_{i=1}^{k} \frac{x_i}{k}$$

The mean of a discrete population with N samples , μ, is also given by the previous equation. However, the lower-case k becomes the full population count, and not the number of measurements in a sample subset, namely:

$$\mu = \sum_{i=1}^{K} \frac{x_i}{K}$$

On the other hand, the true population mean, μ, of a continuous population is more difficult to compute. If we let $f(x_i)$ describe the probability of a continuous distribution on some interval between x and $x + \Delta x$, then the population mean is computed by:

$$\mu = \int_{-\infty}^{+\infty} x f(x) dx$$

We will explain the function $f(x)$ more thoroughly below. There are also two other common measures of central tendency that are particularly useful in discrete sample sets. One of these is the *median value*. This is the value at the center of a sample set. In a discrete data set, such as exam scores in a class of 75 students, the median score may be a better representation of the central tendency of the group, rather than the mean value. This is particularly so if there are some outlier scores compared to the bulk of the grades. One example that comes to mind is an exam that was given several years ago in a Signals & System class that had a mean value of 78.2 for an exam, but the median value was only 71. This occurred because two exceptional students aced the exam, while most others performed below that level. Indeed, by eliminating the two high scores from the set the mean of the reduced set was much closer to the median. Another measure of central tendency is the *mode*. This value is the value that occurs the greatest number of times in a sample set. Obviously this means that some data sets may not have a specific mode (e.g. a uniform distribution), but most distributions have a mode.

5.3 Measurement Spread

Besides the mean, or more generally the central tendency, it is also of interest to discover how spread out the data is around that central value. The most common measure for this is the *standard deviation* of the data set. The standard deviation is, as its name suggests, a measure of how each value deviates from the mean value. If we define how much a single data point, x_i, deviates from the mean, $\overline{x}$, as $d_i = x_i - \overline{x}$, then the sample standard deviation is computed as:

$$S = \sqrt{\sum_{i=1}^{k} \frac{(x_i - \overline{x})^2}{(k-1)}}$$

The reader may wonder why the deviation term in the numerator is squared. This is because the deviation may sometimes be positive, and sometimes negative. Because we wish to track the deviations for either side of the mean, we square the term to make all values positive, and then take the square root at the end. A related term is the *sample variance*, which is just the sample standard deviation squared, or S^2. Clearly, this is simply expressed as:

$$S^2 = \sum_{i=1}^{k} \frac{(x_i - \overline{x})^2}{(k-1)}$$

these values can also be computed for discrete populations by substituting μ for $\overline{x}$. The population standard deviation symbol is σ. Thus, we have the population standard deviation:

$$\sigma = \sqrt{\sum_{i=1}^{k} \frac{(x_i - \mu)^2}{(k-1)}}$$

and the population variance:

$$\sigma^2 = \sum_{i=1}^{k} \frac{(x_i - \mu)^2}{(k-1)}$$

For the battery sample set in Figure 5.1, the following *sample* parameters are computed in Table 5.1

Table 5.1. *Battery Voltage Measurement Statistics*

Parameter	**Symbol**	**Value**
Mean	$\overline{x}$	12.0002 volts
Median	x_{median}	12.0003 volts
Mode	x_{mode}	11.9971 volts
Standard Deviation	S	0.0012 volts
Variance	S^2	0.0000014766 volts^2

It should come as no surprise, given the computer-based nature of many measurement systems, that we most often compute sample parameters. While we may infer population-based values from our data, the computer-based computations that we make are almost always the sample-based computations above. However, to be thorough the reader should note that the *population variance* is defined as

$$\sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 f(x)dx$$

In this case, the population standard deviation is defined as the square root of the population variance, namely

$$\sigma = \sqrt{\int_{-\infty}^{+\infty} (x-\mu)^2 f(x)dx}$$

EXAMPLE

Figure 1.2 shows the plot of repeated acquisitions of an ECG. In cardiac electrophysiology (the study of the electric circuit of the heart) it is recognized that there are some subtle signals that may appear in the ECG, but that are so small that their signal-to-noise ratio is too low for them to be seen. A technique called *signal averaging* was developed to improve signal-to-noise ratio [Berbari]. In Figure 1.2, the bottom trace represents the average of each separate ECG acquisition, on a point-by-point basis. This assumes that each *measurement* (which incorporates about a half-second of data) can be time-aligned so that the average is being taken over the same point in time for the data set. Originally this method simply acquired 200 ECG samples and averaged them across each time-aligned point. An improvement to the signal-averaging process uses the variance to determine whether adding the next new ECG beat into the average reduces (improves) the noise level or not [Lander]. So, a running computation of the variance at a quiescent portion of the ECG cycle is computed for each new beat. When the variance decreases with a new beat, then that beat is added into the average. If the variance increases with the new beat, then that beat is discarded and not included in the average. The upper panel of Figure 5.8 shows 20 time-aligned ECG beats selected from a recording for averaging. Also shown is the quiescent time period in which the mean variance of the beat average will be computed. By this, it should be understood that the variance will be computed across the same time points for each successive beat, and then the mean of the variance over the time-period will be computed so the *noise* in the system is reduced to a single parameter. The lower panel of the figure shows the average beat (from the 20 individual beats), and shows the mean variance over the quiescent zone to be 0.013424 volts.

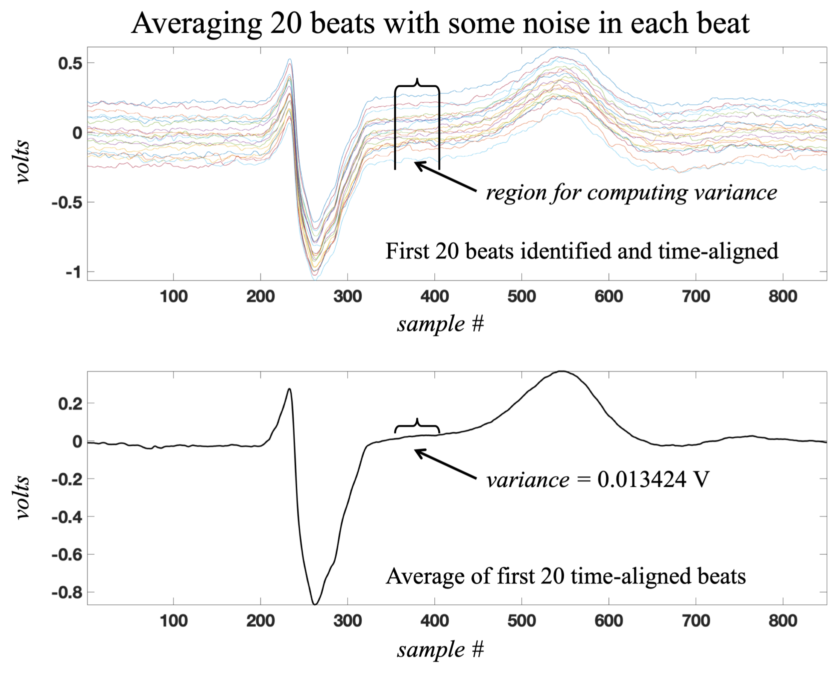


Figure 5.8. ***Beat Averaging in the ECG:*** *The upper panel shows 20 separate ECG beats extracted from a recording. Each trace represents a single* experimental measurement *of the ECG. When we average these beats we choose the same index (point) across all experiments and average that point. This is done for each sample point in the ECG. The lower panel shows the resulting average beat, and the mean variance in the quiet region is computed to be 0.013424 volts.*

When the 21st beat is extracted we compute the new mean variance for all 21 beats, and find that the variance *increases* to 0.014316 volts. This does *not* help improve the signal-to-noise ratio, so we decide to exclude the 21st beat from the average. This is shown in Figure 5.9 where the upper panel is the 20-beat average (from Figure 5.8) and the lower panel is the average beat that includes beat number 21. Although it is subtle, there is a slight increase in the irregularity of the average beat in the lower panel due to the addition of an excessively noisy individual beat. It is shown that the variance (our proxy estimate of the noise level) increases with the addition of the 21st beat.

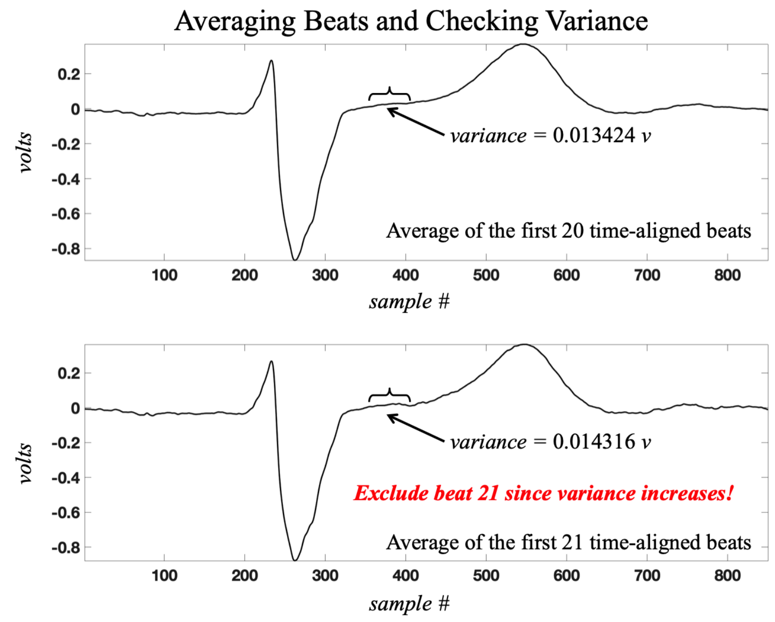


Figure 5.9. ***Beat Exclusion in the Averaged ECG:*** *The upper panel shows the averaged beat for the first 20 beats, and is the same as the lower panel of Figure 5.8. The lower panel shows the average beat, including the 21st beat. Including this beat in the average* increases *the noise level, so this beat should be rejected from the average.*

In general, most of our interest in statistics with respect to making measurements is found in the first two major parameters (moments): sample mean and sample variance (or its square root, standard deviation). There are higher-order statistical measures, such as *skewness* and *kurtosis*, but the interested reader should consider studying a statistics course to further understand the deeper implications of higher-order statistics.

5.4 Probability Distributions

Because the measurement data have a random component, they are randomly distributed across the sample space. The manner in which the data are distributed across the sample space gives us information about the underlying statistics of the distribution. In essence, if we plot the number of occurrences of each result (within the sample space), we can develop a picture of the probability of each sample. When we compute this from the actual data, we call this a *histogram*. Histograms are discrete by nature, since they are generated from discrete data sets, but the underlying process being measured is often continuous. To simplify the histogram process, we generally assign data *bins* that cover some subset of the sample space. For example, if we measure a 3.3 Volt battery and our dynamic range is set in the 0 to 5 Volt range, we might divide this sample space every tenth of a volt (0.1 Volts) so that the first bin covers any measured voltage between 0 and less than 0.1 Volts. The second bin covers from 0.1 to values less than 0.2 Volts, etc. Figure 5.10 shows this process for 1024 measurements of a 3.3 Volt battery. The data were read from a low-cost multimeter and the sample mean (in the figure, the sample mean is cavalierly noted as μ) was computed to be 3.3 Volts. Also, the sample standard deviation was computed to be 0.05 Volts. By dividing the histogram into 50 bins (about 0.1 Volts per bin), and counting the number of times a measurement fell into each bin range, we produce the histogram in the lower panel of the figure.

The reader may note in Figure 5.10 that the histogram (in blue) does not precisely match the red trace that indicates the Gaussian distribution for the computed mean and standard deviation. It is important to realize that this figure is based on 1024 data points, while a continuous Gaussian distribution is theoretically infinite. If many more data points were collected and added to the computations, the histogram in Figure 5.10 would more closely match the red Gaussian distribution.

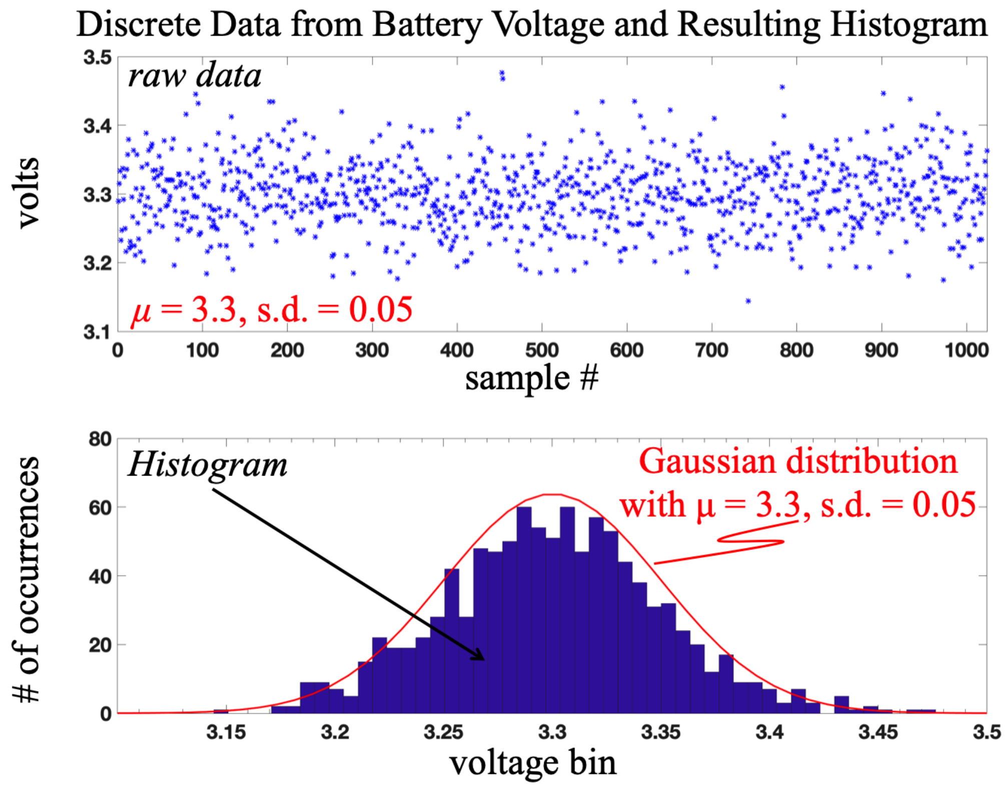


Figure 5.10. ***Discrete Sampled Data and Resulting Histogram:*** *The upper panel shows the 1024 discrete voltage measurements of a 3.3 Volt battery. The lower panel shows the resulting histogram computed from the data set. The computed sample mean and standard deviation are 3.3 and 0.05, respectively. A continuous normal (Gaussian) distribution is overlaid on the histogram based on the computed mean and standard deviation.*

Probability distributions often have mathematical models associated with them that help describe the probability of a specific value, or range of values, occurring. When we use the mathematical model to graph these probabilities over the possible sample space, we get a plot of the probability density function (pdf). As pointed out previously we can always expect at least some value within the sample space to occur. Some may be more likely than others, but if we account for *all* values in the sample space, the probability that we will get one of the values from an experiment is 1 (meaning it is a certainty). The two most important pdf's in measurement and instrumentation are the *normal* distribution and the *uniform* distribution. These distributions will be discussed in more detail next.

5.4.1 Normal Distribution

The normal distribution, also frequently referred to as the *Gaussian* distribution, is probably the most common distribution with which technical communities deal. It turns out that many processes in the world have a normal distribution. For example, the average height of men and women in the United States has been determined to be a normal distribution. Figure 5.11 shows a histogram of the heights in the US, split out to account for women and men separately. Statistics suggest that the average height for all women in the US is 64″ (not separating for ethnic background) with a standard deviation of 3.5″. On the other hand, men in the US have an average height of 70″, with a standard deviation of 4″. The difference in standard deviations is seen in the width and height of the distribution function. The probability density function is plotted in a darker color (dark pink for women and dark blue for men). The scale on the y-axis is in terms of probability for each height, with 1 being certainty. Theoretically, the sample space is infinite if we can measure each height with an infinite precision. However, practically, the sample space is finite and discrete, but with enough granularity to be able to treat it as a continuous distribution.

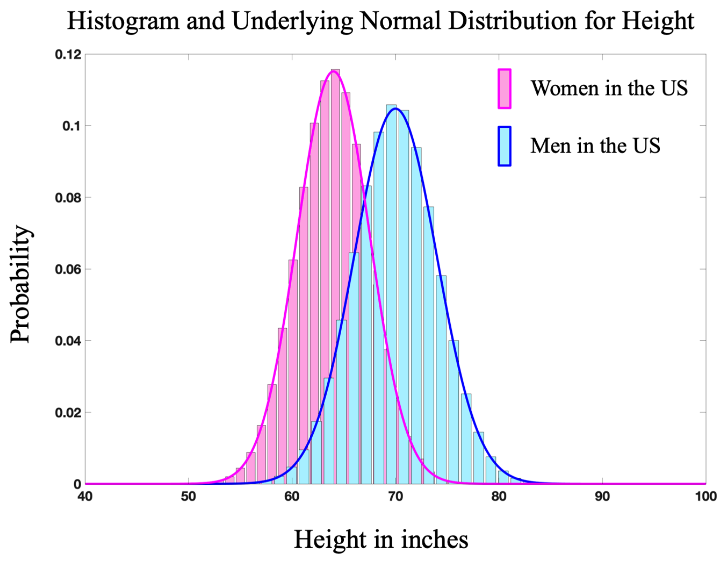


Figure 5.11. ***Histogram and Distribution of Height in the US:*** *The lightly colored bars show a histogram of height distribution amongst both women and men in the US (not accounting for ethnic differences. The darker solid lines show the probability density function (pdf) for each data set. The y-axis values are normalized for the histogram based on the population size so that they appear as probabilities and match the corresponding pdf probabilities.*

So, what is the probability density function of the normal distribution? Mathematically, this is given by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

where μ is the population mean and σ is the population standard deviation. The population mean, μ, describes where the distribution is centered on the x-axis. The population standard deviation, σ, describes how spread out the central region of the graph is. Higher σ values will cause the distribution peak to be lower, and the central lobe to be wider. This is expected, because the standard deviation describes the spread of the data set. As the standard deviation increases the data become more spread. Figure 5.12 shows three different normal distributions having a mean of 1, and standard deviations of 0.1, 0.2, and 0.3. For a given standard deviation, σ, we know from this distribution that approximately 68.26% of the population falls within $\pm\sigma$. If we look beyond this to $\pm 2\sigma$, we find that about 95.5% of the population falls within this range. When we go to a range of $\pm 3\sigma$ we incorporate about 99.7% of the population. In other words, if we chose an item randomly from a population, the probability of choosing an item between $\mu - \sigma$ and $\mu + \sigma$ is ≈ 0.6826. We determine these probabilities associated with standard deviation by integrating the pdf from $\mu - \sigma$ to $\mu + \sigma$. Mathematically, we would write

$$\Pr(\mu - \sigma \leq X \leq \mu + \sigma) \quad = \quad \int_{\mu-\sigma}^{\mu+\sigma} \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

which we can state as meaning "the probability that X falls within one standard deviation of the mean."

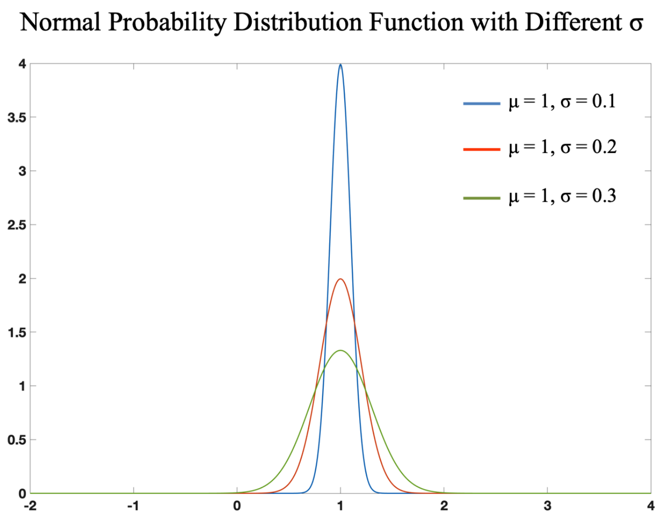


Figure 5.12. ***Three Normal Probability Distribution Functions with Different Standard Deviation:*** *The mean for each distribution shown is 1. However, the standard deviation increases from 0.1 to 0.3. This indicates how spread out the data are.*

Many noise processes in engineering have a normal distribution. For example, we might assume that the noise corrupting an ECG recording is zero-mean, white, Gaussian

(ZMWG) noise. This means that, if we could remove the *known* ECG from the data and compute the mean and standard deviation of the residual signal (which we call noise, whether it is from measurement uncertainty, 60 Hz coupling from fluorescent lights, or other sources) we would find that $\mu = 0$. The *white* term indicates that it fills the spectrum within our sampling criterion. So, if $f_{max} = 500$ Hz for the signal, then the noise fills the spectrum from 0 to 500 Hz. The term white is a reference to white light, which includes all frequencies of the visible light spectrum. Of course, the astute reader will know that we cannot remove the ideal ECG (because we don't actually know it), but if we remove the average beat from a number of ECG beats (each time-aligned beat represents an *experiment*), and we do this for a number of different patients, it can be shown that the residual noise is close to zero-mean across many patients and many collected beats. In instances when we have no information from which to even make an assumption about the corrupting noise process, a normal distribution is a good assumption if we can at least determine the mean value of the noise. Indeed, some *ideal* filters–such as the Kalman Filter–are only ideal when the additive noise process is Gaussian in nature.

However, the reader should keep in mind that the normal distribution is inherently continuous in nature and is *not* a discrete distribution. But, we know that our discretization of measurements into the computer is (hopefully) a good approximation of the continuous-time process in the physical world, so we assume that we can infer a continuous-time distribution in many instances of measured data sets. Students should not find it surprising that the noise processes that we tend to try and filter out of our computer-based measurement systems are frequently distributed normally. Thermal noise in electronics, for example, is usually modeled with a normal distribution. The ubiquitous nature of the normal (Gaussian) distribution is an underlying reason why many of the signal processing algorithms developed assume a Gaussian noise distribution.

5.4.2 Uniform Distribution

The uniform distribution has a different characteristic; namely that every value in the sample space is equally probable. In other words, the probability of any particular value occurring is uniform across the entire sample space (i.e. the probability is the same for all possible samples in the sample space). Mathematically, this is defined as being the area of the probability density function divided by the difference between the upper and lower limits of the sample space. The observant reader will recognize that the area under the pdf curve is always 1, so the probability density function associated with a uniform distribution whose sample space ranges from a to b, is given by

$$\Pr(X) = \begin{cases} 0 & x < a \\ \dfrac{1}{b-a} & a \le x \le b \\ 0 & x > b \end{cases}$$

The uniform distribution, since the probabilities are equal for every value in the sample space, is simply a horizontal line. Figure 5.13 shows a continuous uniform distribution with a sample space between 3 and 8. Thus, $\Pr(X)$ becomes 0.2 for all values between 3 and 8, and zero for all other values.

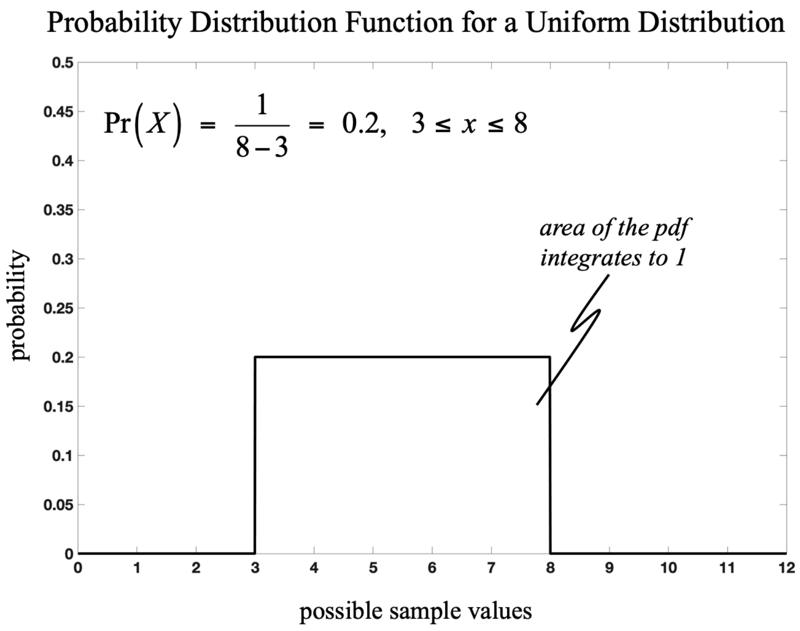


Figure 5.13. ***Uniform Probability Density Function:*** *The uniform probability distribution has an equal probability for all possible values in the sample space. If the sample space is continuous on the interval from 3 to 8, then the probability is simply* $1/(8-3) = 0.2$ *for all values between 3 and 8. The probability is zero outside this interval.*

Uniform distributions do not have to be natively continuous. A common example of this is the tossing of a fair die. Dice all have six sides with corresponding numbers of dots. This is inherently discrete, in that we have six discrete values (our sample space is discrete and finite), namely {1, 2, 3, 4, 5, 6}. If a die is fair, we expect that each face to be equally likely as the up-face when we toss the die. In this case, the difference between the upper and lower limits is 6-1 = 5, except that for discrete data sets we must determine if the edge values are included or not. Since they are in this case, there are a total of six equally likely outcomes and the probability of each outcome is 1/6. This is shown in Figure 5.14 where it should be observed that it is plotted as a set of discrete probabilities. There is no possibility that a value between one of the integer values can occur. Note that when the data is discrete it is referred to as a probability *mass* function (pmf).

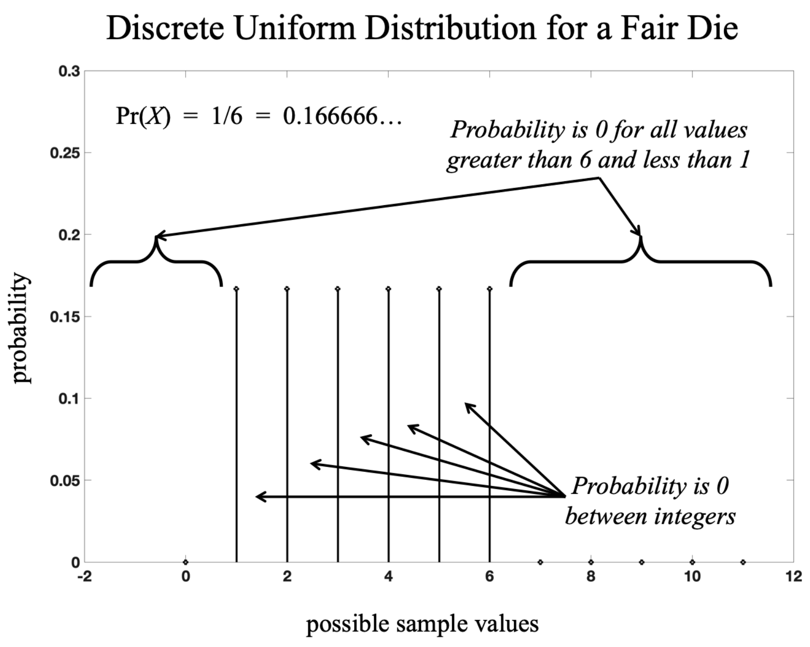


Figure 5.14. ***Discrete Uniform Probability Mass Function:*** *The discrete uniform probability distribution has an equal probability for all possible* discrete *values in the sample space. In addition to the probability of an outcome outside the sample space (1 to 6 inclusive in this case) being zero, the probability between values in the sample space is also zero, by definition.*

Engineers can expect to come across both continuous and discrete uniform distributions, but the discrete uniform distribution is very common in probability theory related to gambling. This occurs because both dice and cards are discrete in nature. Also, a roulette wheel has a discrete set of slots which we expect to all have equal probability (assuming it is a *fair* roulette wheel).

5.4.3 Poisson Distribution

Another discrete distribution is the Poisson distribution. The possible outcomes are countable whole numbers, and each occurrence (event) is independent of all others. The Poisson distribution provides information about the probability of x number of events occurring within some specific time period. When trying to decide how many cash registers to have active during some time period in the day, an Industrial Engineer might use the Poisson distribution to look at the probability of x customers coming to check out half hour, or perhaps every hour. The engineer might gather data over an entire day, and repeat the process every day for a month. Then for each half-hour (or hour) increment, one can estimate the number of people who arrive for checking out. People cannot be subdivided, so one must take a whole-number value for the number of people needing to check out in a given time period.

The probability density function is given by

$$p(k) \quad = \quad e^{-\lambda T}\frac{(\lambda T)^k}{k!}$$

where λ is the occurrence rate, T is the time interval, and k is the number of occurrences being tested against the distribution. The pmf of the Poisson distribution is shown in Figure 5.15 for several values of λ.

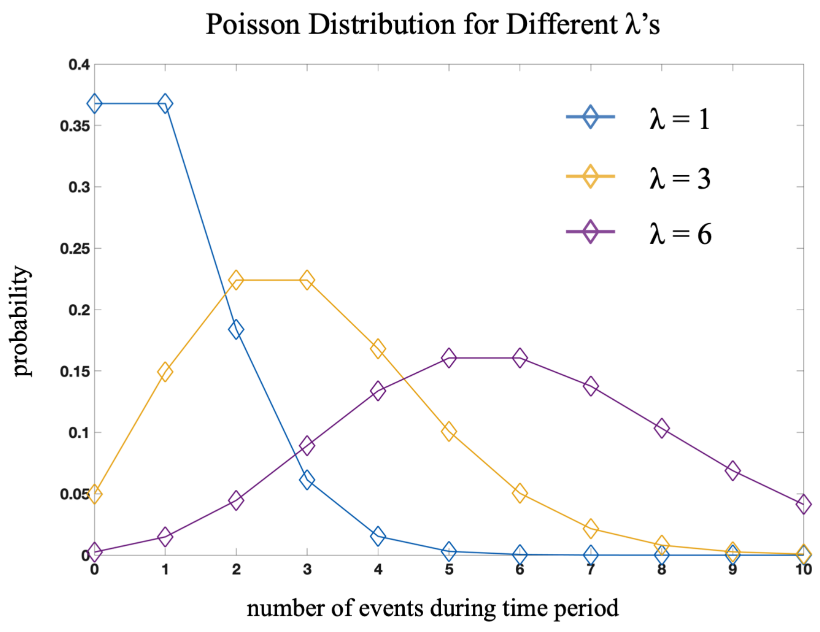


Figure 5.15. ***Poisson Probability Mass Function:*** *The probability of each discrete event is shown with a diamond. The probabilities do* not *exist between the whole numbers, but the connecting line is used to help the reader see the shape of the distribution for each λ. $T = 1$ in this example.*

EXAMPLE

In wireless communication with one receiver and multiple transmitters, each transmitter is unaware of the other transmitters and one–or more–transmitters may attempt to send a packet to the receiver at the same time. When multiple transmitters transmit simultaneously a collision occurs, and the packets must be re-transmitted after some random delay (to try and avoid future collisions). The arrival of external packets to the receiver unit is modeled with a Poisson distribution. A successful transmission occurs if zero other arrivals occur when an initial packet arrives in a time slot. Letting the average number of arrivals for a time slot be N and let S be the successful arrivals, $S = \lambda T$. Note that the successful arrival assumes that only one packet was transmitted, or more specifically, zero extra packets were transmitted so no collision occurred. Then the probability of a successfully transmission occurs when zero extra packets are transmitted (no collisions), so that $S = NP_0$ where N is the average attempts per frame (note that N is not an integer value, but a mean value) and P_0 is the probability of zero *other* packets being transmitted. Using the Poisson distribution P_0 can be defined as

$$P_0 \quad = \quad (2\lambda N)^0 \frac{e^{-2\lambda N}}{0!} \quad = \quad e^{-2\lambda N}$$

Then the successful transmission rate can be determined to be $S = Ne^{-2N}$. This is shown in Figure 5.16.

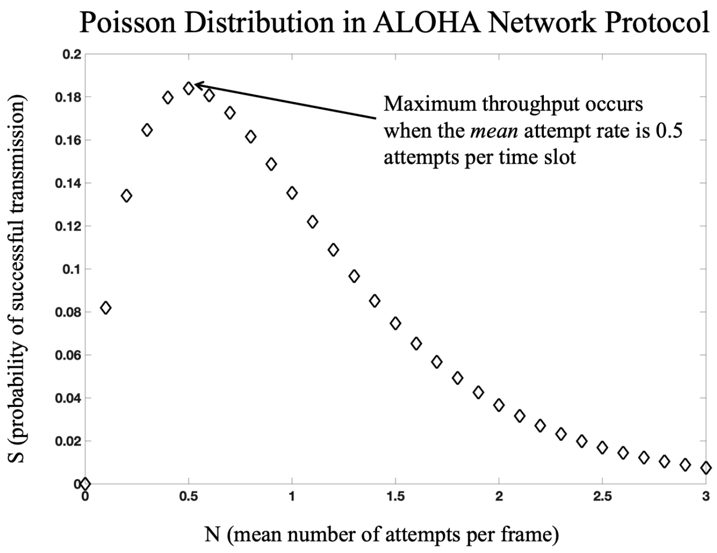


Figure 5.16. ***Poisson Distribution Applied to the ALOHA Communications Protocol:*** *After deriving the expression for probability of a successful transmission using the Poisson distribution, finding the mean attempt rate that corresponds to the highest success rate defines the maximum throughout of the system.*

Note that the derivative of this expression gives us the peak, which is where the maximum throughput for this system can be achieved. In this case the maximum throughput occurs when the mean number of attempts is 0.5 attempts per time unit (the mean attempts that provides the highest probability of success determines the throughput because a lower probability of a success transmission increases the number of retires, which clearly reduces throughput).

