LESSON 11

MOTOR PWM AND CALIBRATION

OBJECTIVE

In this lesson, you will learn how to use Pulse Width Modulation (PWM) signals from GPIO pins to control the speed of the drive motors on your robot. You will also learn how to adjust these PWM values to help your robot drive most accurately.

MATERIALS

- Assembled Robot from Lesson D-10
- Wireless home network for connecting multiple devices
- Windows PC or Chromebook to establish a remote connection with the Raspberry Pi

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- Pulse Width Modulation (Lesson B-4)
- Networking and Remote Access (Lesson C-1)
- Driving Motors (Lesson D-10)

LESSON

In the previous lesson, you connected the drive motors on your robot to the L293DNE motor drive IC and tested both motors in both directions. This test was conducted with the motors running at full speed, but there are often times when you might want the motors to run at lower speeds:

- The robot is near an object.
- The robot is trying to follow a line using its sensors.
- The robot is turning too quickly to be easily controlled remotely.

The input lines of the motor drive IC are digital, meaning they will only recognize high or low signals. This means that the only options for driving those inputs are high or low, which will result in the connected motor either being off or driving at full speed. Since the speed of the motor can't be controlled by the motor input lines, the only way to control the speed is to use the enable lines of the motor drive IC.

PWM CONTROL OF DC MOTORS

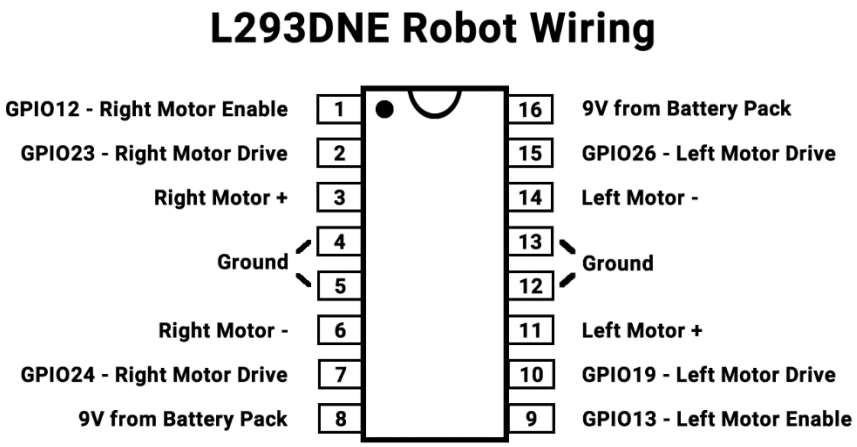
You may remember from previous lessons that a PWM signal allows you to rapidly turn a signal on and off. We previously used a PWM output to control the brightness of an LED. With standard high and low signals, the LED will either be on or off. Brightness values in between can be attained by rapidly turning the LED on and off. If this high/low switching is done fast enough, the LED will appear to be on, but dim. This brightness value can be adjusted by manipulating the duty cycle of the signal which can be adjusted to any value between 0%, or never high, to 100%, or always high.

- A higher duty cycle means the signal is high more often, and the LED will be brighter.
- A lower duty cycle means the signal is high less often, and the LED will not be as bright.

These principles for controlling the brightness of an LED can be directly applied to speed control of a motor. By using a PWM signal to control the enable line of the motor drive IC, you can control how often that enable line will be high, which will control how often the enable line will allow the motors to drive.

MOTOR DRIVE IC PINOUT

Here is the pinout of the L293DNE motor drive IC as it will be used in our robot:



GPIO12 will be controlling the enable line for the right motor, and GPIO13 will be controlling the enable line for the left motor. Sending out PWM signals from these GPIO pins on the Pi will allow the speed of each motor to be controlled independently.

