Raspberry Pi part nº 7

les recettes de tante MLI

Tony Dixon
(Royaume-Uni)

Après avoir passé en revue de nombreux signaux numériques (GPIO, UART série, SPI et I²C) et manipulé quelques signaux analogiques (via SPI), nous allons voir ici comment exploiter la modulation de largeur d'impulsion avec un Raspberry Pi.

Interfaces matérielles MLI

La modulation de largeur d'impulsion (MLI, ou PWM en anglais) consiste à faire varier la largeur des impulsions d'un signal rectangulaire. Cette modulation entraîne une variation de la valeur moyenne que représente la forme d'onde. La MLI sert principalement à contrôler la puissance délivrée à des dispositifs électriques comme des lampes ou des moteurs. Le Raspberry Pi est équipé d'un BCM2835 de chez Broadcom. Ce système sur puce possède deux canaux MLI/PWM : un fournit l'audio, l'autre, PWM0, est libre et se trouve sur la broche 12 (GPIO18) du connecteur d'extension (**tableau 1**).

Gradateur de LED

Montrons comment utiliser ce canal pour contrôler la luminosité d'une LED par MLI. La LED est connectée entre la masse et PWM0 (GPIO18/broche 12) avec une résistance-talon de 330 Ω.

Nous pourrions être tentés de lancer Python pour profiter de la bibliothèque de bas niveau *RPi.GPIO*, mais étrangement elle ne contient aucune méthode d'accès à l'interface matérielle PWM0. Nous allons donc installer *wiringPi*, une bibliothèque C de bas niveau pour Raspberry Pi développée par Gordon Henderson [1]. Similaire à Wiring, qui forme

le cœur logiciel des entrées/sorties d'Arduino, *wiringPi* peut être utilisée avec de nombreux langages, dont Python.

Commencez par télécharger l'installateur de paquets Python appelé PiP (*Python Package Installer*) :

```
sudo apt-get install python-dev
python-pip
```

Pour installer *wiringPi* il suffit alors de taper :

```
sudo pip install wiringpi2
```

Lancez l'environnement IDLE pour Python, et copiez dans l'éditeur le code du **listage 1**. Ce programme définit d'abord le port GPIO18/PWM0 comme sortie numérique. Il entre ensuite dans une boucle où la valeur du registre PWM est incrémentée à chaque passage. La luminosité de la LED augmente ainsi en même temps que le signal MLI.

Interfaces logicielles MLI

Nous avons une broche MLI à disposition, très bien, mais comment utiliser la MLI sur plus d'un signal ? Nous pourrions recourir aux canaux supplémentaires qu'offrent certaines puces, comme la PCA9685 de Texas Instruments (16 canaux sur 12 bits), mais il y a moins coûteux : la MLI logicielle.

Notez toutefois que si la MLI logicielle ne pose habituellement pas de problème sur une plateforme embarquée comme Arduino, où le processeur n'exécute que le code de ses applications, il en va tout autrement sur un ordinateur non spécialisé comme le Raspberry Pi, où un système d'exploitation complet jongle simultanément avec différentes tâches et programmes. Cette complexité supplémentaire ne permet pas de garantir que la MLI « purement logicielle » sera exempte d'infortunes, p. ex. une certaine gigue ou une baisse de la résolution attendue, puisqu'à tout moment le SE peut suspendre l'exécution du programme MLI au profit d'un autre. Un vrai problème pour qui a besoin d'une précision bien définie et d'une faible gigue. Le Pi dispose heureusement de plusieurs canaux d'accès direct à la mémoire (DMA). Ce sont eux que nous utiliserons pour la gestion temporelle de la MLI : comme le DMA fonctionne indépen-