(Northwestern University, ECE 333 Lecture, http://www.ece.northwestern.edu/~rberry/ECE333/Lect2001/lec14.PDF)

5.4.4 Binomial Distribution

Yet another discrete statistical distribution that we encounter in the technical fields is the binomial distribution. The binomial distribution measures the probability of *success* for n independent trials (experiments, outcomes). In this scenario, the outcome of an experiment is either a *success* or a *failure*. Because there are two opposing options for the outcome, the distribution is referred to as binomial. An individual experiment with a true/false, success/failure, yes/no type of result is called a *Bernoulli trial*. Repeating a Bernoulli trial n times gives us the binomial distribution. Mathematically, the binomial distribution's probability mass function is written as

$$\Pr(X = k) \quad = \quad \binom{n}{k} p^k (1-p)^{n-k} \qquad k = 0, 1, 2, 3, 4, ..., n$$

where X is the random variable that counts the number of successes, k is the number of successes (note that this is the variable by which we test the probability for various values of k), p is the probability of success in a single Bernoulli trial, $(1 - p)$ is the probability of failure, and n is the total number of trials executed. The first term on the right-hand side of the equation is a notation used for choosing k out of n trials, and is computed using the following expression

$$\binom{n}{k} \quad = \quad \frac{n!}{k!\,(n-k)!}$$

In many cases the probability of success, p, may be considered constant. But it should be noted that we do not have to always let p be the successes. We might look at this from the other side of the coin and, if we desire to track failures actively, let p be the probability of failure and $1 - p$ be the probability of success. The point is that it is binomial so the two probabilities must sum to one. The binomial distribution is useful in Quality Control (QC) departments in a manufacturing facility. Typical manufacturing facilities generate N widgets each day. The QC department is tasked with determining if the manufacturing process is adequate, i.e. whether the widgets conform to expected operational specifications, or not.

EXAMPLE

An appliance manufacturer produces washing machines at one of its facilities. It is determined that if not more than 2 units fail a system test for 95% of the machines produced each day, then the production conforms to specifications. If the number exceeds 2, then the entire day's production must be re-verified. If we let X represent the random variable for the number of successes (this is an instance when we are treating finding a *failure* of system test as a "success."), we want to know the probability that $X \leq 2$, or $\Pr(X \leq 2)$. The head of the QC department wants to know the chance that the quality engineer will incorrectly certify a day's production when only 70% of the machines meet the desired specifications. Note that if 70% meet specifications, this is the same as 30% not meeting specifications. It has been determined that 8 machines will be pulled for quality control testing each day (averaging one per hour, but not necessarily taken exactly one per hour).

We need to find the probability, $\Pr(X \leq 2) = \Pr(X = 0) + \Pr(X = 1) + \Pr(X = 2)$. So we need to solve the binomial distribution equation for $k = \{1, 2, 3\}$, $n = 8$, and $p = 0.7$. Mathematically, we have

$$\begin{aligned}
\Pr(X \leq 2) &= \Pr(X = 0) + \Pr(X = 1) + \Pr(X = 2) \\
&= \binom{8}{0}(0.3)^0(1-0.3)^{8-0} + \binom{8}{1}(0.3)^1(1-0.3)^{8-1} + \binom{8}{2}(0.3)^2(1-0.3)^{8-2} \\
&= 0.05765 + 0.19765 + 0.29648 \\
&= 0.5518
\end{aligned}$$

This is an example, so the numbers are not very realistic...or perhaps they are. When the manager of the QC department performs these calculations, it will be determined that the 70% number needs increasing, or possibly decrease the expectation of the number of failures (i.e. no more than one failure instead of two). The point is that the binomial distribution has applications in many engineering environments.

5.5 Spectral Analysis

One of the important facets of signal analysis is spectral analysis. In a series of measurements we might be able to detect errors of a periodic nature, such as line noise. The *power spectrum* of a discrete signal is usually the starting point for spectral analysis. The power spectrum is simply the Fourier Transform of the signal squared. However, the reader should keep in mind that the Fourier coefficients are complex numbers, so squaring them involves multiplying each coefficient times its complex conjugate. Mathematically, the student should recall that the discrete Fourier transform (DFT) for a discrete signal, $x[n]$, is given by

$$\tilde{X}[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j\left(\frac{2\pi kn}{N}\right)} \qquad k = 0, 1, 2, 3, ..., N-1$$

Recalling that sampled frequencies are integer multiples of a fundamental frequency, we know that $\omega = 2\pi/NT$ where T is the sampling period. Replacing this term into the equation above we have

$$\tilde{X}[\omega_k] \quad = \quad \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j\omega nT}$$

Regardless of which formulation is used (the first one is what a computer program will do because it understands discrete integer values k and n), the power spectrum is then simply computed by

$$\tilde{P_x}[\omega_k] \quad = \quad \frac{1}{N}\tilde{X}[\omega_k] \cdot \tilde{X}^*[\omega_k]$$

where the * denotes conjugation. While it is good to have a solid understanding of the theory, the practical aspect is that one can compute the DFT using the FFT built into nearly every engineering software package. It is noteworthy that the FFT (Fast Fourier Transform) is a specific computer algorithm shown to be very efficient in computing the DFT. MATLAB® has an FFT algorithm, LabVIEW has an FFT algorithm, Mathematica has an FFT algorithm, as do other packages. In MATLAB® for example, if we have a vector $x[n]$ we can find its power spectrum by

```
> X = fftshift(fft(x));
> P = X.*conj(X);
> plot(X);
```

This produces a *dual-sided* spectrum, namely from $-\frac{\omega_s}{2}$ to $+\frac{\omega_s}{2}$, which can be adjusted by 2π to $-\frac{f_s}{2}$ to $+\frac{f_s}{2}$ (the s subscript denotes sampling frequency). A simple example dealing with Dual Tone Multi-Frequency (DTMF) signaling follows.

EXAMPLE

When the telephone companies converted from analog rotary dials to push-button dialing, they developed an encoding scheme called Dual Tone Multi-Frequency (DTMF) signaling (Note: Chapter 8 will go into more details of DTMF). Each digit on a push-button phone encodes two sinusoids (frequencies). These sinusoids, during transmission, are corrupted by some level of additive noise. However, even with poor signal-to-noise ratio the frequencies are easy to detect by power spectrum analysis. Each digit is encoded with one high frequency and one low frequency according to the following table:

Table 5.2. *DTMF Frequency Assignments*

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

It is useful to know that the phone company samples analog signals at 8000 Hz, so all of these frequencies are well below the Nyquist-defined frequency of 4000 Hz. Furthermore, they were carefully designed to not have any common harmonics so that spectral estimation is definitive. Figure 5.17 shows a plot of 0.1 seconds of DTMF data encoding the digit "6." There is gaussian additive noise that provides a signal-to-noise ratio (SNR) of -11 dB (meaning the noise amplitude is more than 3 times greater than the sinusoid amplitude).

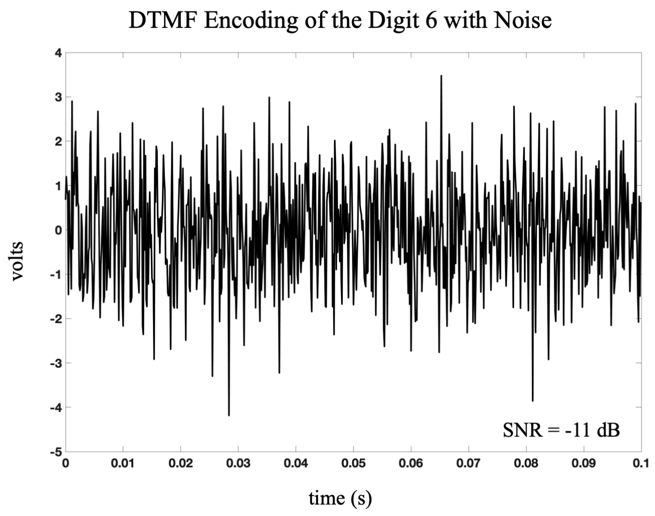


Figure 5.17. ***DTMF Encoding of the digit 6, with SNR = -11 dB:*** *The plotted signal has two pure sinusoids of 770 Hz and 1477 Hz, with additive noise that is more than 3 times greater in amplitude than the combined sinusoids.*

After computing the FFT of the signal, and multiplying the result by its complex conjugate, we get a spectrum that clearly shows the sinusoids. This is shown in Figure 5.18, where the spectral peaks dominate the residual noise. This is partly because Gaussian noise spreads its frequencies across much of the available spectrum (4000 Hz in a system that samples at 8000 Hz), so the power at any given frequency for just the noise is smaller in the presence of even the slightest periodic frequency content. The reader should keep in mind that the SNR in this case is -11 dB. By extending the test with more noise, we find that the simple power spectrum can still detect the two sinusoids even when the SNR reaches -20 dB.

One practical concern in spectral analysis is the mean value of a signal being analyzed. The mean value of a signal is often referred to as its *DC offset*. In the frequency domain, a constant DC offset in a signal shows up at the zero-frequency end. Indeed, in small signal analysis, a DC offset can swamp any other frequency content. Therefore, it is critical to always *remove the mean of a signal before performing any spectral analysis*. If the mean value is of importance, then it can be computed and stored, but the mean (DC offset) should always be removed before any further spectral or statistical analysis is performed.

Power spectral analysis is not necessarily exactly how the phone company decodes each button push, but this example shows the utility of the basic power spectrum. It is beyond the scope of this text, but there are extensions to the power spectrum that make it even more robust when significant noise is present. The periodogram breaks the data into (possibly) overlapping sections, computes the power spectrum of each section and then takes the average. This, of course, relies on the data statistics to be relatively stationary over the recording epoch. There are also parametric methods that are generally taught in graduate level signal processing classes. A good text for more in-depth information about the power spectrum in the discrete (PC-based) setting is a text by Monson Hayes [Hayes].

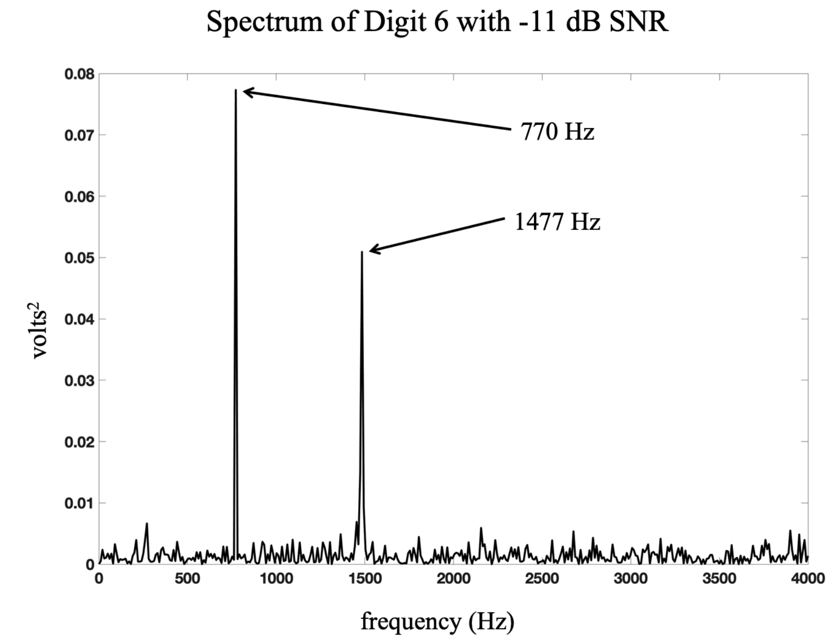


Figure 5.18. ***Power Spectrum of DTMF-encoded digit 6, with SNR = -11 dB:*** *The power spectrum (the positive frequency half) clearly shows the two sinusoids, 770 Hz and 1477 Hz, even with SNR = -11 dB.*

5.6 Conclusion

This chapter exposes the student to some statistical concepts in the measurement arena that one needs to understand. Measurement uncertainty finds its roots in these statistical computations and must be understood to properly appreciate the veracity (or lack thereof) of a given measurement. The most commonly used statistics across all disciplines are certainly the sample mean and sample standard deviation. But there are other parameters presented here that are also useful in the measurement and instrumentation field. Particularly, the power spectrum and its derivatives are often employed in understanding underlying characteristics of a measured process, as demonstrated in the practical applications discussed in Chapters 6 through 8.

Chapter 6

There are many example projects that could be included in this text that would allow us to apply some of the concepts covered in the first five chapters to actual systems. Chapters 6, 7, and 8 will provide the details for three such projects. For each of these projects, the background information is explained, and the details of the projects are outlined to the extent that the reader can attempt to complete these projects on their own in order to solidify their understanding of the underlying concepts. We will begin in Chapter 6 with a classical mass, spring, damper second order mechanical system.

6.1 Physics of a second-order mechanical system

Many mechanical systems can be modeled with a second-order differential equation. For example in car suspension systems the car is the mass, the suspension is the spring-and-damper component, and the road is the fixed surface. This is diagrammed in Figure 6.1, where the position of the car is represented as $x(t)$. As the car experiences bumps in the road, the suspension system works to minimize the vertical movement that the driver experiences. Even though the y variable is often used for vertical position, we will use x as the variable in this example because it is by far the most frequently used variable to represent displacement in linear mechanical systems. The expression governing the vertical position of the car is based on Newton's Second Law, which in modern times we frequently couch simply as $F = ma$. In the mass-spring-damper problem, let us refer to the mass as M, the spring constant as k, and the damper value as D. To maintain the conservation of forces we must set all forces acting on the mass to zero. So, what vertical forces are acting on the mass, and how do we create a mathematical model of them?

- Gravity: This one is the most obvious, because this force is directly computed as $F = Ma$. However, we must note that acceleration, a (or $a(t)$ to be more accurate), is simply the second derivative of position, $x(t)$. So, the force exerted on the mass from gravity is modeled as:

  $F_{mass} = M\frac{d^2x(t)}{dt^2}$

- Damper: The damper is a rate-of-change based device. The faster the damper tries to change position, the more damping occurs (it is noteworthy that in a car, the shock absorber is the damper, but shock absorbers are designed to have different damping rates for compression and extension; in this example we will assume a single damping rate). Because the force related to damping is related to velocity, we need the first derivative of position, so that the damping force is modeled as:

  $F_{damper} = D\frac{dx(t)}{dt}.$

- Spring: The spring rate is a constant related to the amount of spring displacement from rest. Therefore, the force related to the spring is directly proportional to the position, $x(t)$. Using Hooke's Law, we can model the force due to the spring as:

  $F_{spring} = Kx(t).$

- External Force: The external force occurs from bumps in the road, as illustrated in the figure below. In the absence of a non-zero external force, the mass should not be moving vertically and the forces due to the mass, spring, and damper should sum to 0.

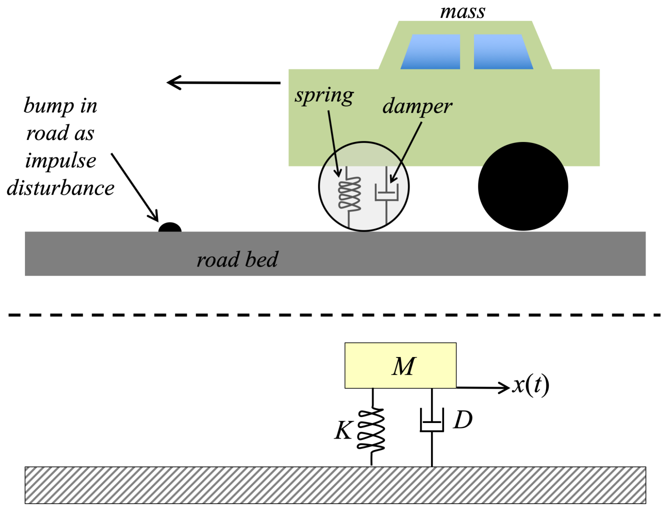


Figure 6.1. ***Mass-Spring-Damper Example:*** *The mass-spring-damper problem is a good simplified version of a car suspension. The car is the mass, the suspension spring is the spring, and the shock absorber is the damper. Position, x(t), of these things relative to the fixed surface helps define the forces exerted.*

If we define positive $x(t)$ to be upward, then we can sum these forces as follows:

$$F_{spring} + F_{damper} + F_{mass} = F_{ext}$$
$$\Downarrow$$
$$M\frac{d^2x(t)}{dt^2} + D\frac{dx(t)}{dt} + Kx(t) = F_{ext}(t)$$

The interested student should review the Laplace Transform to truly appreciate this case study. We know that when a system is energized by an impulse function (the input is an impulse), the Laplace Transform of the resulting output is referred to as the *Transfer Function*. Denoting the operation of the Laplace transform using the notation $\mathcal{L}()$. Letting F_{ext} be the system input, and taking the Laplace transform of both sides of the force equation, we have

$$M \cdot \mathcal{L}\left(\frac{d^2x(t)}{dt^2}\right) + D \cdot \mathcal{L}\left(\frac{dx(t)}{dt}\right) + K \cdot \mathcal{L}\left(x(t)\right) = \mathcal{L}\left(F_{ext}(t)\right)$$

If we take the Laplace transform of $x(t)$ and assume that the initial vertical velocity and position are zero (i.e. zero initial conditions), then the Laplace transform of the second derivative of $x(t)$ is given by:

$$\mathcal{L}\left(\tfrac{d^2x(t)}{dt^2}\right) = s^2X(s) \quad (ROC \subset R)$$

and the Laplace transform of the first-order derivative of x(t) is given by:

$$\mathcal{L}\left(\tfrac{dx(t)}{dt}\right) = sX(s) \quad (ROC \subset R).$$

Therefore, the Laplace transform of the time domain model of our system becomes

$$Ms^2X(s) + DsX(s) + KX(s) = \mathcal{L}\left(F_{ext}(t)\right)$$

The transfer function of this system can be found by rearranging the equation so that the output variable, $X(s)$, is divided by the input variable $F_{ext}(s)$ and isolated on the left side of the equal sign.

$$H(s) = \frac{X(s)}{F_{ext}(s)} \quad = \quad \frac{1}{Ms^2X(s) + DsX(s) + KX(s)}.$$

Finally, a Laplace Transform table will show that the transform of an impulse function is given by $\mathcal{L}(\delta(t)) = 1$, so $H(s)$ is simply equal to the output $X(s)$ because

$$H(s) = \frac{X(s)}{F_{ext}(s)} \quad = \quad \frac{X(s)}{1} \quad = \quad X(s) \quad = \quad \frac{1}{Ms^2X(s) + DsX(s) + KX(s)}.$$

The previous observation is important to note because when solving the model for $X(s)$ you are not only obtaining the displacement in the Laplace domain due to an impulse input (i.e. the impulse response), but at the same time you are also obtaining the transfer function. When the denominator of this transfer function is set equal to zero, the expression is called the *characteristic equation*. This is an extremely important equation because the roots of the denominator tell us the underlying, unforced behavior baked into the system by design (in other words, ignoring external perturbations of the system). This unforced behavior is often referred to as the *natural response*. Recall that the general solution to a linear, constant coefficient differential equation is based on the natural exponent, $e^{\alpha t}$, where we let $\alpha \in \mathbb{C}$ for the most general case. Because $e^{\alpha t}$ is an *eigenfunction*, the roots of the characteristic equation determine the *eigenvalues* of the system, where the value of α is determined from the real part of those roots. When α is negative, the system is considered stable because the exponential, $e^{-\alpha t}$, decays over time so that the natural response goes to zero as time approaches infinity. When the eigenvalues (α_1 and α_2 for a second order system) are real, the resulting impulse response in the time domain is damped to the extent that the output decays over time without any oscillations. Examples of stable systems with real, repeated roots ($\alpha_1 = \alpha_2$) and real, distinct roots ($\alpha_1 \neq \alpha_2$) are shown in Figure 6.2.

On the other hand, a second order system with complex roots ($\alpha \pm j\omega$)) has an oscillatory impulse response as shown in Figure 6.3. This type of system is referred to as an underdamped system because there is not enough damping to remove the oscillations. As for the systems with real roots that were shown in Figure 6.2, α must be negative for the system to be stable. An unstable, underdamped system (i.e. roots with $\alpha > 0$) would produce oscillations that get larger over time. It should be noted that complex roots *always* come as complex conjugate pairs, so any odd-order system must have at least one real root. Whether they are complex or real, the roots occur in the Laplace domain. To get the system transformed into the time domain a process called *partial fraction expansion* (PFE) is used. If the roots are a pair of complex conjugate roots in the Laplace domain they will translate into a decaying sinusoid in the time domain.

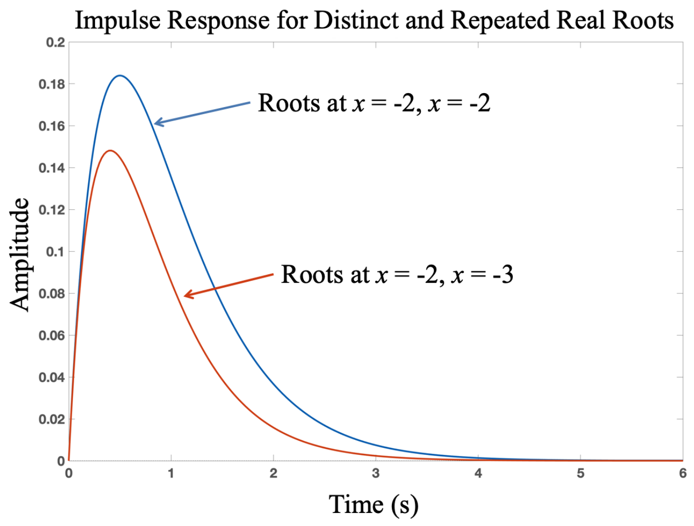


Figure 6.2. ***Impulse Response of Two Second-Order System With Real Roots:*** *The taller trace is due to a second order system with two real, repeated roots as shown. The slightly lower-amplitude trace represents the impulse response of a second order system with two real, distinct roots. In each case, after reaching a peak the signal monotonically decays to zero over time.*

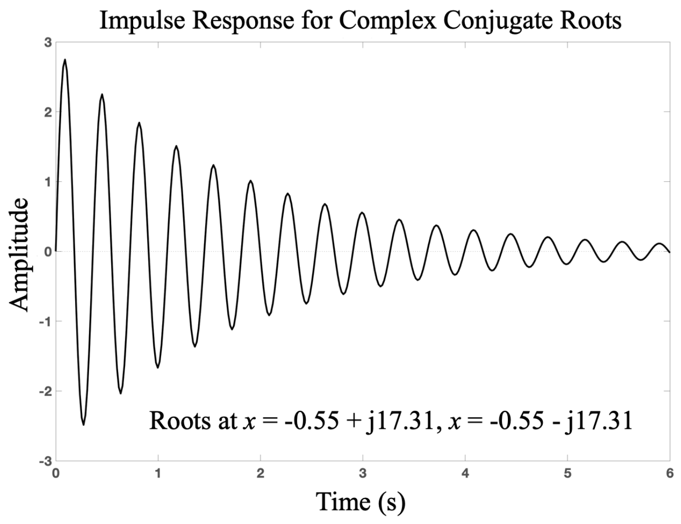


Figure 6.3. ***Impulse Response of a Second-Order System With Complex-Conjugate Roots:*** *In a second order system with complex conjugate roots the system oscillates, as shown by the sinusoidal behavior. If the sinusoid decays, as in this figure, then the damping factor is negative and the overall response decays.*

The reader should recall that one of the seminal Laplace Transform relationships is given by:

$$Ae^{\alpha t} \overset{\mathcal{L}}{\longleftrightarrow} = \frac{A}{s+\alpha}$$

However, we know that in the general case, where $\alpha \in \mathbb{C}$, we get a complex exponential in the time domain. If we have two terms in the Laplace domain from complex conjugate roots, then we will get two complex-conjugate exponentials in the time domain. The oscillatory nature of the time domain response that is produced from these two complex-conjugate exponentials can be understood by Euler's identity for a cosine, which is shown below.

$$\cos(\omega t) = \frac{e^{j\omega t} + e^{-j\omega t}}{2}$$

So far in this example the value of the numerator and the effect it has on the partial fraction expansion process has not been addressed. In a second-order system with complex conjugate roots, the numerator of the two Laplace domain terms (that are found using partial Fraction Expansion) will also be a complex conjugate pair. The following shows how the time domain impulse response is derived from the Laplace domain transfer function of a second order system with complex conjugate eigenvalues.

$$F(s) = \frac{-12}{s+6} + \frac{10e^{-j(0.3\pi)}}{s+3-j4} + \frac{10e^{j(0.3\pi)}}{s+3+j4}$$

From the Laplace table we have these two relationships:

$$\mathcal{L}^{-1}\left\{\frac{k}{s+\alpha-j\beta} + \frac{k^*}{s+\alpha+j\beta}\right\} = 2|k|e^{-\alpha t}\cos(\beta t + \theta)u(t)$$

where θ is the angle argument of k

$$\mathcal{L}^{-1}\left\{\frac{k}{s+\alpha}\right\} = ke^{-\alpha t}u(t)$$

Then the time domain representation of $F(s)$ above is given by:

$$\mathcal{L}^{-1}\{F(s)\} = \left[-12e^{-6t} + 20e^{-3t}\cos(4t - 0.3\pi)\right]u(t)$$

Applying this concept to the second order transfer function, $H(s)$, that describes the suspension system, the time domain representation of the *impulse response* can be determined.

$$
\begin{aligned}
H(s) &= \frac{1}{Ms^2 + Ds + K} \\
&\Downarrow \qquad \textit{taking the roots of the denominator} \\
H(s) &= \frac{1}{\left(s + \frac{D+\sqrt{D^2-4MK}}{2M}\right)\left(s + \frac{D-\sqrt{D^2-4MK}}{2M}\right)} \\
&\Downarrow \qquad \textit{letting each root be represented by} -\alpha \pm j\omega \\
H(s) &= \frac{1}{(s + (\alpha - j\omega))\,(s + (\alpha + j\omega))} \\
&\Downarrow \qquad \textit{partial fraction expansion to get two first-order terms} \\
H(s) &= \frac{-j\frac{1}{2\omega}}{s + (\alpha - j\omega)} + \frac{j\frac{1}{2\omega}}{s + (\alpha + j\omega)} \\
&\Downarrow \qquad \textit{convert the residuals (numerators) to polar form} \\
H(s) &= \frac{\frac{1}{2\omega}e^{-j\frac{\pi}{2}}}{s + (\alpha - j\omega)} + \frac{\frac{1}{2\omega}e^{j\frac{\pi}{2}}}{s + (\alpha + j\omega)} \\
&\Downarrow \qquad \textit{using the inverse Laplace transform as shown above} \\
h(t) &= 2\left(\frac{1}{2\omega}\right)e^{-\alpha t}\cos\left(\omega t - \frac{\pi}{2}\right)
\end{aligned}
$$

The final term uses cosine with a $\frac{\pi}{2}$ shift rather than using sine, but either expression would be correct. The reader should note that the residuals in the denominator are often referred to as k in textbooks, but this should not be confused with the spring constant, K. Many Circuits 2 textbooks use the inverse Laplace relationship shown above for $F(s)$, so that results are often couched in terms of cosine. An important feature of the resulting time domain expression is the phase value. As seen in the formula above the numerator values (after PFE), k and k^* can be represented in polar form so the complex exponent represents the phase. Recall that the phase is determined by:

$$\theta = \tan^{-1}\left(\frac{\mathcal{I}m(k)}{\mathcal{R}e(k)}\right)$$

The quadrant of θ is important and cannot be ignored. The inverse tangent computation of $\frac{-\beta}{\rho}$ or $\frac{\beta}{-\rho}$ produces the same value in many calculators or simple tangent functions, but are clearly not in the same quadrant. The student should take care to ensure the proper quadrant!

Setting aside the details of this derivation, the fundamental message is that the *impulse response* (the output function that results from an impulse function input in the time domain) has the form of a decaying exponential, where the decay rate is controlled by the variable α in the above equations, and the oscillation is determined by ω. In the mass-spring-damper system it is possible to control the variables so that the roots of the system are real, and the system is highly damped. In a car, this would provide an unacceptably stiff and undesirable ride. On the other hand, one does not want the car to bounce down the road, so auto designers have worked hard to find the appropriate amount of damping and spring action to provide a comfortable ride while still maintaining good handling characteristics.

6.2 Project

A project was created that uses the second order underdamped system described in section 6.1, but instead of resting on top of a fixed surface like the car suspension system the mass-spring-damper system is suspended from a fixed surface. Referring to Figure 4.21, we see the same mathematical derivation that provides a second order differential equation for the position of the mass with respect to time. For this project, a mass-spring-damper system (a *bouncy toy*!) is suspended from the ceiling with a tape measure affixed from the ceiling to a table beneath the toy. The tape measure should be assumed to be parallel to the vertical motion of the toy. We do not know M (the mass of the toy), k (the spring constant for the spring suspending it), or D (the damping constant of its paddle blades). The toy is shown in Figure 6.4.

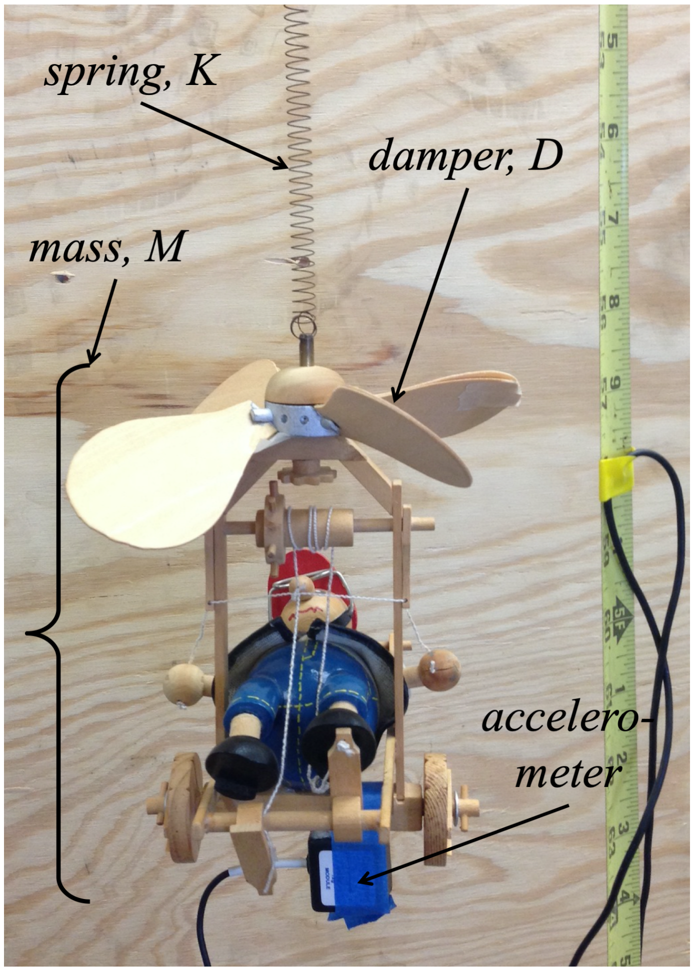


Figure 6.4. ***Bouncy Toy Example of Mass-Spring-Damper System:*** *The bouncy toy (including the accelerometer attached to the bottom) has some mass, and is suspended from a spring with spring constant K. The non-ideal nature of the spring and the big paddle-shaped, rotor-like blades provide the damping factor. Photo of bouncy toy by John Dyer.*

The sensor that will provide provide us with data that allows the system to be analyzed is an accelerometer. We do not know the sensitivity of the accelerometer, but we do know that its output is a single-ended voltage that ranges between 0 volts and the power supply voltage of +5 volts. It is noteworthy that both positive and negative acceleration is represented in this voltage range, so the user should expect a DC offset when the system is completely idle. The tape measure is provided so that the user can measure the initial displacement and use it as an input when trying to determine the mathematical expression for $x(t)$.

The goal of this project is to acquire the accelerometer data with a DAQ device and use the data to determine the *motion equations* for the device. For our purposes, the term *motion equations* implies the equation for $x(t)$ (position), the equation for $v(t)$ (velocity), and the equation for $a(t)$ (acceleration). The available data to be collected are a series of voltages that represent acceleration of the device over time, but we do not know the relationship between the recorded voltages and acceleration. In general, we might take the following steps in executing this project.

1. Choose the sensor. In this case, the user would consider what sort of g-load is expected, what kind of bandwidth is needed (how fast will it oscillate), and what type of electrical output it will provide.

2. Choose the data acquisition device: single-ended or differential capability, dynamic range capability, etc.

3. Assemble the measurement system, including the sensor wiring and the connection from the sensor to the data acquisition device.

4. Produce the data acquisition software, accounting for dynamic range, sample rate, format (single-ended versus differential), duration of acquisition, etc.

5. Analyze the data (presumably using software) and determine the motion equations.

For the implementation of this project in the Measurement and Automation course at OU, steps 1 through 3 have already been completed in that the accelerometer has already been chosen (not that it is guaranteed to be the ideal sensor for this system) and a National Instruments USB-6211 data acquisition device is already present and connected to the accelerometer. So, the students are only required to complete steps 4 and 5.

For step 5, the student must understand the fundamental concepts provided in Chapter 2. Specifically, the student must determine the sampling rate. How does one go about this? Also, the student must determine the dynamic range. How does one determine that? The connection configuration must also be correctly set. The implementation of this project at OU employs the LabVIEW programming environment to acquire and analyze data. Appendix (A.2) provides a basic tutorial for LabVIEW. However, students should internalize the idea that engineering courses do not award ABET accredited engineering credits for learning a commercial software product. So, this book is not in the nature of a LabVIEW training mechanism. Engineering students all take one or more programming courses, and the fundamental concepts of programming are no different in LabVIEW than they are in C, Java, Python, etc. A WHILE loop is a WHILE loop, a FOR loop is a FOR loop, conditional statements (if-then) are just that, regardless of the programming language. So, the task in this case is to learn enough of another language to get the job done. It is a bit like a native English-speaking person, who also speaks French, finding herself in need of interacting in Spanish at a restaurant. With a little help from a vocabulary book, this can be done, because the so-called romance languages all have the same (or very similar) fundamental grammar constructs.

