

Table des matières

I. Installation.....	1
1.1 Procédure.....	1
1.2 Tests.....	1
1.3 Réinitialisation du GrovePi.....	2
II. Description.....	2
2.1 Disposition des ports.....	2
2.2. Système de numérotation.....	3
2.3 API (Application Programming Interface) – Fonctions GPIO.....	4
2.4 Précaution.....	5
III. Projets.....	6
3.1 Allumer une Led.....	6
3.2 Led à intensité variable.....	8
3.3 Capteur de température.....	11
3.4 Capteur de température et d'humidité Pro.....	13
3.5 Capteur de distance par ultrasons.....	15
3.6 LedBar.....	17
3.7 Capteur de distance par ultrasons et ledBar.....	23
3.8 Convertisseur de lumière.....	25
3.10 Capteur de lumière.....	32
3.11 Détecteur de mouvement.....	34
3.12 Capteur de son.....	37
3.13 Écran LCD.....	39
3.14 Écran LCD avec capteur de température et d'humidité Pro.....	42
3.15 Capteur tactile.....	45
3.16 Relais.....	47
3.17 Analog Servo.....	49
3.17 Stepping motor.....	51



I. Installation.

1.1 Procédure.

Tuto en anglais :

<https://www.dexterindustries.com/GrovePi/get-started-with-the-grovepi/>

Bien penser au préalable à activer les différentes interfaces du raspberry !

```
sudo raspi-config
5 Interfacing Options
P5 I2C
P7 1-Wire
P8 Remote GPIO
```

GrovePi est une plate-forme open-source permettant de connecter des capteurs Grove au Raspberry Pi. Vous avez besoin d'un accès Internet pour la ou les étapes suivantes.

Le moyen le plus rapide d'installer GrovePi consiste à entrer la commande suivante :

```
curl -kL dexterindustries.com/update_grovepi | bash
```

```
sudo reboot
```

La même commande peut être utilisée pour mettre à jour une version plus récente de GrovePi.

Lors de l'installation de GrovePi, vous trouverez la bibliothèque GrovePi dans le répertoire :

```
cd /home/pi/Dexter/GrovePi/
```

Pour flasher le dernier firmware sur le GrovePi, exécutez :

```
cd ~/Dexter/GrovePi/Firmware
bash firmware_update.sh
```

1.2 Tests.

Pour exécuter la suite complète des tests du GrovePi, suivez ces instructions :

```
cd ~/Dexter/GrovePi/Troubleshooting
sudo bash all_tests.sh
```

À la fin de ce processus, vous obtiendrez un fichier **log.txt** sur votre bureau à l'adresse que vous pouvez éditer avec les commandes suivantes :

```
cd ~/Desktop/
sudo nano log.txt
```

Pour quitter :

```
Ctrl + X
```

De plus, pour voir avec quelle version du firmware la bibliothèque installée sur le Raspberry Pi fonctionne, vous pouvez aller dans le répertoire :

```
cd ~/Dexter/GrovePi/Software/Python
```

et exécuter :

```
python grovepi.py
```

Cela devrait générer un numéro de version (du firmware de GrovePi). Les anciennes versions du micrologiciel (<= v1.2.7) ne seront pas affichées lors de l'appel `python grovepi.py`.

1.3 Réinitialisation du GrovePi.

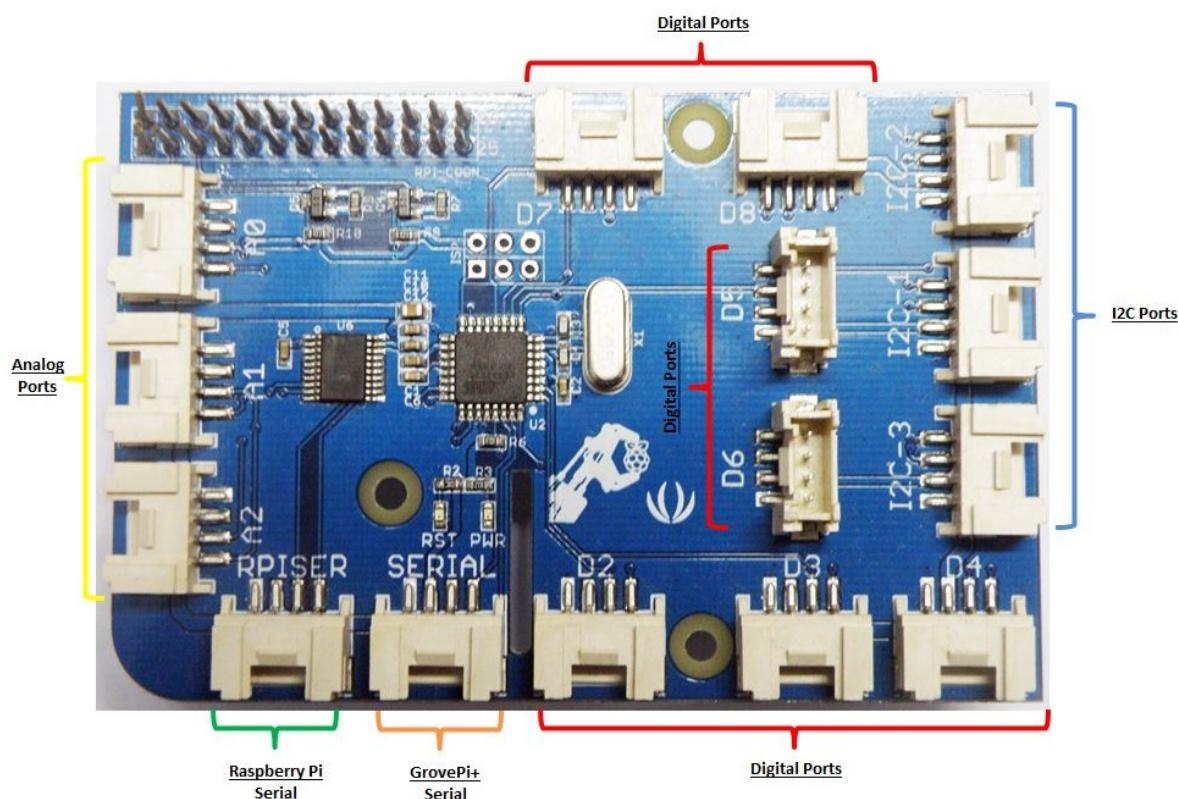
Pour réinitialiser le GrovePi à partir de votre Raspberry Pi, exécutez la commande suivante, à condition que vous ayez installé la bibliothèque GrovePi sur votre image :

