Advanced lab course for bachelor students in physics

Experiment T15
Arduino mini-experiments

February 2025

You are welcome to bring your own laptops to interface with the Arduinos

The student will be expected to provide the following items in the event that COVID restrictions require the experiment to be performed off-site:

- Water kettle (Wasserkocher) or some form of heated water
- Tape measure, ruler, meterstick, or some form of accurately measuring distances
- Laptop

Prerequisites

- Arduino/Microcontrollers+ROOT
- Basic Radiation Principles and Detectors

Goal of the experiment

- Gain Familiarity with Microcontrollers
- Perform experiments to explore basic physics results using Microcontrollers
- Perform experiments that apply basic knowledge of quantum physics concepts

Contents

1 Introduction and overall purpose of the lab experiment

The goal of this experiment is to introduce students to modern lab practices using microcontroller technology. This will be done by guiding the student through several mini-exercises to measure basic physics observables that the student has been expected to have encountered previously. The emphasis, here, will be understanding how to build microcontroller circuits to make a physics measurement and understand if the microcontroller makes an accurate and precise measurement.

Many interesting and useful circuits can be controlled by microcontrollers. These circuits are used to perform otherwise operationally difficult measurements. There are a large number of sensors and break-out boards that are available and can easily be integrated with most common microcontrollers available that simplify many circuits. As such, the increase in the ease of using microcontrollers, coupled with the ever-reducing cost of building microcontroller circuits allows them to find many uses among electronics hobbyists and scientific researchers alike.

There is a large universe of microcontrollers that one can select from. So, which microcontroller is the best to use? The answer to this question depends alot on the specifics of the application for which you want use them. And it often comes down to the following, though not completely exhaustive, list of questions:

- How many connections do I need for connecting components to the microcontroller?
- Does my circuit have special voltage or current requirements that only a small subclass of microcontollers are able to operate at?
- Does my microcontroller support the appropriate peripherals for my measurement?

Many times standard general purpose microcontrollers are sufficient for the measurement that needs to be made. These classes of measurements will be the focus of this lab experiment. To introduce the student to the exciting world of microcontroller technologies, we will use the industry standard Arduino Uno Rev3 to perform measurements. These boards find an exceedingly common use in microcontroller applications as they are easy to use, inexpensive, and there is plenty of online-based support for new and experienced users alike. Even though we are using a standard microcontroller, all of the mini-experiments performed in this experiment are able to be performed with most other microcontroller boards available on the market.

To read and store the data collected by the Arduino, a computer is used. The Arduino IDE provides a way of writing code to control the Arduino operation using the C programming language and facilitates uploading the code to the Arduino. It can be downloaded at https://www.arduino.cc/en/software. Simply navigate to the link and select the version supported by your operating system (OS). Click the download button and follow the installation instructions for your OS. The instructions for your OS may require an online search. In addition, Python programs (often referred to as scripts) can be written and run to easily collect data from the Arduino as well as controlling how the Arduino changes the state of the circuit.

Each chapter that follows is an experiment that stands on its own. The structure of each chapter follows the same general structure. First, a brief parts list is provided followed by a brief motivation of the measurement. If necessary, a brief discussion of physics concepts will be provided. This is followed by a description of each of the prominent circuit components used in the mini experiment. After this, a guide to setting up the circuit and making the appropriate connections is provided. Following this, a discussion of setting up the Arduino program (also called a sketch) is developed. Lastly, any relevant suggested Python coding is discussed, though the motivated student is welcome to develop the code with their own novel ideas, provided they include appropriate motivating comments when submitting the program as requested with the analysis deliverables at the end of the experiment.

In the interest of saving the reader some valuable time, we will describe the microcontroller hardware that is used in all of the mini-experiments here, rather than repeating them in each chapter.

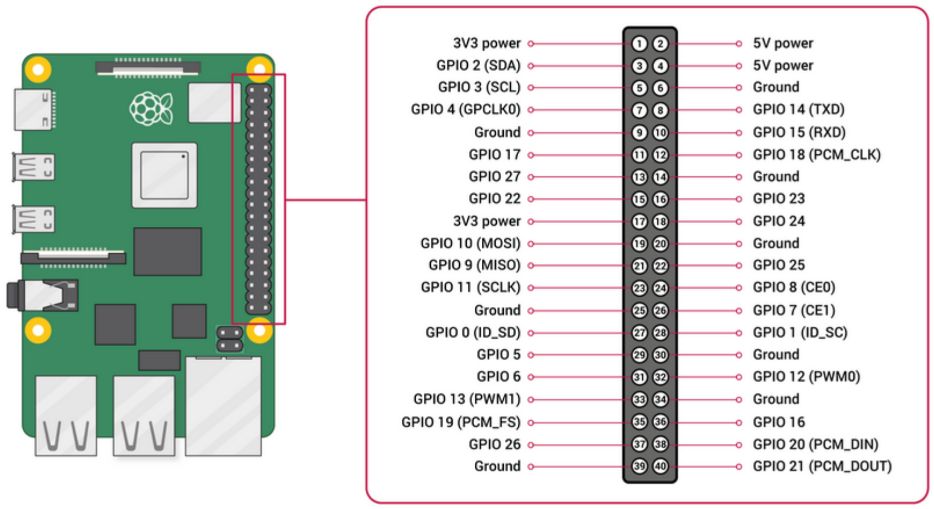
1.1 Raspberry Pi

For this experiment, we use the Raspberry Pi 4B as the data collection and storage medium. The Rasberry Pi is a single-board computer that has recently found popularity for teaching basic computer science principles as well as more niche applications such as robotics. There are several different models of the Rasberry Pi 4 available, the 4A, 4B, and 4Zero. The primary difference between the different models is the size and maximum available memory. We choose to use the 4B model as it optimizes size and available computing resources.

The RaspberryPi is similar to most other Linux-based PCs in that it has an operating system, CPU, RAM, and hard data storage. It comes equipped with a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor. It does not come with a pre-built OS like most PCs that you could buy from a retailer, however its parent company provides two supported OSs. These are Raspbian and Raspebrry Pi OS. The two are very similar, though the primary difference is that Raspberry Pi OS is offered in a 64-bit version that can utilize the maximum RAM available on the PC, while Raspbian is a 32-bit architecture that utilizes a maximum of 3 GB of RAM. The only hardware resource that differentiates the types of 4B models is the amount of available RAM. Thus, the Raspberry Pi is available to be purchased with 1, 2, 4, and 8 GB of RAM.

The Raspberry Pi 4B has a fair number of peripherals that can be connected. It also has 4 USB A connections, 2 USB 2.0 and 2 USB 3.0. The 4B has two micro-HDMI connection supporting two monitors simultaneously. The Rasberry Pi has a USB C connection that allows appropriate AC/DC power supplies to be connected. It also has an RJ-45 gigabit ethernet connection to support wired internet. The Pi also comes with on-board Wifi and Bluetooth 5.0 capability. Lastly, the Raspberry Pi provides 40 GPIO pins giving it the ability to perform similar operations to the Arduino. We will make use of this functionality during the experiment.

The Raspberry Pi provided will contain the desktop version of Rasbian which is a debian-based LINUX operating system. If you are unfamiliar with operating a LINUX system, there are many great tutorials available online. However, we will only use simple commands on the Terminal. The Terminal is a program very similar to that of the Command Prompt on Windows PCs. To open a terminal, simply click on the terminal icon and it should pop open a terminal prompt.



https://www.raspberrypi.org/documentation/computers/os.html

Figure 1: Pinout of the Raspberry Pi 4B showing each pin's functionality

The commands that will be used are quite simple. We will cover the most common ones here. If you want to make a new directory named 'data', you would execute (excluding the quotes): 'mkdir data'. The 'cd' command is used when you wish to change directories. For example, if you are in the home directory (in LINUX this is the ~/) and you want to change to the 'data' directory you would execute (again, without quotes): 'cd data'. To copy a file, it has the form: 'cp <path_to_file> <path_to_where_file_should_go>'. To read a file, the EMACS text editor is included. If you are familiar with a different editor (e.g. pico, nano, vim, etc.) you are welcome to use them if they are already included. To open a file with EMACS, use the command: 'emacs -nw <name_of_file>'. If you are unfamiliar with how to use EMACS special key short cuts, brief explanations can be found online, or ask your lab supervisor. If we need any LINUX commands that are more complicated than this in any of the mini-experiments, we will explicitly describe them in the relevant sections.

1.2 Arduino Uno Rev3

The Arduino Uno is a microcontroller board consisting of many different connectors, inputs, outputs, etc. that hosts the ATmega328P microcontroller IC. This IC is the workhorse of the Arduino Uno board. It controls all of the inputs, outputs, and communications between the board and the circuit that it is interacting with. It hosts 14 digital input/output (I/O) pins. 6 of these pins can be used for pulse width modulation (PWM), which essentially takes the average electronic signal by chopping the incoming signal into discrete parts. The Uno also hosts 6 analog I/O pins capable of 10 bit resolution when converted to/from digital. In addition, the Uno has a 16 MHz ceramic resonator for on-board clocking. The board can be powered either with an AC/DC wall adaptor, or by connecting the USB B port on the board to a USB A port on a computer via a USB cable. The USB cable also provides a means of Serial communication for data transfer.

The Uno also has some specification limits. The DC current limit on I/O pins is 20 mA and the maximum input voltage range is 6-20 V, with a recommended range of 7-12 V. The Uno does support some basic memory for temporary storage. There is 32 kB of flash memory that the ATmega can use, 0.5 kB is utilized by the bootloader. In addition, the ATmega has 2 kB of SRAM and 1 kB of EEPROM memory available.

The board is also quite compact for one with so much utility. The board is 68.8 mm in length and 53.4 mm wide. On top of all of this, it only weighs 25 g. Which means that it has a size similar to that of a credit card. There are smaller versions of the Arduino available, e.g. the Nano, as well as larger versions, e.g. Leonardo and Mega. Though for our purposes, we happily choose the Uno for these experiments.

1.3 Arduino Nano

The Arduino Nano is another microcontroller, similar to the Uno, but smaller and the pinout is somewhat different. There are four primary differences. Firstly, the physical dimensions are noticeably smaller at 18 mm wide by 45 mm. Secondly, there's no AC power jack available. In addition, the USB connection uses a mini(not micro!)-USB B connector to facillitate both the 5 V power connection and the Serial connection. Lastly, and most importantly, the Nano connects via header pins soldered to the board connections. This makes utilization of a breadboard extremely convenient, otherwise female-to-female jumper pins would need to be used to connect to the other circuit elements.