Pushing an enable line high will only cause a motor to move if one of the motor drive inputs to that channel is also high. Let's say for example, the right motor drive lines on GPIO23 and GPIO24 are both low. You can drive GPIO12 high permanently, or intermittently using PWM, but the right motor will still never move. The motor drive IC will pass the signals from GPIO23 and GPIO24 to the motors, but since both are low, the motor will not move. A high would also be required on either GPIO23 or GPIO24 in order to make that motor move:

GPIO12 (Enable)	GPIO23	GPIO24	Motor
Low	High	Low	No movement – Enable line is low
Low	Low	High	No movement – Enable line is low
High	High	Low	Fast movement
High	Low	High	Fast movement
PWM	High	Low	Speed controlled movement
PWM	Low	High	Speed controlled movement

LOW SPEED MOTOR CHARACTERISTICS

PWM control of the enable lines on the motor drive IC is a great way to control the speed of motors, and high speeds are no problem. As the value of the PWM signal approaches 100% duty cycle, the motor will just run faster and faster, until it's running full speed.

Caution must be exercised when reducing the PWM duty cycle value to levels that cause the motor to move too slowly, or not at all. Low speed operation is not exactly the same as high speed. A PWM duty cycle value of 99% will run the motor at almost full speed, or 100%. This is not the case for low speeds. At 0% duty cycle, the motor will be off. At 1% duty cycle, the motor will be turning on 1% of the time, which will not be enough to get the motor to move. Trying to drive the motor with a PWM duty cycle value that's too low will cause a condition called a stall.

A motor stall occurs when there is not enough current running through the motor windings to create movement. This stall condition can cause unwanted heat in the motor, motor drive IC, or associated wiring if allowed to continue too long. A stall can often be identified by a high-pitch whine coming from a motor when it should be moving. This is an indicator that the PWM duty cycle that you're sending the motor is too low.

The minimum PWM duty cycle value that can be used to drive a motor will be based on many variables:

- Construction of the motor – Variables from one motor to another can affect the lowest speed that a motor can run at without stalling.
- Gear ratio of the gearbox, if one is present – A gearbox will generally lessen the load on the motor, allowing the motor to turn at higher speeds, while the wheel/tire turns more slowly.
- Size of the wheel/tire combination attached to the motor – Larger diameter wheel/tire combinations will be more difficult for the motor to rotate. A larger the wheel/tire combination will cause the motor to stall more easily.
- Voltage being supplied to the motor drive IC – It's easier to stall a motor when the motor drive IC is being run by a lower voltage supply. A motor drive IC running on 5-volts will cause motors to stall easier than one running at 9-volts or more.
- Resistance of the driving surface – A smooth driving surface like hardwood or concrete will allow for lower PWM values before stalling. Surfaces like carpet will cause more resistance for the robot to drive over and will require a higher PWM value to avoid stalling.

- Weight of the robot – A light robot will be easier for the motors to propel, so lower PWM values can be used. As the robot increases in weight, driving the motors will require more force, so a higher PWM value will be required.

With so many factors involved, the only way to figure out the minimum PWM duty cycle that can be used on your robot before stalling the motors is experimentation. Every robot and driving surface will be a little different.

LEFT AND RIGHT MOTOR VARIABILITY

In a perfect world, when commanded to drive straight, your robot would drive in a perfectly straight line. As we found in the last section, many variables will come into play when determining exactly how fast a motor will move. This is especially problematic when the enable lines of the motor drive IC are not controlled using PWM.

Without PWM control, each motor will run as fast as possible when commanded. This could result in one motor running at 195 revolutions-per-minute (RPM), and the other motor running at 205 RPM. This may not sound like much, but that means that wheel will rotate 10 more times than the other over the course of a minute. Breaking that down even further, one of the wheels will make one extra revolution every 6 seconds. Even over short distances, this will result in a slow turn when the robot is supposed to be driving straight.

PWM control of the left and right enable lines on the motor drive IC will allow you to “calibrate” the left and right motor speeds independently. If you command both motors to drive forward at a PWM value of 95% and the robot is turning slightly left, that means the right motor is driving a little faster than the left when using 95% PWM values. Slightly decreasing the PWM value of the right motor will slow down that motor and should cause the robot to drive more accurately in a straight line. Reducing the right motor PWM value too much could cause the robot to start turning to the right. Finding the right value might take a couple of attempts, but the driving accuracy that’s gained will be well worth the effort.

