Understanding GPIO and PWM

1. Objective

In this tutorial we shall learn to control the velocity of DC motors (using PWM) interfaced with Raspberry Pi (GPIO Pins) using a motor driver IC.

2. Prerequisites

• Python programming skills

• R-Pi

• An idea about the working of a DC motor

3. Hardware Requirement

1. Windows Laptop/Computer that can connect to Wi-Fi Networks.
2. Raspberry Pi (We will be using Model B+)
3. DC motors
4. L298
5. Connecting wires

4. Software Requirement

1. IDLE (Python version 2.7 or above)
2. MobaXterm

Theory and Description

The motor we are using is a 60 RPM Single Shaft L-Shaped DC Motor. It is a great battery operated motor-gearbox combination for many low voltage mobile robot projects.

Motor Driver:

An individual pin in Raspberry pi can only supply enough power to light a LED, i.e., a maximum of 16mA. A DC Motor needs at least 40 - 80mA of current to start turning. If we directly connect the motors to the output of the Raspberry Pi, the pins may get damaged and it may not be able to drive the DC Motors. There is a need of a circuitry that can act as a bridge between the Raspberry Pi and the DC Motors. This is where a Motor Driver plays a crucial role. It regulates the current flowing through the circuit hence preventing any damage to the device. One way to construct that is given below:

- Using L298

NOTE: Do not connect a Motor, no matter how small its rating is directly to the Raspberry Pi, it will damage your Raspberry Pi.

L298 Motor Driver:

The Motor driver that we are using here is called L298. L298 is a 15-pin IC which allows the DC Motor to drive in either direction. Figure 1 shows the Pin description of a typical L298 IC.

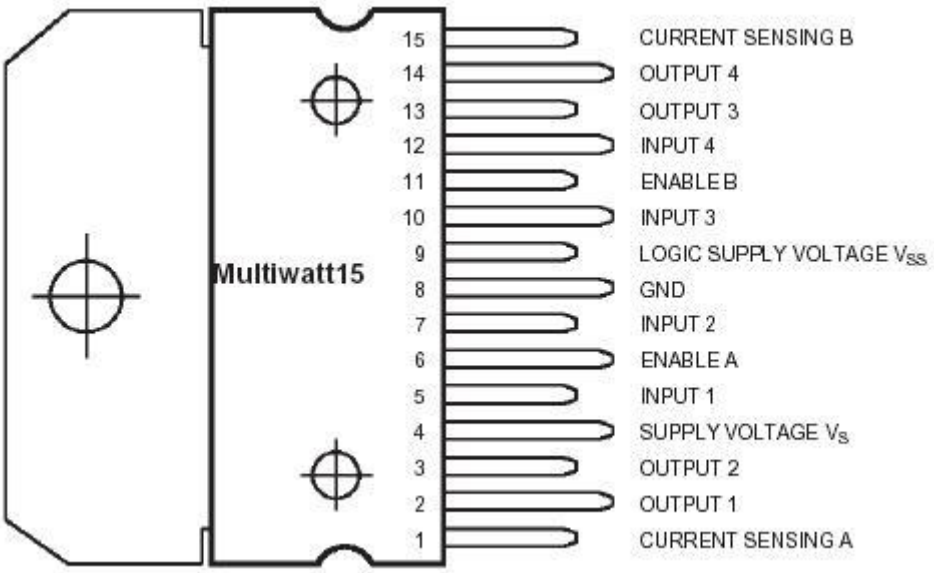

Figure 1: Pin Diagram of L298 IC

The supply voltage at pin 9 is equal to 5V. It is used to provide power for the chip's logic. Functionalities of the IC, are as explained below:

- The **Inputs** to the IC are pins 5, 7, 10 and 12. They are named INPUT1, INPUT2, INPUT3 and INPUT4 respectively. Similarly, the **Outputs** of the IC are pins 2, 3, 13 and 14. They are OUTPUT1, OUTPUT2, OUTPUT3 and OUTPUT4 respectively.
- There are 2 **Enable** pins, Enable A and Enable B (pins 6 and 11), which are used to activate either half of the IC.
- From the functional table, we can understand that the IC will replicate the set of given inputs at the outputs if and only if the Enable pins are activated.

Table 1: Functionalities of the IC

Enable	Inputs	Outputs
HIGH	HIGH	HIGH
HIGH	LOW	LOW
LOW	X	Z

X = don't care (HIGH or LOW), Z = HIGH IMPEDANCE

HIGH = 5V, LOW = 0

Channels in L298:

L298 Motor driver is divided into four channels.