1.4 PYTHON

To collect data and interact with the Arduino PYTHON scripts will be used. These experiments will assume that the student possesses a basic prior knowledge of PYTHON. In particular students should understand:

- how to import libraries
- if/else statements
- for and while loops
- how to define simple functions and call them

If you need a refresher, there are many good turotials on-line. If you get stuck with specific questions, feel free to contact the lab supervisor. The important new PYTHON item for most students will be using the PySerial library to communicate with the Arduino. Essentially, this adds only two new lines that the student will need to become familiar with. First, at the beginning of the script, the PySerial library will need to be imported with the call: 'import serial' Then somewhere later, but still near the beginning of the code, the line: 'ser = serial.Serial('<Port_Name>', <baud_rate>)' will need to be added to establish the Arduino Serial connection. Specifics on where this line should occur will be discussed later in the experiment, for now we only want to give an overview of its use for new users. This connection will allow data to be transferred between the Arduino and PYTHON. The baud rate is the rate of serial communication in Hz. This value must match the value in the Arduino sketch, which will be discussed shortly. The port name is the path defining the physical location of the of the USB port that the Arduino is utilizing. There are a few different ways that the port name can be found. This is most easily found by looking in the Arduino IDE, which we will describe now.

1.5 Arduino IDE

The Arduino individual development environment is a program that facilitates interaction between the user and the Arduino. It is in this IDE that the student will write all of the code that governs how the Arduino controls the experiment's circuit. To open the arduino IDE, if not already done, open a terminal and type 'arduino &'. This brings up the IDE and has already populated the code with two shell functions that are almost always used in every single Arduino program. As a reminder, we continue here in C.

The first function is the 'setup()' function. In this function you place all initializations that need to be done at the start of the program. Everything called/performed in the setup() function will only run once. If they need to be called again, or otherwise modified, they must occur in the second loop, or some other user-defined function.

The second function that is automatically added for the user is the 'loop()' function. This function is called immediately after the setup function completes and is repeated as long as the Arduino is operating. This is the function where the majority of the data acquisition code will reside. It is possible to have other user-defined functions, however all of the experiments are simple enough that no other function definitions will be necessary, so we will not discuss them. The interested student can look up how to define functions using C in one of the many online tutorials. A brief list of common functions that could be useful in this experiment as well as a brief description of each function can be found in Appendix C.

The typical code structure within the IDE will be the following:

- Author comments
- Include any necessary sensor libraries
- Define any global variables needed
- Perform any setup calls that need to be done.
- Put the data acquisition code in the loop function

1.5.1 Author comments

Proper programming etiquette, as well as the experiment deliverables, requires that the program author interject his/her name, followed by the date and a brief description of how the code should function. This piece of the code is often referred to as the preamble. In most C-style languages, single line comments are designated by starting the line with a double slash: '//'. It's possible to make multi-line comments. These comments start with a slash-star: ‘/*’ and end with a star-slash: ‘*/’. An example of an author comment would be:

```
/**************************************
**
** Author: Max Mustermann
**
** Date: Mar. 17, 2017
**
** This code collects data from the
** experiment by doing the following
** really cool methods.
**
**************************************/
```

1.5.2 Include libraries

It is impossible for the designers of any programming languages to forsee the implementation of every piece of code that one may need in a program. However, to write all of the code over and over again creates alot of unnecessarily long programs. To simplify this, developers will write entire libraries that can be included into a program that give the user access to powerful functions that have been predefined by the developers. To gain access to these functions, they are included into the program by using the ‘#include’ command. For example if the user wants to include the specifc library for the ADS1115 4 channel 16 bit gain amplifer analog to digital converter, they would use the command ‘#include <Adafruit_ADS1015.h >’ This library give access to specific developer written functions that make accessing the data processed by the ADS1115 significantly easier because the user need not define these functions directly inside the code. It should also be noted here that many of these libraries are not built into the IDE by default.

In order to give the IDE access to these libraries, they must be downloaded to the local Arduino/libraries folder. This allows the libraries to be linked properly when building the program. In order to download these, you must connect to the internet. For the Raspberry Pis, they connect via the RWTH wifi. For other PCs, you must provide your own connection. In order to do this within the Arduino IDE, you must:

1. In the ‘Tools’ Menu select the ‘Manage libraries...’ item
2. After a few seconds a pop-up search menu opens
3. In the search bar, type the name of the library that you want to add.
4. The list will update.
5. Within the description of the library there will be two items, a drop down menu and an ‘Install’ button

6. The drop down menu lists all of the available version of the library. Unless you have a good reason to do otherwise, you should select the latest version of the library

7. Click the 'Install' button

8. Once the library has finished installing, it can be used like any other C library.

1.5.3 Variable instantiation

Often it is useful to have variables defined to store and manipulate data. Initializing variables in C is similar to how this is done in PYTHON, though variable instantiation is a little different. C makes use of many of the same variable types that PYTHON does, e.g. float, character, string, double, integer, boolean, etc. Where C is most different is that the user is required to tell the compiler how it should treat the variable. Thus in C, for the compiler to know that a variable should be treated as a boolean, you need to define: 'bool var = True;' Whereas in PYTHON, the program instinctively knows that, by definition, none of the other variable types could possibly take 'True' to be a valid value. Thus in PYTHON, the program optimizes the variable type without the user having to specify, which can save a lot of typing. Another, C-style note to make here is the semi-colon at the end of the statement. Unlike PYTHON, C-style languages require punctuation to know when the end of a statement occurs. This is almost always done by using the semi-colon.

When writing Arduino code, it is usually best to define all variables as global variables when possible. This is especially important if you plan on using the same variable in the setup, loop, or other user-defined functions. The best place to define your global variables is at the start of the program, after the library include statements, but before the setup function.

1.5.4 The setup() function

The setup function is the place where any variables, circuit component states, or processes that need to start, or be initialized should occur. This is also the place for any commands that need to run once, and only once should occur. Anytime that Serial communication needs to occur between the Arduino and a computer, or bluetooth module, this should be initialized in the setup function. To initialize the Serial connection, the command 'Serial.begin(<baud_rate >);' should be used. The baud rate (in Hz) is the serial communication rate that data is transferred between the Arduino and other device that is used. This value **must** match on the sending **and** receiving part of the serial communication. That way the PC knows how fast to expect the Arduino to talk, and vice versa. The typical baud rate used in this experiment is 9600 Hz, but this can be different depending on timing issues are present in the circuit.

This is also a good place to set the pin mode of any pins used by the Arduino. For example, if you have two pins, say 7 and 8, that read and write states to the circuit, respectively, you can tell the code how to set this up here. For pin 7 that reads data, it needs to be set as an input and can be done with the command: 'pinMode(7, INPUT);'. For pin8, it writes the circuit to be in a particular state, and needs to be defined as an output. This can be done with the command: 'pinMode(8, OUTPUT);'. You may put as many commands here as needed.

1.5.5 The loop() function

The loop function is where you will take in and work with most of your data on the Arduino side. For optimization reasons, you should only perform data collection processes here and minimize the number of non-essential operations. This is because the Arduino will process each line of code, which takes some (usually) small amount of time. If this process is simply doing math and not collecting new data, the Arduino will not be listening for new data while it is performing this other process. It is best to perform math processes in the PYTHON script on the host PC, when possible.

It is impossible to discuss all general cases of how to setup the loop function. When specific strategies should be employed, they will be discussed in the appropriate section of the lab manual.

2 Mini-experiment #1: Blink LEDs with Microcontrollers

2.1 Mini-experiment #1A: Blink an LED with an Arduino Uno

2.1.1 Parts list

1. 1 x Arduino Uno R3
2. 1 x 1000 Ω Resistor
3. 1 x LED (Can be any color not UV or IR)
4. 1 x Breadboard
5. Jumper Wires

A full list of parts is given in Appendix B.

2.1.2 Purpose and Procedure

The purpose of this mini-experiment is to introduce the concept of microcontroller technology with a simple example. The simplest experiment that can be done with an Arduino is to use the Arduino to light an LED. The Arduino has already been described in the introduction text. We place all of our circuit components on a breadboard.

In this introductory experiment, you will be introduced to the concept of making an Arduino circuit and collecting data. The procedure that you will follow is to setup a circuit and data collection process that will store the LED state for five oscillations blinking an LED. An oscillation of a blinking LED will be defined as the LED being ON for 4 seconds and OFF for 2 seconds. You will make the proper connections to build the circuit. Then write an Arduino sketch and PYTHON program to control and store the state of the LED. Let's get started.

2.1.3 Circuit Component Description

LED:

Light emitting diodes (LEDs) are common circuit components used for a variety of reasons in electronic circuits. They are most often used as indicators to tell the user if a particular process is running within the circuit. LEDs are a subclass of diodes, which is itself a class of electronic circuit component.

A diode is an electronic circuit component that, due to its physical properties, only allows electric current to flow in one direction. A diode is formed by taking two materials, one that has an excess of negative charge carriers (electrons) and one that has a deficiency of negative charge carriers (holes), and mating them together. The material that has an excess of negative ion is referred to as n-doped, while the one that has ion holes is referred to as p-doped. Because one material has an excess and the other a deficiency, this causes current to tend to only flow in one direction. It is possible to force a diode to operate in reverse bias, a state which is called avalanche breakdown, but this is beyond the scope of this experiment.

As electric current flows through the circuit, electrons passing from the n-region to the p-region of the LED recombine with the electron holes. If these electrons recombine in an excited state within the atom, a process of electroluminescence in which the electron releases a photon when it transitions between the conduction band and valence band of the atom. Electrons initially begin in the conduction band as they conduct by definition in a solid. When they transition to the valence band, which is the outer-most ground state that the electron will occupy, they pass through the band gap. The band gap is a region in which no electronic states can form. The energy difference, corresponding to the band gap determines the wavelength of the photon that is released by the LED.

Resistor:

The resistor is one of the most common electronic component that is encountered in electronic circuits. The resistor is a passive circuit element that is intended to be operated in the Ohmic region of circuit design, which is a region in which the linear relationship between voltage and current is given by Ohm's law:

$$V = I \cdot R \tag{1}$$

where V is the voltage applied to the resistor, I is the current passing through it, and R is its resistance. The resistance of a resistor is generally temperature dependent, though at common operating temperatures we can take this diffence to be negligible allowing us to calculate the resistance using the following:

$$R = \frac{\rho \cdot L}{A} \tag{2}$$

where L is the resistor's length, A is its cross sectional area, and ρ is the resistivity of the material used to construct the resistor. Resistivity is simply a measure of a material's ability to resist conduction of electrons.

Resistors can be added to circuits, and depending on how they are added, can change the resistance of the circuit in different ways. If a resistor is added in series to another resistor, this can be thought of as increasing the overall length of resistance to electron flow and increases the total resistance of the circuit:

$$R_{eq} = R_1 + R_2 + ... + R_n \tag{3}$$

where R_{eq} is the total, or equivalent, resistance of the circuit, and R_i is the resistance of the ith resistor. It is also possible to add resistors in parallel branches to the initial resistor. Adding a new resistor branch adds an extra path for electrons to flow, thus reducing the resistance of the overall circuit. This can be described mathematically as:

$$\frac{1}{R_{eq}} = \sum_i \frac{1}{R_i} \tag{4}$$

Breadboard:

The breadboard, also known as prototyping board, is a popular way to connect circuits in a temporary way to test circuit designs. Once circuits have been determined to be final and will see repeated use, often printed circuit boards are created as they are more resilient in devices relying on the circuit to operate reliably. Breadboards come in many different shapes and sizes.

