GEAR has in its inventory one Logitech gamepad model F310. The gamepad has a number of control buttons and two joysticks and can be connected to one of the USB ports of the Raspberry Pi. This lesson covers methods for reading the gamepad output signals in a Python 2 program.

Linux operating systems (Raspbian for example) create device files for USB-connected devices. Let us examine these files. Open the Terminal on the RPi and enter this command:

```
ls /dev/input
```

When I enter the above command on the GEAR1 RPi, I get the following listing (you should have similar):

```
by-id by-path event0 event1 event2 mice mouse0
```

The first two items on the list are directories the others are event files. Now issue this command:

```
ls /dev/input/by-id
```

I get this output in the terminal:

```
usb-Logitech_Lenovo_USB_Optical_Mouse-event-mouse
usb-Logitech_Lenovo_USB_Optical_Mouse-mouse
usb-NOVATEK_USB_NETVISTA_FULL_WIDTH_KEYBOARD-event-if01
usb-NOVATEK_USB_NETVISTA_FULL_WIDTH_KEYBOARD-event-kbd
```

From the above we can see that the event files are associated with the mouse and keyboard connected to the RPi by USB ports.

Now connect the gamepad to the RPi, wait a few seconds, and issue the command again.

```
ls /dev/input
```

I get this:

```
by-id by-path event0 event1 event2 event3 js0 mice mouse0
```

Notice that there are now two additional items: **event3** and **js0.** These are the items associated with the gamepad. In our case we are interested in the **event3** file, not js0. **Write down the name of the event file that you get after connecting the gamepad. You will need this later in the lesson**. From this point on, do not disconnect the gamepad from the RPi. If you do, you might get a different event file assigned upon renewing a connection. This could happen if you connect another USB device and reboot the RPi.

Now issue this command:

```
ls /dev/input/by-id
```

My output is this:

```
usb-Logitech_Lenovo_USB_Optical_Mouse-event-mouse
usb-Logitech_Lenovo_USB_Optical_Mouse-mouse
usb-Logitech_Logitech_Dual_Action_C970A499-event-joystick
usb-Logitech_Logitech_Dual_Action_C970A499-joystick
usb-NOVATEK_USB_NETVISTA_FULL_WIDTH_KEYBOARD-event-if01
usb-NOVATEK_USB_NETVISTA_FULL_WIDTH_KEYBOARD-event-kbd
```

Now we see the name for our connected joystick device.

Now try this command:

```
cat /dev/input/event3
```

If the name of your event file for the gamepad is not event3, then substitute it in the above command. Press a button on the controller to see what happens. When I pressed the red B button I got this output (I provide only the first three lines of output):

```
7��Z�v
   ──────────────────────────────�7��Z�v
       ──────────────────────────────7��Z�v
```

Well, that is not a set of characters that we can readily recognize. However, the key is that when a button on the gamepad is pressed, characters of some kind can be read from the event file. This seems promising. We need some help in interpreting the output of the gamepad and will find it in the **evdev** package, which works with Python.

Install evdev:

```
sudo pip install evdev
```

After you have installed evdev, create a new file using Python 2. Enter the code below:

```
from evdev import InputDevice, categorize, ecodes

gamepad = InputDevice('/dev/input/event3')
#if your event is not event3, then change to yours

print (gamepad)
```

Save and run the program. Your output should be similar to mine:

```
device /dev/input/event3, name "Logitech Logitech Dual Action", phys "usb-3f980000.usb-1.4/input0"
```

From the above we see that the name of the device connected to event3 is "Logitech Logitech Dual Action." Now let us see if we can translate some output from the gamepad. Edit your Python file to this:

```
from evdev import InputDevice, categorize, ecodes

gamepad = InputDevice('/dev/input/event3')

for event in gamepad.read_loop():

    print (event)
```

When I run the above program and press the red B button on the gamepad, I get the output listed on the next page.

```
event at 1518534972.266917, code 04, type 04, val 589827
event at 1518534972.266917, code 290, type 01, val 01
event at 1518534972.266917, code 00, type 00, val 00
event at 1518534972.422922, code 04, type 04, val 589827
event at 1518534972.422922, code 290, type 01, val 00
event at 1518534972.422922, code 00, type 00, val 00
```