ACTIVITIES

In the following activities you will use the program created in Lesson 10 to verify the motors are driving in the correct directions when commanded. You will then modify this program to add PWM speed control to both motors and find the best left and right PWM values for driving the robot in a straight line.

ACTIVITY #1 – MOTOR DIRECTION TEST

In the last lesson you confirmed that both motors would turn in both directions with the robot held above your work surface. This is a good test, but there could still be problems that exist with your motor wiring that could be causing reversed commands to the drive motors. In this activity, you will modify the program from Lesson 10 to drive both motors in forward and then reverse to confirm the motors propel the robot in the expected direction when using the GPIO pin numbers below:

High on GPIO Pin	**Motor Movement**
GPIO23	Right Motor Reverse
GPIO24	Right Motor Forward
GPIO19	Left Motor Forward
GPIO26	Left Motor Reverse

> NOTE – The enable lines required for these motor movements have been omitted from this chart for clarity.

We will first be powering the robot using AC wall power to modify the program, and then switching to battery power for mobile testing once the program is complete.

STEP #1

If the Pi is currently up and running, shut the Pi down before proceeding.

Disconnect battery power input from the Pi by disconnecting the micro USB power connector. Connect the 5V wall power adapter to wall power and then to the micro USB power connection of the Pi. The Pi will power up and automatically connect to your WiFi network.

STEP #2

Connect to the VNC server of the Raspberry Pi using a desktop computer or laptop connected to your WiFi network. Once connected, you should be viewing the Desktop of the Pi just as if you had a monitor, keyboard, and mouse connected directly to the Pi.

Open the folder named **`robot`** on the Desktop and double-click on the file named **`motor_test.py`** to open the file in Thonny.

STEP #3

Save a new copy of this program in Thonny by selecting **`File`**, and then **`Save As`** from the upper-left menu. Double-click on the **`robot`** folder in the displayed list of files to save your file in the same location as the original. Type **`motor_fwd_rev.py`** in for the file name and click on the **`OK`** button in the lower-right of the Thonny window.

You will now be working in a new copy of the file so the original can be used again later, if needed.

STEP #4

The first modification will be to allow the function named **drive_motor** to allow for multiple arguments. The arguments that we will use will be pin1, pin2, and state:

Pin1 – A pin to control

Pin2 – A pin to control

State – The desired state of high or low

This will save quite a bit of code since we will be controlling multiple motor pins to drive the motors. Change the word **pin** after the function name to **pin1, pin2, state**:

```
def drive_motor(pin):
```

will become

```
def drive_motor(pin1, pin2, state):
```

STEP #5

The next modification will be to replace the code the function will run when called. The new function behavior will be to change the high/low state of **pin1** and **pin2** to the **state** specified in the arguments when the function is called. Remove the current content from the **drive_motor** function and replace it with the following highlighted lines of code:

```
def drive_motor(pin1, pin2, state):
    GPIO.output(pin1, state)
    GPIO.output(pin2, state)
    time.sleep(1)
```

This will cause **pin1** and **pin2** to pushed high or pulled low based on the value of the state argument, and then a **1** second delay will occur before returning to the main program.

STEP #6

It's now time to modify the main program motor command to include the additional pin and state information that was added to the function. Here are the motor commands that will be used:

1. The **r_for** and **l_for** pins will be pushed high to drive the robot forward.
2. The **r_for** and **l_for** pins will be pulled low to stop the robot.
3. The **r_rev** and **l_rev** pins will be pushed high to drive the robot in reverse.
4. The **r_rev** and **l_rev** pins will be pulled low to stop the robot.