Since we will not need to make many different connections between circuit components and sensors, we opt for a standard breadboard that is split into two halfs that operate the same way. Along both sides of the breadboard, there are two sets of connections, one labeled blue and one red, that run the entire length of the side. All of the pins along the length of a set of connections are connected together. It is generally intended to use these to supply power to your circuit. It is conventional to use the red to connect to the operating voltage and blue to connect to ground. In the center of the board, you will find 30 rows of connections that are split in half. All of the connections in the same numbered row, on the same half of the board are connected together. This allows us to use jumper wires to connect the componets on the breadboard to each other as well as connect to the Arduino.

2.1.4 Making the connections

To connect the circuit shown in Fig. 4, do the following:

1. Connect the 'GND' pin on the Arduino to one of the pins located on the blue rail of the breadboard.
2. Connect the LED into pins on two different numbered rows in the center of the breadboard. Make sure to note where the positive (long) pin is plugged in.
3. Connect pin 12 on the Arduino to a pin in the same row, on the same side of the breadboard as the positive pin on the LED.
4. Connect one of the legs of the resistor to one of the pins int the same row, on the same side of the breadboard as the negative (short) leg of the LED.
5. Connect a jumper wire between one pin in the blue rail that the GND pin is connected to and a pin in the same row, on the same side of the breadboard as the unconnected leg of the resistor.

The circuit should now look similar to that shown in the figure.

2.1.5 Setting up the Arduino Sketch

To setup the Arduino sketch, open the Arduino IDE. You should now see a shell sketch that consists of two void type functions, setup() and loop(). The very first thing to do would be to create

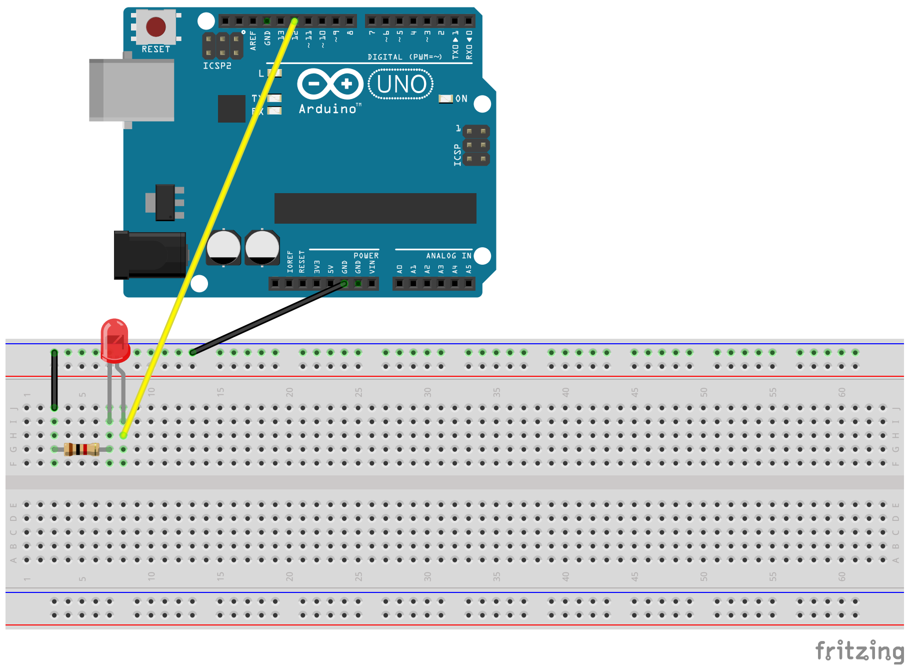


Figure 2: Schema connection for mini-experiment # 1

a multi-line comment to list the author of the sketch, date, and brief description of the experiment, which we will refer to as the 'preamble' in the remainder of this lab.

After completing the preamble, you will need to define two variables. One for the pin location on the Arduino so that the code knows to talk with the appropriate pin. And one for the incoming information byte from the PC. Both of these values should be of type 'int'.

After these variables have been defined, the setup function can be populated. Only two commands will need to be placed here, the Serial.begin() function and the pinMode to set pin 12 on the Arduino as an output. For help on setting up these commands, please refer to the introduction section on the Arduino sketch.

Once the setup function has been setup properly, we may now move on to the final function, the loop() function. In this function only three methods will be needed.

The very first thing that the Arduino should do is to initialize the state of the LED. It should set the state of the pin to LOW. This is done by making a digitalWrite(<LED_PIN>,LOW) Hold for some amount of time and then change the state of the pin from LOW to HIGH in the digitalWrite command. The Arduino should hold the state of the LED for some amount of time in each case. This can be done using the delay(NNNN) command where NNNN is some amount of time in milliseconds. Each time that the Arduino changes state it should perform as Serial.Write() command sending the state of the LED out over the Serial bus to be read by the PC.

Once this logic has been implemented into the Arduino sketch, click the checkmark icon in the top left of the tool bar to compile the code and check for errors. The program may prompt you to provide a name for your project to save under. This will check for SYNTAX errors ONLY. If it compiles with no errors, there could still be logic errors which can be much tougher to catch. After the code compiles without errors, under the ‘Tools’ menu on the upper menu bar, find the ‘Board’ option. Make sure that it has selected the ‘Arduino Uno’ option. Next, check the ‘Port’ option to confirm that it is talking to the correct port, i.e. the one connected to the Arduino. It will usually have the Arduino device in parentheses next to the port name. Note this port name as this is the same port that your PYTHON code will need to interact with. Lastly, click the right-pointing arrow in the tool bar. This will upload the sketch directly to the Arduino. Next we will move on to writing the PYTHON code to interact with the Arduino.

2.1.6 PYTHON

We will use a PYTHON program, or script to interact with the Arduino and store the LED state data. We will need two external PYTHON libraries to interact with the Arduino and read the state of the LED properly, the PySerial and time libraries. You may need to install these libraries for your version of PYTHON. To include these libraries, you will need to import them as ‘serial’ for PySerial, and ‘time’.

After this create a file with an appropriate name to store the data. Next, define a serial connection variable using the port name and baud rate used in the Arduino sketch. Then create a loop to read the Arduino and write data to file.

Then tell PYTHON to hold the same state for a specific time by using the time.sleep() method. The argument should have units of seconds and be of type float.

2.2 Mini-experiment #1B: Blink Two LEDs with an Arduino Nano

2.2.1 Parts list

1. 1 x Arduino Nano
2. 2 x 1000 Ω Resistor
3. 2 x LED (Can be any color(s) not UV or IR)
4. 1 x Breadboard
5. Jumper Wires

A full list of parts is given in Appendix B.

2.2.2 Making the connections

To connect the circuit shown in Fig. 4, do the following:

1. Insert the Nano's header pins into the breadboard such that the Nano straddles the center divider of the breadboard.
2. Connect the 'GND' pin on the Arduino to one of the pins located on the blue rail of the breadboard.
3. Connect the LED into pins on two different numbered rows in the center of the breadboard. Make sure to note where the positive (long) pin is plugged in.
4. Connect pin D10 on the Arduino to a pin in the same row, on the same side of the breadboard as the positive pin on the LED.
5. Connect one of the legs of the resistor to one of the pins int the same row, on the same side of the breadboard as the negative (short) leg of the LED.
6. Connect a jumper wire between one pin in the blue rail that the GND pin is connected to and a pin in the same row, on the same side of the breadboard as the unconnected leg of the resistor.
7. Repeat the process for a second pair of LED and resistor, using pin D8.

The circuit should now look similar to that shown in the figure.

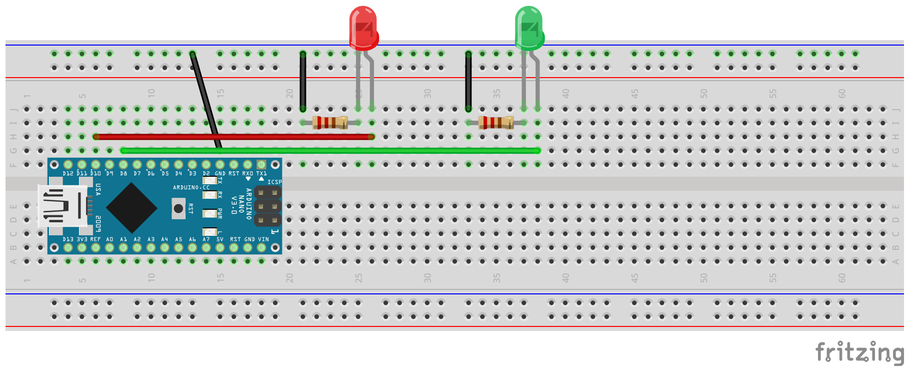


Figure 3: Schema connection for mini-experiment # 1

2.2.3 Setting up the Arduino Sketch

Follow the same procedure as mini-experiment 1.A with the following exceptions:

1. Replace the pin variables with the appropriate pin information for the Nano.
2. Add additional variables for the second LED.
3. Change the board option to Nano

2.2.4 PYTHON

We will use a PYTHON program, or script to interact with the Arduino to store the LED state data. We will need two external PYTHON libraries to interact with the Arduino and read the state of the LED properly, the PySerial and time libraries. You may need to install these libraries for your version of PYTHON. To include these libraries, you will need to import them as ‘serial’ for PySerial, and ‘time’.

After this create a file with an appropriate name to store the data. Next, define a serial connection variable using the port name and baud rate used in the Arduino sketch. Then create a loop to read the Arduino and write data to file.

Then tell PYTHON to hold the same state for a specific time by using the time.sleep() method. The argument should have units of seconds and be of type float.

2.3 Mini-experiment #1C: Blink Three LEDs with a Raspberry Pi **If performed at home, skip this**

2.3.1 Parts list

1. 1 x Raspberry Pi
2. 3 x 1000 Ω Resistor
3. 3 x LED (Can be any color(s) not UV or IR)
4. 1 x Breadboard
5. Jumper Wires

A full list of parts is given in Appendix B.

2.3.2 Making the connections

To connect the circuit shown in Fig. 4, do the following:

1. Connect the ‘GND’ pin on the Raspberry Pi to one of the pins located on the blue rail of the breadboard.
2. Connect the LED into pins on two different numbered rows in the center of the breadboard. Make sure to note where the positive (long) pin is plugged in.
3. Repeat this for the other two LEDs, making sure that no two LED pins are in the same column.

4. Connect one leg of a 1 kΩ resistor to an open pin in the same column as the first LED. The other leg of the resistor should connect to an open pin of a column with no other connections
5. Repeat this paring of a resistor with the other LEDs
6. Connect an open pin from the GND (blue rail) to the other leg of the LED
7. Connect a wire from GPIO pin GPIO23 on the Raspberry Pi to the other leg of the resistor.
8. Do the same connecting GPIO pin GPIO 24 to the second resistor and GPIO 25 to the third resistor.

