Raspberry Pi Rezepte Teil 7

PWM – hart oder weich gekocht?

Von **Tony Dixon** (UK)

Beim bisherigen Kochen und Braten ging es hauptsächlich um digitale Signale wie GPIOs, serielle UARTs, SPI und I²C. Hinzu kam etwas Würze durch analoge Signale via SPI. In dieser Folge wird die äußerst wichtige Zutat PWM hin und her gewendet.

PWM-Hardware

Bei PWM (PulseWidth-Modulation) handelt es sich um ein Rechtecksignal, dessen Pulsbreite verändert (= moduliert) wird. Eine veränderte Pulsbreite sorgt für einen veränderten Mittelwert des Signals. PWM wird daher hauptsächlich zur verlustarmen Steuerung der Leistung von Lampen, Motoren etc. eingesetzt.
Ein SoC des Typs BCM2835 von Broadcom ist der zentrale Chip eines RPi. Er ist mit zwei hardware-basierten PWM-Kanälen ausgestattet. Einer davon wird vom System für die Audio-Ausgabe benutzt. Der Kanal PWM0 bleibt frei und findet sich auch an RPis Expansion Header an Pin 12 als GPIO18 (siehe **Tabelle 1**).

LEDs dimmen

Mit PWM0 kann man schön die Helligkeit einer LED einstellen. Zur Demonstration braucht man die LED lediglich mit einem 330-Ω-Vorwiderstand zwischen PWM0 (GPIO18/Pin 12) und Masse schalten.

An diesem Punkt sollte man eigentlich Python anfeuern und die Sache per RPi.GPIO-Low-Level-Library gar werden lassen, doch überraschenderweise verfügt RPi.GPIO über keine

Methode, um PWM-Hardware anzusprechen. Vor der Demonstration der PWM-Hardware muss daher „wiringPi“ von Gordon Henderson [1] installiert werden. Bei wiringPi handelt es sich um eine C-basierte Low-Level-Library für RPi, die von vielen Programmiersprachen - auch von Python - verwendet werden kann. Die Library ähnelt der Software-Plattform für Arduino „wiring“ sehr.
Und es gibt noch mehr Software zu installieren. Zuerst PiP (Python Package Installer):

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

Anschließend holt man sich wiringPi für Python aus dem Netz und tippt:

```
sudo pip install wiringpi2
```

Nun startet man IDLE, die Python-IDE und gibt den Code von **Listing 1** ein.
Dieses Python-Programm definiert GPIO18/PWM0 als digitalen Ausgang. Dann springt es in eine Schleife, in der das Tastverhältnis von PWM0 erhöht wird. Mit diesem Wert wird die LED heller.

PWM-Software

RPi stellt also ein Hardware-PWM-Signal am Expansion Header zur Verfügung. Und was ist, wenn man mehr als ein PWM-Signal braucht? Man könnte dann ein zusätzliches IC wie den 16-kanaligen 12-bit-PWM-Chip PCA9685 von Texas Instruments einsetzen. Man kann aber auch mit Software zaubern - das kostet nichts außer etwas Code.

Eine kleine Warnung: Normalerweise ist PWM per Software auf Embedded-Plattformen wie Arduino recht problemarm, da die CPU hier lediglich mit dem Anwendungsprogramm beschäftigt ist. Auf „richtigen Computern“ wie RPi aber läuft ein richtiges Betriebssystem, das quasigleichzeitig mit vielen verschiedenen Aufgaben jongliert. Deshalb muss man befürchten, dass eine Software-PWM viel Jitter und nur geringe Auflösungen bietet, da die CPU zwischendurch immer wieder andere Aufgaben wichtiger nimmt. Braucht man hohe Auflösungen und niedrigen Jitter, hat man ein Problem. Glücklicherweise verfügt RPi über genug DMA-Kanäle (Direct Memory Access), wodurch die Software durch Hardware-Timing

Tabelle 1. Pin-Belegung des Expansion Headers

Pin-Name	Pin-Funktion	Alternative	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

Pin-	Board Revision 1		Board Revision 2	
Name	Pin-Funktion	Alternative	Pin-Funktion	Alternative
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	-

Achtung: I2C0_SDA und I2C0_SCL (GPIO0 & GPIO1) sowie I2C1_SDA und I2C1_SCL (GPIO2 & GPIO3) haben 1,8-kΩ-Pull-up-Widerstände gegen 3,3 V.

