



WPI

ME 3902: Online Project-Based Engineering Experimentation

May 18th, 2020

A Major Qualifying Project (MQP) Report
Submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the
Degree of Bachelor of Science

Submitted by:

Benjamin Duquette

Cameron Gould

Nicolae Opincaru

Eric Solorzano

Tiffany Ta

Submitted to Project Advisors:

Professor John Sullivan, Worcester Polytechnic Institute

Professor Ahmet Can Sabuncu, Worcester Polytechnic Institute

This report represents the work of five WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the project program at WPI, please see:

<http://www.wpi.edu/Academics/Projects>

Abstract

This course is designed to develop experimental skills in engineering measurement methods, based on electronic instrumentation and computer-based data acquisition systems, such as the Raspberry Pi (primarily digital) and Arduino (primarily analog) microcontrollers. The online lectures are concerned with the engineering analysis and design as well as the principles of instrumentation, whereas the laboratory project-based modules afford the student an opportunity to use these devices in actual experiments. Lecture topics include discussions of standards, measurement and sensing devices, experiment planning, data acquisition, analysis of experimental data, and report writing. Laboratory experiments address mechanical (force/torque/strain measurements, motion/vibration measurements), energy (heat transfer, temperature, flow measurements), and materials measurements (materials processing, measurement of strain and position in mechanical testing of materials), and instrumentation. The course culminates with an open-ended project of the students' choosing. This open-ended project will illuminate the skills gained by the student to utilize multiple sensors and equipment to monitor and/or control physical situations.

Executive Summary

The purpose of this project was to address the need for ME 3901: Engineering Experimentation, a laboratory class in the WPI Mechanical Engineering undergraduate curriculum, to be modernized. In its current form, the class does not meet its objectives of teaching students the importance of instrumentation, data collection/analysis, and measurement techniques. This is due to a number of factors, such as the course material's lack of significant updates and its convoluted presentation. Additionally, the need to utilize data acquisition (DAQ) boxes both restricts a student's ability to perform work outside the laboratory setting and results in high maintenance and replacement costs the university must maintain to keep the equipment running. In order to address these issues, ME 3902: Online Project-Based Engineering Experimentation was created. It satisfies the same degree requirements as ME 3901, but the two are distinct enough that both could be taken if desired.

The primary difference between the two courses is the replacement of DAQ boxes in ME 3901 with Arduino and Raspberry Pi microcontrollers in ME 3902. Students in ME 3902 choose one of the two microcontrollers to work with, and all students receive a kit containing the materials needed to perform the experiments. These materials are provided by the university and purchased by students as would normally occur for classes requiring supplementary materials. The switch from DAQ boxes to microcontrollers accomplishes three objectives. First, it significantly reduces the financial cost needed for the university to run the course, since student-purchased materials help offset the price of ordering new microcontrollers and kit materials, which themselves are less expensive than the DAQ boxes. Second, it modernizes the course by exposing students to microcontrollers, which are more relevant to what engineering students entering today's workforce will be expected to understand. Third, it enables a much greater degree of flexibility for the students, as they are no longer confined to a designated laboratory space, enabling the class to be completed more easily.

ME 3902 was designed using a module-based structure. The modules cover an introduction to microcontrollers, temperature, proximity, motors, concentrations, strain, and motion. They were created by researching each topic to generate introductory modules that explain the module's main concept in detail, then broken down into individual explanations for each specific device and their corresponding experiments. The initial pilot class for ME 3902 was run in C term 2020. Our group acted as Peer Learning Assistants (PLAs) for the class, working in conjunction with a GLA, a TA, and the professor to address student questions during laboratory sessions. The existing class structure for ME 3901 was used, with alternating 1-hour in-class lectures and 2-hour laboratory sessions. The course culminated in an open-ended project which students created independently or in 2-person groups, using 3 of the sensors/devices from previous modules to design a system that performed a task of their choosing.

During the pilot class, student feedback was collected in order to understand how effective the course was. From this feedback, we have concluded that the primary areas of improvement are twofold - a need for the introduction module to have a more detailed explanation of the basics of Arduino/Raspberry Pi operation and setup, and an outline of basic electrical engineering principles equivalent to an ECE 2010 level. In the future, the class should ideally be offered online only and self-paced, once it has progressed to a point where the curriculum content is fully realized.

Acknowledgments

We would like to thank those who assisted our group in completing this project. Our advisors, Professor John Sullivan and Professor Ahmet Can Sabuncu, provided us with extensive guidance, feedback, and ideas throughout the span of this project. The engineering experimentation lab manager, Peter Hefti, assisted with the ordering of parts and components for the student kits and helped set up the lab space with the necessary tools. The GLA and TA for the course, Zachary Zolotarevsky and Mengqiao Yang, assisted us in running the laboratory sessions and providing guidance to the students for their experiments and projects. The students for the pilot version of this course provided valuable feedback which shaped our vision for the direction the course should take in future offerings.

Table of Contents

Abstract	2
Executive Summary	3
Acknowledgments	4
Introduction	10
Background	11
Figure 1: ME 3902 Canvas Modules	13
Overview Module Design	13
Module 1: The Microprocessor-Computer	14
LED Experiment	14
Figure 2A: Arduino Uno R3	14
Figure 2B: Raspberry Pi 4 Model B	14
Module 2: Temperature	14
Resistive Temperature Device (RTD)	14
Figure 3: Diagram of a Wire-Wound RTD	15
Infrared (IR) Temperature Sensor	16
Figure 4: Diagram of an IR Temperature Sensor	16
Thermocouple	17
Figure 5: Common Types of Thermocouples	18
Silicon Band Gap Sensor	19
Figure 6: Silicon Band Gap Sensor Max Efficiency vs. Band Gap Graph	19
Figure 7: TMP36 Silicon Band Gap Temperature Sensor	20
Thermistor	20
Figure 8: Example Thermistors	21
Figure 9: PTC Thermistor Control Circuit	21
Figure 10: Temperature Detection/Compensation NTC Thermistor Circuit	22
Figure 11: Ametherm NTC and PTC Thermistor Temperature Curves	23
Module 3: Proximity	23
Infrared (IR) Motion Sensor	23
Figure 12: Passive IR Sensor Diagram	24
Figure 13: IR Sensor Heat Source-Output Transmission Diagram	24
Figure 14: HC-SR501 PIR Sensor	25
Ultrasonic Sensor	25

Figure 15: Ultrasonic Sensor Ultrasonic Wave Interaction Diagram	26
Module 4: Motors	27
DC Motor	27
Figure 16: Graph of Basic Current Types	27
Figure 17: Brushed DC Electric Motor	28
Figure 18: ROB-11696 DC Motor	28
Servo Motor	29
Figure 19: Servo Motor Components	29
Figure 20: Servo Motor Diagram with Potentiometer	30
Figure 21: ROB-09605 Micro Servo Motor	30
Stepper Motor	31
Figure 22: Internal Stepper Motor Components	31
Figure 23: Stepper Motor Step Path	32
Figure 24: 28BYJ-48 Stepper Motor	32
Module 5: Concentrations	33
Gas Sensor	33
Figure 25: Gravity pH Sensor	33
Figure 26: Electrochemical Sensor	34
Figure 27: MQ-3 Gas Resistance Sensor	35
Module 6: Strain	35
Strain Gage	35
Figure 28: Schematic of Strain Gage With Test Specimen	35
Module 7: Motion	36
Accelerometer	36
Figure 29: ADXL345 Accelerometer Breakout Board	37
Figure 30: MEMS Accelerometer Schematic	38
Figure 31: Applications of MEMS Accelerometers	38
Module 8: Open-Ended Project	39
Methodology	40
Experiment Module Design	40
Module 1: The Microprocessor-Computer	41
LED Experiment	41
Figure 32A: LED Experiment Setup for Arduino	41

Figure 32B: LED Experiment Setup for Raspberry Pi	41
Module 2: Temperature	41
Silicon Band Gap Sensor	41
Figure 33: Fritzing Diagrams of Arduino and Raspberry Pi Silicon Band Gap Sensor Experiment	43
Thermistor	44
Figure 34: Fritzing Diagrams of Arduino and Raspberry Pi Thermistor Experiment	46
Module 3: Proximity	46
Infrared (IR) Motion Sensor	46
Figure 35: Physical Setup of IR Motion Sensor	48
Figure 36: Fritzing Diagram of Arduino IR Motion Sensor Experiment	48
Figure 37: Fritzing Diagram of Raspberry Pi IR Motion Sensor Experiment	49
Ultrasonic Sensor	49
Figure 38: Fritzing Diagrams of Arduino and Raspberry Pi Ultrasonic Distance Sensor Experiment	51
Module 4: Motors	52
DC Motor	52
Figure 39: Fritzing Diagrams of Arduino and Raspberry Pi DC Motor Experiments	54
Servo Motor	54
Figure 40: Fritzing Diagrams of Arduino and Raspberry Pi Servo Motor Experiments	56
Stepper Motor	57
Figure 41: Fritzing Diagrams of Arduino and Raspberry Pi Stepper Motor Experiments	58
Module 5: Concentrations	59
Gas Sensor	59
Figure 42: Fritzing Diagrams of Arduino and Raspberry Pi Gas Sensor Experiment	60
Figure 43: Gas Sensor Setup Code for Raspberry Pi and Arduino	61
Figure 44: Gas Sensor Running Code for Raspberry Pi and Arduino	62
Figure 45: MQ3 Sensor Resistance vs. Concentration Data	62
Module 6: Strain	63
Strain Gage	63
Figure 46: TAL221 Load Cell Nodes	64
Figure 47: Example Strain Gage Experiment Setup	65
Module 7: Motion	66
Accelerometer	66

Figure 48: Fritzing Diagrams of Arduino and Raspberry Pi Accelerometer Experiment	67
Module 8: Open-Ended Project	68
Figure 49: Example of a Plant Monitoring System	68
Results	69
ME 3902 Pilot Class-Theory	69
Figure 50: ME 3902 Pilot Class Student Academic Year Demographics	71
Module 1: The Microprocessor-Computer	71
Module 2: Temperature	72
Figure 51: Fritzing Diagrams of Arduino and Raspberry Pi Experiment with and without ADC	73
Module 3: Proximity	73
Figure 52: Cone Angle Diagram	74
Module 4: Motors	74
Module 5: Concentrations	75
Module 6: Strain	76
Figure 53: Strain Gage Used In ME 3902 Pilot Class	76
Module 7: Motion	76
Module 8: Open-Ended Project	77
ME 3902 Pilot Class in Practice	77
Figure 54: Summary of Student Questions by Module	78
Conclusions	79
Improve Module 1 Content	79
Diversify Types of Sensors Used	79
Focus on One Microcontroller Type	79
Create Independent Class Resources	80
References	81
Appendices	86
Appendix 1: ME 3902 Pilot Class Bill of Materials	86
Figure 55: ME 3902 Pilot Class Materials Purchased From Adafruit	86
Figure 56: ME 3902 Pilot Class Materials Purchased From Amazon	87
Figure 57: ME 3902 Pilot Class Materials Purchased From Blick	87
Figure 58: ME 3902 Pilot Class Materials Purchased From Home Depot	87
Figure 59: ME 3902 Pilot Class Materials Purchased From McMaster Carr	88

Figure 60: ME 3902 Pilot Class Materials Purchased From Sparkfun (Bulk Supplies)	88
Figure 61: ME 3902 Pilot Class Materials Purchased From Sparkfun (Sensors/Microcontrollers)	88
Appendix 2: ME 3902 Pilot Class Student Questions/Issues	89
Figure 62: ME 3902 Pilot Class Student Questions/Issues	91
Appendix 3: ME 3902 Pilot Class Experiment Code	91
Module 2: Temperature	91
Module 3: Proximity	96
Module 4: Motors	100
Module 5: Concentrations	107
Module 6: Strain	111
Module 7: Motion	115

Introduction






























In this growing age of technology, more and more students are choosing to attend college online. According to the National Center for Education Statistics, over 13% of all students in the US are currently enrolled in at least one distance learning class, and that number continues to grow every year [1]. This rise in online learning has been caused by a multitude of factors. For one, it costs the school less to offer the class online since it would only take a professor's salary to teach it rather than having to give up valuable classroom and lab space. For most schools, they tend to pass these savings on to the students, usually costing \$100-\$400 dollars a credit hour. This is very cheap by college standards and also eliminates any additional costs related to student housing or a meal plan [2]. Online classes can also be recorded and played back at any time. Due to a professor's availability, some classes might only be offered in certain parts of the year, which may be inconvenient for students with complicated schedules. Online schooling would improve this aspect of accessibility by allowing pre-recorded and professionally constructed classes to be offered at any time. Another accessibility issue stems from national borders and the difficulty of international education. To be educated at a US institution requires a student visa, flights and all of the complications of international travel and living. The utility of an online course designed to teach the concepts of engineering experimentation without the need of a physical classroom is large.

ME 3901: "Engineering Experimentation" is a required class for all students at WPI in the Mechanical Engineering degree program. The purpose of this class is to develop skills in methods used by modern engineers in parameter measurement, as well as skills relating to electronic instrumentation and computer-based data acquisition systems. The Accreditation Board for Engineering and Technology, Inc. (ABET) requires that mechanical engineering students be exposed to engineering experimentation in their degree programs for their institution be accredited, which this course satisfies [3]. Up to this point, most of the ME classes are strictly theoretical, providing students the numbers that they need to finish a problem but not explaining where they came from or how they were determined. For example, when doing a stress analysis problem, the students are given a value for the stress in the X, Y, and Z directions of the given beam. To many students' surprise, there is no machine or sensor to provide a direct reading for stress like this problem would suggest. You first have to measure the strain and use that measurement and the properties of your material to calculate stress. This is important for students to know so that when they get out into the field, they will not have unreasonable expectations for what values can and cannot be measured.

ME 3901 also teaches students that you cannot blindly trust the number that a sensor is outputting because a sensor reading is only as good as that given sensor's accuracy and sample rate. If you are designing a heating system for a chemical plant that needs to keep a certain chemical at 105.3 °C with an error of only +/- .5°C for a certain reaction to take place, you would use a very different quality of sensor than you would for residential air conditioning. This class focused on 4 main types of sensors: pressure, temperature, strain, and acceleration. All of the labs for this class heavily rely on the use of very expensive data acquisition boxes (DAQs) and LabVIEW software. This approach is great for preparing students to work for a large and successful company with a lot of resources to provide to engineers. However, it does not prepare students to work at younger companies with tighter budgets that are still trying to get a solid foothold in the market. ME 3902 was designed to meet these challenges.


Background

This background summarizes the introductory/overview modules used in the ME 3902 pilot class. The modules each discuss distinct topics, which are put into practice through several experiments. The Methodology section describes these experiments in detail. Figure 1 below shows a detailed outline of the entire course material from its Canvas page [4]. Additionally, each subsection discusses the specific sensors/devices chosen for use in the pilot class with rationale. The Appendices section contains the bill of materials for these sensors/devices and all other items which were purchased for the pilot class (Appendix 1). These are comprehensive enough for the class to be replicated in its entirety.


▼ Module 01: The Micro-Processor-Computer (Level 1)	▼ Module 02: Temperature - Is it hot in here? (Level 1)
 Discussions Related to Module 01 <small>Jan 17 3 pts</small>	 020 - Temperature Overview
 001 - Course Logistics	 Temperature Measurement Overview
 001-CourseLogistics.pdf	 Resistive Temperature Device (RTD) Sensor Overview
 002 - The Arduino	 Measuring Temperature Using RTDs
 002-The Arduino.pdf	 Infrared (IR) Temperature Sensor Overview
 003-The Raspberry Pi	 Thermocouple Sensor Overview
 003-The RaspberryPi.pdf	 Measuring Temperature Using Thermocouples
RPi Logistics	 Silicon Band Gap Sensor Overview
Connect to the WPI Networks	 Silicon Band Gap Experiment (Arduino)
Use of Real VNC for RPi Control	 Silicon Band Gap Experiment (Raspberry Pi)
Preliminary Python Programming	 Thermistor Sensor Overview
Laboratory Exercises (perform at least one of them)	 Measuring Temperature Using Thermistors
 Written Report Guidelines	 Sample Codes for this Thermistor
 004-Arduino Experiment	 Thermistor Experiment (Arduino)
 004-ArduinoExperiment.pdf	 Thermistor Experiment (Raspberry Pi)
 PythonExampleDocumentation.pdf	
 005-Raspberry Pi Experiment	
 005-RaspberryPiExperiment.pdf	
 006-BasicCircuitry.pdf	


▼ Module 04: Motors (Level 2)


 Motors Overview


 Information on Gears


 AC Motor Overview

 DC Motor Overview

 DC Motor Datasheet


 DC Motor Experiment (Arduino)


 DC Motor Experiment (Raspberry Pi)

 Servo Motor Overview


 Servo Motor Experiment (Arduino)


 Servo Motor Experiment (Raspberry Pi)

 Stepper Motor Overview


 Stepper Motor Datasheet (DM0620)

 Stepper Motor Microchip Web Series

 Stepper Motor Experiment (Arduino)

 Stepper Motor Experiment (Raspberry Pi)


▼ Module 03: - Proximity - Where am I? (Level 1)

 Proximity Overview

 Infrared (IR) Motion Sensor Overview

 IR Motion Sensor Experiment (Arduino)

 IR Motion Sensor Experiment (Raspberry Pi)

 Ultrasonic Sensor Overview

 Ultrasonic Sensor Experiment (Arduino)

 Ultrasonic Sensor Experiment (Raspberry Pi)

▼ Module 05 - Concentrations - Who Farted! (Level 2)	
Concentrations Overview	
Sensor Types Overview	
Alcohol Experiment (Breath Analyzer) Arduino	
Alcohol Experiment (Breath Analyzer) Raspberry Pi	
OPTIONAL: Methane Experiment (Fart Detector) Arduino	
OPTIONAL: Methane Experiment (Fart Detector) Raspberry Pi	
▼ Module 06: Strain (Level 3)	
Strain Overview	
Strain In-Depth	
Wheatstone Bridge Overview	
Pressure Transducer Overview	
Strain Gage Info	
Strain Gage Experiment (Arduino)	
Strain Gage Experiment (Raspberry Pi)	
▼ Module 07: Motion - Are My Teeth Chattering? (Level 3)	
Acceleration Overview	
Motion/Vibration/Acceleration/MEMs Info	
OPTIONAL: Accelerometer Experiment (Arduino)	
OPTIONAL: Accelerometer Experiment (Raspberry Pi)	
▼ Module - 08: Open-Ended Experiment - The Summit (Level 4)	
Open-Ended Experiment Overview (Use 3 or more sensors)	
Plant Maintenance	
Alcohol Breath Detector with Car Key Lock	
CO Detector with Window Controls and Alarms	
Pen Plotter Apparatus	

Figure 1: ME 3902 Canvas Modules

Overview Module Design

The overview modules acted as an introduction to each topic. Before starting each experiment, students were to read over both the introduction to the overall topic and the introduction to the specific sensor/device being used for that experiment. These modules acted as brief explanations of the topic in order to provide context for the experiment and establish the broad objectives (measuring temperature, powering a motor, etc.) General use cases, governing equations, basic specifications, and operating principles were discussed for each sensor/device. It was necessary to balance detail with accessibility and ease of understanding, so that students had sufficient information to work with but were not overwhelmed with unneeded details. Various animations, diagrams, and plots were implemented in order to better visualize the concepts being presented.

Module 1: The Microprocessor-Computer

LED Experiment

This module was designed to provide students with an explanation of the basics of Arduino and Raspberry Pi microcontrollers, discussing their layouts and explaining the differences between the two devices. Figures 2A and 2B below show the specific models for each microcontroller that were used in the pilot class, the Arduino Uno R3 [5] and the Raspberry Pi 4 Model B [6]. An important distinction between both microcontrollers is that the Arduino primarily uses analog signals, while the Raspberry Pi primarily uses digital signals. This meant that some labs required different wiring and coding setups between both microcontrollers. In order to distinguish between the two, subsequent diagrams referencing Arduino in this paper will be labeled (ARD), while diagrams referencing Raspberry Pi will be labeled (RPI).

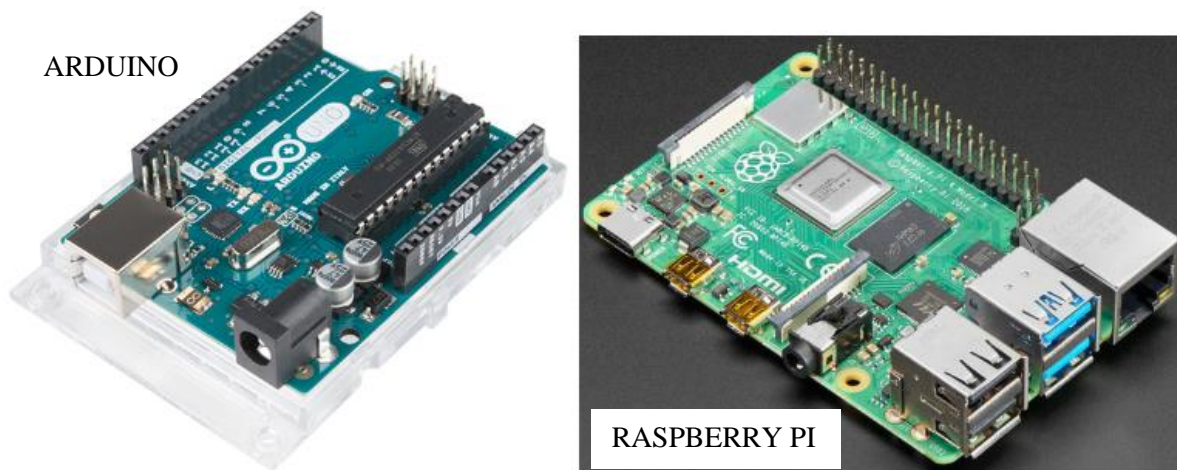


Figure 2A: Arduino Uno R3

Figure 2B: Raspberry Pi 4 Model B

Module 2: Temperature

Resistive Temperature Device (RTD)

Resistive temperature devices (RTDs) are probe-like apparatuses that measure temperature based on changes in electrical resistance. Essentially, they act as a variable resistor whose resistance is directly proportional to changes in temperature. They are best known for their good accuracy, even better than that of thermocouples, and have a wide temperature range (approximately -200 to 850°C). Generally, RTDs consist of a length of finely coiled wire wrapped around a non-conductive core, typically made of ceramic or glass, inside a protective sheath. The wire coil is made of a pure temperature-sensitive material, such as platinum (most common), nickel, or copper, in order to measure the resistance. Figure 3 below details the parts of a typical RTD [7]. This one contains platinum as its conducting element and stainless steel as a sheath.

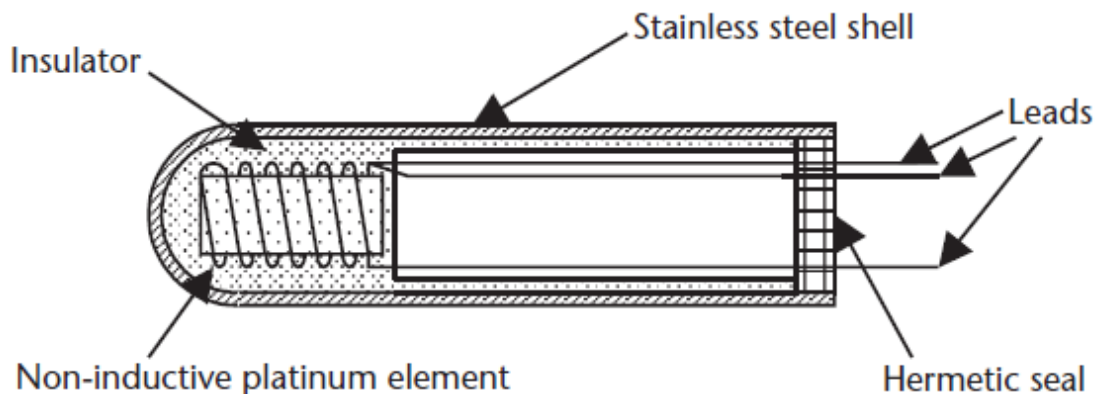


Figure 3: Diagram of a Wire-Wound RTD

To measure temperature with the RTD, an electric current is run through the leads into the wire coil, and the change in the coil's resistance is determined based on the voltage and current. This resistance will vary depending on the temperature of the coil--as the coil's temperature increases, the resistance will also similarly increase. Different materials tend to have different temperature coefficients of resistance (TCR), which influence how much the resistance will increase or decrease with changes in temperature.

RTDs are available in two, three, and four-wire configurations, which provide different levels of measurement accuracy. Models that contain more wires tend to be more accurate when converting the resistance to a temperature reading since they have minimized the lead wire resistance changes in their measurement connections. At the same time, however, this means that the circuits used to produce a reading have to compensate for the increasing complexity resulting from additional connections. More wires in the RTD will eliminate more resistance from the circuit and change the magnitude of resistance change per degree of temperature. Therefore, more complex bridge circuits containing more resistors and other elements will be needed in order to compensate for the changes in resistance.

While the team chose to include a section describing the RTD within the temperature module so that students would be aware of its workings and applications, the team chose to omit this particular temperature sensor from the course experiments and therefore from the laboratory kits. This was mainly due to the extremely high costs of platinum RTDs, which are the most common types on the market and can easily be at least \$50. Since the team sought to keep the kits as affordable for students as possible, with a very limited budget for each sensor module, this particular sensor was not included in the kit.

Infrared (IR) Temperature Sensor

Infrared technology has a dual function; it can detect temperature and whether or not an object crosses its field of view. Infrared is therefore capable of being used for both temperature detection and as a motion sensor. Unfortunately, the same sensor cannot be used for both temperature and motion functions. You may have seen and used IR thermometers, which work by focusing an incoming beam of IR light from the object in question. This IR radiation is absorbed by an internal component called a thermopile, essentially a bundle of thermocouples, which produces a voltage relative to the heat it absorbs. This voltage is then interpreted by an integrated circuit of the thermometer and displayed for the operator's use. Those are the basic workings of handheld IR thermometers, although oftentimes IR thermometers are used without the beam focusing components. For example, temperature sensors used in computers could be using similar techniques to control fan speeds for the computer cooling system. Overall the principle of operation is that IR sensors absorb IR either passively or through a focusing lens. This heat is turned into electricity through a series of components. The higher the heat, the higher the electricity and as such we have a higher display temperature, corresponding to the heat measured originally. Figure 4 below shows a diagram of an IR sensor [8].

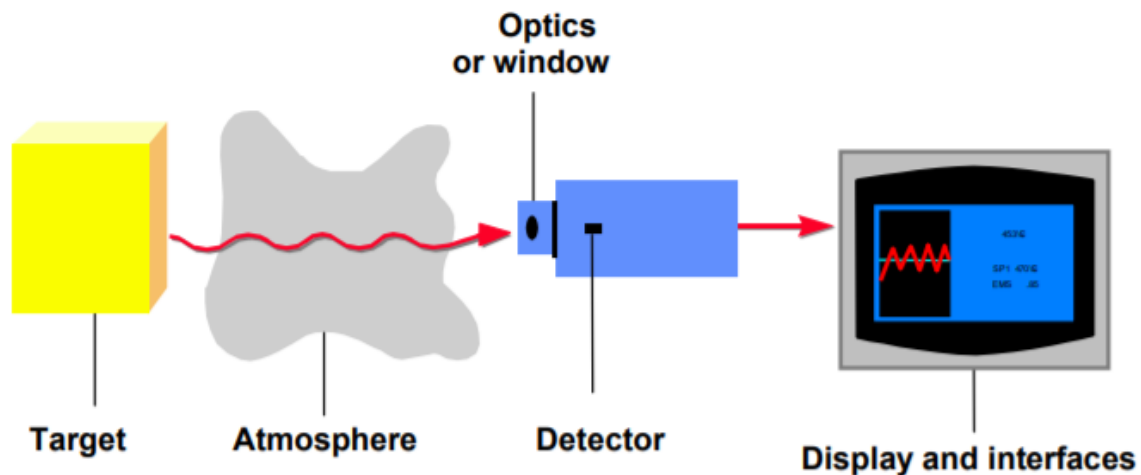


Figure 4: Diagram of an IR Temperature Sensor

There are many reasons to use IR sensors for temperature measurements. The main advantage is the ability to measure temperatures without need for physical contact. This reduces wear on the sensor and allows for the measurement of many different temperatures without specialized temperature equipment. For example, they are used quite often in monitoring the temperature of car brakes under loads during road tests, as well as precise temperature monitoring in applications such as laser welding processes. IR sensors also have a large temperature range due to using the gradient between the object and the surroundings to determine its temperature.

The accuracy and reliability of these beam-based sensors come into question when looking at objects that do not reflect or absorb IR light with unknown emissivity. Due to their high emissivity, IR sensors have difficulty taking accurate measurements of shiny, metallic surfaces unless they have been adjusted for the emissivity values of those surfaces. This is done by calibrating the output of the sensor to the emissivity of the surface (information on emissivity values could be found itemized in different tables). Another weakness to IR thermometers is their inability to measure through visibly transparent material. Substances such as glass are transparent to the visible spectrum yet emit and absorb IR radiation that is then picked up by the sensor. Expensive focusing lenses are also necessary to measure small targets due to the convergence of the beam coverage. An IR thermometer cannot measure something that is not IR-visible to the sensor, and dust or smoke particles in the air could reduce the accuracy of the measurement.

The team chose to omit IR sensors from the kits included in the lab for a variety of reasons. First of all, the cost of low-level IR sensors was well above the budget of each kit. The operation of the tested IR sensor was fairly limited, coupled with the lab-testing limitations of testing temperature against mercury thermometers in varied temperature water, it became clear that an experiment with an IR temperature sensor would be more of a hindrance to the course.

Thermocouple

Thermocouples are one of the most common instruments used to measure temperature in the engineering field due to their low cost, fast response time, and their ability to monitor a wide range of temperatures. Thermocouples measure temperature by taking advantage of what is known as the Seebeck or thermoelectric effect. This situation is when two dissimilar metals are connected at one end, they produce a small electric voltage when a temperature difference exists between the joined junction location and the other ends of the dissimilar metals. The voltage difference created by the temperature difference is highly nonlinear. The National Institutes of Standards and Testing (NIST) provides polynomial approximations of temperature as a function of voltage but generally the thermocouples require a 6th to 10th order polynomial to describe the temperature behavior. Choosing different types of metals will produce different changes in voltage as the temperatures change, which is why there are different types of thermocouples. A few common types are shown in Figure 5 below [9]. This figure shows both the European and American standards for coloring, such that one can tell which type of thermocouple is used just by looking at its wiring. It also lists the temperature ranges and metals types used in each type of thermocouple.

ANSI Code	ANSI MC 98.1 Color Coding		Alloy Combination		Maximum T/C Grande temp. range	EMF(mv)Over Max.temp.range	IEC 584-3 Color Coding	IEC Code
	Thermocouple	Extension	+ Lead	- Lead				
K			NICKEL-CHROMIUM Ni-Cr	NICKEL-ALUMINIUM Ni-Al	-270 to 1372°C -454 to 2501°F	-6.458 to 54.888		K
J			IRON Fe (magnetic)	CONTANTAN COOPER-NICKEL Cu-Ni	-210 to 1200°C -348 to 2193°F	-8.095 to 69.553		J
T			COPPER Cu	CONTANTAN COOPER-NICKEL Cu-Ni	-270 to 400°C -454 to 752°F	-6.258 to 20.872		T
E			NICKEL-CHROMIUM Ni-Cr	CONTANTAN COOPER-NICKEL Cu-Ni	-270 to 1000°C -454 to 1832°F	-9.835 to 76.373		E
N			NICROSIL Ni-Cr-Si	NISIL Ni-Si-Mg	-270 to 1300°C -450 to 2372°F	-4.345 to 47.513		N
S	NONE ESTABLISHED		PLATINUM-10% RHODIUM Pt-10%Rh	PLATINUM Pt	-50 to 1768°C -58 to 3214°F	-0.236 to 18.693		S
R	NONE ESTABLISHED		PLATINUM-13% RHODIUM Pt-13%Rh	PLATINUM Pt	-50 to 1768°C -58 to 3214°F	-0.226 to 21.101		R
B	NONE ESTABLISHED		PLATINUM-30% RHODIUM Pt-30%Rh	PLATINUM-6% RHODIUM Pt-6%Rh	0 to 1820°C 32 to 3308°F	0 to 13.820		B

Figure 5: Common Types of Thermocouples

The team felt that it was important to include thermocouples as part of the kit for a few reasons. The first is their common use in industry especially in startup companies needing to get temperature data fast and cheaply. We also thought it was important for students to be able to both read thermocouple tables as well as use NIST data and see that they were getting the same result. They were also a cheap addition to the box, so they added a lot to the course for a small amount of cost. Normally, you would need some kind of amplifier to go along with a thermocouple since the voltages that they produce are so small but since the team chose to include a 16 bit ADC in the kit they were able to cut this additional cost out and give more utility to the another component already included in the box. Another good reason to include a thermocouple in the box is that they have a lot of different coding libraries available for all the different types of thermocouples. This means that if a student did not understand one library that was provided, they could easily find another and complete the lab with that code instead. This was extremely important especially for the first few modules because this class acted as some students' first introduction into coding so having as many possible tools to help them was very important.

