CamJam EduKit Robotics – Control and Calibration

Project Control and Calibration

Description Before you set your Robot loose on the world, you first need to get better control of it.

Equipment Required

For this worksheet, you will require:

- Your robot
- A battery for your Pi

Control and Calibration

In worksheet three, you learnt how to make your robot move, but you may have noticed that it is quite quick. Before you try to control it either autonomously or with a remote controller, you need to be able to slow it down so that it is easier to manoeuvre.

How can you do that when you can only turn GPIO pins on and off? The GPIO Python library has something called PWM – or Pulse-Width Modulation – a way of turning the GPIO pins off and on very quickly. It happens so quickly that you will not see the motors stop and start. By changing how quickly the GPIO pins are switched on and off, and how long the pulses last, you can change the speed of the motors.

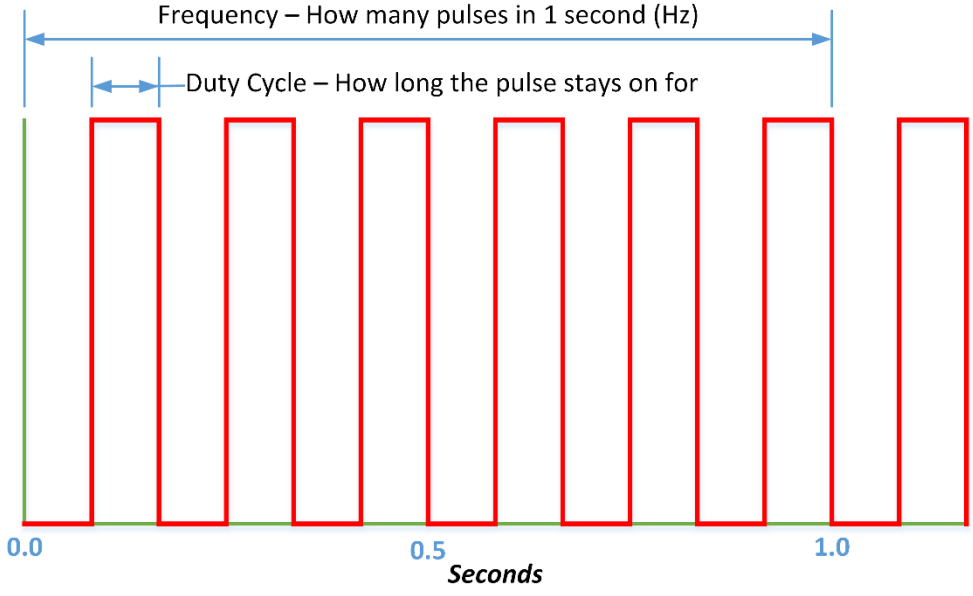
There are two important parameters that determine PWM: "frequency" and "duty cycle".

Frequency

Frequency, in Hertz (Hz) is the number of times per second that a pulse is generated. This counts from the start of one pulse to the start of the next. i.e. from when the pulse starts to rise, to the next time it starts to rise. So, it includes all the "on" time and "off" time and "in between" time for one complete wave cycle.