The circuit should now look similar to that shown in the figure.

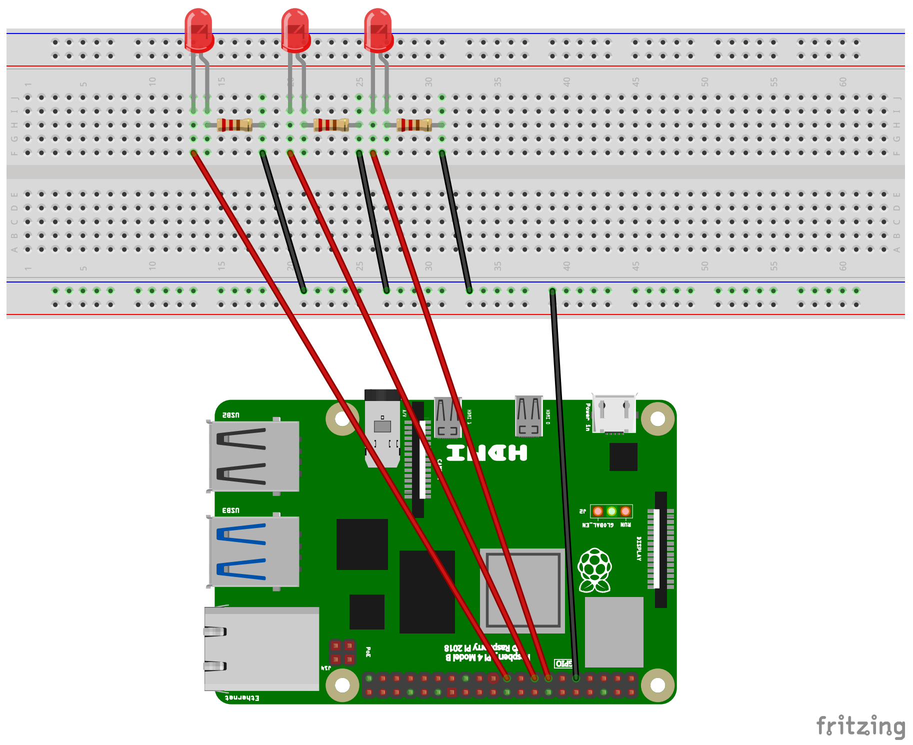


Figure 4: Schema connection for mini-experiment # 1

2.3.3 Setting up the Arduino Sketch

As we will only be using the Raspberry Pi, there is no setup using the Arduino IDE.

2.3.4 PYTHON

We will use a PYTHON program, or script to interact with the Arduino to change the LED state and store the LED state data. We will need two libraries RPi.GPIO and time. You may need to install these libraries for your version of PYTHON.

After this create a file with an appropriate name to store the data. Next, set LED pin variables. These are done using the 'pin' number not the 'GPIO' number listed in the 'Making the connections' subsection. For example the GPIO23 pin number is 16. These numbers can be found in the Raspberry Pi pinout map in the introduction.

Now, the Raspberry Pi has to be setup in the PYTHON code. The first command issued is GPIO.setmode(GPIO.BOARD) initializes the overall GPIO state. Next, GPIO.setwarnings(False) removes the warning verbosity. Lastly, GPIO.setup(ledPinN,GPIO.OUT) for each LED pin variable ledPinN, this sets the GPIO state of the pin to be an output.

Then create a loop to control the Raspberry Pi GPIO output and write data to file. To turn the GPIO on us GPIO.outPut(ledPinN, GPIO.HIGH) To turn off, replace GPIO.HIGH with GPIO.LOW In this loop, write the state of the LED to a data file for plotting

3 Mini-Experiment # 2: Temperature measurement

Note this mini-experiment should be operated in parallel with mini-experiments 3 and 4

3.1 Parts List

1. 1 x breadboard
2. 1 x Arduino Uno R3
3. 1 x DS18B20 Waterproof Termperature sensor
4. Jumper Wires
5. 1 x cup
6. 1 x mass scale
7. Water kettle (ask for use with lab supervisor)

3.2 Purpose and Procedure

The purpose of this experiment is to gain some understanding of how sensors can be used to measure thermal properties of materials. Here we will use a temperature sensor to measure the specific heat capacity of water by doing the following:

- Confirm that the Arduino is operating with the proper code.
- Take a cup and weigh it to get the mass of the cup.

- Fill the kettle with water from the sink.
- Heat the water with the kettle.
- Carfeully fill the cup with water.
- Place the temperature sensor in the water and allow it to read a stable reading.
- Make sure that the sensor data is being recorded by the PYTHON script.

3.3 Background

Solid materials are composed of atoms. These atoms ocillate and vibrate with a kinetic energy that is dependent on their temperature. Their energy is in a constant state of flux, dependent on their mass and a constant called the specific heat capacity. A material's specific heat capacity is a measure of how much energy the material releases to its environment based on its termperature and mass. An object's thermal energy, E_{th} at a given temperature, T, is given by:

$$dE_{th} = m \cdot c \cdot dT, \tag{5}$$

where m is the object's mass and c is its specific heat capacity.

The above only holds when there are no phase transitions. When there are phase transitions, the object does not change temperature as energy, heat, is exchanged, however it changes phase and the atoms spread out to form a liquid or gas as energy is added, or contract towards each other to form a liquid or solid as energy is removed. The energy transferred is proportional to the mass of the material that has changed phase and a constant called the latent heat. A material's latent heat is a measurement of how much energy is required to change the state of a certain amount of the material's mass. Materials have two different latent heat constants, one for the solid-liquid transition, called latent heat of fusion. The other is the latent heat of vaporization, which measures how much energy must be added or removed to change the materials phase in a liquid-gas state change. This is expressed mathematically by:

$$dE_{th} = L \cdot dm, \tag{6}$$

where L is the object's latent heat constant and m is its mass. For most puproses, any energy loss to the environment should be neglected for short time durations.

3.4 Circuit Component Description

The DS18B20 termperature sensor is waterproofed and ideal for measuring temperatures in wet environments where typical circuits can be damaged by water or humidity. The sensor measures well for temperatures in the range of -55 to +125$°C$. However, the heat shrink jacket is made from PVC, so temperatures should remain below 100$°C$ to avoid damage to the waterproof shielding. The expected input voltage range is 3.3 to 5.0 V. The circuit setup for this sensor is quite simple as it only has GND, V_{CC} and SIG pins for the ground, input voltage and signal outputs respectively.

3.5 Making the Connections

Making connections for this circuit are straight-forward:

1. Connect the 5 V pin on the Arduino to the red rail on the breadboard.
2. Connect the GND pin on the Arduino to the blue rail on the breadboard.
3. Plug the sensor circuit into the breadboard.
4. Run a jumper wire from the red voltage rail to a pin in the same numbered row on the same side of the breadboard as the V_{CC} sensor pin.
5. Run a jumper wire from the blue ground rail to a pin in the same numbered row on the same side of the breadboard as the GND pin on the sensor.
6. Run a jumper wire from digital pin 5 on the Arduino to a pin in the same numbered row on the same side of the breadboard as the DQ(S) pin on the sensor.

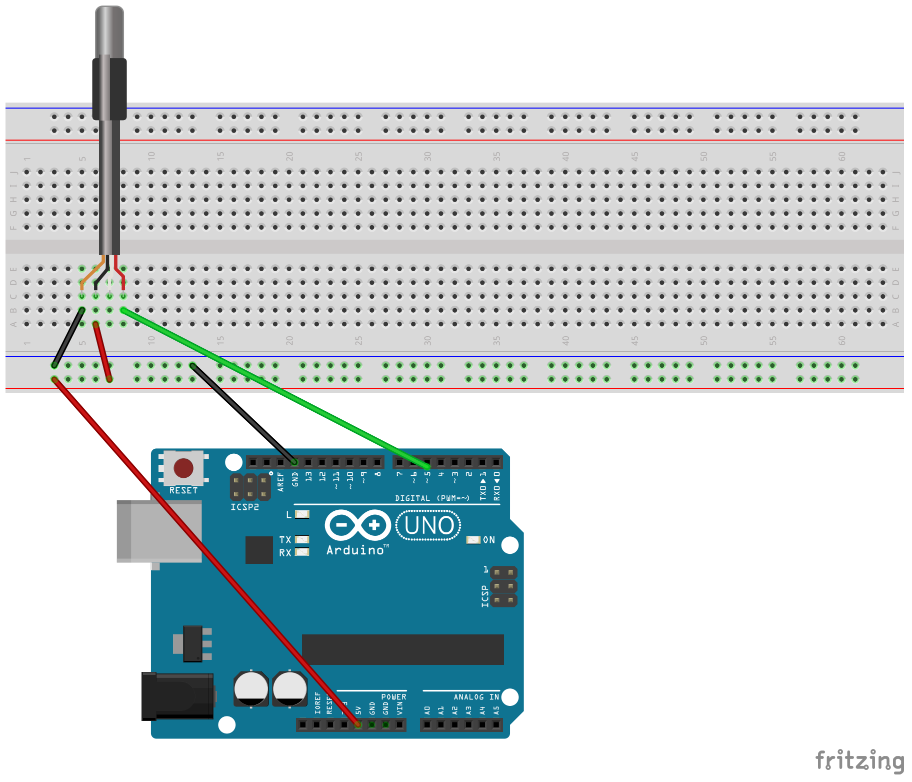


Figure 5: Fritzing model of the specific heat capacity circuit.

3.6 Setting up the Arduino Sketch

After creating the appropriate preamble, you will need the following libraries to use the temperature sensor:

1. The OneWire library (usually installed by default)
2. The Dallas Temperature Control Arduino library

Once these libraries are installed, include their header files in the sketch. Then define a ONE_WIRE_BUS variable attached to pin 5 of the Arduino. Instantiate a OneWire variable with: OneWire oneWire(ONE_WIRE_BUS). The temperature sensor needs to know the OneWire address to look at: DallasTemperature sensors(&oneWire). In the setup() function, begin the serial connection and the sensors communications.

In the loop function, you can access the temperature information by using: sensors.requestTemperatures(); To get the latest metric temperature reading: sensors.getTempCByIndex(0); Send this information out over the Serial line and select a reasonable delay for reading the temperatures.

3.7 PYTHON

The recommendation for setting up the PYTHON script remains similar to the previous experiments. Simply confirm that you are able to read in the temperature data and that the timing between the sketch and the script are synchronized properly.

4 Mini-Experiment #3 Basic kinematics and the speed of sound

4.1 Parts list

1. 1 x Raspberry Pi
2. 1 x Arduino Uno R3 (optional)
3. 1 x HC-SR04 Ultrasonic sensor
4. Jumper Wires
5. Implement of measurement, e.g. tape measure or ruler
6. 1 x Breadboard

4.2 Purpose and Procedure

The goal of this experiment is to begin to see how one can start adding specific sensors to work with microcontrollers. In this experiment you will use the HC-SR04 sensor to measure the speed of sound as well as basic kinematic quantities.

