Introduction to Robotics with the Raspberry Pi

INTRODUCTION TO GPIO

Jeff Cicolani
Saturday, November 26, 2016

This workshop is assuming a you are using Windows. If you are using a Mac or Linux machine, good luck… I mean, you will need to look up the proper instructions for your OS. Sorry, I'm a Windows guy, now with a little Linux, but Windows is my got to OS for general use.

Introduction

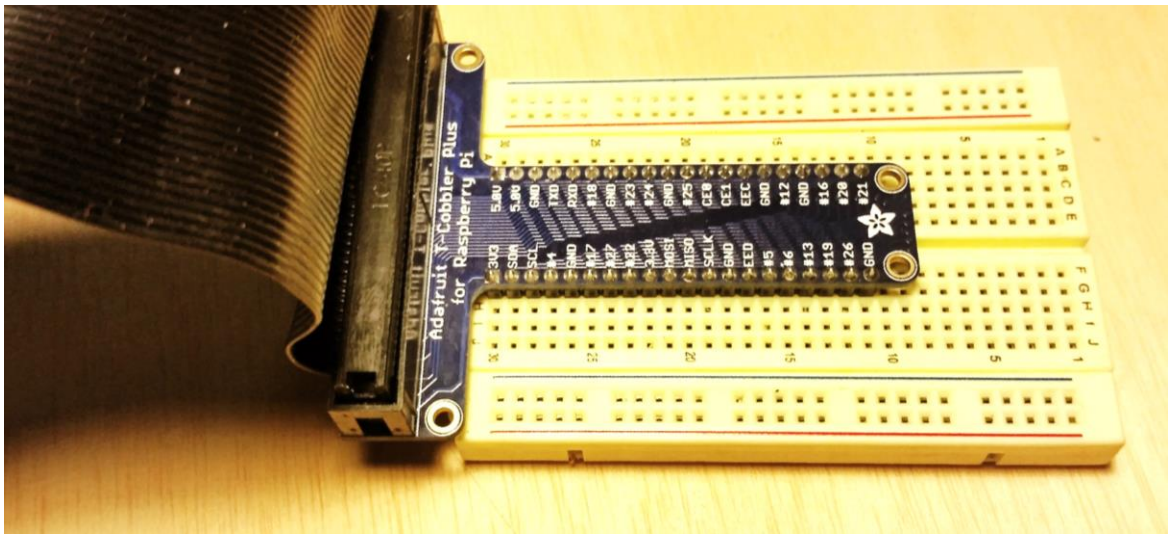
So far, we've installed the Raspberry Pi in preparation for developing. By now you should have headless access to your Pi. As we move forward from here, it is assumed you have accessed your Pi and are able to develop directly on it.

GPIO

GPIO stands for "General Purpose Input and Output." This is the interface between the electronics and the rest of the world. GPIO is how we will be connecting sensors and other devices to the Pi. The GPIO header is the double row of 40 pins running along one edge of the board.

There are several ways to connect the pins from this header to other devices. The motor controller hat we will be using is an example of a board that sits directly on top of the header. In Raspberry Pi terms, these boards are referred to as hats or plates. The pins can be connected to directly using jumpers. For many, this is a preferred method during prototyping. I prefer the third method, which is to use another board from Adafruit, called the "Pi Cobbler." There are a few versions the cobbler. For these workshops, I listed the Adafruit T-Cobbler Plus for Raspberry Pi. This board is designed to attach to a breadboard via a ribbon cable.

It uses a 40 pin header configured perpendicular to the pins that plug into the breadboard. This moves the ribbon cable attachment off of the breadboard and allows better access to the holes.



One of the advantages of using the cobbler is the pin break-outs are clearly marked. When we start building our circuits it will be very easy to see exactly what you are hooking up. This also makes it easier to identify which pins are being used for you code. When you declare pin 21 as an output pin, you will know exactly which pin that is on the board.

Limitations

There are a couple things you will need to keep in mind as you are working with GPIO.