Silicon Band Gap Sensor

A silicon band gap sensor (SBGS) is a device commonly used for measuring temperature in electronic devices. Unlike other temperature sensors, the SBGS is capable of producing a voltage independent of outside sources. Furthermore, it can be implemented in tandem with silicon integrated circuits, which enables simultaneous manufacturing, as opposed to other sensor-circuit systems, which are assembled separately. This means that SBGS systems often use less material and perform better than other temperature sensing methods.

The driving principle of silicon band gap sensors is the concept of the band gap. Band gaps represent ranges of energy in a solid material where it is impossible for electron states to exist, lying below the conduction band (lowest range of vacant electron states) and above the valence band (highest range of energies where electrons are typically present). The band gap, therefore, indicates the level of energy needed for electrons to transition between these two bands, and by extension the ability of the material to conduct electricity. Silicon is used for most band gap sensors because its band gap is approximately 1.1 electronvolts (eV), meaning that it is one of the most efficient semiconductors available. One application of band gap sensors is demonstrated by the Shockley-Queisser limit, which characterizes the relationship between the maximum efficiency of single-junction solar cells and band gap length. This relationship can be approximated as a bell curve, and silicon lies very close to the peak of this bell curve as shown in Figure 6 below [10]. This demonstrates that silicon is a very ideal material to use for band gap sensors due to its semiconducting properties.

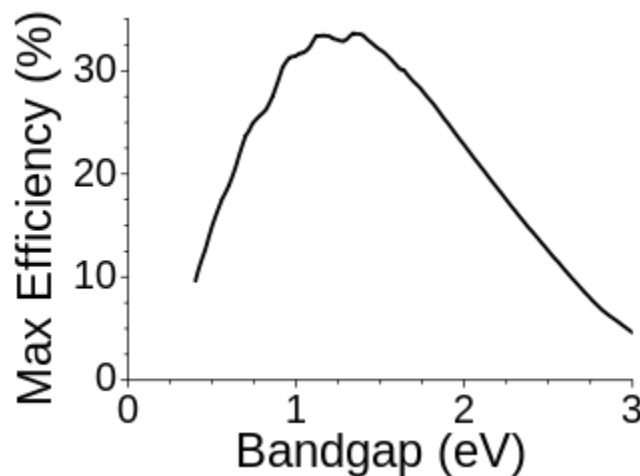


Figure 6: Silicon Band Gap Sensor Max Efficiency vs. Band Gap Graph

When using a silicon band gap sensor, temperature is measured using a diode in the sensor. The forward voltage of this diode is temperature-dependent, so voltage readings from the sensor's voltage output prong can be converted into temperature readings using a governing equation. This equation depends on the specific sensor being used and is provided in manufacturer datasheets. Silicon band gaps can typically measure temperatures in a range from -70°C to between 200-250°C. Above this upper range, leakage currents become significant enough in magnitude that the accuracy of the measurements becomes insufficient. If the user wishes to accurately measure temperatures above this limit, other materials must be used in place of silicon.

Figure 7 below shows the TMP36 silicon band gap temperature sensor that was selected for the student kit [11]. This model was chosen due to its simplicity, as it does not require any external calibration to establish its ideal temperature range and only has 3 prongs for power, ground, and voltage output.



Figure 7: TMP36 Silicon Band Gap Temperature Sensor

Thermistor

The thermistor, short for “thermal resistor,” is one of the most used temperature reading devices in most electronic systems today. Thermistors are similar to thermocouples in which they measure temperature, but thermistors fluctuate in electrical resistivity in response to a temperature change. They are generally used in electrical systems when a temperature change can be a benefit or a detriment to the rest of the system. From car engines to computer systems, thermistors help maintain the efficiency and health of our most beloved machines. There are two common types of thermistors: Positive Temperature Coefficient (PTC) thermistors and Negative Temperature Coefficient (NTC) thermistors. Figure 8 below shows some examples of thermistors [12].



Figure 8: Example Thermistors

In a PTC thermistor, the resistance increases when the temperature increases. PTC thermistors are more commonly used in heating applications where heat must be added to a system in order for it to function properly. An everyday application is in the heating of automobiles, where PTC thermistors will allow heat from the engine into the cabin of a car. If the thermistor is in a colder climate when the car is started, the thermistor will be at a low temperature, allowing heat from the engine to flow into the cabin. Once the temperature increases in the cabin and in this thermistor, the resistance in the circuit will increase, eventually cutting off heat from the engine to the car. This allows for less energy to be used in the heating process, as engine heat can be utilized for this purpose rather than wasting energy by using the car's normal heater.

Another more common application is in switch-mode power supplies (SMPS) used in most hand-held electronic devices. When they are activated a high inrush current is applied to the entire circuit, which if left at a high flow rate, can damage the power supply entirely. The PTC thermistor is used to help regulate the current in the system based on the temperature increase. This way, the resistor isn't overloaded from the current flow and the temperature in the power supplies doesn't melt itself. Figure 9 below shows a control circuit containing a PTC thermistor, demonstrating this principle [13]. Other applications include prevention of insulation or circuit damage and wax expansion.

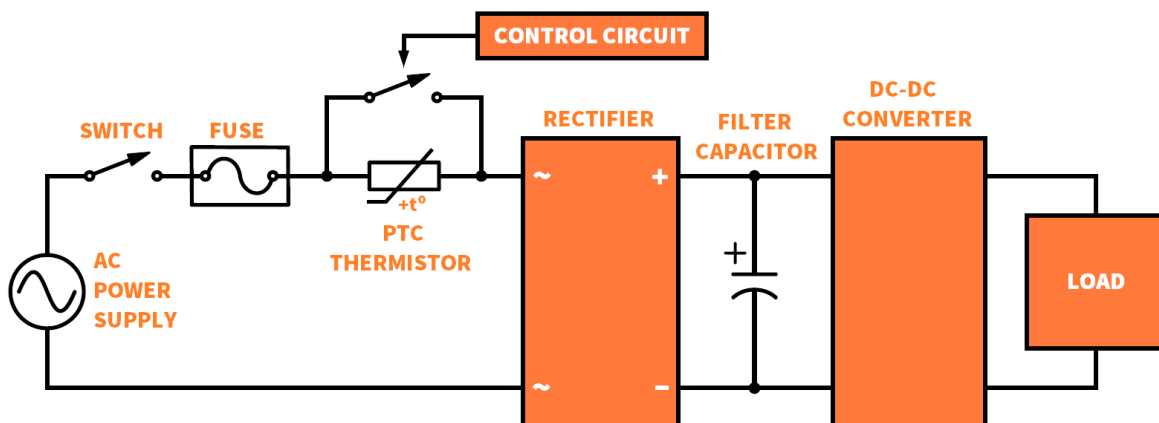


Figure 9: PTC Thermistor Control Circuit

In an NTC thermistor, the resistance decreases when the temperature increases. NTC thermistors are more commonly used in cooling applications in which heat must be removed from a system in order for it to function properly. An everyday application is in computer cooling fans, in which PTC thermistors will reduce heat inside a PC by turning on a cooling fan when it gets to a certain temperature. The thermistor is wired to cooling fans inside most PCs and will allow an electrical current to flow through the cooling fan circuit when the CPU generates enough heat. Other applications include automobile coolant temperature sensors, digital thermostats for battery packs, inrush current limiters and 3D printers. With smaller and more compact CPUs powering today's electrical systems, NTC thermistors play a much bigger role in removing heat and allowing these systems to run more efficiently. Figure 10 below showcases this with NTC thermistors integrated into circuits used for temperature detection and compensation [14].

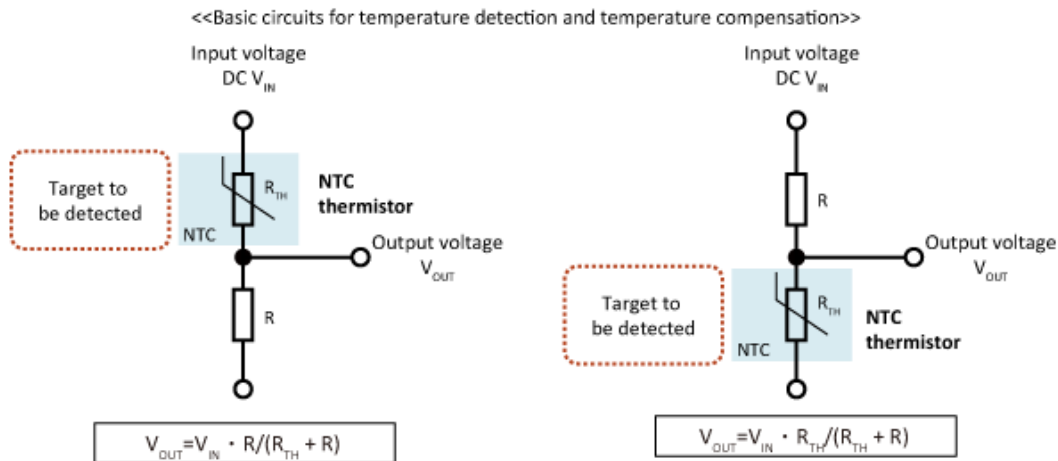


Figure 10: Temperature Detection/Compensation NTC Thermistor Circuit

The normal operating temperature range in most thermistors is generally between -50 and 150 degrees Celsius, in which the resistor over temperature change is exponential rather than linear. Although the temperature range is low compared to RTDs and thermocouples, thermistors are easier to use in simple engineering experiments, utilizing temperatures from freezing point water to boiling point water. The temperature ranges for a specific thermistor are generally disclosed by the manufacturer but can be found through a simple cold water and boiling water experiment. The graph would be flipped if a PTC thermistor was measured instead. Figure 11 below shows NTC and PTC thermistor temperature reference curves from Ametherm [15]. Most thermistors cost anywhere from \$2 to \$10 depending on the temperature range and accuracy of a temperature measurement. With startup companies only having so many resources, it is important for you as the newly hired engineer to figure out how to do tests and finish company work on a budget. While you might not have as high of a temperature range or as accurate of a temperature reading, you can fill a classroom with thermistors for about \$100.

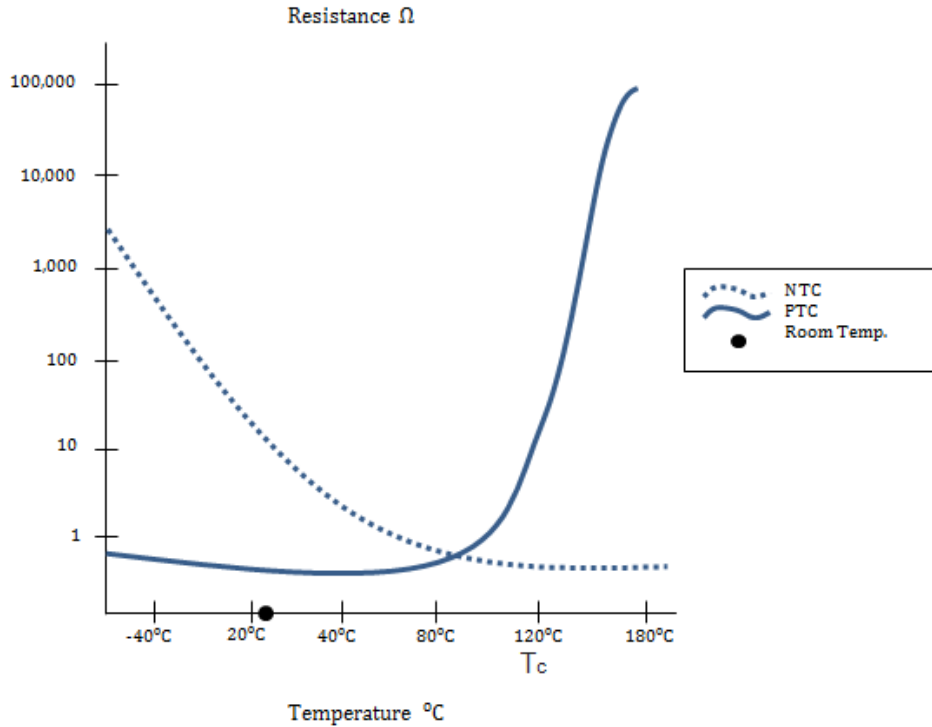


Figure 11: Ametherm NTC and PTC Thermistor Temperature Curves

Module 3: Proximity

Infrared (IR) Motion Sensor

While you've heard of and used infrared technology previously to measure temperature, the same technology could be used to detect motion, or more specifically detect if objects cross the path of the sensor. You might have seen infrared sensors near certain doors, like those leading into the first floor of Founders. If you pay attention, you'll see its LED light red when you approach and hear a click. Experimenting further with the approach to the sensor will allow you to figure out its range and angle of sight. Another key thing to notice while experimenting is the fact that the sensor will most likely not trigger if moving slow enough. Why would this happen? To find out, a look into the internals of these sensors is necessary.

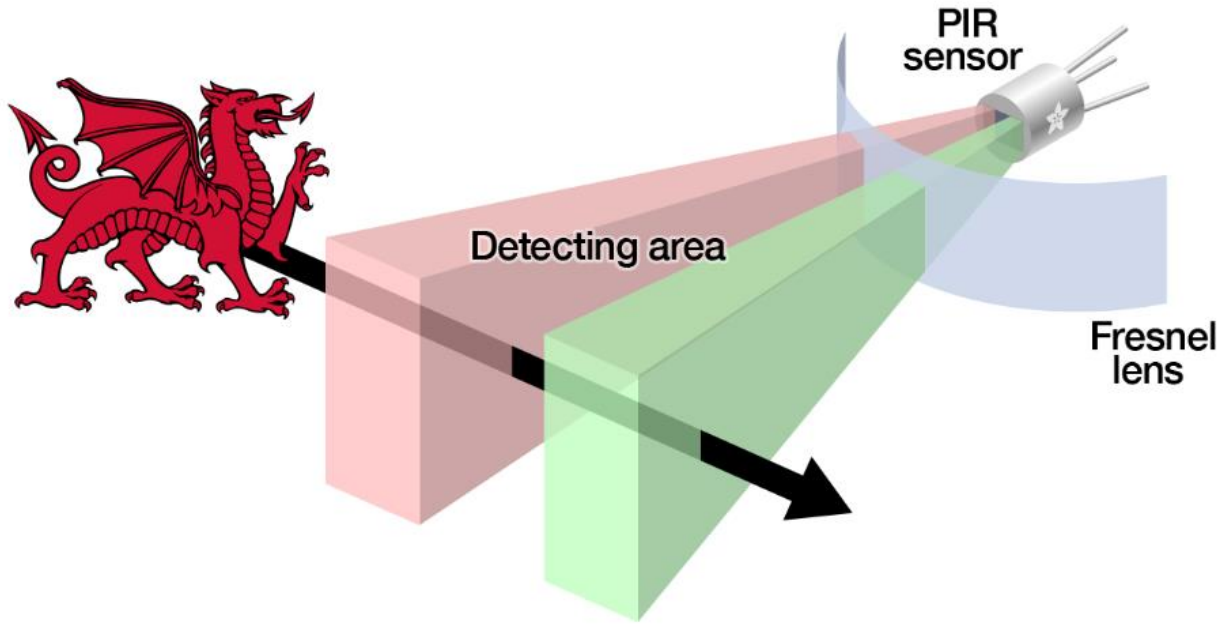


Figure 12: Passive IR Sensor Diagram

When talking about Passive Infrared Sensors (PIR), their inner workings are similar to temperature reading sensors, except that they interpret their data differently. As seen in Figure 12 above, there are two detection areas that are being read by the sensor [16]. The areas are read by producing a positive voltage change, and when it interrupts the second detecting area it produces a negative voltage change. When not detecting any other bodies of radiation, both areas of the sensor passively measure the background radiation of the surroundings and no voltage differential forms. When a body crosses the area, it creates a differential voltage, as seen in Figure 13 below [17]. This differential voltage pattern is then interpreted as movement by the circuitry of the IR sensor and triggers any following actions.

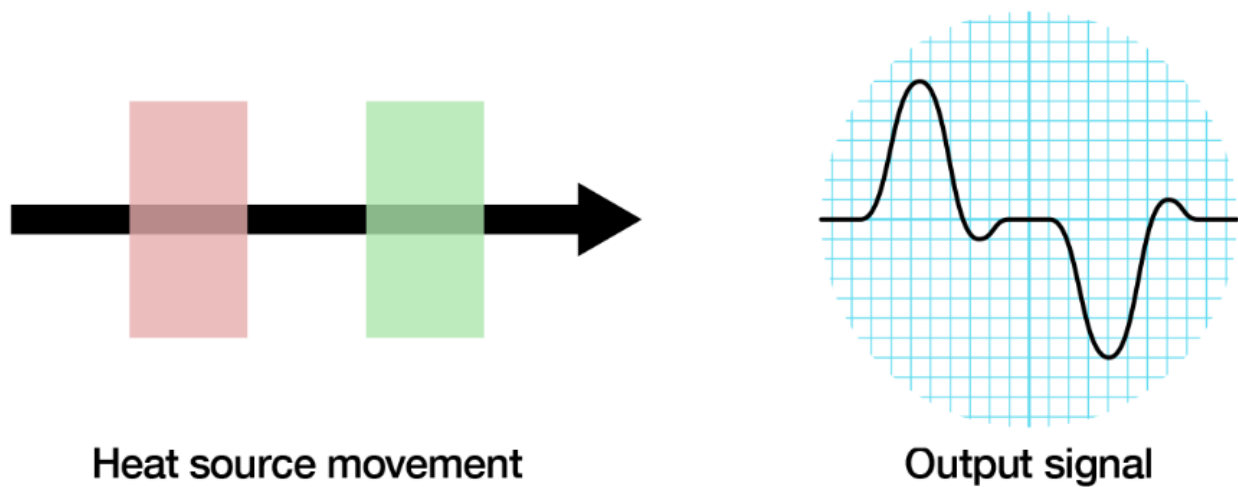


Figure 13: IR Sensor Heat Source-Output Transmission Diagram

When it comes to utilizing these sensors, there are some advantages and some disadvantages. Passive infrared technology can detect motion of objects, but it cannot discriminate between what it detects. As you've most likely seen with your garage lights at night, a cat passing the sensor is just as viable of a trigger as you pass it. Another issue is the fact that it cannot determine if there are more than 1 object in its field of vision. PIR sensors cannot operate in an environment greater than 35 degrees C, are insensitive to slow-moving objects and while having effective line of sight, it has issues detecting motion in corners. As a security sensor, they are a liability since thieves can utilize the slotted detection zone to avoid detection. They also cannot detect motion through glass or any other IR emitting substance.

The team chose to use the HC-SR501 PIR sensor due to its low cost, availability and reliability in function. Since this was a digital sensor, it could be hooked up directly to the Raspberry Pi without any need for an ADC, which simplified the experiment. External potentiometers were also included, which easily allowed students to adjust the sensitivity and time delay of the sensor. Figure 14 below shows the HC-SR501 PIR sensor [18].



Figure 14: HC-SR501 PIR Sensor

Ultrasonic Sensor

An ultrasonic sensor is a device that measures the distance of objects by using ultrasonic waves. Typically, it will use a transducer that converts electrical impulses into ultrasonic sound waves to send and receive. These waves will then bounce back from any objects within their range and return to the sensor, which then detects the returning waves and determines the amount of time between sending out the ultrasonic waves and sensing them again. This phenomenon is visualized below in Figure 15 [19].

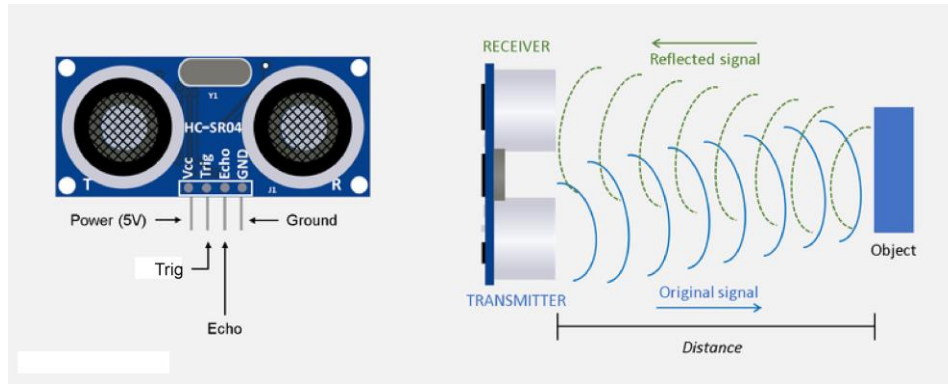


Figure 15: Ultrasonic Sensor Ultrasonic Wave Interaction Diagram

There are two main types of ultrasonic transducers that utilize different methods of converting the incoming electrical impulses to ultrasonic sound waves: piezoelectric and electrostatic. A piezoelectric transducer uses a crystalline or ceramic material to take advantage of the piezoelectric effect, a phenomenon where an electric current is created by applying mechanical stress onto a material. The crystal or ceramic is cased in a metal layer and is stimulated by a signal, which causes it to grow or shrink. This creates an ultrasonic pulse that can then be used to measure proximity. Electrostatic transducers are made up of two plates, an immovable one typically made of aluminum and a moveable one made of gold coated Kapton, a type of insulating material. The plates are drawn closer to each other when they receive a signal, which then causes an ultrasonic impulse.

Ultrasonic sensors are widely used because of their relatively good accuracy for their ranges, as well as being extremely affordable and simple to set up. Additionally, they also tend to not be as affected by external conditions, such as air clarity. Despite their accuracy, however, the ranges of ultrasonic sensors tend to be extremely limited, typically being up to 5 meters (about 16 feet). They are best used to measure the distance of objects they are facing perpendicularly to and may give inaccurate readings when detecting an object from an angle. Additionally, they cannot distinguish from different object sizes, since any object that the ultrasonic waves hit will return back to the sensor regardless of size. Generally, this shortcoming is combated by either installing multiple sensors or having the sensors rotate in place.

For the class's laboratory kit, the team chose to use the HC-SR04 ultrasonic distance sensor since it was an extremely affordable option that was relatively simple to wire, both to the Arduino and the Raspberry Pi. All of the pins could be wired directly to the Arduino, while the Raspberry Pi only required a simple voltage divider in order to utilize the sensor since the HC-SR04 operates digitally. The sensor model also contained a lot of online documentation for the code required to operate it, which was fairly straightforward, so students would not have to struggle as much to ensure that the sensor worked properly.

Module 4: Motors

DC Motor

A DC motor is the most fundamental type of electric motor. Its driving principle is the use of a direct current (DC) to provide power to whatever system it is connected to. Unlike alternating current (AC), which reverses direction and fluctuates in magnitude periodically, direct current flows in one direction at a constant magnitude. Figure 16 below shows a graphical representation of several basic types of currents used to power systems [20]. As shown here, direct current is constant over time, while alternating current changes in a sinusoidal manner over time.

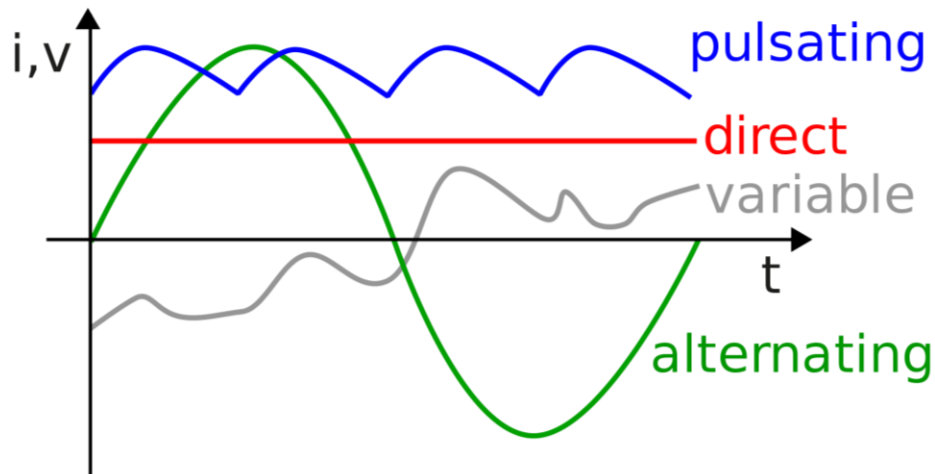


Figure 16: Graph of Basic Current Types

DC motors can be brushed or brushless (each uses a different method of generating torque to spin the motor). Brushed motors such as the one displayed in Figure 17 below directly generate torque via stationary and rotating magnets, along with physical commutation via an internal commutator [21]. Brushless motors also use magnets, but instead implement motor control via electronic commutation. Brushless motors are favored over brushed motors in most industrial applications due to their reduced maintenance and greater ease of control, but brushed motors are still sufficient for low-end applications such as the DC motor lab exercise.

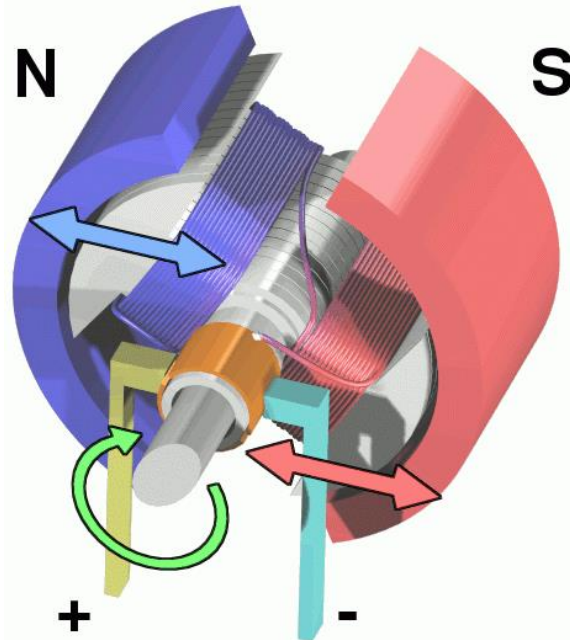


Figure 17: Brushed DC Electric Motor

DC motors are useful for applications which require large amounts of power to be delivered quickly. These applications include battery charging, fuel cells, automobiles, and telecommunication. Modern manufacturing processes have allowed these motors to be built rapidly and inexpensively, especially for more applications not requiring a significant amount of torque, such as children's toys. Figure 18 below shows the ROB-11696 DC Motor that was selected for the student kit [22]. This model was chosen due to its very low price point relative to its advertised performance.



Figure 18: ROB-11696 DC Motor

Servo Motor

Unlike other types of motors, servo motors are automatic devices, relying on negative feedback in order to function. This negative feedback is typically delivered via encoders and is used to correct the mechanism being powered in order for it to produce the desired result. This is most commonly seen in applications utilizing radio control, such as RC cars, where servo motors are used to enable control of the steering mechanisms.

Figure 19 below shows many of the internal components of a typical servo motor [23]. It consists of a central motor driving a gear reduction train, with the output shaft connected to a potentiometer. This potentiometer is visible in Figure 20 below [24]. The same drive shaft connected to the potentiometer has a horn attached to it, which can vary in shape (the horn present here is the black, X shaped piece). This motor-gear-potentiometer interface is neatly packaged in a central housing. The servo motor is powered using a three-wire interface, with one wire for the power input, one for the ground, and one for transmitting pulse width modulated (PWM) signals.



Figure 19: Servo Motor Components

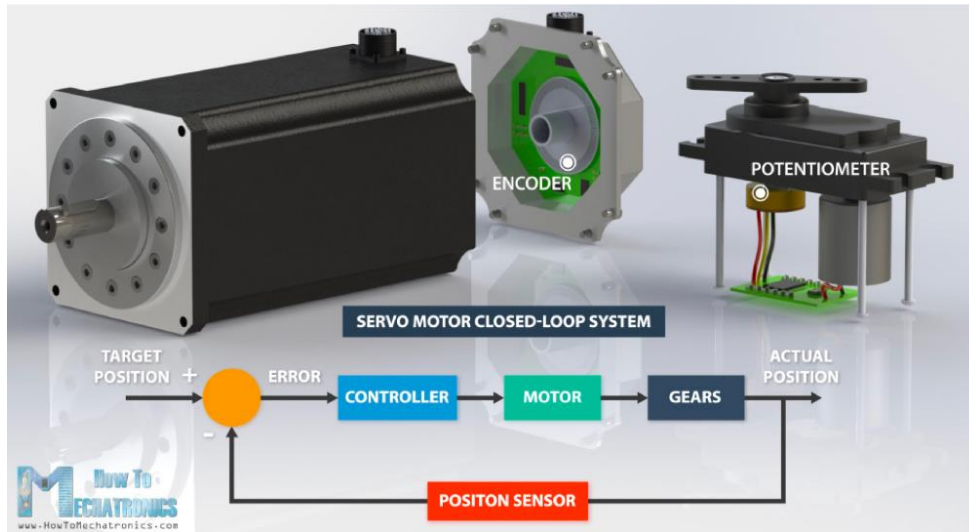


Figure 20: Servo Motor Diagram with Potentiometer

Figure 21 below shows the ROB-09605 servo motor that was selected for the student kit [25]. This model is a micro servo, which was chosen due to its lower price point. While micro servos are less powerful than normal servos, the team felt that the reduction in power was offset by the reduced cost.

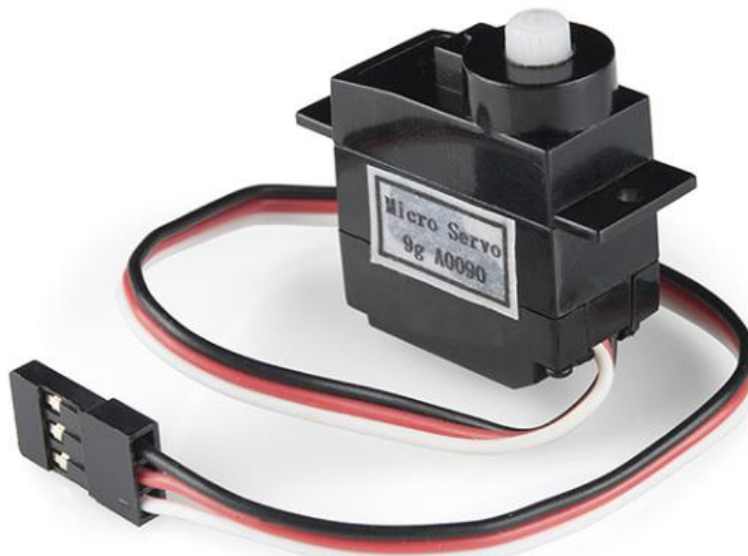


Figure 21: ROB-09605 Micro Servo Motor

Stepper Motor

A stepper motor is a specific variation of a brushless DC electric motor. The main selling point of stepper motors is their unique ability to move in discrete increments, or steps. Unlike most motors, which can only be run continuously and at a fixed input speed, stepper motors can be made to start and stop many times while being run. Due to the high level of control this gives the user, stepper motors can achieve great results in cases where higher precision is required than a typical motor can provide. Figure 22 below shows the internal construction of a stepper motor, consisting of various electromagnetic coils surrounding a central gear-rotor interface [26].