Looks like we are getting closer to something we can use. It appears that just a single button push creates six events. Edit your Python file to match the code below:

```
from evdev import InputDevice, categorize, ecodes

gamepad = InputDevice('/dev/input/event3')

for event in gamepad.read_loop():

   if event.type == ecodes.EV_KEY:

      print (categorize(event))

   else:

     print ('Not a button')
```

When I run the above program and press the red B button, I get the output below:

```
Not a button
key event at 1518536141.992223, 290 (BTN_THUMB2), down
Not a button
Not a button
key event at 1518536142.132234, 290 (BTN_THUMB2), up
Not a button
```

From the above output we learn that only two of the six events are an event.type of ecodes.EV_KEY (i.e. a button event). The first event is the recording of the button pressed down and the second is the button released (up). Another important piece of information is that the button is named **BTN_THUMB2**.

If we press all four of the colored buttons on the gamepad, we discover these names: yellow Y = **BTN_TOP**; green A = **BTN_THUMB**; red B = **BTN_THUMB2**; blue X = **BTN_JOYSTICK, BTN_TRIGGER**.

Now you will write code that is close to controlling a robot with the gamepad. The code is on the next page. Notice that we are adding a method named KeyEvent from evdev.

```
from evdev import InputDevice, categorize, ecodes, KeyEvent

gamepad = InputDevice('/dev/input/event3')

for event in gamepad.read_loop():
    if event.type == ecodes.EV_KEY:
        keyevent = categorize(event)
        if keyevent.keystate == KeyEvent.key_down:
            if keyevent.keycode == 'BTN_THUMB':
                print "Back"
            elif keyevent.keycode == 'BTN_TOP':
                print "Forward"

            elif keyevent.keycode == 'BTN_THUMB2':
                print "Right"
            elif keyevent.keycode == ['BTN_JOYSTICK', 'BTN_TRIGGER']:
                print "Left"
```

Run the above program and press each of the colored buttons. The program will respond by printing a word indicating a direction of movement (forward, back, right, left). Now we have a control structure. All that is needed is to substitute some drive code for a robot for the print commands.

You may also wish to use other buttons on the gamepad for various control functions. Just use the instructions provided earlier to find the button names. Then you can add them to the above program.

There are three additional controls on the gamepad that are not buttons. These are: D-pad (on left side of controller) and two joysticks. All of these can be used for directional control of a robot. In addition, the joysticks can be used to add speed control to the direction. For this lesson we will cover the joysticks (you can explore the D-pad on your own if you wish).

Let us alter the Python program so that we can discover the names of the joysticks and work with the values output with various positions of the stick. Copy the code on the next page.

```
from evdev import InputDevice, categorize, ecodes
gamepad = InputDevice('/dev/input/event3')
for event in gamepad.read_loop():
   if event.type == ecodes.EV_ABS:
      print (categorize(event))
   else:
     print ('Not a joystick or D-pad')
```

Run the above code. Move each joystick forward and backward, left to right. You can also press the four buttons of the D-pad if you like. In the output you should notice these names:

For right joystick, forward/back motion is named **ABS_RZ**, left/right motion is named **ABS_Z**

For left joystick, forward/back motion is named **ABS_Y**, left/right motion is named **ABS_X**

For D-pad, forward/back buttons named **HAT0Y**, left/right buttons named **HAT0X**

Your next program will print off the direction the joystick is pointed and the value. The value is an eight-bit number (it can vary from 0 to 255). When the joystick is in its central resting position, the values for ABS_RZ and ABS_Z will be 127 or 128, the midpoint of the range. When the joystick is full forward the ABS_RZ value is 0 and full backward is 255. When full to left the ABS_Z value is 0 and full right the value is 255. Create a Python program with the code on the next page.

```
 from evdev import InputDevice, categorize, ecodes, KeyEvent
gamepad = InputDevice('/dev/input/event3')
for event in gamepad.read_loop():
        if event.type == ecodes.EV_ABS:
            absevent = categorize(event)
            if ecodes.bytype[absevent.event.type][absevent.event.code] == 'ABS_RZ':
                if absevent.event.value > 128:
                    print 'reverse'
                    print absevent.event.value
                elif absevent.event.value < 127:
                    print 'forward'
                    print absevent.event.value
            if ecodes.bytype[absevent.event.type][absevent.event.code] == 'ABS_Z':
                if absevent.event.value > 128 :
                    print 'right'
                    print absevent.event.value
                elif absevent.event.value < 127:
                    print 'left'
                    print absevent.event.value
```

