In a previous lesson you learned how to control the GPIO pins of the Raspberry Pi by using the **gpiozero** library. In this lesson you will use the library named **RPi.GPIO** to write your programs. You will write Python programs designed to control servos and direct current (DC) motors.

If you want a DC motor to run at full speed in one direction of rotation, then wiring it with an on/off switch might be sufficient. If you want to control the speed of rotation and the direction of rotation, then some type of controller is needed.

A servo is a special kind of DC motor. Inside the servo case, in addition to the DC motor, there are electronic components. These components receive an electrical signal and translate that signal into a specific output to run the motor.

GEAR club has in its inventory several types of servos and motor controllers. You will use some of these in this lesson.

Servos and motor controllers are frequently designed to receive control instructions by pulse-width modulation (PWM) signals. The RPi.GPIO library includes methods for outputting PWM signals on the GPIO pins.

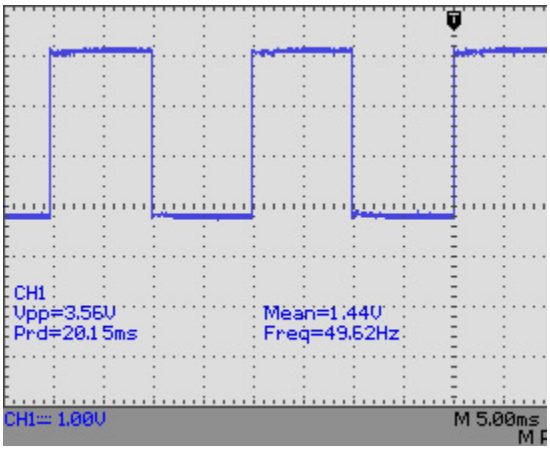

Figure 1 50 Hz, 50% duty cycle

The image on the left is a copy of an oscilloscope screen, depicting the output of a PWM signal from a GEAR Raspberry Pi. The signal has a square-wave pattern. The low part of the waves mark a voltage output of 0 volts while the top part of the waves mark a voltage of slightly more than 3 volts. This is the classical shape of a digital signal. Over time, the signal consists of two states, low and high (0 volts and 3.3 volts in this case).

The PWM signal illustrated has a **duty cycle** of 50 percent. Half of the time the signal is high and half the time it is low. A signal of 25% duty cycle would be high 25% of the time and low 75% of the time.

The signal is also characterized by its **frequency**. The frequency is the number of times the wave pattern is repeated in one second. Each square on the oscilloscope screen is set to represent 5 milliseconds. Notice that the signal is high for two squares and low for two squares. Therefore, the signal is high for 10 milliseconds (mS) and low for 10 mS. After 20 mS of time, the wave pattern repeats. The pattern repeats 50 times each second (1 second/0.020 second = 50). The unit for frequency is named **Hertz** (Hz) which means cycles per second.

You will use a Hitech model HS-485 servo for this lesson. The specifications for this servo are as follows:

Voltage range to power the servo: **4.8 to 6.0** volts DC

PWM signal range for 90 degrees rotation: about **1 to 2 milliseconds**

Current drain at 4.8 v, no load: **150 mA** (the amount of current it consumes during operation if doing no work)

In most cases it is not a good idea to connect a motor directly to the Raspberry Pi. In the case of the HS-485 servo, we can power the servo from the Raspberry Pi as long as we are careful.

Notice that the servo has a wiring cable containing three wires. The yellow wire is for the PWM control signal. The red and black wires are the standard color code for power (red = positive power, black = ground or negative). To be extra safe, the red and black wires of the servo can be connected to a separate power supply and only the signal wire connected to the Raspberry Pi. If a separate power supply is used for the servo, then a ground wire must be connected from the Raspberry Pi to the power supply.

Connecting the Raspberry Pi to the servo

1. connect the yellow wire to physical pin number 11 on Raspberry Pi
2. connect red wire to pin number 2 on Raspberry Pi (+5 volts)
3. connect black wire to pin number 9 on Raspberry Pi (ground)