avrdude -c gpio -p m328p

II. Description.

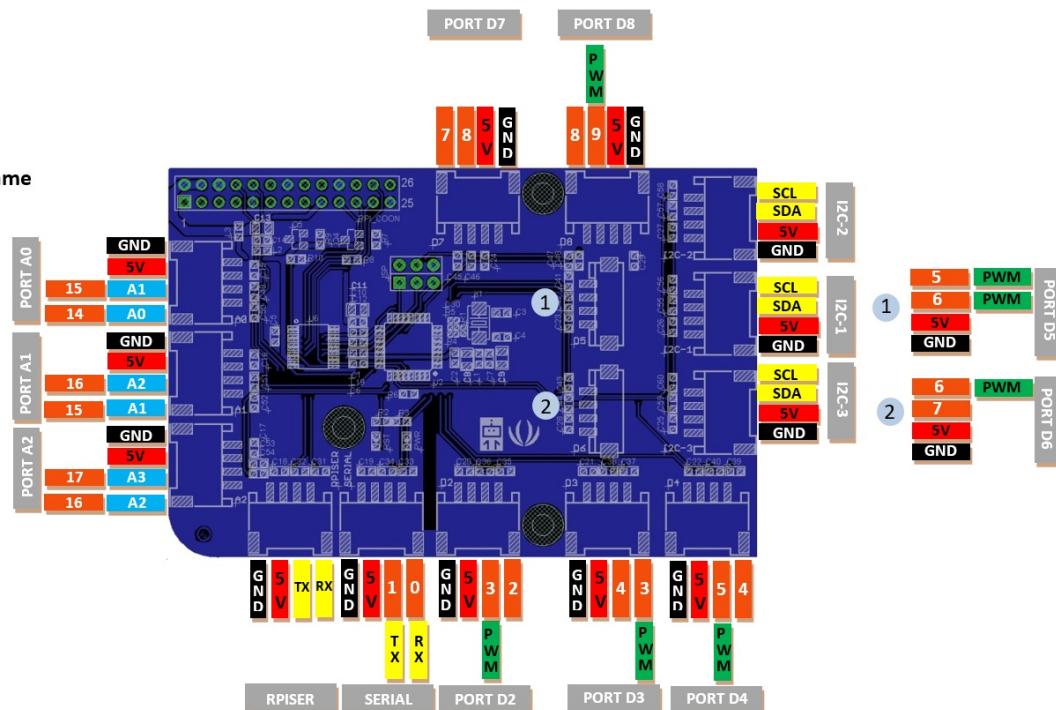
2.1 Disposition des ports.

GROVEPI+ PORT DESCRIPTION



- █ Power
- █ Ground
- █ Digital
- █ Analog
- █ PWM
- █ Control
- █ Port Name

GROVEPI+



Le GrovePi + dispose de plusieurs types de ports:

- Ports analogiques : A0, A1 et A2. Ces ports vous permettent de lire la tension de sortie des capteurs. Dans ce diagramme, ces ports sont colorés en bleu. Lorsque vous utilisez les ports avec notre API, utilisez uniquement des entiers pour désigner le port de votre choix, tel que 0, 1 ou 2.
- Ports numériques : D2, D3, D4, D5, D6, D7 et D8. Avec ces ports, vous pouvez lire et écrire des valeurs numériques de 1 ou 0. Dans le diagramme ci-dessus, ces ports sont colorés en orange. Lorsque vous utilisez les ports avec notre API, utilisez uniquement des nombres entiers pour désigner le port de votre choix tels que 2, 3, ... 8.
- Port PWM : 3 , 5 , 6 , 9 - Avec ces ports, vous pouvez définir une sortie de tension spécifique entre 0 V et 5 V en utilisant le concept de rapport cyclique / PWN / PPM.
- Ports I2C : qui sont colorés en jaune et portent les acronymes SDA & SCL. Le maître de cette connexion est le Raspberry Pi.
- Ports série - situés dans le coin inférieur gauche du diagramme ci-dessus. Le port SERIAL est le port du GrovePi, tandis que RPISER est un contournement du port du Raspberry Pi, qui dispose d'un convertisseur de niveau implanté pour accepter les signaux 5V.

2.2. Système de numérotation.

La plupart des capteurs / actionneurs que nous prenons en charge ont la ligne de signal à l'extérieur du port de la rainure. Par exemple, la DEL Grove a la ligne de signal sur le bord du port de Grove qui correspond au numéro de port 4 sur le nom de port D4, c'est donc pour nous que les noms de ports que nous avons donnés dépendent en réalité du lieu où les lignes de signal résident.

En regardant un câble Grove typique fourni avec un kit GrovePi, la ligne de signal est généralement le fil jaune et le fil blanc reste inutilisé. Comme vous l'avez peut-être deviné, le fil jaune est le fil le plus à l'extérieur du câble Grove.

2.3 API (Application Programming Interface) – Fonctions GPIO.

Vous ne pouvez pas appeler GrovePi à partir de plusieurs processus car cela mettrait GrovePi dans un état défaillant. Les fonctions ne vérifient pas si les paramètres d'entrée sont valides et doivent donc être vérifiés / validés auparavant. L'appel d'une fonction avec des paramètres incorrects peut entraîner un comportement non défini pour GrovePi.

grovepi.digitalRead(pin)

Indique si l'entrée d'un port est élevée ou basse sur le GrovePi.

Paramètre : **pin {Integer}** un numéro pour identifier le port (D2-D8) à partir duquel faire la lecture.

Retourne : 0 ou 1 en fonction de la valeur d'entrée.

grovepi.digitalWrite(pin, value)

Définit la valeur de sortie sur 0 ou 1 sur un port numérique du GrovePi.

Paramètres :

- pin {Integer} un numéro pour identifier le port (D2-D8) sur lequel écrire
- value {Integer}, soit 0 pour 0 volt ou 1 pour la tension de sortie maximale (généralement 5 volts).

Retourne : 1 tout le temps

grovepi.analogRead(pin)

Détecte une tension d'entrée en tant que valeur provenant d'un port donné sur le GrovePi.

Paramètre : pin {Integer} un numéro pour identifier le port (A0-A2) à partir duquel faire la lecture.

Retourne : un nombre de 10 bits {Integer} qui correspond à la tension d'entrée sur le port.

grovepi.analogWrite(pin, value)

Définissez une tension de sortie sur un port compatible PWM en mappant la valeur sur la tension souhaitée sur le GrovePi.

Paramètres :

- pin {Integer} un numéro permettant d'identifier le port (ports 3, 5, 6, 9) sur lequel écrire
 - value {Integer} un nombre à 8 bits qui correspond de 0V à la tension référencée du GrovePi (5V)
- Retourne : 1 tout le temps.

grovepi.pinMode(pin, mode)

Définit un port comme étant un port OUTPUT ou un port INPUT sur le GrovePi.

Paramètres :

- pin {Integer} un numéro pour identifier le port (D2-D8) sur lequel effectuer le changement.
- mode {String} "OUTPUT" pour écrire des valeurs ou "INPUT" pour lire.

Retours : 1 tout le temps.

2.4 Précaution.

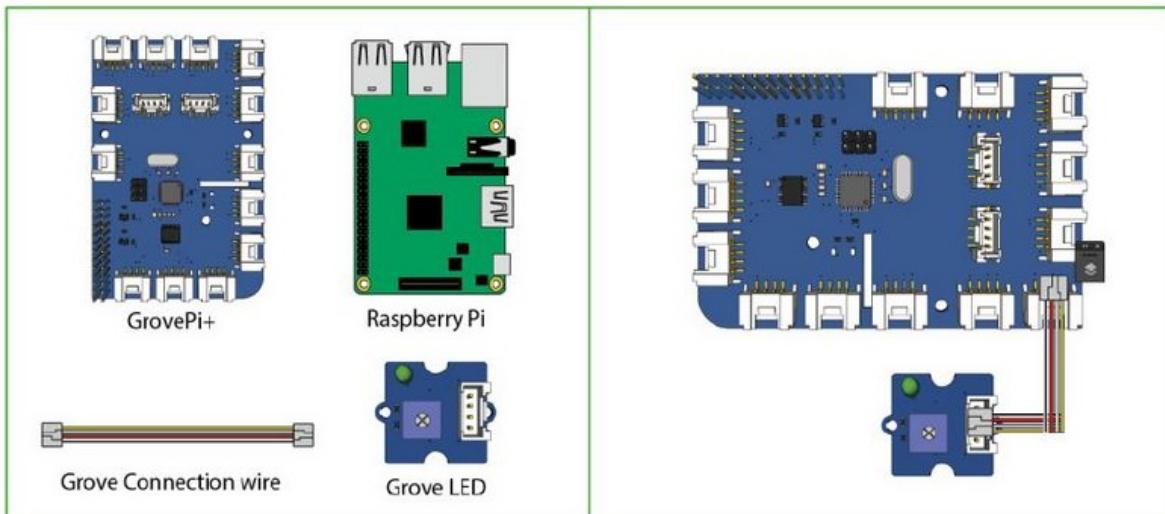
Pour le bon fonctionnement des programmes, il faut toujours se placer dans le répertoire courant suivant :