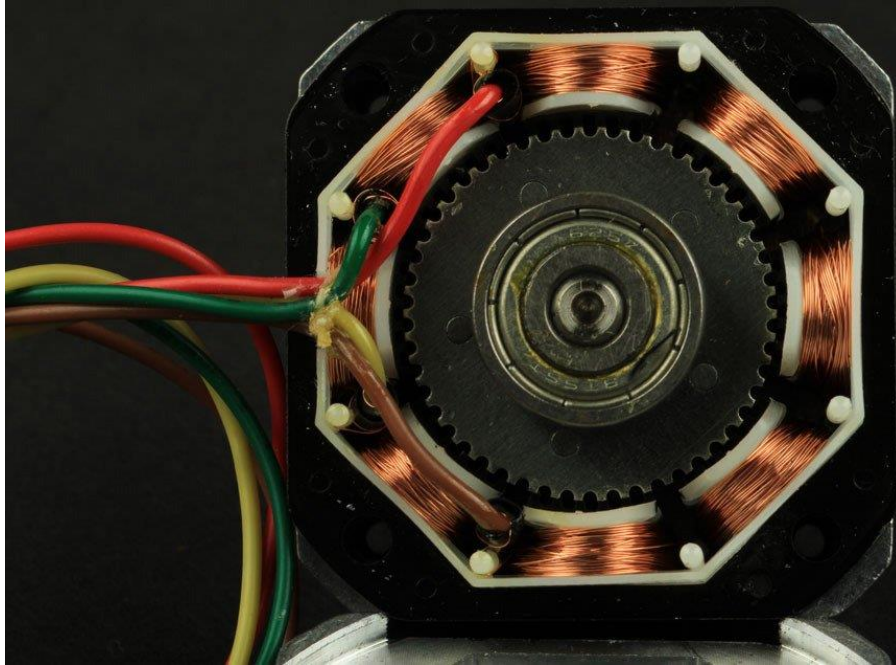


Figure 22: Internal Stepper Motor Components

Figure 23 below shows an image of the internal process through which stepper motors move between steps [27]. The black line shows the initial position of the gear, which moves in small angle increments corresponding to each individual step. The motion is achieved from one of the electromagnets activating while the other three are deactivated, causing the gear to move one step further along its rotation. This process is repeated as desired. As shown here, stepper motors allow for very specific positions and speeds to be used, making them ideal for applications which require high levels of control. For this reason, they are commonly used in small CNC mills and 3D printers.

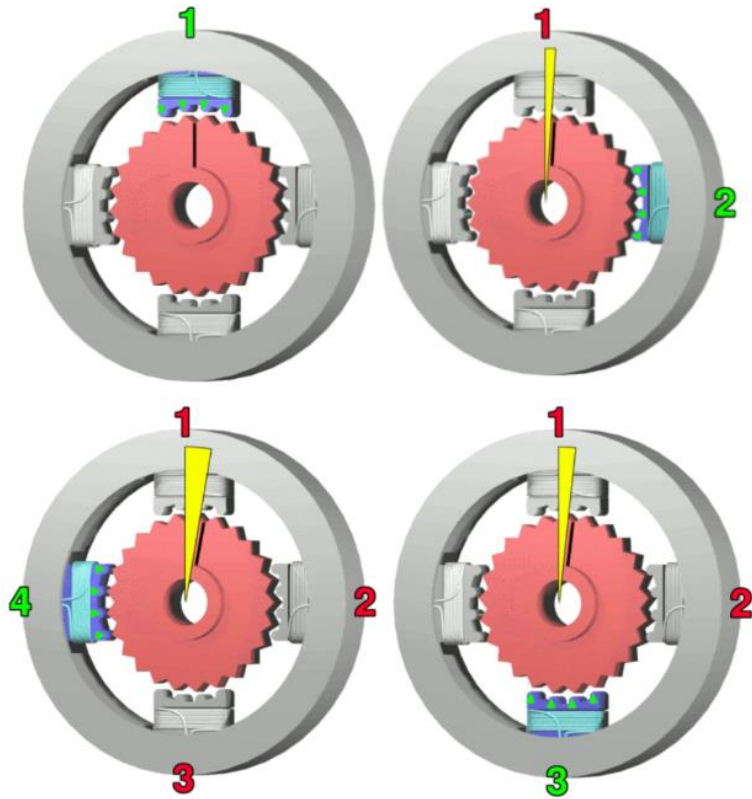


Figure 23: Stepper Motor Step Path

Figure 24 below shows the 28BYJ-48 5-wire stepper motor that was selected for the student kit [28]. This model was chosen due to the motor driver board that came included, which simplified the wiring by directly connecting to the motor, needing only 4 wires to interface with the microcontroller.



Figure 24: 28BYJ-48 Stepper Motor

Module 5: Concentrations

Gas Sensor

Why is measuring concentration even important? Well changing the concentration of some substances even by a few ppm can lead them to turn from safe to toxic. For example, OSHA will allow workers to experience 50 ppm of carbon monoxide for 8 hours straight but if that level reaches 100 ppm, which is an increase of .005%, then people need to immediately evacuate. Also, when people have been drinking the difference between being completely sober and blowing a .08 on a breathalyzer meaning that you are legally drunk is only .4 mg of alcohol per liter of breath. So, you can see that not only does concentration make a big difference when it comes to safety but also legal matters so being able to measure concentration accurately and quickly is very important.

When considering which type of sensor to put in the box the team considered a lot of different types of concentration sensors. One sensor that was suggested was a gravity pH sensor, shown in Figure 25 below, which works by measuring the amount of voltage between two electrodes suspended in a liquid [29]. By measuring the voltage produced when passing through a few liquids of known pH you can calibrate this sensor to measure the pH of any liquid. Both a positive and negative electrode are placed in the liquid and electrons will flow from the negative to the positive electrode. This obviously produces a voltage however how much voltage is produced depends on how easily the liquid lets electrons pass through. This varies directly with pH because the number of positive ions which affects the pH also will affect how easily the electricity flows. However, this sensor happened to be out of the price range of what the team was looking for when adding a module to the box.

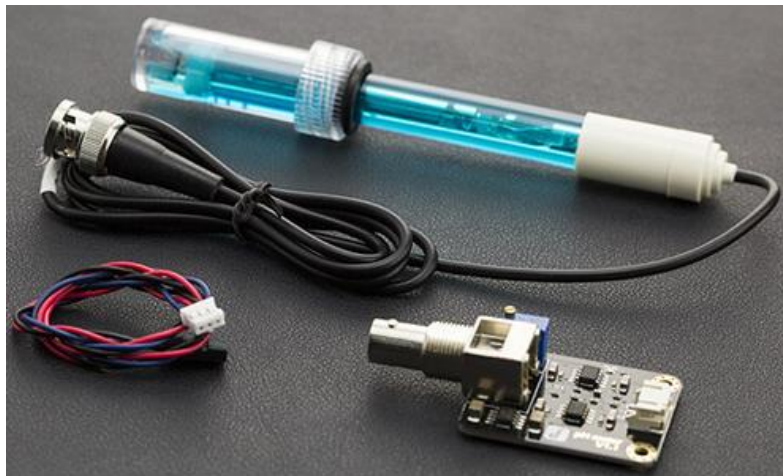


Figure 25: Gravity pH Sensor

The next option the team looked at was an electrochemical sensor, shown in Figure 26 below [30]. This type of sensor works by measuring the amount of current flowing between two electrodes similar to the gravity pH sensor. Firstly, they filter the air through a small screen mesh to take out any large particles like dust and keep the sensor clean on the inside. Next the gas we would like to measure reacts with water vapor in the air which produces both hydrogen ions (H^+) and electrons. Now that we have both hydrogen ions and electrons, we allow the electrons to flow through an ammeter and the hydrogen ions diffuse through an electrolyte layer between the two electrodes. Finally, the hydrogen and electrons meet up on the other side of the electrode and combine with oxygen in the air to form H_2O . The rate at which these electrons flow through the electrodes is proportional to the amount of the chemical we are measuring in the air. We can use test data sheets provided by the manufacturer to see what concentrations will cause what voltage. This type of sensor has a very long lifespan and very high reliability so they are used more often when the level of precision needs to be higher. This also makes this type of sensor more expensive and therefore we could not include it in our box.



CM-32910

Figure 26: Electrochemical Sensor

The final sensor that we looked at was a gas resistance sensor, shown in Figure 27 below, which is what we chose to include in the student kit for the pilot class [31]. This type of sensor works by changing resistance based on how much of the given molecule is in the air around the sensor. This type of sensor only works with molecules that are both airborne and flammable which means it works great for ethanol (alcohol) and methane. First, a heating element is allowed to warm up for 15 minutes to let the sensor reach an equilibrium state. This makes the sensor very hot and will allow the flammable particles to burn on impact with the sensor. Doing this reaction requires some oxygen to be used some of which comes from the surroundings, but the rest comes from inside the sensor. The MQ-X sensors, which was the brand that the team ended up selecting, use a tin oxide (SnO_2) layer to trap the oxygen but this does not mean that the oxygen comes from that molecule but rather the lattice traps some air in its structure and this is part that is used when the flammable particle is burned. When the oxygen is in the lattice it causes the sensor to have a much higher resistance than when it is removed so this means that as more oxygen gets used the more the resistance decreases. We can then use this resistance to see how much oxygen has been burnt and using a mixture of chemical equations and testing that the sensor company has done to see how much of a particle has been burned.



Figure 27: MQ-3 Gas Resistance Sensor

Module 6: Strain

Strain Gage

One of the most common methods of measuring the amount of strain in an object is through the usage of a strain gage. This strain sensor has a wide variety of applications and can be found both in the home and in industry. In fact, any digital scale, from kitchen scales to bathroom scales, will contain a series of strain gages that are used to determine the weight being applied onto the scale plate. They can also be found in the automobile and energy industries, used to determine torque in engines or generators through the amount of strain and rotational speed a shaft is experiencing.

A strain gage can detect fluctuations in strain through changes in resistance as a result of deforming the fine wire or metallic foil contained within the gage itself. The apparatus is typically attached directly to an object, where it will be able to experience the same stresses and strains that the object is under. Therefore, when the object is undergoing any form of strain, the gage will experience the same strain, resulting in a linear change in electrical resistance. A schematic of the parts of a strain gage are shown below in Figure 28 [32]. As the gage experiences deformation as a result of mechanical strain, the metallic grid will change its electrical resistance in response.

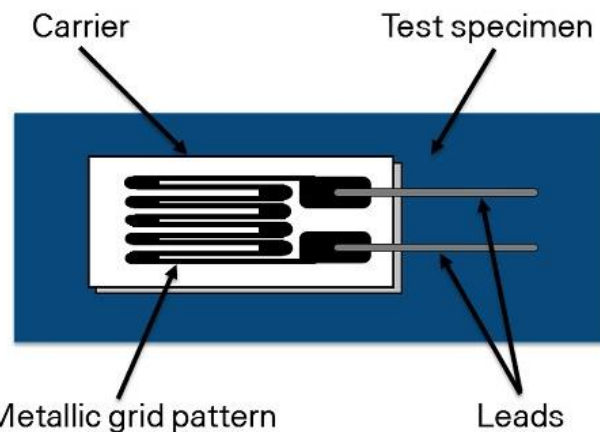


Figure 28: Schematic of Strain Gage With Test Specimen

Depending upon the material the strain gage is made up of, it will have a certain gage factor, which is the ratio between the fractional change in electrical resistance and the strain it is experiencing. Since changes in strain tend to be so minuscule in most practical applications, the changes in resistances will similarly be quite small. Therefore, in order to measure these tiny changes in strain and electrical resistance, strain gages utilize a Wheatstone bridge to produce a readable voltage.

Strain gages are widely used in industry due to their versatility. They are extremely simple structurally and do not require much difficulty to set up. Additionally, they are made of extremely light wire or foil, so they make a very minimal impact on affecting the structure of the object they're attached to. However, being as small and delicate as they are, strain gages are easily influenced by excessive tension and compression, which could potentially affect the measurement readings. They are also extremely temperature-sensitive, which could compromise the integrity of the strain measurements when placed in a hotter environment and can even potentially melt with a large enough electric current.

When selecting a sensor to use for the strain module, the team opted for the TAL220 load cell, which has a mass limit of 500g, and the HX711 load cell amplifier. The load cell, which consisted of strain gages attached to a metal bar, happened to be the most affordable out of the other load cells that could be used with the HX711 amplifier. Because the strain experiment was designed to use much smaller loads, the smaller load cell was suitable for the experiment's needs. Additionally, the HX711 load cell amplifier comes with a built-in analog-to-digital converter, along with other electrical components, so the circuit would be significantly simplified for students to wire together.

Module 7: Motion

Accelerometer

The accelerometer is one of the most utilized movement reading devices in most electronic systems today. Accelerometers measure acceleration, or the acceleration of the system it is connected to. They are used for two purposes; to sense vibrations in connected systems and for plane orientation applications. Accelerometers naturally measure the acceleration of gravity but can fluctuate in acceleration measurements depending on the environment's gravitational pull or elevation. They can measure acceleration from 1 to 3 axes or dimensions depending on the complexity of the system. This is what differentiates a one-dimensional accelerometer and a three-dimensional accelerometer. A common example to better understand an accelerometer's functionality is in your everyday smartphone. The accelerometer in your smartphone understands the orientation of the device in 3D space and takes actions accordingly. If you flip your phone horizontally, the accelerometer will recognize this and flip your screen to match your phone's orientation. Figure 29 below shows the ADXL345 accelerometer breakout board used in the ME 3902 pilot class [33].

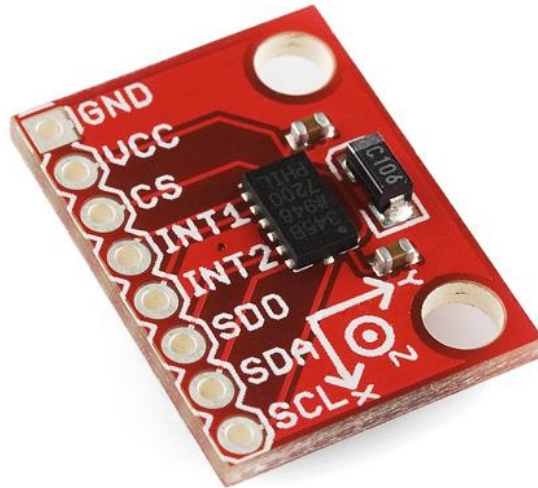


Figure 29: ADXL345 Accelerometer Breakout Board

Accelerometers have a mass attached to a spring which is confined to move along one direction with fixed outer plating along the mass. When an acceleration is applied in a particular direction, the mass will move and the capacitance between the plates and the mass will change. This measured capacitance change is converted to a specific acceleration value. Accelerometers consume energy in the milli-volts, only requiring a 5V supply for the connected system. This is useful especially in everyday electronics like phones and GPS systems, as lower energy consumption allows for a longer battery life and overall lifespan in the device. Accelerometers also have a variety of ranges they can measure from, mainly depending on the application and required sensitivity. The ranges vary from +/- 1g to +/- 250 g, which allows for adjustability in the type of experiment one might want to pursue with one. As an example, a small vibration might be better suited for a smaller range accelerometer with a higher sensitivity. Meanwhile, a vibration to the size of a magnitude 5 earthquake would better be suited for a higher range accelerometer with a lower sensitivity.

These systems, specifically accelerometers, gyroscopes and magnetometers, are silicon integrated systems but are mechanical in nature. They have tiny mechanical structures that can interface to electronics. Specifically, for vibration-based accelerometers, there is a capacitor with a tiny weight on the end of it and when it vibrates, the combs in the capacitor shift. Similarly, to the other accelerometer type, the capacitance change is measured and converted to a specific acceleration value. The MEMS type of accelerometer is more used for vibration or acoustic experiments and applications. Figure 30 below shows a schematic of a MEMS accelerometer [34], and Figure 31 below shows common applications for these devices [35].

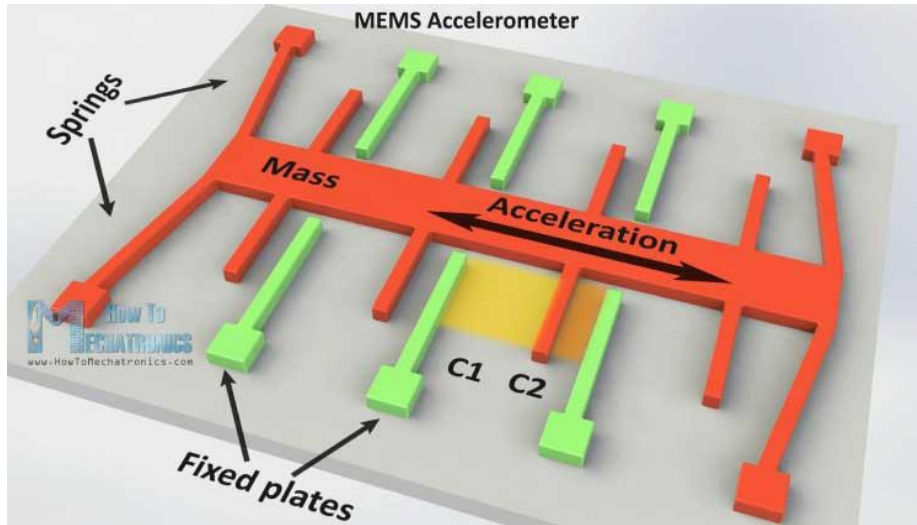


Figure 30: MEMS Accelerometer Schematic

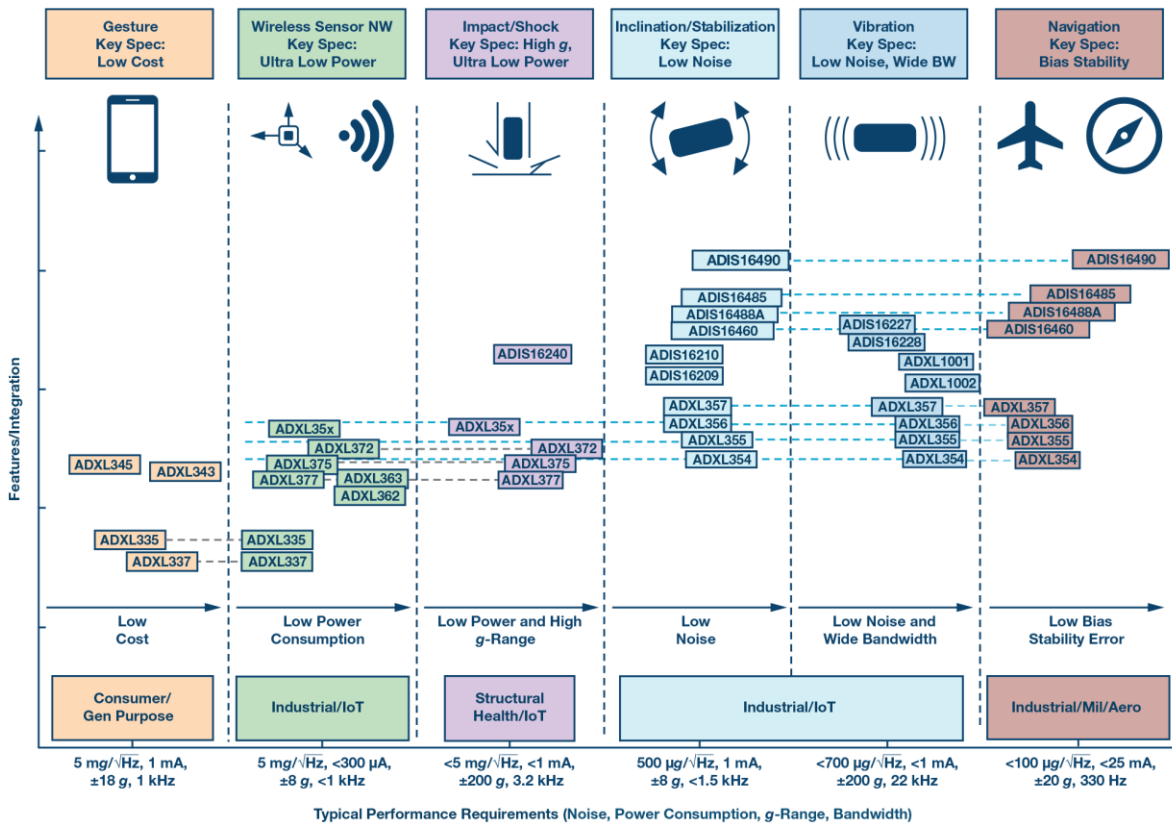


Figure 31: Applications of MEMS Accelerometers

Module 8: Open-Ended Project

The entire class structure built up to the final module, an open-ended project where students were required to construct a device that utilized a combination of 3 sensors and/or devices. The purpose of this was to allow students to gain an understanding of how to create and use a device that utilized more than one sensor at a time, similar to how most products on the market today are designed. Utilizing the knowledge gained through work in the previous modules, students proposed their original project ideas that were modified and revised as needed. The professor, PLAs, GLA, and TA assigned themselves to different student groups of their choosing in order to act as their advisors, offering feedback on ideas and existing work and suggesting changes in cases where the original project idea was found to be unfeasible. The scope of the projects was kept within realistic parameters, but the students themselves were fully in charge of their projects.

Methodology

The following subsections discuss the process of designing the ME 3902 experiment modules, as well as how these experiments were run during the pilot class.

Experiment Module Design

The goal of each and every module was to create an approachable yet deep introduction to any particular set of sensors. The modules themselves were separated into different levels of complexity, presented in the following order: Level 1 for Modules 1-3, Level 2 for Modules 4-6, Level 3 for Module 7, and Level 4 for Module 8. The modules were built with these different levels in mind. The levels symbolized the level of knowledge that the students would acquire as the course continued. The main difference between the levels was the amount of information given to students that would help them along with their module. Modules within level one assisted students by providing the full code to the problem at hand and instructing the student as to how to wire the demonstration. Most of the work of the student in level one modules would be to understand the theory behind the procedure as they are conducting the modules, information that would be very important in the upcoming modules.

At level two, modules offer information about the sensor at hand and how its operational capabilities but includes less coding information. Level two assumes that students are able to learn from and apply previously used code in these labs. The level one labs were designed in such a way that the code could be very easily copied and utilized in other labs to accomplish the same task. For example, the while loops from module 3 kept the IR motion sensor scanning indefinitely for triggers in motion. The same while loop format with little modification could be copied and used in the motor module to continuously drive a motor. This knowledge from previous levels also allowed students to execute different functions during these labs and to create things beyond the scope of the lab.

Level three and level four offered little to no direct coding help. The level four module was the final project of the course, a summary of the knowledge obtained during the previous modules into one final project. Students were required to propose and work with staff in a final project that was entirely their own. Assistance was provided for them when necessary, but the bulk of the work was done independently by students with little direction. The principles of the previous levels of modules assisted students that previously had no wiring or coding skills to create functional sensor assemblies in their final project.

The methodology behind the online component of this course also shaped the way the modules themselves were created. Knowing the goal of this project was to create an online experimentation course, some hardware sacrifices had to be made and some simplifications to modules had to occur. Many modules utilized items easily accessible in a home environment, such as cups of water or rubbing alcohol, while trying to stay self-sufficient on the provided box of equipment as much as possible. Heat was a reoccurring necessary tool to multiple modules, and it was kept at a safe temperature with explicit instructions of use. The key things that needed to be taught, the measuring of temperature, strain and the operation of motors among others, will still be delivered at an ABET standard regardless of home or lab environment.

Module 1: The Microprocessor-Computer

LED Experiment

The goal of Module 1 was to introduce students to the basics of microcontrollers. In order to showcase this with a relatively basic example, an LED experiment was chosen. The experiment consisted of wiring an LED to the microcontroller in order to turn it on. Figures 32A [36] and 32B below [37] show the LED experiment setup for an Arduino and a Raspberry Pi, respectively. Guidelines and instructions for the experiment for both Arduino and Raspberry Pi are linked in the figure caption. For all subsequent experiments, a summary of the objectives, materials, and procedure is provided, in addition to referenced codes located in Appendix 3.

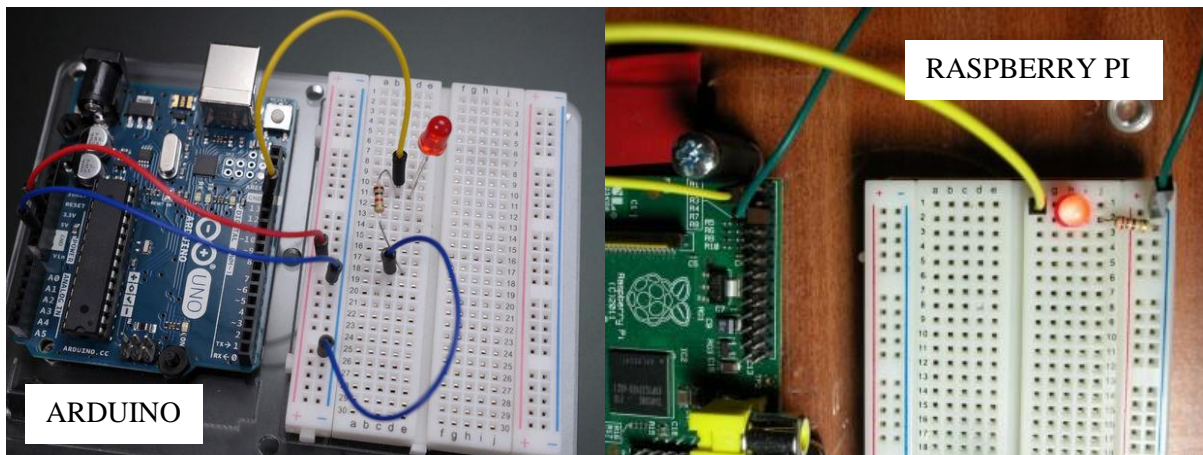


Figure 32A: LED Experiment Setup for Arduino

Figure 32B: LED Experiment Setup for Raspberry Pi

Module 2: Temperature

Silicon Band Gap Sensor

- **Objectives**
 - Create a system that can measure temperature using the TMP36 silicon band gap temperature sensor and an Arduino or Raspberry Pi
 - Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
 - Write a program for an Arduino or Raspberry Pi that reads the temperature using the sensor
 - Ensure the code can run without any errors
 - Run the code and examine the results
 - Observe the system to see how it behaves
 - Test the system by holding the sensor in your fingertips, comparing the temperature reading to the reading obtained when the sensor is reading ambient air, and examining the results
 - Determine if the system demonstrates the expected behavior
- **Materials**
 - Arduino

- (1) Arduino
 - (3) male to male wires
 - (1) breadboard
 - (1) TMP36 silicon band gap sensor
 - Raspberry Pi
 - (1) Raspberry Pi
 - (13) male to female wires
 - (1) breadboard
 - (1) TMP36 silicon band gap sensor
 - (1) MCP3008 analog to digital signal converter (ADC)
- **Procedure**
 - Arduino
 - Code [38] (See Appendix 3)
 - Wiring (Figure 33 below)
 - Arduino Analog Channel A0 -> TMP36 Input Voltage VIN
 - Arduino Power Supply 5V -> TMP36 Output Voltage VOUT
 - Arduino Ground GND -> TMP36 Ground GND
 - Raspberry Pi
 - Code [39] (See Appendix 3)
 - Wiring (Figure 33 below)
 - Raspberry Pi, Pin 1: 3.3V (red wire) -> TMP36 V_In, MCP3008 V_DD, MCP3008 V_Ref
 - Raspberry Pi, Pin 6: GND (black wire) -> TMP36 GND, MCP3008 V_AGND, MCP3008 V_GDND
 - Raspberry Pi, Pin 19: MOSI (yellow wire) -> MCP3008 V_Din
 - Raspberry Pi, Pin 21: MISO (cyan wire) -> MCP3008 V_Dout
 - Raspberry Pi, Pin 23: SCLK (blue wire) -> MCP3008 V_CLK
 - Raspberry Pi, Pin 24: CE0 or CS0 (orange wire) -> MCP3008 V_CS/SHDN
 - MCP3008, Terminal 1: CH0 -> TMP36 V_Out
- **Notes**
 - The red rails on the breadboard are positive, and the blue rails are negative. It is necessary to orient the power with the red rail and the ground with the blue rail.
 - With the flat side of the TMP36 facing you, the left prong should be wired to power and the right prong wired to ground, with the middle prong wired to the data channel (reversed on Raspberry Pi schematic).
 - Raspberry Pi only
 - Pay attention to the orientation of the MCP3008 - the small semicircular indentation should be used as a reference to position it on your breadboard. In the diagram, the indentation is on the left side of the “MCP3008” text, facing the Raspberry Pi board.
 - Make sure that the MCP3008 bridges both halves of the breadboard as shown below (connect it across the middle gap).

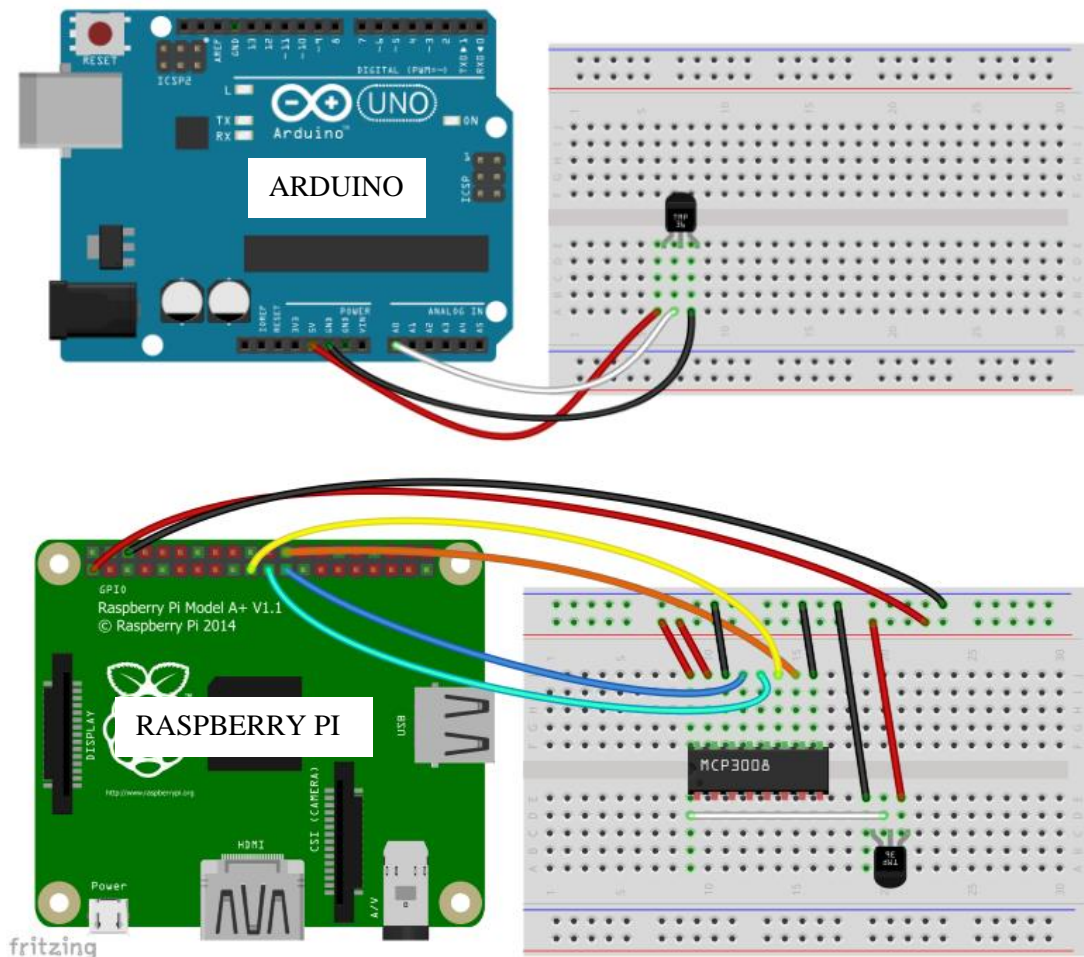


Figure 33: Fritzing Diagrams of Arduino and Raspberry Pi Silicon Band Gap Sensor Experiment

The goal of this experiment was to obtain a mathematical relationship for obtaining temperature values from the voltage output of the silicon band gap sensor. This was done by running code to convert the voltage output of the sensor into a corresponding temperature value, which displayed the temperature of the area surrounding the sensor. After running the code initially to verify the system was operating properly, the user was then instructed to place their fingers around the sensor to see if the system responded by outputting a higher temperature. The temperature values obtained for ambient air and with fingers on the sensor were compared and used to estimate the temperature relationship for the sensor, which was checked against provided reference data.

The Sparkfun TMP36 silicon band gap sensor performed well when used, but the setup for the Raspberry Pi version of the experiment is disproportionately more complex than the setup for the Arduino version. This is because the TMP36 is an analog sensor, which requires an analog-to-digital converter (ADC) to be used in order for the microcontrollers to be able to read the sensor's output. The Arduino has an ADC built in, but the Raspberry Pi does not, so an external ADC must be wired in addition to the sensor itself. This makes the physical setup much more cumbersome. If both Arduino and Raspberry Pi are used in future offerings of the course, a solution will be needed so that the differences in difficulty level for the same experiment are minimized. Additionally, silicon band gap sensors, in general, cannot measure temperatures of liquids because they are not safe to immerse, like other temperature sensors such as thermocouples.