The output of the above program is provided in the table below (it is not output in a table, I just put it in a table to make it more compact). Here is what I did: **1)** move the right joystick full forward quickly and release right away **2)** move right joystick full backward quickly and release **3)** move joystick full to right and release **4)** move joystick full to left and release. The values in the table confirm the range of numbers. For example, in the forward column the first value is 122, close to the central stick position value of 127 or 128. Notice that the value drops first to 0 (full forward value) and that the last value is 121, back to near central value.

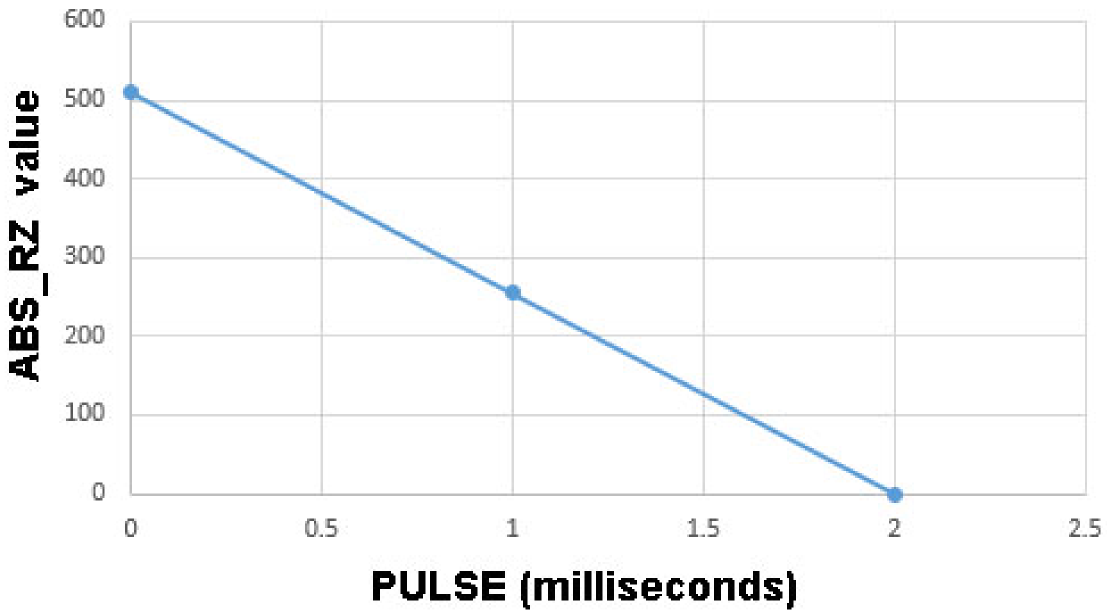
```
Python 2.7.13 (default, Nov 24 2017, 17:33:09)

>>>

============== RESTART: /home/pi/LaFavre_Logitech/joystick3.py ==============
```

forward 122	reverse 130	right 145	left 125
forward 122	reverse 136	right 161	left 120
forward 111	reverse 147	right 177	left 112
forward 95	reverse 162	right 193	left 101
forward 80	reverse 178	right 208	Left 87
forward 64	reverse 194	right 224	Left 71

forward 48	reverse 210	right 239	Left 55
forward 32	reverse 226	right 250	left 39
forward 17	reverse 240	right 255	left 25
forward 6	reverse 250	right 238	left 14
forward 0	reverse 255	right 222	left 6
forward 14	reverse 235	right 207	left 1
forward 29	reverse 219	right 191	left 0
forward 45	reverse 203	right 175	left 16
forward 61	reverse 187	right 159	left 32
forward 77	reverse 171	right 143	left 48
forward 93	reverse 155		left 64
forward 109	reverse 139		left 80
forward 121			left 96
			left 112
			left 123