6.2.1 Data Acquisition Programming

Once the hardware setup is complete, the next step is to write a simple LabVIEW program to acquire the data. The first step in this process is to program the DAQ device itself. Fortunately, LabVIEW has several Express Wizards to help the user begin data acquisition in short order. Assuming the student knows how to open a LabVIEW VI session, we first make the *Block Diagram* the active Window. This is done with the key combination CTRL-e, or by going to the Window sub-menu at the top of the Main Diagram window and selecting Block Diagram. Once the Block Diagram is the active window, the user should *right-click* to bring up the *Functions Palette* (see Figure 6.5).

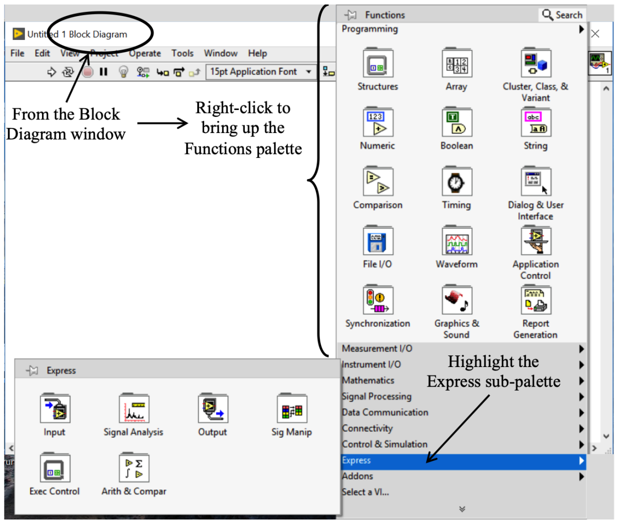


Figure 6.5. ***Block Diagram of the VI with the Functions Palette Activated:*** *To get to the functions that we program with in the Block Diagram, right-click to get the Functions Palette.*

When the Functions Palette is visible, highlighting (but not clicking) on the Express sub-palette brings up anther small window that has a heading of "Express" (see Figure 6.5).

Once the Express sub-palette is open, hovering the mouse over the Input icon brings up yet another sub-palette. In this final sub-palette, the user should click on the DAQ Assist icon (see Figure 6.6). The DAQ Assist is a Wizard that walks the user through the following data acquisition steps: 1) choosing a device; 2) selecting one or more channels; and entering the acquisition parameters associated with the selected channel, or channels. Figures 6.6 through 6.11 show how to accomplish these three steps.

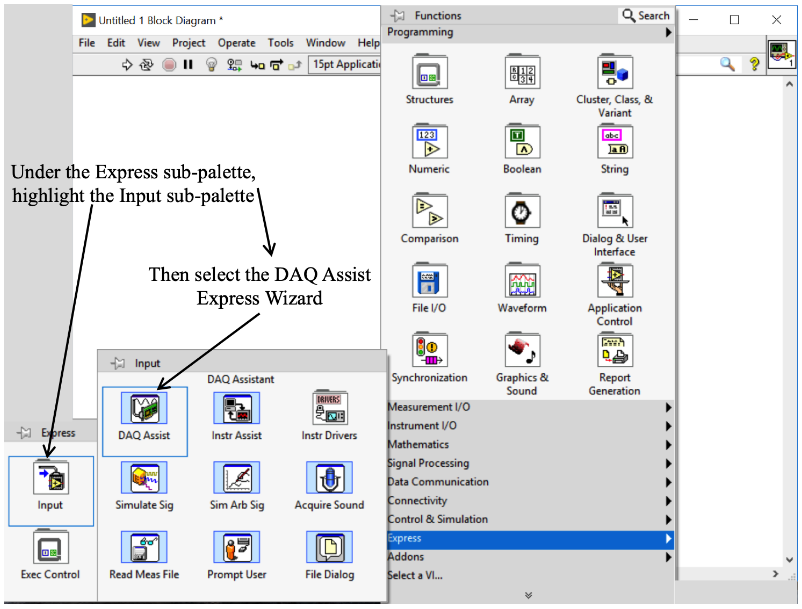
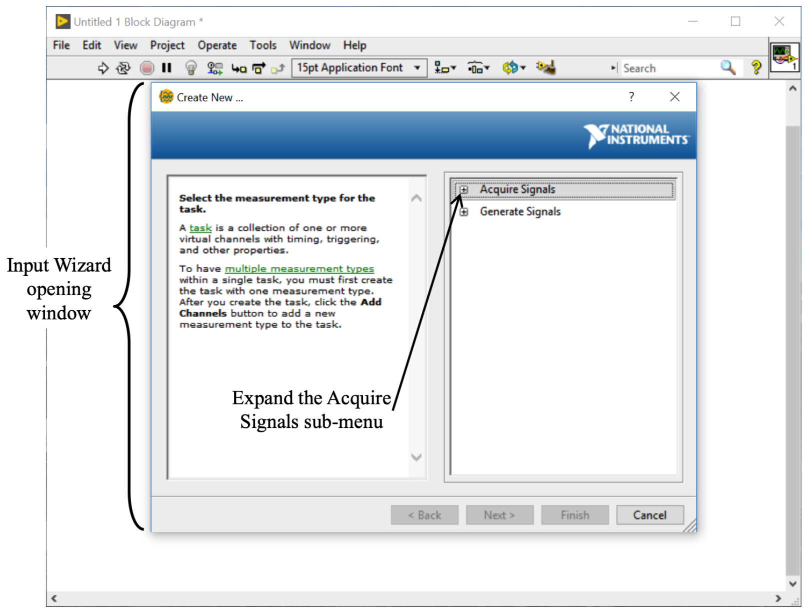


Figure 6.6. ***Getting to the DAQ Express Wizard:*** *Going through several sub-palettes we finally arrive at the palette containing the DAQ Express icon. Selecting this icon activates a Wizard that will walk the user through the setup process.*

Once the DAQ Assist window opens, the user is guided through the process of selecting a device, channels, etc. The first step is to determine if the user is acquiring or generating signals. For this project the user only needs to acquire a single channel of data, so the Acquire Signals option is expanded (see Figure 6.7).


Untitled 1 Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Create New ...
NATIONAL INSTRUMENTS
Acquire Signals
Generate Signals
< Back
Next >
Finish
Cancel
Input Wizard opening window
Expand the Acquire Signals sub-menu

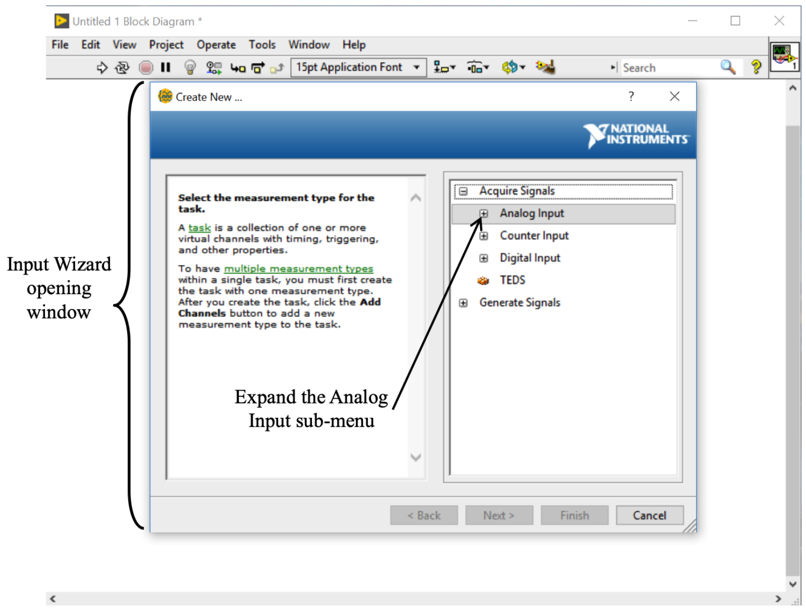
Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure 6.7. ***Acquisition Wizard–Expanding Acquire Signals:*** *Once the initial acquisition window opens, the user selects Acquire Signals to find a device and select channels for data acquisition.*

After expanding the Acquire Signals sub-menu, the user should expand the Analog Input sub-menu (see Figure 6.8).



Select the measurement type for the task.

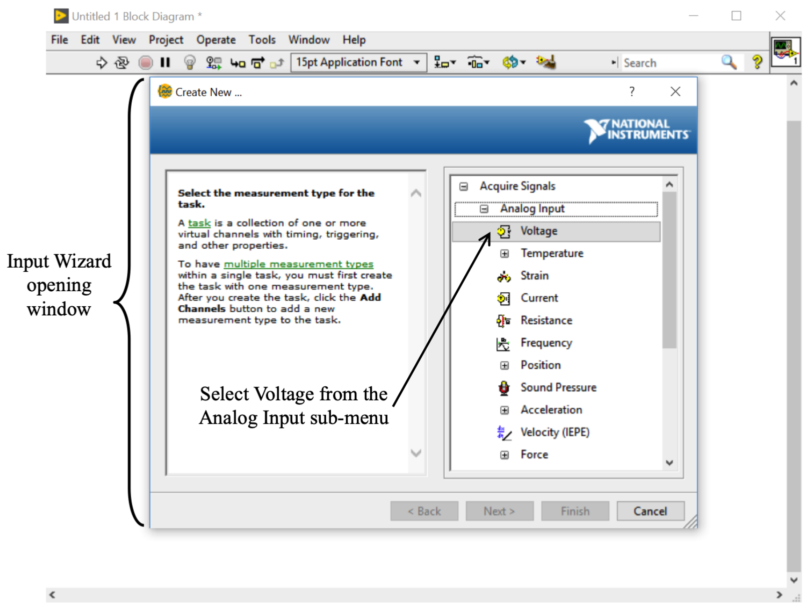
A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure 6.8. ***Acquisition Wizard–Expanding Analog Input:*** *Expand the Analog input option so you can select Voltage as the type of signal.*

Although not always the signal option for any given measurement project, the projects presented in this text are all voltage based, so the user should select Voltage (the first option) under the Analog Input submenu (see Figure 6.9). Indeed, the user can see from

Figure 6.9 that there are a number of other options for analog input. Some of these are also voltages, but they are conditioned voltages, such as the Strain option, or the Temperature option. The LabVIEW software environment is designed to accommodate a number of industry sensor types, and helps the user accommodate those sensor types in terms of bridge constants, or thermocouple type.



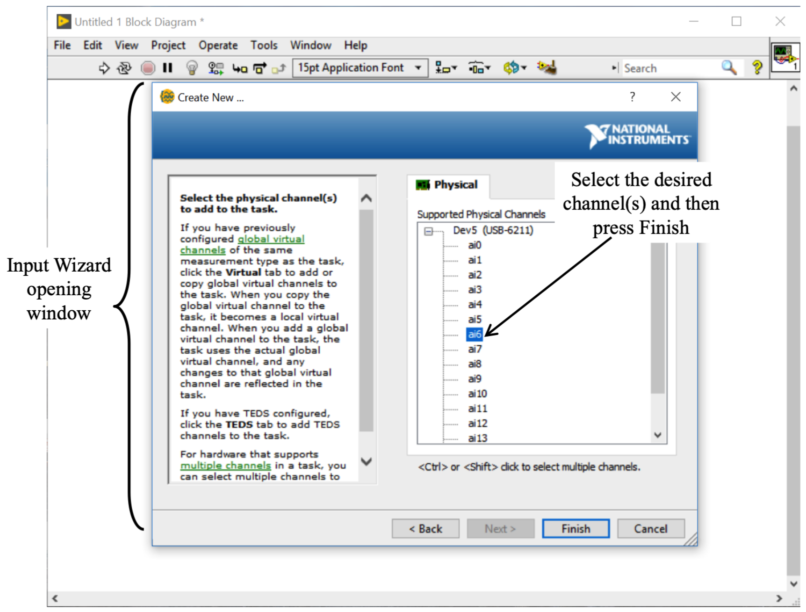
Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure 6.9. ***Choosing Voltage as the Analog Input Type:*** *Although some projects may require a thermistor input, or some other type of input, the accelerometer was chosen to have a simple voltage output that is proportional to the g-load, so the user should choose Voltage as the analog input type.*

The user has finally arrived at the point of selecting the desired input channel(s). For this specific project there is only one input channel and the instructor will inform the student which channel to use. In general, one can select multiple channels in two separate ways. If the multiple channels are sequential, the user can select the first desired channel, then hold the SHIFT key and select the last desired channel. This will highlight all channels in between these two selected ones. If the multiple channels are not sequential, the user can hold down the CTRL key and then individually select each desired channel (see Figure 6.10).


Untitled 1 Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Create New ...
NATIONAL INSTRUMENTS
Input Wizard opening window
Physical
Supported Physical Channels
Dev5 (USB-6211)
ai0
ai1
ai2
ai3
ai4
ai5
ai6
ai7
ai8
ai9
ai10
ai11
ai12
ai13
Select the desired channel(s) and then press Finish
< Back
Next >
Finish
Cancel

Select the physical channel(s) to add to the task.

If you have previously configured global virtual channels of the same measurement type as the task, click the **Virtual** tab to add or copy global virtual channels to the task. When you copy the global virtual channel to the task, it becomes a local virtual channel. When you add a global virtual channel to the task, the task uses the actual global virtual channel, and any changes to that global virtual channel are reflected in the task.

If you have TEDS configured, click the **TEDS** tab to add TEDS channels to the task.

For hardware that supports multiple channels in a task, you can select multiple channels to

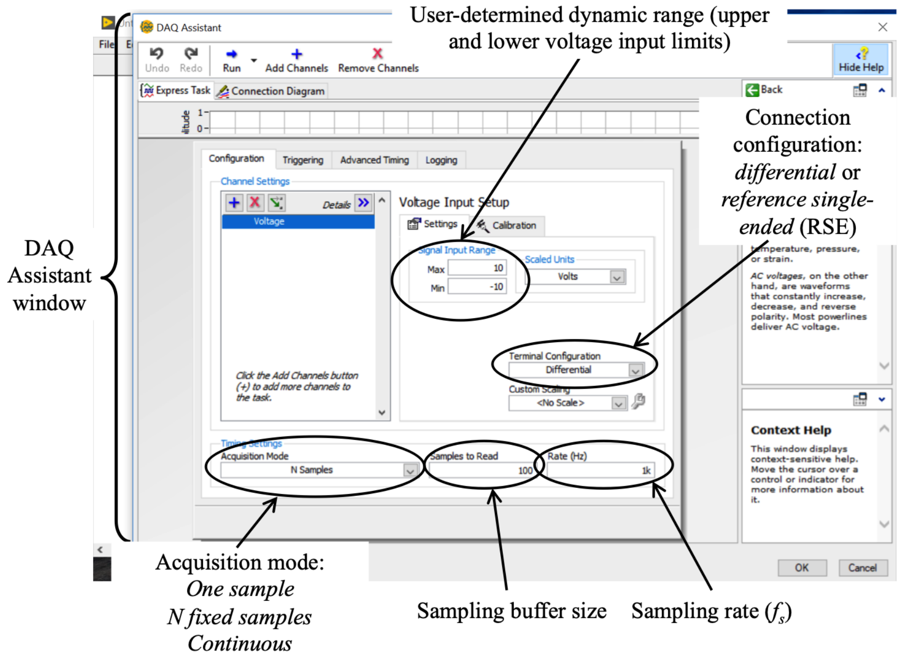
<Ctrl> or <Shift> click to select multiple channels.

Figure 6.10. ***Choosing Individual Voltage Input Channels:*** *The user selects each channel with a mouse click. A single channel can be chosen (as in this project), or multiple channels can be chosen (as described in the text).*

Once the channel(s) have been chosen, the channel configuration window appears, and the user must supply values to configure the channel(s) appropriately (see Figure 6.11). These parameters include:

- **Dynamic range**: both upper and lower voltage limits expected for the incoming signal.
- **Connection configuration**: Differential or Reference Single-ended (RSE) are the two most common.
- **Sampling rate**: This is the sampling frequency the user selects. Care must be taken to make sure the Nyquist criterion ($F_s > 2 \cdot F_{max}$) is achieved.
- **Bin Size (or *number of samples*)**: this is a buffer associated with the actual A/D hardware; if the bin size is set to k samples, then A/D converter converts k samples, collecting them in the buffer (bin), and then empties the buffer to the processor and collects another k samples. The conversion rate is *always* the sampling rate, but the bin size determines how often the buffer of sampled values is sent to the processor. A good starting point is 10-40% of the sample rate. The update rate of the loop affects how plots update from one cycle to the next. Using a Collector block (shown in Figure 6.14) is a good way to keep bin sizes small while still acquiring a sufficient amount of data.

- **Acquisition Mode**: The user must decide to acquire a *single sample*, *N samples*, or *continuous samples*. The *continuous samples* mode requires the process to be placed in a WHILE loop; the option to place the WHILE loop properly is part of the Wizard and the user should ALWAYS accept this option when asked if you will allow it.



Signal Input Range: Max 10, Min -10

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Timing Settings: Acquisition Mode: N Samples; Samples to Read: 100; Rate (Hz): 1k

Click the Add Channels button (+) to add more channels to the task.

temperature, pressure, or strain.

AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

This window displays context-sensitive help. Move the cursor over a control or indicator for more information about it.

Figure 6.11. ***Setting the Acquisition Parameters:*** *The acquisition parameters must be set correctly to ensure data values that are representative of the actual process being measured.*

There are several subtle issues in setting these parameters that can have a significant effect on the results. The bin size, relative to the sample rate, has been discussed, but it mostly affects the visual aspect of the user interface. A more fundamental issue occurs with ongoing data collection and the mode of acquisition. Specifically, if we collect data over many cycles of a WHILE loop, we want to choose *continuous* acquisition. The reader might wonder why we cannot just choose N samples each time through the loop. There are two problems with this:

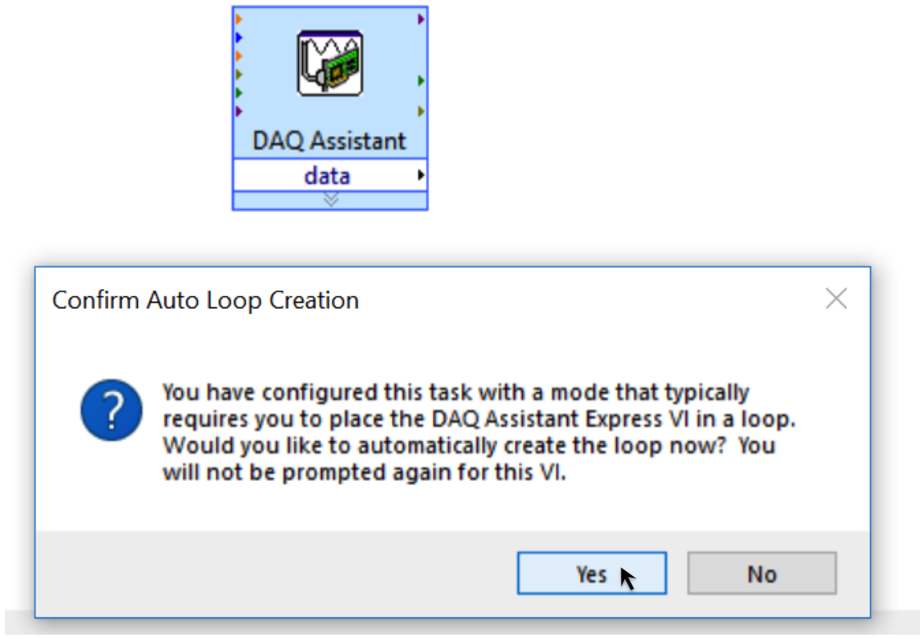
1. **Timing**: The A/D converter converts analog values to digital samples at some sampling rate, f_s. In continuous mode, the bin (buffer) collects k of the digital conversions into the bin, and then passes those samples in bulk to the computer processor (over a USB connection in this case). Because the system is set to a continuous acquisition mode, the A/D converter continues to convert samples at the specified sample rate. Therefore, the acquisition device must ensure that it can empty its buffer (bin) and

return for the next converted sample within the sample period. Note that the sample period is simply the inverse of the sample rate ($T_s = 1/F_s$), so if we set our sample rate to 100 Hz and our bin size to 20 samples, then the A/D converter converts and places the 20^{th} sample into the bin, and from that moment the acquisition device has 0.01 seconds to empty the buffer (i.e. transfer those 20 samples from the bin to the computer's processor) before the A/D converter will try to put a new converted sample into the bin. Furthermore, the remaining code must execute before the buffer gets full again, which in this example is 20/100 = 0.2 seconds. Of course, 0.2 seconds is a *long* time in computer processing! However, the continuous mode guarantees that the data are just that, continuous (and evenly sampled).

If the N samples acquisition mode is chosen, the buffer timing problem does not occur, because the A/D converter stops converting until the next loop cycle begins and the computer initiates another data acquisition of N samples. Once the data are collected the A/D converter is flushed and shut down and the data are transferred to the computer's processor. Therefore, the transfer time and the duration of the remainder of the code are irrelevant. There are no samples being queued up by the converter because it was told to collect N samples and stop. But this means that the interval between the last sample of cycle n and the first sample of cycle $n+1$ is not necessarily the same as the regular interval between the N samples of each loop cycle. Indeed, from a programming perspective, we can't even guarantee that the loop cycle intervals are consistent because the central processor may have other operating system processes running that can take precedence for several cycles at random times. This mode of data acquisition is not good for most ongoing data collection processes.

2. **Filtering**: The discrete filters in LabVIEW are quite intelligent functions. However, if one chooses N samples as the acquisition mode, the filter treats each new set of N samples (each time through the loop) as a new set of data with no knowledge of previous data. This problem is overtly demonstrated in higher-order filters that have transient responses for the first several samples. For example, a lowpass filter may take 50 out of 500 samples to initialize and produce a valid output. This would cause a cyclic transient in the long-term output signal that might be confused for real data. In continuous acquisition mode, the filters are designed to maintain the end state from the previous loop so that the filters carry over with no transients.

After completing all the required fields, the user can press OK and leave the DAQ Assistant. As LabVIEW compiles the code for execution, it will discover that the Continuous Acquisition mode has been selected and pop up a window telling the user that this mode requires a WHILE loop. It gives the user the option of having the DAQ Express block automatically placed in a WHILE loop. You should always accept (click Yes to) this option (see Figure 6.12). LabVIEW sets up the stop control on the user interface, and wires it so that pressing the stop control shuts down the A/D converter before terminating the loop and ending the program see Figure 6.13). This is essential, because it is undesirable to have the A/D converter aborted during operation. Note: The abort button is the red circle next to the run arrow on the menu controls. This button should *never* be pressed when connected to hardware because it leaves the DAQ in an unknown state, with a partially filled buffer, and it may not communicate properly with the computer moving forward.



Confirm Auto Loop Creation

You have configured this task with a mode that typically requires you to place the DAQ Assistant Express VI in a loop. Would you like to automatically create the loop now? You will not be prompted again for this VI.

Figure 6.12. ***User Prompt for WHILE Loop Configuration:*** *When Continuous Acquisition mode is chosen, the user prompt appears asking if the user wishes LabVIEW to go ahead and create the WHILE loop correctly for proper program termination.*

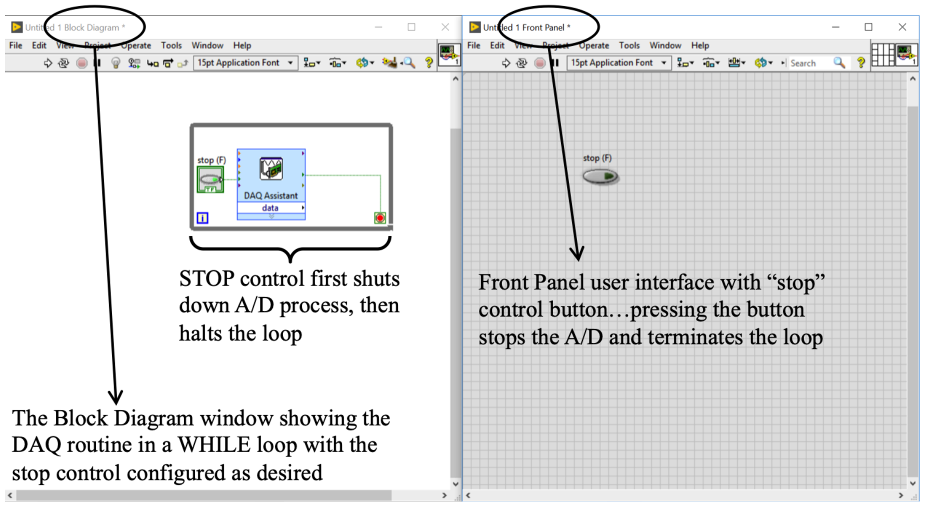

Untitled 1 Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
stop (F)
DAQ Assistant
data
STOP control first shuts down A/D process, then halts the loop
The Block Diagram window showing the DAQ routine in a WHILE loop with the stop control configured as desired
Untitled 1 Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
stop (F)
Front Panel user interface with "stop" control button…pressing the button stops the A/D and terminates the loop

Figure 6.13. ***Correctly Configured WHILE Loop for Continuous Acquisition:*** *Once properly configured, the stop control button on the user interface, when pressed, will first terminate the A/D process in a clean manner–ensuring that the converter is idled and buffer(s) are flushed), before halting the loop itself.*

Once the actual acquisition process is programmed, the user must still make some decision about how to collect the measurements. What do we do with the *k* measurements in the buffer each cycle? For this project, we are going to need to collect data over some time period–it is left for the user to make this engineering decision–and we want that data to be sequential. We might use a FIFO (first-in-first-out) buffer, as this collects the data and places it in the buffer in time-order. In LabVIEW, a FIFO buffer can be implemented using a *Collector*. As with other processes, you can find this by typing it into the Search window (press CTRL and the space bar to bring up a quick search box). The collector can be set to hold *N* samples, so this must be determined based on how many seconds of data one collects, and the sample rate (samples per second). For example, if we set our sample rate to 100 Hz (100 samples per second), our bin size to 50 samples, and our Collector to 3000 samples, then the Collector will store 30 seconds of data at any given time. After the first 30 seconds of data collection, once the buffer (Collector) is initially filled, each new cycle of the loop brings another 50 samples into the buffer. Since the buffer size is set to 3000 samples, each cycle after 30 seconds first drops the 50 *oldest* samples from the Collector (they were the first ones *into* the buffer, so they are the first ones *out of* the buffer), shifts the remaining 2950 samples down, and finally places the newest 50 samples into the empty buffer slots. The buffer will always contain the latest 30 seconds of measurements. Figure 6.14 shows the collector in place, as well as a *graph indicator* so the process can be observed by the user during acquisition. Another reason that the Collector is important to include when acquiring data is that it allows the user to intervene at much smaller intervals and stop the program if something goes wrong. In this example 50 samples are taken at a sampling frequency of 100 Hz so the program can be stopped every half second. If the collector wasn't used, the user would have to wait 30 seconds to see any data and the program could only be stopped by pressing the abort button during these 30 seconds. Seeing no activity for 30 seconds would make many people think the computer and/or DAQ device is locked up and needs to be rebooted.

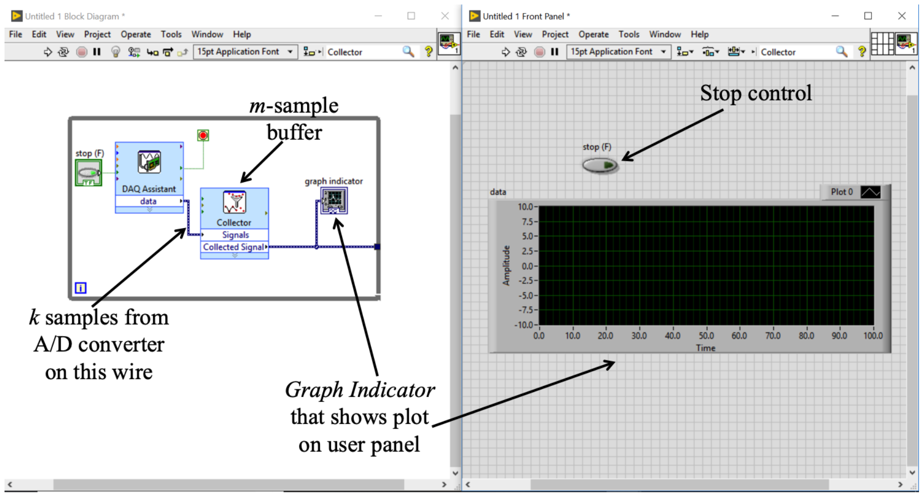


Figure 6.14. ***Data Acquisition Code for Bouncy Toy:*** *The data acquisition code is straightforward once the acquisition parameters have been set in the Wizard. The collector is a buffer and can store any desired length. It is useful to have a graph indicator attached to the collector output to see the overall data during acquisition.*

With the program running and data being put into the Collector, the user can displace the toy by a known amount (recall that the tape measure can be used to know the distance the toy is displaced), and then watch the graph indicator where the data from the Collector is plotted until the buffer has a desired set of data points. By manually exiting the acquisition loop based on inspection of the graph the user can ensure that all transient data points from the startup phase of the process have been flushed out of the Collector. When the data look good, the user can press the Stop Control on the Front Panel to terminate the loop and pass the collector data buffer out of the loop to the analysis portion of the software.

The student is left the task of using the LabVIEW knowledge gained from the Appendix, and other sources, to create the analysis code, but the following things should be considered:

- Why does double integration not provide us the information we desire? Aside from the underlying problems with numerical integration, what would we need to know about our experimental setup to get a unit-correct position equation from the raw accelerometer data?

- By simply exercising the toy, it is obvious that the position equation has complex conjugate roots because the toy has a decaying oscillatory response. Since the position equation is formed of two eigenfunctions (yes, cosine and sine are eigenfunctions, as well as $e^{\alpha t}$), what relationship between eigenfunctions and eigenvalues allows us to glean the position equation parameters from the accelerometer data?

- The student can use the search function in LabVIEW (CTRL+space) to find the VIs needed to analyze the data. For example, there is an Express VI under Signal Analysis for finding frequency content (Spectral Analysis for all frequencies, or Tone Analysis for a limited number of frequencies). The student should consider what points on the decaying oscillation plot contribute to the decaying portion; what approach might be taken to identify these points?

- For each VI the student identifies, it is extremely important for the student to first read the Help information (press CTRL-H while hovering over the VI) and understand precisely the required inputs and outputs, data types, value ranges, etc.

Figure 6.15 provides some hints into what the student is looking for in order to determine the position equation of the toy with respect to time. Hopefully the student has inferred from this exposition that the position equation is the first thing to find, followed by the velocity and acceleration equations. The reader should be considering why we don't just integrate the raw data twice to get velocity (first integration) and position (second integration) directly. The answer lies in our knowledge of the accelerometer and an understanding that numerical integration is fraught with pitfalls that take significant signal processing efforts to mitigate. However, by finding the position equation first (with correct units!) it is much easier for the student to analytically compute the velocity (first derivative) and acceleration (second derivative), with proper units for those phenomena.

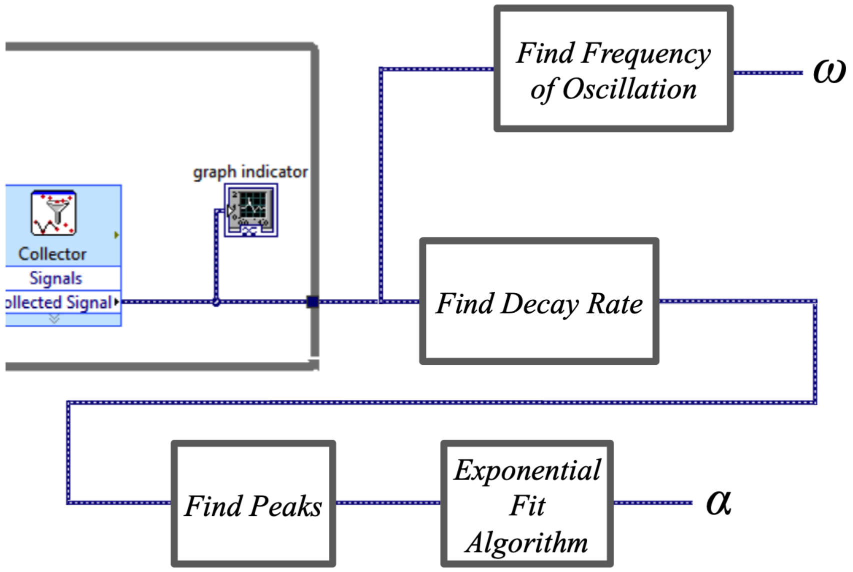


Figure 6.15. ***Data Analysis Goals:*** *After the WHILE loop is exited by the user, the analysis code must solve for oscillation frequency and decay rate in order to determine the position function (finding the decay rate may take several steps, as indicated.*

The following are some of the deliverables that are usually required for the implementation of this project in the Measurement and Automation course at OU:

- A VI that acquires the appropriate data for analysis. The data should be saved to a file so that the analysis can be done away from the lab.
- Either within the same VI, or as a separate VI, the code to analyze the data and find the equations for $x(t)$, $v(t)$, and $a(t)$. This VI needs to be able to run in the following modes: 1) **Live Mode** – connected to the DAQ device to record new data and then immediately analyze it; and 2) **Playback Mode** – analyzes data from a file that was previously recorded.
  - There should be a plot of each equation ($x(t)$, $v(t)$, and $a(t)$), with both the axes labeled correctly for units.
  - There should be a text expression on the front panel near the corresponding graph for each of the motion equations ($x(t)$, $v(t)$, and $a(t)$). Note that while cosine and sine operate on radians, industry invariably reports frequency in units of Hertz (Hz). Your equations should show the frequency in Hertz (recall that $\omega = 2\pi f$). Make sure the expression has units at the end of the time domain mathematical expressions.

In order to implement Playback Mode, the student should consider using the *Write to Measurement File* VI under File IO in the Block Diagram Functions Palette. This VI can be used by taking the data from the Collector and saving it as an LVM file. The data that is stored can be recalled later in Playback mode using the Read from Measurement File VI that is located in the same palette. If data is stored using this method, *make sure to include a time column*!! If you select a setting in the wizard of the Write to Measurement File VI that ignores the time column, the analysis portion of the program will not have enough information about sampling rate to perform an informed analysis.