cd /home/pi/Dexter/GrovePi/Software/Python

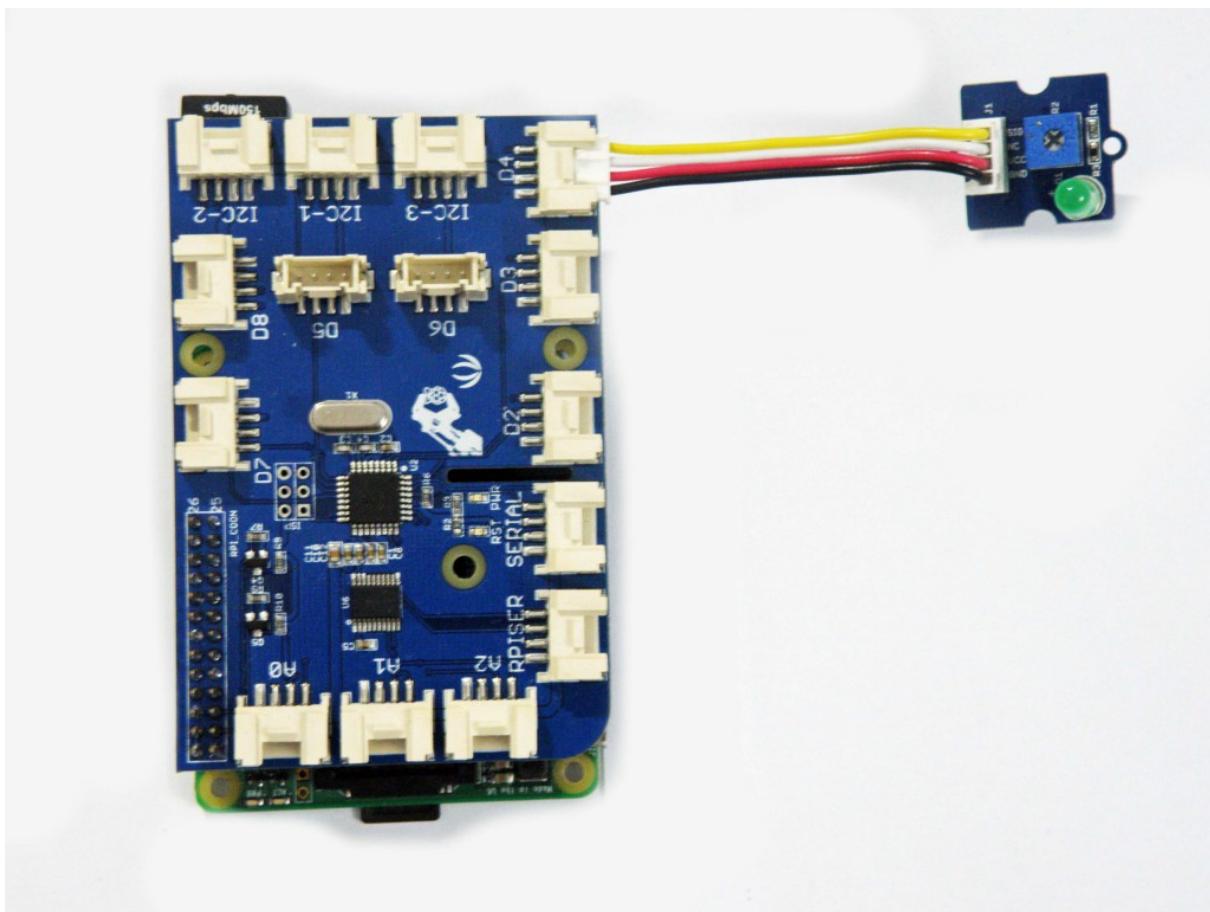
Pour interrompre un programme, il faut appuyer sur **Ctrl + C**.

III. Projets.

3.1 Allumer une Led.



Branchements :



Connecter la led sur le port D4 et allumer le raspberry

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Stéphane KELLER - Lycée agricole Louis Pasteur

Exécuter le programme python **grove_led_blink.py** avec la commande :
python grove_led_blink.py

Code :

```
#!/usr/bin/env python

import time
from grovepi import *

# Connect the Grove LED to digital port D4
cled = 4

pinMode(led,"OUTPUT")
time.sleep(1)

print ("This example will blink a Grove LED connected to the GrovePi+ on the port D{}.\nIf you're having trouble seeing the $")
print (" ")
print ("Connect the LED to the D{} port !".format(led))

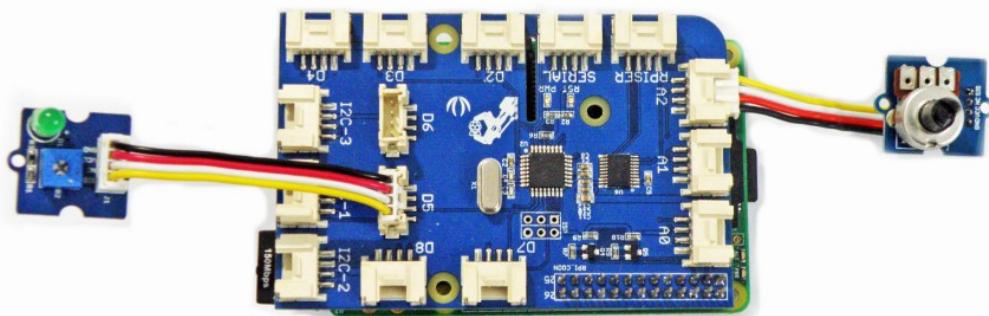
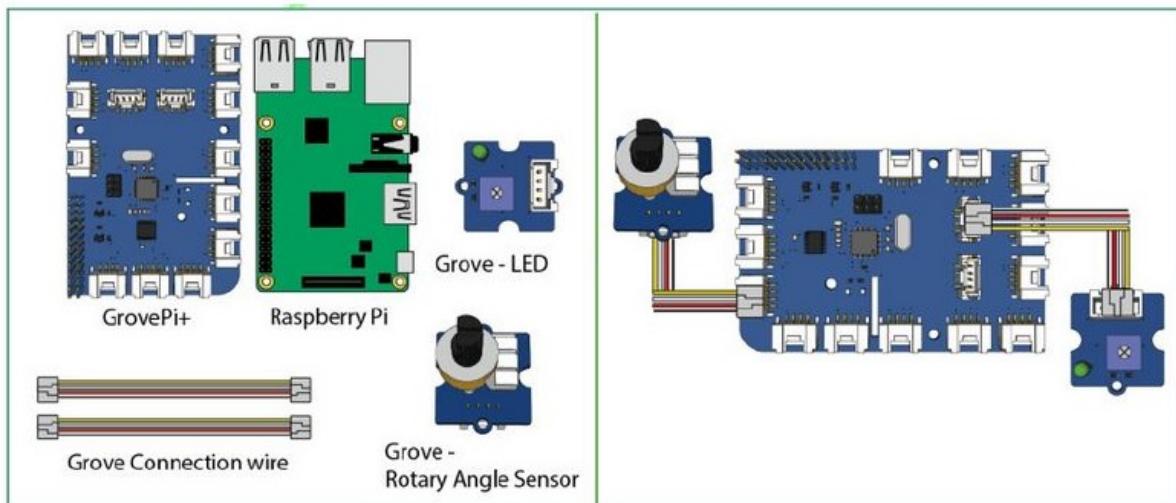
while True:
    try:
        #Blink the LED
        digitalWrite(led,1)          # Send HIGH to switch on LED
        print ("LED ON!")
        time.sleep(1)

        digitalWrite(led,0)          # Send LOW to switch off LED
        print ("LED OFF!")
        time.sleep(1)

    except KeyboardInterrupt: # Turn LED off before stopping
        digitalWrite(led,0)
        break
    except IOError:           # Print "Error" if communication error encountered
        print ("Error")
```