4.2.1 Procedure

1. Confirm that the Arduino code is loaded to the Arduino.
2. Confirm that the PYTHON script runs properly in concert with the Arduino.
3. With the sensor running, move towards the sensor at a slow but constant speed for 5-10 seconds.
4. Stop and stand still for 2 seconds.
5. Move away from the sensor at a slow but constant speed for 5-10 seconds.
6. Repeat steps 3-5 at a faster but constant speed.
7. After concluding the kinematics portion, measure the distance from an object using three different distances, e.g. 10 cm, 50 cm, and 100 cm; where these distances were first measured with a tape measure or meter stick.
8. Use these distance to calculate the speed of sound.
9. Repeat the previous two steps outside to calculate the speed of sound outside.

4.3 Background

4.3.1 Kinematics

In an introductory physics course, one of the first concepts that is taught is that of basic kinematics. Kinematics is the study of how motion of an object can be treated as a series of points for which some basic equations of motion hold true. If we consider an object in one dimensional space, an objects motion can be treated as a series of dots on number line. The distance between any two dots within some interval Δt of time is defined to be the object's displacement, $\Delta \vec{x}$. For the remainder of this discussion, consider that successive dots are spaced equally in time.

As the object moves through space, there are more and more dots on the number line and one can begin to consider what the spacing of the dots along the line means. If the dots between two equal time intervals are spaced equally, this means that the object is moving at the same speed during each of those time intervals. If we also consider two adjacent time intervals in which the distance between the first two dots is larger than the second pair of dots, the object is slowing down. If the reverse is true, then the object is speeding up. In both of these cases, the speed of the object is changing. This corresponds to an acceleration.

To put this mathematically, the time rate of change of the displacement, is the derivative of the displacement. We have already briefly discussed the speed, and this is the magnitude of the change in displacement. If we consider that this change in displacement also has a direction, this quantity is the object's velocity. The velocity can be expressed as:

$$\vec{v} = \frac{d\vec{x}}{dt} \tag{7}$$

As we have discussed earlier, the object's speed can change, and thus it's velocity changes when it experiences an acceleration. An acceleration can also cause a change in direction without causing a change in speed, however this would require motion to be allowed in more than one direction and we will restrict ourselves to the 1-D case for this experiment. If an object has constant velocity, it does not experience an acceleration. Keeping with our dots of motion analogy, the object is slowing down if displacement between successive time intervals is decreasing. This is equivalent to the object speeding up in the opposite direction, thus the direction of the velocity and acceleration vectors must be opposite to each other. In the case where the object's displacement is increasing in successive time intervals, the object is speeding up. In this case, the object's acceleration and velocity vectors are aimed in the same direction. To frame this mathematically, the acceleration is the time rate of change of the velocity:

$$\vec{a} = \frac{d\vec{v}}{dt} \tag{8}$$

Which can be related to the displacement by:

$$\vec{a} = \frac{d^2\vec{x}}{dt^2} \tag{9}$$

4.3.2 Pressure Wave and the Speed of Sound

The phenomenon known as sound can be best described as interactions between atoms in the air where the particles collide with each other and impart their momentum to the eardrum in a person's ear. The interactions of the particles produces longitudinal waves, whose momentum transfer imparts a force on an object of a certain area, and thus these waves are often referred to as pressure waves. This pressure is registered by the eardrum as sound level intensity. The greater the pressure, the greater the sound level intensity, thus the louder the perceived sound.

The rate at which the particles collide is termperature dependent. The velocity at which the sound wave is traveling is determined by the rate at which the air particles collide with one another. This rate is primarily determined by the air particles' kinetic energy, which is temperature dependent. The more kinetic energy the particles have, the faster they are able to collide with their adjacent particles and the faster the sound wave propagates through the air. The astute student will also note that the larger temperature allows for greater momentum transfer which translates to louder sound, however we will focus primarily on the speed of sound.

4.4 Circuit Component Description

The only circuit component that will be used in this experiment is the HC-SR04 Ultrasonic Motion Sensor. This sensor has two subcomponents to it. One subcomponent, the trigger that emits a 40 kHz sound wave that reflects off of objects. The reflection of the sound wave is received by the echo subcomponent. The sensor's internal circuitry measures the time between when the wave was emitted and the time at which it is received by the sensor.

The sensor is designed such that it issues a series of 4 full cycles of 40 kHz sound per 5 μs interval of time that the trigger is set to digital HIGH.

4.5 Making the connections for the Arduino

The connections for the sensor are reasonably straight-foward.

1. Connect the 5 V pin on the Arduino to the red power rail on the breadboard.
2. Connect the GND pin on the Arduino to the blue power rail on the breadboard.
3. Plug the HC-SR04 sensor into the the breadboard.
4. Jumper the V_{cc} pin on the sensor with the red power rail.
5. Jumper the GND pin on the sensor to the blue power rail.
6. Connect the Echo pin on the sensor to pin 2 on the Arduino.
7. Connect the Trigger pin on the sensor to pin 3 on the Arduino.

4.6 Making the connections for the Raspberry Pi

1. Connect the 3.3 V pin on the Raspberry Pi to the red power rail on the breadboard.
2. Connect the GND pin on the Raspberry Pi to the blue power rail on the breadboard.
3. Plug the HC-SR04 sensor into the the breadboard.
4. Jumper the V_{cc} pin on the sensor with the red power rail.
5. Jumper the GND pin on the sensor to the blue power rail.
6. Connect the Echo pin on the sensor to an available GPIO pin on the Raspberry Pi.
7. Connect the Trigger pin on the sensor to an available GPIO pin on the Raspberry Pi.

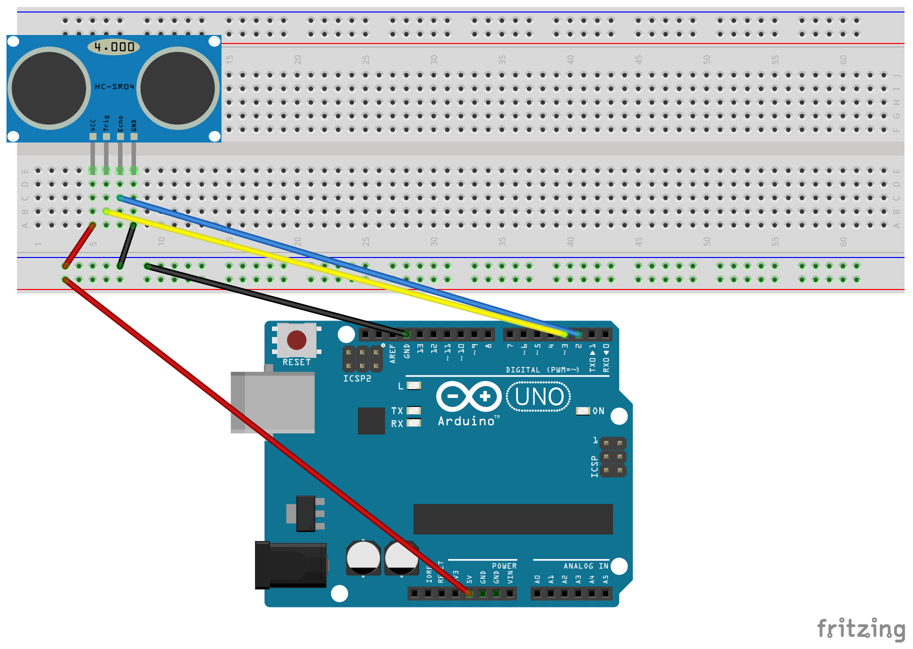


Figure 6: Fritzing model of the motion sensor circuit.

4.7 Setting up the Arduino Sketch

After creating the appropriate preamble, define two pin variables for the TX and RX (Trigger and Echo) pins. Also, define two more variables, one to store the distance and another to store the time measurement from the sensor. In the setup function, start the Serial communication for the Arduino and set the pin modes appropriately.

In the loop function the recommended steps are to set the TX pin LOW for a few microseconds before starting the data collection. This guarantees that even if the sensor was stuck in some weird initial state that we start data collection from a well defined state. Next set the TX HIGH for 10 μs then set the TX to LOW. To hold the system in a particular state for some number of microseconds, it is recommended to use the delayMicroseconds() function that takes an integer value of microseconds as its argument.

To collect data from the sensor, the pulseIn(RXpin, HIGH) function will return the amount of time between the emittance and collection of the wave in microseconds. The distance that the sensor is away from an object can be calculated using equation 5.

4.8 PYTHON

For the Kinematics plots, you should write a PYTHON script that stores the distance in well defined time intervals that can be used to generate displacement, velocity, and acceleration plots by reading in the output of the Arduino's measurement from the HC-SR04 sensor. To read from the Arduino, utilize the serial.Serial.readlines() function to read the incoming data. To read and store the data properly, make sure that the timing between when the Arduino sends data and the data collection process are synchronized well.

5 mini-Experiment #4 Acceleration

5.1 Parts list

1. 1 x Raspberry Pi
2. 1 x Battery Pack for the Raspberry Pi
3. 1 x LSM9DS1 Breakout board
4. Jumper Wires
5. 1 x Breadboard
6. Protractor

5.2 Purpose and Procedure

The goal of this experiment is to measure the acceleration that one experiences in an elevator and see that this is different depending on wheter the elevator is moving upwards or downwards. A separate goal is to measure the magnetic field of the earth as well as that of a solenoid. In this experiment you will measure these quantities with the help of the LSM9DS1 sensor.

5.2.1 Procedure for Acceleration ***If performed off-site SKIP THIS PART***

- After making the connections for the electronics, take the apparatus into the elevator in the Physikzentrum.
- Confirm that the Arduino is operating with the proper sketch running.
- Start the PYTHON data collection running.
- Starting from the ground floor, take the elevator up to the top floor.
- Starting at the top floor, take the elevator to the ground floor.
- Confirm that the data is recorded, otherwise repeat.

5.2.2 Measure the Angle of Earth's Gravitational Force and Magnetic Field

- Create a unit circle using a protractor.
- Note: on the accelerometer, the axes are drawn on the PCB.

- Collect acceleration data by rotating the accelerometer through π rad along each axis.
- The goal is to search for the orientation of the local graviational force and magnetic field vectors of the Earth (specifically in Aachen: Can be/and is likely slightly different from what may be reported online at official sites).

5.3 Background

5.3.1 Forces and Acceleration

As discussed in the previous mini-experiment with the motion sensor, acceleration is an integral part of life as we experience accelerations of many different kinds. Of them, one of the more notable is the acceleration due to the Earth's gravitational field. A gravitational field exists in the vicinity of masses, such that masses exert an attractive force on other masses that is proportional to the masses of the objects, and is inversely proportional to the separation between the two. Isaac Newton developed the mathematical expression to describe this with the following famous equation:

$$\vec{F} = G\frac{m_1 m_2}{(\vec{r_2} - \vec{r_1})^2} \tag{10}$$

where m_1 and m_2 are the respective masses of the two objects, $\vec{r_1}$ and $\vec{r_2}$ are the distance vectors relative to their coordinate origin, and G is Newton's gravitational constant, $6.67 \times 10^{-11}\frac{Nm^2}{kg^2}$.