Chapter 7

This chapter will discuss an audio-based project that has been one of the most popular projects in the Measurement and Automation course at OU for many years. We will begin by discussing some of the background theory needed to do this project.

7.1 The Musical Spectrum

Music, whether vocal, instrumental, modern pop, or classical, provides a rich data source in both dynamics and frequency content. The industry standard range of the audio spectrum is typically considered to be from 20 Hz to 20 KHz. However, it should be noted that this range only applies when we are very young. As we age, and our eardrums stiffen (slightly, but it happens), we begin to lose the ability to hear the upper frequency range. Of course, the 20Hz-20KHz is an idealization, as most of us only perceive up to about 10 KHz.

How do we represent the frequency content of a musical signal? The obvious answer is to compute the Fourier Transform, which plots the magnitude of contribution each frequency makes to the recorded signal. As an example of this, the upper panel of Figure 7.1 shows a plot in the time domain of the first four chords of Beethoven's Fifth Symphony in C minor. The lower panel of the figure shows the *power spectrum* of the same signal. By definition if we say a plot is showing the spectrum, the data are in the frequency domain.

Music is often analyzed in the frequency domain, although it is "massaged" in the time domain. For instance, if we are designing an equalizer that can adjust the emphasis of small frequency bands within the overall spectrum, we design our filters (usually bandpass) using frequency-domain algorithms and theory, and we certainly look at the designed filter behavior in the frequency domain. However, we cannot lose sight of the fact that the filters themselves are acting on the actual time-domain signal. Looking at the spectrum from Figure 7.1, we might design a bandpass filter that passes frequencies from 20 Hz up to 150 Hz. We can plot the *frequency response* of the bandpass filter and overlay it with the music spectrum. In Figure 7.2 the frequency behavior of the music and the bandpass filter are shown in the frequency domain. The bandpass filter's frequency response (see the dashed curve) is actually a second order filter. Note that it is not an ideal filter, which would have an ideal gain of one from 20 Hz to 150 Hz. The student may recall from

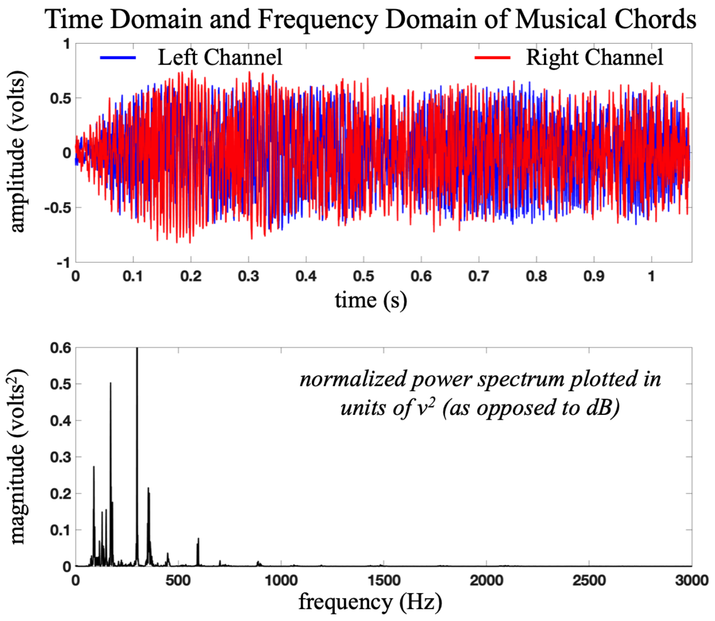


Figure 7.1. ***Music in Time and Frequency Domain:*** *The opening chords of Beethoven's Fifth Symphony (C-minor) are shown in the upper panel. This is a stereo recording so there are two channels, as marked. The lower panel shows the power spectrum of the left channel for the time-domain snippet shown in the upper panel.*

a Signals and Systems course that the *half-power point* is defined as the frequency cutoff. The half-power point occurs at a voltage (or current) gain of 0.707 (times the maximum passband magnitude), and this is shown in the figure.

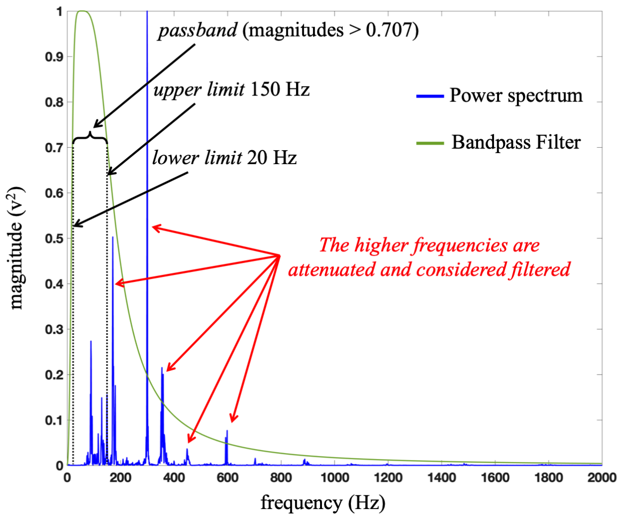


Figure 7.2. ***Bandpass Filter in the Frequency Domain:*** *We examine filter behavior in the frequency domain. The frequency magnitude response of a second-order bandpass filter from 20-150 Hz is shown in green. This is superimposed on the spectrum of the left music channel. In filter parlance, anything below the -3 dB point (a voltage gain of 0.707 in raw numbers) is "filtered." The eliminated (strongly attenuated) frequencies are pointed to with red arrows.*

Before proceeding with a discussion of filters, a brief review of the dB units is in order, as most spectrum plots and frequency response curves are couched in terms of dB. Many filters have transfer functions that cause the plot, with respect to frequency, to drop off exponentially. To provide a more informative graph, the dB (decibel) unit is often employed. But the reader should be aware that the conversion from gain in volts/volt (V/V) or amps/amp (A/A) *is different* than the conversion for power gain (W/W). These are defined as follows:

$$Voltage\ Gain\ (dB) = 20 \cdot \log_{10}\left(\frac{V}{V}\right) \tag{7.1}$$

$$Current\ Gain\ (dB) = 20 \cdot \log_{10}\left(\frac{A}{A}\right) \tag{7.2}$$

$$Power\ Gain\ (dB) = 10 \cdot \log_{10}\left(\frac{W}{W}\right) \tag{7.3}$$

The standard definition of frequency cutoff in a filter is the *half-power* point, i.e. when the power gain is one half of the passband gain. If we assume a passband gain of 1, then the half-power point occurs at a power gain of 0.5 W/W. Plugging this into the equation for power gain dB, we get $10 \cdot \log_{10}(0.5) = -3\ dB$. The voltage (or current) gain, in dB, at this half-power point can be found by setting the dB at -3, and then computing the voltage gain, V/V. In other words, $-3 = 20 \cdot \log_{10}(V/V) \rightarrow V/V = 0.707$. It is important to realize that when you view a *power spectral density* plot (or simply power spectrum), if the y-axis is in dB, the multiplier of 10 was used to generate those dB units from the W/W gain. If the graph is a *magnitude spectrum* plot, then dB units represent the conversion for V/V, using the multiplier of 20. In either case, because of the different conversion formulas, the *half-power point* always occurs at -3 dB.

Signal and filter response graphs are often shown in the frequency domain, which is why the term *spectrum* is used. In signal processing, one can define the power spectral density plot, or simply the power spectrum, as the squared magnitude of the Fourier transform:

$$S_{xx}(j\omega) = X(j\omega) \cdot X^{*}(-j\omega)$$

The reader should bear in mind that a complex number, times its complex conjugate, produces a real value. The magnitude of the Fourier transform is simply the square root of the above term. The discrete version of the power spectrum must be normalized by the number of samples used in the spectrum computation, but the interested reader can refer to any discrete-time textbook for a full description. The message to the student is to understand the difference between the dB conversion for magnitude and power, and understand what the y-axis means when these types of plots are encountered.

We apply filters in the time domain. Although it is possible to take the Fourier Transform of the time-domain signal, zero out undesired frequencies, and then take the Inverse Fourier Transform, this is computationally inefficient in the extreme. If we are filtering in the computer, by definition we are implementing a discrete filter. This is a time-domain process as described by the following difference equation:

$$y[n]+a_1y[n-1]+a_2y[n-2]+\cdots a_py[n-p] \quad = \quad b_0x[n]+b_1x[n-1]+b_2x[n-2]+\cdots+b_mx[n-m]$$

where p and m describe the order of the filter. The interested student might refer to a Digital Signal Processing textbook [e.g. Proakis, Stearns, Bose] for an in-depth understanding of discrete filter design and implementation. Figure 7.3 shows the left channel of music from Figure 7.1 before and after filtering with a second-order, 20-150 Hz bandpass filter. It is clear that the filtered signal (in red) has lost its high-frequency components (sharp changes in the amplitude), compared to the dark gray unfiltered signal.

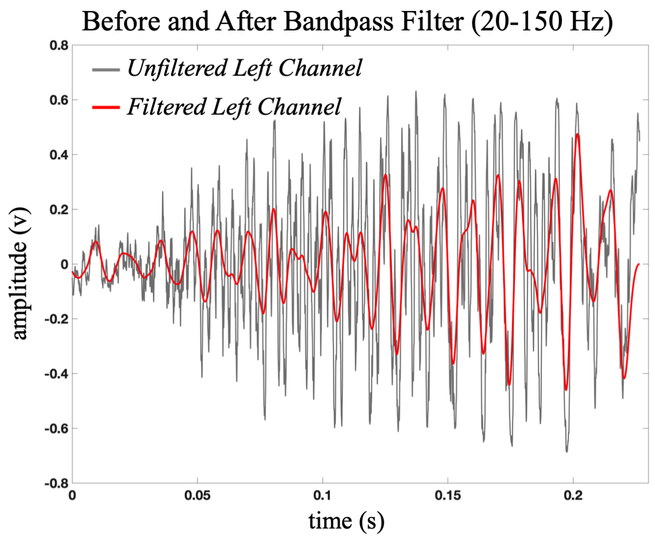


Figure 7.3. ***Unfiltered and Filtered Music Signal in the Time Domain:*** *Filters operate on the signal in the time domain, even though we study their effects in the frequency domain. In this figure the left channel of music is subjected to a 20-150 Hz bandpass filter. The raw signal in dark gray has many abrupt and sharp transitions, while the red (filtered) signal is much smoother, retaining only the lower frequencies.*

Finally, we can observe the expected effect on the music spectrum in the frequency domain. In Figure 7.4 the spectrum of the filtered music is computed. Also shown in the plot is the frequency response of the filter used to modify the music spectrum. Amplitudes were normalized so that the plot is scaled from 0 to 1. Clearly, the spectral peaks in the passband remain, while the spectral peaks outside the passband (see Figure 7.2) have been eliminated.

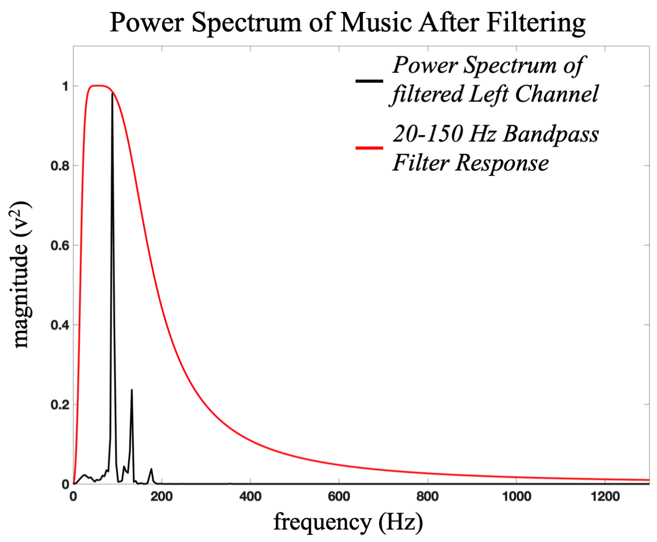


Figure 7.4. ***Music Spectrum After Filtering:*** *The spectrum of the music signal, after bandpass filtering from 20-150 Hz, clearly has fewer spectral peaks. The frequency magnitude response of the bandpass filter used for filtering the music is shown in red (the same response as the one shown in Figure 7.2). It is clear that spectral peaks from the original music spectrum were eliminated outside the passband.*

It is possible to divide music into multiple sub-bands, which is what an equalizer in an audio system does. Furthermore, it is possible to exercise external control of something based on the strength of the signal within a sub-band. In the frequency domain, we know that one measure of the power in the spectrum would be to square the frequency response and sum it (integrate the area under the curve, as it were), or simply integrate the area under the curve if we are using the power spectrum. Of course, since we are operating in the discrete world of a computer processor, we actually sum the values. But after we find the power of a sub-band in the frequency domain, can we infer anything about the time domain, where we prefer to operate? The answer is *yes*! We can invoke Parseval's Theorem. You might want to review your Signals and Systems notes, or if you've had your Calculus series, review what the mathematicians like to refer to as Plancherel's Theorem (the authors are engineers and will stick with Parseval's Theorem). Parseval's theorem states that the signal power in the time domain is proportional to the power in the frequency domain. Mathematically (and in discrete terms)

$$\sum_{n=0}^{N-1} |x[n]|^2 \quad = \quad \frac{1}{N}\sum_{k=0}^{N-1} |X[k]|^2$$

recalling that the square on the right-hand side of the equation requires us to multiply the complex-valued $X[k]$ coefficients times their complex conjugates, or $|X[k]|^2 = X[k] \cdot X^*[k]$. Given Parseval's Theorem, it is clear that we can use a bandpass filter to access a specific sub-band of music, then measure the power in the time domain and determine if the signal strength is above or below some threshold. Why don't we just compute the mean value of the signal? Because the signal is made up of many different sinusoids, and we know that the average value of sine or cosine is zero! So, even if we add up a bunch of different frequencies, if each one separately has a mean of zero, then the whole thing will have a mean of zero. A much better measure of signal strength (in either domain) is the root-mean-square (RMS), which is poorly named since we perform the operations in the reverse order. The following is the three-step process of taking the root-mean-square (or should we say square-mean-root) of a signal.

- **S Step**: For some signal vector of k data points, we first square $x[i]$, for $i = 0, 1, 2, ..., k$
- **M Step**: We then compute the mean of the k points in the vector of squared values
- **R Step**: Finally, we take the square root of the mean value.

The fundamental concept here is that we can reduce the entire data set (say the k samples in our bin size, for example!) to a single scalar parameter that estimates signal strength.

7.2 Project

The reader may wonder what might be controlled from music frequency analysis. Well, this is how *dancing light shows* work. The different colored lights flash on and off based on the character of the music. The goal of this project is to create a VI that brings in the stereo (two) channels of music and then controls four colored lights using digital outputs (on-off). The user interface for this project is important, and there are several requirements to make it user-friendly, and interesting.

The following are some of the requirements for the implementation of this project in the Measurement and Automation course at OU.

First, the specifications of the project are:

- The data will be in RSE configuration and available on two channels, specified by the instructor.
- The front panel (user interface) must include (at a minimum) the following:
  - A time domain plot of both channels (two channels being plotted on a single graph)
  - A *single* Power Spectrum plotted with a linear y-axis (one plot, so the student must decide how to reduce two channels to a single spectrum)
  - A Lissajous plot (discussed later)
  - Four sensitivity knobs, one for each light color (needed because music character changes from one song to another and you want to make sure the lights still dance).
- The dancing light behavior must be independent of the stereo volume level.
- Color coordinated front panel indicators that indicate the state of each light (the digital control for the lights is tied to the same-colored LEDs on the student's protoboard at the local workstation, so all testing can be done at the local workstation before the instructor connects the room's colored floodlights for demonstration).
- All graphs should clear when the VI is turned off, and all lights should be turned off.

The student has plenty of things to consider from an engineering perspective before simply jumping into this project. First, we must determine the data acquisition parameters. How fast should you sample the incoming data streams? How big should the bin size be to ensure that the graph windows on the user screen update in a smooth fashion? What should you set the dynamic range to? Continuous or n-sample acquisition mode? Differential or reference-single-ended connection configuration? Is each channel zero-mean? If not, what can be done to mitigate this issue? There are four lights to control. How can the music be analyzed to create four different decision gates through which we can control the lights? **What the heck *IS* a Lissajous plot**? How do we make the light control insensitive to stereo volume?

Sampling rate can be assessed by considering the standard for audio frequency range, or perhaps by investigating the sampling rate at which a music CD is sampled. Bin size should be kept to a size that allows the front panel graphs to update smoothly. But if the bin size is too small, the A/D converter will be making an excessive number of trips to pass data to the processor, and the program will fail. Given a certain sampling rate, a bin size that keeps the loop iteration time to about 1/10th of a second or less is reasonable. Dynamic range can be assessed by monitoring the channels in the DAQ Assistant wizard wizard (click on the run button at the top to see the data as you are setting up the DAQ settings). The acquisition mode is something you can experiment with, as both continuous and N-Sample modes can provide workable solutions. Any offset can be removed after acquisition by either subtracting the offset, or highpass filtering. The latter seems like a reasonable choice since the low end of the audio band is considered to be 20 Hz. Music analysis is detailed in the first section of this chapter, and the student should carefully consider bandpass filtering and RMS computations. However, the student is left to determine whether to divide the spectrum evenly or asymmetrically.

A Lissajous plot is a parametric plot of two time domain signals against each other. In other words, if we have a signal, $x(t)$ and another signal, $y(t)$, we can plot $x(t_k)$ on the x-axis and the corresponding $y(t_k)$ on the y-axis. Note that both $x(t)$ and $y(t)$ are functions of time, but we must time-align the values and then plot y versus x. What two signals do we have available?

Finally, digital control can be found in the Express palette under Output (see Figure 7.5):

Express→Output→Generate Signals→DAQ Assist

Once the DAQ Assist is selected and the pop-up window is presented, the user should choose (see Figure 7.6):

Generate Output→Digital Output→Line Output

and select all four lines of Port 1 (it is easier to choose all four lines as separate lines so that the programmatic control can operate on a per-light basis). Once the DAQ Assistant wizard appears the only thing that needs to be checked is that the Generation Mode is set to 1 sample (On Demand).

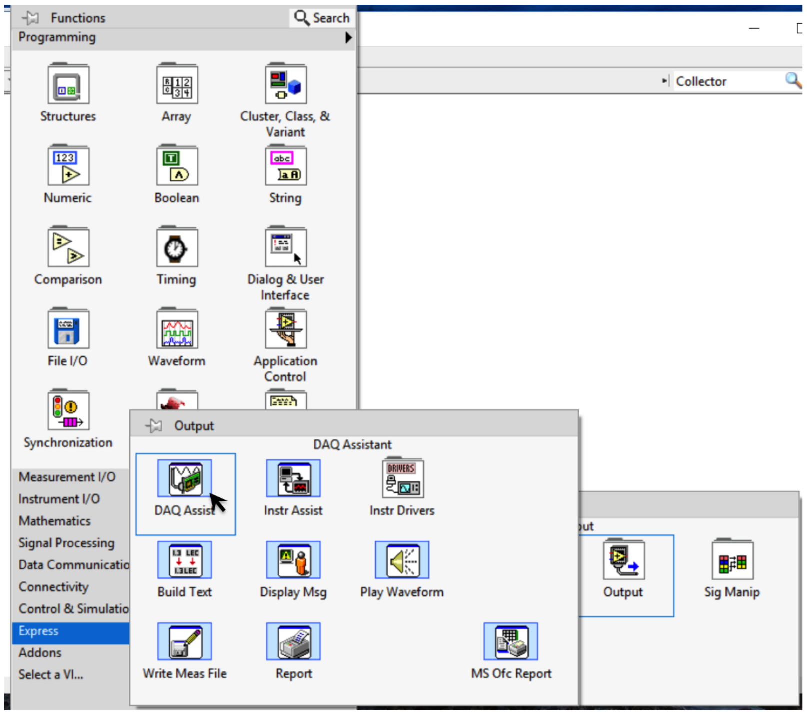
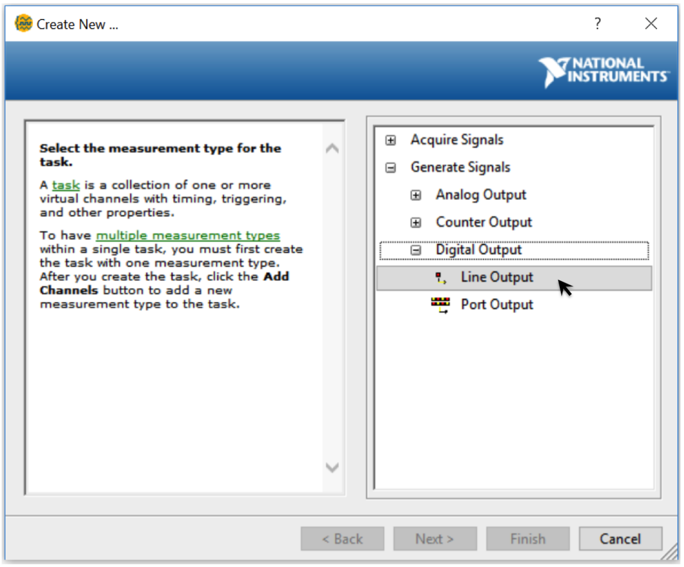


Figure 7.5. ***Starting Digital Output Control:*** *The first steps to instantiating digital output line control.*



Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure 7.6. ***Selecting Digital Output Lines:*** *The user should choose the Line Output option and select all four lines, Line0-Line3, from Port1. This makes the input to the control a simple 4-element boolean vector with one boolean value for each light.*

When wiring the Booleans to the Digital Output DAQ assistant VI that was set up in Figure 7.6 you must add a build array block between the Booleans and the input terminal of the DAQ Assistant (see Figure 7.7 for details). Also, make sure that you have an extra

DAQ assistant that is called when the program stops that turns off all of the LEDs. This could be accomplished by copying the code in Figure 7.7 but replacing the 4 Boolean controls with False constants. If you don't turn all of the LEDs off by sending a Boolean false to them they will stay on until the computer is turned off.

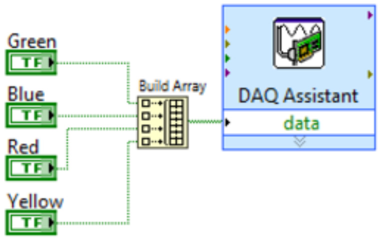


Figure 7.7. ***Wiring the Boolean values to the DAQ Assistant:*** *The Build Array VI must be added for the Boolean values to be received by the DAQ assistant.*

The deliverable for this project is a VI. The user interface itself should have the appearance of a stereo and have the required items included that were previously mentioned. The items above are to be considered a minimum set of elements. The student is encouraged to think of other elements that could provide useful information and appealing visual effects. Make sure that the focus is on making the display look good and also provide balance and symmetry. Indeed, a key feature of good user interfaces is the balance and symmetry of displays, control knobs, and any other elements (e.g. power button). If you simply accept the default configuration of each element in LabVIEW, the interface will be functional, but get low marks. Because engineers need to do some user-interface programming, it is important to get some experience in this area. LabVIEW provides a reasonable environment for helping develop these skills, because much of the detail work is well-defined and easily edited.

Chapter 8

This chapter will discuss the most complicated of the three provided in this text. It involves a classical communications system and relies on many of the things that were learned in the previous two projects.

8.1 Dual-Tone Multi-Frequency (DTMF) Signaling

Using buttons that you press to dial a telephone number was introduced in the mid-1960s by Bell Telephone. Prior to the use of so-called "touch tone" phones the standard dialing method was the *rotary dial*. The rotary dial was rotated to the desired number, and as it relaxed to the un-dialed position, N pulses were generated. The telephone switching equipment counted those pulses to determine the desired digit. Touch tone technology was introduced as a replacement technology because it could operate effectively on lower signal strengths, was more robust, and more user-friendly (i.e. quicker to dial the numbers). As the engineers at Bell Telephone were designing the touch tone system they worked diligently to develop a signaling scheme that was robust and immune to confusion between symbols. The dual-tone multi-frequency standard was the result of their work. This standard uses two sets of frequencies, four high frequencies and four low frequencies in combinations of two (one low and one high). This provides up to 2^4 (or 16) separate symbols, which was enough to handle the 12 buttons that were on the standard touch tone phones of the late 20^{th} century. The specific telephone-related DTMF table is shown in Table 8.1. Although modern cell phones do not use DTMF to encode the desired number being dialed, they still incorporate DTMF on the virtual keypad so that users can navigate automated answering systems that request number presses for menu selection.

Table 8.1. *Telephone DTMF Frequency Assignments*

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

For example, the number "4" is pressed and the sound recorded in the presence of additive noise. This is shown in the upper panel of Figure 8.1. In this case, the sound was sampled at 8 KHz, meaning that the highest frequency that can be represented in the Discrete Fourier Transform is 4 KHz. The frequency magnitude response is shown in the lower panel of Figure 8.1. The magnitude of the Fourier Transform is simply the square root of the complex magnitude squared. In this case, the peaks are identified at indices that correspond to the frequencies 771 Hz and 1211 Hz. These are not specifically in the table, but we know that the closest values are 770 Hz and 1209 Hz, and this represents the number "4." Because of the sample rate, it is not likely that the computer's processor can identify the frequencies precisely. However, this is one of the primary reasons for developing the set of frequencies chosen by Bell Labs.

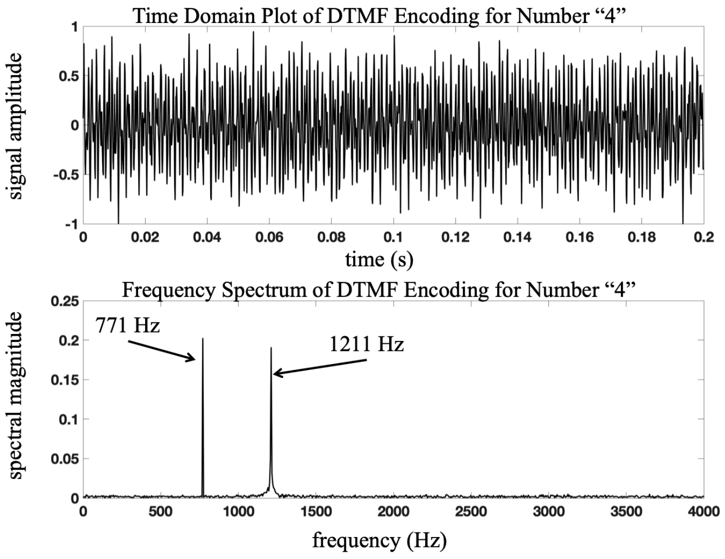


Figure 8.1. ***Time Domain and Frequency Domain for DTMF Encoding of "4":*** *When the number "4" is encoded using DTMF, 770 Hz and 1209 Hz sinusoids combine with the ubiquitous additive noise to produce the voltage signal shown in the upper panel. Sampling 0.2 seconds of this signal at 8000 Hz and computing the spectrum (via FFT), produces the spectrum shown in the lower panel. Searching the frequency magnitude vector shows peaks at 771 Hz and 1211 Hz.*

The reader should consider whether spectral analysis is the only way to determine the two frequencies and, thus, the intended symbol (number). Computationally, using this method involves computing the FFT of some short segment of sampled data, implementing some peak detection algorithm to locate the indices of the two peaks, locating the two corresponding frequencies based on the vector indices of the peaks, and then finally coerce the detected frequency value to the closest known DTMF value.

Filtering is an alternate method that could be used to determine which number is represented in the signal. One might consider a parallel set of narrow, bandpass filters. If the signal from Figure 8.1 is passed through a bandpass filter centered on 770 Hz, and another bandpass filter centered at 697 Hz, the output of the 770 Hz bandpass filter will have more signal power than the output of the 697 Hz bandpass filter. So in this case, the signal would be passed through seven parallel bandpass filters, the signal power at the output of the filters would be computed, and then the two maximum signal powers would indicate the two frequencies. It may seem a bit excessive using seven parallel bandpass filters, but the difficult-to-implement coercion step is removed by this method. As an example of the process, there would be four narrow band filters used to identify the low frequency (from the possibilities of: 697 Hz, 770 Hz, 852 Hz, and 941 Hz). The DTMF signal for the 4 button is passed into all four narrow band filters, as shown in Figure 8.2, in parallel. Since the low frequency of the digit 4 is 770 Hz, the power (estimated using the RMS measure) in the 697 Hz filter, the 852 Hz filter, and the 941 Hz filter will all be significantly lower than the power in the 770 Hz filter. A similar approach can be used to determine the high frequency.

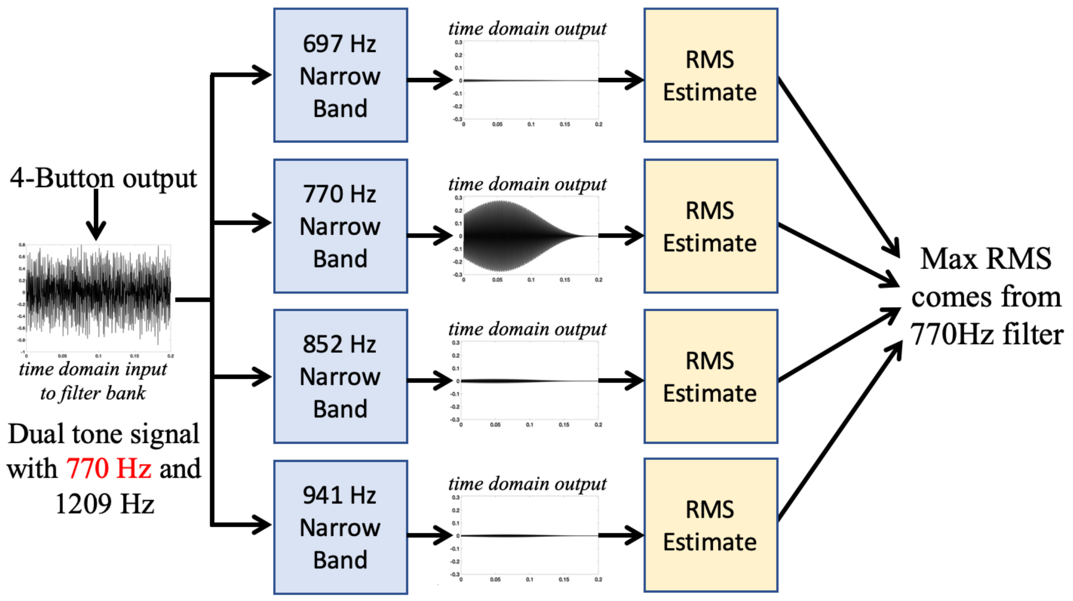


Figure 8.2. ***Frequency Filter Bank to Determine Low Frequency:*** *Using parallel narrow band filters can help identify which low frequency is present. The power of the remaining signal after filtering will be much greater from the narrow band filter that is associated with the tone that is present.*

Another possible detection algorithm implements the *correlation* process. In this algorithm, there is a *gold standard* version of each symbol (12 different gold standards for a DTMF symbol), and the incoming signal is cross-correlated with each of the twelve gold standards. The cross-correlation that produces the highest value corresponds to the encoded symbol. For example, in the upper panel of Figure 8.3 the ideal dual-sinusoid signal for the symbol "4" is cross-correlated with the recorded encoding of the symbol "2."

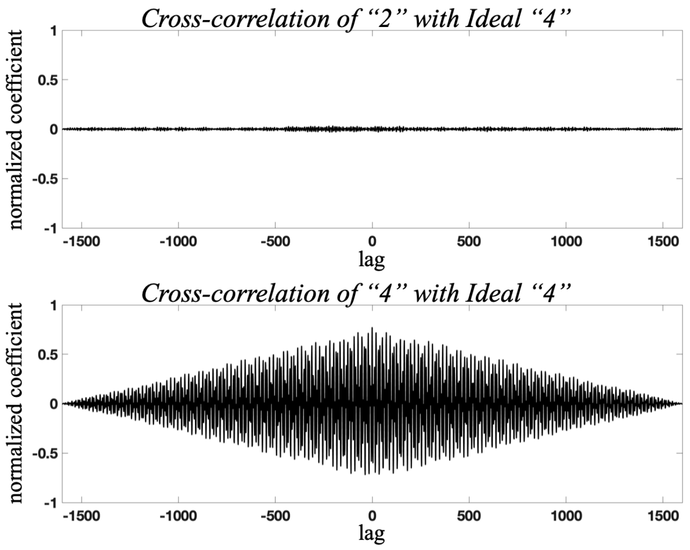


Figure 8.3. ***Cross-Correlation with Gold Standard to Identify Symbol:*** *In the upper panel the incoming signal encodes the symbol 2. It is correlated with the ideal representation of the symbol 4, and the resulting correlation is near 0. In the lower panel, the incoming signal encodes the symbol 4, and is correlated to the ideal representation of the symbol 4. The correlation is significantly higher (0.77). Correlation is not perfect due to the additive noise in the signal.*

The correlation coefficients are normalized (so perfect correlation is one), and it is clear that there is no correlation. In the lower panel of Figure 8.3 the ideal dual-sinusoid for the symbol "4" is cross-correlated with the signal from Figure 8.1. At the *lag* 0 point, the correlation coefficient is about 0.77. The signal-to-noise ratio for both signals is -2 dB. Cross-correlation has the advantage that one does not have to separate low and high frequencies, nor does one have to execute the cross-correlation across the entire signal (meaning a reduced number of computations for each cross-correlation). Of course, there is the memory needed to store 12 separate gold-standard signals and the 12 partial correlations to be executed in order to detect the encoded symbol.

8.2 Project

This project provides an analog, DTMF-encoded, seven-digit telephone number that the student must be able to create and decode. An example of the transmitted signal is shown in Figure 8.4. The higher amplitude sections are the phone numbers. There are smaller amplitude sections between each of the numbers and a longer section of smaller amplitude before the first number is sent.