- In L298 IC, INPUT1and OUTPUT1 constitute channel 1. INPUT2 and OUTPUT 2 constitute channel 2. Similarly we have channels 3 and 4 for INPUT3/OUTPUT3 and INPUT4/OUTPUT4 respectively. All these channels share a common GND pin (pin 8).
- Enable A pin- used to power two channels i.e. channel 1 and channel 2.
- Enable B pin- used to power two channels i.e. channel 3 and channel 4.
- For any system to work, required power supply must be channeled to both the sections.
- The distribution of power from the Raspberry Pi to the Motor driver is done by the Enable pins and is used to direct the power to various channels in L298 Motor Driver.
- Whenever HIGH signal supplied to the Enable pins, it powers two respective channels as given above.
- Speed of Motors can also be controlled by supplying pulses of defined widths to Inputs of Motors or to Enable A and Enable B.
- This process where the information is transmitted on a series of pulses of varying widths in order to control the power supplied to the load (here geared DC Motor) is called as **Pulse Width Modulation** commonly known as **PWM.**

When the signal is HIGH, we call this "ON TIME" (T_{ON}). To describe the amount of "ON TIME", we use the concept of **Duty cycle.** Duty cycle is measured in percentage. The percentage duty cycle specifically describes the percentage of time a digital signal is ON over an interval or period of total time (T). This period is the inverse of the frequency of the waveform.

$$D = \frac{T_{ON}}{T} \times 100$$

If a digital signal is ON for T/2 and OFF for T/2, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave. If the percentage is higher than 50%, the digital signal stays more time in the HIGH state than the LOW state and vice versa. Here is a graph that illustrates these three scenarios:

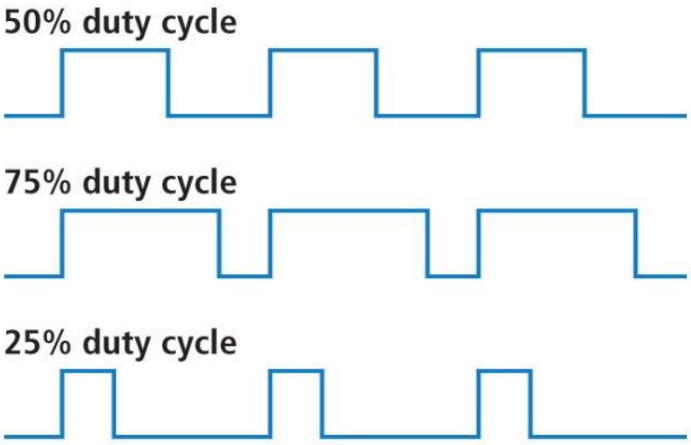

Figure 3: Duty Cycle

Hence by changing the value of Duty cycle we can control the time for which the Motor is in the ON state thus controlling the speed of Motor.

PWM can be generated in a number of ways on the Raspberry Pi:

1. **Inbuilt Hardware**: The Pi can perform PWM in hardware, but this can only be done on one pin (GPIO18 in R-Pi).

2. **Software** (using delay function**)**: PWM can be generated through software which is an easy option. It is not as precise as hardware PWM, however in most cases software PWM works well for all GPIO pins.

Note: We have used Software PWM.

Interfacing DC Motors with Raspberry Pi:

Figure 7 shows the Pictorial view of Interfacing of DC Motors with the Raspberry Pi with control inputs A1, A2, A3 and A4 connected to the GPIO pins of Raspberry Pi. (Connection details are given in the Experiment Section below)

