Introduction to Robotics with the Raspberry Pi

RASPBERRY PI & ARDUINO BASICS

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

In the previous workshop, we used the GPIO pins on the Raspberry Pi to interact with an LED and an ultrasonic sensor. We also discussed some of the shortcomings of Raspberry Pi GPIO. In particular, we talked about the lack of sufficient analog and pulse width modulation (PWM) pins.

Raspberry Pi's GPIO Weaknesses in Review

Real Time or Near Real Time Processing

Real time processing is the system's ability to interact directly with GPIO and external devices. It is crucial for CNC applications or other applications where immediate response is required. In robotics terms, it is necessary for closed loop systems where an immediate response to stimuli is required.

A good example would be an edge detector for a mobile robot. You would want the robot to stop moving should it be driving itself over a cliff or off the edge of a table. Taking the time to process through the many abstraction layers of an operating system to reach the logic to determine to stop, and then sending the signal through the many layers, again, to the motor controller, could prove catastrophic. And, if the OS were to delay the operation or hang, the robot will happily plummet to its demise, never the wiser. Instead, you want your robot to stop immediately.

Whereas there are flavors of Linux that does facilitate near-real-time processing, these are specialty operating systems and the Raspbian installation we are using is not one of them.

Analog Input

We have seen digital input working on the Pi. In fact, we used the ultrasonic rangefinder to detect range by detecting when a digital pin turned on and then off (go high then low). With a little math, we were then able to convert that signal into useful data. That was digital; simply detecting when a pin had a high voltage, and then when the same pin had a low voltage.

Analog signals comprise of a varied signal. Not just high or low, white or black, on or off, but a range of values or shades of gray, using the white and black analogy. This is very useful when you are using a sensor that measures intensity or the level of something. An example would be a light sensor which uses a photo-resistor. As the light intensity changes, so does the resistance, and therefore the voltage, on the sensor. A device, called an analog to digital converter, transforms that analog signal into a digital value the program can use.

The Raspberry Pi has a single analog pin. This is not very useful, especially when it's tied to another function that the board will likely be using. In this case, serial communication.

Analog Output (A.K.A., PWM)

Analog output is similar, in nature, to analog input. With the LED exercise, we did earlier, we used a digital signal to turn an LED on and off. Analog would allow us to change the brightness, or intensity of the LED. However, a digital system, such as a computer or microprocessor, cannot create a true analog signal.

What it can do is adjust the frequency and duration of the digital signal. The duration of a digital signal is referred to as a pulse. Adjusting the how often a pulse is active in a given time period, and the length of that pulse, is called Pulse Width Modulation, or PWM. When we were measuring the signal from the ultrasonic rangefinder, we were actually measuring the pulse returned from the device.

The Raspberry Pi has two PWM pins available. And, again, this is not as useful as we would like.

Arduino to the Rescue

Fortunately, there is a class of devices designed, specifically, to manage input and output of various types, in real time; microprocessors. There are many types of microprocessors out there. Some of the more common, and easy to use, are the AVR ATTiny and the ATMega processors.

However, these are chips. And, unless you're used to working with them, can be difficult to access and use. In order to make these devices easier to use, the manufacturers will create what are known as development boards. These boards connect the pins on the chip to headers that are easier to access for prototyping. They will also add the electronics needed to use the pins, such as voltage regulators, pull up resistors, filter caps, diodes, etc. So, in the end, all you have to do is connect your specific electronics the device and prototype your product.

A few years back, a group of engineers in Italy got together and did something a little unprecedented. They developed their own development board around the AVR ATMega chip, made the design open to the public (open hardware), and then marketed it to hobbyists and students. And they called this board, "Arduino". This

had the completely intended consequence, I'm sure, of becoming a defacto standard in the hobby and Maker movement.

We will be using an Arduino Uno with our Raspberry Pi. Why, you may ask? First off, it is a real-time processor. The Arduino is communicating directly with the pins and attached peripherals. There is no lag due to OS or program layer abstraction. Second, it provides a lot more pins to work with. Among them, we are adding six analog pins and six hardware based PWM pins. I say hardware based because, due to the fact that the board is real-time, we can simulate PWM signals through software, on any of the pins. Of which there are 20, by the way.

And that is just the Arduino Uno. There is a larger version of the Arduino board called the Mega. The Mega has 54 digital pins and 16 analog pins. That is a total of 70 pins of IO goodness.

Connecting an Arduino

When I had originally outlined this workshop, the intent was to provide multiple ways to connect the Arduino to the Raspberry Pi. However, I quickly discovered to use anything but the USB port was going to introduce another layer of complexity I am not prepared to work through. It essentially involves telling the Pi you are activating the UART pins and then disabling a number of native functions that use this channel. So, we will be using the USB connection and focusing on an introduction to using Arduino as it relates to the Pi.

Before we connect the Arduino to our Raspberry Pi, we'll want to install the software and drivers. Fortunately, this is super easy.

Installing the Arduino IDE

Open a terminal window

1. Type

```
sudo apt-get install Arduino
```

2. Answer yes to any prompts.
3. Grab a drink, this may take a surprisingly long time.

When the installation process is done, it will add the Arduino IDE to your programming menu.