3.2 Led à intensité variable.

Branchements :



Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python
```

Exécuter le programme python **grove_led_fade.py** avec la commande :
python grove_led_fade.py

Code :

```

#!/usr/bin/env python

import time
import grovepi

# Connect the Grove LED to digital port D5
# SIG,NC,VCC,GND
led = 5

# Digital ports that support Pulse Width Modulation (PWM)
# D3, D5, D6

# Digital ports that do not support PWM
# D2, D4, D7, D8

grovepi.pinMode(led,"OUTPUT")
time.sleep(1)
i = 0

while True:
    try:
        # Reset
        if i > 255:
            i = 0

        # Current brightness
        print (i)

        # Give PWM output to LED
        grovepi.analogWrite(led,i)

        # Increment brightness for next iteration
        i = i + 20
        time.sleep(.5)

    except KeyboardInterrupt:
        grovepi.analogWrite(led,0)
        break
    except IOError:
        print ("Error")

```

Modifier le programme précédent pour tenir compte du potentiomètre (il faut rajouter deux lignes et remplacer une partie du programme précédent) :

```

# Adjust LED brightness by rotating Potentiometer

import time
import grovepi

# Connect the LED to digital port D5
led = 5

```

```
# Connect the Rotary Angle Sensor to analog port A2
potentiometer = 2

grovepi.pinMode(led,"OUTPUT")
time.sleep(1)
i = 0

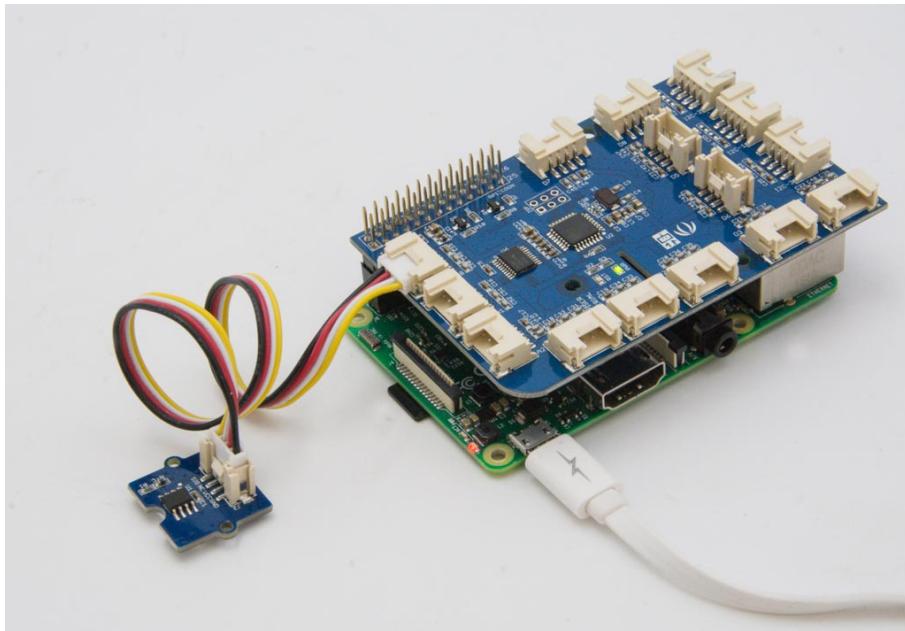
while True:
    try:
        # Read resistance from Potentiometer
        i = grovepi.analogRead(potentiometer)
        print(i)

        # Send PWM signal to LED
        grovepi.analogWrite(led,i//4)

    except IOError:
        print("Error")
```

3.3 Capteur de température.

Le capteur de température Grove utilise un thermistor pour détecter la température ambiante. La résistance d'une thermistance augmente lorsque la température ambiante diminue. C'est cette caractéristique que nous utilisons pour calculer la température ambiante. La plage détectable de ce capteur est de -40 à 125 °C et la précision de ± 1,5 °C.



Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_temperature_sensor.py** avec la commande :
python grove_temperature_sensor.py

grovepi.temp(pin, model='1.0')

Lire la température à partir du capteur de température Grove sur le GrovePi.

Paramètres :

- pin {Integer} un numéro pour identifier le port (A0-A2) à partir duquel faire la lecture
- model {String} "1.0", "1.1"en "1.2"fonction du modèle utilisé

Retours : {Float}nombre représentant la température en °C

Code :

```
import time
import grovepi

# Connect the Grove Temperature Sensor to analog port A0
# SIG,NC,VCC,GND
sensor = 0

while True:
    try:
```

```
temp = grovepi.temp(sensor,'1.1')
print("temp =", temp)
time.sleep(.5)

except KeyboardInterrupt:
    break
except IOError:
    print ("Error")
```

3.4 Capteur de température et d'humidité Pro.



Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python
```

Exécuter le programme python **grove_dht_pro.py** avec la commande :
python grove_dht_pro.py

Une modification dans le fichier est nécessaire selon le type de capteur et donc sa couleur (bleu ou blanc).

Ses performances sont plus complètes et précises que la version de base. La plage de détection de ce capteur est de 5% HR - 99% HR et de -40 ° C à 80 ° C. Et sa précision atteint jusqu'à 2% HR et 0,5 ° C.