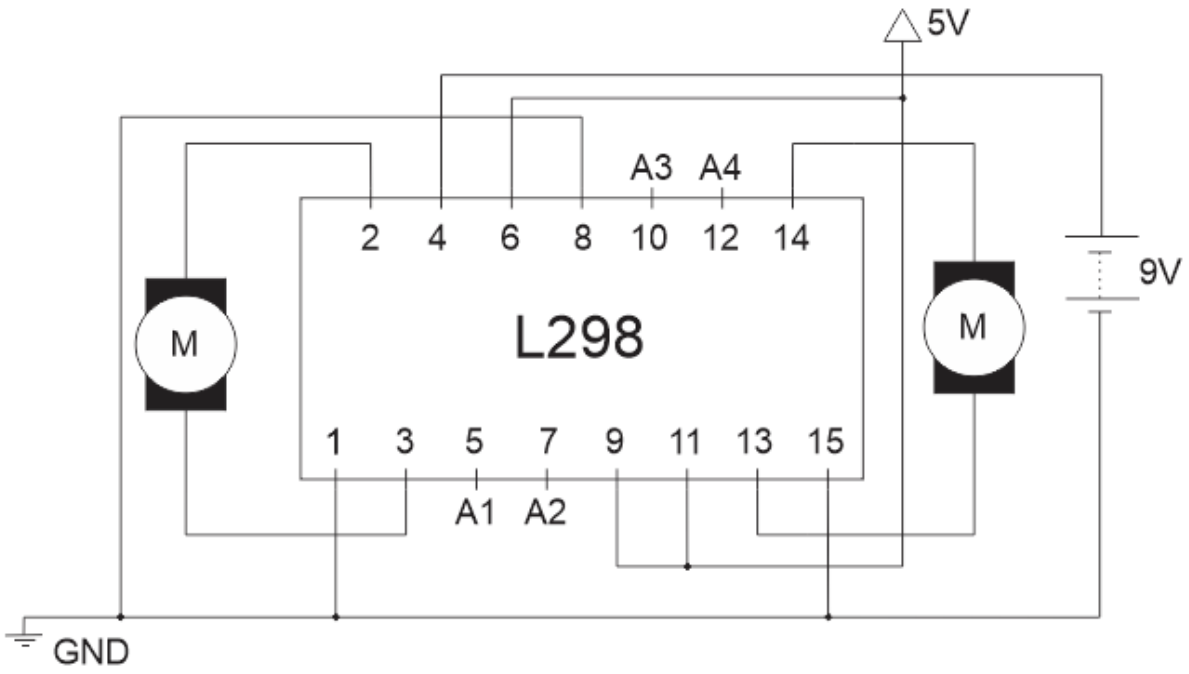

Figure 7: Interfacing DC Motor with Raspberry Pi

Let us consider interfacing of the Left motor (The same method can be applied to interface the Right motor as well).

Consider the following cases when Enable1 pin is HIGH:

Case 1: When Logic 0 is applied to both the inputs A1 and A2, the Motor does not rotate since there is no supply of power to it.

Case 2: When Logic 1 is applied to input A1 and Logic 0 is applied to input A2, the Motor rotates in clockwise direction.

Case 3: When Logic 0 is applied to input A1 and Logic 1 is applied to input A2, the Motor rotates in anticlockwise direction.

Case 4: When Logic 1 is applied to both the inputs A1 and A2, the Motor does not move since both the control inputs are HIGH. It is like the 'toggle state' in digital electronics.

These cases are presented in Table 2.

Table 2: Motor reaction Table

A1	A2	OUTPUT1	OUTPUT2	Motor 1
Logic 0	Logic 0	0	0	Stop
Logic 1	Logic 0	5V	0	Clockwise
Logic 0	Logic 1	0	5V	Anti-clockwise
Logic 1	Logic 1	5V	5V	Stop

Experiment:

In this experiment we will be controlling a DC Motor using software PWM generation technique.

Use following connections while performing experiment:

- Pin 2 of Raspberry Pi supplies 5V:

This pin is used for supplying 5V to L298.

- Pin 6 of Raspberry Pi supplies GND:

This pin is used for supplying Ground L298 IC.

- Pin 16 and Pin 18 of Raspberry Pi:

These are used to supply software generated PWM to RIGHT MOTOR.

- Pin 11 and Pin 15 of Raspberry Pi:

These are used to supply software generated PWM to LEFT MOTOR.

Table 3: Motion Control

PIN 18 (Duty Cycle)	PIN 16 (Duty Cycle)	PIN 13 (Duty Cycle)	PIN 15 (Duty Cycle)	MOTION
100%	0%	100%	0%	FORWARD
100%	0%	0%	0%	SOFT RIGHT
100%	0%	0%	100%	HARD RIGHT
0%	0%	100%	0%	SOFT LEFT
0%	100%	100%	0%	HARD LEFT
0%	100%	0%	100%	BACKWARD

Please refer to this link in order to test and control the Rover Assemble's DC motors. Please use the components given in the kit.

Note: Here we have used Pin number of the Raspberry Pi, these pins numbers can be mapped with the GPIO pin numbers. For example, refer Figure 8, PIN 16 = GPIO 23, PIN 18 = GPIO 24, to map Pin numbers with the GPIO pin numbers.

Raspberry Pi Pin-out Diagram

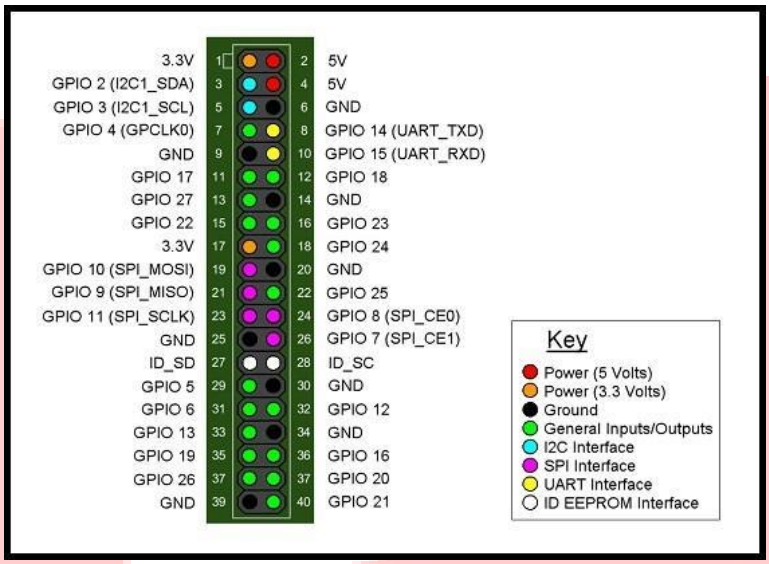

Figure 8: Pin- Out Raspberry Pi

CODE:

We will now write a program to make the robot move forward for 5 seconds and move backwards for 5 seconds in loop.

Note: In the code we have used GPIO values instead of Pin numbers.

```
import RPi.GPIO as GPIO
import time
```

import RPi.GPIO – Used to access GPIO pins

import time – For using **time.sleep( )** to generate delays in your Python script

```
GPIO.setmode (GPIO.BCM)
GPIO.setwarnings (False)
```

After including the **RPi.GPIO** module, the next step is to determine which of the two **pin-numbering**schemes to be use:

1. **GPIO.BOARD:** Board numbering scheme. The pin numbers follow the pin numbers on header P1.

2. **GPIO.BCM:** Broadcom chip specific pin numbers. These pin numbers follow the lower-level numbering system defined by the Raspberry Pi's Broadcom-chip brain.

Use the **GPIO.setmode()** function to specify which numbering-system is being used in your code.

GPIO.setwarnings(False) function stops any warning generated due **GPIO**.

```
#setting the GPIO pin as Output
GPIO.setup (24, GPIO.OUT)
GPIO.setup (23, GPIO.OUT)
GPIO.setup (27, GPIO.OUT)
GPIO.setup (22, GPIO.OUT)
```

To declare "pin mode" before you can use it as either an Input or Output. To set a pin mode, use the **GPIO.setup([pin]), [GPIO.IN,GPIO.OUT]**

In the example above pins 24,23,27,22 are set as Output.

```
#GPIO.PWM( pin, frequency ) it generates software PWM
PWMR = GPIO.PWM (24, 60)
PWMR1 = GPIO.PWM (23, 60)
PWML = GPIO.PWM (27, 60)
PWML1 = GPIO.PWM (22, 60)
```

To initialize PWM, use **GPIO.PWM([GPIO pin], [frequency])** function. To make rest of your script-writing easier you can assign that instance to a variable.

In the case above, variables**PWMR, PWMR1, PWML and PWML1.**

```
#Starts PWM at 0% dutycycle
PWMR.start (0)
PWMR1.start (0)
PWML.start (0)
```

```
PWML1.start (0)
```

Use **variable.start([duty cycle])** function to set an initial value. Here we are initializing it to "zero" duty cycle.

Duty cycle can have values: 0 to 100

While True:

```
#FORWARD
print"Going Forward"
PWMR.ChangeDutyCycle (100) #changes duty cycle to 100%
PWML.ChangeDutyCycle (100) #changes duty cycle to 100%
time.sleep(5)
```

To adjust the value of the PWM output use the **variable.ChangeDutyCycle([duty cycle])**

As explained earlier under the Topic PWM change the speed of Motor by changing the Duty cycle. Here we are using 100% Duty cycle. It can be changed to any value ranging from 0 to 100.

100% Duty cycle Motor will rotate at maximum speed and gradually decreasing the value of Duty cycle will correspondingly decrease the speed of Motor.

```
#BACKWARD
print"Going Backward"
PWMR1.ChangeDutyCycle (100) #changes duty cycle to 100%
PWML1.ChangeDutyCycle (100) #changes duty cycle to 100%
time.sleep (5)
```

To adjust the value of the PWM output the **variable.ChangeDutyCycle([duty cycle])** can be used.

```
PWMR.stop ( ) #stops software PWM
PWMR1.stop ( ) #stops software PWM
PWML.stop ( ) #stops software PWM
PWML1.stop ( ) #stops software PWM
```

variable.stop() function is used stop software PWM being generated corresponding to that variable.

```
GPIO.cleanup ( ) #cleans the GPIO pins of the previous
```

Once your script has run its course use the **GPIO.cleanup( )** command at the end of your script to release any resources your script may be using.

NOTE: If your motors are rotating in the opposite directions compared to the directions given in Table 2, simply interchange the motor connections. For example, for

PWMR.ChangeDutyCycle (100)

PWML.ChangeDutyCycle (100)

If your right motor moves forward but the left motor moves backwards, just interchange the connections of the following pins: GPIO 27 (PIN 13) and GPIO 22 (PIN 15).

… END …