This law holds for all classical gravitational problems that one could encounter. Near the surface of the Earth, $m_2 = m_{earth}$ and the quantity $|\vec{r_2} - \vec{r_1}| = r_{earth}$ and the portion of the above equation: $G\frac{m_{earth}}{R^2_{earth}}$ is a constant redefined as $g = 9.81\frac{m}{s^2}$. This reduces the equation for the gravitational force to the following:

$$\vec{F} = m_1\vec{g}. \tag{11}$$

The direction of $\vec{g}$ is towards the geometric center of the Earth, which to reasonably good approximation, we take to be the downward direction normal to the surface of the Earth on which we are standing, provided the piece of Earth that we are standing on is locally flat.

Newton also developed his three famous laws of motion, of which the second law will be most useful in this experiment. Newton's second law of motion describes the inertial force that an object with mass m experiences due to an acceleration $\vec{a}$:

$$\vec{F} = m\vec{a}. \tag{12}$$

If the only force that an object experiences is gravitational, then it is clear to see by direct comparison of the above equations that the acceleration the object experiences is the gravitational acceleration. Which causes one to wonder if a person is standing still on the surface of the Earth and is not moving, shouldn't the net acceleration be zero? This, of course, is true and there must be another force balancing the effects of the gravitational force. This force is called the normal force and is due to the electrostatic interactions between the atoms in contact between the person and the Earth's surface such that it keeps the person from falling into the Earth. In fact when one steps on a scale, the scale is in fact reading the normal force applied to the person. This becomes more interesting when thinking about what a scale would read with an object placed on it in a moving elevator.

Let's consider this problem with a bathroom scale and a box placed on top of the scale in an elevator. In the case where the elevator is not moving, obviously this is the same case as earlier and the scale is expected to read the gravitational force applied to the box. Next consider the two cases: one in which the elevator is accelerating upwards, and the other in which the elevator is accelerating downwards. What do we expect the scale to read in each case? How about the acceleration? We can measure the acceleration using the LSM9DS1 following the procedure outlined in the previous section.

5.3.2 Magnetic Field and Forces

Many of us are familiar, to some degree, what a magnet is. Magnets have two poles, a north and a south pole. The south pole of the magnet will align to point toward the north pole of the magnetic field, while the north pole of the magnet will point towards the south polarity of the field.

Even electrically charged particles are affected by magnetic fields. Particles with electric charge, q, moving with velocity, $\vec{v}$, in a mgnetic field, $\vec{B}$, experience a force given by:

$$\vec{F}_B = q\vec{v} \times \vec{B} \tag{13}$$

In the last half of the 19th Century, James C. Maxwell showed, mathematically, that changing electric fields induce magnetic fields and changing magnetic fields induce electric fields. From this, moving charges, i.e. electric currents produce magnetic fields as they pass through wires. We will be interested in measuring the magnetic field due to a solenoid, which is given by:

$$B = \mu_0 n I, \tag{14}$$

where μ_0 is the magnetic permeability of free space, n is the number of turns per unit length of the solenoid, and I is the magnitude of the electric current moving through the coil. *Hint:* The LSM9DS1 measures the magnitude of the magnetic field from all sources.

5.4 Circuit Component Description

The LSM9DS1 is a 9 degree of freedom (DoF) breakout board that contains three sensors: an accelerometer, a magnetometer, and a gyroscope. Each of these sensors is capabale of measuring along three Cartesian axes, thus giving a total of 9 DoFs. This sensor allows you to measure the linear acceleration, magnetic field, and rotation that the user is experiencing. These boards are primarily used for orienting the user or circuit, or to complement a GPS circuit.

The sensor communicates with the accelerometer using the I^{2}C protocol. I^{2}C is predominantly used for allowing lower-speed ICs to communicate with processors and microcontrollers over short-distances and between several different boards. This process utilizes two open, bi-directional communication lines for Serial data (SDA) and Serial clock (SCL) controlled with the help of pull-up resistors and allow for 5 V and 3.3 V powering. The I^{2}C allows for master and slave relationships between multiple processors. The master controls the communication. For our purposes, the Arduino is the master and all other sensors are the slaves. For most complicated circuits with multiple breakout boards, the I^{2}C protocol is the current preferred standard of communication of data and clocking control between circuits. Others exists, but we will focus our attention on I^{2}C.

5.5 Making the Connections for Arduino

To connect the accelerometer properly do the following:

1. Connect the 5 V pin on the Arduino to the red rail of the breadboard.
2. Connect the GND pin on the Arduino to the blue rail of the breadboard.
3. Plug the LSM9DS1 sensor into the center of the breadboard such that one set of pins is on one side of the middle divider of the breadboard and the other set of pins is on the other side of the middle of the breadboard.
4. Run a jumper wire from the red rail of the breadboard to an adjacent space in the same numbered row of the V_{in} pin on the LSM9DS1.
5. Run a jumper wire from the blue rail of the breadboard to an adjecent space in the same numbered row of the GND pin on the LSM9DS1.
6. Run a jumper wire from the SDA pin on the Arduino to an adjacent space in the same numbered row of the SDA pin on the LSM9DS1.
7. Do the same for the SCL connections on the Arduino and the LSM9DS1.
8. If using the battery pack,

5.6 Making the Connections for Rasberry Pi

To connect the accelerometer properly do the following:

1. Connect the 3.3 V pin on the Raspberry Pi to the red rail of the breadboard.
2. Connect the GND pin on the Raspberry Pi to the blue rail of the breadboard.
3. Plug the LSM9DS1 sensor into the center of the breadboard such that one set of pins is on one side of the middle divider of the breadboard and the other set of pins is on the other side of the middle of the breadboard.
4. Run a jumper wire from the red rail of the breadboard to an adjacent space in the same numbered row of the V_{in} pin on the LSM9DS1.

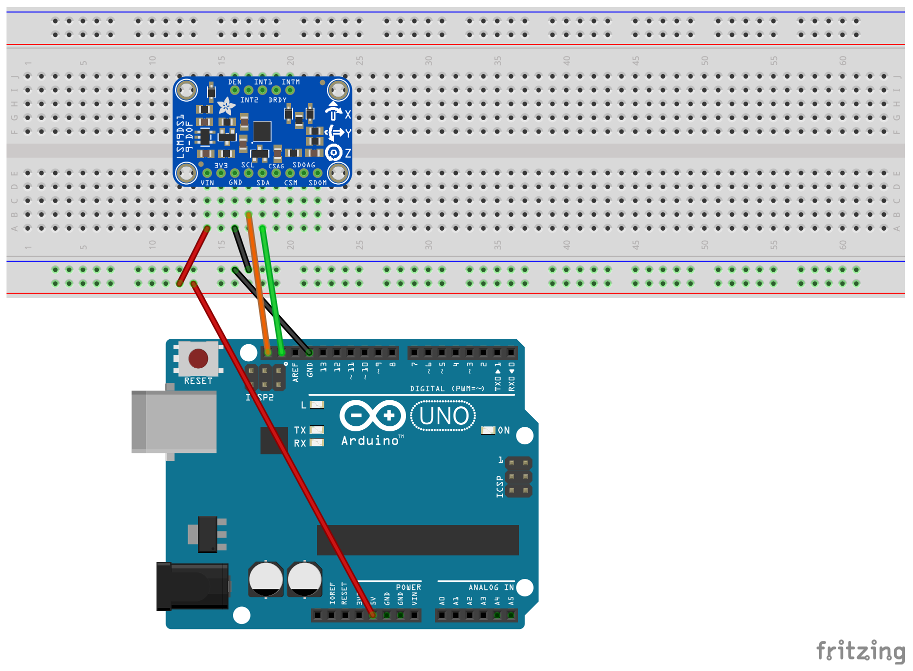


Figure 7: Fritzing model of the accelerometer circuit.

5. Run a jumper wire from the blue rail of the breadboard to an adjecent space in the same numbered row of the GND pin on the LSM9DS1.
6. Run a jumper wire from the SDA pin on the Raspberry Pi to an adjacent space in the same numbered row of the SDA pin on the LSM9DS1.
7. Do the same for the SCL connections on the Raspberry Pi and the LSM9DS1.
8. If using the battery pack, save all of the data on the Raspberry Pi, turn off the Raspberry Pi, unplug the Power supply from the Raspberry Pi, plug the battery pack in to the power port on the Raspberry Pi.

5.7 Setting up the Arduino Sketch (If using an Arduino Microcontroller)

For more complex breakout boards, like the LSM9DS1, new libraries will need to be installed within the Arduino IDE. To do this, in the 'Tools' Menu, select the 'Manage Libraries...' option. This will pop-up a new search window. In the search bar of this new window, search for: 'Adafruit LSM9DS1'. This will populate a list of libraries with a similar name in addition to the library that you want. Find the Adafruit LSM9DS1 library box and confirm that the dropdown option has the latest version of the library selected and click installed. This may take a minute to complete. Once the install has finished, the library is available to use within the IDE.

For particularly complex breakout boards, oftentimes, the library download will also include example code to get started and test the code. To do this go into the 'File' menu and look for the ‘Examples’ submenu. Find the Adafruit LSM9DS1 library under the ‘Examples from Custom Libraries header. Selecting this option will pop-up a predefined Arduino sketch for using the LSM9DS1. Going through this code there are alot of items that will be new to the user and we will only describe the items relevant to collecting the data needed. We should note that in the setup function, the Serial connection baud rate that is required is specific to the LSM9DS1 and should be made sure to be used in the PYTHON script. The first change that the user should make in the loop function is to remove the print lines related to the gyroscope as these are not used. From here, the user should modify the print statements to send the information out via the Serial connection in a way that is easy to parse for the PYTHON script. Also, confirm that delay is reasonable in collecting data at intervals that make sense for data collecting.

5.8 PYTHON

Setup the PYTHON script to communicate with the Arduino at the proper baud rate. Also setup the script to store the magnetic field strength and acceleration value in a way that is easy to plot as a function of time.

6 Mini-experiment #5: Planck’s Constant

6.1 Parts list

1. 1 x Raspberry Pi
2. 1 x Arduino Uno R3 or Nano (optional)
3. 1 x 1000 Ω Resistor
4. 1 x MCP4725 Digital to Analog Converter (DAC)
5. 1 x ADS1115 Analog to Digital Converter (ADC)
6. 10 x Light Emitting Diode (LED) - each one should be a different color (wavelength
7. Jumper Wires
8. 1 x Breadboard

6.2 Purpose and Procedure

The procedure is to repeat 10 times, once for each color LED (wavelength/frequency), measure the current vs. voltage (I-V) curve of the LED. To do this, step through the analog voltage ranges from 1.4 V to 3.4 V in 5 mV steps. The setup measures the voltage across the safety resistor. This voltage measurement is used to calculate both the current through the circuit, and the voltage across the LED. The calculated current and voltage values will generate the I-V plot for the LED.