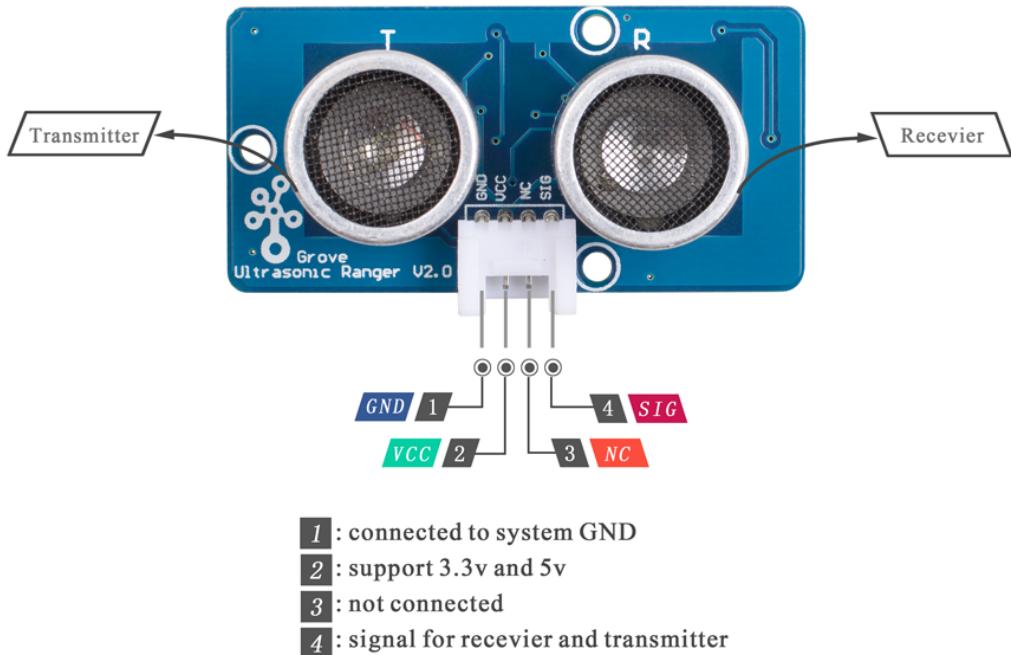
Code :

```
import grovepi
import math
# Connect the Grove Temperature & Humidity Sensor Pro to digital port D4
# This example uses the blue colored sensor.
# SIG,NC,VCC,GND
sensor = 4 # The Sensor goes on digital port 4.

# temp_humidity_sensor_type
# Grove Base Kit comes with the blue sensor.
blue = 0 # The Blue colored sensor.
white = 1 # The White colored sensor.
```

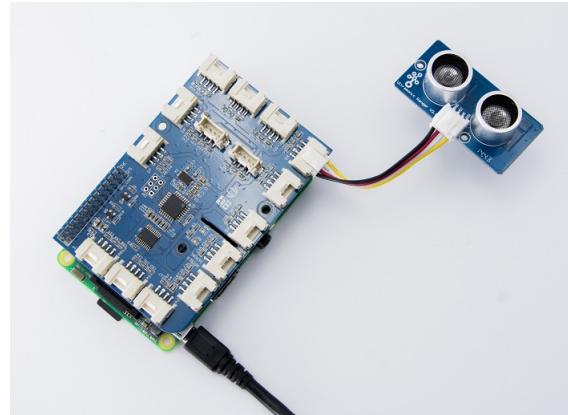
```
while True:  
    try:  
        # This example uses the blue colored sensor.  
        # The first parameter is the port, the second parameter is the type of $  
        [temp,humidity] = grovepi.dht(sensor,white)  
        if math.isnan(temp) == False and math.isnan(humidity) == False:  
            print("temp = %.02f C humidity =%.02f%%"%(temp, humidity))  
  
    except IOError:  
        print ("Error")
```

3.5 Capteur de distance par ultrasons.



Le capteur de distance à ultrasons Grove est un transducteur à ultrasons qui utilise des ondes ultrasonores pour mesurer la distance. Il peut mesurer de 3 cm à 350 cm avec une précision allant jusqu'à 2 mm. C'est un module ultrasonore parfait pour la mesure de distance, les capteurs de proximité et le détecteur à ultrasons.

Ce module possède un émetteur à ultrasons et un récepteur à ultrasons, ce qui vous permet de le considérer comme un émetteur-récepteur à ultrasons. Familiar avec le sonar, lorsque l'onde ultrasonore de 40 KHz générée par l'émetteur rencontre l'objet, l'onde sonore est renvoyée et le récepteur peut recevoir l'onde ultrasonore réfléchie. Il suffit de calculer le temps écoulé entre l'émission et la réception, puis de multiplier la vitesse du son dans l'air (340 m / s) pour calculer la distance entre le capteur et l'objet.



Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_ultrasonic.py** avec la commande :
python grove_ultrasonic.py

Code :

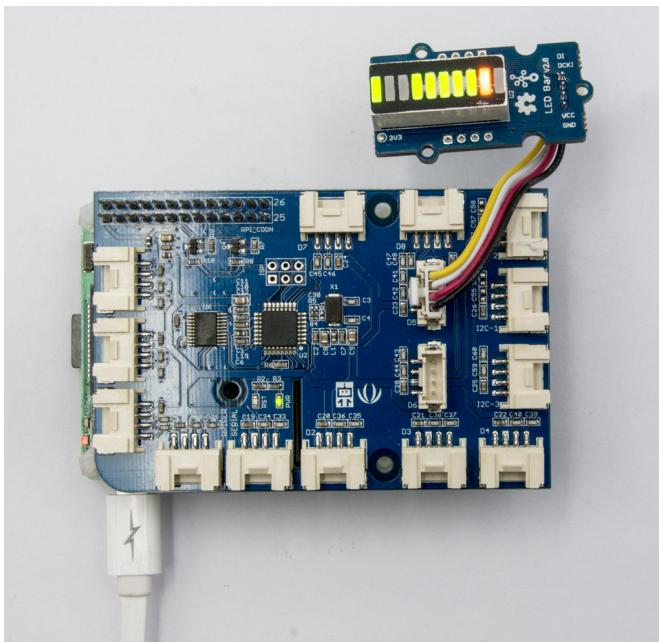
```
import grovepi  
  
# Connect the Grove Ultrasonic Ranger to digital port D4
```

```
# SIG,NC,VCC,GND
ultrasonic_ranger = 4

while True:
    try:
        # Read distance value from Ultrasonic
        print(grovepi.ultrasonicRead(ultrasonic_ranger))

    except TypeError:
        print ("Error")
    except IOError:
        print ("Error")
```

3.6 LedBar.



Grove - LED Bar est composé d'une barre de jauge à LED de 10 segments et d'une puce de contrôle de LED MY9221. Il peut être utilisé comme indicateur de la durée de vie restante de la batterie, de la tension, du niveau d'eau, du volume de la musique ou d'autres valeurs nécessitant un affichage en dégradé. Le graphique à barres comporte 10 barres DEL: une barre rouge, une barre jaune, une barre verte et sept barres vertes. Le code de démonstration est disponible pour vous permettre de démarrer rapidement. Il allume les voyants de manière séquentielle du rouge au vert, de sorte que tout le graphique à barres est allumé à la fin.