Duty Cycle

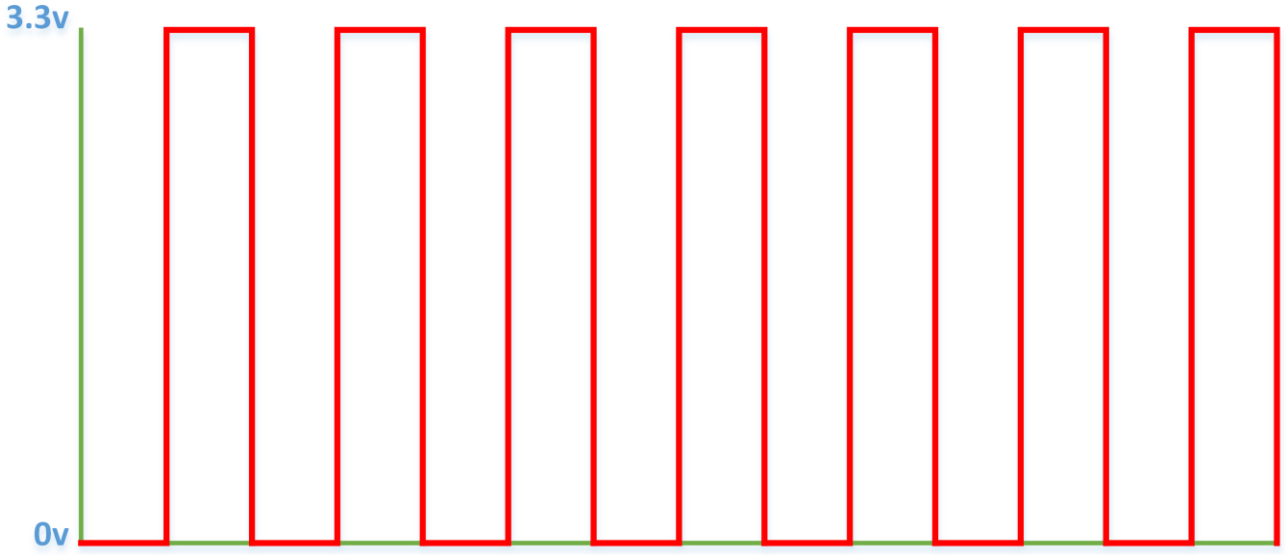
Duty cycle is the percentage of time between pulses that the signal is "High" or "On". So if you have a frequency of 50 Hz and a duty cycle of 50%, it means that every 1/50th (0.02) of a second a new pulse starts and that pulse is switched off half way through that time period (after 1/100th or 0.01s).



Explaining PWM

Example - Frequency 50 Hz, Duty Cycle 50%

This gives a pulse 50 times per second (or every 0.02 seconds). During each 0.02 second time period, the port will be "High" (3.3V) half the time and "Low" (0V) the other half. An oscilloscope trace would look like this:

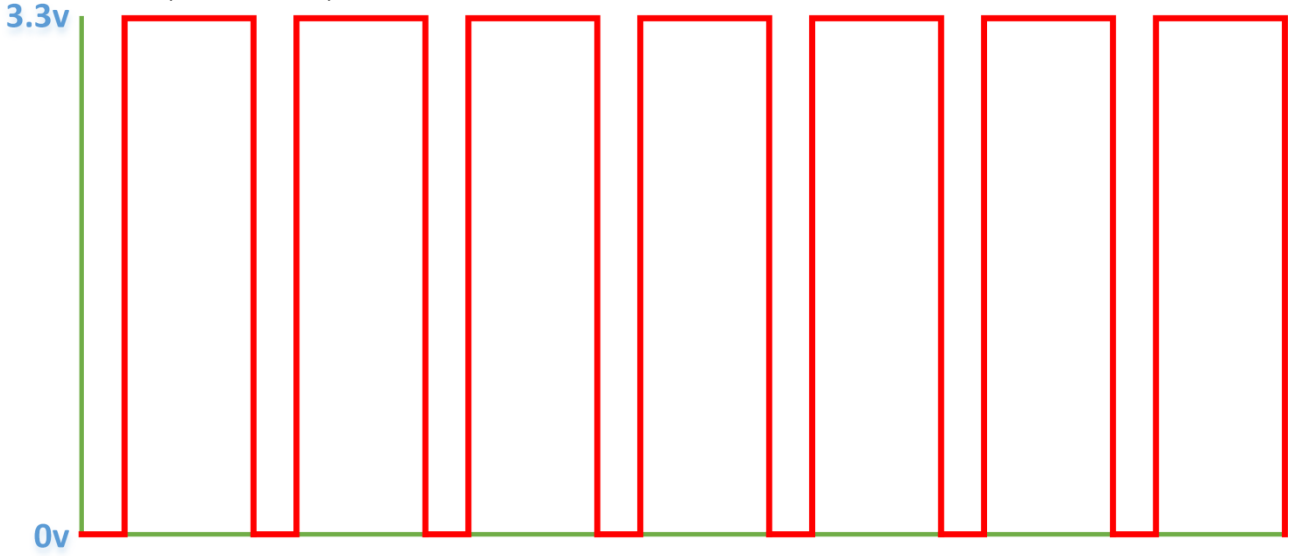


50 Hertz with 50% duty cycle

You can see that the signal is high (3.3v or 3V3) half of the time. A multimeter will let you measure the duty cycle if you set the meter to volts. A 50% duty cycle should read 50% of 3.3 Volts = 1.65V, although it might fluctuate a bit.

Example - Frequency 50 Hz, duty cycle 75%

This one has a duty cycle of 75%, so the 3.3v pulse is "High" 75% of the time and the average overall voltage delivered is 2.5v (75% of 3.3v).



50 Hz with 75% duty cycle

If duty cycle is 100 or 0, frequency becomes irrelevant as the signal is either completely on or completely off.

So how does this help you? Well, by supplying less power to the motors, you will be able to slow them down, and therefore make the robot more controllable.

Code

You are going to edit the code you wrote in Worksheet 4 to make the motor speed changeable.

Copy the code from that worksheet and edit the new copy using the following in a terminal window:

```
cd ~/EduKitRobotics
cp 4-driving.py 7-pwm.py
nano 7-pwm.py
```

Edit the code to make it look like the following:

```
# CamJam EduKit 3 - Robotics
# Worksheet 7 – Controlling the motors with PWM

import RPi.GPIO as GPIO # Import the GPIO Library
import time # Import the Time library

# Set the GPIO modes
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Set variables for the GPIO motor pins
pinMotorAForwards = 10
pinMotorABackwards = 9
pinMotorBForwards = 8
pinMotorBBackwards = 7

# How many times to turn the pin on and off each second
Frequency = 20
# How long the pin stays on each cycle, as a percent (here, it's 30%)
DutyCycle = 30
# Setting the duty cycle to 0 means the motors will not turn
Stop = 0

# Set the GPIO Pin mode to be Output
GPIO.setup(pinMotorAForwards, GPIO.OUT)
GPIO.setup(pinMotorABackwards, GPIO.OUT)
GPIO.setup(pinMotorBForwards, GPIO.OUT)
GPIO.setup(pinMotorBBackwards, GPIO.OUT)

# Set the GPIO to software PWM at 'Frequency' Hertz
pwmMotorAForwards = GPIO.PWM(pinMotorAForwards, Frequency)
pwmMotorABackwards = GPIO.PWM(pinMotorABackwards, Frequency)
pwmMotorBForwards = GPIO.PWM(pinMotorBForwards, Frequency)
pwmMotorBBackwards = GPIO.PWM(pinMotorBBackwards, Frequency)

# Start the software PWM with a duty cycle of 0 (i.e. not moving)
pwmMotorAForwards.start(Stop)
pwmMotorABackwards.start(Stop)
pwmMotorBForwards.start(Stop)
pwmMotorBBackwards.start(Stop)

# Turn all motors off
def StopMotors():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)
```

```
# Turn both motors forwards
def Forwards():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycle)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycle)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn both motors backwards
def Backwards():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycle)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycle)

# Turn left
def Left():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycle)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycle)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn Right
def Right():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycle)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycle)

# Your code to control the robot goes below this line
Forwards()
time.sleep(1) # Pause for 1 second

Left()
time.sleep(0.5) # Pause for half a second

Forwards()
time.sleep(1)

Right()
time.sleep(0.5)

Backwards()
time.sleep(0.5)

StopMotors()

GPIO.cleanup()
```

Once complete, use “Ctrl + x” then “y” then “enter” to save the file.

Running the Code

Type the following into the terminal window to run the code:

```
python3 7-pwm.py
```

You should see that the robot is moving at about a third of the speed that it was in Worksheet 4.