First, the Raspberry Pi, as we have it set up, is not a real-time device. Debian Linux is a full operating system with layers of abstraction from the hardware. Python operates in, yet another, abstraction layers. Each of these layers introduces a certain degree of lag. It's generally nor perceivable to us, but it can make a huge difference in robot operations. There are distributions of Debian that are much more real-time, but we're not using one of those.

Second, there is no analog input on the Pi. Well, there's one, but it shared with the serial port. The serial port we will likely be using for something else later. So, it's just better to accept that there are no analog input pins.

Third, the Pi only as a single PWM capable pin. This means there is only one pin on the header that can simulate an analog output.

The good news is there's a simple solution to all of these issues. That is simply to introduce an external microcontroller that is real-time and has analog input. We will be doing this in a later workshop with the Arduino. The Arduino is, basically, a prototyping board for the AVR AT series of microcontrollers. These chips are directly connected to the hardware and do not have the layers of abstraction you find in most SoC (System on a Chip) processors like those on the Pi.

Accessing GPIO with Python

To access the GPIO pins and do something useful, we'll be using the RPi.GPIO library. The GPIO library provides objects and functions to control the GPIO pins. Raspberry Pis, such as the B, 2, and 3, come with the GPIO library installed, so it should be ready to go. For more information on how to use the package, visit https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/.

In order to use the GPIO library we will need to do two things; import the package and then tell it which mode we'll be using to access the pins. There are two modes, board and BCM, and they, essentially tell the system which numbering reference we will be using.

Board mode references the numbering on the P1 header of the Raspberry Pi. Since this numbering remains constant, for backwards compatibility, you won't need to change your pin numbering in your code based on the board revision.

In contrast, BCM mode refers to the pin numbering from the Broadcom SoC, which means, on newer versions of the Pi it is possible for the pin layout to change. Fortunately, this pin layout has not changed between the BCM2836, used in the Pi 2, and the BCM2837 used in the Pi3.

For the purposes of this, and future workshops, we'll be using BCM mode, simply because that is what is illustrated on the T-Cobbler.

Every program using the GPIO header will include the following two lines of code:

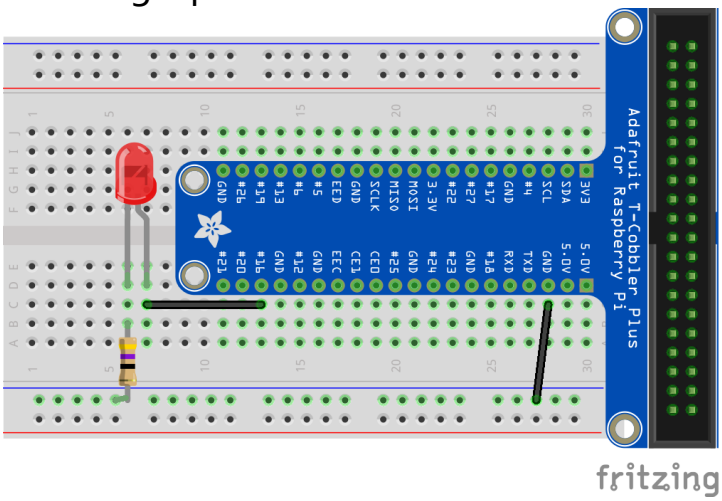
```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
```

Simple Output

LED Example

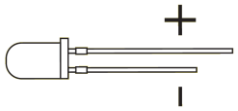
The simplest example is the ubiquitous hardware version of "Hello World." The blinking LED. Our first project in GPIO is connecting an LED to Pi and using a Python script to make the LED blink. This tutorial is a paraphrased version of the tutorial provided by Third Eye Visions at http://www.thirdeyevis.com/pi-page-2.php.

Hooking Up the Circuit



This is a very simple circuit to start off with. It uses an LED, a 220Ω resistor, and some 2 male to male jumpers.

1. Connect the 220Ω resistor between the ground rail and a blank rail.
2. Connect the cathode of the LED to the same rail as the resistor. The cathode is the pin closest to the flat side of the LED.



3. Connect the anode of the LED to another empty rail.
4. Connect a jumper from anode's rail to rail connected to pin 16.
5. Connect a jumper from the ground rail and any rail connected to a ground pin of the T-Cobbler.