Listing 1. LED-Dimmer

```
#!/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)
```

unterstützt wird. Da DMA unabhängig von der CPU ist, bekommt man auf diese Weise ein Hardware-Timing, das weniger durch CPU-Interrupts gestört wird und daher akzeptable PWM-Präzision und wenig Jitter ermöglicht.

Motor-Steuerung

Eine gängige PWM-Anwendung ist die Steuerung der Drehzahl eines Elektromotors. Die H-Brücke von **Bild 1** erlaubt es sogar, den Motor in beiden Richtungen zu betreiben. Mit einem passenden PWM-Signal kann man nicht nur die Drehzahl ändern, sondern sogar die Drehrichtung umschalten.

PiiBOT rollen lassen

PiiBOT ist ein Roboterfahrzeug mit Vierradantrieb, das mit einem Nintendo-Wii-Controller ferngesteuert wird (siehe **Bild 2**). Für Bewegung sorgen Motor-Controller-Chips vom Typ L293D, welche die vier kleinen DC-Motoren treiben. Ein Bluetooth-USB-Dongle empfängt die Befehle vom Wii-Controller.

Schaltung

Bild 3 zeigt die Schaltung des Motor-Controller-Shields für RPi [2]. Zwei L293D-Chips sind vorgesehen. Das IC L293D wird in vielen Projekten mit kleinen Gleichstrommotoren eingesetzt. Es enthält eine volle H-Brücke, die einen Strom von 600 mA bei einem Spitzenwert von 1,2 A verkraftet und sehr einfach anzusteuern ist. Jede H-Brücke verfügt über einen Enable-Eingang (EN1/2) und zwei Ein-

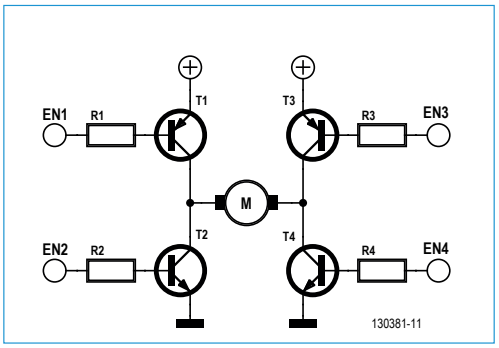

Bild 1.
Schaltung einer H-Brücke.

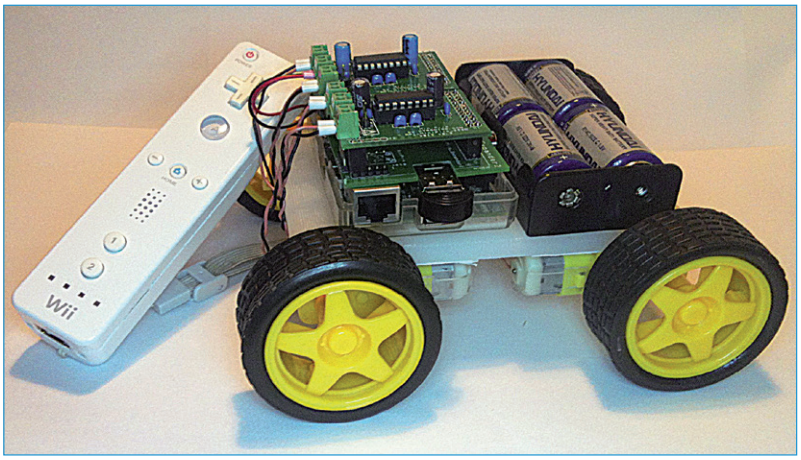

Bild 2.
PiiBOT, ein einfaches Fahrzeug mit Vierradantrieb.