Tableau 1. Brochage du connecteur d'extension

nom de la broche	fonction	autre fonction	RPi.GPIO
P1-02	5,0V	-	-
P1-04	5,0V	-	-
P1-06	GND	-	-
P1-08	GPIO14	UART0_TXD	RPi.GPIO8
P1-10	GPIO15	UART0_RXD	RPi.GPIO10
P1-12	GPIO18	**PWM0**	RPi.GPIO12
P1-14	GND	-	-
P1-16	GPIO23		RPi.GPIO16
P1-18	GPIO24		RPi.GPIO18
P1-20	GND	-	-
P1-22	GPIO25		RPi.GPIO22
P1-24	GPIO8	SPI0_CE0_N	RPi.GPIO24
P1-26	GPIO7	SPI0_CE1_N	RPi.GPIO26

nom de la broche	révision 1 de la carte		révision 2 de la carte	
	fonction	autre fonction	fonction	autre fonction
P1-01	3,3V	-	3,3V	-
P1-03	GPIO0	I2C0_SDA	GPIO2	I2C1_SDA
P1-05	GPIO1	I2C0_SCL	GPIO3	I2C1_SCL
P1-07	GPIO4	GPCLK0	GPIO4	GPCLK0
P1-09	GND	-	GND	-
P1-11	GPIO17	RTS0	GPIO17	RTS0
P1-13	GPIO21		GPIO27	
P1-15	GPIO22		GPIO22	
P1-17	3,3V	-	3,3V	-
P1-19	GPIO10	SPI0_MOSI	GPIO10	SPI0_MOSI
P1-21	GPIO9	SPI0_MISO	GPIO9	SPI0_MISO
P1-23	GPIO11	SPI0_SCLK	GPIO11	SPI0_SCLK
P1-25	GND	-	GND	-

Note : I2C0_SDA et I2C0_SCL (GPIO0 & GPIO1), ainsi que I2C1_SDA et I2C1_SCL (GPIO2 & GPIO3) sont dotées de résistances de rappel de 1,8 kΩ au 3,3 V.

Listage 1 : gradateur de LED

```
#!/usr/bin/python

import wiringpi2 as gpio
import time

#set up gpio
gpio.wiringpiPiSetupGpio ()
gpio.pinMode (18,2)

while True:
  gpio.pwmWrite (18,0)
  for n in range (0,1024):
    gpio.pwmWrite (18,n)
    time.sleep (0.01)
```

damment du processeur, les caractéristiques temporelles de la MLI reposeront cette fois-ci sur le matériel, seront donc moins sensibles aux interruptions du SE, et la précision et la gigue seront acceptables.

Commande de moteurs à CC

La MLI sert également à commander et contrôler la vitesse de rotation des moteurs électriques.

Le pont en H représenté sur la **figure 1** permet de faire tourner un moteur dans les deux sens, et si nous appliquons un signal MLI à l'un des quadrants de commutation, nous pouvons de surcroît aussi contrôler sa vitesse.

Un robot pour la cuisine Raspberry Pi

PiiBOT (**fig. 2**) est un robot à quatre roues qui se commande à distance à l'aide d'une manette Nintendo Wii. L'engin embarque deux puces L293D chargées de piloter quatre petits moteurs à CC avec balais, ainsi qu'un adaptateur USB-Bluetooth pour la réception des instructions envoyées par la manette Wii.

Le circuit

On retrouve ces deux pilotes L293D sur le schéma de la carte d'extension Pi [2] de commande du moteur (**fig. 3**). Le L293D est d'ailleurs utilisé dans de nombreux projets com-

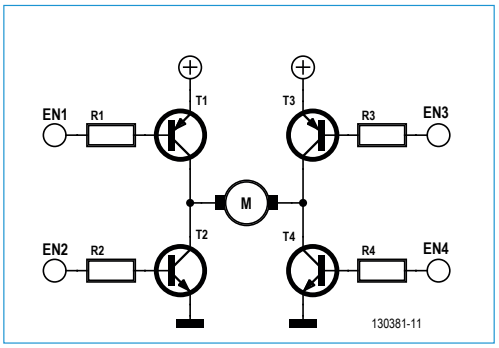


Figure 1.
Schéma du pont en H.

Figure 2. PiiBOT.