Thermistor

- **Objectives**
 - Create a system that can run a thermistor using an Arduino or Raspberry Pi
 - Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
 - Write a program for an Arduino or Raspberry Pi that runs the thermistor
 - Ensure the code can run without any errors
 - Run the code and examine the results
 - Observe the system to see how it behaves
 - Test the system by holding the sensor in your fingertips, comparing the temperature reading to the reading obtained when the sensor is reading ambient air, and examining the results
 - Determine if the system demonstrates the expected behavior
 - Calibrate the thermistor based on varying water temperature readings in correlation with a thermometer
 - Utilize a linear regression test from a graph of temperature readings
 - Find the slope and intercept values and integrate them into the code to calibrate the thermistor
- **Materials**
 - Arduino
 - (1) Arduino
 - (5) male to male wires
 - (1) breadboard
 - (1) 10k Ohm Thermistor
 - (1) 10k Ohm Resistor
 - Raspberry Pi
 - (1) Raspberry Pi
 - (10) male to male wires
 - (1) breadboard
 - (1) 10k Ohm Thermistor
 - (1) 10k Ohm Resistor
 - (1) MCP3008 AC/DC Converter
- **Procedure**

- Arduino
 - Code [40] (See Appendix 3)
 - Wiring (Figure 34 below)
 - Arduino, Analog In: A0 → Between Thermistor and Resistor
 - Arduino, Power: GND → Resistor End
 - Arduino, Power: 5V → Thermistor Beginning
- Raspberry Pi
 - Code [41] (See Appendix 3)
 - Wiring (Figure 34 below)
 - Raspberry Pi, Pin 1: 3.3V → Thermistor Voltage In, MCP3008 VDD, MCP3008 Vref
 - Raspberry Pi, Pin 6: GND → 10k Ohm Resistor GND, MCP3008 VAGND, MCP3008 VGDND
 - Raspberry Pi, Pin 19: MOSI → MCP3008 Vdin
 - Raspberry Pi, Pin 21: MISO → MCP3008 Vdout
 - Raspberry Pi, Pin 23: SCLK → MCP3008 VCLK
 - Raspberry Pi, Pin 24: CE0 or CS0 → MCP3008 VCS/SHDN
 - MCP3008, Terminal 1: CH0 → Voltage Divider Circuit (Between Resistor & Thermistor)
- **Notes**
 - The red rails on the breadboard are positive, and the blue rails are negative. It is necessary to orient the power with the red rail and the ground with the blue rail.
 - Raspberry Pi only
 - Pay attention to the orientation of the MCP3008 - the small semicircular indentation should be used as a reference to position it on your breadboard. In the diagram, the indentation is on the left side of the “MCP3008” text, facing the Raspberry Pi board.
 - Make sure that the MCP3008 bridges both halves of the breadboard as shown below (connect it across the middle gap).

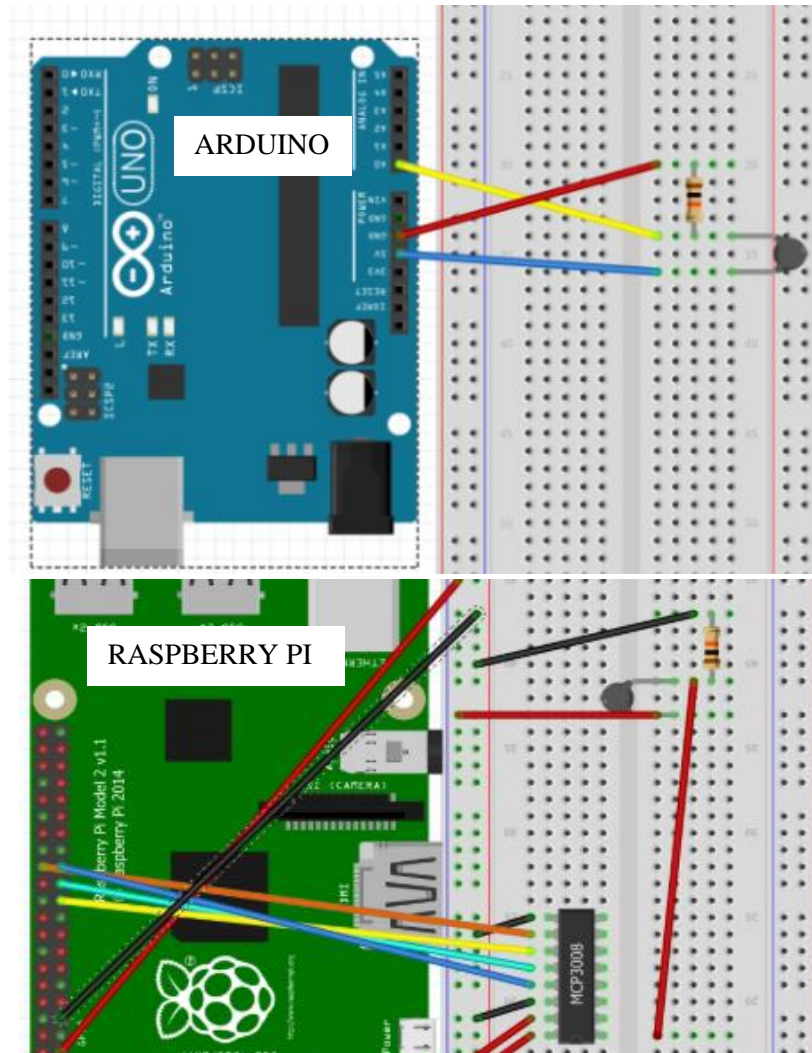


Figure 34: Fritzing Diagrams of Arduino and Raspberry Pi Thermistor Experiment

Module 3: Proximity

Infrared (IR) Motion Sensor

- Objectives:
 - Detect motion using the HC-SR501 PIR sensor
 - Develop code that utilizes the HC-SR501 to create a motion sensing system
 - Understanding the limitations of PIR sensors through testing
- Materials:
 - Arduino
 - 1 Arduino UNO
 - 1 HC-SR501 PIR Motion Sensor (Figure 35 below)
 - 1 LED
 - 1 Breadboard (optional)
 - Raspberry Pi
 - 1 Raspberry Pi

- 1 HC-SR501 PIR Motion Sensor
 - 1 LED
 - 1 Breadboard
 - 1 4.7k Ohm Resistor
- Procedure:
 - Arduino
 - Code [42] (See Appendix 3)
 - Wiring (Figure 36 below)
 - Yellow: Digital Pin 2 - High/Low Output
 - Red: 5V Power – Power
 - Black: Ground - Ground(sensor)
 - LED:
 - Anode - Data Pin 13
 - Cathode - Ground Pin
 - Raspberry Pi
 - Code [43] (See Appendix 3)
 - Wiring (Figure 37 below)
 - Yellow: Pin 8 - High/Low Output
 - Red: 5V Power – Power
 - Black: Ground - Ground (sensor)
 - LED:
 - Anode - Pin 10
 - Resistor between Anode and pin 10
 - Cathode – Ground
- Notes:
 - The original code did not fully control the time delay between motion activations. The potentiometers mounted onto the sensor provided some level of adjustment but in practice offered very little change in the sensitivity of the sensor.
 - The cone of the sensor could not be controlled with software either, the potentiometers mounted onto the sensor also provided very little relief.

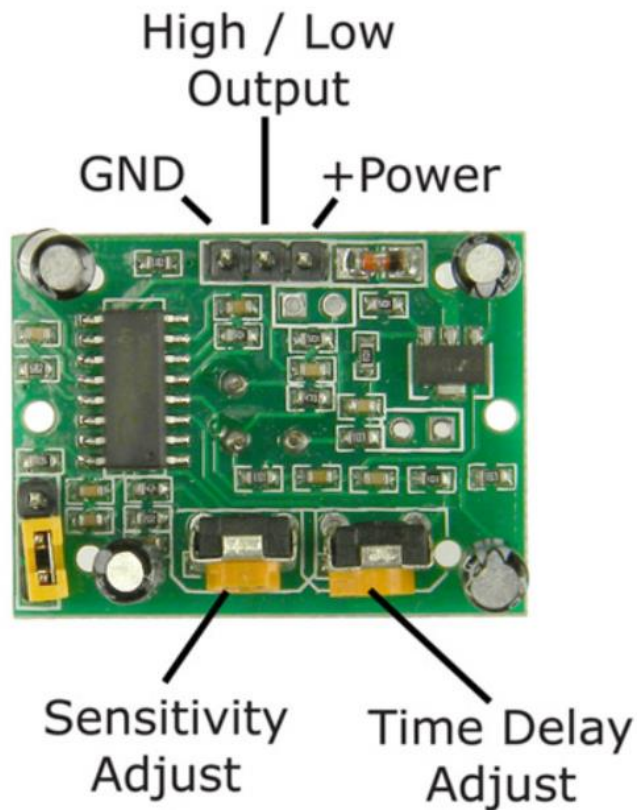


Figure 35: Physical Setup of IR Motion Sensor

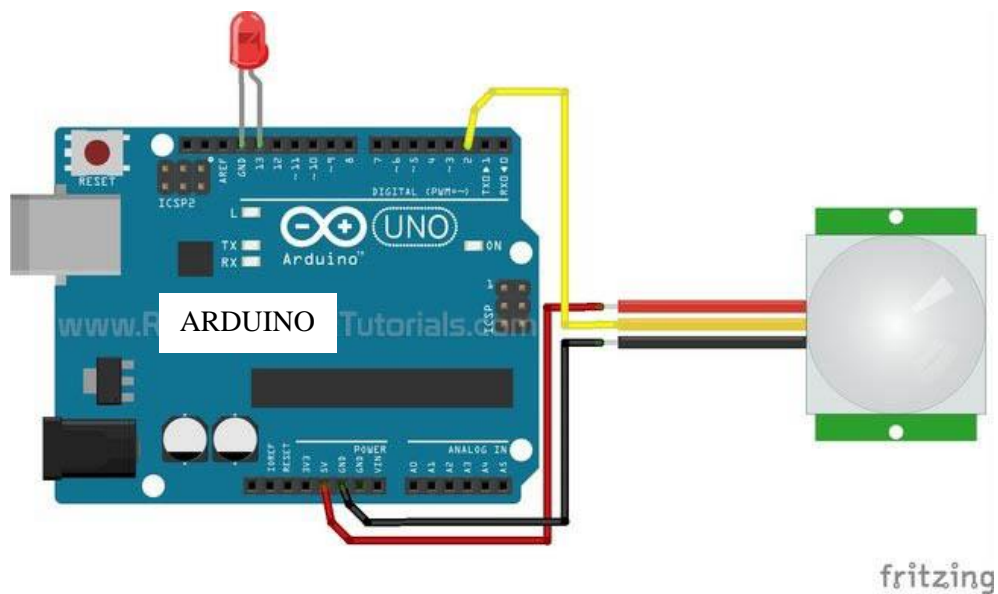


Figure 36: Fritzing Diagram of Arduino IR Motion Sensor Experiment

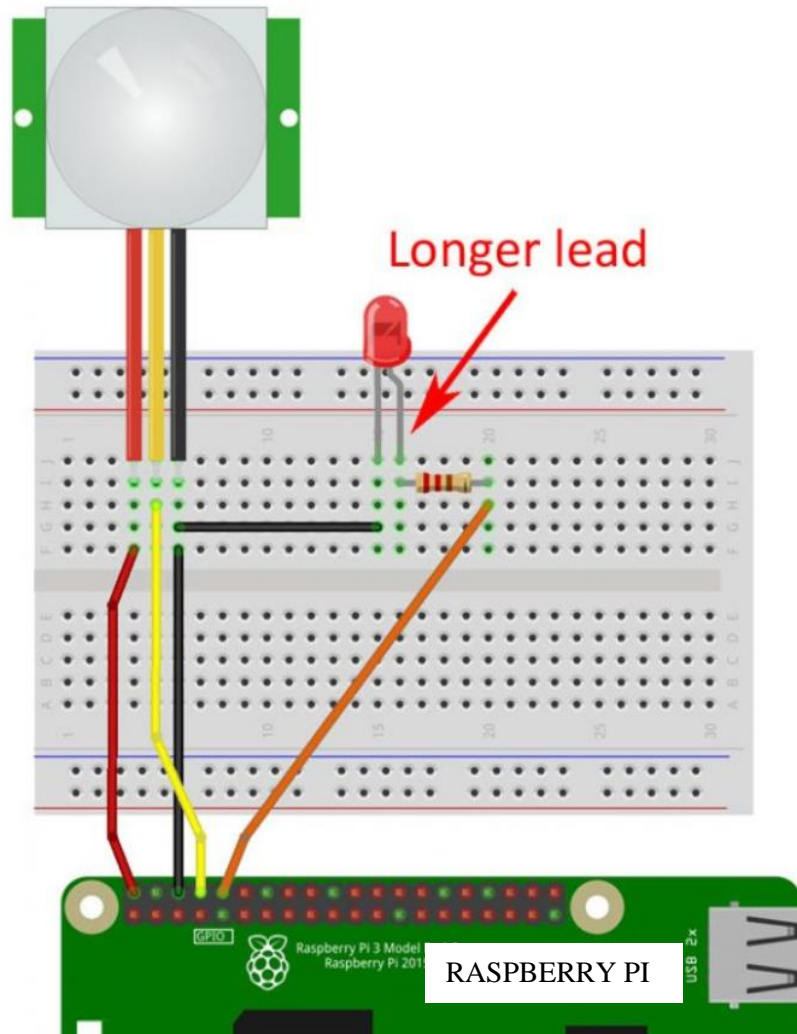


Figure 37: Fritzing Diagram of Raspberry Pi IR Motion Sensor Experiment

The goal of this experiment was to introduce students to the world of motion sensing through a simple passive infrared sensor like the HC-SR501. The code itself facilitated learning by allowing students to modify and add any other secondary actions to a successful motion trigger. The students were encouraged to test the sensor in multiple environments to see how noise affects the sensor, as well as to determine the operational range of the sensor.

The HC-SR501 proved to be a bit unreliable. The sensor behaved correctly and triggered on motion, yet the delay between correct motion sensor triggers was too large and sluggish. As such, the sensor was cumbersome to test and utilize. Overall, the sensor was sufficient in teaching the principle operation of PIR sensors but could not be utilized correctly during the final open-ended project and as such, would not be recommended unless the sluggish trigger reset could be solved.

Ultrasonic Sensor

- Objectives:
 - Measure proximity using an HC-SR04 ultrasonic distance sensor and an Arduino

- Develop code that allows the HC-SR04 sensor to produce proximity readings
- Use the ultrasonic sensor to determine the proximity of a person standing .5, 1, 2, and 4 meters away from the sensor
- (Raspberry Pi only) Understand what voltage dividers are and how they allow sensors in a circuit to produce readings
- Materials:
 - Arduino
 - 1 Arduino UNO
 - 6 male-to-male wires
 - 1 breadboard
 - 1 HC-SR04 ultrasonic sensor
 - 1 computer
 - Raspberry Pi
 - 1 Raspberry Pi
 - 6 male-to-male wires
 - 1 breadboard
 - 1 HC-SR04 ultrasonic sensor
 - 1 330-ohm resistor
 - 1 470-ohm resistor
 - 1 computer
- Procedure:
 - Arduino
 - Code [44] (See Appendix 3)
 - Wiring (Figure 38 below)
 - Red: Pin 8 – Trig
 - Blue: Pin 9 – Echo
 - Black: Gnd (sensor) – Ground
 - Yellow: VCC - Power supply
 - Green: Power supply - 5V
 - Gray: Gnd (Arduino) – Ground
 - Raspberry Pi
 - Code [45] (See Appendix 3)
 - Wiring (Figure 38 below)
 - Red 1: 5V - Power supply
 - Red 2: Power supply – VCC
 - Black 1: Gnd (Raspberry Pi) – Ground
 - Green: Trig - Pin 18
 - Black 2: Gnd (sensor) – Ground
 - 330-ohm resistor: Echo - 470-ohm resistor – Purple
 - 470-ohm resistor: 330-ohm resistor - Purple – Ground
 - Purple: 330-ohm resistor - 470-ohm resistor - Pin 24
- Notes

- The original code does not contain a time delay, so the proximity measurements will be printed faster than a readable speed. To combat this, a time delay function should be introduced, although the measurements may seem more “inconsistent” as a result.
- The wiring diagram for the Arduino has slightly different pins than the original code, so care must be taken to ensure the pins and code correspond with one another.
- The sensor has a lot of noise in the readings, which becomes more obvious with a time delay introduced but is perfectly normal.

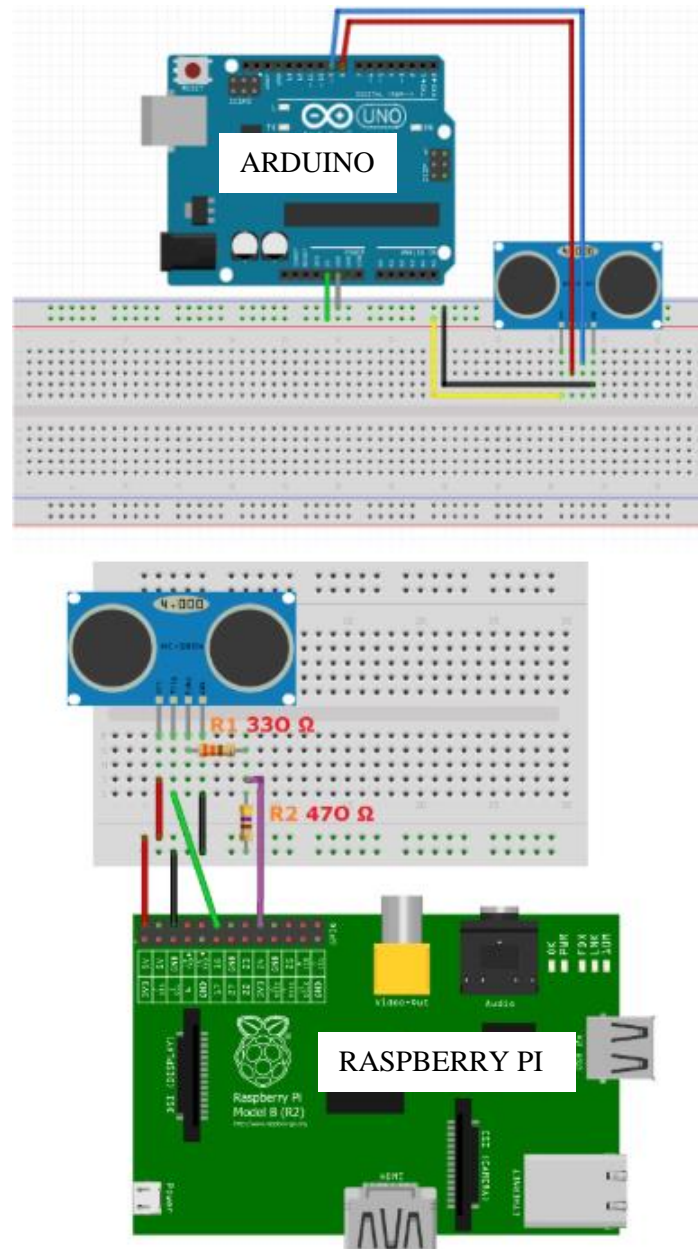


Figure 38: Fritzing Diagrams of Arduino and Raspberry Pi Ultrasonic Distance Sensor Experiment

The goal of this experiment was to measure proximity using an HC-SR04 ultrasonic distance sensor and a microcontroller. This was done by developing code that allowed the sensor to produce proximity readings, which was then tested by measuring a person standing at different distances within the sensor's proximity range. Raspberry Pi users also had to utilize a voltage divider in order to produce accurate readings with the sensor. Students were encouraged to add a delay function to the code provided in order to produce easier-to-track readings.

The HC-SR04 ultrasonic distance sensor worked very well, and students found the experiment straightforward to set up and perform. However, when measuring different distances for a standing person, the readings became very noisy due to the numerous obstacles located within the lab room. While students found greater success taking their measurements in the hallway outside the lab classroom, a method of reducing sensor noise would be important for future implementations. Overall, this sensor model would be sufficient for future offerings of the course.

Module 4: Motors

DC Motor

- **Objectives**
 - Create a system that can run a DC motor using the Adafruit TB6612 motor driver and an Arduino or Raspberry Pi
 - Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
 - Write a program for an Arduino or Raspberry Pi that runs the motor
 - Ensure the code can run without any errors
 - Run the code and examine the results
 - Observe the system to see how it behaves
 - Test the system by changing the code parameters for rotation speed and direction
 - Determine if the system demonstrates the expected behavior
- **Materials**
 - Arduino
 - (1) Arduino
 - (7) male to female wires
 - (1) breadboard
 - (1) ROB-11696 DC motor
 - (1) Adafruit TB6612 motor driver
 - Raspberry Pi
 - (1) Raspberry Pi
 - (8) male to female wires
 - (1) breadboard
 - (1) ROB-11696 DC motor
 - (1) Adafruit TB6612 motor driver
- **Procedure**
 - Arduino
 - Code [46] (See Appendix 3)

- Wiring (Figure 39 below)
 - Arduino Power Supply 5V -> TB6612 Vcc
 - Arduino Power Supply 5V -> TB6612 VM
 - Arduino Ground GND -> TB6612 GND
 - Arduino Pin 3 (Pulse Width Modulated) -> TB6612 PWMA
 - Arduino Pin 8 -> TB6612 AIN2
 - Arduino Pin 9 -> TB6612 AIN1
 - (2) DC Motor Wires -> (2) TB6612 MOTORA
 - Raspberry Pi
 - [Code \[47\]](#) (See Appendix 3)
 - Wiring (Figure 39 below)
 - Raspberry Pi Pin 1 3.3V -> TB6612 VM
 - Raspberry Pi Pin 1 3.3V -> TB6612 Vcc
 - Raspberry Pi Pin 6 GND -> TB6612 GND
 - Raspberry Pi Pin 7 GPCLK0 -> TB6612 PWMA (blue wire in diagram)
 - Raspberry Pi Pin 11 BCM 17 -> TB6612 AIN2 (white wire in diagram)
 - Raspberry Pi Pin 12 BCM 18 -> TB6612 AIN1 (purple wire in diagram)
 - Raspberry Pi Pin 13 BCM 27 -> TB6612 STBY (orange wire in diagram)
 - (2) DC Motor Wires -> (2) TB6612 MOTORA
- **Notes**
 - The specific colors of the motor wires on the diagram (yellow and green) are different from the actual motor you will be using, and these colors are arbitrary. The polarity of the motor is dictated by which wire gets the (+) vs (-) connection. The motor will rotate in the opposite direction from what is expected if the polarity is reversed. If this is not desirable, simply switch the wires.
 - The red rails on the breadboard are positive, and the blue rails are negative. It is necessary to orient the power with the red rail and the ground with the blue rail.

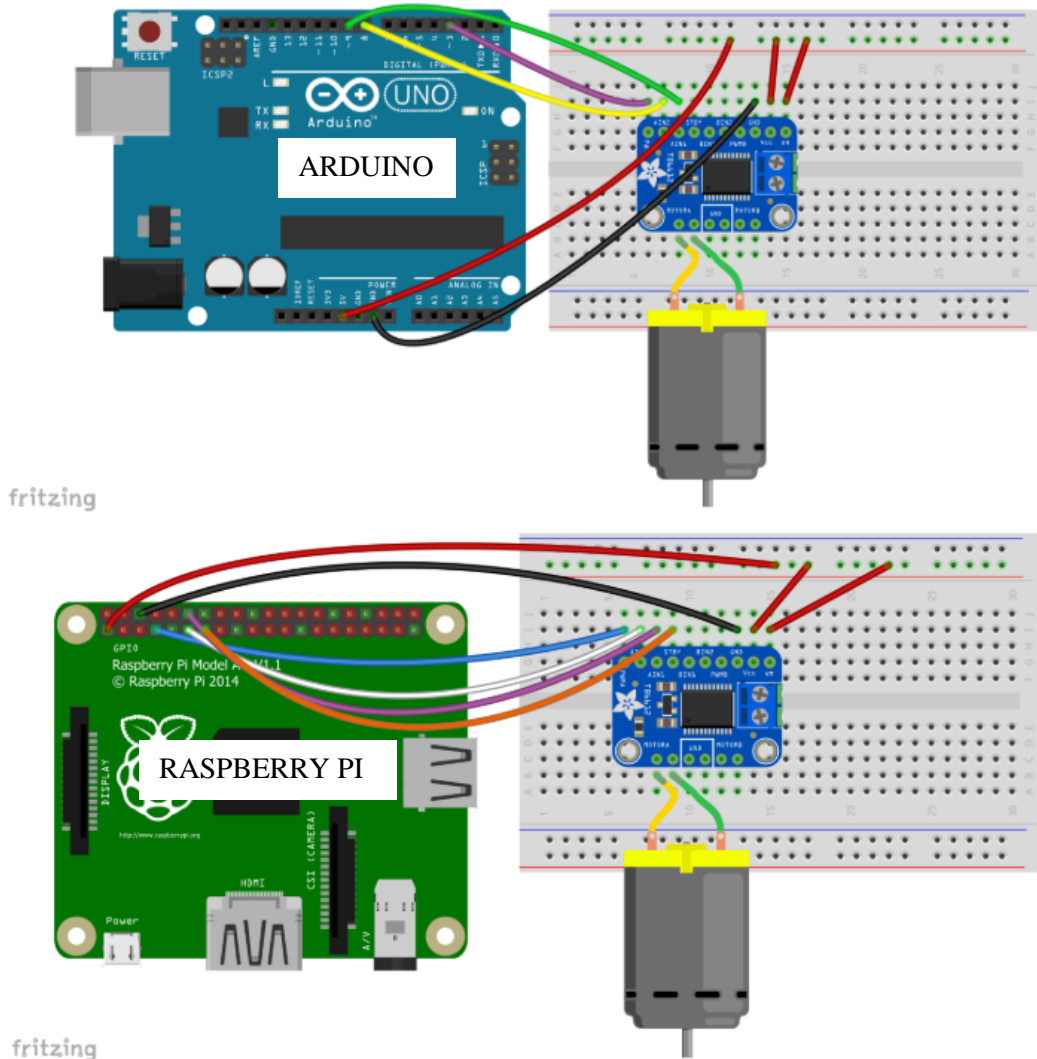


Figure 39: Fritzing Diagrams of Arduino and Raspberry Pi DC Motor Experiments

The goal of this experiment was to make a DC motor run in a user-defined pattern. This was done by running code to run the motor for a set amount of time in one direction, stop, run for the same amount of time in the other direction, stop, and repeat continuously. After running the code initially to verify the system was operating properly, the user was then instructed to modify the code to adjust the time, direction, and rotation speed of the motor as desired to examine how the motor's behavior changed.

The Sparkfun ROB-11696 DC motor performed very well, running smoothly and being able to deliver sufficient torque for projects in which it was used. This model would be sufficient to reuse for future offerings of the course.

Servo Motor

- **Objectives**
 - Create a system that can run a servo motor using an Arduino or Raspberry Pi

- Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
 - Write a program for an Arduino or Raspberry Pi that runs the motor
 - Ensure the code can run without any errors
 - Run the code and examine the results
 - Observe the system to see how it behaves
 - Test the system by changing the code parameters for rotation speed and direction
 - Determine if the system demonstrates the expected behavior
- **Materials**
 - Arduino
 - (1) Arduino
 - (3) Male to male wires
 - (1) ROB-09065 servo motor
 - Raspberry Pi
 - (1) Raspberry Pi
 - (1) ROB-09065 servo motor
- **Procedure**
 - Arduino
 - Code [48] (See Appendix 3)
 - Wiring (Figure 40 below)
 - Arduino Power Supply 5V -> Servo Red Wire
 - Arduino Ground GND -> Servo Black Wire
 - Arduino Digital Channel 11 (Pulse Width Modulated) -> Servo Yellow Wire
 - Raspberry Pi
 - Code [49] (See Appendix 3)
 - Wiring (Figure 40 below)
 - Raspberry Pi Pin 1 3.3V -> Servo Red Wire
 - Raspberry Pi Pin 3 GPIO2 -> Servo Yellow Wire
 - Raspberry Pi Pin 6 GND -> Servo Black Wire

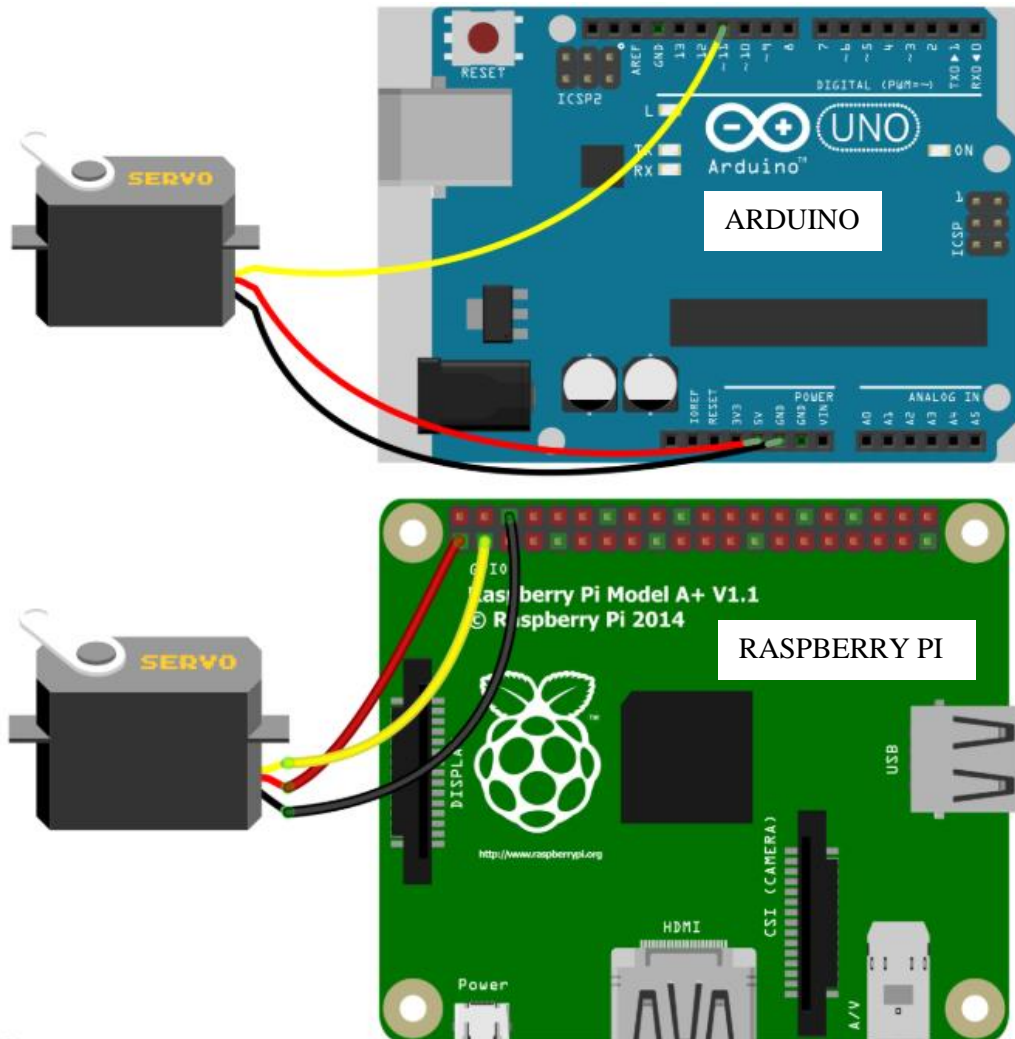


Figure 40: Fritzing Diagrams of Arduino and Raspberry Pi Servo Motor Experiments

The goal of this experiment was to make a servo motor rotate to a user-defined angle. This was done by running code to define the initial angle, set up a function to calculate angle movement, and specify the angle the servo would move to upon running. After running the code initially to verify the system was operating properly, the user was then instructed to modify the code to have the servo rotate to a different angle, as well as multiple different angles in sequence.

The Sparkfun ROB-09065 micro servo selected for the student kit did not perform very well, mainly because of its lack of utility for the open-ended projects. Its full angle range was very limited, only achieving a roughly 160° rotation. This was advertised by the seller but was more of a problem than we had anticipated, as several students who intended to use the servo motor for their project had to revise their ideas due to its rotation being insufficient for what they intended to achieve. The micro servo's rotation speed was too low for it to drive anything sufficiently. Additionally, the code that was used for the Arduino was based on the percentage of the servo's angle range, while the code used for the Raspberry Pi was based on the angle value itself, causing confusion as this was not initially realized by the students or communicated within the referenced code.