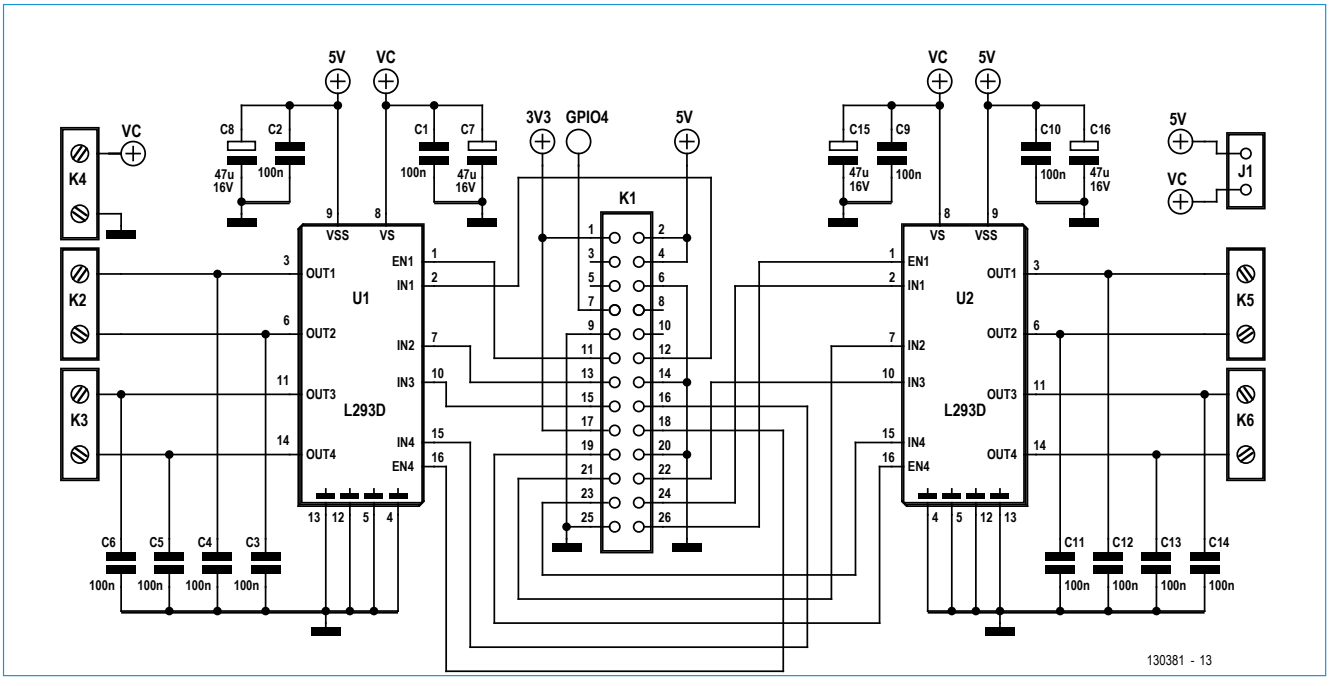

Bild 3.
Schaltung des Motor-Controller-Shields für den PiiBOT.

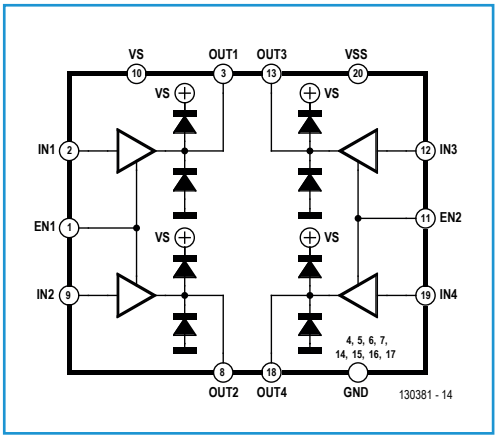

Bild 4. Motor-Controller-IC L293D.

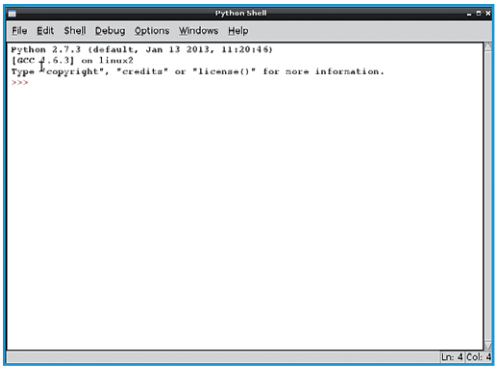

Bild 5. Python-Shell.

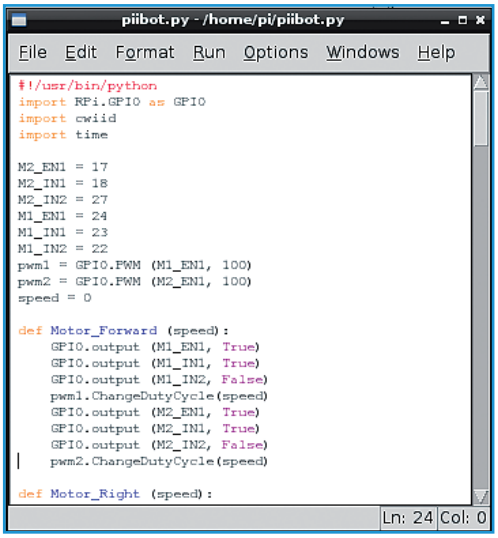

Bild 6. IDLE-Editor mit dem Skript „piibot.py“.