If you wanted to test the LED before moving on to the code, you can move the jumper from pin 16 to one of the 3.3v pins. If your Pi is powered on, the LED will illuminate. Make sure you move the jumper back to pin 16 before continuing.

Writing the Code

The code for this project is very simple. This is written in Python 3.x. One of the lines will not work in Python 2.7. Specifically, the print line at the end uses the `end` parameter, which is not compatible. If you are using Python 2.7 you will need to omit this parameter.

The `end` parameter replaces the default "/n" that appended to each line, with a "/r". The "/r" is a carriage return as opposed to the new line represented by "/n". This means the cursor will return to beginning of the current line and overwrite it.

GPIO access system level memory and has to be run with super user or root access. This means you will need to run Python with sudo or grant yourself permanent root permissions, which is dangerous. What we'll be doing instead, is running the file from the terminal.

1. Create a new Python3 file by either;
    a. With Idle
        i. Open Idle for Python3
        ii. Click new
        iii. Save the file as “gpio_led.py” in your project folder.

    or

    b. In a terminal window
        i. Navigate to your project folder. On my Pi it would be

```
$ cd ~/TRG-RasPi-Robot/code
```

        ii. Type `touch gpio_led.py`
        iii. Type: `idle3 gpio_led.py`
        This will open the empty file in the Idle IDE for Python3.
2. Enter the following code:

```
# GPIO example blinking LED

# Import the GPIO and time libraries
import RPi.GPIO as GPIO
import time

# Set the GPIO mode to BCM and disable warnings
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Define pins
led = 16

GPIO.setup(led,GPIO.OUT)

# Make sure LED is off
GPIO.output(led,False)

# Begin Loop
while True:

    # Turn LED on
    GPIO.output(led,True)

    # Wait 1 second
    time.sleep(1)

    # Turn LED off
    GPIO.output(led,False)

    # Wait 1 second
    time.sleep(1)
```

3. Open a new terminal window and navigate to your project folder

4. Type: `chmod +x gpio_led.py`
   This will make the file executable
5. To run the code type: `sudo python3 gpio_led.py`

There you have it; a blinking LED. Hello world.

Pulse Width Modulation (PWM)

Even though there is only one PWM pin on the Pi's GPIO header, it is useful to know how to control it properly. The sole PWM pin on the board is pin 18. For this example, we'll be setting up the LED to use this pin and pulse the LED.

Hooking Up the Circuit

Alright, this is the complicated part. To set up this circuit you will need to follow these directions very closely. Using the circuit we built for the LED exercise;

i. Move the jumper from pin 16 to pin 18.

Phew. Now that we've gotten through all of that, let's code.

Writing the Code

1. Create a new Python3 file by either;
   a. With Idle
      i. Open Idle for Python3
      ii. Click new
      iii. Save the file as "gpio_pwm_led.py" in your project folder.

   or

   b. In a terminal window
      i. Navigate to your project folder. On my Pi it would be

```
$ cd ~/TRG-RasPi-Robot/code
```

      ii. Type `touch gpio_pwm_led.py`
      iii. Type: `idle3 gpio_pwm_led.py`
      This will open the empty file in the Idle IDE for Python3.
2. Enter the following code:

```
# GPIO example blinking LED

# Import the GPIO and time libraries
import RPi.GPIO as GPIO
import time

# Set the GPIO mode to BCM and disable warnings
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

```
# Define pins
pwmPin = 18

GPIO.setup(pwmPin,GPIO.OUT)
pwm = GPIO.PWM(pwmPin,100)

# Make sure LED is off
pwm.start(0)

# Begin Loop
while True:

    count = 1
    # begin while loop to brighten LED
    while count < 100:

        # set duty cycle
        pwm.ChangeDutyCycle(count)

        # delay 1/100 of a second
        time.sleep(0.01)

        # increment count
        count = count + 1

    # begin while loop to darken LED
    while count > 1:

        pwm.ChangeDutyCycle(count)

        time.sleep(0.01)