Making the Robot Move in a Straight Line

You may find that your robot does not drive in an exact straight line. That is because the motors are not precision devices, and there will always be a variation in them that will affect how fast they go. However, using different PWM for each motor, you will be able to vary the speed of each motor to get them to match a little better.

Once again, edit your code (saving the original):

```
cd ~/EduKitRobotics
cp 7-pwm.py 7-pwm2.py
nano 7-pwm2.py
```

Replace the `DutyCycle` variable with two, one for each motor:

```
DutyCycleA = 30
DutyCycleB = 30
```

Then, replace every occurrence of `DutyCycle` with either `DutyCycleA` or `DutyCycleB`, depending on whether the line of code is controlling motor A or B.

Your code should look like this:

```
# CamJam EduKit 3 - Robotics
# Worksheet 7 – Varying the speed of each motor with PWM

import RPi.GPIO as GPIO # Import the GPIO Library
import time # Import the Time library

# Set the GPIO modes
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Set variables for the GPIO motor pins
pinMotorAForwards = 10
pinMotorABackwards = 9
pinMotorBForwards = 8
pinMotorBBackwards = 7

# How many times to turn the pin on and off each second
Frequency = 20
# How long the pin stays on each cycle, as a percent
DutyCycleA = 30
DutyCycleB = 30
# Setting the duty cycle to 0 means the motors will not turn
Stop = 0

# Set the GPIO Pin mode to be Output
GPIO.setup(pinMotorAForwards, GPIO.OUT)
GPIO.setup(pinMotorABackwards, GPIO.OUT)
GPIO.setup(pinMotorBForwards, GPIO.OUT)
GPIO.setup(pinMotorBBackwards, GPIO.OUT)

# Set the GPIO to software PWM at 'Frequency' Hertz
pwmMotorAForwards = GPIO.PWM(pinMotorAForwards, Frequency)
pwmMotorABackwards = GPIO.PWM(pinMotorABackwards, Frequency)
pwmMotorBForwards = GPIO.PWM(pinMotorBForwards, Frequency)
pwmMotorBBackwards = GPIO.PWM(pinMotorBBackwards, Frequency)
```

```
# Start the software PWM with a duty cycle of 0 (i.e. not moving)
pwmMotorAForwards.start(Stop)
pwmMotorABackwards.start(Stop)
pwmMotorBForwards.start(Stop)
pwmMotorBBackwards.start(Stop)

# Turn all motors off
def StopMotors():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn both motors forwards
def Forwards():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycleB)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn both motors backwards
def Backwards():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycleB)

# Turn left
def Left():
    pwmMotorAForwards.ChangeDutyCycle(Stop)
    pwmMotorABackwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorBForwards.ChangeDutyCycle(DutyCycleB)
    pwmMotorBBackwards.ChangeDutyCycle(Stop)

# Turn Right
def Right():
    pwmMotorAForwards.ChangeDutyCycle(DutyCycleA)
    pwmMotorABackwards.ChangeDutyCycle(Stop)
    pwmMotorBForwards.ChangeDutyCycle(Stop)
    pwmMotorBBackwards.ChangeDutyCycle(DutyCycleB)

# Your code to control the robot goes below this line
Forwards()
time.sleep(1) # Pause for 1 second

Left()
time.sleep(0.5) # Pause for half a second

Forwards()
time.sleep(1)

Right()
time.sleep(0.5)

Backwards()
time.sleep(0.5)
```

```
StopMotors()

GPIO.cleanup()
```

Now run your code with:

```
python3 7-pwm2.py
```

Your robot will still not be going in a straight line as both duty cycles are still the same.

Calibration

You now need to experiment, changing one or both duty cycle variables until your robot goes in a straight line. You can use decimal numbers too (e.g. `DutyCycleA = 27.6`).

Challenges

1. Try adding more functions to your robot so that it can drive at different speeds.
2. Change the sleep times when turning left and right to make your robot turn 90° at a time.