Tabelle 2. GPIO-Verwendung

Motoren 1 und 2 (U1 L293D)		Motoren 3 und 4 (U2 L293D)	
Motorfunktion	**GPIO**	**Motorfunktion**	**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

gänge (IN1/3 und IN2/4) zur Steuerung der Drehrichtung (siehe **Bild 4**).

An K4 kann ein externes Netzteil mit einer Spannung von 4,5 V bis 36 V angelegt werden. Alternativ lässt sich die Stromversor-

Installation der Bluetooth-Treiber und der CWii-Library

Zum Einsatz des Nintendo-Wii-Controllers muss man die Bluetooth-Treiber für das USB-Bluetooth-Modul installieren. Da nur grundlegende Bluetooth-Kommunikation erforderlich ist, reicht die Option „`--no-install-recommends`“ bei der Befehlseingabe:

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

Mit beim RPi eingestecktem Bluetooth-Dongle testet man so das Interface:

```
sudo service bluetooth status
```

Wenn alles in Ordnung ist, sollte folgende Antwort kommen:

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

Nach der Installation der Bluetooth-Treiber kann man sich die Python-Wii-Library CWii [4] aus dem Netz holen. Hierzu tippt man den Befehl:

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

Nachdem nun alles installiert ist, kann man RPi (oder genauer: PiiBOT) per Wii-Controller fernsteuern. Wen interessiert, wie man die Daten dieses Controllers ausliest, für den sind die Informationen von [5] hilfreich.

gung über den internen Jumper J1 von der 5-V-Versorgung von RPi abzwacken. Besser aber verwendet man ein externes Netzteil via K4.
Für ein L293D sind sechs GPIOs notwendig. Insgesamt werden also 12 GPIO-Signale zur Steuerung der vier Motoren des Roboters benötigt. **Tabelle 2** gibt einen Überblick über die benötigten Signale.

Zur Steuerung der Motordrehzahl wird ein PWM-Signal an den Enable-Eingang jeder H-Brücke gelegt. Da hier vier PWM-Signale für vier Enable-Eingänge benötigt werden, ist Software-PWM angesagt.

Beispiel: „piibot.py“
Nachdem die Hardware steht, geht es zum zugehörigen Steuerprogramm in Python. Bevor man loslegt, müssen zunächst noch die Treiber und die Libraries für den Nintendo-Wii-Controller installiert werden. Die Anleitung dazu findet sich im Kasten.

Nun wird das IDLE-Icon auf dem Desktop von RPi doppelgeklickt, um damit die Shell und die IDE von Python zu starten (**Bild 5**).

Mit der Menü-Option „File“ erzeugt man ein neues Programm. Hierdurch startet der IDLE-Editor (**Bild 6**), mit Hilfe dessen das Programm von **Listing 2** eingegeben wird. Aufgrund seiner Länge kann man es auch bequem von der Elektor.LABS-Webseite zu dieser Artikelserie [3] herunterladen.

Nachdem dies alles erledigt ist, sollte man das neue Programm auch sichern, bevor man LX Terminal startet. Durch den folgenden Befehl wird das Programm ausführbar:

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

Anschließend startet man das Programm mit:

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

Wenn das Programm läuft, wird man zum Pairing des Wii-Controllers mit RPi aufgefordert. Hierzu muss man gleichzeitig die Tasten 1 und 2 am Wii-Controller betätigen. Nach dem Pairing ist PiiBOT startklar zum ausrollen!

(130381)

Weblinks

[1] WiringPi GPIO-Library: http://wiringpi.com

[2] MiniPiio Motor293D-Add-On-Board: www.dtronixs.com

[3] RPi-Support-Webseite @ Elektor.LABS: www.elektor-labs.com/RPi

[4] CWiid-Library für Nintendo-Wii-Controller: http://abstrakraft.org/cwiid/

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

Listing 2. „piibot.py“ (Download des Programms @ [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)
```