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_ledbar.py** avec la commande :

python grove_ledbar.py

Code :

```
import time
import grovepi
import random

# Connect the Grove LED Bar to digital port D5
# DI,DCKI,VCC,GND
leddbar = 5

grovepi.pinMode(leddbar,"OUTPUT")
time.sleep(1)
i = 0

# LED Bar methods
# grovepi.ledBar_init(pin,orientation)
# grovepi.ledBar_orientation(pin,orientation)
```

```

# grovepi.ledBar_setLevel(pin,level)
# grovepi.ledBar_setLed(pin,led,state)
# grovepi.ledBar_toggleLed(pin,led)
# grovepi.ledBar_setBits(pin,state)
# grovepi.ledBar_getBits(pin)

while True:
    try:
        print ("Test 1) Initialise - red to green")
        # ledbar_init(pin,orientation)
        # orientation: (0 = red to green, 1 = green to red)
        grovepi.ledBar_init(ledbar, 0)
        time.sleep(.5)

        print ("Test 2) Set level")
        # ledbar_setLevel(pin,level)
        # level: (0-10)
        for i in range(0,11):
            grovepi.ledBar_setLevel(ledbar, i)
            time.sleep(.2)
        time.sleep(.3)

        grovepi.ledBar_setLevel(ledbar, 8)
        time.sleep(.5)

        grovepi.ledBar_setLevel(ledbar, 2)
        time.sleep(.5)

        print ("Test 3) Switch on/off a single LED")
        # ledbar_setLed(pin,led,state)
        # led: which led (1-10)
        # state: off or on (0,1)
        grovepi.ledBar_setLed(ledbar, 10, 1)
        time.sleep(.5)

        grovepi.ledBar_setLed(ledbar, 9, 1)
        time.sleep(.5)

        grovepi.ledBar_setLed(ledbar, 8, 1)
        time.sleep(.5)

        grovepi.ledBar_setLed(ledbar, 1, 0)
        time.sleep(.5)

        grovepi.ledBar_setLed(ledbar, 2, 0)
        time.sleep(.5)

        grovepi.ledBar_setLed(ledbar, 3, 0)
        time.sleep(.5)

```

```

print ("Test 4) Toggle a single LED")
# flip a single led - if it is currently on, it will become off and vice versa
# ledbar_toggleLed(ledbar, led)
grovepi.ledBar_toggleLed(ledbar, 1)
time.sleep(.5)

grovepi.ledBar_toggleLed(ledbar, 2)
time.sleep(.5)

grovepi.ledBar_toggleLed(ledbar, 9)
time.sleep(.5)

grovepi.ledBar_toggleLed(ledbar, 10)
time.sleep(.5)

print ("Test 5) Set state - control all leds with 10 bits")
# ledbar_setBits(ledbar, state)
# state: (0-1023) or (0x00-0x3FF) or (0b0000000000-0b1111111111) or (inverted)
for i in range(0,32):
    grovepi.ledBar_setBits(ledbar, i)
    time.sleep(.2)
time.sleep(.3)

print ("Test 6) Get current state")
# state = ledbar_getBits(ledbar)
# state: (0-1023) a bit for each of the 10 LEDs
state = grovepi.ledBar_getBits(ledbar)
print ("with first 5 leds lit, the state should be 31 or 0x1F")
print (state)

# bitwise shift five bits to the left
state = state << 5
# the state should now be 992 or 0x3E0
# when saved the last 5 LEDs will be lit instead of the first 5 LEDs
time.sleep(.5)

print ("Test 7) Set state - save the state we just modified")
# ledbar_setBits(ledbar, state)
# state: (0-1023) a bit for each of the 10 LEDs
grovepi.ledBar_setBits(ledbar, state)
time.sleep(.5)

print ("Test 8) Swap orientation - green to red - current state is press")
# ledbar_orientation(pin,orientation)
# orientation: (0 = red to green, 1 = green to red)

```

```

# when you reverse the led bar orientation, all methods know how to han$ 
# green to red
grovepi.ledBar_orientation(ledbar, 1)
time.sleep(.5)

# red to green
grovepi.ledBar_orientation(ledbar, 0)
time.sleep(.5)

# green to red
grovepi.ledBar_orientation(ledbar, 1)
time.sleep(.5)

print ("Test 9) Set level, again")
# ledbar_setLevel(pin,level)
# level: (0-10)
# note the red LED is now at index 10 instead of 1
for i in range(0,11):
    grovepi.ledBar_setLevel(ledbar, i)
    time.sleep(.2)
time.sleep(.3)

print ("Test 10) Set a single LED, again")
# ledbar_setLed(pin,led,state)
# led: which led (1-10)
# state: off or on (0,1)
grovepi.ledBar_setLed(ledbar, 1, 0)
time.sleep(.5)

grovepi.ledBar_setLed(ledbar, 3, 0)
time.sleep(.5)

grovepi.ledBar_setLed(ledbar, 5, 0)
time.sleep(.5)

print ("Test 11) Toggle a single LED, again")
# ledbar_toggleLed(ledbar, led)
grovepi.ledBar_toggleLed(ledbar, 2)
time.sleep(.5)

grovepi.ledBar_toggleLed(ledbar, 4)
time.sleep(.5)

print ("Test 12) Get state")
# state = ledbar_getBits(ledbar)
# state: (0-1023) a bit for each of the 10 LEDs
state = grovepi.ledBar_getBits(ledbar)

```

```

# the last 5 LEDs are lit, so the state should be 992 or 0x3E0

# bitwise shift five bits to the right
state = state >> 5
# the state should now be 31 or 0x1F

print ("Test 13) Set state, again")
# ledbar_setBits(ledbar, state)
# state: (0-1023) a bit for each of the 10 LEDs
grovepi.ledBar_setBits(ledbar, state)
time.sleep(.5)

print ("Test 14) Step")
# step through all 10 LEDs
for i in range(0,11):
    grovepi.ledBar_setLevel(ledbar, i)
    time.sleep(.2)
time.sleep(.3)

print ("Test 15) Bounce")
# switch on the first two LEDs
grovepi.ledBar_setLevel(ledbar, 2)

# get the current state (which is 0x3)
state = grovepi.ledBar_getBits(ledbar)

# bounce to the right
for i in range(0,9):
    # bit shift left and update
    state <<= 1;
    grovepi.ledBar_setBits(ledbar, state)
    time.sleep(.2)

# bounce to the left
for i in range(0,9):
    # bit shift right and update
    state >>= 1;
    grovepi.ledBar_setBits(ledbar, state)
    time.sleep(.2)
time.sleep(.3)

print ("Test 16) Random")
for i in range(0,21):
    state = random.randint(0,1023)
    grovepi.ledBar_setBits(ledbar, state)
    time.sleep(.2)

```

```

time.sleep(.3)

print ("Test 17) Invert")
# set every 2nd LED on - 341 or 0x155
state = 341
for i in range(0,5):
    grovepi.ledBar_setBits(ledbar, state)
    time.sleep(.2)

    # bitwise XOR all 10 LEDs on with the current state
    state = 0x3FF ^ state

    grovepi.ledBar_setBits(ledbar, state)
    time.sleep(.2)
time.sleep(.3)

print ("Test 18) Walk through all possible combinations")
for i in range(0,1024):
    grovepi.ledBar_setBits(ledbar, i)
    time.sleep(.1)
time.sleep(.4)

except KeyboardInterrupt:
    grovepi.ledBar_setBits(ledbar, 0)
    break
except IOError:
    print ("Error")

```

3.7 Capteur de distance par ultrasons et ledBar.

Concevoir le programme utilisant le capteur de distance par ultrasons et le ledbar afin de concevoir un radar de recul.

Si l'obstacle est loin, alors la ledbar affiche du vert

Au fur et à mesure que l'obstacle se rapproche alors le vumètre augmente jusqu'au rouge qui signifie que l'obstacle est très proche.