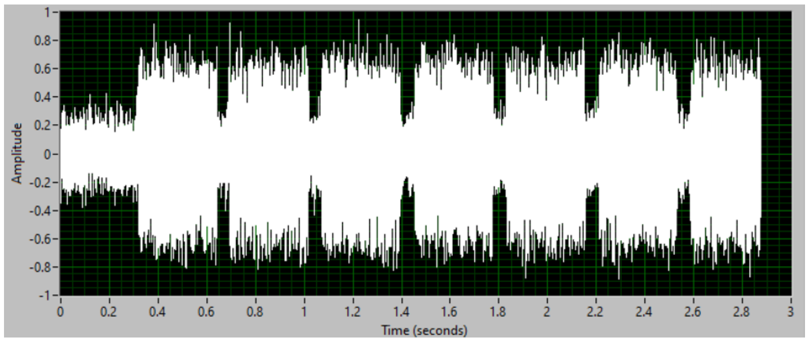


Figure 8.4. ***Example of Transmitted DTMF Signal:*** *The reader should observe that the periods when no button is being pressed still have some level of additive noise. However, the button-press occurrences clearly have greater power and can easily be distinguished.*

The following are some of the requirements for the implementation of this project in the Measurement and Automation course at OU.

- Using the Sine Waveform VI that is in the Signal processing–Wfm Generation Palette or the Simulate Signal Express VI create a VI that can add two sinusoids together and build a 7-digit telephone number. An example of the structure of the 7-digit phone number is shown in Figure 8.4. Approximate the amplitudes of the numbers and spaces so that it looks similar to the number in Figure 8.4. Also, set the length of samples of each number and spaces between each number so that it looks similar to Figure 8.4. The beginning dead zone should be a random length of time that ranges between 0.1 and 0.5 seconds. Since in an actual system you wouldn't know how long it would take for someone to start dialing this is more realistic. Once the person begins to dial the amount of time the buttons are pressed and the time the buttons are released are likely similar in length so you are allowed to set them all the same length and use the approximate length shown in Figure 8.4.

- Have a program that can decode a live phone DTMF signal like the one shown in Figure 8.4 that is sent through the DAQ and also decode a DTMF signal from a file. Have a switch that toggles between live and from a file. The phone number needs to be displayed as a string with a – sign between the 3rd and 4th digit (i.e. 555-1234)

- For this project (both the DTMF creation VI and the decoding VI), the students are required to use a State Machine for this project. The task could be accomplished by dividing the detection steps into distinct processes and developing a state for each process.

In general, the following items should be considered by the student to complete this project:

- Sampling rate, bin size, dynamic range, and channel configuration need to be set appropriately.
- Discriminate between button presses and the time between button presses
- Determine what algorithm (method) to use to decode each button press (correlation, bandpass filters, FFT peak detection and coercion, etc). Multiple algorithms could also be used to minimize the likelihood of an error by looking for agreement between methods.
- Determine a method to decode a button press *only once* and then ignore the button press until it is released.
- Develop a user interface that clearly informs the user of the decoded telephone number

One way to know you have moved from a space (or dead zone) to a number is by breaking up the signal into smaller arrays and looking for a change in RMS. Make sure you don't have too many points in each array as you are searching for the beginning or end of a number. Once you know where the number begins you should take just enough data points from the signal to accurately represent the number so that your algorithm can determine what number it is. Then, you need to start looking for an RMS drop that will signify the next space. Then, the cycle repeats.

References

[1] A. J. Wheeler and A. Ganji, *Introduction to Engineering Experimentation*. Upper Saddle River, NJ: Pearson Education, Inc., 2004.

[2] A. Oppenheim, A. Willsky, and S. Nawab, *Signals and Systems*. Upper Saddle River, NJ: Prentice Hall, Inc., 1983.

[3] N. Bose, *Digital Filters: Theory and Applications*. New York, NY: Elsevier Science Publishing, Inc., 1985.

[4] M. Hayes, *Statistical Digital Signal Processing and Modeling*. New York, NY: John Wiley & Sons, Inc., 1996.

[5] J. Proakis and D. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications, 1st Ed.* New York, NY: Macmillan, 1992.

[6] D. Halliday and R. Resnick, *Fundamentals of Physics, 3rd Extended Ed.* New York, NY: John Wiley & Sons, Inc., 1988.

[7] A. Sedra and K. Smith, *Microelectronic Circuits, 3rd Ed.* New York, NY: Oxford University Press, 1991.

[8] C. Davis, *Electromechanical Systems 1st Ed.* Norman, OK: University of Oklahoma Libraries, 2018.

[9] J. Dyer, "Course notes developed for teaching at the University of Oklahoma from 2008-2018," 2018.

Appendix A

A.1 Graphical Programming Environments

Although the point of this text is not to make the reader a LabVIEW programming expert, one must develop some knowledge of this programming environment in order to perform the projects in this book. LabVIEW is a *graphical programming environment*, which means that the user selects icons from a function palette, places them on the workspace, and connects them in some workflow as determined by the user. The LabVIEW programming environment was developed by National Instruments, and more thorough familiarization with this package can be found at http://www.ni.com/getting-started/labview-basics/. A simple example is shown in Figure A.1. In this case, the icon for addition is placed on the block diagram page, two numbers are placed on the block diagram, and wired two the addition icon as inputs, and then an *indicator* is connected to the output. The indicator shows up on the user interface (front panel).

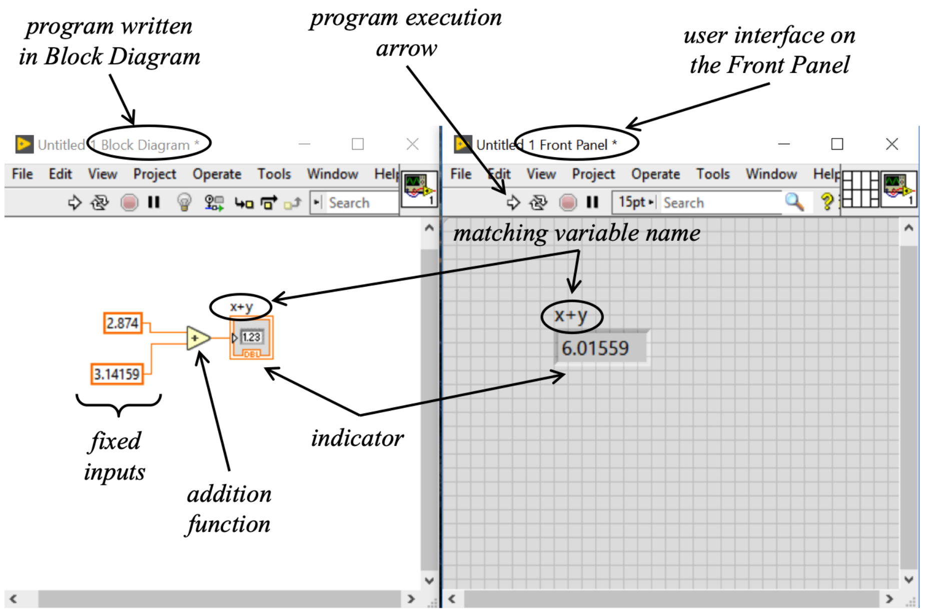


Figure A.1. ***Simple LabVIEW Program (VI):*** *LabVIEW programming is graphical. The programmer places* functions, *like the addition icon, on the Block Diagram, and then uses* indicators *to allow the user to see the results on the Front Panel. The inputs to the adder are hard-wired in this example, but may be controlled by the user through the Front Panel (as shown below).*

Of particular interest in this type of programming is the concept of *data flow*. Data flow simply indicates that, when an element has all of its inputs available, it can execute. If two or more of these elements all have their inputs available, then the compiler determines which element operates first, second, third, etc. This is unlike textual coding, which executes the lines of code sequentially. In order to control the order of operation in the LabVIEW programming environment, one must control the flow of data (inputs) to the program elements. Each of the program elements is called a *function*. So, the addition icon is a function; the equality comparison icon is a function. The basic programming functions are grouped by broad subject areas: Structures; Arrays; Clusters, Classes, & Variants; Numerics; Booleans; Strings; Comparisons; Timing; Dialog & User Interface; File I/O; Waveform; Application Control; Synchronization; Graphics & Sound; Report Generation. Added toolboxes provide extended capabilities in signal processing, instrument Input-output with many devices, controls, vision, mathematics, and many other options.

A.2 LabVIEW Environment Basics

A LabVIEW program is called a VI, which stands for *Virtual Instrument*. When one starts a new VI, two windows become available. The window with the gray background and grid pattern is the Front Panel. The Front Panel is the user interface where users can input information into the program, and read results from the program (textually and graphically). The second window is the Block Diagram. The Block Diagram contains the fundamental program–all the user-placed functions wired together to execute the desired process. Each window has a palette associated with it. The two windows are shown in Figure A.1. The user has a choice to make when writing a LabVIEW program. One can either being by building the Front Panel (user interface), and then add functionality in the Block Diagram, or one can start with the Block Diagram and program the functionality, followed by developing a nice user interface on the Front Panel. In this text, we will start with the Block Diagram.

The instantiation of a new LabVIEW VI has changed across several versions of LabVIEW, but is intuitive, so this text will not discuss installing or starting LabVIEW. Assuming the user has started a new VI, the Block Diagram has a basic menu as shown in Figure A.2. The white, block arrow is the execution icon. As long as the arrow is a block, white arrow, the program is acceptably wired and will execute. If the arrow becomes broken and grayed-out (see Figure A.3), the program will not execute because something is not wired correctly. Clicking on the broken, gray arrow will bring up a window that displays a list of wiring problems. However, correct wiring so that it can compile does not guarantee the programmer's logic will cause it to execute exactly as intended!

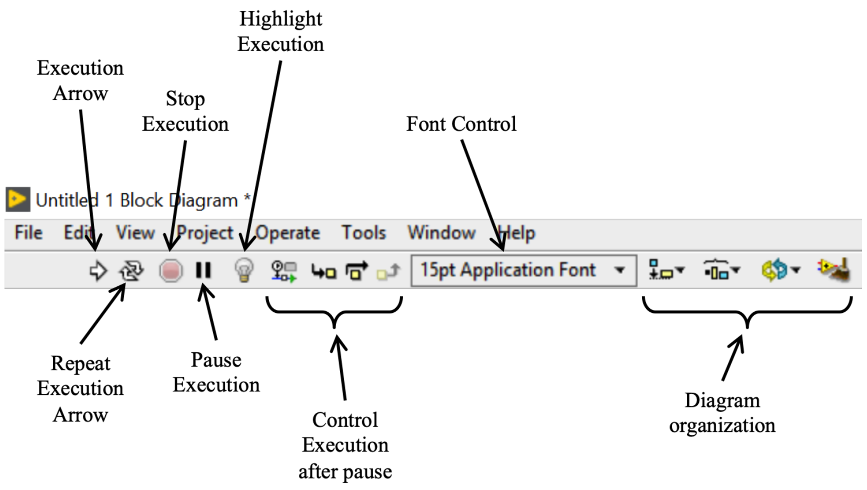


Figure A.2. ***Block Diagram Menu Bar:*** *The Block Diagram Menu bar has several icons for developing and executing a VI.*

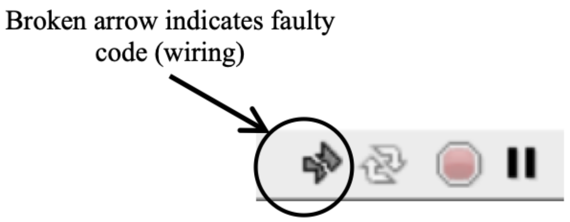


Figure A.3. ***Broken Execution Arrow:*** *When the execution arrow is broken and gray, there is a wiring problem in the Block Diagram. This might be a wire left off, or an input unwired, or any number of issues. Clicking on the broken arrow brings up a list of detected faults.*

The double-arrow icon executes a VI iteratively. Although this might be useful for testing small pieces of code, any program that needs to execute things iteratively should simply be programmed properly in a loop. The STOP sign becomes available when a VI is executing. This icon ***aborts*** a program during execution. This is *not* a good choice to stop programs, in general. Proper controls should always be put into place to halt execution. When the STOP icon is pressed and the program aborts, any data acquisition hardware is abandoned in an unknown state, buffers are abandoned, and the code is terminated in a very ungraceful manner that can make its initial state during the next execution unstable.

The parallel black bars are simply a pause tool during execution. The lightbulb next the pause icon allows one to highlight program execution. This can be a useful debugging tool to understand the data flow of a program. The next several icons are all associated with debugging. If a program is paused during execution, these buttons help the programmer step through the unexecuted code to slowly trace the programming action.

The next icon allows the user to control fonts for display and controls. The default font and size is displayed, but the down arrow indicates an entire sub-menu where one can adjust justification, style (bold, italics, underline), size, color, or the font itself. The remaining icons represent tools to help arrange the graphical elements of the block diagram. One can vertically or horizontally align functions, distribute them evenly, and the last icon (with the small broom) is a universal "clean diagram" tool that uses a built-in algorithm to reduce and arrange wiring and size.

The Block Diagram's palette is accessed by right-clicking the mouse while the Block Diagram is the active window. The palette opens and has the Functions group available, as well as other sub-palettes listed below (see Figure A.4).

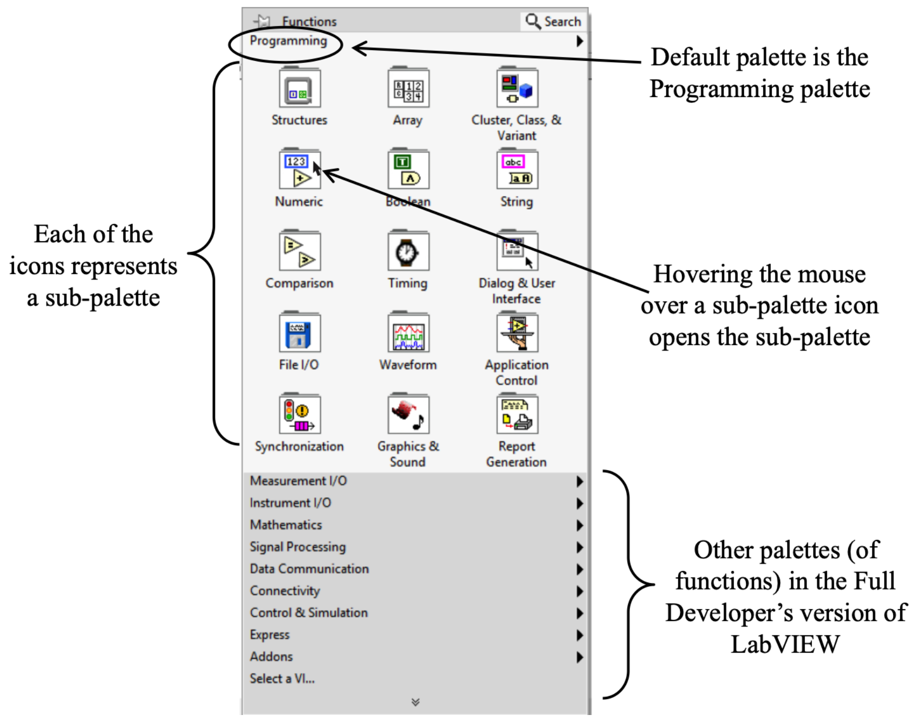


Figure A.4. ***Block Diagram Functions Palette:*** *Most programming functions are accessed from the Block Diagram Functions palette.*

The user can access any of the sub-palettes in the main palette (either the Functions palette in the Block Diagram or the Controls palette in the Front Panel) by hovering the mouse over the sub-palette icon. This brings up the sub-palette menu. For example, the Array sub-palette in the Functions palette of the Block Diagram is shown in Figure A.5. Each of the sub-palette icons of the Block Diagram has a number of functions with which the reader should familiarize himself.

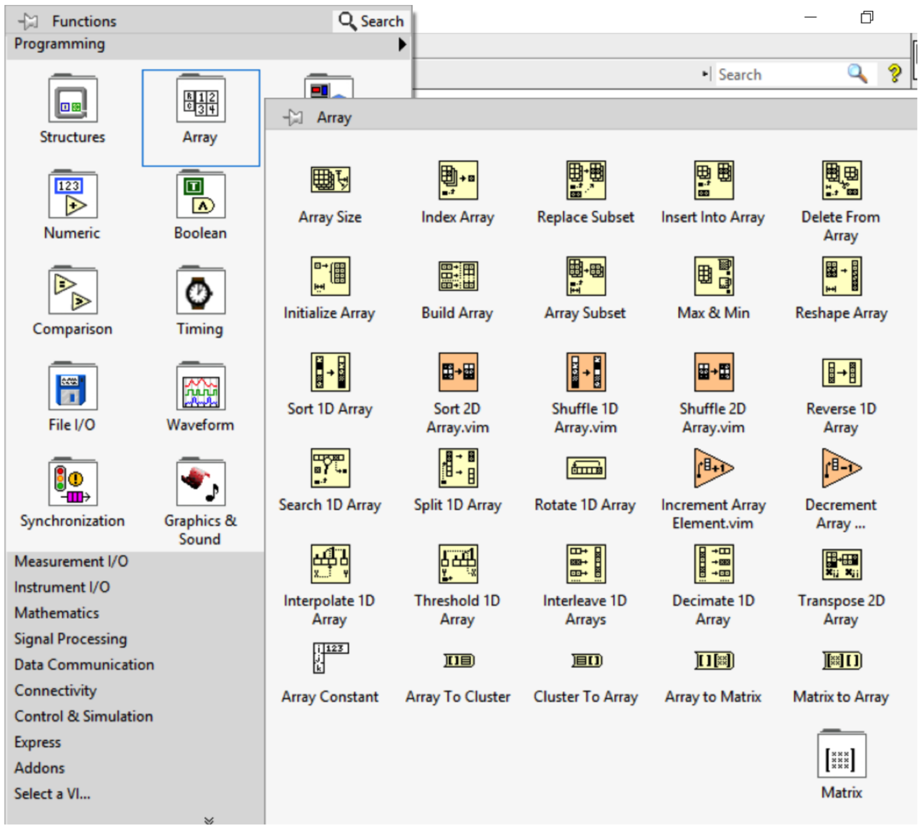


Figure A.5. ***The Array Sub-Palette:*** *The Array sub-palette has a number of functions associated with creating and manipulating arrays.*

Before digging into the basics of programming in the Block Diagram, the reader should review the basic data types. Although not an exhaustive list, the fundamental data types of interest in this text are: real numbers (floats), integers, Booleans (T/F, 1/0), and strings. Data structures of interest include arrays and clusters. Real numbers are represented in orange, and can be Fixed Point (FXP), Single (SGL), Double (DBL), and Extended (EXT). The default precision for a real is DBL. The integers are represented in blue, and can be I8 (8-bit), I16 (16-bit), I32 (32-bit), or I64 (64-bit). The default integer precision is I32, though some functions require input(s) of different precision. Boolean data are represented in green and do not have a precision associated with them. Finally, the string type is represented in pink. Figure A.6 shows each of these placed on the Block Diagram with an indicator attached. It should be noted that one can determine the precision of the numeric data types from their instantiation on the Block Diagram. However, it is very small and difficult to read. The indicator tells us the data type, but the best way to *know* is to simply right-click the numeric value and select Representation from the pop-up menu.

LabVIEW Data Types

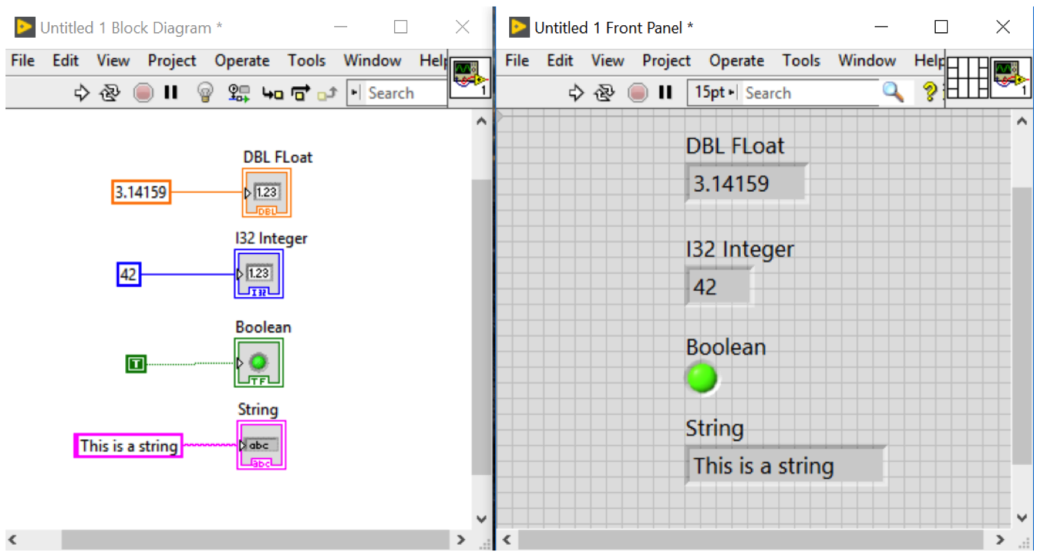


Figure A.6. ***LabVIEW Data Types:*** *The data types in LabVIEW are color-coded. However, with numeric data, the precision of the variable is not always obvious. The user can right-click the variable on the block diagram and select Representation to determine the precision.*

A.2.1 Block Diagram Programming–First Program

A great deal of any program is accomplished from the Block Diagram. The right-click always brings up the palette containing all the functions. In general, each function has its input wiring terminals on the left side of the icon, and the output wiring terminals on the right side of the icon. The functions are divided into sub-palettes. The sub-palettes each contain a number of low-level functions; for example in the Numeric sub-palette, one can find the addition function, the subtraction function, multiply, divide, increment, square, square root, and many others. In the Array sub-palette, the programmer can initialize a new array, check array size, replace an array subset, insert into or remove from an array, etc. Each of the programming sub-palettes are shown in the following figures.

A key feature of LabVIEW is that control and indicator elements can be instantiated from the Block Diagram, even though they are inherently user interface components. For example, we can implement a simple addition routine that allows the user to select the two values to be added by first placing the addition function (found under Numeric) on the Block Diagram, and then right click each input terminal and select Create→Control from the pop-up menu. Checking the Front Panel, one finds two input boxes labeled "x" and "y." These are the default labels given to the inputs of the addition function. Right-clicking on the output terminal of the addition function, one can select Create→Indicator to place an indicator on the Front Panel. It is worth noting that when one right-clicks an input terminal in the Block Diagram to place either a constant, or a control, LabVIEW ensures that the input data type matches the function's desired input type. We can execute this simple program by going to the Front Panel, entering a number into each of the input controls (x and y), and then clicking the Run icon. The Run icon on the Front Panel menu is a large, white block arrow. After the program runs, the result of the addition shows up in the "x+y" indicator window. The user may observe that the program runs once and then terminates, so the values to be added must already be entered before execution. The simple program is shown in Figure A.7.

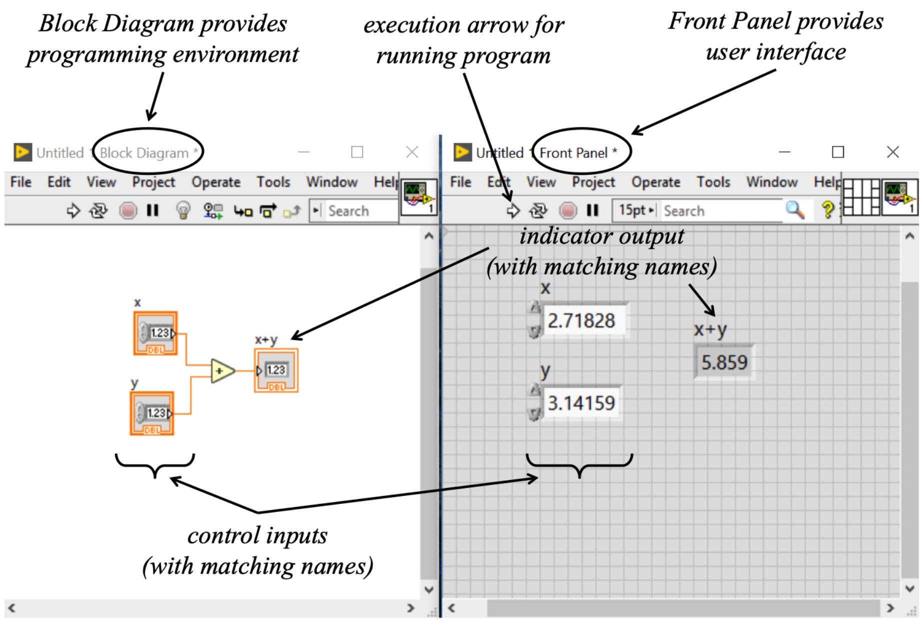

Block Diagram provides programming environment
execution arrow for running program
Front Panel provides user interface
Untitled Block Diagram *
File Edit View Project Operate Tools Window Help
Untitled Front Panel *
File Edit View Project Operate Tools Window Help
indicator output (with matching names)
x
2.71828
y
3.14159
x+y
5.859
1.23
DBL
control inputs (with matching names)

Figure A.7. ***Addition VI with User-controlled Inputs:*** *This VI shows an addition program with user-controlled input values. After the input values for x and y are set, the user can press the execution arrow and see the result. LabVIEW coordinates the variable name for any item that exists on both the Front Panel and the Block Diagram.*

How can we modify the program so that the user can change the numbers and re-compute without having to run the program over and over? Generally, when we want to do something repeatedly, we invoke a loop of some kind. The two basic loop types are the FOR loop and the WHILE loop. Although the student is expected to have already completed a basic programming course for engineers, we will briefly review each. The FOR loop executes its internal process a specified number of times. FOR n iterations, do this. In textual programming, this typically looks something like

```
for n = 1 to 10
     step 1
     step 2
        ...
     last step
end
```

The key element in a FOR loop structure is the number of iterations, n. The loop will execute its internal steps exactly n times, no more, and no less. On the other hand, a WHILE loop executes its internal steps *while* a condition is true. When the condition ceases to be true, the loop terminates. The astute reader will realize that a necessary element in this type of loop is to *check the loop condition every time through the loop*. In other words, one of the process steps within the loop must be a check on the loop condition. Textually, this looks something like

```
cond = true
while cond == true
     step 1
     step 2
        ...
     check cond
end
```

When the check on `cond` returns a `false`, the loop will terminate.

The fundamental concepts are no different in a graphical programming language, in general, and in LabVIEW specifically. In each case, there is a loop with code inside the loop, and the loop either terminates after a specific number of iterations, or it terminates based on some condition (which resolves to a Boolean value of true or false, e.g. is $x > 3$, or is $\epsilon < 0.001$). Each type of loop includes an iteration counter, which may be useful for counting up or down. For example, Figure A.8 shows the LabVIEW code for computing the factorial of a number. Recall that the factorial is given by $k! = 1\times2\times\cdots\times k$. The desired integer, k, determines the number of loop iterations. The result value is initialized to 1, and then each time through the loop, the previous result is multiplied by the next integer value. It is important to recognize that the iteration counter of a loop starts at 0!! Thus, to count from 1 to k, we need to add 1 to the iteration count. So, the first time through the loop, $i = 0$, so we take $i + 1$ times the initial value (of 1). The result, 1, is passed back to the beginning of the loop through a *shift register*. The second time through the loop the iteration counter is set to 1, but we want to multiply by 2, so we take $i + 1$. This proceeds for k times. At the termination of the loop, $i = k - 1$, but the loop has executed k times.

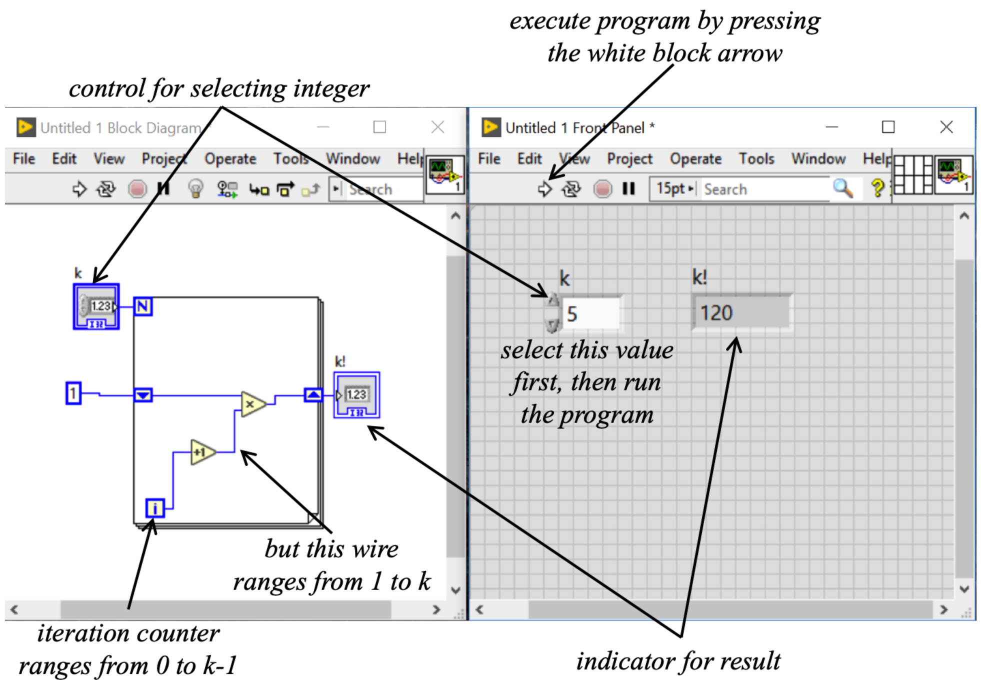

execute program by pressing the white block arrow
control for selecting integer
Untitled 1 Block Diagram
Untitled 1 Front Panel *
File Edit View Project Operate Tools Window Help
k
N
1
k!
i
5
120
select this value first, then run the program
but this wire ranges from 1 to k
iteration counter ranges from 0 to k-1
indicator for result

Figure A.8. ***Computing the Factorial in LabVIEW:*** *Using the FOR loop, we can calculate the factorial of a number. The loop iterates k times, which is what we need for the factorial, but the iteration count goes from 0 to k − 1, so we have to increment it by +1 to go from 1 to k. It is noteworthy that LabVIEW ensures that the variable names correspond between the Block Diagram and the Front Panel.*

Before proceeding, let us examine how to find the various components needed to "write" this program. We need a FOR loop, a control for k, the +1 increment function, the addition function, an indicator, and a *shift register*. All of these things are primarily accessed in the Block Diagram by right-clicking the mouse and bringing up the Functions Palette. For example, the Structures sub-palette is found as shown in Figure A.9. From here one can choose either the FOR loop or the WHILE loop (the Case Structure will be discussed below.

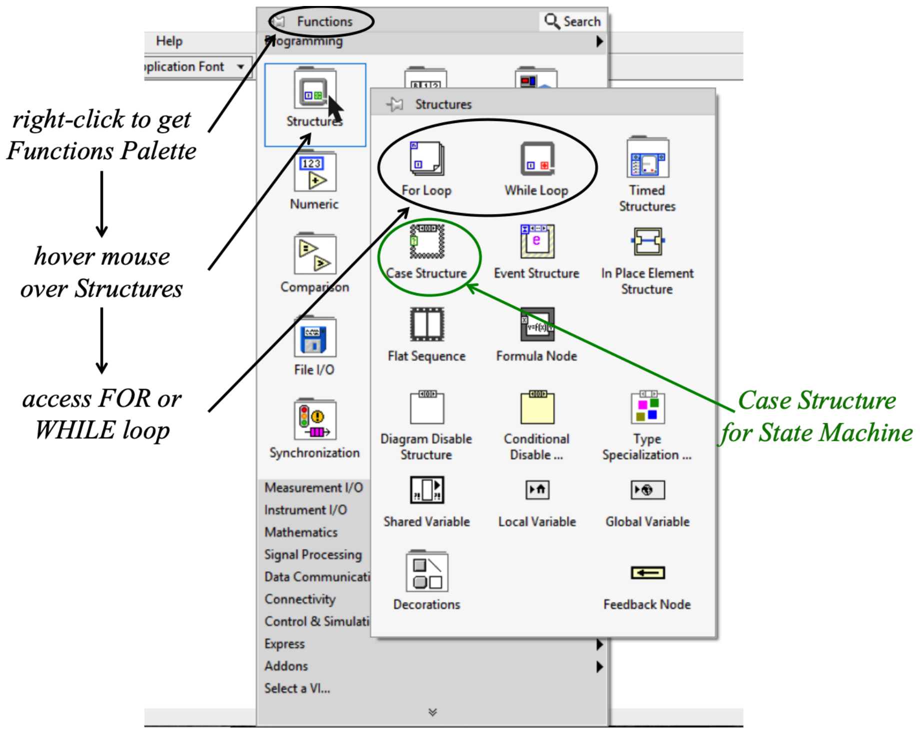


Figure A.9. ***Accessing the Structures Sub-palette:*** *The mouse right-click is the most powerful tool in LabVIEW and is used to bring up sub-menus for almost anything of interest. In the Block Diagram, a right-click brings up all the programming tools, defaulting to the Functions palette. The Structures palette is a sub-palette of the Functions palette.*

The addition and increment functions are *numeric* functions found in the Numeric sub-palette. This is shown in Figure A.10. The shift register is created by converting the normal tunnel to a shift register using a mouse right-click. The *tunnel* is the colored block that connects a wire between the outside of a loop, and the inside of a loop. After wiring the constant 1 to the left side of the FOR loop, pointing the mouse at the tunnel and right-clicking brings up a sub-menu that provides the choice of converting to a "Shift Register." After selecting this option, the corresponding shift register on the output side of the loop appears. To be clear, the shift register takes an output variable from the loop and *shifts* the variable back to the beginning of the loop. In other words, if a variable, s, is updated inside the loop, then when the loop enters iteration i, the shift register provides the value $s(i-1)$ at the beginning of the loop operations. Indeed, shift registers can be expected to provide $s(i-2)$, $s(i-3)$, etc. Shift Register selection is shown in Figure A.11.