        # set duty cycle
        pwm.ChangeDutyCycle(count)

        # delay 1/100 of a second
        time.sleep(0.01)

        # decrement count
        count = count - 1
```

6. Open a new terminal window and navigate to your project folder
7. Type: `chmod +x gpio_pwm_led.py`
   This will make the file executable
8. To run the code type: `sudo python3 gpio_pwm_led.py`

Simple Input

Sonic Rangefinder Example

Now that we've seen how easy it is to send a signal out, it's time to get some information back into the Pi. For this example, we'll be using the HC-SR04 ultrasonic sensor to determine the distance to an object. We will

put the call into a loop that will allow us to get constant distance readings. You'll be using the libraries used in the previous example to access the GPIO pins.

This exercise will also introduce you to one of the key factors we have to watch out for with the Pi and many other devices. That is, voltage difference between the sensors and the pins. Many sensors are designed to work at 5v. The Pi, however, uses 3.3v in its logic. That means all of the I/O pins are designed to work with 3.3v. Applying a 5v signal to any of these pins can cause severe damage to your Pi. The Pi does provide a couple of 5v source pins. But we will need to reduce the returning signal to 3.3v.

This part of the workshop is, pretty much, a wholesale copy of the ultrasonic tutorial from ModMyPi.com. You can read the original article here: https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi.

Hooking Up the Circuit

This time the circuit is a bit more complicated. Really. One thing we have to keep in mind is the sensor works at 5 volts. The Pi's GPIO pins work on 3.3 volts. Feeding a 5v return signal into a 3.3v pin can damage the Pi. So, to keep that from happening we will add a simple voltage divider to the echo pin.

Let's do some math.

$$Vout = Vin \times \frac{R2}{R1 + R2}$$

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2}$$

We have 5v in and want 3.3v out and are using a 1kΩ resistor as part of the circuit. So…

$$\frac{3.3}{5} = \frac{R2}{1000 + R2}$$

$$0.66 = \frac{R2}{1000 + R2}$$

$$0.66(1000 + R2) = R2$$

$$660 + 0.66R2 = R2$$

$$660 = 0.34R2$$

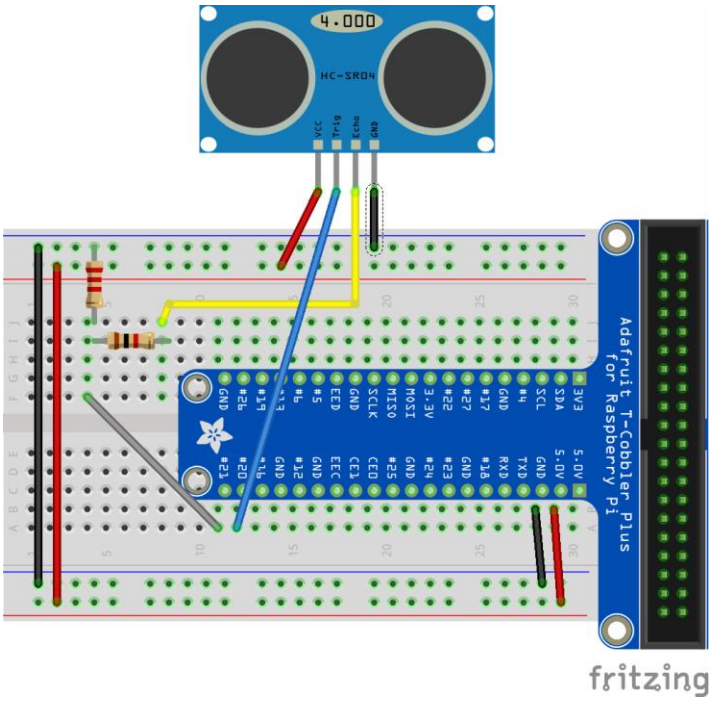
$$1941 = R2$$

Our parts list then is:

- HC-SR04
- 1kΩ resistor

- 2kΩ resistor. You can use 2 1kΩ resistors in series, or a more popular, similar, resistor is the 2.2kΩ. That's what we'll use.
- 4 male to female jumpers
- 4 male to male jumpers

Here's the set-up:



1. Make sure the ground jumper is secure between the gnd pin and the ground rail.
2. Add a jumper between one of the 5.0v pins and the power rail.
3. Using a male to female jumper, connect the gnd pin on the SR04 to the ground rail.
4. Connect the vcc or 5v pin from the SR04 to the power rail.
5. Connect the trig pin on the SR04 to pin 20 of the T-Cobbler.
6. Connect the 2kΩ resistor from an empty rail to the ground rail.
7. Connect the 1kΩ resistor from the 2kΩ rail to another empty rail.
8. Connect a jumper between the 2kΩ rail and pin 21 on the T-Cobbler.
9. Connect the echo pin of the SR04 to the rail the other end of the 1kΩ resistor is connected to.

That completes the wiring. Now let's get the code set up.

Writing the Code

The HC-SR04 ultrasonic rangefinder works by measuring the time it takes for an ultrasonic pulse to return to the sensor. We'll be sending out a 10-microsecond pulse and then listening for the pulse to return. By measuring the length of the returned pulse, we can use a little math to calculate the distance in centimeters.

Distance is calculated as speed x time. It's derived from the formula speed = distance / time. We know that, at sea level, sound travels at a rate of 343m per second, or, for out purposes, 34,300cm per second. Since we are actually measuring the time it takes for the pulse to reach its target and return, we really only want half of that value. So we'll be working with the formula:

Distance = 17150 x time

So, this code simply sends out a 10µS pulse, measures the time it takes to return, calculates the estimated distance in centimeters, and displays it in the terminal window.

1. Create a new Python3 file by either;
    a. With Idle
        i. Open Idle for Python3
        ii. Click new
        iii. Save the file as "gpio_sr04.py" in your project folder.

    or

    b. In a terminal window
        i. Navigate to your project folder. On my Pi it would be