Remove the existing **drive_motor** function calls and replace them with the highlighted lines of code below:

```
    time.sleep(1)

drive_motor(r_for, l_for, 1)
drive_motor(r_for, l_for, 0)
drive_motor(r_rev, l_rev, 1)
drive_motor(r_rev, l_rev, 0)

GPIO.cleanup()
```

The program is now fully modified, and all changes have been highlighted below:

```
import RPi.GPIO as GPIO
import time

motors = [13,19,26,12,23,24]
r_enable = 12
r_for = 24
r_rev = 23
l_enable = 13
l_for = 19
l_rev = 26

GPIO.setmode(GPIO.BCM)
GPIO.setup(motors, GPIO.OUT)
GPIO.output(motors, GPIO.LOW)

GPIO.output(r_enable, 1)
GPIO.output(l_enable, 1)

def drive_motor(pin1, pin2, state):
    GPIO.output(pin1, state)
    GPIO.output(pin2, state)
    time.sleep(1)

drive_motor(r_for, l_for, 1)
drive_motor(r_for, l_for, 0)
drive_motor(r_rev, l_rev, 1)
drive_motor(r_rev, l_rev, 0)

GPIO.cleanup()
```

STEP #7

Save the program and shut down the Raspberry Pi. Save your program, exit Thonny, and shut down the Pi. Once the SD card activity LED is no longer flashing green, remove micro USB power cable from the Pi.

Connect the micro USB power cable from the on-board 5V converter to the Pi. Next, connect the 9-volt battery pack to the 5V converter, powering up the robot.

Caution – When running on power from the battery pack, the motors will be powered up and ready to move. **Make sure that your robot is on the floor or in some location where driving off the edge of a desk or tabletop is not a possibility**, as this could result in damage to the robot that cannot be easily repaired.

Do not continue to run the robot if the red power LED on the Pi is flashing or turning off. Discontinue use of the robot immediately and charge or replace the batteries. Running the robot in a low battery pack could result in corruption or data loss on the SD card in the Pi.

STEP #8

Connect to the VNC server of the Raspberry Pi using a desktop computer or laptop connected to your WiFi network. Once connected, you should be viewing the Desktop of the Pi just as if you had a monitor, keyboard, and mouse connected directly to the Pi.

Open the folder named robot on your Desktop and double-click on the program named **`motor_fwd_rev.py`** to open it in a new Thonny window.

Ensure the area around the robot is clear of any obstacles and safe for driving. Run the program to confirm that the robot drives forward for one second, stops, reverses for one second, and stops. If the robot deviates from this expected behavior, power the robot down and check the list of possible wiring causes below:

- Robot drives in reverse and then forward – Both motors have swapped connections, check connections of both motors from GPIO pin, through motor drive IC, and onto motor per wiring connections made in Lesson 10.
- Robot drives in circles instead of forward/reverse – One motor is reversed, determine which motor drives in reverse when program starts. Check wiring for this motor per Lesson 10 and correct reversed wiring as needed

Do not proceed to the next activity until running this program results in the robot driving forward and then backward.

ACTIVITY #2 – ADD PWM DRIVE CONTROL TO MOTORS

In this activity you will modify the program from the last activity to include PWM control of the enable lines on the motor drive IC. This will enable independent speed control of the left and right motors. This program modification will be quick and can be done on battery power.

STEP #1

If not already running from the last activity, power up the robot on battery power and connect to the onboard VNC server using another device on your network.

Once at the Desktop, open the folder named **`robot`** and double-click on the file named **`motor_fwd_rev.py`** to open the file in Thonny.

Save a new copy of this program in Thonny by selecting **`File`**, and then **`Save As`** from the upper-left menu. If needed, double-click on the **`robot`** folder in the displayed list of files to save your file in the same location as the original, which is inside the folder named **`/home/pi/Desktop/robot`**. Type **`motor_pwm.py`** in for the file name and click on the **`OK`** button in the lower-right of the Thonny window.

You will now be working in a new copy of the file so the original can be used again later, if needed.