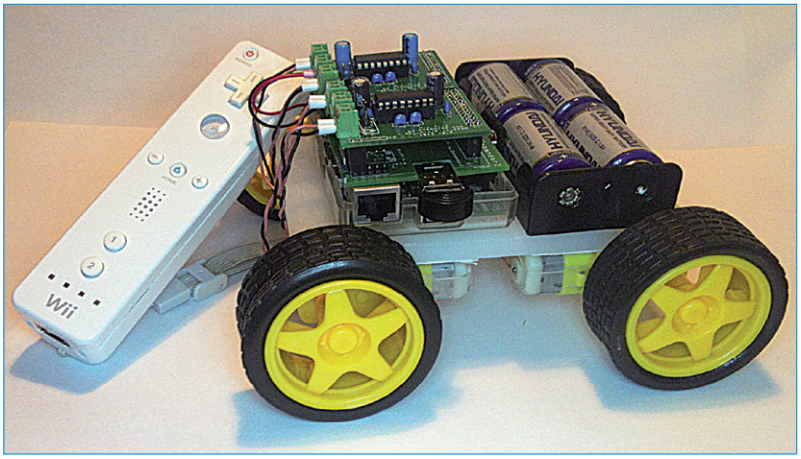
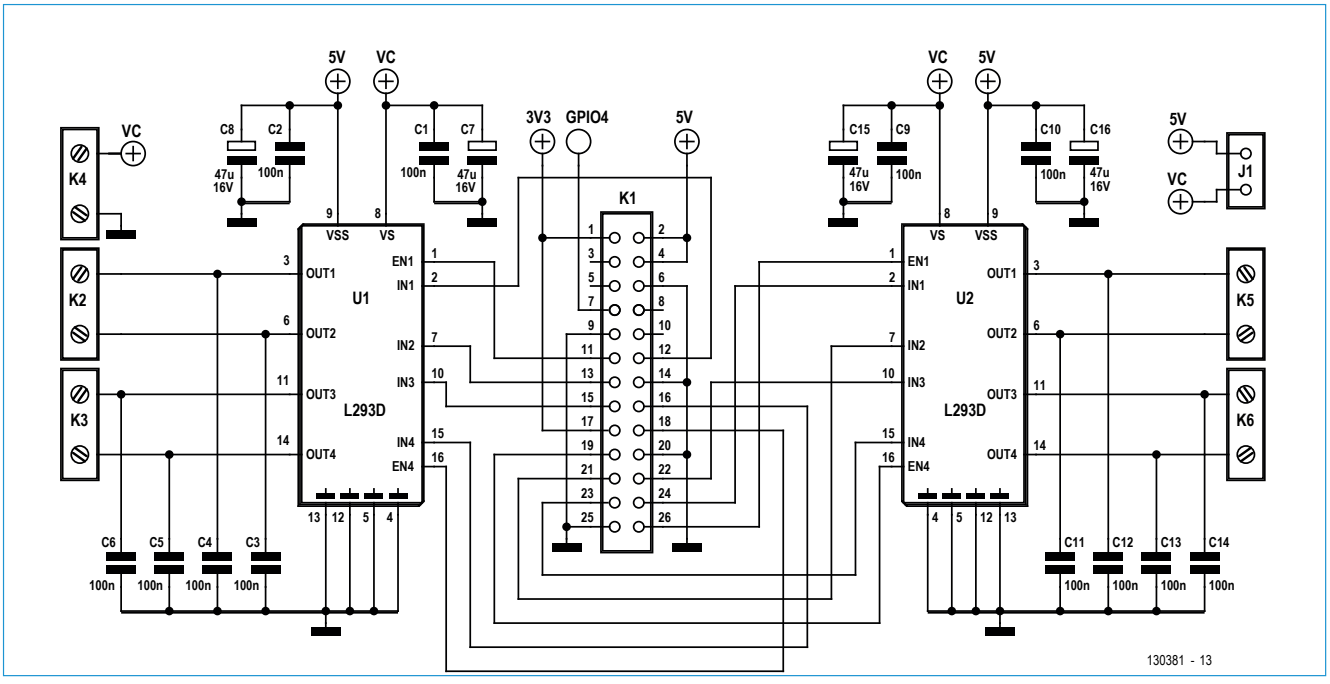


Figure 3.
Schéma de la carte d'extension pour la commande du moteur de PiiBOT.