Open Python 3 IDE on Raspberry Pi and save the program below

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
pwm=GPIO.PWM(11, 50)
pwm.start(7.5)
for i in range(0, 5):
    x=input("Where do you want the servo?  0-90")  #indent this and next 2 lines 5 spaces
    x=int(x)
    if(x>90):
        x=90 #indent this line and next one 10 spaces
        print("you entered a value greater than 90, a value of 90 was applied")
    desiredPosition=float(x)  #indent this line and next two lines 5 spaces
    DC=5.0/90*(desiredPosition)+5
    pwm.ChangeDutyCycle(DC)
pwm.stop()
GPIO.cleanup()
```

Run the above program. When the program asks you for input, put in a number from 0 to 90 and then press the Enter key. This number will determine the rotation position of the servo. Enter different

numbers from 0 to 90 and watch the servo rotate to the specified position. After you have entered five different numbers, the program will terminate.

Now let us examine the code line by line:

import RPi.GPIO as GPIO

import the library named RPi.GPIO and give it the name GPIO

GPIO.setmode(GPIO.BOARD)

there are two ways to refer to the GPIO pins: 1) the physical pin number or 2) the GPIO number. If you want to use the physical pin numbers in your code, then use this: (GPIO.BOARD), if you want to use the GPIO numbers then use this: (GPIO.BCM)

GPIO.setup(11, GPIO.OUT)

a pin can be set to be either an output pin or input pin. You have connected the servo signal wire to physical pin 11 on the Raspberry Pi. You want to have that pin output the PWM signal. (11, GPIO.OUT) makes pin 11 an output pin

pwm=GPIO.PWM(11, 50)

you want to output a pulse-width modulated signal on pin 11 with a frequency of 50 Hz. (11, 50) makes that setting.

pwm.start(7.5)

this line starts the signal output on pin 11 at a duty cycle of 7.5%. Since we have a 50 Hz frequency, each cycle lasts 20 mS. The signal is high for 7.5% of that time or 20 X 0.075 = 1.5 mS. A 1.5 mS high pulse should set the servo at its midpoint of rotation according to its specifications.

for i in range(0, 5):

this line specifies a **for** loop. Everything inside the loop is marked by indenting the lines by 5 spaces. That is the proper syntax for the Python programming language. The loop will be executed five times (0, 5).

x=input("Where do you want the servo? 0-90")

the program halts at this line waiting for the user to input a number with the keyboard. When a number is input, the variable named x is assigned the value of the input.

x=int(x)

the number input by the user is actually a string character. We can't do math with string characters. We need to change the number to an integer before using it in math calculations. This line changes the number to an integer.

if(x>90):

we have instructed the user to input a number between 0 and 90, but what if they put in a number more than 90? This would cause the servo to turn more than it is designed to do and might cause an excess of current to be drawn by the motor. This if statement takes care of the problem by converting any number above 90 to 90.

x=90

print("you entered a value greater than 90, a value of 90 was applied")

desiredPosition=float(x)

in the line below we will do some math using floating point numbers. If we try to do the math with an integer, the program will issue an error message and halt execution. In the line above, the value contained in the variable named x is converted to a floating point number and that value is assigned to the variable named desiredPosition

DC=5.0/90*(desiredPosition)+5

this is the equation to convert the desired angle of servo rotation to a duty cycle value. According to the servo specifications, a positive PWM pulse of 1.0 mS should cause the servo to rotate to a position of zero degrees and a pulse of 2.0 mS should cause a rotation to 90 degrees. For a frequency of 50 Hz, a 1.0 mS positive pulse is 5% duty cycle and a 2.0 mS pulse is 10% duty cycle. We can graph the degrees rotation (x axis) against the duty cycle (y axis) to get a slope of the line. The slope is calculated as follows: slope = (y2-y1)/(x2-x1) = (10-5)/(90-0)=5/90. We can convert a desired angle of rotation to a duty cycle value by multiplying the rotation angle by the fraction 5/90. Now take a look at the program line above, the part: **5.0/90*(desiredPosition)** In English this part translates to "multiply the fraction 5.0/90 by the value of the variable named desiredPosition" Notice that we are adding a value of 5 at the end of the line. Why is this necessary? Well suppose we leave it off and calculate the duty cycle for a rotation position of zero degrees. 5/90*0=0 the answer is a duty cycle of zero. However, a duty cycle of zero will produce a positive pulse of 0 mS and for zero degrees rotation we need a value of 5% duty cycle. That is why we must add a value of 5. Notice that the value of the calculation will be stored in a variable named DC.

pwm.ChangeDutyCycle(DC)

when the pwm.start() was issued a value of 7.5% duty cycle was specified so that the servo would rotate to the midpoint of its rotation. Now that the user has specified a rotational position, we need to change the duty cycle value, which is done by this line in the program. The value to change to is the value stored in the variable named DC.

pwm.stop()

when we want to stop the output of the pwm signal, we issue this command

GPIO.cleanup()

after the pins have been assigned a task there will be a problem running another program unless the pin assignments are closed off. Use this line of programming to close off the pin assignments.

Let us try another coding example. For this exercise we will add a button to the breadboard and use the button to control the position of the servo. When the button is in the up position the servo will rotate to the zero degree position and when the button is pressed, it will rotate to the 90 degree position. Leave the servo wired as in the previous lesson and add the wiring specified below.

Connecting Button to Raspberry Pi

1. insert button on breadboard
2. connect one terminal of button to physical pin number 12 on Raspberry Pi
3. connect the remaining terminal of button to physical pin number 6 (ground) on RPi

Open the Python 3 IDE on the RPi and enter the code provided below. Text following the # characters are just comments, you don't need to add the comments.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)      # use physical pin-numbering scheme
GPIO.setup(11, GPIO.OUT)      # pin 11 set as output type
pwm = GPIO.PWM(11, 50)        # Initialize PWM on pin 11 with 50 Hz frequency
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)
  #line above, pin 12 set as input type with software pull-up, input will be high unless button
  #connects pin 12 to ground pin 6

pwm.start(5)                  #start pwm signal with duty cycle of 5%

print("PWM signal started, press and hold button to change duty cycle. Press CTRL+C to exit")
    #print this message on the screen
try:
  while 1:            #this loop runs continually until user presses control key and c key together
    if GPIO.input(12):
                  # if button is not pressed then input will be high on pin 12 because we set
                  # pull_up_down=GPIO.PUD_UP
      pwm.ChangeDutyCycle(5)
```

```
                    #the duty cycle of the pwm is set to 5%, which at 50 Hz results in 1.0 mS
                    # positive pulse
            time.sleep(1) #wait one second then repeat loop


        else: # button is pressed:
            pwm.ChangeDutyCycle(10)
                    #the duty cycle of the pwm is set to 10%, which at 50 Hz results in 2.0 mS
                    #positive pulse
            time.sleep(1)  #wait one second then repeat loop

except KeyboardInterrupt:          # If CTRL+C is pressed on keyboard, exit cleanly:
    pwm.stop()                     # stop PWM
    GPIO.cleanup()                 # cleanup all GPIO
```