Stepper Motor

- **Objectives**

- Create a system that can run a stepper motor using an Arduino or Raspberry Pi
 - Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
- Write a program for an Arduino or Raspberry Pi that runs the motor
 - Ensure the code can run without any errors
- Run the code and examine the results
 - Observe the system to see how it behaves
- Test the system by changing the code parameters for rotation speed and direction
 - Determine if the system demonstrates the expected behavior

- **Materials**

- Arduino
 - (1) Arduino
 - (6) male to female wires
 - (1) breadboard
 - (1) 28BYJ-48 stepper motor
 - (1) X113647/ULN2003 5-wire stepper motor driver
- Raspberry Pi
 - (1) Arduino
 - (6) male to female wires
 - (1) breadboard
 - (1) 28BYJ-48 stepper motor
 - (1) X113647/ULN2003 5-wire stepper motor driver

- **Procedure**

- Arduino
 - Code [50] (Appendix 3)
 - Wiring (Figure 41 below)
 - Arduino Power Supply 5V -> X113647/ULN2003 (+) Pin (red wire)
 - Arduino Ground GND -> X113647/ULN2003 (-) Pin (black wire)
 - Arduino Digital Pin 9 -> X113647/ULN2003 IN1 Pin (blue wire)
 - Arduino Digital Pin 10 -> X113647/ULN2003 IN2 Pin (pink wire)
 - Arduino Digital Pin 11 -> X113647/ULN2003 IN3 Pin (orange wire)
 - Arduino Digital Pin 12 -> X113647/ULN2003 IN4 Pin (yellow wire)
- Raspberry Pi
 - Code [51] (Appendix 3)
 - Wiring (Figure 41 below)
 - Raspberry Pi Pin 2 5V -> X113647/ULN2003 (+) Pin (red wire)
 - Raspberry Pi Pin 6 GND -> X113647/ULN2003 (-) Pin (black wire)
 - Raspberry Pi Pin 11 BCM 17 -> X113647/ULN2003 IN1 Pin (blue wire)
 - Raspberry Pi Pin 15 BCM 22 -> X113647/ULN2003 IN2 Pin (pink wire)

- Raspberry Pi Pin 16 BCM 23 -> X113647/ULN2003 IN3 Pin (orange wire)
- Raspberry Pi Pin 18 BCM 24 -> X113647/ULN2003 IN4 Pin (yellow wire)

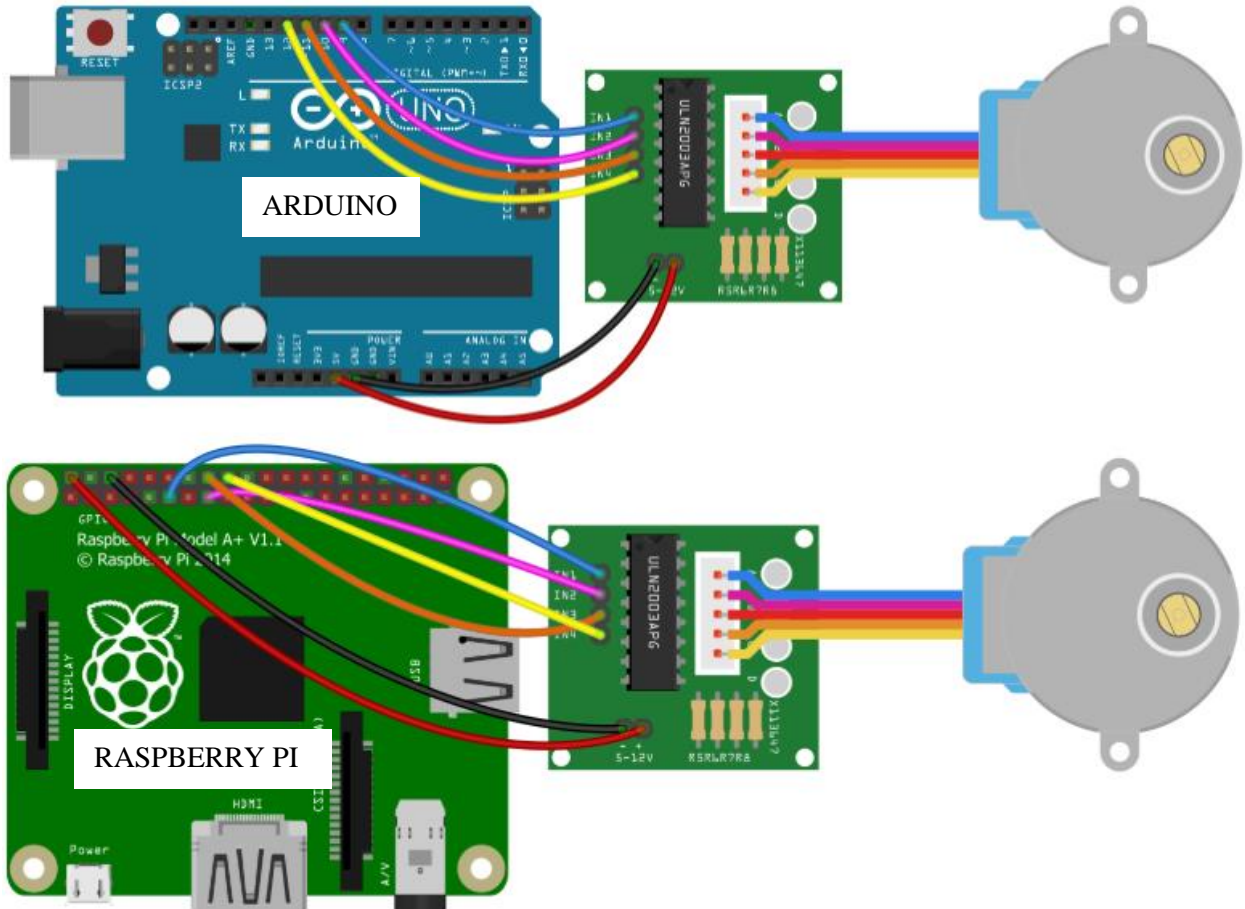


Figure 41: Fritzing Diagrams of Arduino and Raspberry Pi Stepper Motor Experiments

The goal of this experiment was to make a stepper motor run continuously in set increments, or number of steps per revolution. This was accomplished by running code to define the full range of steps, add logic for controlling the sequencing, and defining the number of steps desired. After running the code initially to verify the system was operating properly, the user was then instructed to modify the code to change the number of steps taken and alter the logic sequence as desired to make the motor move in different directions, change the number of steps, and stop and start.

The 28BYJ-48 stepper motor was a good choice for the kit, as its ease of use and good performance met expectations. However, the convenience of having the driver board meant that students did not need to think about how to properly wire the motor to the microcontroller, or what functionality each wire on the motor corresponded to. This may have taken away from the learning opportunities in this experiment. Additionally, rectangular 4-wire stepper motors with a large central shaft are more commonly used in industry than this 5-wire model. Therefore, it may be worthwhile to consider using a more conventional; 4-wire stepper motor in the future in order to better reflect standard stepper motors used.

Module 5: Concentrations

Gas Sensor

Objectives:

The main objective of this lab was to teach the students how to use the MQX line of resistance gas sensors, which measure the concentration of various substances in the air. For this, the student was given 2 sets of code that they had to modify based on their respective device. The first code was a setup code and found the resistance of the MQ sensor when it is in the background. The second code uses the output from the first code and finds the current resistance of the sensor, which is then converted to mg/L of the substance in the air, mg/L of substance in the breath, and in the case of the alcohol sensor, BAC. In order to keep the students interested in the subject of concentrations, the MQ3 sensor was chosen for this lab because the team believed a lot of students would want to build a breathalyzer function into their final open-ended project. This presented its own set of challenges when incorporating this type of sensor into a class while not encouraging the students to drink or incorporate alcohol in any way, which is covered further in the procedure. This lab, being on the intermediate level in difficulty, required the students to write their own code at the bottom of the second script to turn on an LED if the MQ3 measured a BAC over the legal limit.

- Materials:
 - Arduino or Raspberry Pi
 - MCP3008 ADC (Raspberry Pi only)
 - MQX sensor
 - Breadboard
 - 10 mL isopropyl alcohol
 - 3 Liters of water
 - 3 containers that can hold liquid
 - Jumper wires
 - 20 kilo Ohm resistance (think about what you have at your disposal)
 - 220 Ohm resistor
 - An LED (preferably red since we will be using it as a stoplight)

Procedure:

Before the students even start with the wiring or the code, they need to make the 3 “people” that they will be testing. Obviously, we cannot use real people to verify our results so we will mix 3 different solutions of isopropyl alcohol and water in order to get the sensor to read different levels of alcohol. For the pilot class, 3 solutions were made each with 1 L of water as a base. To the first liter 1 mL of 80% isopropyl alcohol was added and mixed to the second solution 3 mL of isopropyl alcohol was added and to the final liter 5 mL of isopropyl alcohol was added. In order to keep the students as safe as possible instead of having them blow across the surface of the liquid into the sensor, they were instructed to hold the sensor an inch above the surface of the liquid to try to keep everyone's results consistent. Once these “people” were set up the student could begin to follow the wiring diagrams below in Figure 42 to set up their MQ3 sensor.

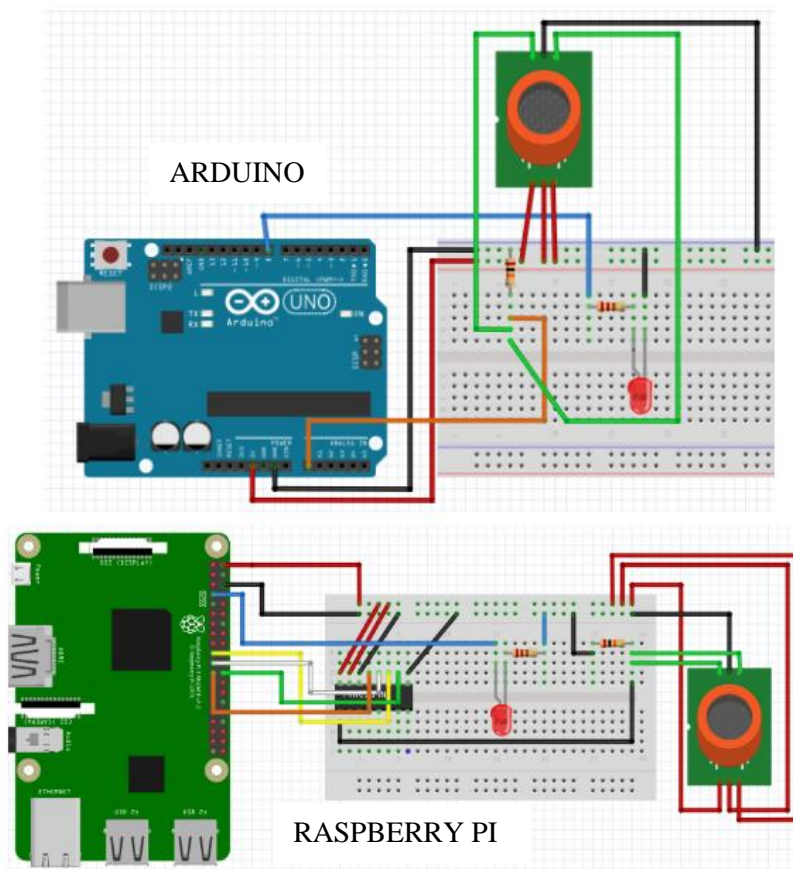


Figure 42: Fritzing Diagrams of Arduino and Raspberry Pi Gas Sensor Experiment

The students are asked to use a 20 kilo Ohm resistor but one of these was not included in their kit this means that the students will have to think resourcefully and use what they are given to get the same result. In this case, they were given two 10 kilo Ohm resistors that they could put into parallel in order to create the 20 Ohm resistor or they could just use the 10 kilo Ohm resistor and edit the code to reflect that. Now that the wiring was set up the students could move on to the coding section of the experiment. The setup code, which can be seen in Figure 43 below and viewed in Appendix 3, was run first and was meant to give the students a good baseline for the resistance of the sensor when it was fully heated up and ready to test.

```

from time import sleep
#Importing a delay function
from gpiozero import MCP3008
#Importing a library built for the MCP3008
#This means that you wont have to deal with the spi bus stuff

def Din_Resistance(gen) :
#Make a function to convert between your ADC value
#and the resistance of the sensor
    for value in gen :
#For every value that we put into this function
        yield(((5*20000)/(value*5))-20000)
#Put it into this equation and output the result
#This sets up the equation Ro=((Vo*R2)/(Voutput))- R2
# Vo is the Voltage being hooked up to the circuit
# R2 is the known voltage of the other resistor (20 kOhm)
# Vout is the voltage from the ADC
# The ADC puts out a number from 0 to 1 based on how close it
# is to the maximum so by mulitplying it by 5 volts we get our voltage
Din = MCP3008(channel=0)
# Setting up a variable to be equal to the output from the ADC

for Ro in Din_Resistance(Din.values):
#Define your variable that you are putting into the function
#So now all of the outputs from our function are called Ro
    print(Ro)
#Print the output
    sleep (1)
#Sleep for a second to let the computer catch up

```

RASPBERRY PI

```

//Set R2 to 20000 ohms
int R2 = 10000;
// Set Vo to 5 Volts
float Vo = 5;
// Declare all variables
float Vd,Va,Ro;

// The setup routine runs once when you press reset
void setup() {
// Initialize serial communication at 9600 bits per second
Serial.begin(9600);
}

// The loop routine runs over and over again forever
void loop() {
// Read the input on analog pin 0
Vd = analogRead(0);
// Change the digital input 0-1023 into a voltage between 0V-5V
Va = Vd*(Vo/1023);
// Take the known variables (Va,Vo, and R2) and
// using them to calculate the unknown R of the sensor
Ro = ((Vo*R2)/(Va)-R2);
// Print various values to make sure everything looks correct
// and trouble shoot your code
Serial.print(Vd);
Serial.print(",");
Serial.print(Va);
Serial.print(",");
Serial.println(Ro);
delay(1500);
}

```

ARDUINO

Figure 43: Gas Sensor Setup Code for Raspberry Pi and Arduino

The reason the students had to run the setup code was due to the imperfections in the manufacturing of the sensor that made each sensor's baseline resistance a little different so in order to get the best results the students needed to use this code to calibrate their sensor. After the students had gotten their sensors calibrated correctly, they had to edit three sections in the second code shown in Figure 44 below.

```

from time import sleep
#Importing a delay function
import math
#We will need this library to use exponents
from gpiozero import MCP3008
#Importing a library built for the MCP3008
#This means that you wont have to deal with the spi bus stuff

def Din_Resistance(gen) :
#Make a function to convert between your ADC value
#and the resistance of the sensor
    for value in gen :
#For every value that we put into this function
        yield ((.365*(((5*20000)/(value*5))-20000)/35000)**(-1.62))- .4)/5
#Put it into this equation and output the result
#This sets up the equation BAC=((.365*(Rs/Ro)^-1.62)-Calibration)/5
#Rs is calculated the same we calculated Ro in the previous code
#Ro is the Resistance that we calculated in the previous code (25 kOhm)
#The .365 and -1.62 are from the Rs/Ro equation that you calculated earlier
#The calibration is to take into account the varying alcohol levels in the room
# this should be set so that when you blow across the sensor you read 0 BAC
#The reason we divide by 5 is to convert between mg/L in breath to BAC
#the link to where I got this number will be in the lab description
Din = MCP3008(channel=0)
# Setting up a variable to be equal to the output from the ADC

for BAC in Din_Resistance(Din.values):
#Define your variable that you are putting into the function
#So now all of the outputs from our function are called Ro
    print("%.2f" % BAC)
#Print the output
    sleep (1)
#Sleep for a second to let the computer catch up

```

RASPBERRY PI

```

int R2 = 10000;
float Ro =110000;
float Vo = 5;
float Vd,Va, Rs,x,mgl,mglbreath,BAC;
void setup() {
    Serial.begin(9600);
}
// The loop routine runs over and over again forever
void loop() {
// Read the input on analog pin 0
Vd = analogRead(0);
// Change the digital input 0-1023 into a voltage between 0V-5V
Va = Vd*(Vo/1023);
// Take the known variables (Va,Vo, and R2) and
// using them to calculate the unknown R of the sensor
Rs = ((Vo*R2)/(Va)-R2);
// Use the Rs we have just calculated and the known
//starting resistance of the sensor
x = Rs/Ro;
// Use the relationship given in the data sheet to get
// the the mg/L at the given Rs/Ro value
mgl = .356*pow(x,-1.62);
// Calibrate the sensors to make sure that when the sensor
// is sitting it is reading 0 mg/L breath even though there
// is around .4 mg/l in the air according to the data sheet
mglbreath = mgl-.310;
// Using the link given in the lab convert mg/L to BAC
BAC = mglbreath/50;
// Print various values to make sure everything looks correct
// and trouble shoot your code
Serial.print(x);
Serial.print(" Rs/Ro,");
Serial.print(mglbreath);
Serial.print(" mg/L,");
Serial.print(BAC);
Serial.println(" BAC");
delay(1500);
}

```

ARDUINO

Figure 44: Gas Sensor Running Code for Raspberry Pi and Arduino

First, they had to edit the baseline resistance of their sensor which they got from the first code. The next thing the students needed to change was the equation to go from current resistance to mg/L alcohol in the air. Rather than just giving the students this equation without any real explanation besides “it came from the data sheet” the team decided that it would be best to have the students derive it themselves. The students would look at the MQ3 datasheet which included a graph that is shown in Figure 45 below and plug the data points as best they could into excel to get a Power series best-fit curve [52]. Once the students got their own values, they would replace the example values given in the code.

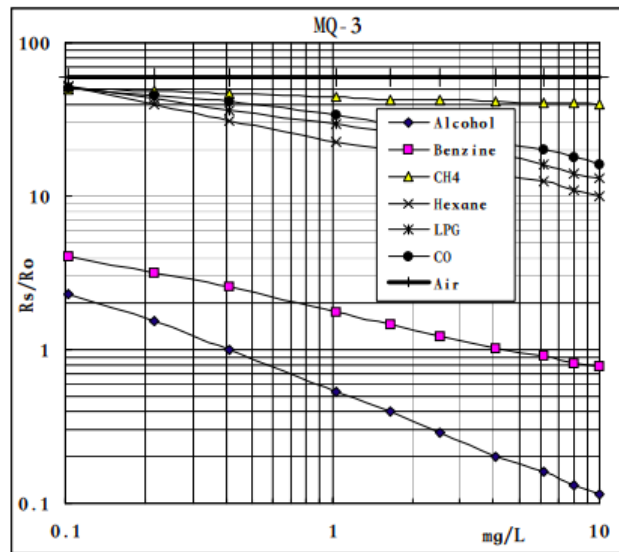


Figure 45: MQ3 Sensor Resistance vs. Concentration Data

Finally, they would need to adjust the equation that goes between mg/L in the air and mg/L in the breath. The datasheet says that when the resistance is at its baseline reading that there is about .4 mg/L of alcohol in the air which would throw off our readings because we are interested in the added amount of alcohol not the total amount. The students would then change this value by +/- .5 to get their specific sensor to read 0 mg/L in the breath when the sensor was not supposed to be reading anything.

The MQ3 was a good sensor for its price point but was not very breadboard friendly because the prongs on the bottom of the sensor are just slightly too large to fit into the breadboard and are not laid out in such a way that each of the pins would be isolated. This meant that the students had to solder wires to each of the prongs then plug those into the breadboard which gave the students more soldering experience but was largely an inefficient way to use lab time. I think in future iterations of this class they should opt to use an MQ3 sensor with a built-in breakout board on it. These sensors are slightly more expensive but come with both digital and analog outputs making them easy to use for both Arduino and Raspberry Pi and are usually breadboard compatible saving on wires and simplifying the wiring which will reduce confusion.

This lab was designed so that the students could easily troubleshoot their setups because the code does not just print the final answer that the students need to record but also the key steps in between. For example, in the first code for the Arduino, it prints the digital voltage, the analog voltage, and the resistance. This means that you can test your code with known values by plugging in a 3V source into the analog read pin and see what results you get. This helps students tell the difference between a software problem and a hardware problem which is a skill that the team hopes the students will take away from this class.

Module 6: Strain

Strain Gage

- Objectives:
 - Create a makeshift scale constructed from a TAL221 load cell and the corresponding HX711 load cell amplifier
 - Develop code that allows the HX711 load cell amplifier to produce force readings from the strain imposed on the TAL221 load cell
 - Use the load cell “scale” to measure the weight of the unknown weights provided and your smartphone
- Materials
 - Arduino
 - 1 Arduino
 - 2 sets of header pins
 - Additional wire for soldering
 - 5 male-to-female wires
 - 4 female-to-female wires
 - 4 hex spacers
 - 4 M3 screws and corresponding nuts
 - 2 MDF plates
 - 2 rubber feet
 - 1 TAL221 500g load cell
 - 1 HX711 load cell amplifier
 - 1 computer
 - Various-sized weights
 - Raspberry Pi
 - 1 Raspberry Pi
 - 2 sets of header pins
 - Additional wire for soldering
 - 9 female-to-female wires
 - 4 hex spacers
 - 4 M3 screws and corresponding nuts
 - 2 MDF plates
 - 2 rubber feet
 - 1 TAL221 500g load cell
 - 1 HX711 load cell amplifier

- 1 computer
- Various-sized weights

Procedure:

This particular lab was fairly straightforward with the code, as students were expected to examine the code given to them in order to properly calibrate their code and produce accurate readings from their load cells. However, much of the complexity from the lab derived from actually assembling the load cell scale together. This was done by soldering higher-gaged pieces of wire to the thin wires of the load cell, as well as the header pins for the load cell amplifier. Each of the load cell wires represents a different Wheatstone bridge node, which is detailed below in Figure 46 [53].

Wheatstone Bridge Node	Wire Color
VCC (E+)	Red
GND (E-)	Black
O+/S+/A+	White
O-/S-/A-	Green

Figure 46: TAL221 Load Cell Nodes

The load cell was then wired to the corresponding Wheatstone bridge nodes located on the HX711 load cell amplifier, and the other side was matched to the Arduino or Raspberry Pi as follows:

- VDD - 5V
- VCC - 5V
- DAT - any pin
- CLK - any other pin
- GND - GND

The load cell was then attached to the two pieces of MDF board using the M4 screws, with spacers in between each side of the load cell and the boards, shown in Figure 47 below [54]. This formed a Z-shape from the boards and load cell, which maximized the amount of torque created when force is applied onto either side of the makeshift scale and therefore produces the most accurate reading. Since the scale was slightly out of balance due to the screw heads not being flush with the board's surface, students had to attach rubber feet to the other side of the bottom MDF board in order to balance the scale properly.

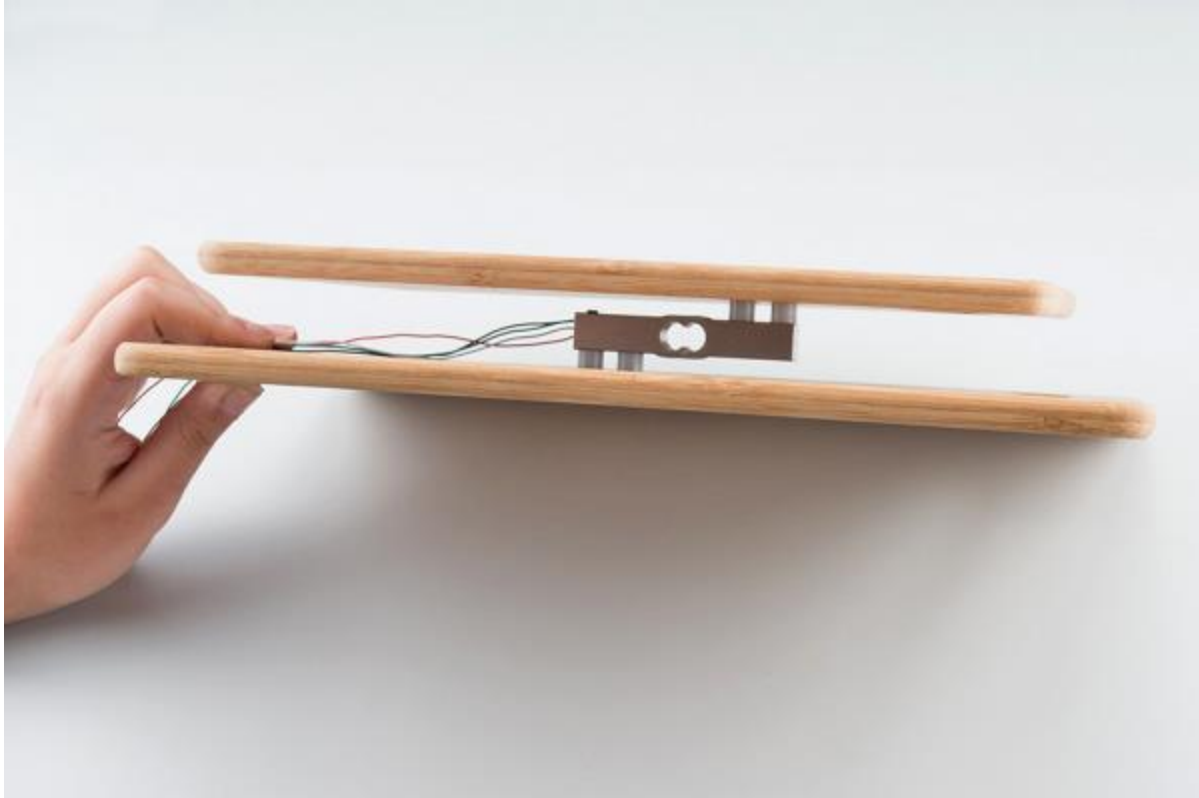


Figure 47: Example Strain Gage Experiment Setup

Once the physical setup was finished and everything was plugged in, students began calibrating the scale by running the provided calibration code, adjusting the calibration factor as needed based on the readings they were deriving from a given weight. The code for the Arduino can be found [here](#) [55], while the code for the Raspberry Pi can be found [here](#) [56]. Although the lab initially planned to use a set of predetermined weights, this was switched to a set of quarters that would serve as known weights for calibration. After the scale was properly calibrated, students continued to add quarters to their scales, taking multiple measurements for each unique weight in order to perform a regression analysis. Students then weighed their smartphones on their scales, once again taking multiple measurements to find the average and standard deviation for the weights of their smartphones. This was then compared to the manufacturer's specifications in order to determine the accuracy of their scale.

Students found the experiment quite straightforward to understand and execute, although there were a few issues relating to materials and setup. While the code was simple to understand and most students did not struggle with calibration, the TAL220 load cell was extremely fragile, and the wires would break off easily. Students also tended to spend a lot of time trying to screw in the spacers since they were threaded, and soldering took up a majority of the lab section time due to the heavy amount of soldering required as a result of the load cell wires being too small to plug into the female connectors securely. While this experiment would be good to continue offering in the course, the load cell and some of the other parts would have to be altered in order to ensure that students can carry out the experiment with adequate equipment.

Module 7: Motion

Accelerometer

- **Objectives**

- Create a system that can run an accelerometer using an Arduino or Raspberry Pi
 - Record a video of yourself completing the setup (or have a PLA sign-off on you doing the work)
- Write a program for an Arduino or Raspberry Pi that runs the thermistor
 - Ensure the code can run without any errors
- Run the code and examine the results
 - Observe the gravitational acceleration reading from your accelerometer
- Change the orientation of your accelerometer in which the X, Y and Z axes are affected by gravity
 - Determine if the system demonstrates the expected behavior or if further calibrations need to be made
- Calibrate the accelerometer based on the sensitivity of each X, Y and Z gravitational reading
 - Test the different functions of the accelerometer based on the different movements it will pick up from the program
 - Inactivity
 - Activity
 - Double Tap
 - Tap

- **Materials**

- Arduino
 - (1) Arduino
 - (5) female to male wires
 - (5) male to male wires
 - (1) breadboard
 - (2) 4.7k Ohm Resistors
 - (1) ADXL345 Accelerometer
- Raspberry Pi
 - (1) Raspberry Pi
 - (5) female to female wires
 - (1) breadboard
 - (1) ADXL345 Accelerometer

- **Procedure**

- Arduino
 - Code [57] (See Appendix 3)
 - Wiring (Figure 48 below [59])
 - Arduino Pin GND → ADXL345 GND
 - Arduino Pin 3V3 → ADXL345 VCC
 - Arduino Pin 3V3 → ADXL345 CS
 - Arduino Pin GND → ADXL345 SDO

- Arduino Pin A4 → ADXL345 SDA
- Arduino Pin A5 → ADXL345 SCL
- Raspberry Pi
 - Code [58] (See Appendix 3)
 - Wiring (Figure 48 below [59])
 - Raspberry Pi Pin 6 (GND) → ADXL345 GND
 - Raspberry Pi Pin 1 (3V3) → ADXL345 VCC
 - Raspberry Pi Pin 3 (SDA) → ADXL345 SDA
 - Raspberry Pi Pin 5 (SCL) → ADXL345 SCL

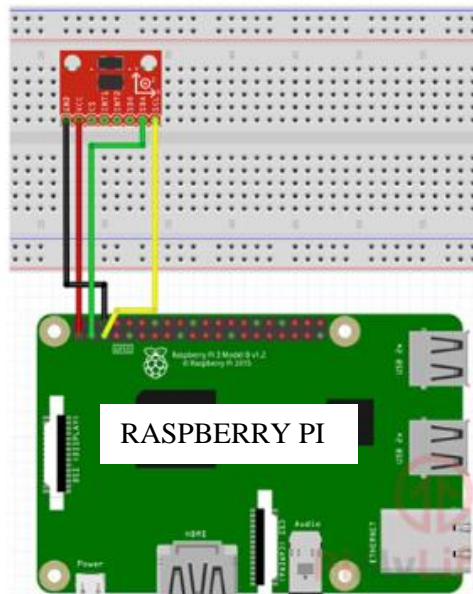
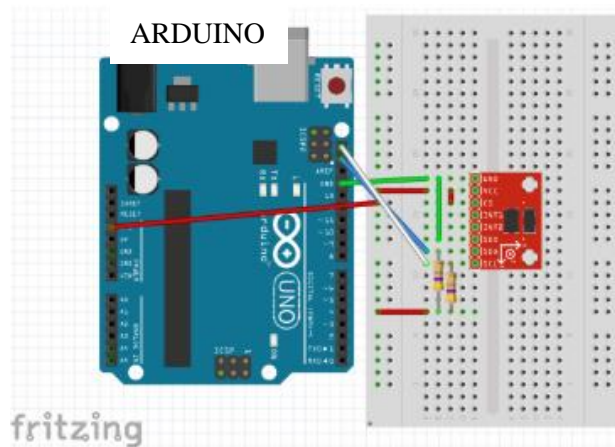


Figure 48: Fritzing Diagrams of Arduino and Raspberry Pi Accelerometer Experiment

Module 8: Open-Ended Project

The open-ended projects addressed a wide variety of areas. Some ideas that were proposed and executed were a mechanical sunflower that followed any light source, a vision detecting system connected to solenoids that played the “Guitar Hero” video game perfectly, a liquid dispenser with a proximity sensor as a safeguard, and a CNC pen plotter. Other projects had titles such as “Alcohol Lock Box”, “Smart Fridge Device”, and “Pressure Sensor Mat for Athletes”. These projects exemplified student creativity by integrating multiple sensors together to create complex devices. The projects were displayed at an end-of-term showcase. Figure 49 below shows an example of a plant monitoring system similar to ones designed by student groups in the class [60].

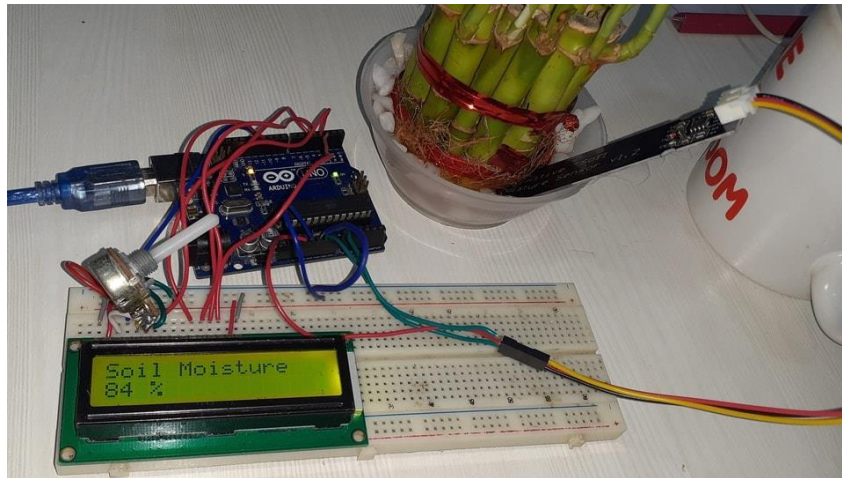


Figure 49: Example of a Plant Monitoring System

Results

ME 3902 Pilot Class-Theory

When designing a course like ME 3902, there is a large variety of important and interesting topics to cover but only a limited amount of class time in which to do so. This leads to some topics being covered more in-depth than others and some not being covered at all. For instance, at the beginning of ME 3902, the team hoped to create a module that covered flow meters and possibly some kind of fluids-related topic, like a depth or pressure gage, but they did not make it into the final version of the course. The main reason that certain topics were cut while others stayed was due to how expensive the necessary equipment is to use these sensors. ME 3902 is made to be a distance learning experience that comes in a small package in the mail that contains everything you would need to complete each lab. This box becomes more and more expensive with each additional sensor that is added to it, so in order to keep the cost low, the team handpicked the labs that could be done using only a few affordable components that still provided a good learning experience.