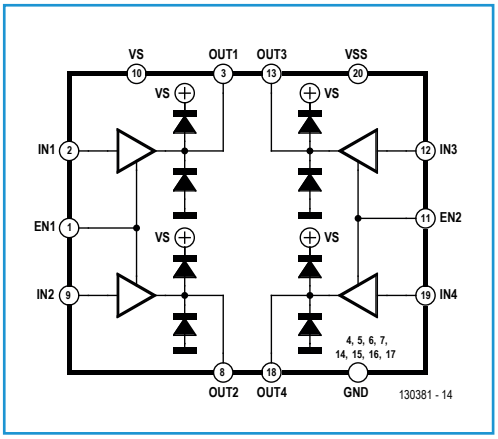


Figure 4. Le pilote de moteur L293D.

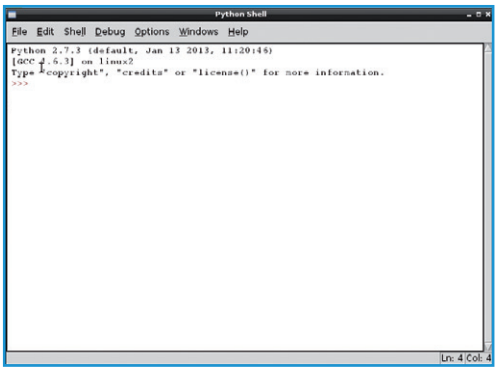


Figure 5. L'interpréteur de commandes Python.

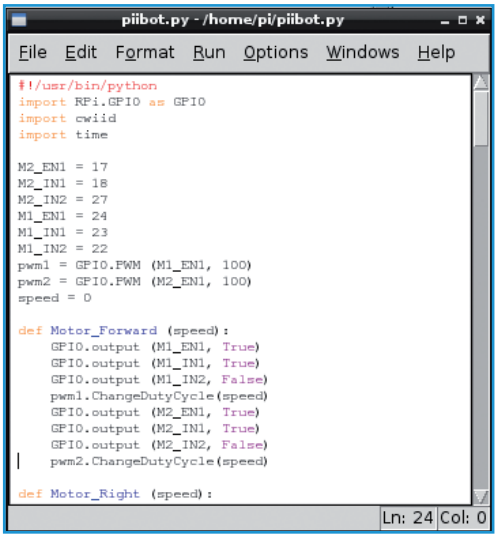


Figure 6. Le code de *piibot.py* dans l'éditeur IDLE.

Tableau 2. Fonction des GPIO

Moteurs 1 et 2 (U1 L293D)		Moteurs 3 et 4 (U2 L293D)	
Fonction	GPIO	Fonction	GPIO
EN1	GPIO17	EN1	GPIO7
IN1	GPIO18	IN1	GPIO8
IN2	GPIO27	IN2	GPIO9
EN2	GPIO24	EN2	GPIO10
IN3	GPIO22	IN3	GPIO25
IN4	GPIO23	IN4	GPIO11

portant un petit moteur. Il comporte deux ponts en H pouvant chacun délivrer 600 mA CC (1,2 A crête) et il est très simple à interfcer. Comme on le voit sur la **figure 4**, chaque pont en H possède un signal *enable* (EN1/2) et deux signaux de commande (entrées directionnelles IN1/3 et IN2/4).
Il est possible d'utiliser l'alimentation de 5 V du Pi via le cavalier J1, mais une alimentation externe (4,5 à 36 V CC) délivrée via le connecteur K4 est recommandée.