I won't give you the complete program for making the joystick control a robot. However, here are some things to consider. The range of values for the joystick are 0 to 255. A value of 0 for 'ABS_RZ' is full forward and a value of 255 is full reverse. If we are going to control the robot with one of the motor controllers in GEAR stock, that controller takes a PWM ranging from 1.0 millisecond to 2.0 milliseconds. A pulse of 2.0 ms is full speed forward and a pulse of 1.0 ms is full speed reverse. A pulse of 1.5 ms will stop motor rotation. Therefore, we must convert an ABS_RZ value of 0 to 2.0 ms, a value of 255 to 1.0 ms. A value of 127 must convert to a value of 1.5.

I am not sure if you have had linear equations in your math class yet. We need to determine the linear equation for our data in order to make the conversions. Sometimes it helps to graph a linear equation, which I have done for you on the next page.



I have already stated that we need to have pulses between 1 and 2 milliseconds to control the robot. I have extended the graphed line out to the left to meet the Y axis at a value of 0 milliseconds for the pulse. You will see the reason for doing this soon.

A linear equation for the line graphed above will include a slope of the line and an offset (because the line does not go through the origin at 0 ABS_RZ and 0 PULSE). We can see that the line makes the conversions we want. For example, at 1 ms the ABS_RZ value is 255 and at 2 ms the ABS_RZ value is 0. If we had a very large graph, we could accurately determine various values of ABS_RZ and the equivalent PULSE values. However, teaching our robot to read a graph is beyond this lesson! We need a mathematical equation that describes the line.

We can start by determining the slope of the line. For one unit of change in PULSE, how much does the ABS_RZ value change? For an increase from one to two milliseconds, the ABS_RZ value changes from 255 to 0. That is a change of negative 255. We express the slope this way: change in Y/change in X:

-255/1 or just -255

The slope of the line is -255. Now we must account for the offset of the line from the origin. Just doing this mentally we might realize that at a PULSE of 0, the value of ABS_RZ should be 510 (at 2 ms it is 0, at 1 ms 255, at 0 ms 510, a linear equation). We put this all together and get this equation:

ARS_RZ = 510 – 255(PULSE)

or, rearranging the equation

PULSE = ARS_RZ – 510/ -255

Let us solve the equation using two values of ARS_RZ to see how this works.

PULSE = ARS_RZ – 510/ -255
PULSE = **255** – 510/ -255
PULSE = 1.0 ms

The equation computes to a value of 1.0 ms PULSE for an ARS_RZ value of 255, which is correct.

PULSE = ARS_RZ – 510/ -255
PULSE = 0 – 510/ -255
PULSE = 2.0 ms

The equation computes to a value of 2.0 ms PULSE for a ARS_RZ value of 0, which is correct.

Here is correct syntax for the second example in Python:

PULSE = (ARS_RZ – 510)/ -255

Suppose you want to use PWM code available in the RPi.GPIO library. In that case, you can't specify pulse width in units of milliseconds, but by the duty cycle percentage. If we set the PWM frequency to 50 Hz, then a duty cycle value of 5 is equivalent to 1.0 ms and a value of 10 is equivalent to 2.0 ms. I won't graph this out, but it is easy to determine the slope: change in Y/change in X:

-255/5 = -51

The linear equation we need to convert joystick values to percent duty cycle at 50 Hz is:

DUTY CYCLE = (ARS_RZ – 510)/ -51

For a joystick value of 0 (full forward) we need a duty cycle of 10 percent at 50 Hz. Using the linear equation below, we find the correct answer, 10:

DUTY CYCLE = (0 – 510)/ -51 = 10

For a joystick stick value of 255 (full back) we need a duty cycle of 5 percent at 50 Hz:

DUTY CYCLE = (255 – 510)/ -51 = 5

You now have enough information to create a Python 2 program for controlling a robot with the joystick. If you feel this would be a valuable experience in developing your IoT skills, go ahead and give it a try!