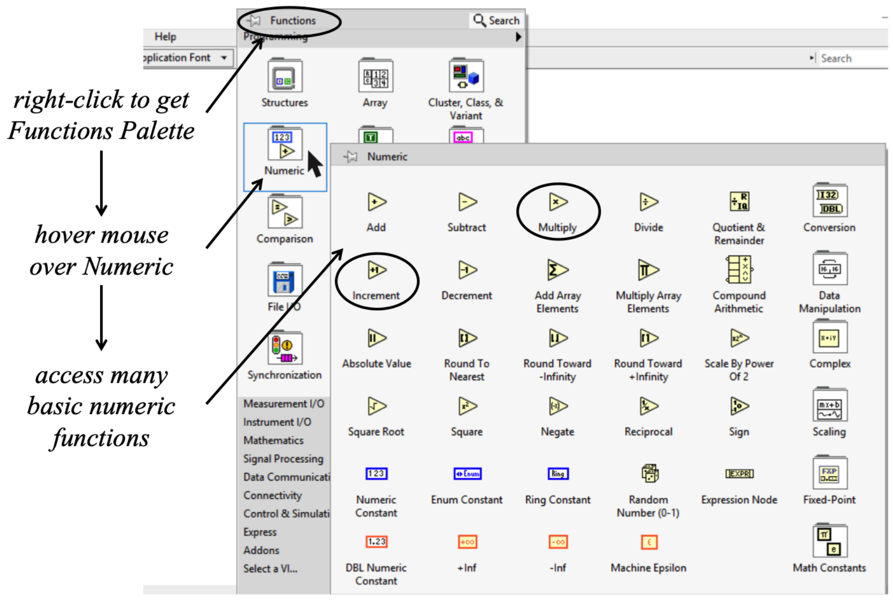


Figure A.10. *Accessing the Numeric Sub-palette: The mouse right-click (from the Block Diagram) is used to bring up the Functions palette. Hovering the mouse of the Numeric icon brings up the Numeric palette.*

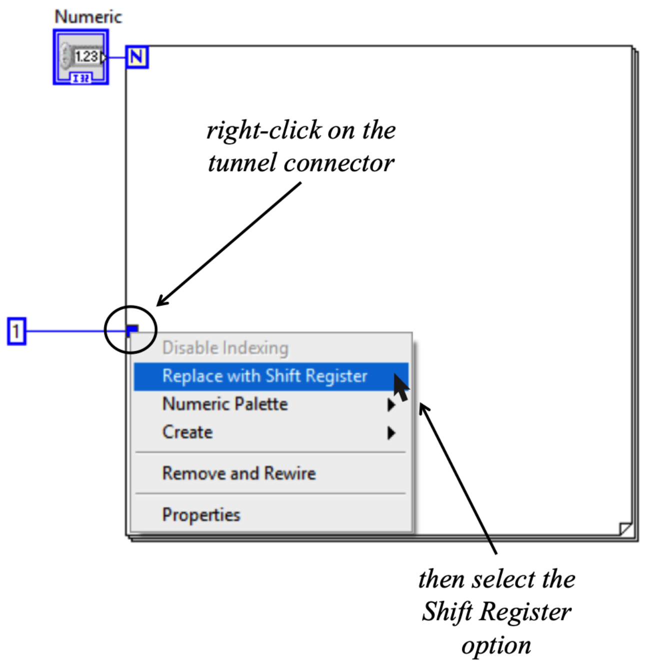

Numeric
1.23
N
1
right-click on the tunnel connector
Disable Indexing
Replace with Shift Register
Numeric Palette
Create
Remove and Rewire
Properties
then select the Shift Register option

Figure A.11. *Converting a Tunnel to a Shift Register: The shift register provides s(i − 1) to the loop for iteration i. Shift Registers are implemented by converting regular tunnels. By pointing at the tunnel with the mouse, and right-clicking, the sub-menu appears where on can choose Replace with Shift Register.*

Finally, we can create indicators on the Front Panel in two ways. From the Block Diagram, the user hovers over the output terminal of the loop (the right side of the shift register in this case) and right-clicks the mouse. This brings up a sub-menu and the user can choose Create→Indicator. An indicator will appear on the front panel. A second method is to make the Front Panel active and right click to bring up the Controls palette. Hovering over the Numeric icon brings up the Numeric sub-palette. From this palette, the user can select the Numeric Indicator and place it on the workspace. An icon for the indicator immediately appears on the Block Diagram as well, but the user must actively wire the indicator to the desired variable. These two methods are shown in Figure A.12.

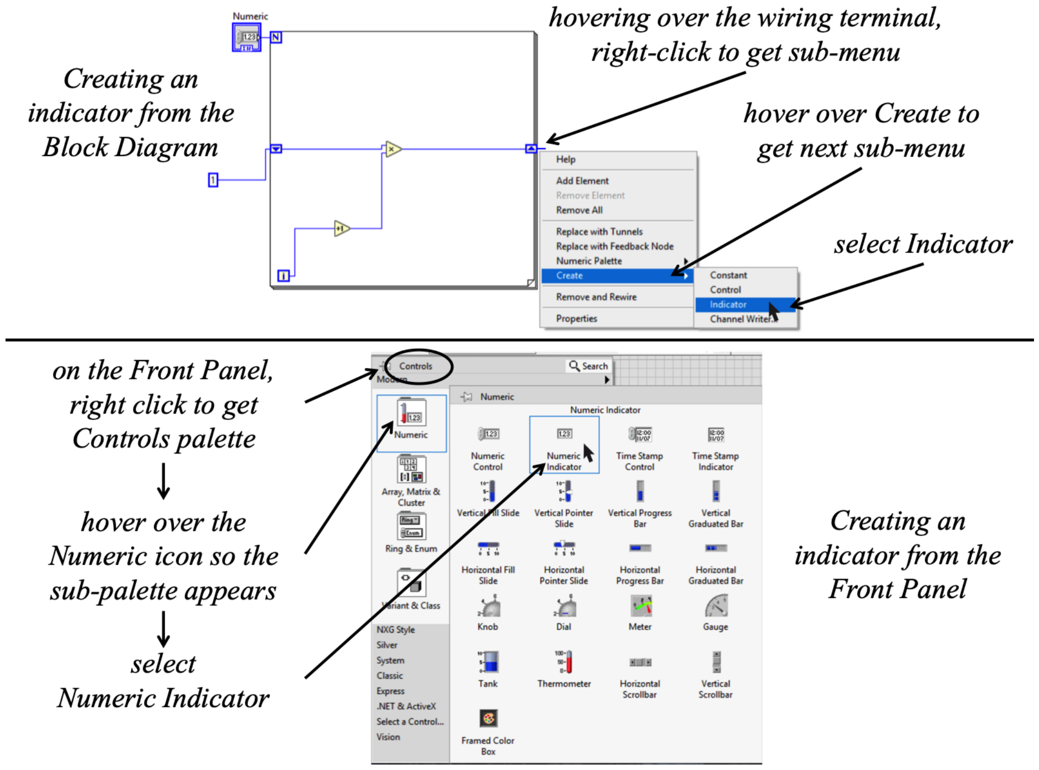


Figure A.12. *Two Methods to Create an Indicator: There are two methods to create an indicator to show the value of a variable on the Front Panel. On the Block Diagram the user hovers over the wiring terminal of the desired variable, right-clicks, and goes to Create→Indicator. From the Front Panel, the user right-clicks to bring up the Controls palette, hovers over the Numeric icon, and then selects Numeric Indicator.*

For this program, the Numeric Control wired to the "N" terminal of the FOR loop was created from the Block Diagram using the same method as the Indicator. However, when one right-clicks over the wiring terminal the choice is Create→Control.

Returning to the concept of loops, it may not be obvious, but most iterative tasks can be accomplished with either type of loop. Indeed, we can easily construct a program that uses a WHILE loop to compute the factorial of an integer. Although this is arguably *not* an efficient method, it is a good example of using either type of loop to accomplish the same task. Because we have to have a *condition* that keeps the WHILE loop operating, we might check the value of the iteration against the factorial integer during each iteration. If the iteration value is stored in the i icon, then we can compare the $(i+1)^{th}$ value to $k-1$, noting that k is a fixed value once we choose the integer for which we are computing the factorial. In the VI shown in Figure A.13 the comparison operator is the *greater than* function. This was chosen in preference to the *equal to* function, because we need to account for the case where $k = 0$.

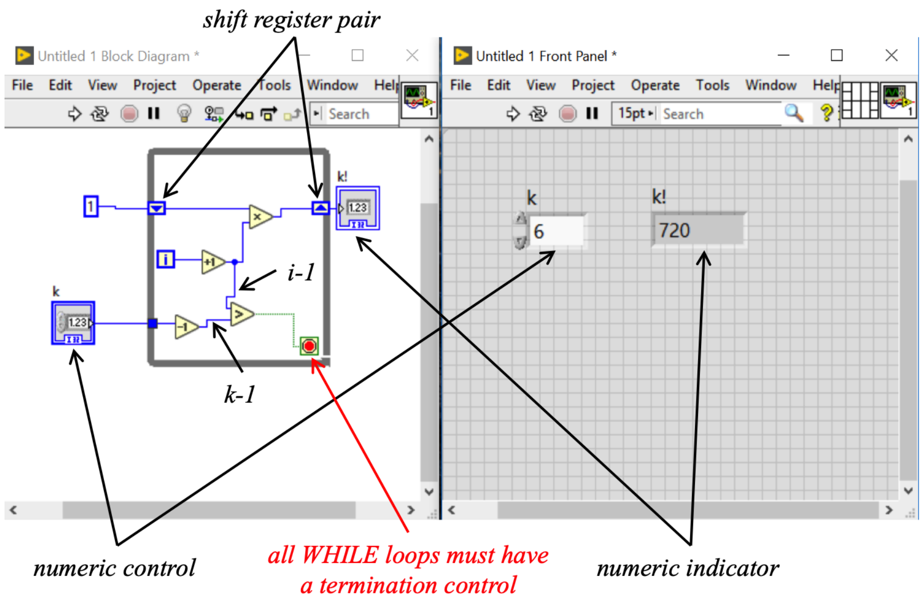

shift register pair
Untitled 1 Block Diagram *
File Edit View Project Operate Tools Window Help
Search
k!
1.23
i-1
k
1.23
k-1
Untitled 1 Front Panel *
File Edit View Project Operate Tools Window Help
15pt
Search
k
6
k!
720
numeric control
all WHILE loops must have a termination control
numeric indicator

Figure A.13. ***Computing the Factorial with a WHILE Loop:*** *The WHILE loop structure can also perform the iterative process needed to compute the factorial of an integer. However, to terminate the loop, we must check some condition. In this case, we check the comparative relationship between* $k-1$ *and* $i+1$*. When* $i+1 > k-1$*, the loop terminates. The stop sign icon is the loop termination terminal. When it is set to the stop sign, a* true *Boolean condition terminates the loop (i.e. stop = true).*

The student should try and program both of these VIs and verify their functionality. Further, they are designed so that the correct result for 0! occurs (recall that we define 0! = 1), and this should be checked as well.

A.2.2 State Machine Programming

The State Machine is an overall programming structure that divides a program into various states, with each state performing the requisite computations to determine which state should be next. For example, a line-following robot might have a program with four or five states as follows: initialization; go straight; veer left; veer right; halt. First, the program initializes and acquires its first measurements about its environment (i.e. the optical sensors determine if either–or both–optical sensors detect the line). After initialization, the program first goes straight. After each pass through the straight state, the program must decide whether to continue to go straight, veer left, or veer right. Assuming the robot starts moving in the Go Straight state, it might keep doing that while it checks the optical sensors. If it detects that it is drifting to the right of the line, it must go to the veer left state. Once the veering left brings it back onto the line, it returns to going straight until it needs to veer again. This may happen to the right or the left. When the end of the course is met, the robot comes to a halt. Using named circles to represent each state, we can draw a diagram of the states and how they relate to each other, as shown in Figure A.14.

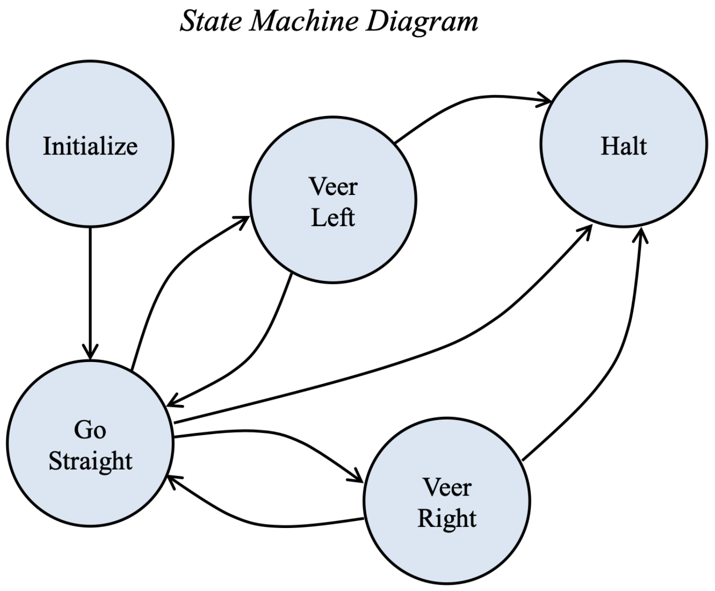


Figure A.14. ***State Diagram for State Machine:*** *The state diagram shows each state, and how one gets from one state to another. For example, the Initialize state* only *leads to the Go Straight state in this program. But the Veer Right case can lead back to the Go Straight state, or go to the Halt state. In each state, the computations must be made to decide what state to enter next.*

In LabVIEW a State Machine is created by using a WHILE loop as the outer structure, a Case Structure inside the WHILE loop that contains a case for each state, and a state-selector variable passing through a shift register. The desired starting state must be passed into the shift register at the beginning of the program. As part of the computations in the first state, the program must determine the next state to enter. This is passed out of the case structure into the output shift register at the end of the first loop iteration. At the beginning of the second loop iteration the program selects the new case based on the state selection passed through the shift register. In each case that executes, some logic decision must be made to determine the next case, based on how the user has designed the program structure. Based on the state diagram of Figure A.14, the LabVIEW program can be created as shown in Figures A.15 to A.19. The student should note that the program starts in the Initialization State, and then automatically goes to the Go Straight State. However, once in the Go Straight State, the program uses sensor values from the two optical sensors and the end-of-course Boolean sensor (a switch!) to decide the next state. The Veer Right State is shown next, but there is no preference between the Veer Right and Veer Left states based on the order the cases are programmed. It is all based on sensor value assessment.

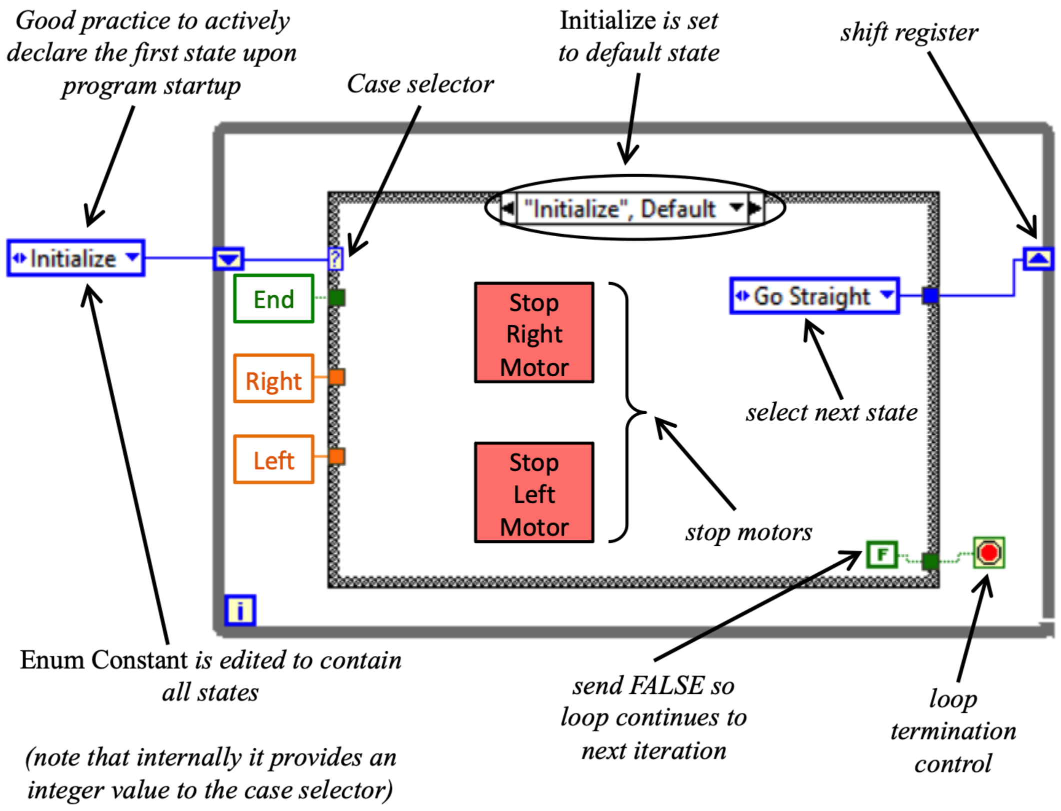

Good practice to actively declare the first state upon program startup
Case selector
Initialize is set to default state
shift register
"Initialize", Default
Initialize
End
Right
Left
Stop Right Motor
Stop Left Motor
Go Straight
select next state
stop motors
F
Enum Constant is edited to contain all states
(note that internally it provides an integer value to the case selector)
send FALSE so loop continues to next iteration
loop termination control

Figure A.15. ***Initialization State (Default):*** *Many programs need to initialize variables, set controlled outputs to a known state, etc. In this case, the main task is to ensure the robot drive motors are off.*

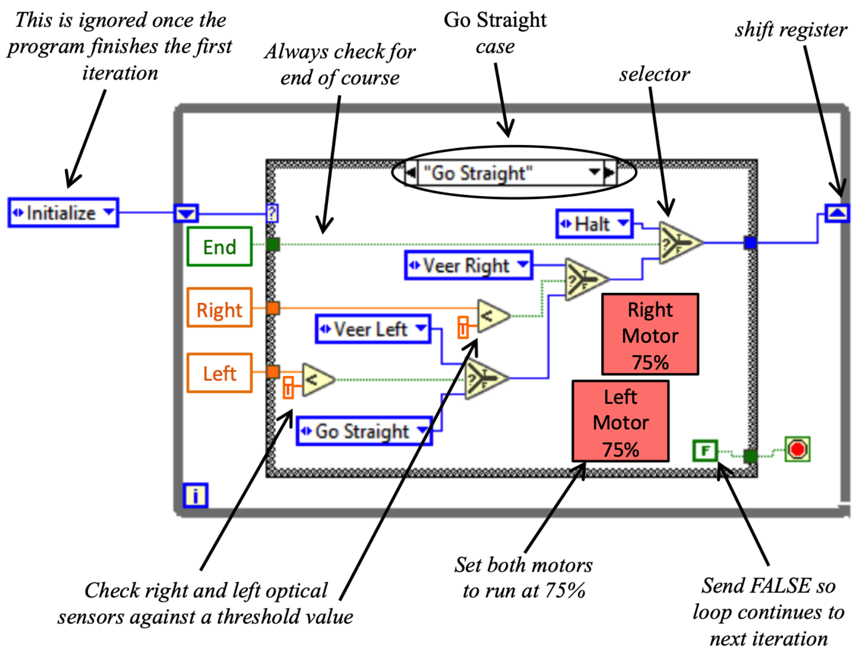

This is ignored once the program finishes the first iteration
Always check for end of course
Go Straight case
selector
shift register
Initialize
"Go Straight"
End
Right
Left
Halt
Veer Right
Veer Left
Go Straight
Right Motor 75%
Left Motor 75%
F
Check right and left optical sensors against a threshold value
Set both motors to run at 75%
Send FALSE so loop continues to next iteration

Figure A.16. ***Go Straight Case:*** *In the Go Straight case the motors are both turned on to 75% power and the left and right optical sensors are checked to make sure they both see the line. The optical sensors produce a real-valued output that is checked against a threshold. The end-of-course switch is checked in all active states so as soon as it sets itself to TRUE, the Halt state is chosen.*

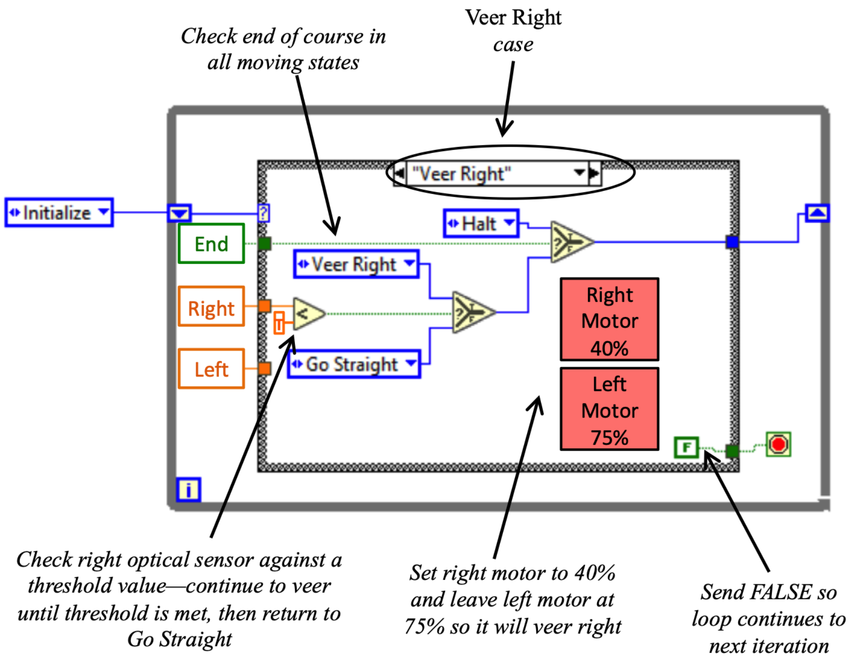

Check end of course in all moving states
Veer Right case
Initialize
"Veer Right"
End
Right
Left
Halt
Veer Right
Go Straight
Right Motor 40%
Left Motor 75%
F
Check right optical sensor against a threshold value—continue to veer until threshold is met, then return to Go Straight
Set right motor to 40% and leave left motor at 75% so it will veer right
Send FALSE so loop continues to next iteration

Figure A.17. ***Veer Right Case:*** *In the Veer Right case the left motor is left at 75% power, but the right motor is reduced to 40%. This causes the robot to veer to the right. The right optical sensor is tested until it is in the "normal" range. While the right optical sensor is above the threshold, the Veer Right case continues to be selected. Once the sensor value falls below the threshold, the Go Straight case is selected, and the robot returns to that state.*

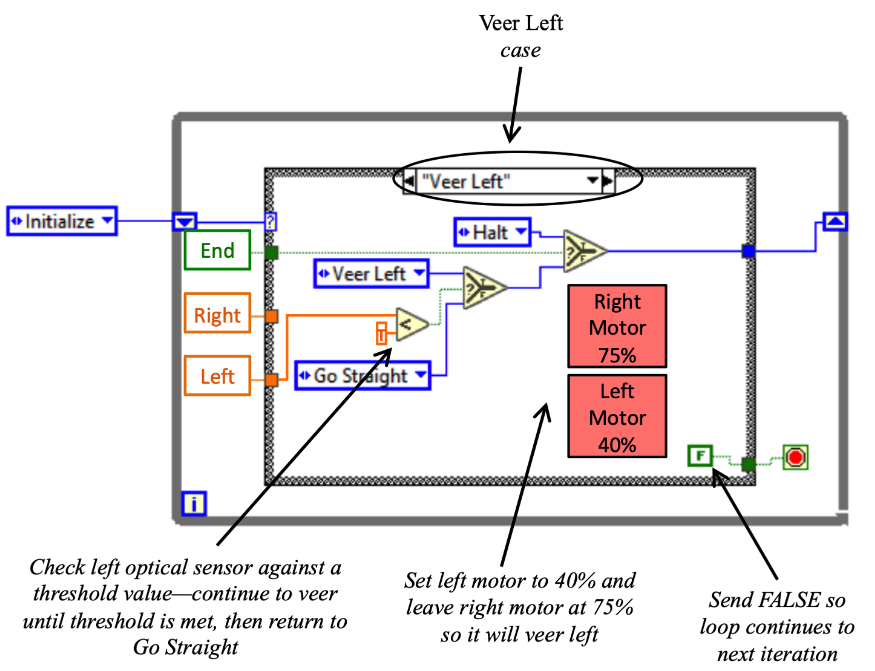

Veer Left
case
"Veer Left"
Initialize
End
Right
Left
Halt
Veer Left
Go Straight
Right Motor 75%
Left Motor 40%
F
i
Check left optical sensor against a threshold value—continue to veer until threshold is met, then return to Go Straight
Set left motor to 40% and leave right motor at 75% so it will veer left
Send FALSE so loop continues to next iteration

Figure A.18. ***Veer Left Case:*** *In the Veer Left case the right motor is left at 75% power, but the left motor is reduced to 40%. This causes the robot to veer to the left. The left optical sensor is tested until it is in the "normal" range. While the left optical sensor is above the threshold, the Veer Left case continues to be selected. Once the sensor value falls below the threshold, the Go Straight case is selected, and the robot returns to that state.*

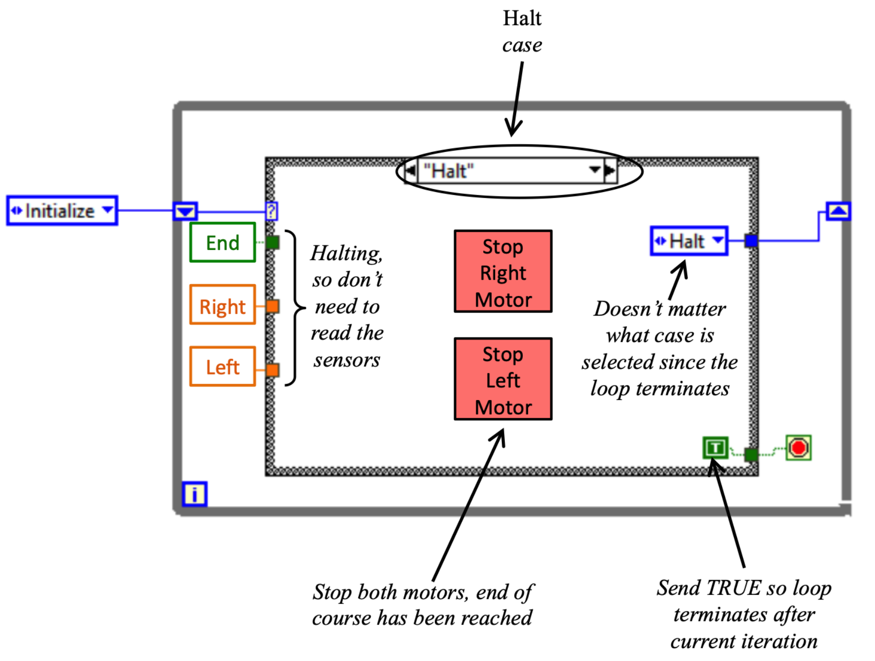

Halt
case
"Halt"
Initialize
End
Right
Left
Halting, so don't need to read the sensors
Stop Right Motor
Stop Left Motor
Halt
Doesn't matter what case is selected since the loop terminates
T
i
Stop both motors, end of course has been reached
Send TRUE so loop terminates after current iteration

Figure A.19. ***Halt Case:*** *From any of the movement states (Go Straight, Veer Right, Veer Left), if the END sensor detects the end of course, it will become TRUE and the Halt case will be selected. In the Halt case, both motors are stopped and the signal to terminate the WHILE loop is passed out of the case structure.*

A State Machine can have a number of different states, but the user should first sit down and think through the desired application, develop a state diagram, and understand how each state relates to the others before ever starting to code the VI. This helps get the number of states set correctly from the outset, which is important when using the Enum Constant as the State Selection variable. Once the code is created, if the Enum Constant is edited to add a state, then *every* instance of the variable must be edited. An alternative to this is to make the Enum Constant a Type Definition. This is done by pointing at the Enum Constant, right-clicking, and then selecting "Make Type Def." As long as Auto Update is active for the Type Definition, then any editing will be pushed to all instances of the variable. This creates an extra piece of code that must be saved in the local directory and is specific to the VI under development.

Yet another alternative for newer LabVIEW programmers is to use string constants as the state selection variables. For example, if we programmed the above state machine using strings, the program startup would like like Figure A.20.

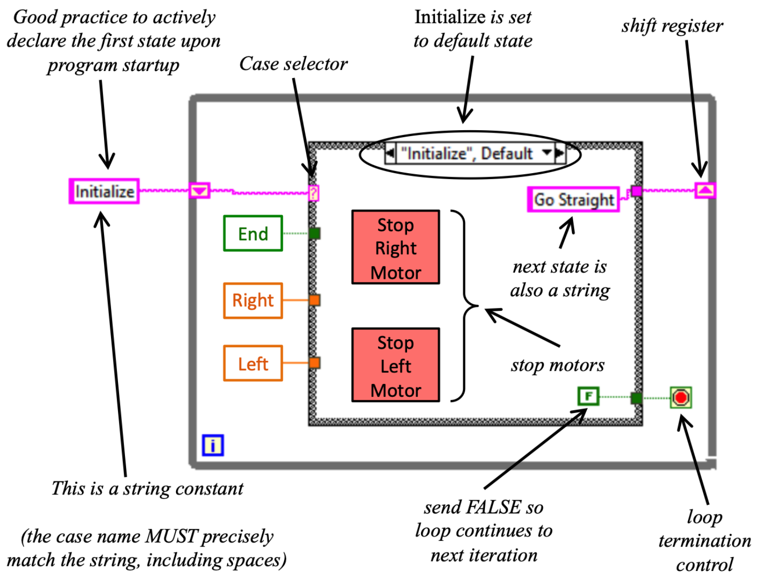

Good practice to actively declare the first state upon program startup
Case selector
Initialize is set to default state
shift register
"Initialize", Default
Initialize
Go Straight
End
Right
Left
Stop Right Motor
Stop Left Motor
next state is also a string
stop motors
This is a string constant
(the case name MUST precisely match the string, including spaces)
send FALSE so loop continues to next iteration
loop termination control

Figure A.20. ***Initialization State (Default) Using String Selection:*** *The string data type can be used as the state selection variable. But if strings are used, the text of the string and the text of the case (state) name* must *match exactly, including spaces, capitalization, etc. New programmers sometimes find strings more forgiving if the user desires to change the number of states.*

A.2.3 Front Panel Programming

The Front Panel is used to develop the user interface. The user can control inputs, as well as define the type and style of outputs. Like the Block Diagram, the Front Panel has a menu bar with several functions. The functions are quite similar to the Block Diagram, because many of them control program execution, which can be done from the Block Diagram of the Front Panel. The Front Panel menu is shown in Figure A.21.

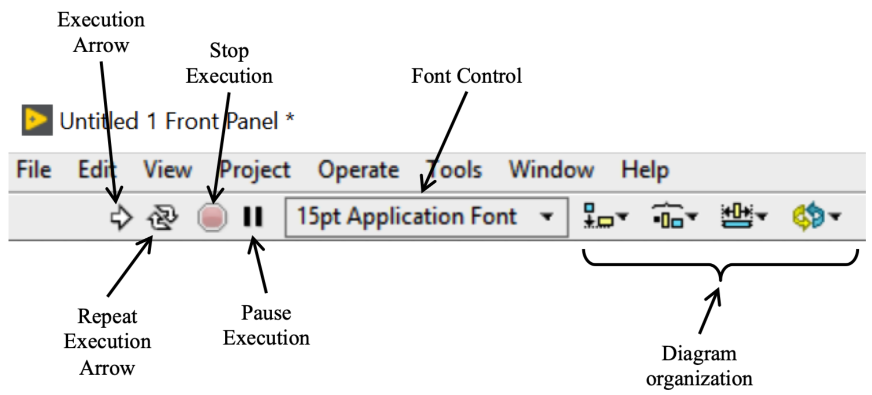


Figure A.21. ***Front Panel Menu Bar:*** *The Front Panel Menu bar has several icons for developing and executing a VI.*

The Front Panel has a main palette just like the Block Diagram. On the Front Panel, the palette is called the Controls palette. These Controls are for both inputs and outputs. The Controls palette is shown in Figure A.22. It is noteworthy that the Controls palette is the Modern palette. Similar functions can be found in the Silver, System, and Classic palettes. The NXG Style palette is specific to programming the Lego Mindstorm system (NXT or EV3). The icons in the Controls palette, such as Numeric, Boolean, String & Path, etc., provide access to sub-palettes. Figure A.23 shows the Numeric sub-palette expanded. Some of the sub-palette icons are controls (used for user input), while some are indicators (used for program display for the user). For example, in Figure A.8 the user input for the variable k is a Numeric Control. The output window for $k!$ is a Numeric Indicator. Of course, there are a number of other Front Panel controls that are useful. For example, the Graph sub-palette contains several different types of graphs that can be placed on the Front Panel. The two options that are most tuned to live data acquisition are the Waveform Chart and the Waveform Graph.