Installation des pilotes Bluetooth et de la bibliothèque Python CWii

Nous devons installer les pilotes de l'adaptateur USB-Bluetooth avant de pouvoir jouer avec la manette Nintendo Wii. La communication Bluetooth est ici basique, nous pouvons donc passer l'option `--no-install-recommends` :

```
sudo apt-get install --no-install-recommends bluetooth
```

Le module USB-Bluetooth enfiché dans le Pi, testez l'interface avec :

```
sudo service bluetooth status
```

La réponse devrait être, si l'installation est correcte :

```
[ ok ] bluetooth is running.
```

Nous pouvons alors télécharger la bibliothèque Python *CWii* [4] avec :

```
sudo apt-get install python-cwiid
```

Une fois l'ensemble installé, vous devriez pouvoir commander le PiiBOT avec la manette Wii. Le lien [5] explique comment lire et interpréter les données envoyées par une manette Wii via Bluetooth.

Six signaux GPIO par puce L293D, soit 12 au total, sont nécessaires pour commander PiiBOT (**tableau 2**).
Pour commander la vitesse du moteur, nous allons moduler la largeur d'impulsion des quatre signaux *enable* de chaque pont, via quatre canaux MLI logiciels.

Programme d'exemple : piibot.py

Le programme Python importe des pilotes et bibliothèques indispensables au contrôle de la manette Nintendo Wii. Suivez les instructions de **l'encadré** pour les installer.
Double-cliquez sur l'icône IDLE du bureau pour lancer l'EDI et la console Python (**fig. 5**). Dans le menu *File*, sélectionnez *New Window* et entrez le **listage 2** (**fig. 6**). Le code est long, vous préférerez sans doute le télécharger depuis la page Elektor.LABS associée à cet article [3].
Sauvegardez le programme saisi et rendez-le exécutable. Pour cela retournez dans l'émulateur LXTerminal, et entrez la commande suivante :

```
chmod +x piibot.py
```

Et pour exécuter le programme :

```
sudo ./piibot.py
```

Le programme vous demandera d'apparier le contrôleur Wii et le Pi. Pour les associer, appuyez simultanément sur les boutons 1 et 2 de la manette Wii. Une fois le pairage effectué, le PiiBOT n'attend plus que son pilote… humain !

(130381 – version française : Hervé Moreau)

Liens

[1] Bibliothèque GPIO WiringPi : http://wiringpi.com

[2] Carte d'extension MiniPiio Motor293D : www.dtronixs.com

[3] Page Raspberry Pi de l'Elektor.LABS : www.elektor-labs.com/RPi

[4] Bibliothèque CWiid pour le contrôleur Nintendo Wii : http://abstrakraft.org/cwiid/

[5] Nintendo Wii Remote, Python and The Raspberry Pi : http://bit.ly/RPi-Wii

Listage 2 : piibot.py (Téléchargez le code depuis [3] !)

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import cwiid
import time

M1_EN1 = 24
M1_IN1 = 23
M1_IN2 = 22
M2_EN1 = 17
M2_IN1 = 18
M2_IN2 = 27
M3_EN1 = 7
M3_IN1 = 8
M3_IN2 = 9
M4_EN1 = 10
M4_IN1 = 25
M4_IN2 = 11

speed = 40
```

```
def Motor_Setup ():
    print 'Setting up..'
    GPIO.setwarnings (False)

    # Configure GPIO
    GPIO.setmode (GPIO.BCM)
    GPIO.setup (M1_EN1, GPIO.OUT)
    GPIO.setup (M1_IN1, GPIO.OUT)
    GPIO.setup (M1_IN2, GPIO.OUT)
    GPIO.setup (M2_EN1, GPIO.OUT)
    GPIO.setup (M2_IN1, GPIO.OUT)
    GPIO.setup (M2_IN2, GPIO.OUT)
    GPIO.setup (M3_EN1, GPIO.OUT)
    GPIO.setup (M3_IN1, GPIO.OUT)
    GPIO.setup (M3_IN2, GPIO.OUT)
    GPIO.setup (M4_EN1, GPIO.OUT)
    GPIO.setup (M4_IN1, GPIO.OUT)
    GPIO.setup (M4_IN2, GPIO.OUT)