The threshold current of the LED is taken to be 10^{-4} A. The corresponding voltage value is taken to be the threshold voltage for the LED to calculate the energy of the electron, and equivalently, the energy of the photon.

From the distribution of these threshold voltages vs associated photon frequency, a linear fit will find the value of the Planck constant.

To step through the spectrum of applied voltages, the Arduino supplies a constant +5 V digital signal to the DAC. The Arduino uses timing code that is communicated via the I2C (SDA/SCL) line to control the analog voltage output from the DAC. As the voltage steps through, the ADC constant sends an analog repsonse to the Arduino, also using the I2C line that is reported to the user.

6.3 Background

The photoelectric effect is one of the prominent, foundational theories that helped to develop the earliest models of quantum theory. In 1905, Albert Einstein formulated the mathematical relationship that described interesting behavior, observed by Thomson and others, in which electromagnetic radiation incident upon a metal had the potential to produce an electric current. It had already been understood, qualitatively, that the light needed to be of sufficient energy to remove electrons and that the relationship between the intensity of the light and the amount of current measured was linear.

Einstein argued that light traveled in individual packets called quanta. For light, the quanta is often referred to as photons. The photons have energy:

$$E = h\nu, \tag{15}$$

where h is Planck's constant and ν is the frequency of the incident light. This relationship between light energy and frequency was found several years before Einstein's

Translating this energy into the minimum energy required to liberate electrons from the atom gives:

$$\Phi = h\nu_0. \tag{16}$$

If the photon has exactly the amount of energy to remove the electron from the atom, then: $\nu = \nu_0$. However, if the energy is larger than that required to liberate the electron, the electron leaves with kinetic energy:

$$E_{kin} = E - \Phi = h(\nu - \nu_0). \tag{17}$$

However, for the case of the LED, the photoelectric effect operates in the reverse direction. Electrons collide with the semiconductor material to produce light that is emitted. As the voltage is increased, the intensity of the light (number of photons) increases, but the energy remains the same as the color (wavelength) of the light emitted does not change. The increase in light intensity is therefore due to the increased current from increasing the voltage.

One equation that is useful for working with diodes is the Shockley equation:

$$I = I_0(exp(\frac{V}{V_0}) - 1) \tag{18}$$

where I is the diode current as a function of the diode voltage, V, I_0 is the diode saturation current, and V_0 is the threshold voltage of the LED. This equation allows an emipirical estimation of the threshold voltage for an LED to turn on (emit photons). It is most useful as a mathematical fitting method.

6.4 Circuit Component Description

The basic principles of the resistor and LED were described in previous mini-experiments. The components that we will describe here are the other two breakout boards, the ADS1115 ADC and the MCP4725 DAC. For the LEDs, we use the following wavelengths:

Color	wavelength (λ) [nm]
Infrared	940
Red	722
Orange	702
Lime	692
Yellow	687
Green	568
Blue	498
Pink	479
Purple	431
Ultraviolet	413

6.4.1 ADS1115 Analog to Digital Converter

The ADS1115 is a 16 bit 4-channel ADC with a programmable gain amplifier (PGA). The Arduino has 6-channels of 10 bit ADCs, however sometimes it is nicer to have finer precision, or simply have the ability to change the precision as necessary. The ADS1115 has four channels that it can read (A0 - A3) that are read out by the Arduino's I2C bus. The 16 bit granularity coupled with the PGA allows for the following available gains: 2/3, 1, 2, 4, 8, and 16. One should take great care when selecting the gain as there are voltage limitations associated with using those settings. For larger gains, the voltage at which the ADC can experience damage is reduced.

One final interesting note on the ADS1115 is that it has an addressing pin that allows up to four ADS1115s to be utilized by the Arduino simultaneously on the same I2C bus. To set the address, the ADDR pin should be sonnected to: GND for master setting, V_{DD} for slave1, SCL for slave2, and SDA for slave3. The master/slave relationship is primarily only used in highly complicated circuits, usually robotics/mechatronics applications where there are significant amounts of data being transmitted. Last, and potentially most importantly, the overall voltage and current limits to the Adruino must also be respected unless driving the ADS1115 from a separate power source. However, we will restrict ourselves to simpler cases for this mini-experiment.

6.4.2 MCP4725 Digital to Analog Converter

The MCP4725 is a 12 bit single channel DAC. The MCP4725 communicates with the Arduino using the I2C bus. To set the analog voltage in the IDE, simply specify the bit value as a ratio of the 12 bit maximum 4096, and this is the ratio of the input voltage V_{DD} that is output at the V_{out} pin to the rest of the circuit. Similar to the ADS1115, it is possible to connect two MCP4725s together along the same I2C bus via the A0 pin, however only two MCP4725s can be connected along the same bus without conflicting. It is also possible to use the MCP4725 with the same I2C bus as the ADS1115s simply by tying into the same SCL/SDA connections of the I2C bus.

6.5 Making the Connections for the Arduino

To setup the hardware to take data do the following:

1. Plug the MCP4725 DAC, ADS1115 ADC, the 1000 Ω resistor, and the LED in to the breadboard.
2. Connect the +5 V pin on the Arduino to the '+' rail on the breadboard.
3. Connect the GND pin on the Arduino to the '-' rail on the breadborad.
4. Connect the V$_{DD}$ pins on the DAC and the ADC to the '+' rail on the breadboard.
5. Connect the GND pins on the DAC and the ADC to the '-' rail on the breadboard.
6. Connect the SCL pin on the Arduino to the SCL pin on the DAC and ADC.
7. Connect the SDA pin on the Arduino to the SDA pin on the DAC and ADC.
8. Connect the ADDR pin on the ADC to the GND wire on the ADC.
9. Connect the V$_{out}$ pin on the ADC to the '+' (long) contact lead on the LED.
10. Connect the '-' (short) contact lead on the LED to one lead on the resistor.
11. Connect the other lead on the resistor to the '-' rail on the breadboard.
12. Connect the A$_0$ pin on the ADC to the positive leg of the LED.
13. Connect the A$_1$ and A$_2$ pins to the LED/resistor connection.
14. Connect the A$_3$ pin to the end of the resistor leg that is connected to ground. This completes the hardware connections for this setup.

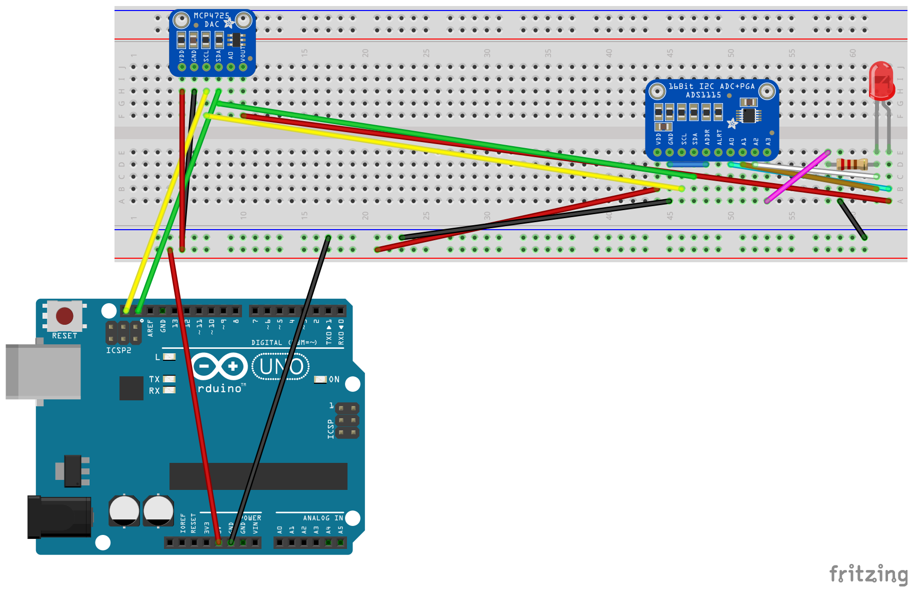


Figure 8: Fritzing model of the Planck's constant circuit.

6.6 Making the Connections for the Raspberry Pi

To setup the hardware to take data do the following:

1. Plug the MCP4725 DAC, ADS1115 ADC, the 1000 Ω resistor, and the LED in to the breadboard.
2. Connect the 3.3 V pin on the Raspberry Pi to the '+' rail on the breadboard.
3. Connect the GND pin on the Raspberry Pi to the '-' rail on the breadborad.
4. Connect the V_{DD} pins on the DAC and the ADC to the '+' rail on the breadboard.
5. Connect the GND pins on the DAC and the ADC to the '-' rail on the breadboard.
6. Connect the SCL pin on the Raspberry Pi to the SCL pin on the DAC and ADC.
7. Connect the SDA pin on the Raspberry Pi to the SDA pin on the DAC and ADC.
8. Connect the ADDR pin on the ADC to the GND wire on the ADC.
9. Connect the V_{out} pin on the ADC to the '+' (long) contact lead on the LED.
10. Connect the '-' (short) contact lead on the LED to one lead on the resistor.
11. Connect the other lead on the resistor to the '-' rail on the breadboard.
12. Connect the A_0 pin on the ADC to the positive leg of the LED.
13. Connect the A_1 and A_2 pins to the LED/resistor connection.

14. Connect the A_3 pin to the end of the resistor leg that is connected to ground. This completes the hardware connections for this setup.

In this circuit you will be performing Single-ended reads of the voltages at different points in the circuit. A Single-ended read compares the voltage read by the analog input pin to that of an internal ground to calculate a voltage difference. Thus one can measure the voltage of a specific point in a circuit in multiple places in this way. A partial circuit schematic showing the essential Single-ended connections for this experiment are shown in the following figure.

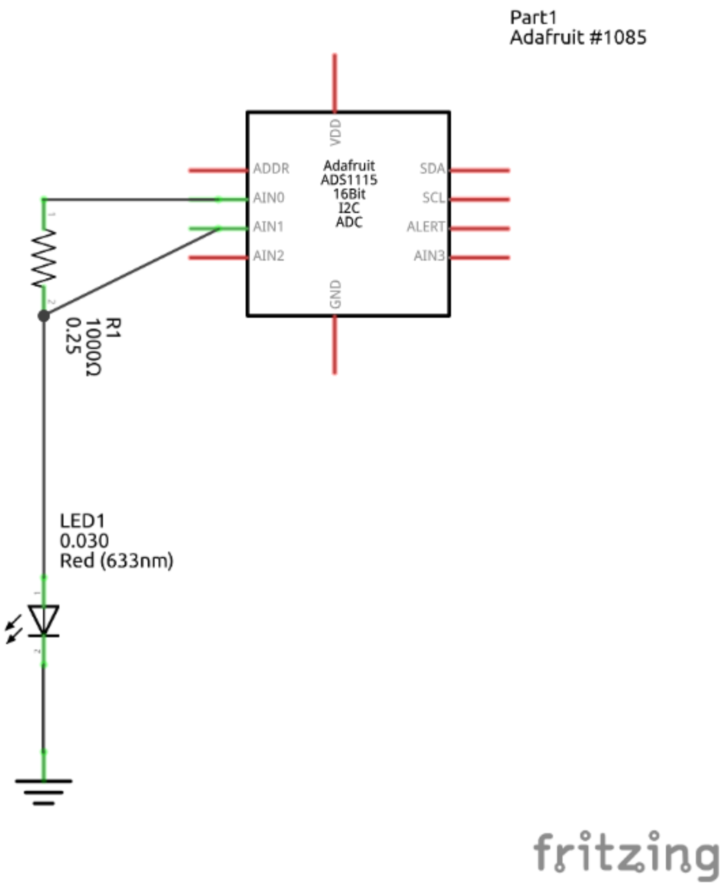


Figure 9: A partial schematic showing the essential connections for making a series of Single-ended reads in the mini-experiment.