In the end, the team decided that doing one set of sensors per week would provide a good trade-off between covering a good variety of sensors as well as giving students time to fully understand the sensors that they were provided. The reason for selecting a week for each set of sensors rather than 3 days or 10 days is because a lot of classes at WPI are set up using this structure. This means that even when the material being covered is new each week, the students will have a certain familiarity with the structure of the course and have enough time to ask questions if they need to. Anything longer than a week and students that already have a good grasp on the material become bored and disengage with the class, and anything shorter than a week does not leave enough time for students that are struggling to catch up and ask questions. Giving the greatest number of students a challenging but doable workload was the team's main goal when selecting the amount of time to be spent on each subject. As a byproduct of this goal, the boxes stayed low cost and the team was allowed to really explore the selected topics.

When using this week-by-week based system the first suggestion to the team was to use modules to set up the course. Modules, for those who are not familiar, are a way to break up a larger course into smaller sections usually focusing on a specific topic or activity in a course. Using modules makes the larger course more approachable because it organizes a lot of information into an easy to navigate format. For example, if you need to find a certain equation that helps you to calculate the cycle life of an engine in a traditional course, you would likely have to remember the date that the teacher presented that information or what chapter of the book the equation would be in and search through that chapter. This makes searching for the information difficult and time-consuming. However, if the class was set up using modules the student would first go through the list of modules until they found the module for engine design, then inside of that module, the student would search for the specific lecture or handout labeled cycle life. This makes the information easily accessible long after the course is finished and makes completing work for the class easier since the information is more readily available.

Another strength of modules is when a course's subject matter is very broad, and most parts of the course can be completed independently of each other. Each of the modules can be treated as their own separate class with a quiz or lab report at the end. This allows the students to focus on one concept per module and really understand it and allows the teacher to monitor the students' progress to see where they

need to put more time when reviewing for the final. If all of the students did well in one module of the course and poorly in another, then when it comes time to review for the final exam or project, the professor can spend more time on the material that the students did not understand the first time. For all of those reasons, the team thought that using modules would be a great way to set up ME 3902.

Now that the team had chosen to use modules and had a good idea of what sensors they wanted to cover; it was time to choose the order in which these modules would be presented. Logically, the module that taught the students how to set up their microcontroller and the basics of how-to code would be presented first, but after that, the team needed some way to decide which order to present the modules because they were all essentially independent. The team thought that the biggest hurdle that the students would have to overcome was the coding to make these sensors work. For the most part, if you understand what the purpose of the wiring is, such as bringing power to a certain component or taking a signal from a sensor, the wiring is ideally intuitive. However, when it comes to coding, everything in your code could be right but you miss one semicolon at the end of a line and your code will not work. Most of the ME students also have more background knowledge of ECE than they do with CS, so the team thought that they should choose the order of the modules with this in mind. As the course progresses, the students will become more and more comfortable with the coding side of the class, which means that they can do more independent work with it.

The team decided the best way to keep all of the students engaged was to break the class into 3 levels based on the required knowledge needed to complete them. The first level was set up as an example of what the code should look like and what the different functions did. These labs would have the code completely done and heavily commented so that the students could look back and reference them at any time in order to see how to set up certain functions or how to define certain variables. These labs were selected because they were easy to read and understand, short in length, and covered most of the skills students would need to complete the course. These modules ended up being temperature and proximity since they used a lot of libraries which contain a lot of pre-written functions that the students can use and customize based on their specific wiring and basic computation like reading a voltage and using math to turn that signal into the value you are interested in like a distance or temperature.

The second level of labs was designed so that the students were given most of the code that they needed but would need to come up with a few things on their own and add them to the code. These modules ended up being motors and concentration because they utilized either a more complicated structure for the code or multiple different codes that built on one another. For example, in one of the labs, the students were given a code that printed a variable based on the change in resistance of the sensor but had to add a function at the bottom that would turn on a light if that variable went above a certain threshold. These activities helped the students become more confident in their coding ability before they had to do a lot of independent work for the later labs and their final project. The third and final level of labs focused on the most complex code to understand and had multiple ways to tackle the problem which required the students to combine all of their previous knowledge together in order to complete these labs. These ended up being the strain lab and the acceleration lab because the students had to include libraries in conjunction with their own code in order to get these labs to work. These labs presented their code in a way that focused more on what the code should do rather than exactly how to do it, which is how all of the other labs had been up to that point.

For many ME students, this class was their first time being exposed to any kind of coding and some students' first time being introduced to more complex circuits. For this reason, the team made it mandatory that the first week would cover the basics of both wiring and coding for the students' respective microcontrollers. This meant teaching the Arduino students how to use the void loop and void set up as well as a few functions like pin mode and print. For Raspberry Pi, the main focus was on getting the Pi's connected to the school Wi-Fi and teaching them the basics of Thonny, which is a simplified version of Python that the students were going to use for their labs. The majority of students seemed to pick up the coding side of things very fast but seemed to struggle a bit more on the wiring side of the first lab. The team set up this first module with the assumption that most of these students had taken ECE2010, which is a sophomore-level course where students are required to set up circuits and gain experience with electrical engineering concepts. For those students who had taken this course, this part of the lab was trivial, but for those who had not, the lab did not go into enough detail. The actual breakdown of the class only ended up being around 50% of juniors, with almost 40% of the class being sophomores, which can be seen in Figure 50 below.

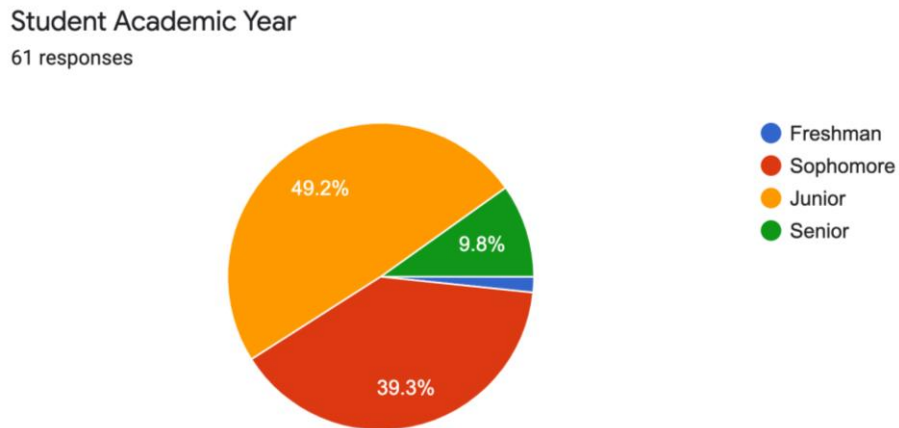


Figure 50: ME 3902 Pilot Class Student Academic Year Demographics

Module 1: The Microprocessor-Computer

The main two problems that the team helped to iron out the first week on the wiring side of things was that LEDs have polarity, so they need to be plugged in the correct way and students must be told how the holes on a breadboard are connected electrically. These are small things that could be added to the course that would make it a lot more approachable for someone who has never made a circuit before or had any experience with wiring. The largest problem of the first week was getting the students that had a Raspberry Pi connected to WPI's Wi-Fi. All of the TA's and PLA's had set up their Pi's using their personal Wi-Fi at home but had not set one up using WPI's certificate system, which added a lot of layers of complexity to what should have been a simple step. This would not be a problem during a distance learning scenario like there was in D term because everyone would have to connect to their home Wi-Fi, which has a lot of online tutorials. If this class were to be offered again at WPI it would be very helpful to have someone set up a video on exactly what to do.

Module 2: Temperature

The temperature module was broken down into two different sections, one covering the silicon band gap sensor and one covering the thermistor. This module was also intended to cover thermocouples but due to an ordering mix up, they were omitted from this pilot course. For the most part, these labs did a great job at teaching students how their respective sensors worked but suffered from some topics not being covered very well during the introduction module. The main issue with the labs themselves had to do with the inclusion of an ADC for those using the Raspberry Pi. Due to the Raspberry Pi using only digital inputs and most of the sensors outputting an analog signal, many of the labs for Raspberry Pi needed to include this piece of hardware so that the Pi could read the incoming signals. The Arduino has one of these ADC's built-in but only has 10 bits of accuracy, which is not sufficient for very precise measurements like thermocouples but gives enough accuracy for all of the other experiments. The chip that the team selected was the MCP3008 which uses SPI (Serial Peripheral Interface) bus in order to communicate with the microcontroller, but the team had a hard time understanding and explaining to the students. Most of the code used for the interpreting of these signals was sourced from open-source coding sites like GitHub, where people with more experience can post sections of code to be used by others.

However, because these bits of code were not easy to understand or use, it led to a lot more confusion than was necessary. Luckily, midway through the module, one of the team members found a coding library for the MCP3008 chip that allowed the students to use one function and just plug in the pins that they wanted to use which made the labs run a lot smoother. Once this first lab was out of the way, the other labs went much smoother for the Pi because the team changed the code to use this library instead of the more confusing original code. Another difference between the labs was the amount of wiring necessary to complete these laboratory exercises. As mentioned before most of the labs for the Raspberry Pi needed to include the use of an ADC, which took some labs from a simple 3-wire setup with positive, ground, and a signal wire to around 11 wires because the ADC required 8 wires to set up: 2 positive, 2 ground, MISO, MOSI, Chip Select, and Clock. This made the wiring for the students more difficult and mistakes harder to catch for students and instructors, which you can see in Figure 51 below. This would be an even larger problem when being used in a distance learning setting where a student would have to show their work through a camera rather than pointing to the physical object on a table.

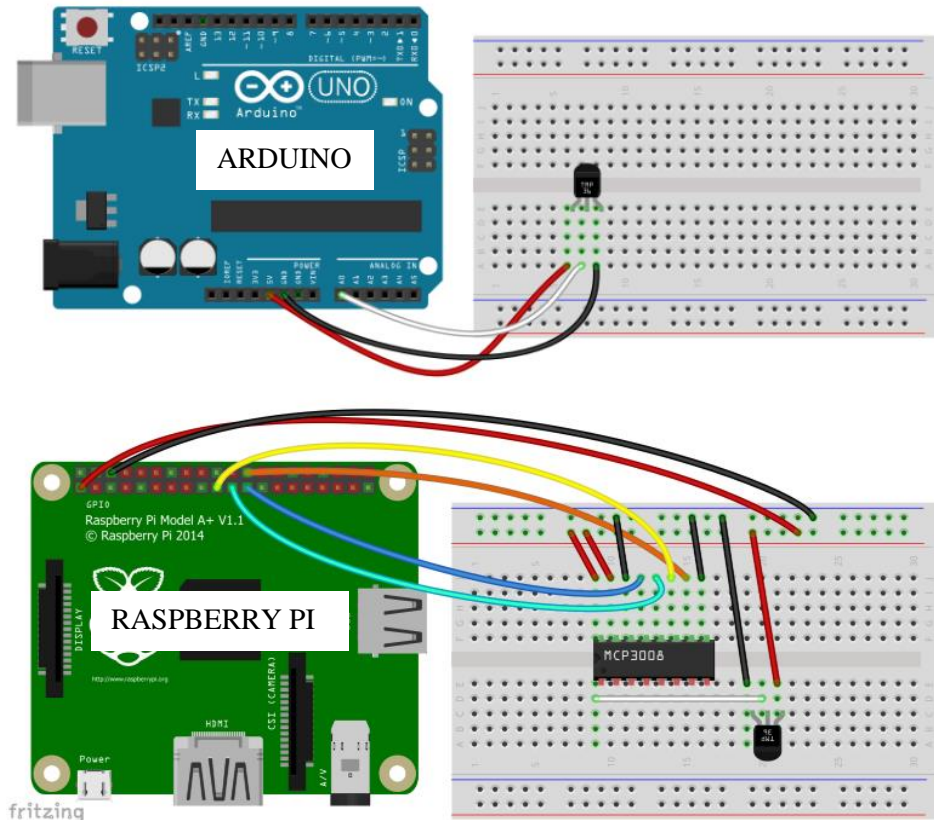


Figure 51: Fritzing Diagrams of Arduino and Raspberry Pi Experiment with and without ADC

Module 3: Proximity

The proximity module consisted of the IR motion sensor and the ultrasonic (sonar) sensor. The ultrasonic lab was very smooth from a coding and wiring perspective because there was no need for an ADC or complex wiring. The main issue with this lab was the inconsistent data that these sensors were outputting. All of these sensors used the same frequency, so if two of the sensors were pointed at each other they could trigger each other's echo pins, making the readings inaccurate. This would not be a concern for a distance learning environment because there would only be one sensor being used in the lab but when there are 20 sensors going at the same time it caused some problems. The other issue with the sensors was that if they were used to measure the distance of a curved or angled surface, this caused the sound waves to bounce from the sensor to the object to the ceiling or floor back to the object then back to the sensor. This made some of the distances measured a lot longer than they should have been, but this issue was fixed by having the students use a wall as a reference and move the sensor closer and further away which fixed the issue.

The second portion of this lab was using IR motion detectors. These do not measure distance but rather detect whether or not there is any kind of motion happening in their field of vision and send a signal accordingly. The IR sensors the team chose to use was the HC-SR501 sensor, which seemed simple. It contained a positive wire, ground wire, and a third wire that was set either high or low depending on whether or not motion was detected. Where the problem arises is with the sensitivity of the sensor and the amount of time that it takes to reset to a neutral position. There is a delay knob on the side

of the sensor that the team thought they could use to shorten the amount of time that the sensor will trigger when motion is detected, but what this actually does is adjust the delay between when the sensor detects motion and when it sends that information to the microcontroller. This means that you cannot adjust the amount of time that this sensor is sending the trigger signal for, which is around 30 seconds, making testing hard and time-consuming. These IR sensors also had a problem with false positives, with some sensors being affected more than others. This meant that sometimes there would be no motion in front of the sensor, but it would trigger anyway. The team believes is due to high-frequency noise in the wires because most of these issues went away when the students were instructed to use two wires wrapped around each other rather than using just one. This concept is called a twisted pair which is meant to introduce more impedance to the circuit by coupling the two magnetic and electrical fields of the wires which cuts down on the noise in the signals which the team believed was causing the issue.

The last issue with the sensor was the field of vision, which the datasheet claimed to be 110 degrees. When these sensors were tested, it was apparent that these sensors did not have the field of view that the manufacturer was stating until the team looked at the datasheet again and found that they were not measuring the angle between the edge of the cone and the center of the cone. The angular measurement provided by the manufacturers actually represented the angle of the arc needed to create the cone that the sensor could measure within, shown as theta in Figure 52 below [61]. In the picture on the left, theta represents the cone angle of the cone on the right. The team thought that the cone angle referred to the angle between the sides of the cone on the right, but it actually refers to theta on the left. This meant that their range was a lot closer to a beam rather than the cone than the team had envisioned, leading to a lot of confusion when it came to what was triggering the sensors and what was not, especially at longer ranges. Once these obstacles were overcome, the rest of the module went smoothly.

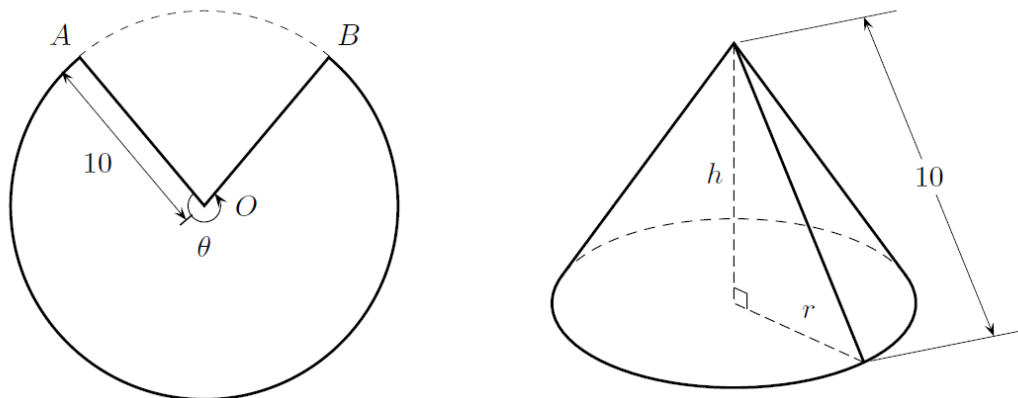


Figure 52: Cone Angle Diagram

Module 4: Motors

This module covered how to control DC, servo, and stepper motors using both the Arduino and Raspberry Pi. The DC motor lab went very smoothly and there were not a lot of problems on the coding or wiring side of this lab. There was an issue however when it came to soldering the H bridge, which is the type of motor controller the team selected for this module. For a lot of students, this was their first time trying to solder, so there were issues with using too much solder, which made two adjacent pins

electrically connected when they shouldn't be. There was also a problem with some students not using enough solder and their connections coming loose mid-experiment. This could have been mitigated by giving students a short overview of how to solder during the introduction module or passing around an example board that was soldered correctly. Since most students only used one motor at a time, the students could have just used a PWM pin and ground to run this lab and cut out the controller altogether, but the team thought that it was important to give the students experience with these types of controllers, so they were included in this portion of the lab. The servo lab went relatively smoothly except for a few parts of the code that needed more explaining. In the code that controlled the servo motor for Raspberry Pi, the code used a percentage of the maximum angle in order to control how far the servo turned, but in the code for the Arduino, the students used the angle directly, which led to some confusion when the students tried to compare the two codes. This difference in style was due to the fact that Arduino used a library that converted the direct angle into a percentage, but that all took place in the background where the student could not see.

Another problem with the servos was not due to the lab but due to the servo selection because this servo did not have as much power as the team expected and only had 90 degrees of freedom with which to spin. This was fine when being used in a lab to demonstrate how to use the motor but not when it is being used to open and close hinges and lids because it simply could not handle the load. As a result, many students had to buy more powerful servo motors for their final project. The final section of this module covered stepper motors, which went a little too well. The team selected a five-wire stepper with a custom driver board which worked very well in conjunction with the code, but the professors thought that this made the wiring of the boards too easy because they just had to give the board power and plug the other wires in using the custom plastic connector. The professors would have rather covered the much more popular 4 wire stepper motor and use the H bridge from the DC lab in order to complete this experiment because the students would get a better understanding of how the stepper motor functions. The team also discovered a [YouTube video](#) that showed a 4 wire stepper motor being stepped by hand, which the team thought could be used as an interesting demonstration for what exactly the code is doing and how the steppers work [62]. The main reason the five-wire steppers were chosen over the four wires was due to pricing concerns, as they were half of the price of the NEMA 17 motors that the team was familiar with from other courses.

Module 5: Concentrations

This module covers concentrations of substances using an MQ3 alcohol sensor. This lab went well for the most part but the largest issue that the students encountered was getting the sensor to give consistent accurate readings. Because the sensor needs to be heated in order to work, a small difference in temperature can cause a large difference in initial resistance, meaning that the students were not allowed to touch the sensors and had to hold on the wires connecting to the sensor. This led to less control than the students would have liked and caused some students who did not put enough solder on their joints to lose wires while they were trying to take measurements, creating a lot of frustration. The last major issue with the concentration module was with the consistency of the “people” being tested. Obviously, we could not use real alcohol in order to test these sensors, so the team had to mix isopropanol with water in different proportions to make three different concentrations of “alcohol” for the students to test. This alcohol would evaporate and not mix thoroughly leading to two different samples poured from the same container testing for drastically different levels of alcohol in the breath by the same experimental setup. This caused

the students a lot of trouble when trying to verify their results with other students who had completed the same lab but got completely different answers. The only other thing that made this lab confusing was the difference between the code for the Arduino and the code of the Pi. In the code for the Pi, a lot of different steps had to be combined into one because the team member that developed this module had to create the code completely from scratch. The way that Thonny was set up makes it a lot easier to combine all of the functions into one larger equation, but it makes the code less clear to the students. The team used a lot of comments that explained what this equation did, but for some students, it was easier to compare the Pi code to the Arduino code so that they get a better understanding of what was going on. If the team had more time, this would be the one piece of code that they would like to develop further.

Module 6: Strain

This module covered strain using a strain gage and an amplifier. This module worked well from the coding and wiring standpoint but had issues when it came to the hardware. The strain gages that the team chose to use, which can be seen in Figure 53, had a liquid-rubber coating where the wires were attached to the strain gage [63]. However, due to the low build quality, these wires would just fall out of the device, making the strain gage useless without the complete removal of the liquid rubber and re-soldering of the wires. In order to prevent as much damage as possible, the students were instructed to use glue or tape to make a strain relief so that when the wires were pulled, they would not become detached from the strain gage.

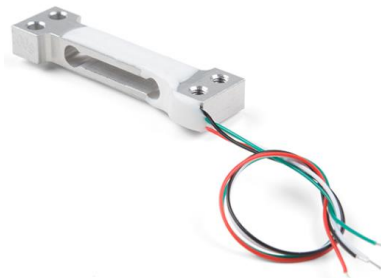


Figure 53: Strain Gage Used In ME 3902 Pilot Class

Another problem the team ran into with the hardware was the spacers used to separate the strain gage from the top and bottom of the scales. Due to an ordering mix-up, the team actually ended up receiving standoffs rather than spacers, which meant that if the threads were not formed well in around half of the spacers. This meant that they needed to be drilled out, which took a lot of extra time and effort. The tops and bottoms of the scales were also made out of laser-cut medium-density fiberboard (MDF), but the MDF provided to the team was too thick to be laser-cut effectively. In future offerings of this class, using acrylic or 0.25-inch birchwood would be a much better option.

Module 7: Motion

The final structured module was acceleration which focused on a three-axis accelerometer. Unfortunately, this lab was made optional due to time constraints with the course because of ordering issues and the course getting off to a slow start. However, the students that did complete this module had a pretty easy time. This lab did not require the use of an ADC for Raspberry Pi because the sensor that the

team selected already came with an ADC built into it. The code was the longest in length, but this was primarily due to the code repeating itself for each of the X, Y, and Z-axes. There was also a library for this sensor that had a read function built into it, making it even easier to code for the students.

The one issue that some people did have with this lab was their inexperience with soldering. During Module 4, there were a lot of students who had never soldered before so it makes sense that this would also cause some issues only a few weeks later. Most of the problems came from either using too much solder, which connected two wires that should have been separated, or not using enough, which made poor electrical connections that created inconsistency between different experiments. This was something that the students just needed more hands-on experience with because soldering is not something that can just be taught using theory, but by the end of the class, every student had become a lot better at soldering than when they started.

Module 8: Open-Ended Project

The “final test” for the course was the open-ended project, where they would have to combine three different sensors into one project. This module was the most successful overall because the students had the longest time to work on it, and they each got to pick their own topic, which made the students very passionate about their work. This module also succeeded because each of the projects was assigned a PLA, TA, or a professor to help them with their project, giving the students the ability to really push themselves to make difficult and awesome projects knowing that they had a resource to reach out to for help. These projects might have suffered a little if they were done in a distance learning environment since a lot of the projects involved using 3D-printed or laser-cut parts, which are harder to come by when not at a university. While all of the coding and wiring could be done at the same level, the projects would not have the build quality they had during the pilot course. The team was very proud of each and every project that the students put forth and could see that they all had poured a lot of time and effort into making these final projects as good as they could be.

ME 3902 Pilot Class in Practice

As the term progressed, the students became more and more confident in their skills in both coding and wiring, which you can see in Figure 54 below. This chart was derived from a survey that tracked the number of major questions asked about each module, which gradually decreased as students began familiarizing themselves with basic concepts. Even though very few questions appeared to have been asked during the last couple modules based on the survey results, the students asked just as many questions during the first module as they did the last; however, the type of questions changed. During the first week, most of the students were asking background questions like how to wire certain things or how a certain function worked, but towards the end of the term, the students were asking educated clarifying questions. These were more to double-check a hunch that they had or to make sure that they fully understood a concept before moving forward. The team was very proud of all the students that participated in the course, being able to experience the students’ growth firsthand as the questions transitioned from the “How and What” stage into the “just making sure” stage. A full list of all of the major questions asked can be found in Appendix 2.

Module Number
61 responses

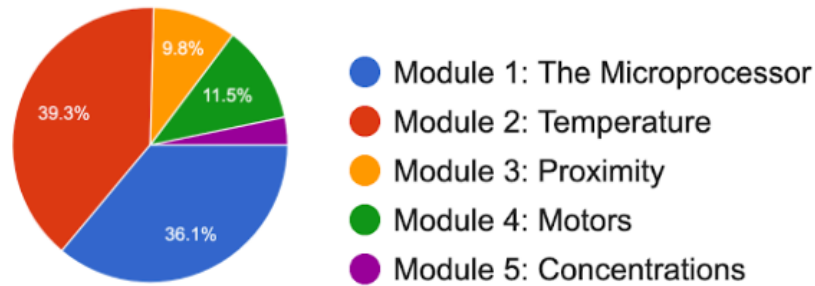


Figure 54: Summary of Student Questions by Module

Overall, the pace of the class was about what the team expected it to be. However, because it was a pilot class, all of the kinks were not fully worked out, which meant that some of the modules took longer to present than others. This meant that module 7, the acceleration lab, became optional due to some of the students needing that extra time in order to complete the other experiments and their final projects. This issue was exacerbated by the class getting a slow start due to the Raspberry Pi Wi-Fi issues, as well as the temperature and motor modules taking a little longer to complete than intended due to the lengthier and more numerous lab reports required. In the future, the team recommends combining the labs for each module into one document and consolidating some of the information to keep the students from doing more repetitive work so they can focus more on understanding the labs than writing the reports.

Midway through the term, one of the team members came up with the idea of checking students off if they came into the lab and got their set up working. This meant the students that came to the lab would not have to rewire their setups at home in order to record the video needed to complete the lab, which helped a little bit on cutting down the unnecessary parts of the course. This class, for being the team's first attempt at creating a class, went very well. The students were great at rolling with the punches and giving the necessary feedback that the team needed to make the course better as it went along. At the end of the day, all of the students came out of the class more confident in their abilities to use microcontrollers, their selection of sensors, and a project that they completed almost completely independently that would have been out of their comfort zone at the beginning of the course.

Conclusions

Ultimately, the course designed during the length of this project is a jumping-off point for future implementations of online-based engineering experimentation courses. Revisions to the course could certainly be made to improve the clarity of message from the course as well as the smoothness of executing the modules. Some of the areas of improvement that were identified during the course operation were:

Improve Module 1 Content

Module 1 functioned as an introduction to the micro-controllers themselves and the principles of each. Simple experiments were provided to aid in the setup of both the Arduino and Raspberry Pi. In future revisions of the course, this module should be expanded to contain detailed basics of electrical engineering such as a simple introduction to circuits, the use of ground, the operation of resistors and other such necessary introductory material. It was noticed that due to the nature of the course and its availability, students were undertaking the experimentation course before completion of other mandatory courses that established this base. Teaching the basics of electrical hardware would be a strong introduction to the course.

Diversify Types of Sensors Used

The first run of the sensor box that was offered was adequate but by no means perfect. Improvements can be made to the box in many different ways. One such way could be to include more sensors with a digital or analog interfacing option. It oftentimes became very challenging to use analog sensors with the Raspberry Pi since the microcontroller can only accept digital input. The Analog to Digital Converters (ADC) were unwieldy to set up and utilize but were necessary to most sensors currently utilized with the Raspberry Pi. A higher quality breadboard would provide a nice quality of life for students, especially since on some occasion some components overheated and caused the plastic on the current breadboards to melt. The strain gage experiment would also benefit from an alternative gage, perhaps one constructed in-house at WPI. The current strain gage utilized very thin wires that snapped after very light operation. The construction of the equipment box should also be outsourced to a third party separate from the staff currently running the course. Work-study students could assemble the box within given specifications to ensure that all the correct components are included.

Focus on One Microcontroller Type

In the future, the course should better explain the differences between the two microcontrollers such that the students have the best information to choose between them. During this project, some students became increasingly frustrated with the complexities of the Raspberry Pi and eventually switched to the Arduino due to its simplicity. While the Raspberry Pi is still an important part of the course and teaches many important principles that the Arduino cannot such as the operation and importance of ADC's and data resolution, the Raspberry Pi on average had a much more difficult time setting up the experiments. Further research should be made to conclude if the inclusion of the Raspberry Pi helps students learn through complexity or obstructs learning through frustration and difficulty.

Create Independent Class Resources

Currently the course sources for experiments are scattered across multiple websites with varying availabilities. Some of the course content is already unavailable, which provided frustration for students during the first module. Utilizing GitHub to compile all of the code and experiment information would be an effective way to future-proof the content of the course against the reliance of multiple other websites. Students would also learn the principles and operation of GitHub, while staff administering the course could easily track the changes done to the course over time through GitHub. Everything being condensed into one repository would also help reduce confusion amongst students.

References

- [3] ABET. "Criteria for Accrediting Engineering Programs, 2018 – 2019." *ABET*, ABET, 2018, www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2018-2019/#GC1.
- [28] Adafruit Industries. "918 STEPPER MOTOR PM GEARED UNI 12V." *DigiKey*, 2020, www.digikey.com/product-detail/en/adafruit-industries-llc/918/1528-1367-ND/5629415?utm_adgroup=Stepper%2BMotors&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Motors%2C%2BSolenoids%2C%2BDriver%2BBoards%2FModules_NEW&utm_term=&utm_content=Stepper%2BMotors&gclid=EAAlQobChMI14G2sO2f6QIVAZyzCh3xRg-cEAQYASABEgIpYvD_BwE.
- [6] Adafruit Industries. "Raspberry Pi 4 Model B - 1 GB RAM." *Adafruit Industries Blog RSS*, Adafruit, 2020, www.adafruit.com/product/4295?src=raspberrypi.
- [46] Adafruit Industries. *TB6612 To Arduino UNO R3 with 2 DC Motors*. Adafruit, 2017, forums.adafruit.com/viewtopic.php?f=25&t=111302.
- [38] Adafruit Industries. "TMP36 Temperature Sensor." *Adafruit Learning System*, 2020, learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor.
- [12] [15] Ametherm. "PTC Thermistors vs. NTC Thermistors for Inrush Current." *Ametherm*, Ametherm Circuit Protection Thermistors, 21 June 2019, www.ametherm.com/blog/inrush-current/ptc-thermistors-vs-ntc-thermistors-for-inrush-current/.
- [5] Arduino. "Arduino Uno - R3." *DEV-11021 - SparkFun Electronics*, SparkFun, 2020, www.sparkfun.com/products/11021.
- [16] [17] Arduino. "HOW PIR SENSOR WORK." *Arduino Project Hub*, Hackster.io, 2020, create.arduino.cc/projecthub/diy-partners/how-pir-sensor-work-9f76b6.
- [31] Banggood. "LM393 MQ3 MQ-3 Sensor Ethanol Gas Analog Sensor TTL Output Module Board from Electronics on Banggood.com." *Www.banggood.com*, 2020, usa.banggood.com/LM393-MQ3-MQ-3-Sensor-Ethanol-Gas-Analog-Sensor-TTL-Output-Module-p-1536606.html?cur_warehouse=CN.
- [2] Best Value Schools. "Is Attending College Online Cheaper Than Attending a Traditional College? - Best Value Schools." *BestValueSchools.com*, 1 July 2019, www.bestvalueschools.com/faq/is-attending-college-online-cheaper-than-attending-a-traditional-college/.
- [4] Canvas. *Project-Based Engineering Experimentation*, WPI, 2020, <https://canvas.wpi.edu/courses/19478>.
- [11] Chiel. "Temperature Sensor - TMP36." *SEN-10988 - SparkFun Electronics*, SparkFun, 2020, www.sparkfun.com/products/10988.

[40] Circuit Basics. *Make an Arduino Temperature Sensor (Thermistor Tutorial)*. Circuit Basics, 22 June 2018, www.circuitbasics.com/arduino-thermistor-temperature-sensor-tutorial/.

[13] CircuitBread. “How Do I Protect My Circuit from Inrush Current?” *CircuitBread*, 2020, www.circuitbread.com/ee-faq/how-do-i-protect-my-circuit-from-inrush-current.

[9] Dave. “Types of Thermocouples with Temperature Ranges & Color Codes.” *WatElectrical.com*, WatElectrical, 10 Apr. 2019, www.watelectrical.com/types-of-thermocouples-with-temperature-ranges-color-codes/.

[29] DFRobot. “Gravity: Analog PH Sensor / Meter Kit For Arduino.” *DFRobot*, 2020, www.dfrobot.com/product-1025.html.