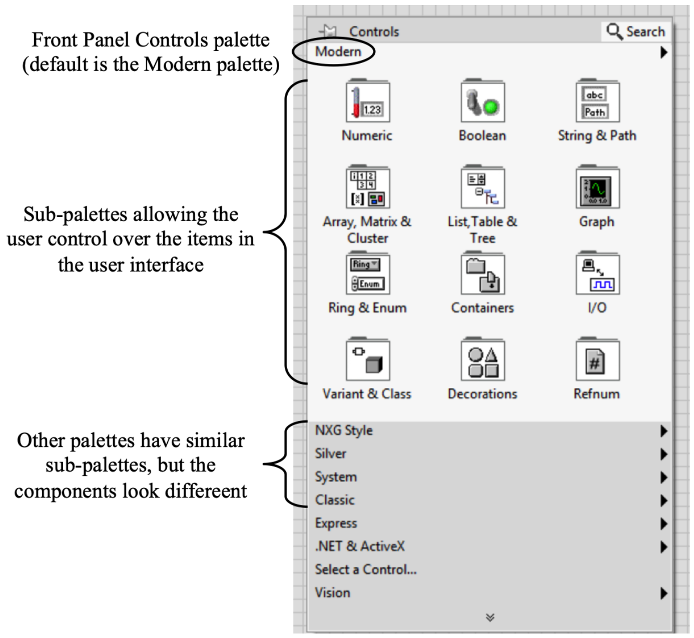


Figure A.22. ***Front Panel Controls Palette:*** *The Front Panel's main programming tool is the Controls Palette, brought up by right-clicking while the mouse is anywhere on the Front Panel. Each icon represents a sub-palette.*

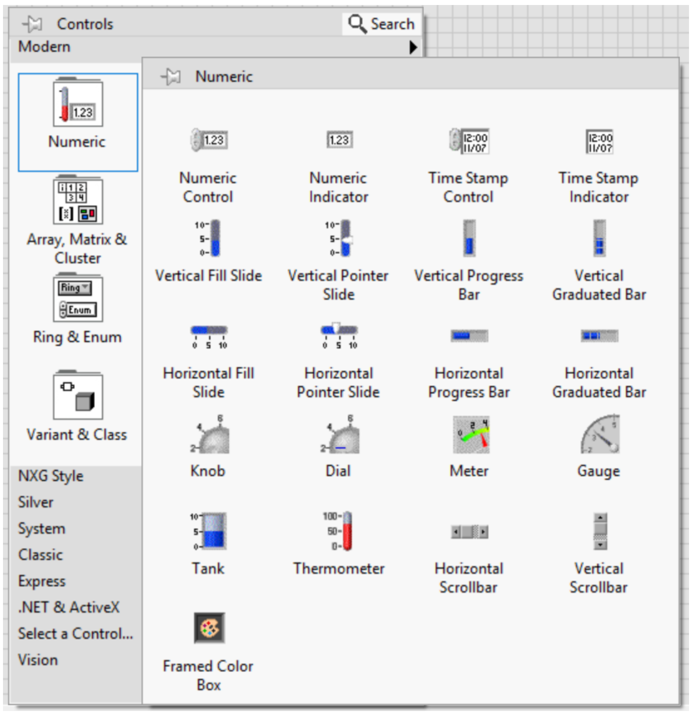


Figure A.23. ***Front Panel Numeric Sub-palette:*** *The numeric sub-palette provides various controls and indicators for numeric variables on the front panel.*

As an example of Front Panel activity, the following VI is a simple program to plot 0.1 seconds of a user-selected sine wave. As defined by the previous sentence, the Front

Panel needs a numeric control for the user to select the frequency, and a Waveform Chart to display the result. We will restrict the user to selecting a frequency between 20 and 50 Hz to limit the programming effort in terms of selecting a sample rate, etc. The Front Panel is a simple one as shown in Figure A.24. Each of the two elements was chosen from the Front Panel Controls palette: the numeric control was selected by the sequence Numeric→Numeric Control; the graph window was selected by the sequence Graphs→Waveform Graph.

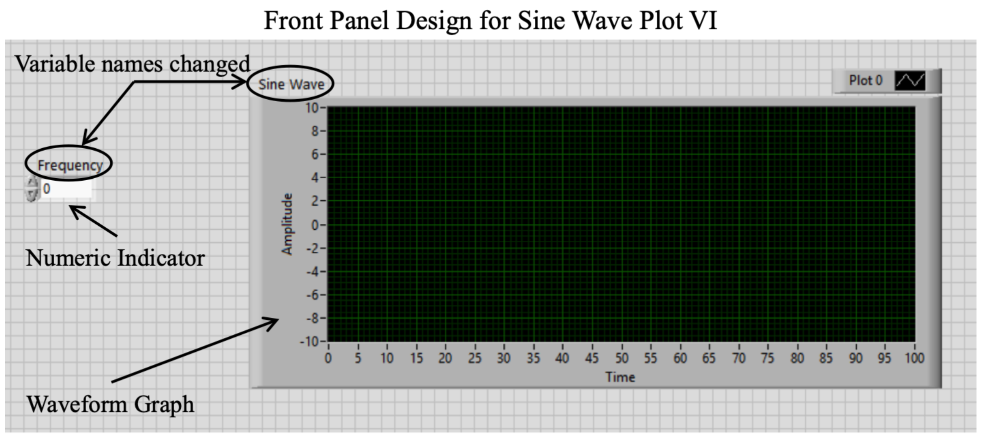
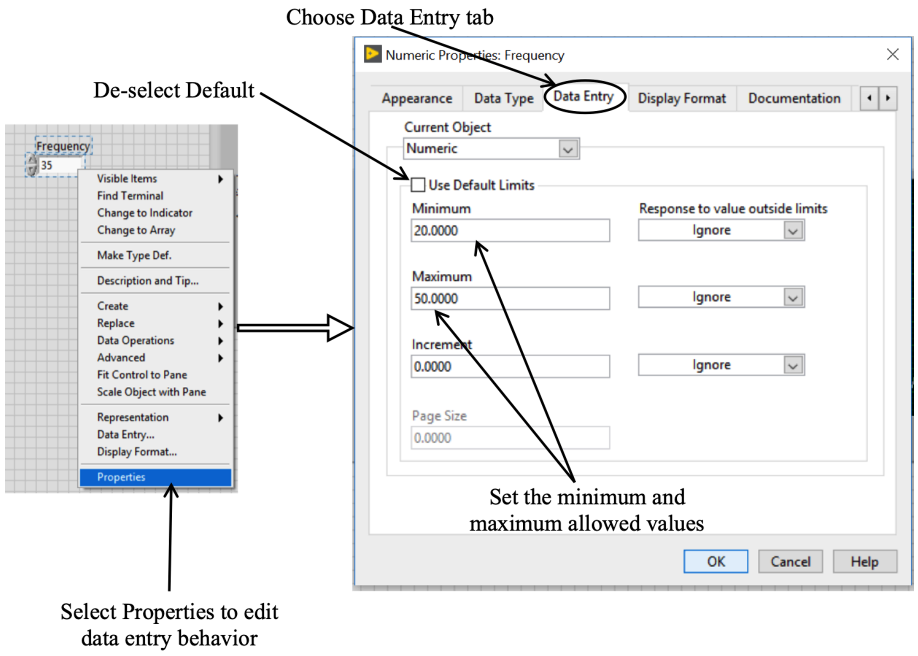


Figure A.24. ***Front Panel of VI to Plot Sine Wave:*** *The Front Panel shows a numeric control and a waveform graph. The programming elements can be completed on the Block Diagram.*

In order to restrict the frequency range, the user must edit the Properties of the numeric control. This is accomplished by right-clicking over the numeric control, and selecting Properties, as shown in Figure A.25. Once the Property menu appears, select Data Entry and de-select the automatic default mode, then edit the minimum and maximum values as desired, pressing OK when finished.



Current Object: Numeric

☐ Use Default Limits

Minimum	20.0000	Response to value outside limits: Ignore
Maximum	50.0000	Ignore
Increment	0.0000	Ignore
Page Size	0.0000	

Figure A.25. ***Editing a Front Panel Property:*** *The properties of a Front Panel object (either a control or an indicator) can be edited by right-clicking the object and selecting Properties. Appearance, color, and data behavior can all be edited.*

Once the Front Panel is set to suit the user, the Block Diagram can be completed. In this case, the Frequency control and the Sine Wave graph indicator are the only two things initially on the Block Diagram. In Figure A.26 a FOR loop has been added to create a time vector ($n \times 1$ array). This is done by setting the loop to run 100 times ($i = 0...99$), and multiplying the iteration count by 0.001 each time. The sample period is clearly being defined as 1 millisecond (or the sample *rate* is 1000 Hz), which is certainly adequate for frequencies between 20 and 50 Hz. The *auto-index tunnel* at the right side of the FOR loop is a special tunnel in LabVIEW. When working with arrays (vectors) LabVIEW gives the user the option of letting the program stack things in order when a sequence is being performed in a loop. Thus, in this case the auto-index tunnel stacks up the values in the loop, in order, so the output is $[0, 0.001, 0.002, ..., 0.099]$. This constitutes 0.1 seconds of time, divided into increments of 0.001 seconds. The other components are hopefully self-explanatory. Sine functions operate on radians, but humans like to think in Hertz, so we must convert the user-selected frequency into radians by multiplying by 2π. Then the full vector $2\pi f t$ is computed, and this is passed through the sine function. Finally, the resulting vector (array) is sent into the graph indicator on the Front Panel.

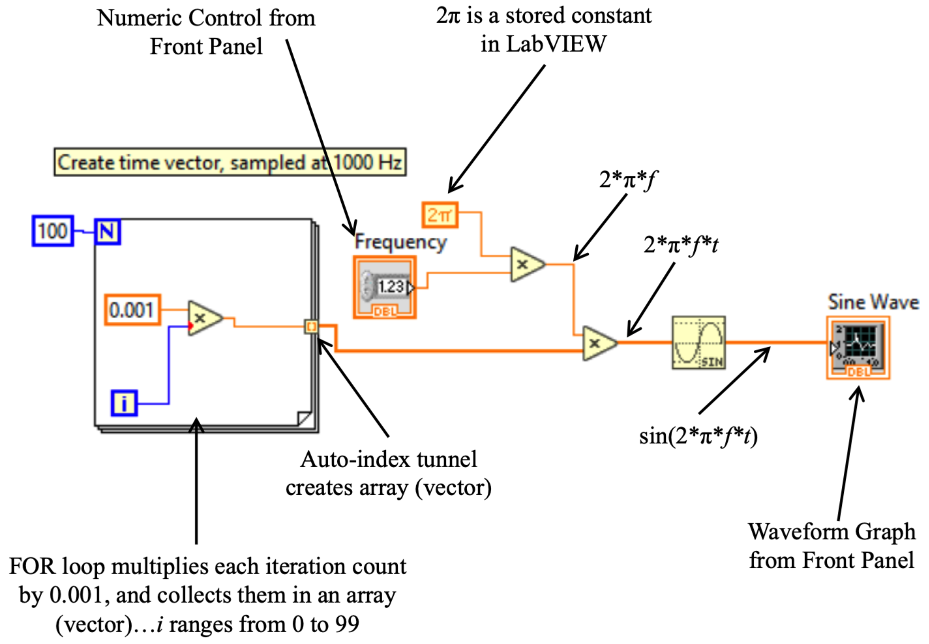

Numeric Control from Front Panel
2π is a stored constant in LabVIEW
Create time vector, sampled at 1000 Hz
100
N
0.001
i
Frequency
1.23
DBL
2π
2*π*f
2*π*f*t
Sine Wave
sin(2*π*f*t)
Auto-index tunnel creates array (vector)
Waveform Graph from Front Panel
FOR loop multiplies each iteration count by 0.001, and collects them in an array (vector)…i ranges from 0 to 99

Figure A.26. ***Block Diagram for Sine Wave VI:*** *The Block Diagram produces a time vector, and then computes* $\sin(2 \cdot pi \cdot f \cdot t)$ *and passes the resulting vector back to the graph indicator.*

Figure A.27 shows the Front Panel after selecting a frequency of 35 and executing the VI. The reader may note that the x-axis is set from 0 to 100. This is because the program does not pass the time vector itself to the indicator. Therefore, the x-axis values correspond to the index of each sine wave value in the vector being plotted. LabVIEW defaults to labeling the x-axis as *Time*, but this is a feature in LabVIEW and should often be edited by the user. To get time on the x-axis, the user would need to place an XY-Graph on the Front Panel, and then in the Block Diagram create a *cluster* with the time vector, and the sine value vector. The Block Diagram change to enable time on the x-axis is shown in Figure A.28. The lower part of this figure also shows the Front Panel after execution and the reader can see that the x-axis of the graph is now truly time.

Sine Wave VI After Execution

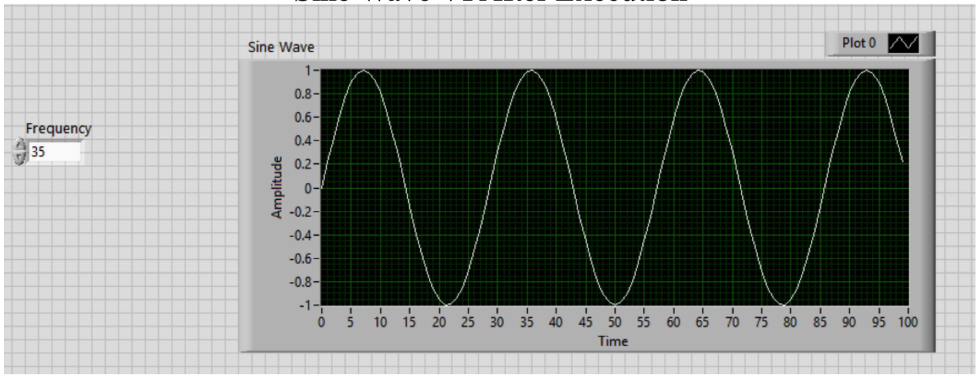


Figure A.27. ***Front Panel after Execution of Sine Wave VI:*** *After the user enters 35 into the Frequency control, and presses the execution arrow, the sine wave appears. As programed, no time vector was passed to the graph indicator, so the x-axis simply provides the index values for each element of the sine wave vector. The "time" label is a default LabVIEW value and is not correct in this instance.*

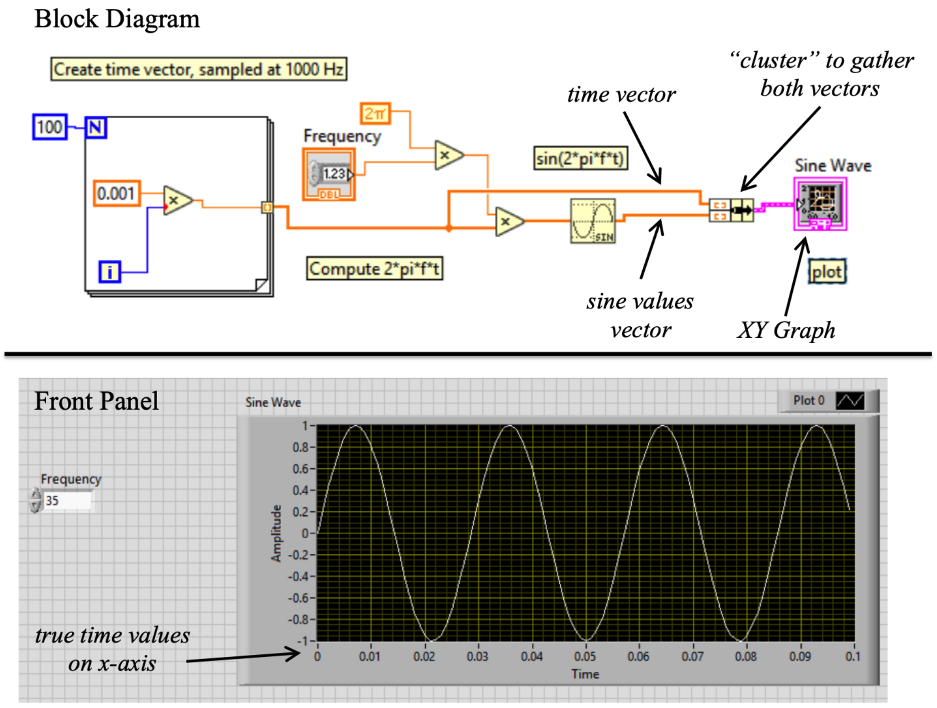


Figure A.28. ***Block Diagram and Front Panel with an XY-Graph Indicator:*** *The user can add the proper time vector to the graph indicator by choosing the XY-Graph option. This requires the user to combine the time vector and sine wave vector into a "cluster" and pass this to the XY-Graph indicator.*

A.2.4 Clusters

The student may wonder what this "cluster" thing is! A cluster is the same thing as a structure (struct) in many textual languages. The cluster allows the user to combine several variables into one group, but the important thing is that the variable types can be different. For example, if we wanted to create a cluster to store a person's bodily characteristics, we might have a string field for hair color, another string field for eye color, a real number field for weight, another real number field for height, and perhaps an integer field for age in years. Of course, Booleans can be grouped as well, but there are very few binary characteristics in humans. A prime example of a cluster in LabVIEW is the *error cluster*. Users can trap errors during VI execution to gracefully warn users of invalid input, failed sensor readings, or other problems. By trapping errors, the program can be either shutdown gracefully, or taken back to the user input state for a re-try on data entry. The error cluster is shown in its Front Panel view in Figure A.29. The error cluster contains a Boolean variable, an integer variable, and a string variable.

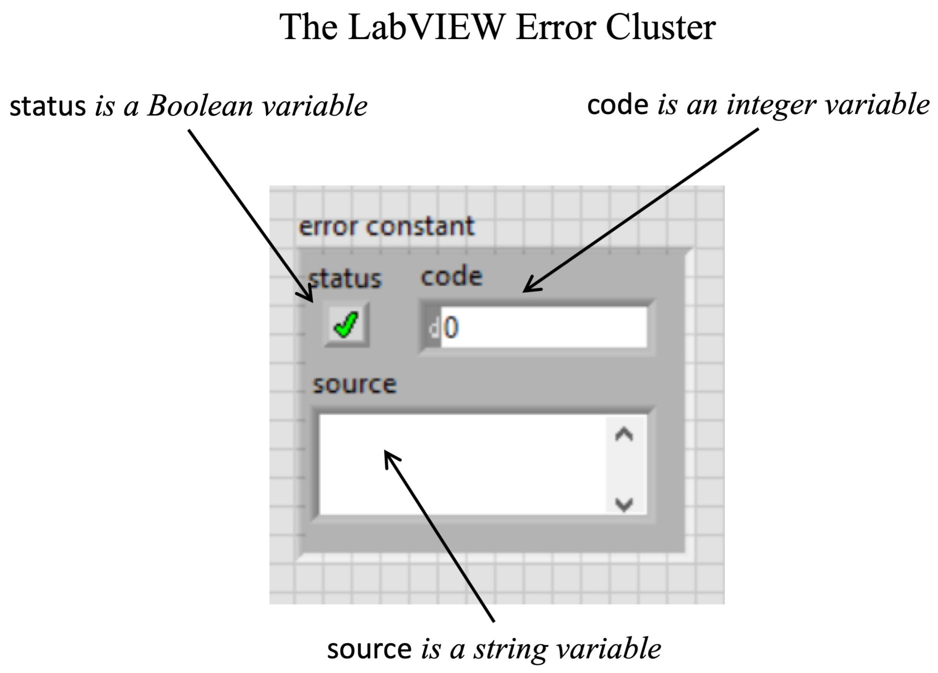

The LabVIEW Error Cluster
status *is a Boolean variable*
code *is an integer variable*
error constant
status
code
0
source
source *is a string variable*

Figure A.29. ***Front Panel View of LabVIEW Error Cluster:*** *The error cluster can be used to trap and handle programmatic errors without the program simply crashing. It is a cluster of three elements: one Boolean value, one integer value, and one string. The values can be unpacked in the Block Diagram using the "Uncluster" function.*

However, clusters can be a convenient way to transport data within the LabVIEW VI so there are instances, like our sine wave program in the previous section, where we cluster two vectors of the same data type and size to pass into the XY-Graph indicator. It might also be noted that we could plot several functions on the same plot by creating a separate cluster for each plot (the same time vector each time, and then the value vector for the y-axis), and then putting all the separate clusters into an array. This would be an *array of clusters*. The XY-Graph is designed to understand that when it sees an array of clusters, it plots each separate cluster as its own plot on the graph.

A.2.5 Select Function

Another function of significant interest to the state machine is the *select* function. This is shown in Figure A.30. The reader may recall seeing this function in each of the states in the state machine developed previously. This function takes three inputs and has one output. The two outer inputs (the upper and lower terminals) must be matched in data type. The middle input terminal is a Boolean terminal and takes a single TRUE-FALSE value. When the Boolean value is TRUE, the data input attached to the upper input terminal is passed to the output. When the Boolean value is FALSE, the data input connected to the lower input terminal is passed to the output. This is how each state can programmatically choose amongst several next-state options.

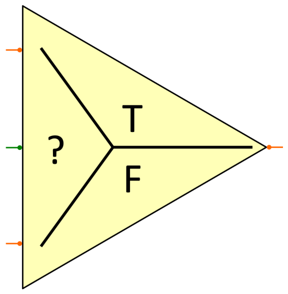


Figure A.30. ***Select Function:*** *The select function chooses between one of two inputs and passes the chosen input to the output. The choice is based on an external Boolean variable that is either TRUE or FALSE. The image shows orange terminals, indicating real values. However, this is just the default. The function is adaptable and will set the terminals to the data type of the first input connected. Of course, the two input data streams must be of the same data type and it should be evident that the output data type is simply designated as the input data type.*

A.2.6 Conditional Statements

Determining the next state in a state machine is always a programmatic requirement for each state. In the state machine above, we see that a sensor value is compared to some threshold, and a decision is made based on the comparative relationship between the sensor value and the threshold value. This is, in fact, a type of *conditional statement*. In the textual coding context, this type of conditional statement reads like this:

```
if x < y then
    choose A
else
    choose B
end
```

So, if the sensor value is less than the threshold, state A is chosen, and if the sensor value is greater than or equal to the threshold, state B is chosen. But we often encounter conditional statements where we want to perform some set of functions if one condition holds, and another set of functions (or possibly no functions) if the complementary condition holds. Of course, this could be implemented by using different states in a state machine, but another method that follows the classical text version is explained next.

In general, we might have a conditional statement such as

```
if x > y then
    perform function set A
else
    perform function set B
end
```

This can be accomplished in LabVIEW using a comparative statement and a *case structure*. The default case structure has a Boolean case selector and two states: TRUE and FALSE. The TRUE case corresponds to the *then* option of a conditional statement. Clearly, this leaves the FALSE case to correspond to the *else* option in a conditional statement (which may not do anything). An example relates to angular measure in radians. We frequently want to limit the angular measure to the interval $[0, 2\pi]$. If the program computes an angular measure that is greater than 2π, we must adjust the value using the *modulo* function. If the angular measure is in the desired interval, we do nothing. Text pseudocode provides the following

```
if theta > 2*pi then
    theta = mod(theta,2*pi)
end
```

It may be noted that if `theta` is less than or equal to `2*pi`, then there is nothing to do, so in text form we simply leave the *else* portion of the condition unstated since the process in the else condition is to do nothing anyway. This conditional statement is shown in LabVIEW code in Figure A.31.

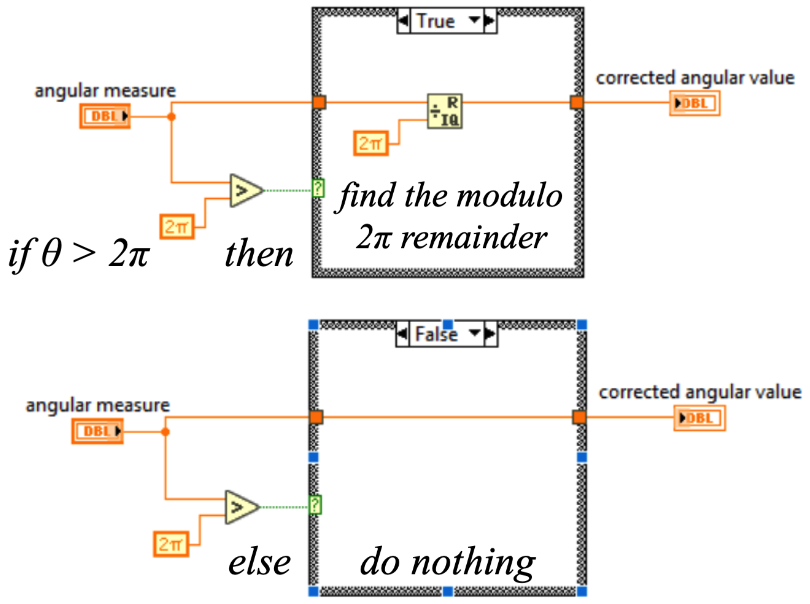


Figure A.31. ***Conditional Statement using Case Structure:*** *In the upper section, the conditional statement, "If $\theta > 2\pi$", is tested. If it is greater, the the Boolean output of the comparison is TRUE,* then *the modulo function is invoked (it is in the Numeric sub-palette). In the lower panel, the angular measure is less than or equal to π, meaning the output of the comparison is FALSE and the* else *condition is executed. The wire going all the way across simply passes the variable through without operating on it (does nothing).*

A.2.7 Arrays

We know that there are at least four different types of data: real, integer, Boolean, and string. There are also types of data structures. Clusters were covered previously. Another important data structure is the array. In very general terms, an array is an n-dimensional spreadsheet. Most often, we deal with one-dimensional, or two-dimensional arrays (and occasionally three-dimensional arrays). As a matter of nomenclature, a one-dimensional array is also referred to as a vector. Whether the array is $k \times 1$ or $1 \times k$, the array is still a vector. When an array has more than one dimension, it is simply an array. The various data types have different wiring representations depending on whether the wire represents a scalar (single value), a vector (one-dimensional array), or an array (two-dimensional or higher). For real and integer numbers, the thin line represents a scalar, a thick line represents a vector, and a double line represents a two-dimensional (or higher) array. On the other hand, Boolean variables are represented with a single green dotted line for scalar, a double-dotted line for a vector (of Booleans), and a triple-dotted line for higher dimensions. This representation also holds true for strings, but in pink. These data types and structures are shown in Figure A.32.

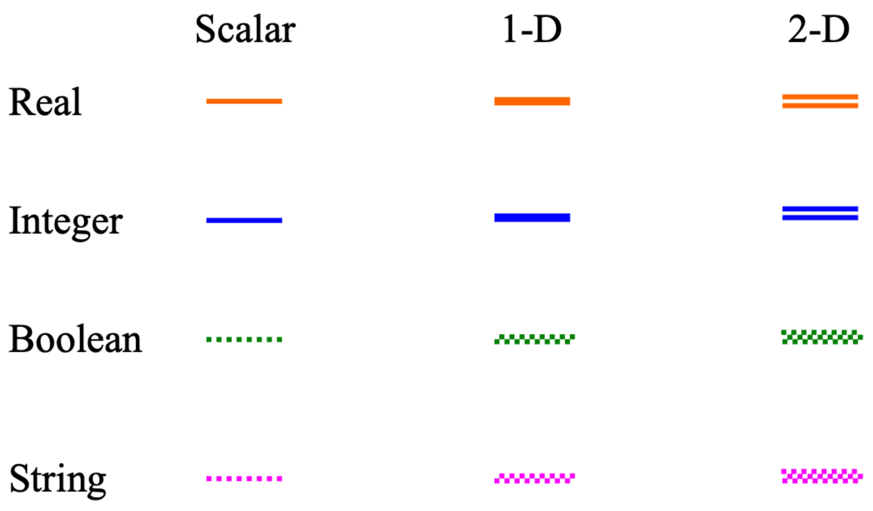


Figure A.32. ***Data Line Types by Structure:*** *Scalars are usually the thinner line–of whatever color. As the structure becomes a vector, or higher-dimensional array, the line type changes.*

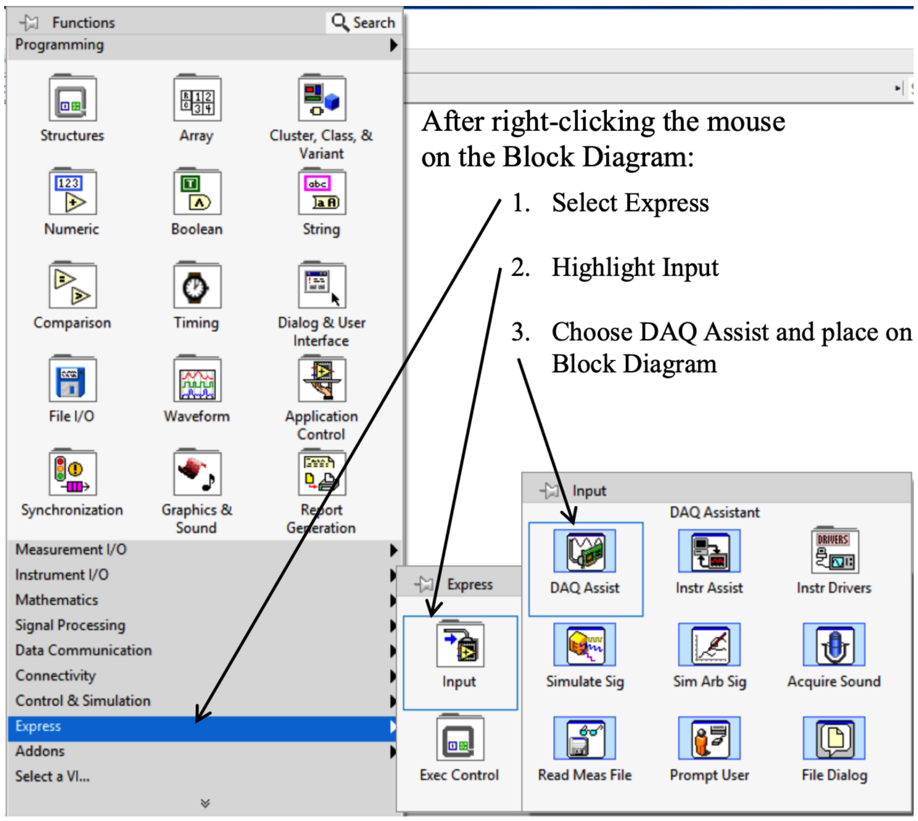
A.2.8 Data Acquisition

To perform data acquisition, the student must have access to a LabVIEW installation with the DAQmx drivers and the proper hardware. The instructor's material for this text contains information about a lab setup used at the University of Oklahoma that provides each student station with a PC running LabVIEW, a National Instruments data acquisition device (USB-6211), and a common data source so that all students receive the same signal. Given the understanding that students have access to a basic National Instruments data acquisition system, the following information provides a short tutorial on basic data acquisition. The methods shown here are not the most efficient, but make use of the Express VI tools for data input and output. Proper LabVIEW programming utilizes primitive functions to implement the data input and output processes, giving the programmer very fine control over the entire procedure.

The data acquisition devices typically have one or more analog input channels, one or more analog output channels, and several lines of digital input/output. Digital lines either read (input) a high or low voltage signal, or write (output) a high or low voltage signal.

Analog signals are acquired using analog-to-digital conversion, and analog outputs are produced via digital-to-analog conversion. The reader should review Chapter 2 for the analog-to-digital, and digital-to-analog conversion topics.

Data acquisition is instantiated from the Block Diagram using the Express VI tools for Input. Figure A.33 shows the menu sequence for finding the DAQ Assist wizard for acquiring data. The user opens the basic palette menu with a right-click on the Block Diagram, and then navigates to Express→Input→DAQ Assist. After placing the DAQ Assist on the Block Diagram, a wizard opens that guides the user through channel selection, and configuration. The National Instruments USB-6211 has 16 single-ended, analog input channels. Each pair can be made into a differential input pair for eight differential channels. The device has two analog output channels, and eight bits of digital I/O. The reader should refer tp Chapter 3 for a review of the difference between a single-ended configuration and a differential configuration. The USB-6211 has an aggregate sampling rate of 250 KHz. The *aggregate* means that this is the total rate that can be achieved across the acquisition channels. Therefore, if one is acquiring five channels of analog data, the maximum sampling rate for each channel is 50 KHz. But if one is only acquiring two channels of analog data, then each channel can sample at a rate as high as 125 KHz. Of course, we can sample at slower rates, even though the sampling rate capacity is there. It should be noted, though, that *only one* sampling rate can be set for an acquisition on a single device, regardless of the number of channels.

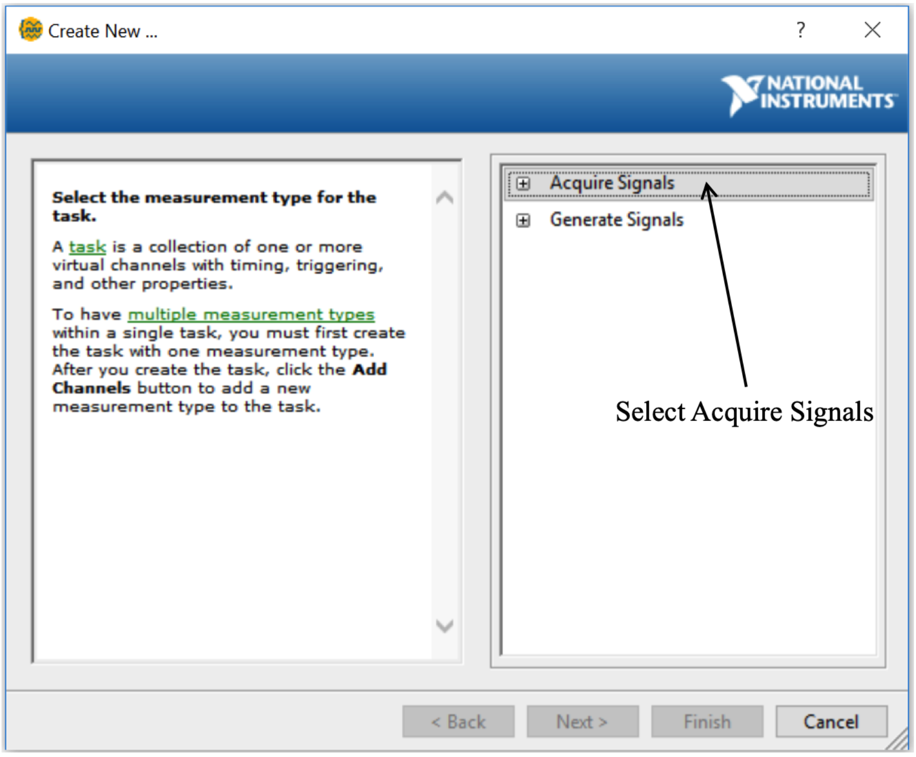


After right-clicking the mouse on the Block Diagram:

1. Select Express
2. Highlight Input
3. Choose DAQ Assist and place on Block Diagram

Figure A.33. ***Navigating to the DAQ Assist Wizard:*** *The DAQ Assist wizard is a good starting point for data acquisition. Expert LabVIEW programming uses primitive functions to manage the data acquisition process, but this text uses the Express wizard to focus on the instrumentation aspect.*

The DAQmx drivers are required so that LabVIEW can interface with the National Instruments hardware. The DAQmx drivers must be properly installed for the DAQ Assist wizard to appear in the Express→Input→DAQ Assist sequence. Once the wizard has been started, the first user selection is whether to acquire input or whether to generate output. This is shown in Figure A.34. Acquisition is addressed first.



