MODULE 4: More Sensors

Even more sensors will be investigated in this lab. The Hall Effect sensor can detect proximity with the use of a magnet, an ultrasonic sensor can detect distance and an accelerometer can detect orientation and movement. With more options for collecting data, more creative solutions can be designed for intelligent systems.

GOALS:

- Investigate the use of new sensors
  - Hall Effect sensor for proximity
  - Ultrasonic sensor to detect distance
  - Accelerometer for sensing orientation and acceleration
- Discuss the use of a Moving Average to help filter data
- More python tricks, how to write code to reuse in multiple programs

Hall Effect Sensor

The Hall Effect sensor can detect the presence of a magnetic field. They can be used for a variety of applications such as detecting if a window or door is closed or detecting the rotational speed of an object. Generally, the sensors consume very low power that can be supplied by the Pi itself.

The sensor used in the lab will require a pull up resistor. An internal transistor is turned on and off by a magnetic field. When the transistor is turned on, a path from ground to the output pin is “turned on” which will results in a logic low signal. When the transistor is off, no path to ground is provided to the output and without the pull up resistor, the output will “float” resulting in an unreliable logical output. It is often common for sensors to require either pull up or pull down resistors. Always consult the datasheets for the various devices to verify the proper operation of the device as well as its limitations [1].

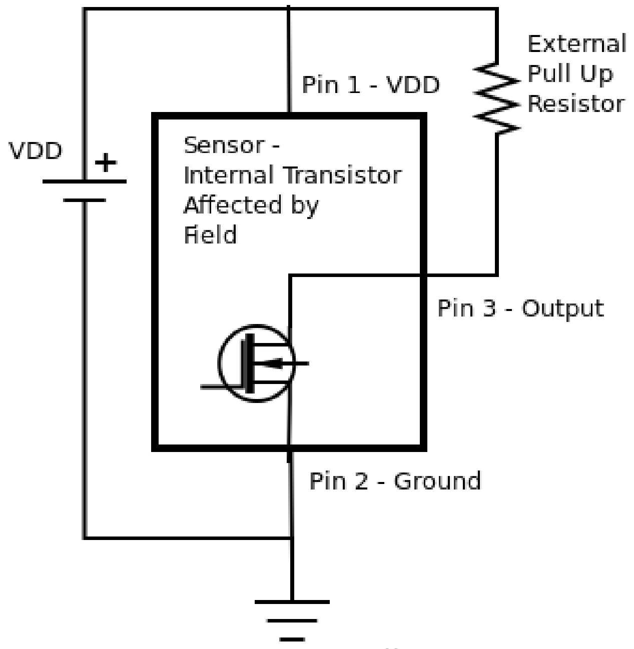
External Pull Up Resistor
Pin 1 - VDD
VDD
+
Sensor - Internal Transistor Affected by Field
Pin 3 - Output
Pin 2 - Ground

Illustration 1: Hall Effect Sensor

Ultrasonic Sensor

Just like many animals can use ultrasound to determine the placement of objects, these sensors use the reflections of sound to determine distance. The HC-SR04 is a low cost sensor that sends out an ultrasonic pulse and measures the return signal that bounces off an object [2]. By factoring in the speed of sound, distance can be determined fairly accurately. The sensor measures the distance directly ahead of the sensor, so if a panoramic perspective is needed, the sensor will need to be rotated.

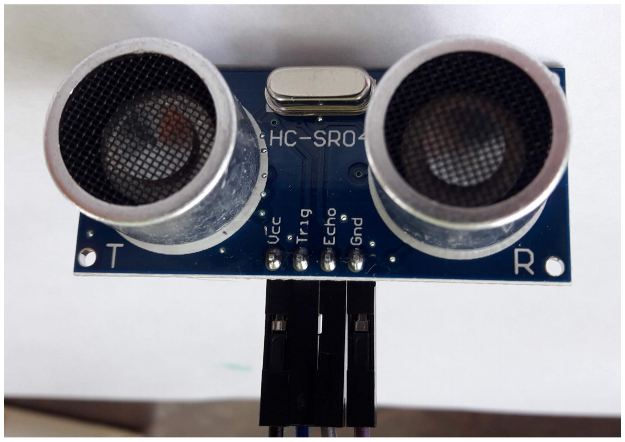

Illustration 2: HC-SR04

The sensor uses four wires, Ground, Vcc (5V), an output from the Pi to trigger the sensor to start a measurement and an input to the Pi that determines when the echo has been sensed.