    print "ready"
```

```
def Motor_Forward (speed):
    print 'Forward, speed = ', speed
    GPIO.output (M1_IN1, True)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(speed) # M1 EN1
    GPIO.output (M2_IN1, True)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(speed) # M2 EN1
    GPIO.output (M3_IN1, True)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(speed) # M3 EN1
    GPIO.output (M4_IN1, True)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(speed) # M4 EN1

def Motor_Right (speed):
    print 'Right, speed = ', speed
    GPIO.output (M1_IN1, True)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(speed)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(0)
    GPIO.output (M3_IN1, True)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(speed)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(0)

def Motor_Left (speed):
    print 'Left, speed = ', speed
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(0)
    GPIO.output (M2_IN1, True)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(speed)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(0)
    GPIO.output (M4_IN1, True)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(speed)

def Motor_Reverse (speed):
    print 'Reverse, speed = ', speed
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, True)
    pwm1.ChangeDutyCycle(speed)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, True)
    pwm2.ChangeDutyCycle(speed)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, True)
    pwm3.ChangeDutyCycle(speed)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, True)
    pwm4.ChangeDutyCycle(speed)

def Motor_Stop ():
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(0)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(0)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(0)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(0)


# Main Program
Motor_Setup ()
pwm1 = GPIO.PWM (M1_EN1, 100)
pwm2 = GPIO.PWM (M2_EN1, 100)
pwm3 = GPIO.PWM (M3_EN1, 100)
pwm4 = GPIO.PWM (M4_EN1, 100)
pwm1.start (0)
pwm2.start (0)
pwm3.start (0)
pwm4.start (0)

Motor_Stop ()

button_delay = 0.1

print 'Press 1 + 2 on your Wii Remote'
time.sleep(1)

# Connect to the Wii Remote
try:
    wii=cwiid.Wiimote()
except RuntimeError:
```

```
    print ‘Error Connecting to Wii Remote’
    quit()

print ‘Wii Remote connected’

wii.rpt_mode = cwiid.RPT_BTN


# Loop
try:
    while True:

        buttons = wii.state[‘buttons’]

        stop = True

        if (buttons & cwiid.BTN_LEFT):
            Motor_Left (speed)
            time.sleep(button_delay)
            stop = False

        if(buttons & cwiid.BTN_RIGHT):
            Motor_Right(speed)
            time.sleep(button_delay)
            stop = False

        if (buttons & cwiid.BTN_UP):
            Motor_Forward (speed)
            time.sleep(button_delay)
            stop = False

        if (buttons & cwiid.BTN_DOWN):
            Motor_Reverse (speed)
            time.sleep(button_delay)
            stop = False


        if (buttons & cwiid.BTN_MINUS):
            speed = speed - 1
            if (speed < 0):
                speed = 0
            print ‘speed =’, speed
            time.sleep(button_delay)

        if (buttons & cwiid.BTN_PLUS):
            speed = speed + 1
            if (speed > 100):
                speed = 100
            print ‘speed =’, speed
            time.sleep(button_delay)

        if (stop == True):
            Motor_Stop ()

except KeyboardInterrupt:
    pass

print “End”
GPIO.output (M1_EN1, False)
GPIO.output (M1_IN1, False)
GPIO.output (M1_IN2, False)
GPIO.output (M2_EN1, False)
GPIO.output (M2_IN1, False)
GPIO.output (M2_IN2, False)
GPIO.output (M3_EN1, False)
GPIO.output (M3_IN1, False)
GPIO.output (M3_IN2, False)
GPIO.output (M4_EN1, False)
GPIO.output (M4_IN1, False)
GPIO.output (M4_IN2, False)

pwm1.stop ()
pwm2.stop ()
pwm3.stop ()
pwm4.stop ()

GPIO.cleanup
exit (wii)
```