6.7 Setting up the Arduino Code

First, create an appropriate preamble for this experiment. The ADS1115 and MCP4725 have special libraries that must be used. To collect the library for the ADS1115, search for and install the library: Adafruit ADS1015. To collect the MCP4725 library, search for and install the library: Adafruit MCP4725. Once these libraries have been installed, make sure to include their header files in the sketch.

Create DAC and ADC constructors similar to:

```
Adafruit_MCP4725 <var_name1>;
Adafruit_ADS1115 <var_name2>(0x48);
```

The hex address is necessary to tell the Arduino what state to expect the ADC to be in. This is related to the address of the ADC: 0x48 is master, 0x49 for slave1, 0x4A for slave2, and 0x4B for slave3. The address must match the ADC ADDR configuration.

In the setup function, start the Serial connection. Then begin the DAC connection, specifying the address: var_name1.begin(0x62); Next set the ADC gain: var_name2.setGain(GAIN_ONE); If you require a different gain, replace 'ONE' with 'TWOTHIRDS','TWO', 'FOUR', etc. Then start the ADC: var_name2.begin();

In the loop function, define the ADC signal variables as int16_t datatypes. Then define the DAC bit value as a uint32_t datatype. Initialize your variable to zero. Create a for loop to loop through the voltage that the DAC will apply making sure that the voltage value is specified in terms of DAC bits corresponding to the voltage that you would like to apply. The conversion factor from bits to voltage is 5.0/4096.0. This conversion should be performed as a float type since the voltage can take on continuous values, not only discrete. To set the voltage: var_name1.setVoltage(dac_value, false); The second argument tells the DAC if it should store the value in the EEPROM memory so that it stores the latest value. This is done in the event that the circuit loses power or other connectivity during operation so that the DAC begins in the state that it left off at. This is not necessary for us, so we set this flag as 'false'.

Within this loop, the ADC values should be read to the variable(s) that you created earlier: var_name2.readADC_SingleEnded(n); Here n takes on values between 0-3 corresponding to the A0-A3 read pins on the ADC. Simply print and store the values to the Serial bus for the data to be collected by the PYTHON script.

6.8 PYTHON

Collection of the data will be similar to previous data collection runs. The voltage will continue to cycle and only one iteration through the loop of all possible voltage values is necessary. Simply take this into account when writing your script.

7 Analysis

Now that you have completed the experiment, it is time to analyze the findings. To do this, you should be prepared to **submit your code**, *both* the Arduino sketch and the PYTHON script that you used for *ALL* mini-experiments in this lab experiment. The preferred method would be email, but it is also acceptable to transfer the files by USB stick to the lab supervisor. You should submit you results as plots with descriptions of your plots and physical conclusions that can be drawn from them. In this experiment, make sure to describe how the Arduino was used to collect the data.

Based on the experiments, as well as their analyses carried out, perform a simple error/uncertianty analysis on each of the anlyses above. Does it make sense to use the microcontroller in these situations? If not, explain why. As best as possible, qualitatively compare these to the uncertainty on measurements if these experiments were (hypothetically)carried out with more traditional materials, e.g. meterstick, analog voltmeter, etc. Give a brief discussion of each. Where would it not make sense to use the Arduino to perform experiments? Are there times where more traditional measurement devices are preferable to use?

7.1 Blink an LED

In this experiment you were able to control an LED using an Arduino and PYTHON. Submit the following:

- A plot of the ON/OFF state of the LED as a function of time.
- A description of how the Arduino turned the LED ON/OFF.

7.2 Temperature measurement

In this experiment you were able to measure the temperature of water as a function of time. Submit the following:

- A plot of the temperature of the water as a function of time.
- Discuss if these results make sense.

7.3 Basic kinematics and the speed of sound

In this experiment, you were able to produce some simple kinematic plots and measure the speed of sound. Submit the following:

- A plot of the 3 main kinematic distributions as a function of time.
- A discussion of these plots and if the readings from the electronics make sense. If they do not, why not?
- Present your findings for the speed of sound and discuss if the speed of sound that you measure is accurate.

7.4 Acceleration

In this experiment, you were able to measure the acceleration, and thus the force applied on a person in an elevator. You were also able to measure the acceleration and magnetic field vectors of the Earth. Submit the following:

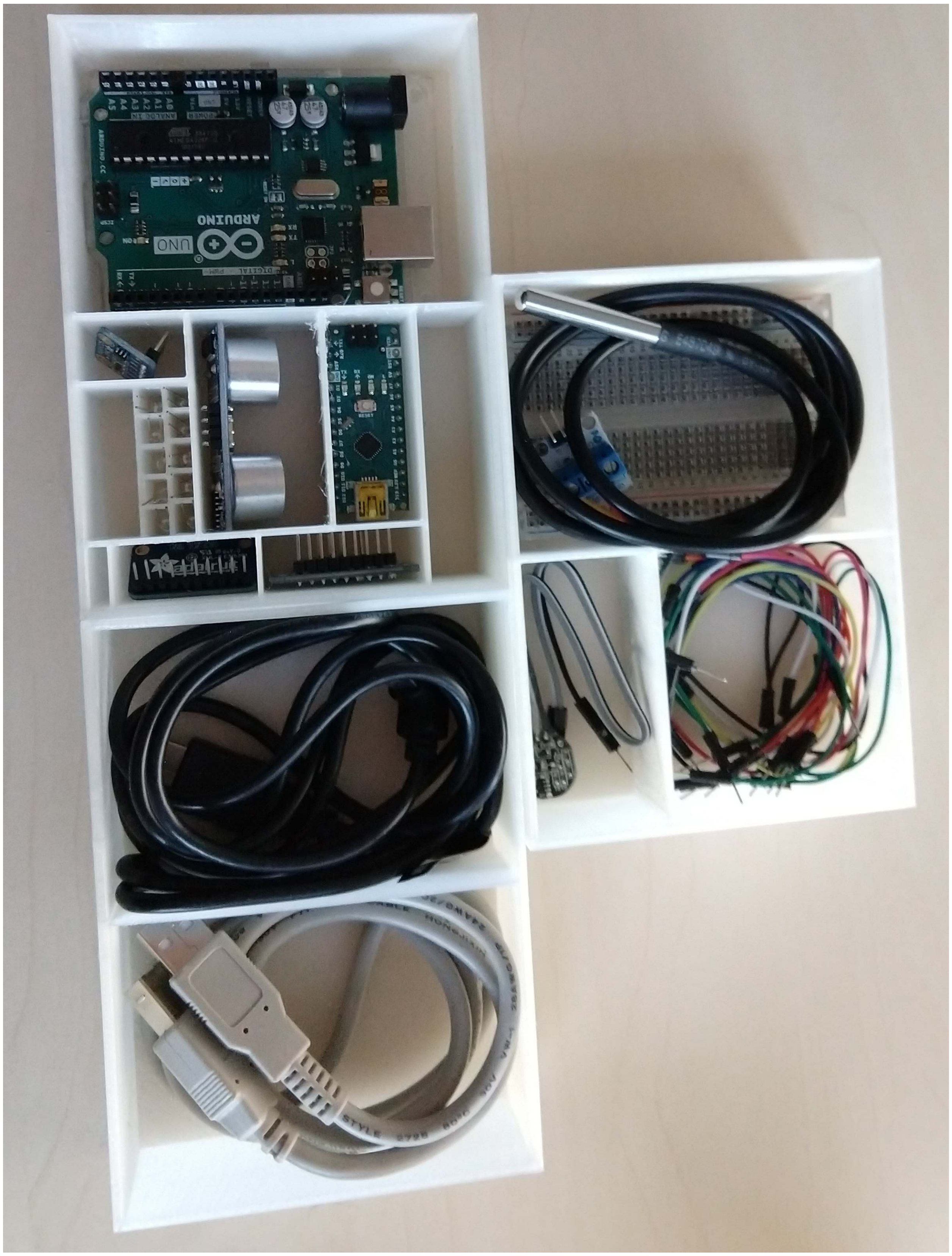
- A plot of the acceleration as a function of time.
- Present the theoretical derivation of the acceleration of the elevator and discuss if the results in the plot make sense.
- A plot of the Earth's gravitational field (acceleration) strength as a function of angle for rotation about each major Cartesian axis.
- A plot of the Earth's magnetic field strength as a function of angle for rotation about each major Cartesian axis.

7.5 Planck's constant

In this experiment, Planck's constant was measured using some simple circuit elements. Submit the following:

- An I vs. V plot for each wavelength LED used. Make sure to note on each plot, the threshold voltage used for calculations.
- Note V_{Th} on the plots...can also be presented as a table if easier.
- A summary I vs. V plot showing all of the LEDs.
- A plot of V vs. ν used to calculate the Planck constant.
- Calculate the Planck constant.
- Do these results make sense?

A The Parts List



Part Number	Part Name
1	Arduino Uno
2	Arduino Nano
3	HC-SR04 Motion Sensor
4	MCP4725 DAC
5	8 LEDs (optical wavelength)
6	1 kΩ resistor + UV and IR LEDs
7	LSM9DS1 accelerometer
8	ADS115 ADC
9	Breadboard
10	temperature sensor
11	Jumper Wires
12	KY-039 Heart rate sensor
13	Arduino Nano USB cable
14	Arduino Uno USB cable

B Common Arduino Commands

In this section we list some common Arduino commands for reference that can be used by the student.

- Serial.begin(<baudRate>); – This command begins a Serial connection at a rate of 'baudRate' in Hz
- Serial.print("message"); – This command prints 'message' to the Serial line
- Serial.println("message"); – Same as Serial.print(), but adds a carriage return at the end of the line
- <pinVar>.pinMode(<pin>,<State>); – Sets the pin associated with pinVar to State either HIGH or LOW
- <pinVar>.DigitalRead(); – Performs a digital read operation for the pinVar pin
- <pinVar>.AnalogRead(); – Performs an analog read operation for the pinVar pin
- millis(); – Returns the number of milliseconds that have elapsed since starting the Arduino
- <pinVar>.DigitalWrite(); – Performs a digital write operation on the pinVar pin