Programme **ultrasonic_ledbar.py**

```
import grovepi
import time

# Connect the Grove Ultrasonic Ranger to digital port D4
# SIG,NC,VCC,GND
ultrasonic_ranger = 4
dist_min, dist_max = 3, 495

# Connect the Grove LED Bar to digital port D5
# DI,DCKI,VCC,GND
ledbar = 5

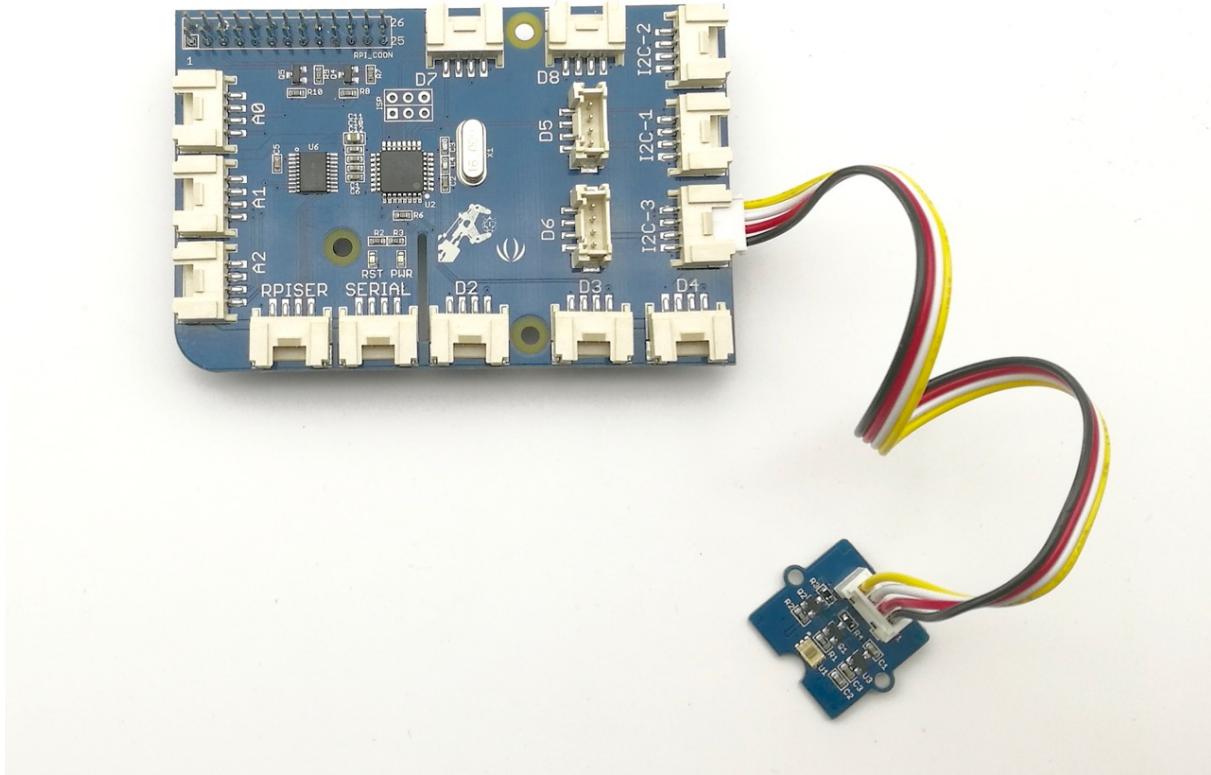
# ledbar_init(pin,orientation)
# orientation: (0 = red to green, 1 = green to red)
grovepi.ledBar_init(ledbar, 1)
time.sleep(.5)

while True:
    try:
        # Read distance value from Ultrasonic
        distance = grovepi.ultrasonicRead(ultrasonic_ranger)
        time.sleep(.5)

        if distance > 40 :
            i = 0
        elif distance > 35 :
            i = 1
        elif distance > 30 :
            i = 2
        elif distance > 25 :
            i = 3
        elif distance > 20 :
            i = 4
        elif distance > 15 :
            i = 5
        elif distance > 10 :
            i = 6
        elif distance > 8 :
            i = 7
        elif distance > 6 :
            i = 8
```

```
elif distance > 4 :  
    i = 9  
else :  
    i = 10  
  
print("Distance =", distance,"Intensite LedBar =",i)  
# ledbar_setLevel(pin,level)  
# level: (0-10)  
grovepi.ledBar_setLevel(ledbar, i)  
time.sleep(.2)  
  
except TypeError:  
    print ("Error")  
except IOError:  
    print ("Error")
```

3.8 Convertisseur de lumière.



Ce module est basé sur le convertisseur lumière-numérique I2C TSL2561 pour transformer l'intensité lumineuse en un signal numérique. Différent du capteur de lumière analogique traditionnel, comme Grove - Capteur de lumière, ce module numérique présente une gamme de spectre de lumière sélectionnable en raison de ses doubles diodes sensibles à la lumière: infrarouge et spectre complet.

Nous pouvons basculer entre trois modes de détection pour prendre vos lectures. Ce sont le mode infrarouge, le spectre complet et le mode visible humain. Lorsqu'il fonctionne sous le mode visible humain, ce capteur vous donnera des lectures proches de vos sensations oculaires.

Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python/grove_i2c_digital_light_sensor
```

Exécuter le programme python **grove_i2c_digital_light_sensor.py** avec la commande :
python grove_i2c_digital_light_sensor.py

Code :

```
from time import sleep
import smbus
from Adafruit_I2C import Adafruit_I2C
```

```

import RPi.GPIO as GPIO
from smbus import SMBus

TSL2561_Control = 0x80
TSL2561_Timing = 0x81
TSL2561_Interrupt = 0x86
TSL2561_Channel0L = 0x8C
TSL2561_Channel0H = 0x8D
TSL2561_Channel1L = 0x8E
TSL2561_Channel1H = 0x8F

TSL2561_Address = 0x29 #device address

LUX_SCALE = 14 # scale by 2^14
RATIO_SCALE = 9 # scale ratio by 2^9
CH_SCALE = 10 # scale channel values by 2^10
CHSCALE_TINT0 = 0x7517 # 322/11 * 2^CH_SCALE
CHSCALE_TINT1 = 0x0fe7 # 322/81 * 2^CH_SCALE

K1T = 0x0040 # 0.125 * 2^RATIO_SCALE
B1T = 0x01f2 # 0.0304 * 2^LUX_SCALE
M1T = 0x01be # 0.0272 * 2^LUX_SCALE
K2T = 0x0080 # 0.250 * 2^RATIO_SCA
B2T = 0x0214 # 0.0325 * 2^LUX_SCALE
M2T = 0x02d1 # 0.0440 * 2^LUX_SCALE
K3T = 0x00c0 # 0.375 * 2^RATIO_SCALE
B3T = 0x023f # 0.0351 * 2^LUX_SCALE
M3T = 0x037b # 0.0544 * 2^LUX_SCALE
K4T = 0x0100 # 0.50 * 2^RATIO_SCALE
B4T = 0x0270 # 0.0381 * 2^LUX_SCALE
M4T = 0x03fe # 0.0624 * 2^LUX_SCALE
K5T = 0x0138 # 0.61 * 2^RATIO_SCALE
B5T = 0x016f # 0.0224 * 2^LUX_SCALE
M5T = 0x01fc # 0.0310 * 2^LUX_SCALE
K6T = 0x019a # 0.80 * 2^RATIO_SCALE
B6T = 0x00d2 # 0.0128 * 2^LUX_SCALE
M6T = 0x00fb # 0.0153 * 2^LUX_SCALE
K7T = 0x029a # 1.3 * 2^RATIO_SCALE
B7T = 0x0018 # 0.00146 * 2^LUX_SCALE
M7T = 0x0012 # 0.00112 * 2^LUX_SCALE
K8T = 0x029a # 1.3 * 2^RATIO_SCALE
B8T = 0x0000 # 0.000 * 2^LUX_SCALE
M8T = 0x0000 # 0.000 * 2^LUX_SCALE