MODULE 4: More Sensors

>>WARNING: 5V vs 3.3V Power <<
The sensor uses a 5V supply, which the Pi has available if it is supplied by USB or a 5V battery. However, the Pi itself runs with 3.3V power that is stepped down from 5V. For this reason, a voltage divider should be used as recommended in the various howtos, see the figure to the right. If 5V from the sensor is connected directly to the 3.3V pins of the Pi, the pins can be damaged irreparably.

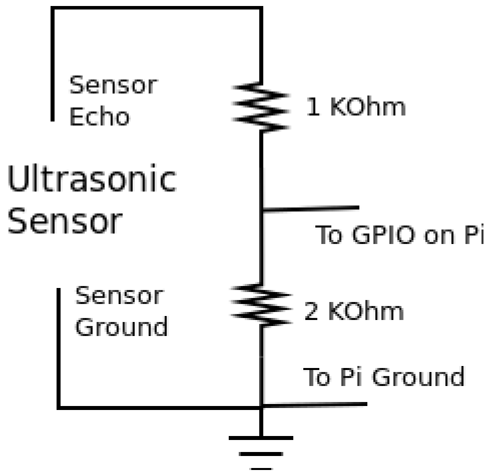

Illustration 3: Ultrasonic Sensor - Echo Pin

A modified version of the program provided on the last reference that can be used to interface the sensor with the Pi is found later in the module.

Accelerometer

Another very useful sensor is an accelerometer. Inexpensive accelerometers are also often called gyroscopes. They can be used to determine not only acceleration, but also the orientation of a given object. They are included in phones or tablets to determine their visual orientation. They are also used in drones to help with navigation and stability. These units measure angular velocity along the x, y and z axis and they also measure acceleration in the x, y and z axis. For this reason they are often referred to as 6 axis accelerometers.

One unit that pairs well with the Pi is the MPU-6050, which has its own onboard A/D converter and uses I2C communication to transmit information to the Pi. Just like the A/D converter used earlier used a serial protocol to transmit data to the Pi, this unit uses a similar protocol, I2C. I2C has the advantage of being able to be used with multiple devices over the same bus or set of pins. The Pi has dedicated pins that are designed to be used with the I2C protocol and many online tutorials exist for connecting the unit to the Pi [3].

>> HINT: Enable I2C <<
Make sure to use `raspi-confi` or another setup tool to enable the I2C communication.

The program provided on the link above provides a good example of how to read all six outputs and may be modified to read just one of the outputs.

Including Python Code

In a previous module, a function was used to convert the raw sensor output into meaningful information. As different sensors are added and different features are added to the code that is used on the Pi, the programs can get more and more complicated. In order to keep things relatively organized, it is useful to create well written functions and reuse them in multiple programs. That is what is happening when the `import` key word is used in a program. Up to now, though, the code being imported was from libraries written by other programmers.

MODULE 4: More Sensors

Reusing code and using imported functions has many advantages:

- It allows the programmer to easily include code that has been debugged
- If a bug is found, it can be changed in one place instead of each program it was pasted into
- Work can be better distributed as different programmers can more easily work on their modules independently

In order to understand how to break up code so that it can be modularized and then later imported, code that can be used for connecting the ultrasonic sensor to the Pi is provided below and then broken up into smaller functional portions.

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)      # Ignore warning for now
GPIO.setmode(GPIO.BOARD)     # Use physical pin numbering

TRIG = 13   # define input and output pins
ECHO = 22

# set up the input and output pins
GPIO.setup(TRIG, GPIO.OUT)
GPIO.output(TRIG, False)
GPIO.setup(ECHO, GPIO.IN)

time.sleep(.5)               # let the sensor initialize

# trigger a reading
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)

# find the start and end of the ultrasonic pulse
while GPIO.input(ECHO) == 0:
    start_time = time.time()
while GPIO.input(ECHO) == 1:
    end_time   = time.time()

# Speed of sound 34300 cm/sec
total_distance = (end_time - start_time) * 34300
# Divide by 2, account for return trip for signal
distance = round(total_distance/2, 1)
print ("Distance Away:",distance, "cm")