[19] DiyI0t. *Ultrasonic Sensor Tutorial for Arduino and ESP8266*, 2020, <https://diyI0t.com/ultrasonic-sensor-tutorial-for-arduino-and-esp8266/>.

[26] Earl, Bill. “All About Stepper Motors.” *Adafruit Learning System*, Adafruit, 2020, learn.adafruit.com/all-about-stepper-motors.

[58] [59] Emmet. *Raspberry Pi Accelerometer Using the ADXL345*. Pi My Life Up, 16 Nov. 2019, pimylifeup.com/raspberry-pi-accelerometer-adxl345/.

[30] Gaslab. “AlphaSense O2-A2 Oxygen Sensor.” *Gaslab.com*, 2020, gaslab.com/products/oxygen-sensor-alphasense-o2-a2?utm_medium=cpc&utm_source=google&utm_campaign=Google%2BShopping&gclid=CjwKCAjw7LX0BRBiEiwA_gNw6HFR94QBtpoESLtcS_1tivKS5Wv6nPfimDQscb2hmHGfc9scQdGDRoCL8gQAvD_BwE&variant=7040537657379.

[39] Gpiozero. “Gpiozero Basic Recipes.” *2. Basic Recipes - Gpiozero 1.5.1 Documentation*, Gpiozero, 2015, gpiozero.readthedocs.io/en/stable/recipes.html#measure-temperature-with-an-adc.

[51] Hawkins, Matt. *Stepper Motor Test RaspberryPi*. Bitbucket, 28 Sept. 2015, bitbucket.org/MattHawkinsUK/rpispys-misc/raw/master/python/stepper.py.

[60] How To Electronics. *Interface Capacitive Soil Moisture Sensor with Arduino*. How To Electronics, 3 May 2020, how2electronics.com/interface-capacitive-soil-moisture-sensor-with-arduino/.

[24] HowToMechatronics. “How Servo Motor Works & How To Control Servos Using Arduino.” *HowToMechatronics*, HowToMechatronics, 21 Apr. 2020, howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/.

[34] HowToMechatronics. “MEMS Accelerometer Gyroscope Magnetometer & Arduino.” *HowToMechatronics*, 5 Apr. 2020, howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyroscope-magnetometer-arduino/.

[44] HowToMechatronics. “Ultrasonic Sensor HC-SR04 and Arduino Tutorial.” *HowToMechatronics*, 2 May 2020, <https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/>.

[36] Instructables. “Arduino Class.” *Instructables*, Instructables, 24 Apr. 2018, www.instructables.com/lesson/Your-First-Experiments/.

[49] Instructables. “Servo Motor Control With Raspberry Pi.” *Instructables*, Instructables, 20 Sept. 2017, www.instructables.com/id/Servo-Motor-Control-With-Raspberry-Pi/.

[47] Levine, Zach. “How to Control a DC Motor (Or Motors) Using Your Raspberry Pi.” *Howchoo*, Howchoo.com, 1 Dec. 2019, howchoo.com/g/mjg5ytzmnjh/controlling-dc-motors-using-your-raspberry-pi.

[61] Mathematics Stack Exchange. *Finding Angle of Sector Which Forms a Cone*. Mathematics Stack Exchange, math.stackexchange.com/questions/728900/finding-angle-of-sector-which-forms-a-cone.

[43] MBTechWorks. “Interface PIR Motion Sensor with Raspberry Pi.” *How to Interface and Program an HC-SR501 PIR Motion Sensor to the Raspberry Pi*, 2 Nov. 2017, www.mbttechworks.com/projects/pir-motion-sensor-with-raspberry-pi.html.

[35] Murphy, Chris. “Choosing the Most Suitable MEMS Accelerometer for Your Application-Part 1.” *Choosing the Most Suitable MEMS Accelerometer for Your Application-Part 1 | Analog Devices*, Analog Devices, 2020, www.analog.com/en/analog-dialogue/articles/choosing-the-most-suitable-mems-accelerometer-for-your-application-part-1.html#.

[10] N/A. “Band Gap.” *Wikipedia*, Wikimedia Foundation, 27 Apr. 2020, en.wikipedia.org/wiki/Band_gap.

[20] N/A. “Direct Current.” *Wikipedia*, Wikimedia Foundation, 27 Apr. 2020, en.wikipedia.org/wiki/Direct_current.

[21] N/A. “Electric Motor.” *Wikipedia*, Wikimedia Foundation, 5 May 2020, en.wikipedia.org/wiki/Electric_motor.

[23] N/A. “Servo (Radio Control).” *Wikipedia*, Wikimedia Foundation, 23 Apr. 2020, [en.wikipedia.org/wiki/Servo_\(radio_control\)](https://en.wikipedia.org/wiki/Servo_(radio_control)).

[27] N/A. “Stepper Motor.” *Wikipedia*, Wikimedia Foundation, 27 Feb. 2020, en.wikipedia.org/wiki/Stepper_motor.

[32] National Instruments. “Measuring Strain with Strain Gages”. *Innovations*, 14 Mar. 2019, <https://www.ni.com/en-us/innovations/white-papers/07/measuring-strain-with-strain-gages.html>

[50] NikodemBartnik. *NikodemBartnik/ArduinoTutorials*. GitHub, 1 June 2018, github.com/NikodemBartnik/ArduinoTutorials/blob/master/28BYJ-48/28BYJ-48.ino.

[18] [42] Random Nerd Tutorials. “Arduino with PIR Motion Sensor.” *Random Nerd Tutorials*, 2 Apr. 2019, randomnerdtutorials.com/arduino-with-pir-motion-sensor/.

[56] Raspberry Pi Tutorials. *Build a Digital Raspberry Pi Scale (with Weight Sensor HX711)*. Raspberry Pi Tutorials, 2020, tutorials-raspberrypi.com/digital-raspberry-pi-scale-weight-sensor-hx711/.

[45] Raspberry Pi Tutorials. *Using a Raspberry Pi Distance Sensor (Ultrasonic Sensor HC-SR04)*. Raspberry Pi Tutorials, 2020, tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/.

[41] Schow, Paul. “Monitoring Temperatures Using a Raspberry Pi and a MCP3208 ADC with Thermistors.” *Monitoring Temperatures Using a Raspberry Pi and a MCP3208 ADC with Thermistors*, 1 Jan. 1970, www.paulschow.com/2013/08/monitoring-temperatures-using-raspberry.html.

[8] South African Instrumentation and Control. “Principles of Infrared Temperature Measurement - Part 1.” *Principles of Infrared Temperature Measurement - Part 1 - August 2006 - R&C Instrumentation - SA Instrumentation & Control*, Technews Publishing, 2020, www.instrumentation.co.za/article.aspx?pkarticleid=4038.

[33] SparkFun. “Accelerometer Basics: What Is an Accelerometer?” *Accelerometer Basics*, 2020, learn.sparkfun.com/tutorials/accelerometer-basics/all.

[57] SparkFun. *ADXL345 Hookup Guide*. SparkFun, learn.sparkfun.com/tutorials/adxl345-hookup-guide/all.

[22] SparkFun. “Hobby Motor - Gear.” *ROB-11696 - SparkFun Electronics*, SparkFun, 2020, www.sparkfun.com/products/11696.

[54] [55] SparkFun. *Load Cell Amplifier HX711 Breakout Hookup Guide*. SparkFun, learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all#installing-the-hx711-arduino-library-and-examples.

[63] SparkFun. *Mini Load Cell - 500g, Straight Bar (TAL221)*. Sparkfun, 2020, <https://www.sparkfun.com/products/14728>

[25] SparkFun. “Servo - Generic (Sub-Micro Size).” *ROB-09065 - SparkFun Electronics*, SparkFun, 2020, www.sparkfun.com/products/9065.

- [53] SparkFun. "TAL221 Miniature Load Cell." *TAL221 – SparkFun Electronics*, Sparkfun, 2020, <https://cdn.sparkfun.com/assets/9/9/a/f/3/TAL221.pdf>
- [52] SparkFun. "Technical Data MQ-3 Gas Sensor." *MQ-3 – SparkFun Electronics*, Sparkfun, 2020, <https://www.sparkfun.com/datasheets/Sensors/MQ-3.pdf>.
- [64] Tatobari. "Tatobari/hx711py." *GitHub*, GitHub, 13 Aug. 2019, github.com/tatobari/hx711py.
- [14] TDK. "PTC Inrush Current Limiters." *How to Use PTC Thermistors as Current Protection | Tech Notes | PTC Inrush Current Limiters | TDK Product Center*, TDK Corporation, 2020, product.tdk.com/info/en/products/protection/current/ptc-limiter/technote/apn-ptc-thermistor.html.
- [7] TEP Technopreneurs. *What are Temperature Sensors*. The Engineering Projects, <https://www.theengineeringprojects.com/2019/09/what-are-temperature-sensors.html>
- [37] The Pi Hut. *Turning on an LED with Your Raspberry Pi's GPIO Pins*. The Pi Hut, 11 June 2015, thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins.
- [1] U.S. Department of Education. "Enrollment in Distance Education Courses by State." nces.edu, 2014, <https://nces.ed.gov/pubs2014/2014023.pdf>.
- [48] Youngblood, Tim. *Servo Motor Control with an Arduino - Projects*. All About Circuits, 3 June 2015, www.allaboutcircuits.com/projects/servo-motor-control-with-an-arduino/.
- [62] YouTube. "Stepper Motor Basics - 4 Wires Bipolar Motor". *Eastern Geek*, 24 May 2015, <https://www.youtube.com/watch?v=IEmGOuMFPKQ>

Appendices

Appendix 1: ME 3902 Pilot Class Bill of Materials

This appendix contains a series of order forms which the team used to record all purchased items used in the ME 3902 pilot class, serving as the overall bill of materials. The costs listed do not include shipping and other fees. The ME 3902 pilot class was run with 71 students, which the quantities below were generally sufficient to accommodate, with the exception of some bulk items such as screws and spacers which needed to be restocked. Extra units of these items were purchased during the pilot class as needed. This may or may not be necessary depending on the size of the class. Per box quantities can be modified if more or less of each item per student kit is desired.

The final cost of all items below was \$15,263.09 USD. The unit price was \$270.05 USD, corresponding to the cost of ordering 1 unit of each item listed. The per-box cost was \$215.58 USD, corresponding to the cost of the materials allocated to an individual student kit. However, the actual price students paid for the kits was approximately \$75 USD for the sensors, \$35 USD for the Arduino, and \$50 USD for the Raspberry Pi, which was subsidized in order to increase the course’s accessibility. The per-box cost originates from the quantities the team divided bulk items into. For example, the M3 threaded hex spacers from the Amazon order came in bulk packaging of 100 spacers per unit. 4 of these units were purchased, and 4 individual spacers were allocated to each student kit from this total pool of 400 spacers, allowing for 100 student kits to have the same number of spacers. This process was followed for all items, such that the maximum possible number of identical student kits could be assembled from the given quantities.

Co. Name: Adafruit	Name: Cameron Gould					
Street Address: 150 Varick St	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: New York, NY 10013						
	MQP Name: User-Driven Experimentation					
Phone #: (646) 248-7822	Account to Charge:					
Fax #:	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Per Box	Per Box	Total	Total
Model or Part #	Item Description:	Price	Quantity	Price	Quantity	Price
MCP3008	MCP3008 Analog to Digital Converter	\$3.75	1	\$3.75	100	\$375.00
TB6612	TB6612 DC/Stepper Motor Driver	\$4.95	1	\$4.95	100	\$495.00
1831	100 kOhm Potentiometer	\$0.95	1	\$0.95	100	\$95.00
4295	Raspberry Pi 4 Model B	\$35.00	1	\$35.00	50	\$1,750.00
Total:		\$44.65	4	\$44.65	350	\$2,715.00

Figure 55: ME 3902 Pilot Class Materials Purchased From Adafruit

Co. Name: Amazon	Name: Cameron Gould					
Street Address: 410 Terry Ave	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: N Seattle, WA						
	MQP Name: User-Driven Experimentation					
Phone #: 1 (888) 280-4331	Account to Charge:					
Fax #: 206-266-1838	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Per Box	Per Box	Total	Total
Model or Part #	Item Description:	Price	Quantity	Price	Quantity	Price
B012ZZ4LPM	HC-SR501 Infrared Motion Sensor	\$9.49	1	\$9.49	100	\$949.00
B01461P5V2	M3 Threaded Hex Spacers (100 pack)	\$5.99	4	\$0.24	4	\$23.96
B01HT871SO	MAX6675 Thermocouple	\$7.99	1	\$7.99	100	\$799.00
B07CDCPSST	Clear Rubber Feet (200 pack)	\$7.99	4	\$0.16	2	\$15.98
B07FM69GN9	Round Spacers (300 pack)	\$8.89	4	\$0.12	2	\$17.78
Total:		\$40.35	14	\$18.00	208	\$1,805.72

Figure 56: ME 3902 Pilot Class Materials Purchased From Amazon

Co. Name: Blick	Name: Cameron Gould					
Street Address: PO Box 1769	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: GALESBURG, IL						
	MQP Name: User-Driven Experimentation					
Phone #: (800) 828-4548	Account to Charge:					
Fax #: (800) 621-8293	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Total	Total		
Model or Part #	Item Description:	Price	Quantity	Price		
28945-1002	24 in x 24 in x 0.125 in Clear Acrylic Sheet	\$18.35	4	\$73.40		
Total:		\$18.35	4	\$73.40		

Figure 57: ME 3902 Pilot Class Materials Purchased From Blick

Co. Name: Home Depot	Name: Cameron Gould					
Street Address: 130 Gold Star Blvd	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: Worcester, MA						
	MQP Name: User-Driven Experimentation					
Phone #: 1 (508)852-6260	Account to Charge:					
Fax #: 1-877-496-9470	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Total	Total		
Model or Part #	Item Description:	Price	Quantity	Price		
1508104	0.25 in x 2 ft x 4 ft Medium Density Fiberboard	\$6.11	8	\$48.88		
Total:		\$6.11	8	\$48.88		

Figure 58: ME 3902 Pilot Class Materials Purchased From Home Depot

Co. Name: McMaster Carr	Name: Cameron Gould					
Street Address: 9630 Norwalk Blvd.	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: Santa Fe Springs, CA						
	MQP Name: User-Driven Experimentation					
Phone #: 1 (562) 692-6911	Account to Charge:					
Fax #: (562) 696-2323	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Per Box	Per Box	Total	Total
Model or Part #	Item Description:	Price	Quantity	Price	Quantity	Price
90592A006	M2.5 Steel Hex Nut (100 pack)	\$1.18	4	\$0.05	4	\$4.72
90592A085	M3 Steel Hex Nut (100 pack)	\$0.88	4	\$0.04	4	\$3.52
92000A132	M3 Stainless Steel Phillips Screw (100 pack)	\$7.77	4	\$0.31	4	\$31.08
92005A075	M2.5 Steel Phillips Screw (100 pack)	\$4.66	4	\$0.19	4	\$18.64
93475A196	M2.5 Stainless Steel Washers (100 pack)	\$1.58	4	\$0.06	4	\$6.32
93475A210	M3 Stainless Steel Washers (100 pack)	\$1.62	8	\$0.13	8	\$12.96
Total:		\$17.69	28	\$0.77	28	\$77.24

Figure 59: ME 3902 Pilot Class Materials Purchased From McMaster Carr

Co. Name: Sparkfun	Name: Cameron Gould					
Street Address: 6333 Dry Creek Parkway	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: Niwot, CO 80503						
	MQP Name: User-Driven Experimentation					
Phone #:303-284-0979	Account to Charge:					
Fax #:303-443-0048	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Per Box	Per Box	Total	Total
Model or Part #	Item Description:	Price	Quantity	Price	Quantity	Price
COM-10302	Tactile Button Assortment (4 pack)	\$1.00	1	\$0.25	25	\$25.00
COM-10969	Resistor Kit (500 pack)	\$7.95	2	\$0.03	8	\$63.60
COM-12062	LED - Assorted (20 pack)	\$3.30	2	\$0.33	10	\$33.00
PRT-11026	Male-to-Male Jumper Wires (30 pack)	\$2.25	15	\$1.13	50	\$112.50
PRT-12794	Male-to-Female Jumper Wires (20 pack)	\$1.95	10	\$0.98	50	\$97.50
PRT-12796	Female-to-Female Jumper Wires (20 pack)	\$1.95	5	\$0.49	25	\$48.75
Total:		\$18.40	35	\$3.20	168	\$380.35

Figure 60: ME 3902 Pilot Class Materials Purchased From Sparkfun (Bulk Supplies)

Co. Name: Sparkfun	Name: Cameron Gould					
Street Address: 6333 Dry Creek Parkway	Email Address: gr-experimentationmqp19-20@wpi.edu					
City, State: Niwot, CO 80503						
	MQP Name: User-Driven Experimentation					
Phone #:303-284-0979	Account to Charge:					
Fax #:303-443-0048	Advisor will provide if not mqp. Otherwise dept. will add					
	**all item descriptions are hyperlinked to the website	Unit	Per Box	Per Box	Total	Total
Model or Part #	Item Description:	Price	Quantity	Price	Quantity	Price
CAB-00512	USB Cable A to B for Arduino	\$3.95	1	\$3.95	50	\$197.50
COM-15107	microSD Card for Raspberry Pi	\$4.95	1	\$4.95	50	\$247.50
DEV-11021	Arduino Uno R3	\$22.95	1	\$22.95	50	\$1,147.50
PRT-12002	Breadboard	\$4.95	1	\$4.95	100	\$495.00
ROB-09065	Servo Motor	\$8.95	1	\$8.95	100	\$895.00
ROB-10551	Stepper Motor	\$7.95	1	\$7.95	100	\$795.00
ROB-11696	DC Motor	\$1.95	1	\$1.95	100	\$195.00
SEN-00250	NTCLE100E3103JB0 10K NTC Thermistor	\$0.75	1	\$0.75	100	\$75.00
SEN-08880	MQ-3 Alcohol Gas Sensor	\$4.95	1	\$4.95	100	\$495.00
SEN-09404	MQ-4 Concentrated Natural Gas Sensor	\$4.95	1	\$4.95	100	\$495.00
SEN-09836	ADXL345 Triple Axis Accelerometer Breakout	\$18.95	1	\$18.95	100	\$1,895.00
SEN-10988	TMP36 Silicon Band Gap Temperature Sensor	\$1.50	1	\$1.50	100	\$150.00
SEN-13879	HX711 Load Cell Amplifier	\$9.95	1	\$9.95	100	\$995.00
SEN-14728	TAL221 Mini Load Cell - 500g Straight Bar	\$9.95	1	\$9.95	100	\$995.00
SEN-15569	HC-SR04 Ultra Sonic Distance Sensor	\$3.95	1	\$3.95	100	\$395.00
TOL-13831	Wall Adapter Power Supply for Raspberry Pi	\$7.95	1	\$7.95	50	\$397.50
TOL-15312	Wall Adapter Power Supply for Arduino	\$5.95	1	\$5.95	50	\$297.50
Total:		\$124.50	17	\$124.50	1,450	\$10,162.50

Figure 61: ME 3902 Pilot Class Materials Purchased From Sparkfun (Sensors/Microcontrollers)

Appendix 2: ME 3902 Pilot Class Student Questions/Issues

This appendix summarizes all student questions and issues that were recorded by the team during lab sessions of the ME 3902 pilot class. Information is recorded for Modules 1-5.

Module Number	Question Asked
Module 1: The Microprocessor	Students are asking for extra screws.
Module 1: The Microprocessor	There is confusion about the types of washers and spacers needed to fix the microprocessor to the board because there are multiple types in the kit.
Module 1: The Microprocessor	How do you upload codes to the Arduino board?
Module 1: The Microprocessor	Do breadboard line numbers matter if we use different ones than the ones in the instructions?
Module 1: The Microprocessor	Which resistor should we use for the LED? We don't have the 1 kilohm resistor the lab says we need.
Module 1: The Microprocessor	How do you connect the Raspberry Pi to the WPI-WIRELESS network?
Module 1: The Microprocessor	Which resistor should be used with the LED?
Module 1: The Microprocessor	The setup is correct, but the system is not working. The student is using an online IDE. Looks like a software issue.
Module 1: The Microprocessor	What do I do next after plugging in the Raspberry Pi?
Module 1: The Microprocessor	Why does the Raspberry Pi give me "Unable to fetch some archives" errors when I run the code?
Module 1: The Microprocessor	Where do I access the information for this lab?
Module 1: The Microprocessor	My code needs some troubleshooting.
Module 1: The Microprocessor	The internet connection to my microcontroller is very poor.
Module 1: The Microprocessor	Do I have to use the given Arduino program to light up the LED setup?
Module 1: The Microprocessor	There is a problem with the given code, the LED would only turn on after debugging it.
Module 1: The Microprocessor	The LED is not working, appears to be a wiring setup issue.
Module 1: The Microprocessor	How do you get the button secure in the breadboard?
Module 1: The Microprocessor	What is "Ground" and why do we need it?
Module 1: The Microprocessor	Why is my LED not lighting up? (The polarity was incorrect)
Module 1: The Microprocessor	A student recommended that in the future, the class uses wires of varying lengths (especially male to male), and that it follows the ECE 2010 practices for student lab kits. In the beginning of the term, students would come into the lab, assemble their own kit, download the Arduino/Raspberry Pi software, and be checked off for having completed all setup tasks.
Module 1: The Microprocessor	The LED is not working. (Wiring issue)

Module 2: Temperature	The code for the silicon band gap experiment is not showing up when I access the Canvas page. (Issue with Canvas image storage, resolved)
Module 2: Temperature	Raspberry Pi OS not showing up on monitor
Module 2: Temperature	Where is the temperature displayed on the microcontroller?
Module 2: Temperature	I don't have a potentiometer in my kit.
Module 2: Temperature	Can the wires connecting to the ADC be plugged into the other side of the breadboard?
Module 2: Temperature	My thermistor temperature output is inaccurate.
Module 2: Temperature	Is my circuit correct?
Module 2: Temperature	The Arduino is recording 0 voltage out of the silicon band gap sensor. (Caused by a faulty breadboard)
Module 2: Temperature	How do I display the resistance value on the Arduino serial monitor?
Module 2: Temperature	Is there a way to record Arduino data into Excel?
Module 2: Temperature	My thermistor is reading values that are very different from the thermometer, but the code is properly configured.
Module 2: Temperature	Is there a way to run the silicon band gap sensor and thermistor at the same time?
Module 2: Temperature	Why is my thermistor reading a negative temperature value on the Arduino?
Module 2: Temperature	The Raspberry Pi thermistor example code seems to be wrong.
Module 2: Temperature	Issue with downloading Arduino code for thermistor and running it correctly.
Module 2: Temperature	How is the thermistor calibrated?
Module 2: Temperature	Arduino code outputs library folder errors for not having a .h extension on a supplementary file type.
Module 2: Temperature	The thermistor's sampling rate was extremely high even though the delay was set to 5 seconds. The thermistor is shorting out because of this. (Solution was to separate the leads)
Module 2: Temperature	How do you transfer files onto and within the Raspberry Pi? (Select the middle icon on the top of the screen in the viewer and follow provided instructions)
Module 2: Temperature	The Arduino won't upload code and kept failing to do so. (There was a bad cable in the student's kit being used, the student borrowed an extra one for the rest of class).
Module 2: Temperature	The Raspberry Pi code is not working with the SPI and busio.i2c command. (Student was told to switch code and download the mcp 3008 library)
Module 2: Temperature	The Raspberry Pi import busio function is not reading correctly. (Student told to import MCP 3008 code, which worked)
Module 2: Temperature	Not getting meaningful data from the silicon band gap sensor. (The wiring connection from the ADC to the Raspberry Pi was wrong)

Module 2: Temperature	Why is a dividing by zero error appearing in my Raspberry Pi code?
Module 3: Proximity	I can't get any data from the IR proximity sensor. (Wiring issue)
Module 3: Proximity	The ultrasonic sensor is outputting a distance of 0. (The code was wrong)
Module 3: Proximity	A student did not have an ultrasonic sensor in their kit.
Module 3: Proximity	What are we supposed to do for the ultrasonic experiment?
Module 3: Proximity	The line of best fit is not accurate. (Student used the wrong data points)
Module 3: Proximity	The ultrasonic sensor is not working. (Two sensors were being run at the same time next to each other, which was causing interference)
Module 4: Motors	Can I cut off extra pins that I don't need for soldering from the provided driver pin sets?
Module 4: Motors	The Raspberry Pi servo motor code seems to be correct, but the motor does not spin. (Some lines were not indented properly)
Module 4: Motors	Do you solder the short part of the wires to the board?
Module 4: Motors	How do you download and set up a Raspberry Pi library?
Module 4: Motors	My breadboard is shorting out. (Bad soldering on DC motor driver)
Module 4: Motors	The code number used for the Raspberry Pi GPIO pins don't match up with the picture. (Inconsistency in lab writeup, fixed)
Module 4: Motors	The servo code is not clear, I'm not sure what each line does. (Fixed)
Module 5: Concentrations	How do I properly wire the alcohol sensor?
Module 5: Concentrations	How do I know if my alcohol sensor is properly calibrated?

Figure 62: ME 3902 Pilot Class Student Questions/Issues

Appendix 3: ME 3902 Pilot Class Experiment Code

This appendix contains the plain text versions of code that was used in the ME 3902 pilot class experiments. This code has been formatted to allow for direct copying into the Arduino and Raspberry Pi interfaces. When copying this code into the Arduino or Raspberry Pi, edits may need to be made relating to indentation and spacing in order for some codes to work, or for greater ease of reading.

Module 2: Temperature

Thermocouple (Arduino)

```
#include <max6675.h>
```

```
int ktcSO = 8;
```

```
int ktcCS = 9;
```

```
int ktcCLK = 10;
```

```
MAX6675 ktc(ktcCLK, ktcCS, ktcSO);
```

```

void setup() {
  Serial.begin(9600);
  delay(500);

  pinMode(8, INPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop() {
  float Ctemp=ktc.readCelsius();
  float Ftemp=ktc.readCelsius()*9.0/5.0+32.0;
  Serial.print("Deg C = ");
  Serial.print(Ctemp);
  Serial.print("\t Deg F = ");
  Serial.println(Ftemp);
  delay(500);
}

```

Thermocouple (Raspberry Pi)

```

#define CLK 5
#define DBIT 6 // so
#define CS 7

#include <wiringPi.h>
#include <stdlib.h>
#include <stdio.h>

// declare function
int Thermal_Couple_Read(void);

int SENSOR_VALUE = 0;
float Ctemp, Ftemp;

int main() {
  if (wiringPiSetup () == -1)
    exit (1) ;

  pinMode(CLK, OUTPUT);
  pinMode(DBIT, INPUT);
  pinMode(CS, OUTPUT);

  digitalWrite(CS, HIGH);

```

```

digitalWrite(CLK, LOW);

SENSOR_VALUE = Thermal_Couple_Read();
if (SENSOR_VALUE == -1) {
    printf ("No sensor connected. \n");
}
else {
    printf ("S %d ", SENSOR_VALUE);
    Ctemp = SENSOR_VALUE * 0.25;
    printf ("C = %4.2f ", Ctemp);
    Ftemp = (Ctemp * 9 / 5) + 32;
    printf ("F = %4.2f", Ftemp);
}

return 0;
}

int Thermal_Couple_Read() {

    int value = 0;
    // init sensor
    digitalWrite(CS, LOW);
    delay(2);
    digitalWrite(CS, HIGH);
    delay(200);

    /* Read the chip and return the raw temperature value
    Bring CS pin low to allow us to read the data from
    the conversion process */

    digitalWrite(CS, LOW);
    /* Cycle the clock for dummy bit 15 */
    digitalWrite(CLK, HIGH);
    // delay(1);
    digitalWrite(CLK, LOW);

    /*
    Read bits 14-3 from MAX6675 for the Temp.
    Loop for each bit reading
    the value and storing the final value in 'temp' */

    int i;
    for (i = 14; i >= 0; i--) {
        digitalWrite(CLK, HIGH);

```

```

// delay(1);
value += digitalRead(DBIT) << i;
digitalWrite(CLK, LOW);
}

// check bit D2 if HIGH no sensor
if ((value & 0x04) == 0x04) return -1;
// shift right three places
return value >> 3;

}

```

Silicon Band Gap Sensor (Arduino)

```

//TMP36 Pin Variables
int sensorPin = 0; //the analog pin the TMP36's Vout (sense) pin is connected to
                    //the resolution is 10 mV / degree centigrade with a
                    //500 mV offset to allow for negative temperatures

/*
 * setup() - this function runs once when you turn your Arduino on
 * We initialize the serial connection with the computer
 */
void setup()
{
  Serial.begin(9600); //Start the serial connection with the computer
                    //to view the result open the serial monitor
}

void loop()          // run over and over again
{
  //getting the voltage reading from the temperature sensor
  int reading = analogRead(sensorPin);

  // converting that reading to voltage, for 3.3v arduino use 3.3
  float voltage = reading * 5.0;
  voltage /= 1024.0;

  // print out the voltage
  Serial.print(voltage); Serial.println(" volts");

  // now print out the temperature
  float temperatureC = (voltage - 0.5) * 100 ; //converting from 10 mv per degree with 500 mV offset
                    //to degrees ((voltage - 500mV) times 100)
  Serial.print(temperatureC); Serial.println(" degrees C");
}

```

```

// now convert to Fahrenheit
float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
Serial.print(temperatureF); Serial.println(" degrees F");

delay(1000);           //waiting a second
}

```

Silicon Band Gap Sensor (Raspberry Pi)

```

from gpiozero import MCP3008
from time import sleep

def convert_temp(gen):
    for value in gen:
        yield (value * 3.3 - 0.5) * 100

adc = MCP3008(channel=0)

for temp in convert_temp(adc.values):
    print('The temperature is', temp, 'C')
    sleep(1)

```

Thermistor (Arduino)

```

int ThermistorPin = 0;
int Vo;
float R1 = 10000;
float logR2, R2, T;
float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-07;

void setup() {
  Serial.begin(9600);
}

void loop() {

  Vo = analogRead(ThermistorPin);
  R2 = R1 * (1023.0 / (float)Vo - 1.0);
  logR2 = log(R2);
  T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2));
  T = T - 273.15;
  T = (T * 9.0) / 5.0 + 32.0;

  Serial.print("Temperature: ");
  Serial.print(T);
}

```

```

Serial.println(" F");

delay(500);
}

```

Thermistor (Raspberry Pi)

```

#thermistor reading function
def temp_get(adc):
    value = readadc(adc) #read the adc
    volts = (value * 3.3) / 1024 #calculate the voltage
    ohms = ((1/volts)*3300)-1000 #calculate the ohms of the thermistor

    lnohm = math.log1p(ohms) #take ln(ohms)

    #a, b, & c values from http://www.thermistor.com/calculators.php
    #using curve R (-6.2%/C @ 25C) Mil Ratio X
    a = 0.002197222470870
    b = 0.000161097632222
    c = 0.000000125008328

    #Steinhart Hart Equation
    # T = 1/(a + b[ln(ohm)] + c[ln(ohm)]^3)

    t1 = (b*lnohm) # b[ln(ohm)]

    c2 = c*lnohm # c[ln(ohm)]

    t2 = math.pow(c2,3) # c[ln(ohm)]^3

    temp = 1/(a + t1 + t2) #calculate temperature

    tempc = temp - 273.15 - 4 #K to C
    # the -4 is error correction for bad python math

    #print out info
    print ("%4d/1023 => %5.3f V => %4.1f Ω => %4.1f °K => %4.1f °C from adc %d" % (value, volts,
    ohms, temp, tempc, adc))
    return tempc

```

Module 3: Proximity

IR Motion Sensor (Arduino)

```

int led = 13;           // the pin that the LED is attached to
int sensor = 2;        // the pin that the sensor is attached to
int state = LOW;       // by default, no motion detected

```



```

int val = 0;          // variable to store the sensor status (value)

void setup() {
  pinMode(led, OUTPUT);  // initialize LED as an output
  pinMode(sensor, INPUT); // initialize sensor as an input
  Serial.begin(9600);    // initialize serial
}

void loop(){
  val = digitalRead(sensor); // read sensor value
  if (val == HIGH) {        // check if the sensor is HIGH
    digitalWrite(led, HIGH); // turn LED ON
    delay(100);             // delay 100 milliseconds

    if (state == LOW) {
      Serial.println("Motion detected!");
      state = HIGH;        // update variable state to HIGH
    }
  }
  else {
    digitalWrite(led, LOW); // turn LED OFF
    delay(200);             // delay 200 milliseconds

    if (state == HIGH){
      Serial.println("Motion stopped!");
      state = LOW;         // update variable state to LOW
    }
  }
}