Connecting to your Arduino

Installing the Arduino IDE will also install any drivers that are needed to work with the Pi. Now all we have to do is connect the USB cable from the Raspberry Pi to the Arduino. Depending on the manufacturer of the board, you may need a different USB cable. Since I am using the Redboard from Sparkfun, I'll be using a USB Mini cable. Some use an A to B cable, and others use a USB Micro.

But, that's it. Next we're going to test your Arduino with the ubiquitous blink program.

Programming

Sketches

Programs for the Arduino are called sketches. The idea being, you are simply sketching code, like you would an idea on a restaurant napkin. And, in all honesty, it does feel that way sometimes.

You will be writing Arduino sketches in a thin version of C made for the Arduino board. It is essentially a reduced set of instructions that the AVR processor will be able to run. Like Python, you can add libraries as you add functionality and complexity. We'll see that in a little while when we do some communication between the two boards.

Hello Arduino

Like with the GPIO workshop, the first thing we're going to do is blink an LED. However, since there is an LED already conveniently added to the board, we won't need to break out the breadboard just yet.

1. Open the Arduino IDE from your programming menu
2. Verify the board is connected and detected
    a. On the Arduino IDE menu, go to Tools and hover over Board
    You should see "Arduino Uno" selected
    b. Now hover over Serial Port
    It should say something like "/dev/ttyUSB0". Yours may be different if your Pi assigned a different port. The point is, there's something there and it's checked.
    c. Close the Tools menu by clicking somewhere outside the menu
3. Enter the following code:

```
int led = 13;

void setup() {
        pinMode(led, OUTPUT);
}

void loop() {
        digitalWrite(led, HIGH);
        delay(1000);
        digitalWrite(led, LOW);
        delay(1000);
}
```

4. Save the file as "blink_test"
5. Click on the check box icon to compile the sketch.
    If you get any errors, make sure you've entered to code correctly. Remember, unlike Python, you have to end each line with a semi colon. And, like Python, capitalization matters.
6. When everything compiles correctly, click on the arrow icon (pointing right). This will upload the sketch to the Arduino.

Wait a couple seconds while it uploads, but then you should see the LED connected to pin 13 blinking.

Congratulations, you just finished your first Arduino program. In addition, you did it from your Pi.

Pingduino

Now, let's jump into it. We are going to setup our HC-SR04 ultrasonic range finder on the Arduino and send the distance information back to the Pi as a serial string. In order to do this, we need to open up a serial connection between the two boards. Then the Arduino will trigger the sensor and, like in our previous workshop, read the pulse returned. We'll calculate the distance and then send the results to the Pi.

On the Pi side, we'll simply have a program that listens to the serial port and then prints whatever it reads from the Arduino.

Setting up the Circuit

Setting up the circuit couldn't be easier. In fact, we'll not even use the breadboard. We're going to connect the sensor directly to the Arduino headers.

1. Connect VCC to the 5v pin on the Arduino
2. Connect GND to one of the GND pins on the Arduino. Doesn't matter which one, but there are two adjacent to the 5v pin.
3. Connect TRIG to pin 7 on the Arduino
4. Connect ECHO to pin 8 on the Arduino

The Code

We will need to write code for both boards in order for this to work.

Arduino

1. Open a new sketch window and save it as "serial_test"
2. Enter the following code:

```
int trig = 7;
int echo = 8;
int duration = 0;
int distance = 0;

void setup() {
        Serial.begin(9600);
        pinMode(trig, OUTPUT);
        pinMode(echo, INPUT);

        digitalWrite(trig,LOW);
{

void loop() {
        digitalWrite(trig, HIGH);
        delayMicroseconds(10);
```

```
        digitalWrite(trif, LOW);

        duration = pulseIn(echo, HIGH);
        distance = duration/58.2;

        Serial.write(distance);

        delay(500);
}
```

3. Save the file and compile it (click the check mark icon)
4. When it compiles correctly, upload the sketch to the Arduino.

Raspberry Pi

1. Open a new IDLE file and save it as "serial_test.py"
2. Enter the following code:

```
import serial

ser = serial.Serial('/dev'ttyAMC0', 9600)

while 1:
        # we need to conver the byte from the Arduino into an int
        distance = int.from_bytes(ser.read(1), byteorder='Big', signed =
'False')

        print("Distance: " + str(distance) + "cm       ", end = '\r')
```

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

Explore

Due to time constraints on my side, this workshop is not nearly as comprehensive as I had intended. There is still a lot to look at. For instance, we are using the Serial.write() command to send a single byte to the Pi. On the Pi's side we are reading that single byte and converting it to an unsigned integer. But that's not nearly precise enough for most operations. So, how would we pass a decimal or float value? We could do it as a string using Serial.println() on the Arduino and ser.readline() on the Pi. But that is passing the data as a text string and has its own complications.

We also did not touch on analog input on the Arduino or sending data the other direction from the Pi to the Arduino. So, there's a lot left to explore there and we will likely come back to this in the near future.

However, I know we are all anxious to make things move. So, the next workshop will be using the motor driver that we assembled at the soldering workshop to spin your motors. Once you've done that, you're pretty much ready to start building a robot.