STEP #2

The first modification that will be made to this program will be to add variables that contain PWM values that will be used for the right and left motors. We will use the variable names **r_speed** and **l_speed** to store values of **99** for both motors as a starting point. Add these variables and their values to the list of variables at the beginning of the program:

```
motors = [13,19,26,12,23,24]
r_enable = 12
r_for = 24
r_rev = 23
r_speed = 99
l_enable = 13
l_for = 19
l_rev = 26
l_speed = 99
```

STEP #3

The next modification will be to configure the PWM signals that will be used to drive the **l_enable** and **r_enable** pins. We will use the names **l_pwm** and **r_pwm** for the left and right PWM signals and we use 1000Hz for the frequency. Add the highlighted code below to the program just before the **drive_motor** function:

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(motors, GPIO.OUT)
GPIO.output(motors, GPIO.LOW)

l_pwm = GPIO.PWM(l_enable, 1000)
l_pwm.start(l_speed)
r_pwm = GPIO.PWM(r_enable, 1000)
r_pwm.start(r_speed)

def drive_motor(pin1, pin2, state):
    GPIO.output(pin1, state)
    GPIO.output(pin2, state)
```

This will configure and start the PWM signals on the left and right enable pins using the **l_speed** and **r_speed** duty cycle values that were specified in earlier in the program.

STEP #4

The goal of this test is to see how straight the robot is driving, which may be a little difficult if the robot only drives for one second. Change the **`time.sleep`** value in the **`drive_motor`** function from **`1`** to **`2`** so you will be able to get a better view of how straight the robot is driving:

```
def drive_motor(pin1, pin2, state):
    GPIO.output(pin1, state)
    GPIO.output(pin2, state)
    time.sleep(2)
```

STEP #5

The program is now ready to drive the motors using PWM speed control values of 99 for both the left and right motors. In a perfect world, these equal values would cause the robot to drive perfectly straight, but as you learned in this lesson, many factors are working against these being the best values.

Ensure the area around the robot is clear of any obstacles and safe for driving. Keep in mind that the robot will be driving twice as far as the first test since we doubled the **`time.sleep`** value in the **`drive_motor`** function.

Run the program and monitor the robot's accuracy when driving forward and then in reverse. Here are the ways to handle adjustments as needed:

- The robot is driving straight – No adjustment is needed, proceed to the next step.
- The robot is turning slightly to the left – This means the right motor is driving too quickly. Substitute a slightly lower number for the value of **`r_speed`** and run the test again. Keep adjusting this value until the robot is driving straight.
- The robot is turning slightly to the right – This means the left motor is driving too quickly. Substitute a slightly lower number for the value of **`l_speed`** and run the test again. Keep adjusting this value until the robot is driving straight.

STEP #6

Now that the robot is driving straight using your PWM values, make note of these values as they will be useful for future programs that you will create that use PWM speed control of the left and right motors on your robot.

Since the PWM speed control test and calibration is now complete, power off the Raspberry Pi. Once the SD card activity LED has stopped flashing, disconnect the battery pack from the Voltage Regulator Module to fully remove power from the robot.

QUESTIONS FOR UNDERSTANDING

1. Will a robot always have perfectly matched left and right drive motors that will allow the drive in a straight line, or will PWM speed control of the motor enable lines be needed to fine-tune driving accuracy?

2. What might cause a drive motor to emit a high-pitch sound and not rotate?

3. If your robot is turning slightly to the right when driving forward, should the left or right motor speed be lowered?

Answers can be found on the next page.

ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

1. *Will a robot always have perfectly matched left and right drive motors that will allow the drive in a straight line, or will PWM speed control of the motor enable lines be needed to fine-tune driving accuracy?*

   ANSWER: PWM motor speed control will almost always be required to calibrate a robot for straight driving due to the number of factors that can cause drive inconsistencies between the left and right sides.

2. *What might cause a drive motor to emit a high-pitch sound and not rotate?*

   ANSWER: The high-pitch sound and lack of rotation indicate a stall condition. This is often caused by a PWM duty cycle that's too low, and a higher value should be used to ensure a stall condition is avoided.

3. *If your robot is turning slightly to the right when driving forward, should the left or right motor speed be lowered?*

   ANSWER: A slight right turn when driving forward indicates that the left motor is turning faster than the right. The left motor speed should be lowered to allow for better straight driving performance from the robot.

CONCLUSION

In this lesson, you learned how to use PWM signals to control the enable lines of the motor drive IC, which can be used to control the speed of DC motors. This will allow you more flexibility in the way the motors are used to the drive the robot in different scenarios.

In the next lesson, you will learn how to document projects to ensure the best outcome, both before the project begins, and after the project has been completed.