GPIO.cleanup()               # reset the GPIO
```

MODULE 4: More Sensors

These first few lines are repeated in many of the python programs used for the Pi (lines 1 thru 4). The first two lines import special modules that first assist with using the general purpose input output pins and then the time module to keep track of the time of the signal. Warnings may exist for various reasons, such as a different program currently is using the GPIO pins while this program also wants to use them. While in general it is not a good idea to ignore warnings, since this is likely to be the only program running on the Pi, it is acceptable and a bit less annoying to do so.

Lines 6 & 7 actually use variable names to label the GPIO pins to be used later. This is a good practice that helps comment the code and also makes it easier to change the pins later by just changing one spot in the code without having to find every reference to the actual pin number later.

Lines 10-14 configure the IO pins for the sensor and allow the communication pins to “settle” before taking measurements.

Lines 17-25 actually take a measurement with 28 & 30 performing the calculation required to get the distance.

Line 33 resets the GPIO pins. It is a good practice to run the cleanup at the end of each program.

Careful examination of the previous program shows that the program is broken up into parts that initialze the sensor and that read the sensor. Part of the initialization is fine in the main, such as lines 1-7. It is best not to hide the GPIO pin numbers inside of a function. The other part that initializes the sensor (lines 9-14), should only be run once when starting the program. Another part reads the output from the sensor and calculates the distance (lines 16-30) and that part could be run in a loop. So, it would seem logical to take portions of code that do specific things and move them into functions, especially if the code is easier to read and understand as a result.

Assuming two new functions were written, `ultrasonic_init`, that initialized the sensor and `ultrasonic_read`, that read the sensor, the resulting man program would be simplified to the following.

```
import RPi.GPIO as GPIO
from ultrasonic_func import ultrasonic_init as u_init
from ultrasonic_func import ultrasonic_read

TRIG = 13   # define input and output pins
ECHO = 22

u_init(TRIG, ECHO)

distance = ultrasonic_read(TRIG, ECHO)

print ("Distance Away:",distance, "cm")

GPIO.cleanup()
```

MODULE 4: More Sensors

The two functions were written in a separate file called ultrasonic_func.py and saved. Note the use of the `as u_init`, which allows the programmer to reference the function with an alternate name, which could help to make things even more clear. The following code would be the two functions in the separate file.

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)      # Ignore warning for now
GPIO.setmode(GPIO.BOARD)     # Use physical pin numberin

def ultrasonic_init(Trigger, Echo):
    # set up the input and output pins
    GPIO.setup(Trigger, GPIO.OUT)
    GPIO.output(Trigger, False)
    GPIO.setup(Echo, GPIO.IN)
    # let the sensor initialize
    time.sleep(.5)

def ultrasonic_read(Trigger, Echo):
    # trigger a reading
    GPIO.output(Trigger, True)
    time.sleep(0.00001)
    GPIO.output(Trigger, False)

    # find the start and end of the ultrasonic pulse
    while GPIO.input(Echo) == 0:
        start_time = time.time()
    while GPIO.input(Echo) == 1:
        end_time   = time.time()

    # Speed of sound 34300 cm/sec
    total_distance = (end_time - start_time) * 34300
    # Divide by 2, account for return trip for signal
    return round(total_distance/2, 1)
```

Moving Average

Values read from an A/D converter or other external sensor can often vary for a variety of reasons. For example voltage or current spikes elsewhere in the circuit can affect the readings from the sensors. The figure to the right shows actual data measurements in blue with a

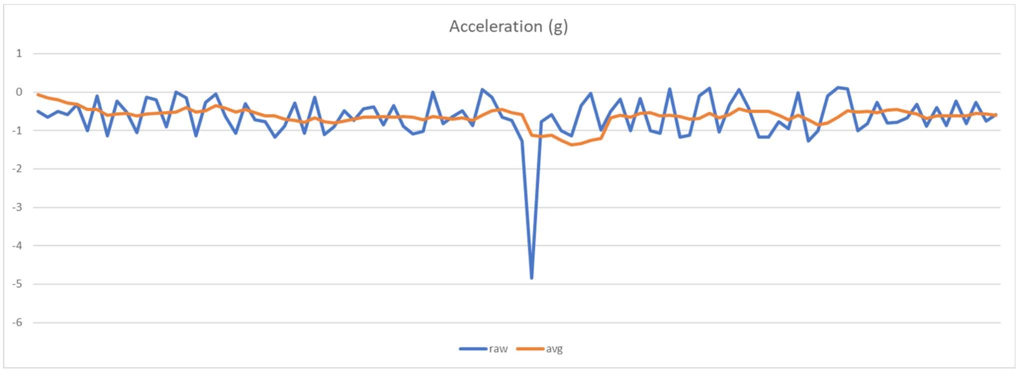

Illustration 4: Moving Average Example

four point moving average in orange that smooths out the actual readings. The idea is to deemphasize readings that might be considered as "noise". Noise mainly refers to data that is in error o not of interest because it is not reflective of the overall system being tracked. One way to reduce the noise in a system is to attempt to "filter" it out or remove it.