```

IR Motion Sensor (Raspberry Pi)

```

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD) #Set GPIO to pin numbering
pir = 8 #Assign pin 8 to PIR
led = 10 #Assign pin 10 to LED
GPIO.setup(pir, GPIO.IN) #Setup GPIO pin PIR as input
GPIO.setup(led, GPIO.OUT) #Setup GPIO pin for LED as output
print ("Sensor initializing . . .")
time.sleep(2) #Give sensor time to startup
print ("Active")
print ("Press Ctrl+c to end program")

```

```

try:
while True:
if GPIO.input(pir) == True: #If PIR pin goes high, motion is detected
print ("Motion Detected!")
GPIO.output(led, True) #Turn on LED
time.sleep(4) #Keep LED on for 4 seconds
GPIO.output(led, False) #Turn off LED
time.sleep(0.1)

except KeyboardInterrupt: #Ctrl+c
pass #Do nothing, continue to finally

finally:
GPIO.output(led, False) #Turn off LED in case left on
GPIO.cleanup() #reset all GPIO
print ("Program ended")

```

Ultrasonic Distance Sensor (Arduino)

```

// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;
// defines variables
long duration;
int distance;
void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}
void loop() {
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculating the distance
distance= duration*0.034/2;
// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);

```

```
}
```

Ultrasonic Distance Sensor (Raspberry Pi)

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
```

```

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped by User")
        GPIO.cleanup()

```

Module 4: Motors

DC Motor (Arduino)

//motor A connected between A01 and A02

//motor B connected between B01 and B02

```
int STBY = 10; //standby
```

```
//Motor A
```

```
int PWMA = 3; //Speed control
```

```
int AIN1 = 9; //Direction
```

```
int AIN2 = 8; //Direction
```

```
//Motor B
```

```
int PWMB = 5; //Speed control
```

```
int BIN1 = 11; //Direction
```

```
int BIN2 = 12; //Direction
```

```
void setup(){
```

```
    pinMode(STBY, OUTPUT);
```

```
    pinMode(PWMA, OUTPUT);
```

```
    pinMode(AIN1, OUTPUT);
```

```
    pinMode(AIN2, OUTPUT);
```

```
    pinMode(PWMB, OUTPUT);
```

```
    pinMode(BIN1, OUTPUT);
```

```
    pinMode(BIN2, OUTPUT);
```

```
}
```

```
void loop(){
```

```
    move(1, 255, 1); //motor 1, full speed, left
```

```
move(2, 255, 1); //motor 2, full speed, left

delay(1000); //go for 1 second
stop(); //stop
delay(250); //hold for 250ms until move again

move(1, 128, 0); //motor 1, half speed, right
move(2, 128, 0); //motor 2, half speed, right

delay(1000);
stop();
delay(250);
}
```

```
void move(int motor, int speed, int direction){
//Move specific motor at speed and direction
//motor: 0 for B 1 for A
//speed: 0 is off, and 255 is full speed
//direction: 0 clockwise, 1 counter-clockwise
```

```
digitalWrite(STBY, HIGH); //disable standby
```

```
boolean inPin1 = LOW;
boolean inPin2 = HIGH;
```

```
if(direction == 1){
  inPin1 = HIGH;
  inPin2 = LOW;
}
```

```
if(motor == 1){
  digitalWrite(AIN1, inPin1);
  digitalWrite(AIN2, inPin2);
  analogWrite(PWMA, speed);
}else{
  digitalWrite(BIN1, inPin1);
  digitalWrite(BIN2, inPin2);
  analogWrite(PWMB, speed);
}
}
```

```
void stop(){
//enable standby
```

```
digitalWrite(STBY, LOW);  
}
```

DC Motor (Raspberry Pi)

```
# Import required modules  
import time  
import RPi.GPIO as GPIO  
  
# Declare the GPIO settings  
GPIO.setmode(GPIO.BOARD)  
  
# set up GPIO pins  
GPIO.setup(7, GPIO.OUT) # Connected to PWMA  
GPIO.setup(11, GPIO.OUT) # Connected to AIN2  
GPIO.setup(12, GPIO.OUT) # Connected to AIN1  
GPIO.setup(13, GPIO.OUT) # Connected to STBY  
  
# Drive the motor clockwise  
GPIO.output(12, GPIO.HIGH) # Set AIN1  
GPIO.output(11, GPIO.LOW) # Set AIN2  
  
# Set the motor speed  
GPIO.output(7, GPIO.HIGH) # Set PWMA  
  
# Disable STBY (standby)  
GPIO.output(13, GPIO.HIGH)  
  
# Wait 5 seconds  
time.sleep(5)  
  
# Drive the motor counterclockwise  
GPIO.output(12, GPIO.LOW) # Set AIN1  
GPIO.output(11, GPIO.HIGH) # Set AIN2  
  
# Set the motor speed  
GPIO.output(7, GPIO.HIGH) # Set PWMA  
  
# Disable STBY (standby)  
GPIO.output(13, GPIO.HIGH)  
  
# Wait 5 seconds  
time.sleep(5)  
  
# Reset all the GPIO pins by setting them to LOW
```

```
GPIO.output(12, GPIO.LOW) # Set AIN1
GPIO.output(11, GPIO.LOW) # Set AIN2
GPIO.output(7, GPIO.LOW) # Set PWMA
GPIO.output(13, GPIO.LOW) # Set STBY
```

Servo Motor (Arduino)

```
#include <Servo.h>          //Servo library

Servo servo_test;          //initialize a servo object for the connected servo

int angle = 0;

void setup()
{
  servo_test.attach(9);      // attach the signal pin of servo to pin9 of arduino
}

void loop()
{
  for(angle = 0; angle < 180; angle += 1)      // command to move from 0 degrees to 180 degrees
  {
    servo_test.write(angle);      //command to rotate the servo to the specified angle
    delay(15);
  }

  delay(1000);

  for(angle = 180; angle >= 1; angle -= 5)    // command to move from 180 degrees to 0 degrees
  {
    servo_test.write(angle);      //command to rotate the servo to the specified angle
    delay(5);
  }

  delay(1000);
}
```

Servo Motor (Raspberry Pi)

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BOARD)
GPIO.setup(03, GPIO.OUT)
pwm=GPIO.PWM(03, 50)
pwm.start(0)
def SetAngle(angle):
```

```
    duty = angle / 18 + 2
    GPIO.output(03, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(03, False)
    pwm.ChangeDutyCycle(0)
SetAngle(90)
pwm.stop()
GPIO.cleanup()
```

Stepper Motor (Arduino)

```
#define STEPPER_PIN_1 9
#define STEPPER_PIN_2 10
#define STEPPER_PIN_3 11
#define STEPPER_PIN_4 12
int step_number = 0;
void setup() {
  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);

}

void loop() {

  OneStep(false);
  delay(2);

}

void OneStep(bool dir){
  if(dir){
switch(step_number){
  case 0:
    digitalWrite(STEPPER_PIN_1, HIGH);
    digitalWrite(STEPPER_PIN_2, LOW);
    digitalWrite(STEPPER_PIN_3, LOW);
    digitalWrite(STEPPER_PIN_4, LOW);
    break;
  case 1:
    digitalWrite(STEPPER_PIN_1, LOW);
```



```
digitalWrite(STEPPER_PIN_2, HIGH);
digitalWrite(STEPPER_PIN_3, LOW);
digitalWrite(STEPPER_PIN_4, LOW);
break;
case 2:
digitalWrite(STEPPER_PIN_1, LOW);
digitalWrite(STEPPER_PIN_2, LOW);
digitalWrite(STEPPER_PIN_3, HIGH);
digitalWrite(STEPPER_PIN_4, LOW);
break;
case 3:
digitalWrite(STEPPER_PIN_1, LOW);
digitalWrite(STEPPER_PIN_2, LOW);
digitalWrite(STEPPER_PIN_3, LOW);
digitalWrite(STEPPER_PIN_4, HIGH);
break;
}
}else{
  switch(step_number){
case 0:
digitalWrite(STEPPER_PIN_1, LOW);
digitalWrite(STEPPER_PIN_2, LOW);
digitalWrite(STEPPER_PIN_3, LOW);
digitalWrite(STEPPER_PIN_4, HIGH);
break;
case 1:
digitalWrite(STEPPER_PIN_1, LOW);
digitalWrite(STEPPER_PIN_2, LOW);
digitalWrite(STEPPER_PIN_3, HIGH);
digitalWrite(STEPPER_PIN_4, LOW);
break;
case 2:
digitalWrite(STEPPER_PIN_1, LOW);
digitalWrite(STEPPER_PIN_2, HIGH);
digitalWrite(STEPPER_PIN_3, LOW);
digitalWrite(STEPPER_PIN_4, LOW);
break;
case 3:
digitalWrite(STEPPER_PIN_1, HIGH);
digitalWrite(STEPPER_PIN_2, LOW);
digitalWrite(STEPPER_PIN_3, LOW);
digitalWrite(STEPPER_PIN_4, LOW);
```

```

}
}
step_number++;
if(step_number > 3){
    step_number = 0;
}
}

```

Stepper Motor (Raspberry Pi)

```

# Import required libraries
import sys
import time
import RPi.GPIO as GPIO

# Use BCM GPIO references
# instead of physical pin numbers
GPIO.setmode(GPIO.BCM)

# Define GPIO signals to use
# Physical pins 11,15,16,18
# GPIO17,GPIO22,GPIO23,GPIO24
StepPins = [17,22,23,24]

# Set all pins as output
for pin in StepPins:
    print "Setup pins"
    GPIO.setup(pin,GPIO.OUT)
    GPIO.output(pin, False)

# Define advanced sequence
# as shown in manufacturers datasheet
Seq = [[1,0,0,1],
        [1,0,0,0],
        [1,1,0,0],
        [0,1,0,0],
        [0,1,1,0],
        [0,0,1,0],
        [0,0,1,1],
        [0,0,0,1]]

StepCount = len(Seq)
StepDir = 1 # Set to 1 or 2 for clockwise
            # Set to -1 or -2 for anti-clockwise

```

```

# Read wait time from command line
if len(sys.argv)>1:
    WaitTime = int(sys.argv[1])/float(1000)
else:
    WaitTime = 10/float(1000)

# Initialise variables
StepCounter = 0

# Start main loop
while True:

    print StepCounter,
    print Seq[StepCounter]

    for pin in range(0, 4):
        xpin = StepPins[pin]
        if Seq[StepCounter][pin]!=0:
            print " Enable GPIO %i" %(xpin)
            GPIO.output(xpin, True)
        else:
            GPIO.output(xpin, False)

    StepCounter += StepDir

# If we reach the end of the sequence
# start again
if (StepCounter>=StepCount):
    StepCounter = 0
if (StepCounter<0):
    StepCounter = StepCount+StepDir

# Wait before moving on
time.sleep(WaitTime)

```

Module 5: Concentrations

Gas Sensor (Arduino)

Set up:

```

//Set R2 to 20000 ohms
int R2 = 10000;
// Set Vo to 5 Volts
float Vo = 5;
// Declare all variables
float Vd,Va,Ro;

```

```

// The setup routine runs once when you press reset
void setup() {
  // Initialize serial communication at 9600 bits per second
  Serial.begin(9600);
}

// The loop routine runs over and over again forever
void loop() {
  // Read the input on analog pin 0
  Vd = analogRead(0);
  // Change the digital input 0-1023 into a voltage between 0V-5V
  Va = Vd*(Vo/1023);
  // Take the known variables (Va,Vo, and R2) and
  // using them to calculate the unknown R of the sensor
  Ro = ((Vo*R2)/Va)-R2;
  // Print various values to make sure everything looks correct
  // and troubleshoot your code
  Serial.print(Vd);
  Serial.print(",");
  Serial.print(Va);
  Serial.print(",");
  Serial.println(Ro);
  delay(1500);
}

```

Running the sensor:

```

//Set R2 to 20,000 Ohms
int R2 = 10000;
//Set Ro to the value from the previous code
float Ro = 110000;
// Set Vo to 5 Volts
float Vo = 5;
// Declare all variables
float Vd, Va, Rs, x, mgL, mgLbreath, BAC;
// The setup routine runs once when you press reset
void setup() {
  // Initialize serial communication at 9600 bits per second
  Serial.begin(9600);
}
// The loop routine runs over and over again forever
void loop() {
  // Read the input on analog pin 0
  Vd = analogRead(0);

```

```

// Change the digital input 0-1023 into a voltage between 0V-5V
Va = Vd*(Vo/1023);
// Take the known variables (Va,Vo, and R2) and
// using them to calculate the unknown R of the sensor
Rs = ((Vo*R2)/Va)-R2;
// Use the Rs we have just calculated and the known
//starting resistance of the sensor
x = Rs/Ro;
// Use the relationship given in the data sheet to get
// the mg/L at the given Rs/Ro value
mgL = .356*pow(x,-1.62);
// Calibrate the sensors to make sure that when the sensor
// is sitting it is reading 0 mg/L breath even though there
// is around .4 mg/l in the air according to the data sheet
mgLbreath = mgL-.310;
// Using the link given in the lab convert mg/L to BAC
BAC = mgLbreath/50;
// Print various values to make sure everything looks correct
// and troubleshoot your code
Serial.print(x);
Serial.print(" Rs/Ro,");
Serial.print(mgLbreath);
Serial.print(" mg/L,");
Serial.print(BAC);
Serial.println(" BAC");
delay(1500);
}

```

Gas Sensor (Raspberry Pi)

Set up:

```

from time import sleep
#Importing a delay function
from gpiozero import MCP3008
#Importing a library built for the MCP3008
#This means that you won't have to deal with the spi bus stuff

```

```

def Din_Resistance(gen) :
#Make a function to convert between your ADC value
#and the resistance of the sensor
    for value in gen :
#For every value that we put into this function
        yield(((5*20000)/(value*5))-20000)
#Put it into this equation and output the result
#This sets up the equation Ro=((Vo*R2)/(Voutput))- R2

```

```

# Vo is the Voltage being hooked up to the circuit
# R2 is the known voltage of the other resistor (20 kOhm)
# Vout is the voltage from the ADC
# The ADC puts out a number from 0 to 1 based on how close it
# is to the maximum so by multiplying it by 5 volts we get our voltage
Din = MCP3008(channel=0)
# Setting up a variable to be equal to the output from the ADC

```

```

for Ro in Din_Resistance(Din.values):
#Define your variable that you are putting into the function
#So now all of the outputs from our function are called Ro
    print(Ro)
#Print the output
    sleep (1)
#Sleep for a second to let the computer catch up

```

Running the sensor

```

from time import sleep
#Importing a delay function
import math
#We will need this library to use exponents
from gpiozero import MCP3008
#Importing a library built for the MCP3008
#This means that you won't have to deal with the spi bus stuff

```

```

def Din_Resistance(gen) :
#Make a function to convert between your ADC value
#and the resistance of the sensor
    for value in gen :
#For every value that we put into this function
        yield ((.365*(((5*20000)/(value*5))-20000)/35000)**(-1.62))-4)/5
#Put it into this equation and output the result
#This sets up the equation BAC=((.365*(Rs/Ro)^-1.62)-Calibration)/5
#Rs is calculated the same we calculated Row in the previous code
#Ro is the Resistance that we calculated in the previous code (25 kOhm)
#The .365 and -1.62 are from the Rs/Ro equation that you calculated earlier
#The calibration is to take into account the varying alcohol levels in the room
# this should be set so that when you blow across the sensor you read 0 BAC
#The reason we divide by 5 is to convert between mg/L in breath to BAC
#the link to where I got this number will be in the lab description
Din = MCP3008(channel=0)
# Setting up a variable to be equal to the output from the ADC

```

```
for BAC in Din_Resistance(Din.values):
#Define your variable that you are putting into the function
#So now all of the outputs from our function are called Ro
    print("%.2f" % BAC)
#Print the output
    sleep (1)
#Sleep for a second to let the computer catch up
```

Module 6: Strain

Strain Gage (Arduino)

Calibration

```
/*
```

This is the calibration sketch. Use it to determine the calibration_factor that the main example uses. It also outputs the zero_factor useful for projects that have a permanent mass on the scale in between power cycles.

Setup your scale and start the sketch WITHOUT a weight on the scale
Once readings are displayed place the weight on the scale
Press +/- or a/z to adjust the calibration_factor until the output readings match the known weight
Use this calibration_factor on the example sketch

This example assumes pounds (lbs). If you prefer kilograms, change the Serial.print(" lbs"); line to kg. The calibration factor will be significantly different but it will be linearly related to lbs (1 lbs = 0.453592 kg).

Your calibration factor may be very positive or very negative. It all depends on the setup of your scale system

and the direction the sensors deflect from zero state

This example code uses bogde's excellent library: <https://github.com/bogde/HX711>

bogde's library is released under a GNU GENERAL PUBLIC LICENSE

Arduino pin 2 -> HX711 CLK

3 -> DOUT

5V -> VCC

GND -> GND

Most any pin on the Arduino Uno will be compatible with DOUT/CLK.

The HX711 board can be powered from 2.7V to 5V so the Arduino 5V power should be fine.

```
*/
```

```
#include "HX711.h"
```

```

#define DOUT 3
#define CLK 2

HX711 scale;

float calibration_factor = -7050; //-7050 worked for my 440lb max scale setup

void setup() {
  Serial.begin(9600);
  Serial.println("HX711 calibration sketch");
  Serial.println("Remove all weight from scale");
  Serial.println("After readings begin, place known weight on scale");
  Serial.println("Press + or a to increase calibration factor");
  Serial.println("Press - or z to decrease calibration factor");

  scale.begin(DOUT, CLK);
  scale.set_scale();
  scale.tare(); //Reset the scale to 0

  long zero_factor = scale.read_average(); //Get a baseline reading
  Serial.print("Zero factor: "); //This can be used to remove the need to tare the scale. Useful in permanent
scale projects.
  Serial.println(zero_factor);
}

void loop() {

  scale.set_scale(calibration_factor); //Adjust to this calibration factor

  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1);
  Serial.print(" lbs"); //Change this to kg and re-adjust the calibration factor if you follow SI units like a
sane person
  Serial.print(" calibration_factor: ");
  Serial.print(calibration_factor);
  Serial.println();

  if(Serial.available())
  {
    char temp = Serial.read();
    if(temp == '+' || temp == 'a')
      calibration_factor += 10;
    else if(temp == '-' || temp == 'z')
      calibration_factor -= 10;
  }
}

```



```
}  
}
```

Operation

```
#include "HX711.h"
```

```
#define calibration_factor -7050.0 //This value is obtained using the SparkFun_HX711_Calibration  
sketch
```

```
#define DOUT 3
```

```
#define CLK 2
```

```
HX711 scale;
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  Serial.println("HX711 scale demo");
```

```
  scale.begin(DOUT, CLK);
```

```
  scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_HX711_Calibration  
sketch
```

```
  scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0
```

```
  Serial.println("Readings:");
```

```
}
```

```
void loop() {
```

```
  Serial.print("Reading: ");
```

```
  Serial.print(scale.get_units(), 1); //scale.get_units() returns a float
```

```
  Serial.print(" lbs"); //You can change this to kg but you'll need to refactor the calibration_factor
```

```
  Serial.println();
```

```
}
```

Strain Gage (Raspberry Pi)

Note: First clone <https://github.com/tatobari/hx711py> [64]. The given code below is example.py

```
#!/usr/bin/python2
```

```
import time
```

```
import sys
```

```
EMULATE_HX711=False
```

```
referenceUnit = 1
```

```

if not EMULATE_HX711:
    import RPi.GPIO as GPIO
    from hx711 import HX711
else:
    from emulated_hx711 import HX711

def cleanAndExit():
    print("Cleaning...")

    if not EMULATE_HX711:
        GPIO.cleanup()

    print("Bye!")
    sys.exit()

hx = HX711(5, 6)

# I've found out that, for some reason, the order of the bytes is not always the same between versions of
python, numpy and the hx711 itself.
# Still need to figure out why it changes.
# If you're experiencing super random values, change these values to MSB or LSB until to get more stable
values.
# There is some code below to debug and log the order of the bits and the bytes.
# The first parameter is the order in which the bytes are used to build the "long" value.
# The second parameter is the order of the bits inside each byte.
# According to the HX711 Datasheet, the second parameter is MSB so you shouldn't need to modify it.
hx.set_reading_format("MSB", "MSB")

# HOW TO CALCULATE THE REFERENCE UNIT
# To set the reference unit to 1. Put 1kg on your sensor or anything you have and know exactly how much
it weighs.
# In this case, 92 is 1 gram because, with 1 as a reference unit I got numbers near 0 without any weight
# and I got numbers around 184000 when I added 2kg. So, according to the rule of thirds:
# If 2000 grams is 184000 then 1000 grams is 184000 / 2000 = 92.
#hx.set_reference_unit(113)
hx.set_reference_unit(referenceUnit)

hx.reset()

hx.tare()

print("Tare done! Add weight now...")

# to use both channels, you'll need to tare them both

```

```

#hx.tare_A()
#hx.tare_B()

while True:
    try:
        # These three lines are useful to debug whether to use MSB or LSB in the reading formats
        # for the first parameter of "hx.set_reading_format("LSB", "MSB)".
        # Comment the two lines "val = hx.get_weight(5)" and "print val" and uncomment these three lines
        # to see what it prints.

        # np_arr8_string = hx.get_np_arr8_string()
        # binary_string = hx.get_binary_string()
        # print binary_string + " " + np_arr8_string

        # Prints the weight. Comment if you're debugging the MSB and LSB issue.
        val = hx.get_weight(5)
        print(val)

        # To get weight from both channels (if you have load cells hooked up
        # to both channel A and B), do something like this
        #val_A = hx.get_weight_A(5)
        #val_B = hx.get_weight_B(5)
        #print "A: %s B: %s" % ( val_A, val_B )

        hx.power_down()
        hx.power_up()
        time.sleep(0.1)

    except (KeyboardInterrupt, SystemExit):
        cleanAndExit()

```

Module 7: Motion

Accelerometer (Arduino)

Main Code

```

/* *****
* SparkFun_ADXL345_Example
* Triple Axis Accelerometer Breakout - ADXL345
* Hook Up Guide Example
*
* Utilizing Sparkfun's ADXL345 Library
* Bildr ADXL345 source file modified to support
* both I2C and SPI Communication
*
* E.Robert @ SparkFun Electronics

```

```

* Created: Jul 13, 2016
* Updated: Sep 06, 2016
*
* Development Environment Specifics:
* Arduino 1.6.11
*
* Hardware Specifications:
* SparkFun ADXL345
* Arduino Uno
* *****/
#include <SparkFun_ADXL345.h>    // SparkFun ADXL345 Library
/***** COMMUNICATION SELECTION *****/
/* Comment Out The One You Are Not Using */
ADXL345 adxl = ADXL345(10);    // USE FOR SPI COMMUNICATION, ADXL345(CS_PIN);
//ADXL345 adxl = ADXL345();    // USE FOR I2C COMMUNICATION
/***** INTERRUPT *****/
/* Uncomment If Attaching Interrupt */
//int interruptPin = 2;        // Setup pin 2 to be the interrupt pin (for most Arduino Boards)
/***** SETUP *****/
/* Configure ADXL345 Settings */
void setup(){

  Serial.begin(9600);          // Start the serial terminal
  Serial.println("SparkFun ADXL345 Accelerometer Hook Up Guide Example");
  Serial.println();

  adxl.powerOn();              // Power on the ADXL345
  adxl.setRangeSetting(16);    // Give the range settings
                                // Accepted values are 2g, 4g, 8g or 16g
                                // Higher Values = Wider Measurement Range
                                // Lower Values = Greater Sensitivity
  adxl.setSpiBit(0);           // Configure the device to be in 4 wire SPI mode when set to '0' or 3 wire
  SPI mode when set to 1
                                // Default: Set to 1
                                // SPI pins on the ATmega328: 11, 12 and 13 as reference in SPI Library

  adxl.setActivityXYZ(1, 0, 0); // Set to activate movement detection in the axes
  "adxl.setActivityXYZ(X, Y, Z);" (1 == ON, 0 == OFF)
  adxl.setActivityThreshold(75); // 62.5mg per increment // Set activity // Inactivity thresholds (0-
  255)

  adxl.setInactivityXYZ(1, 0, 0); // Set to detect inactivity in all the axes "adxl.setInactivityXYZ(X, Y,
  Z);" (1 == ON, 0 == OFF)

```

```

    adxl.setInactivityThreshold(75); // 62.5mg per increment // Set inactivity // Inactivity thresholds (0-
255)
    adxl.setTimeInactivity(10); // How many seconds of no activity is inactive?
    adxl.setTapDetectionOnXYZ(0, 0, 1); // Detect taps in the directions turned ON
"adxl.setTapDetectionOnX(X, Y, Z);" (1 == ON, 0 == OFF)

// Set values for what is considered a TAP and what is a DOUBLE TAP (0-255)
    adxl.setTapThreshold(50); // 62.5 mg per increment
    adxl.setTapDuration(15); // 625 µs per increment
    adxl.setDoubleTapLatency(80); // 1.25 ms per increment
    adxl.setDoubleTapWindow(200); // 1.25 ms per increment

// Set values for what is considered FREE FALL (0-255)
    adxl.setFreeFallThreshold(7); // (5 - 9) recommended - 62.5mg per increment
    adxl.setFreeFallDuration(30); // (20 - 70) recommended - 5ms per increment

// Setting all interrupts to take place on INT1 pin
//adxl.setImportantInterruptMapping(1, 1, 1, 1, 1); // Sets "adxl.setEveryInterruptMapping(single tap,
double tap, free fall, activity, inactivity);"
// Accepts only 1 or 2 values for pins INT1 and INT2. This chooses the
pin on the ADXL345 to use for Interrupts.
// This library may have a problem using the INT2 pin. Default to
INT1 pin.

// Turn on Interrupts for each mode (1 == ON, 0 == OFF)
    adxl.InactivityINT(1);
    adxl.ActivityINT(1);
    adxl.FreeFallINT(1);
    adxl.doubleTapINT(1);
    adxl.singleTapINT(1);

//attachInterrupt(digitalPinToInterrupt(interruptPin), ADXL_ISR, RISING); // Attach Interrupt
}
/***** MAIN CODE *****/
/* Accelerometer Readings and Interrupt */
void loop(){

// Accelerometer Readings
    int x,y,z;
    adxl.readAccel(&x, &y, &z); // Read the accelerometer values and store them in variables declared
above x,y,z
// Output Results to Serial
/* UNCOMMENT TO VIEW X Y Z ACCELEROMETER VALUES */
//Serial.print(x);

```

```

//Serial.print(" ");
//Serial.print(y);
//Serial.print(" ");
//Serial.println(z);

ADXL_ISR();
// You may also choose to avoid using interrupts and simply run the functions within ADXL_ISR();
// and place it within the loop instead.
// This may come in handy when it doesn't matter when the action occurs.
}
/***** ISR *****/
/* Look for Interrupts and Triggered Action */
void ADXL_ISR() {

// getInterruptSource clears all triggered actions after returning value
// Do not call again until you need to recheck for triggered actions
byte interrupts = adxl.getInterruptSource();

// Free Fall Detection
if(adxl.triggered(interrupts, ADXL345_FREE_FALL)){
  Serial.println("*** FREE FALL ***");
  //add code here to do when free fall is sensed
}

// Inactivity
if(adxl.triggered(interrupts, ADXL345_INACTIVITY)){
  Serial.println("*** INACTIVITY ***");
  //add code here to do when inactivity is sensed
}

// Activity
if(adxl.triggered(interrupts, ADXL345_ACTIVITY)){
  Serial.println("*** ACTIVITY ***");
  //add code here to do when activity is sensed
}

// Double Tap Detection
if(adxl.triggered(interrupts, ADXL345_DOUBLE_TAP)){
  Serial.println("*** DOUBLE TAP ***");
  //add code here to do when a 2X tap is sensed
}

// Tap Detection
if(adxl.triggered(interrupts, ADXL345_SINGLE_TAP)){

```

```

Serial.println("*** TAP ***");
//add code here to do when a tap is sensed
}
}

```

Calibration

```

#include <SparkFun_ADXL345.h>
/***** COMMUNICATION SELECTION *****/
/* Comment Out The One You Are Not Using */
//ADXL345 adxl = ADXL345(10); // Use when you want to use Hardware SPI,
ADXL345(CS_PIN);
ADXL345 adxl = ADXL345(); // Use when you need I2C
/***** VARIABLES *****/
/*
int AccelMinX = 0;
int AccelMaxX = 0;
int AccelMinY = 0;
int AccelMaxY = 0;
int AccelMinZ = 0;
int AccelMaxZ = 0;
int accX = 0;
int accY = 0;
int accZ = 0;
int pitch = 0;
int roll = 0;
/***** DEFINED VARIABLES *****/
/*
#define offsetX 0 // OFFSET values
#define offsetY 0
#define offsetZ 0
#define gainX 1 // GAIN factors
#define gainY 1
#define gainZ 1
/***** SETUP *****/
/* Configure ADXL345 Settings */
void setup()
{
Serial.begin(9600); // Start the serial terminal
Serial.println("SparkFun ADXL345 Accelerometer Breakout Calibration");
Serial.println();
adxl.powerOn(); // Power on the ADXL345
adxl.setRangeSetting(2); // Give the range settings
// Accepted values are 2g, 4g, 8g or 16g
// Higher Values = Wider Measurement Range

```

```

// Lower Values = Greater Sensitivity
adxl.setSpiBit(0); // Configure the device to be in 4 wire SPI mode when set to '0' or 3 wire SPI
mode when set to 1
// It is set to 1 by Default.
// SPI pins on the ATmega328 as reference in SPI Library are 11, 12, and 13
}
/***** MAIN CODE *****/
/* Accelerometer Readings and Min/Max Values */
void loop()
{
Serial.println("Send any character to display values.");
while (!Serial.available()){ // Waiting for character to be sent to Serial
Serial.println();
// Get the Accelerometer Readings
int x,y,z; // init variables hold results
adxl.readAccel(&x, &y, &z); // Read the accelerometer values and store them in variables declared
above x,y,z
if(x < AccelMinX) AccelMinX = x;
if(x > AccelMaxX) AccelMaxX = x;
if(y < AccelMinY) AccelMinY = y;
if(y > AccelMaxY) AccelMaxY = y;
if(z < AccelMinZ) AccelMinZ = z;
if(z > AccelMaxZ) AccelMaxZ = z;
Serial.print("Accel Minimums: "); Serial.print(AccelMinX); Serial.print(" ");Serial.print(AccelMinY);
Serial.print(" "); Serial.print(AccelMinZ); Serial.println();
Serial.print("Accel Maximums: "); Serial.print(AccelMaxX); Serial.print(" ");Serial.print(AccelMaxY);
Serial.print(" "); Serial.print(AccelMaxZ); Serial.println();
Serial.println();
/* Note: Must perform offset and gain calculations prior to seeing updated results
/ Refer to SparkFun ADXL345 Hook Up Guide: https://learn.sparkfun.com/tutorials/adxl345-hookup-guide
/ offsetAxis = 0.5 * (Acel+1g + Accel-1g)
/ gainAxis = 0.5 * ((Acel+1g - Accel-1g)/1g) */
// UNCOMMENT SECTION TO VIEW NEW VALUES
//accX = (x - offsetX)/gainX; // Calculating New Values for X, Y and Z
//accY = (y - offsetY)/gainY;
//accZ = (z - offsetZ)/gainZ;
//Serial.print("New Calibrated Values: "); Serial.print(accX); Serial.print(" "); Serial.print(accY);
Serial.print(" "); Serial.print(accZ);
//Serial.println();
while (Serial.available())
{
Serial.read(); // Clear buffer
}
}

```



```
}
```

Accelerometer (Raspberry Pi)

Basic Accelerometer Readings

```
import time
import board
import busio
import adafruit_adxl34x

i2c = busio.I2C(board.SCL, board.SDA)
accelerometer = adafruit_adxl34x.ADXL345(i2c)
```

```
while True:
    print("%f %f %f"%accelerometer.acceleration)
    time.sleep(1)
```

Further Accelerometer Functions

```
import time
import board
import busio
import adafruit_adxl34x

i2c = busio.I2C(board.SCL, board.SDA)
accelerometer = adafruit_adxl34x.ADXL345(i2c)
accelerometer.enable_freefall_detection(threshold=10, time=25)
accelerometer.enable_motion_detection(threshold=18)
accelerometer.enable_tap_detection(tap_count=1, threshold=20, duration=50, latency=20, window=255)
```

```
while True:
    print("%f %f %f"%accelerometer.acceleration)
    print("Dropped: %s"%accelerometer.events["freefall"])
    print("Tapped: %s"%accelerometer.events['tap'])
    print("Motion detected: %s"%accelerometer.events['motion'])
    time.sleep(0.5)
```