In this next exercise, you will use the RPi.GPIO library to create your own pulse width modulation program. This may help you to understand how pulse width modulation works. A PWM signal has a simple structure. It is composed of two parts: a high voltage pulse followed by a low voltage period. With the RPi we can make the output of a GPIO pin high or low (3.3 volts or 0 volts) by a direct instruction. Then all we need to do to make a PWM signal is specify how long the pin should be high and how long it should be low. If we add a **time.sleep()** instruction after each output instruction AND if we put all of this code in a loop, then we can create a PWM signal.

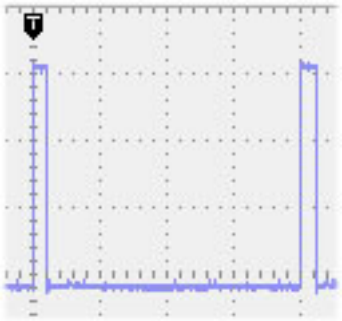

Figure 2 1 millisecond pulse at 50 Hz

Figure 2 is an image of an oscilloscope connected to a GEAR RPi that is outputting a 1 millisecond pulse at 50 Hz. Each square on the grid in a horizontal direction represents 5 milliseconds of time and in the vertical direction 1.0 volt. Notice that the pulses are slightly more than 3 squares high, so slightly more than 3 volts. Notice that each square is divided into 5 subdivisions and that the width of the pulse is one subdivision. Therefore, the high pulse lasts 1 millisecond. Notice also that the pulses are separated by 4 squares, which equals 20 milliseconds. Therefore, there will be 50 cycles or pulses each second (20 mS X 50 = 1,000 mS or 1 second). To create this structure in our program we need to first instruct the output pin to go high, then pause for 1 millisecond of time, then set the output pin to go low and stay there for 19 milliseconds. If we put these instructions in a loop, then we will produce a PWM signal. It is really just that simple.

Open up the Python 3 IDE and enter the program below. When you are finished, Mr. La Favre will assist you in connecting your RPi to an oscilloscope so that you can observe the PWM signal. Text after the # character are comments which you don't need to include, although you might want to do that.

```
import RPi.GPIO as GPIO    #import the RPi.GPIO library and give it the name GPIO
import time                #import the time library, its name will be time since we don't specify a name

GPIO.setmode(GPOIO.BOARD)  #use the physical pin numbering scheme
GPIO.setup(11, GPIO.OUT)   #set physical pin number 11 as an output pin

print("PWM signal has started, press Ctrl + c on keyboard to stop program")

try:
    while 1:      #this loop will run until Ctrl + c are pressed on keyboard, this line indented 5 spaces
        GPIO.output(11, GPIO.HIGH)  #make pin 11 output 3.3 volts, this line indented 10 spaces
        time.sleep(0.001)         #pause for 1 millisecond
        GPIO.output(11, GPIO.LOW)   #make pin 11 output 0 volts
        time.sleep(0.019)         #pause for 19 milliseconds

except KeyboardInterrupt:           #do this if Ctrl + c are pressed
    GPIO.cleanup()     #release GPIO pins so they can be used for another program, line indented 5 spaces
    print("the GPIO pins have been released and are ready for a new program")

#this program puts out a pulse width modulated signal with a positive pulse of 1 millisecond
#the duty cycle is 50 Hz  (one cycle is 1 + 19 = 20 milliseconds)
#therefore there are 50 cycles in one second: 50 X 20 mS = 1,000 mS or 1 sec
```