Create New ...

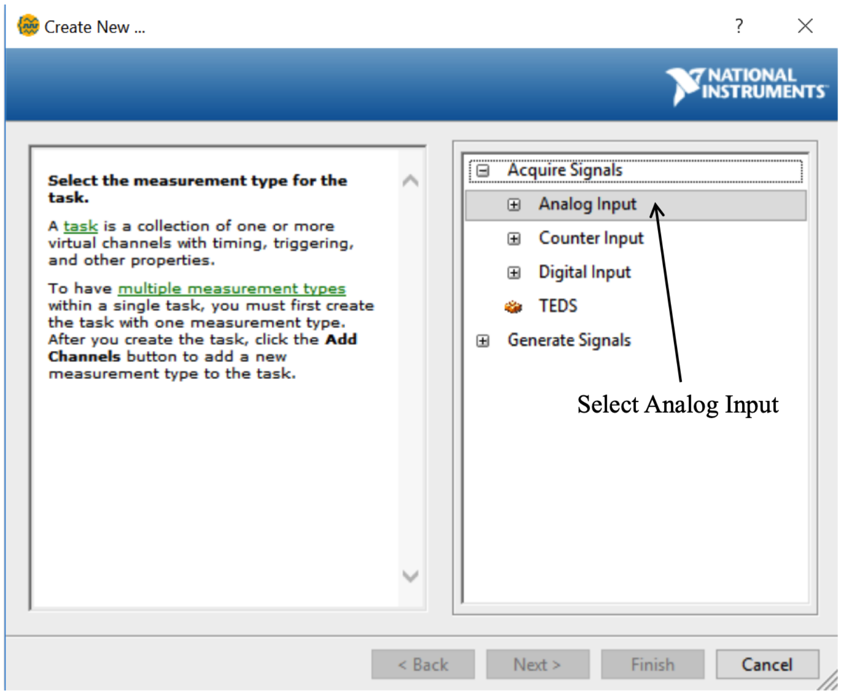
Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure A.34. ***Input or Output Selection:*** *The user must first select whether to input our output a signal.*

After choosing Acquire Signals, the next window asks the user what type of input is going to be acquired. Digital inputs are clearly just digital I/O. Counter inputs acquire clock-like signals and can measure rate, detect pulse-encoded signals, or detect pulse width. Analog inputs are continuous-time electrical signals, and this is what we will explore in this text. Selecting the Analog Input is shown in Figure A.35. The TEDS choice indicates the use of a sensor that complies with the Transducer Electronic Data Sheet (TEDS). This is an emerging technology that relies on *smart* transducers that inherently carry a TEDS that can be transferred to the host program for automated configuration. It is based on the IEEE 1451.4 specification. This is very similar to many *plug and play* devices in our modern technological world. One no longer needs a separate driver for every thumb drive or other device–for example, this author's Garmin Nuvi GPS device–that might be inserted into a USB port. The devices auto-configure themselves. The TEDS implementation of the IEEE 1451.4 specification is an effort to realize a user-friendly level of connectivity in data acquisition with external sensors.


Create New ...

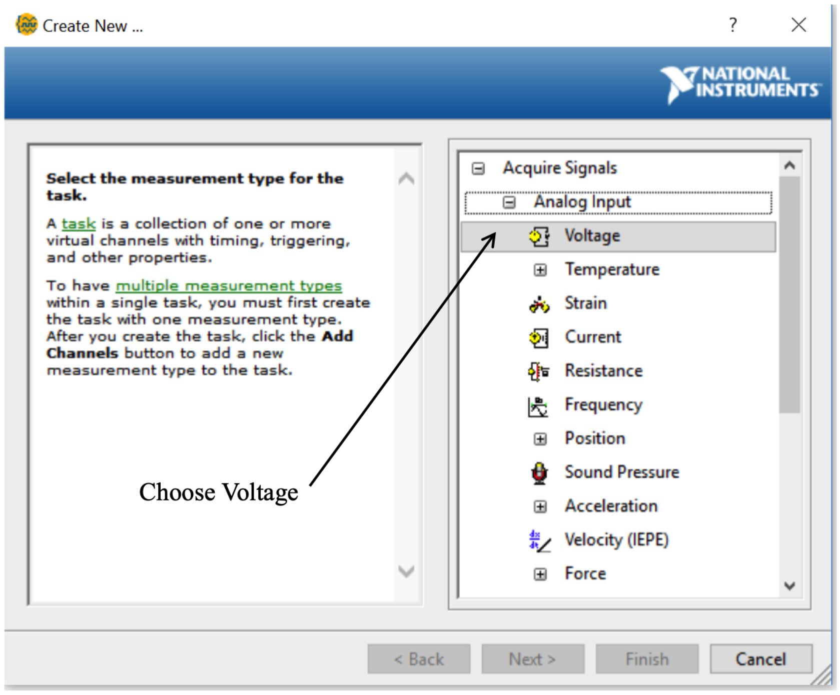
Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure A.35. ***Analog Input Selection:*** *Analog Input allows the user to acquire a continuous-time electrical signal.*

The suggested projects in this text all provide single-ended voltage signals for data acquisition. However, as seen in Figure A.36 current signals can also be acquired. LabVIEW has many built-in conversions for things like temperature and strain. Recalling the sensors chapter (Chapter 4), different types of thermocouples and thermistors exist, and the DAQ Assist tool can help the user correctly configure the data acquisition based on the sensor type. Strain signals typically require a specific National Instruments data acquisition component that supplies part of the resistive bridge, and can be programmed for the zero-strain resistance, the bridge supply voltage, and the gage factor, so that the data acquisition can programmatically read the strain directly, rather than the user having to develop code to specifically convert the detected voltage to a resistance, and then convert that resistance to a strain value. For this example we will focus on Voltage acquisition.


Create New ...

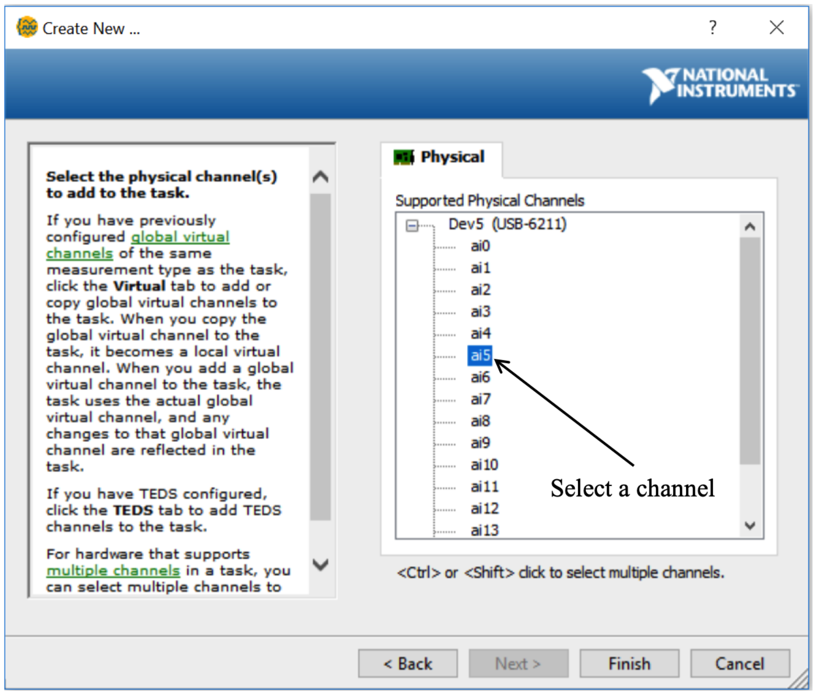
Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure A.36. ***Voltage Acquisition Selection:*** *Data acquisition in this text is voltage-based.*

The channel selection window that appears next relies on the existence of a data acquisition device being connected to the computer. If no acquisition device is connected, the window will not show any available channels and will simply state that no device was detected. Thus, assuming the student has a device properly connected to the computer, the channel selection window appears, as shown in Figure A.37. In this figure we can see that a USB-6211 has been connected to the computer and has been assigned as Device 5. The USB-6211 has 16 single-ended channels, so there are 16 channels available to choose (numbered 0 through 15). In Figure A.37 channel 5 is shown as being selected. After channel selection is completed, the user presses the Finish button in the window and is taken to the configuration window.



Select the physical channel(s) to add to the task.

If you have previously configured global virtual channels of the same measurement type as the task, click the **Virtual** tab to add or copy global virtual channels to the task. When you copy the global virtual channel to the task, it becomes a local virtual channel. When you add a global virtual channel to the task, the task uses the actual global virtual channel, and any changes to that global virtual channel are reflected in the task.

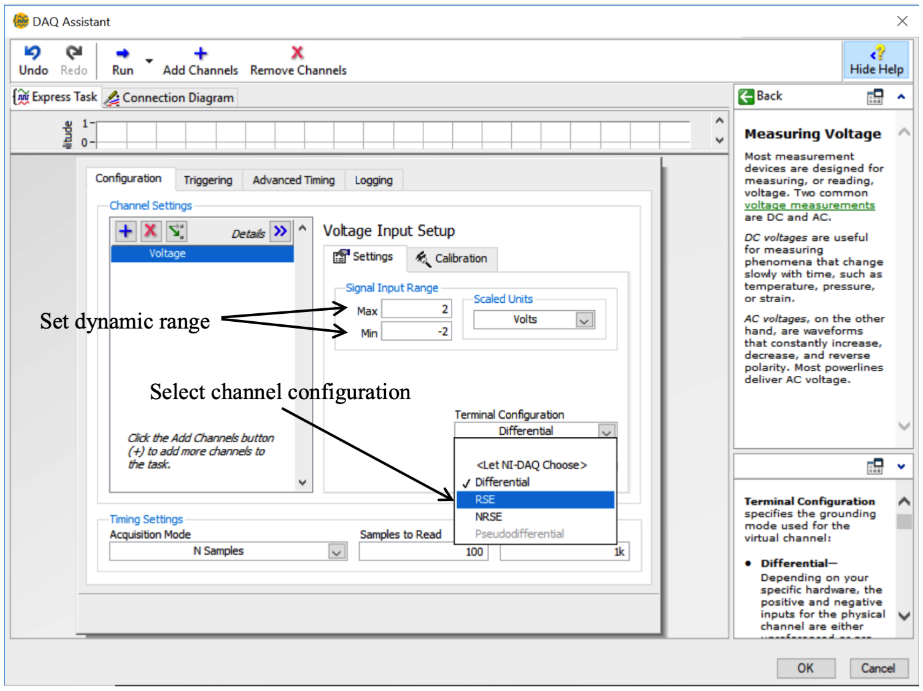
If you have TEDS configured, click the **TEDS** tab to add TEDS channels to the task.

For hardware that supports multiple channels in a task, you can select multiple channels to

<Ctrl> or <Shift> click to select multiple channels.

Figure A.37. ***Channel Selection for the USB-6211:*** *The data acquisition device must be present and connected for the channel selection window to function. Simply clicking on a channel selects it. Multiple channels can be selected by holding down the CTRL key while clicking on each desired channel. A block of channels can be chosen using the SHIFT key.*

After channel selection is complete, the user must configure the chosen channel(s). The next window to appear is the DAQ Assistant window, shown in Figure A.38. This window helps the user configure channel specific information. Chapter 2 discussed the concept of *dynamic range*. This is determined in this window by setting the Max and Min for the *Signal Input Range*. The default values (not shown here) are +10 and -10 volts. The *Terminal Configuration* must also be set. This is chosen from a drop-down menu. The default selection is Differential. This implies that if one chooses channel 5 for input and does not alter the terminal configuration, then channel 13 is automatically committed to the differential pair and is unavailable to be used as a single-ended channel. However, if channel 5 is set to RSE (Reference Single-Ended), then channel 13 is available to also be used for an RSE connection. In this text the projects are all oriented around RSE configuration. The user should be aware that the Dynamic Range and Terminal Configuration settings are unique to each channel. If more than one channel is selected for acquisition, *the user must configure EACH channel*.



Signal Input Range

Max 2

Min -2

Scaled Units Volts

Terminal Configuration Differential

Timing Settings

Acquisition Mode N Samples

Samples to Read 100

1k

Measuring Voltage

Most measurement devices are designed for measuring, or reading, voltage. Two common voltage measurements are DC and AC.

DC voltages are useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain.

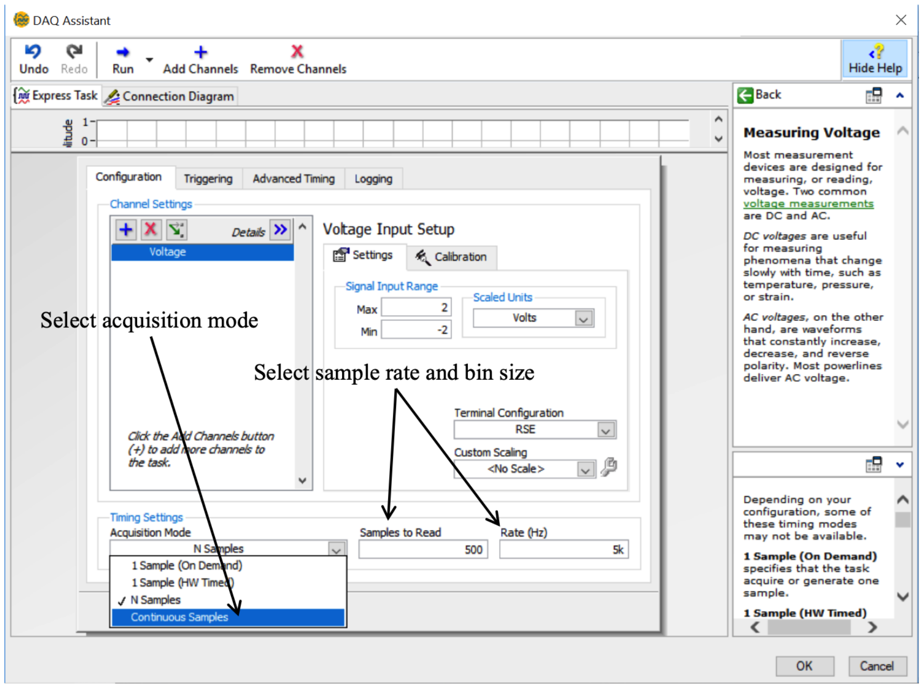
AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

Terminal Configuration specifies the grounding mode used for the virtual channel:

- **Differential**—Depending on your specific hardware, the positive and negative inputs for the physical channel are either

Figure A.38. ***Setting Dynamic Range and Terminal Configuration:*** *Dynamic Range should always be considered and set appropriately. Leaving the default range of +/-10 volts for a +/-1 volt signal wastes resolution that can not be magically added back after the acquisition is over. Terminal Configuration in this text is RSE.*

Once the Voltage Input Setup is complete, the user must configure the Timing Settings. In Figure A.39 the Timing Settings appear at the bottom of the DAQ Assistant window. It is notable that these settings are in their own small box and are universal for all channels in the acquisition process. The *Acquisition Mode* determines whether to continuously acquire data (requiring a WHILE loop), or whether to acquire a specified number of samples and then stop the conversion process. In general, the Continuous Samples mode is the preferred mode. This will be explained in more detail below. The *Rate (Hz)* setting specifies the sampling rate. This was discussed in Chapter 2, and the user must ensure that the sampling rate is fast enough to avoid any concerns about frequency aliasing. The default is 1 KHz, but it has been changed in this example to 5 KHz. Finally the *Samples to Read* must be configured. This is often referred to as the *bin size*. A good rule-of-thumb is to set this to about 10% of the sampling rate.



Max 2
Min -2
Scaled Units Volts
Terminal Configuration RSE
Custom Scaling <No Scale>
Acquisition Mode N Samples
Samples to Read 500
Rate (Hz) 5k

Click the Add Channels button (+) to add more channels to the task.

Measuring Voltage

Most measurement devices are designed for measuring, or reading, voltage. Two common voltage measurements are DC and AC.

DC voltages are useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain.

AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

Depending on your configuration, some of these timing modes may not be available.

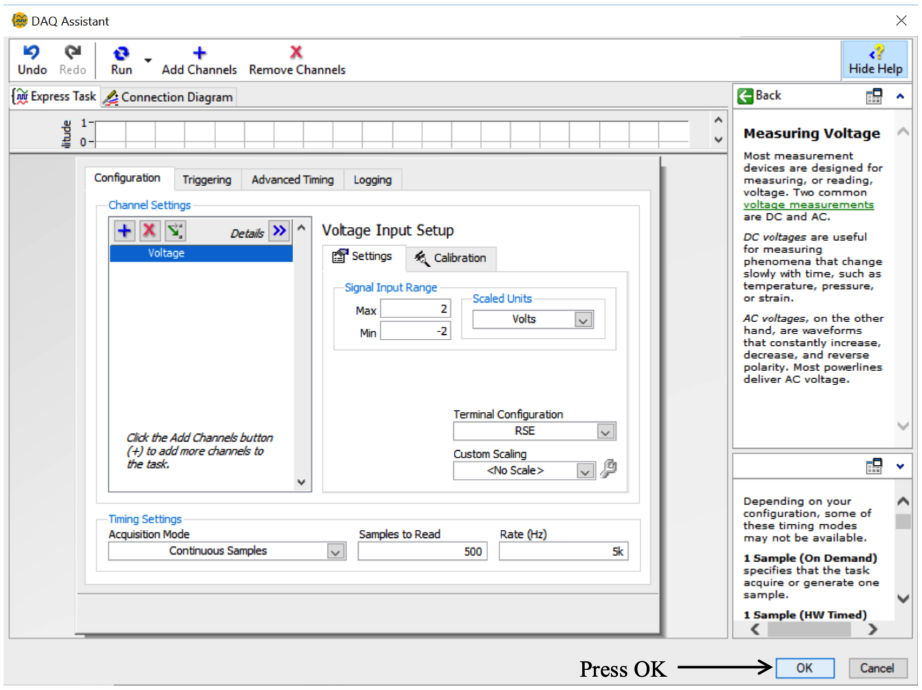
1 Sample (On Demand) specifies that the task acquire or generate one sample.

1 Sample (HW Timed)

Figure A.39. *Setting Timing Characteristics: The sampling rate is an easily understood concept, but setting the Samples to Read can be confusing to students new to data acquisition. The acquisition mode has an effect on overall program behavior as well, as discussed in the text.*

The sampling rate (in Hz) tells us how many times per second the analog-to-digital converter (ADC) will sample a voltage from the channel and digitize it. A sample rate of 5 KHz means that this sample-and-convert process occurs 5000 times every second. Ideally, the samples are *equally spaced*, meaning we expect the clock that controls the sampling process to be exact. Of course, this is not reality, but the clock error is very small in modern devices, so that it is a negligible fraction of the sample period (the time interval between samples). Indeed, to the extent that the clock of the data acquisition device is not perfect, the (possibly) indeterminate period between samples will appear as noise in the process. However, withi modern devices this is of less concern than fundamental noise reduction issues. The data acquisition device has an internal buffer for collecting each digitized voltage value, and it is the size of this buffer that one sets when choosing a value for Samples to Read. The ADC, once configured, will digitize discrete values at the defined rate until it is told to stop. However, if the data acquisition device had to transfer every individual digitized value over the data bus to the computer as the value was digitized, at 5000 samples per second the acquisition device would spend a great deal of time accessing the data bus. There is a certain amount of overhead associated with accessing the bus, and it turns out that it is much more efficient to collect up a batch of samples and incur that overhead only once in awhile. The actual data transfer can be very fast in modern computers, once the data bus is activated (the overhead to activate and then close the bus is the bottleneck). The buffer can be thought of as a storage bin. We collect up some number of digitized values in the storage bin. When the bin is full, we quickly empty it to the data bus, and then return it to its place so the ADC can deposit the incoming sample. The critical feature in this process is that the ADC is converting at the specified rate *the entire time*. This means that in between two conversions the bin must be emptied to the bus and returned to accept the incoming digitized value. For example, with a sampling rate of 5 KHz and a bin size (Samples to Read) of 500 samples, the data acquisition device and computer have 1/5000th of a second (0.0002 seconds) to empty a full bin and get the bin back in place ready to accept the next data sample. Obviously there is a balance between Samples to Read and Sampling Rate.

A further complication occurs in the Acquisition Mode setting. The *1 Sample (On Demand)* and *1 Sample (HW Timed)* options are of little concern in terms of the Samples to Read and Rate. Indeed, if On Demand is chosen, then the Samples to Read and Rate are grayed out, as we are only going to convert a single sample each time the data acquisition routine is called. In the HW Timed version the hardware provides the clock (and, therefore, the Sample Rate is user selectable), but only one sample is collected each time. This represents the case of transferring each sample, as it is converted, over the data bus to the computer. The rate here is limited to bus speeds and whether the host computer has any background processes consuming compute cycles. Of most interest in the current context are the two choices: N Samples; and Continuous. In the N Samples mode, the bin size determines the N samples to read and the ADC is shut down and the bin is flushed to the host computer at the end of conversion. This may seem appealing to avoid timing issues, but it is incompatible with digital filter processes (amongst other things). A digital filter (discrete filter) has memory and stores previous values of the input and output to properly compute the next filtered value. When a filtering process first begins, the *previous* values are, of course, unknown, so the output of the filter is not considered to be valid until all the stored values for the filter have been populated by the current incoming data process. In view of the acquisition mode issue, this can cause a problem because the filter function will be restarted every time a new data acquisition is begun. The effective result is that signal distortions due to filter transients can appear cyclically when one chooses the N Samples acquisition mode. Of course, expert LabVIEW programmers can avoid these issues by using primitive functions to control the filter state, but this is not the only problem. Another issue occurs with timing , in general. In a normal VI, there are multiple other functions in the Block Diagram processing the data stream being delivered from the DAQ Express VI. These other functions take a certain amount of time (it's fast in modern computers, but still a finite amount of time). If N Samples is chosen as the acquisition mode, then the ADC is shut down and flushed after the N samples are collected. Then the remaining functions are executed. Because the ADC has been shut down it can lose priority in the system, so that other background processes (from the operating system, not necessarily within the LabVIEW program) may usurp the timing such that the remaining functions don't always execute in precisely the same amount of time. This significantly increases the risk of creating variable sample spacing. This appears as another noise source in the data, and it should be noted that the math theory behind digital signal processing (at this level) always assumes identical sample spacing in a discrete signal. The message to the student is that the Continuous Mode of acquisition is generally the preferred mode for the projects presented in this text. When the continuous mode is selected, another window pops up during compilation telling the user that the chosen mode typically requires a WHILE loop, and the window asks the user if it should go ahead and configure the loop. The wise user selects YES to this question. LabVIEW configures the WHILE loop so that the STOP control button passes through the DAQ function before terminating the loop. This ensures that the ADC is shut down properly and the buffers cleared prior to program termination. By doing this the data acquisition device is left in a known state when it is next called into action. Figure A.40 shows the completed configuration for data acquisition when the user then selects OK to complete the process.



Channel Settings

Voltage

Click the Add Channels button (+) to add more channels to the task.

Voltage Input Setup

Signal Input Range

Max 2

Min -2

Scaled Units: Volts

Terminal Configuration: RSE

Custom Scaling: <No Scale>

Timing Settings

Acquisition Mode: Continuous Samples

Samples to Read: 500

Rate (Hz): 5k

Measuring Voltage

Most measurement devices are designed for measuring, or reading, voltage. Two common voltage measurements are DC and AC.

DC voltages are useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain.

AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

Depending on your configuration, some of these timing modes may not be available.

1 Sample (On Demand) specifies that the task acquire or generate one sample.

1 Sample (HW Timed)

Figure A.40. ***Completed Acquisition Configuration:*** *Once all fields have been correctly set and proper values entered, the user presses OK to complete the assistant.*

The signal wire coming out of the DAQ VI contains what is referred to as *Dynamic Data*. Dynamic data is its own data structure and contains several pieces of information for a signal stream. The dynamic data wire is a hatched blue wire, as shown in Figure A.41. The Dynamic Data stream contains information about the start time, the signal name (what you named each voltage channel being acquired), the Δt between samples, the actual sample value, and the units associated with the channel. If one desires just the raw sample values, there is a function for converting dynamic data to a scalar vector (or array if collecting more than one channel of data).

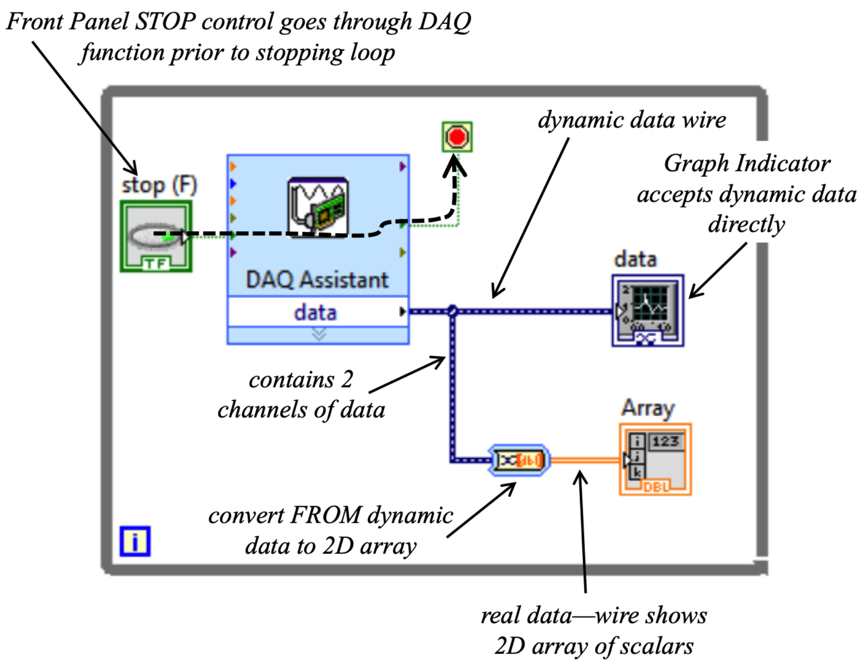
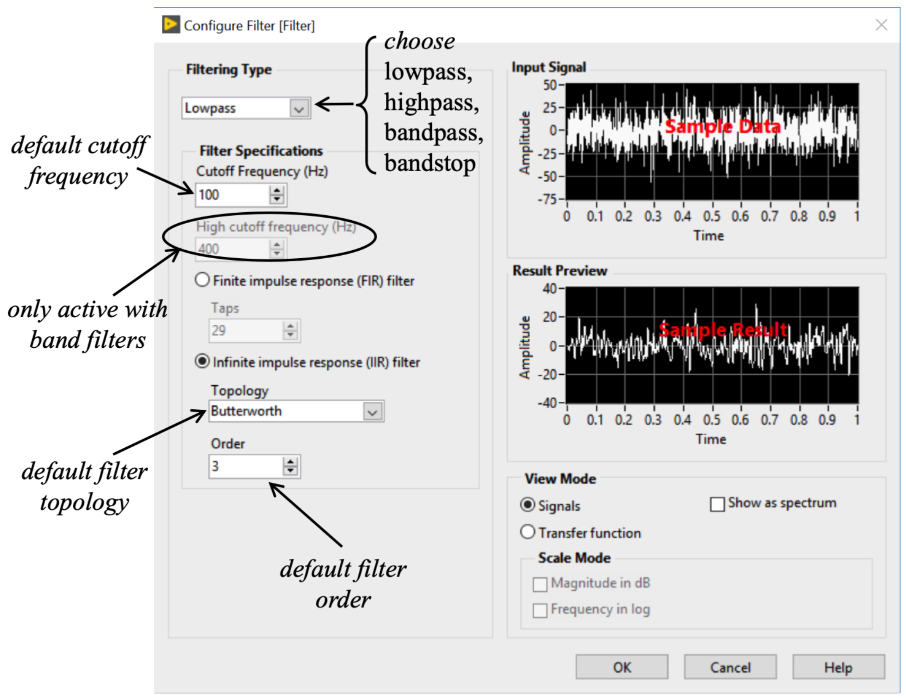

Front Panel STOP control goes through DAQ function prior to stopping loop
stop (F)
DAQ Assistant
data
dynamic data wire
Graph Indicator accepts dynamic data directly
data
contains 2 channels of data
Array
convert FROM dynamic data to 2D array
real data—wire shows 2D array of scalars

Figure A.41. ***Dynamic Data from DAQ Process:*** *The Dynamic Data structure contains more than just raw data values. Many LabVIEW functions are designed to operate with the dynamic data structure, including the various graphical indicators of the Front Panel. This figure also shows the proper configuration of the STOP control when using the Continuous Acquisition mode.*

All of the Express functions are designed to work with dynamic data to simplify the engineering process of acquiring and processing signals. For example, if we wanted to clean up residual noise in the signal, we could implement a lowpass filter using the Express→Signal Analysis→Filter function. Like all Express functions, choosing this option creates a wizard to help the user implement a filter. Figure A.42 shows the assistant window for configuring a filter. The default type is a lowpass, Butterworth, 3rd-order filter with a cutoff frequency of 100 Hz. The filter behavior (lowpass, highpass, etc) is selected from a drop-down menu. The filter type (Butterworth, FIR, etc) is also selected from a drop-down menu. Corner frequencies are selectable, as is order. As an aside, the Butterworth filter is often the preferred filter if signal morphology is important because the Butterworth has a maximally flat passband behavior in the frequency domain, so it does not distort the desired signals.



Configure Filter [Filter]

Filtering Type: Lowpass

Filter Specifications
Cutoff Frequency (Hz): 100
High cutoff frequency (Hz): 400

Finite impulse response (FIR) filter
Taps: 29

Infinite impulse response (IIR) filter [✓]
Topology: Butterworth
Order: 3

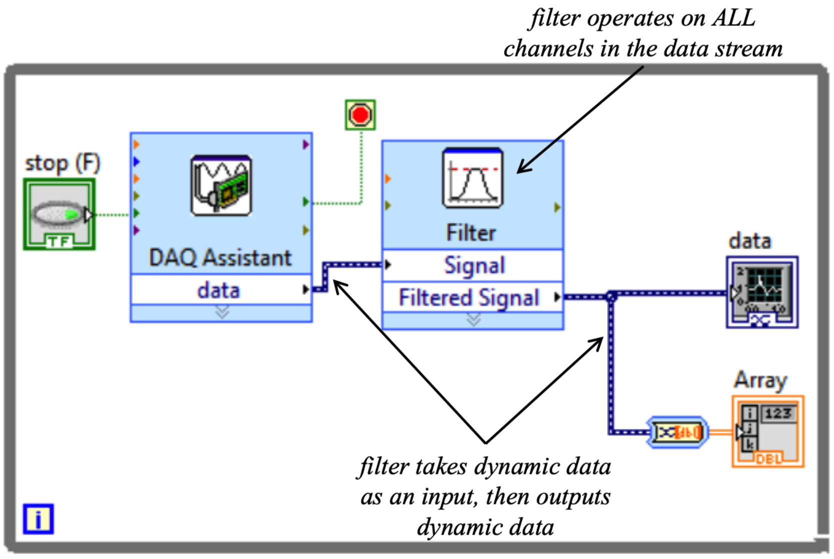
Input Signal
Result Preview

View Mode
Signals [✓] / Transfer function
Show as spectrum [ ]

Scale Mode
Magnitude in dB [ ]
Frequency in log [ ]

Figure A.42. ***The Express Filter Assistant:*** *The Express functions all come with helpful wizards (assistants) that guide the user through implementation.*

Once the filter has been configured (or whatever Express function is being implemented), it can be wired into the data stream as shown in Figure A.43. It is important to understand that the Express functions are aware of the nature of the data stream *because* the dynamic data stream contains information critical to performance (such as sample rate, or period). This includes the number of channels being acquired. The reader should take the leap in understanding and recognize that a single instance of the Express filter function will filter ALL the channels in the data stream, whether it is one channel, or eight channels. Each channel is filtered using the same filter, but one does not need to separate the channels and filter each one separately.


filter operates on ALL channels in the data stream
stop (F)
DAQ Assistant
data
Filter
Signal
Filtered Signal
data
Array
filter takes dynamic data as an input, then outputs dynamic data

Express Functions are oriented toward Dynamic Data

Figure A.43. ***Express Functions use Dynamic Data:*** *The Express functions use dynamic data and operate on all channels in the data stream using a single instance of the function. Express functions also output dynamic data for the next function in the data flow.*

A.3 Conclusion

As stated at the beginning of this appendix, this text is not oriented toward becoming an expert LabVIEW programmer. The LabVIEW programming environment is an extensive, highly-developed environment with many powerful tools. There is nothing that cannot be accomplished in this graphical programming language that can be accomplished in a textual coding language. This appendix has not covered every aspect of basic LabVIEW programming in depth. The student is encouraged to open each sub-palette from the Block Diagram and study the available functions (of particular interest would be the Numeric and Comparison palettes, but all the palettes have needed functions and the dedicated student would be wise to familiarize herself with each sub-palette). The same can be asserted for the Front Panel sub-palettes. There is some more detailed information about specific items covered in the Case Study chapters. However, this text is focused on the concepts of measurement and instrumentation. LabVIEW and National Instruments equipment provide an excellent launching point that allow users to quickly develop the basic programming skills needed to study this subject. Indeed, many industry applications use the combination of National Instruments hardware and software precisely because of the ease in getting started at a basic data acquisition level. Many universities also have National Instruments Student Ambassador programs in which a sponsored student teaches the LabVIEW basics course each semester. If the student is fortunate enough to have this program at his university, the short course is an excellent adjunct to an engineering class on instrumentation and measurement. Of course, another option mentioned at the beginning of this index is the National Instruments website: http://www.ni.com/getting-started/labview-basics/.