```
$ cd ~/TRG-RasPi-Robot/code
```

        ii. Type `touch gpio_sr04.py`
        iii. Type: `idle3 gpio_sr04.py`
        This will open the empty file in the Idle IDE for Python3.
2. Enter the following code:

```
# GPIO example using an NC-SR04 ultrasonic range finder

# import the GPIO and time libraries
import RPi.GPIO as GPIO
import time

# Set the GPIO mode to BCM mode and disable warnings
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Define pins
trig = 20
echo = 21

GPIO.setup(trig,GPIO.OUT)
GPIO.setup(echo,GPIO.IN)

print("Distance measurement in progress")

# Begin while loop
while True:
    # Set trigger pin low got 1/10 second
```

```
GPIO.output(trig,False)
time.sleep(0.1)

# Send a 10uS pulse
GPIO.output(trig,True)
time.sleep(0.00001)
GPIO.output(trig,False)

# Get the start and end times of the return pulse
while GPIO.input(echo)==0:
    pulse_start = time.time()

while GPIO.input(echo)==1:
    pulse_end = time.time()

pulse_duration = pulse_end - pulse_start

# Calculate the distance in centimeters
distance = pulse_duration * 17150
distance = round(distance, 2)

# Display the results. end = '\r' forces the output to the same line
print("Distance: " + str(distance) + "cm        ", end = '\r')
```

9. Open a new terminal window and navigate to your project folder
10. Type: `chmod +x gpio_sr04.py`
    This will make the file executable
11. To run the code type: `sudo python3 gpio_sr04.py`

Explore

Normally we would want to do some cleanup before exiting the program. The cleanup, essentially, closes out the GPIO ports and makes them available for the next thing. To do this we would add the `GPIO.cleanup()` function when we exit. Since we are running our program in an open loop and are exiting using `ctrl-c`, the program would never run this code. So, we simply don't include it.

The GPIO header offers other pins, obviously. Take time to research and experiment. We'll be using the serial I2C pins when we work with the motor controller. We didn't work with switches, though that can easily be done with what learned in this workshop.

Python offers some interesting ways to work with GPIO. There are things we simply didn't cover. For instance, pins can be assigned, referenced, and controlled with lists and tuples.

In the next workshop, we'll be overcoming the limitations of the Raspberry Pi by adding an Arduino. We'll program the Arduino directly from the Pi and use serial commands to communicate through the USB port.