A moving average is one such technique that can be used to deemphasize faulty readings. A moving average just averages the last few readings and use the average as the actual reading. A formula for a moving average that averages the last 4 readings with y being the output reading and k being the time increment would be:

$$yAVG(k) = \frac{1}{4} [y(k-1) + y(k-2) + y(k-3) + y(k-4)]$$

The table below provides values for readings and moving average values.

Time in k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
y(k)	5	6	10	6	5	6	5	7
yAVG(k)	NA	NA	NA	6.75	6.75	6.75	5.5	5.75

Note that the formula needs to have enough readings for it to work properly. Also, the one reading of 10 above does pull up the overall estimate, but not significantly. It is important to understand the data that is measured so that values that are not reflective of the state of the system are identified, removed or somehow deemphasized before the data is used in making an informed decision. That decision could be how to steer a car, adjust the output of a production line or purchase a particular stock.

The data above corresponds to a four point moving average. The more points used for the moving average results in an estimated value that is "smoother" and generally less susceptible to "noise" or affects on the measurement that do not reflect its actual value. The less points used for the moving average, the quicker the estimated value will reflect changes in the measured value, but the more susceptible it is noise. Ultimately, the person designing the system, must balance these trade offs in the final system.

Homework - *Prelab* – Have this completed prior to showing up in the lab

1. Write a program that will use a Hall affect sensor as input and will light an LED when the sensor is activated. Include the schematic required to connect the sensor. Use a 10K Ohm resistor for the pull up resistor.
2. Consult the datasheet for the ultrasonic sensor and determine the following limits.
   a. How many samples could be measured in a second?
   b. The distance ranges in which it is recommended for use?
3. Write a program that will use the ultrasonic sensor to detect distance. You may start with the code provided in the module.
   a. Add a python function, `movingAverage` that will accept two arguments: an array (IN) and an int (n). The output will be the average of the first n values of the array. NOTE: I'm flexible on how you implement this part. It could be as described, or you can allow for `movingAverage` to accept an array, number of items to average, starting point in array to pick points, and max items in array so you know to cycle back around to the start if you reach the max in count.
   b. The program should read in 3 samples, use the function to compute the average and then output a value once every second for 10 seconds.
   c. The program should run through two 10 second loops and then stop.

4. Modify the program from the link for the accelerometer.
   a. Embed the register addresses used by the MPU_Init method in the method itself.
   b. Allow for user input of 1-6, read and display the value 1 – x acceleration, 2 – y acceleration, 3 – z acceleration, 4 – gyro x axis data, …. 7 – stop loop and end program, wait one second and then wait to read another input from the user.

Lab

Document the steps you took to perform the various tasks. Demonstrate the successful completion of each task to the TAs or instructor.

1. Use a Hall affect sensor and the program written for the prelab to demonstrate the use of the sensor.
   a. Make sure to have a detailed schematic, use a 10K resistor as the pull up resistor.
   b. Save the program as module4a.py.
2. Use the code written for the prelab to demonstrate the use of the ultrasonic sensor.
   a. Include a schematic showing how the pins were connected.
   b. Save the program used as module4b.py
3. Use the accelerometer and the code written and demonstrate your results
   a. Break the program into two parts, one with the functions and one with the main and import as necessary.
   b. Again, save a schematic showing the pin connections
   c. Save the program as module4c.py
4. Upload the code from the modules to Git.

Bibliography

[1] US5881 Datasheet – https://cdn-shop.adafruit.com/datasheets/US5881_rev007.pdf, Accessed 7/30/19

[2] Raspberry Pi Distance Sensor: How to setup the HC-SR04, https://pimylifeup.com/raspberry-pi-distance-sensor/, Accessed 7/30/19

[3] MPU6050 (Accelerometer+Gyroscope), https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi, Accessed 7/30/19