B1C = 0x0204 # 0.0315 * 2^LUX_SCALE
M1C = 0x01ad # 0.0262 * 2^LUX_SCALE
K2C = 0x0085 # 0.260 * 2^RATIO_SCALE
B2C = 0x0228 # 0.0337 * 2^LUX_SCALE
M2C = 0x02c1 # 0.0430 * 2^LUX_SCALE
K3C = 0x00c8 # 0.390 * 2^RATIO_SCALE
B3C = 0x0253 # 0.0363 * 2^LUX_SCALE

```

```

M3C = 0x0363 # 0.0529 * 2^LUX_SCALE
K4C = 0x010a # 0.520 * 2^RATIO_SCALE
B4C = 0x0282 # 0.0392 * 2^LUX_SCALE
M4C = 0x03df # 0.0605 * 2^LUX_SCALE
K5C = 0x014d # 0.65 * 2^RATIO_SCALE
B5C = 0x0177 # 0.0229 * 2^LUX_SCALE
M5C = 0x01dd # 0.0291 * 2^LUX_SCALE
K6C = 0x019a # 0.80 * 2^RATIO_SCALE
B6C = 0x0101 # 0.0157 * 2^LUX_SCALE
M6C = 0x0127 # 0.0180 * 2^LUX_SCALE
K7C = 0x029a # 1.3 * 2^RATIO_SCALE
B7C = 0x0037 # 0.00338 * 2^LUX_SCALE
M7C = 0x002b # 0.00260 * 2^LUX_SCALE
K8C = 0x029a # 1.3 * 2^RATIO_SCALE
B8C = 0x0000 # 0.000 * 2^LUX_SCALE
M8C = 0x0000 # 0.000 * 2^LUX_SCALE

# bus parameters
rev = GPIO.RPI_REVISION
if rev == 2 or rev == 3:
    bus = smbus.SMBus(1)
else:
    bus = smbus.SMBus(0)
i2c = Adafruit_I2C(TSL2561_Address)

debug = False
cooldown_time = 0.005 # measured in seconds
packageType = 0 # 0=T package, 1=CS package
gain = 0      # current gain: 0=1x, 1=16x [dynamically selected]
gain_m = 1    # current gain, as multiplier
timing = 2     # current integration time: 0=13.7ms, 1=101ms, 2=402ms [dynamically
selected]
timing_ms = 0  # current integration time, in ms
channel0 = 0   # raw current value of visible+ir sensor
channel1 = 0   # raw current value of ir sensor
schannel0 = 0  # normalized current value of visible+ir sensor
schannel1 = 0  # normalized current value of ir sensor

def readRegister(address):
    try:
        byteval = i2c.readU8(address)

        sleep(cooldown_time)
        if (debug):
            print("TSL2561.readRegister: returned 0x%02X from reg 0x%02X" %
(byteval, address))
        return byteval
    except IOError:
        print("TSL2561.readRegister: error reading byte from reg 0x%02X" % address)
        return -1

```

```

def writeRegister(address, val):
    try:
        i2c.write8(address, val)

        sleep(cooldown_time)
        if (debug):
            print("TSL2561.writeRegister: wrote 0x%02X to reg 0x%02X" % (val,
address))
    except IOError:

        sleep(cooldown_time)
        print("TSL2561.writeRegister: error writing byte to reg 0x%02X" % address)
        return -1

def powerUp():
    writeRegister(TSL2561_Control, 0x03)

def powerDown():
    writeRegister(TSL2561_Control, 0x00)

def setTintAndGain():
    global gain_m, timing_ms

    if gain == 0:
        gain_m = 1
    else:
        gain_m = 16

    if timing == 0:
        timing_ms = 13.7
    elif timing == 1:
        timing_ms = 101
    else:
        timing_ms = 402
    writeRegister(TSL2561_Timing, timing | gain << 4)

def readLux():
    sleep(float(timing_ms + 1) / 1000)

    ch0_low = readRegister(TSL2561_Channel0L)
    ch0_high = readRegister(TSL2561_Channel0H)
    ch1_low = readRegister(TSL2561_Channel1L)
    ch1_high = readRegister(TSL2561_Channel1H)

    global channel0, channel1
    channel0 = (ch0_high<<8) | ch0_low
    channel1 = (ch1_high<<8) | ch1_low

    sleep(cooldown_time)
    if debug:

```

```

        print("TSL2561.readVisibleLux: channel 0 = %i, channel 1 = %i [gain=%ix,
timing=%ims]" % (channel0, $

def readVisibleLux():
    global timing, gain

    powerUp()
    readLux()

    if channel0 < 500 and timing == 0:
        timing = 1
        sleep(cooldown_time)
        if debug:
            print("TSL2561.readVisibleLux: too dark. Increasing integration time from
13.7ms to 101ms")
            setTintAndGain()
            readLux()

    if channel0 < 500 and timing == 1:
        timing = 2
        sleep(cooldown_time)
        if debug:
            print("TSL2561.readVisibleLux: too dark. Increasing integration time from
101ms to 402ms")
            setTintAndGain()
            readLux()

    if channel0 < 500 and timing == 2 and gain == 0:
        gain = 1
        sleep(cooldown_time)
        if debug:
            print("TSL2561.readVisibleLux: too dark. Setting high gain")
            setTintAndGain()
            readLux()

    if (channel0 > 20000 or channel1 > 20000) and timing == 2 and gain == 1:
        gain = 0
        sleep(cooldown_time)
        if debug:
            print("TSL2561.readVisibleLux: enough light. Setting low gain")
            setTintAndGain()
            readLux()

    if (channel0 > 20000 or channel1 > 20000) and timing == 2:
        timing = 1
        sleep(cooldown_time)
        if debug:
            print("TSL2561.readVisibleLux: enough light. Reducing integration time from
402ms to 101ms")
            setTintAndGain()
            readLux()

```

```

if (channel0 > 10000 or channel1 > 10000) and timing == 1:
    timing = 0
    sleep(cooldown_time)
    if debug:
        print("TSL2561.readVisibleLux: enough light. Reducing integration time from
101ms to 13.7ms")
    setTintAndGain()
    readLux()

powerDown()

if (timing == 0 and (channel0 > 5000 or channel1 > 5000)) or (timing == 1 and
(channel0 > 37000 or channel1 >
# overflow
return -1

return calculateLux(channel0, channel1)

def calculateLux(ch0, ch1):
    chScale = 0
    if timing == 0: # 13.7 msec
        chScale = CHSCALE_TINT0
    elif timing == 1: # 101 msec
        chScale = CHSCALE_TINT1;
    else: # assume no scaling
        chScale = (1 << CH_SCALE)

    if gain == 0:
        chScale = chScale << 4 # scale 1X to 16X

    # scale the channel values
    global schannel0, schannel1
    schannel0 = (ch0 * chScale) >> CH_SCALE
    schannel1 = (ch1 * chScale) >> CH_SCALE

    ratio = 0
    if schannel0 != 0:
        ratio = (schannel1 << (RATIO_SCALE+1)) / schannel0
        ratio = (ratio + 1) >> 1

    if packageType == 0: # T package
        if ((ratio >= 0) and (ratio <= K1T)):
            b=B1T; m=M1T;
        elif (ratio <= K2T):
            b=B2T; m=M2T;
        elif (ratio <= K3T):
            b=B3T; m=M3T;
        elif (ratio <= K4T):
            b=B4T; m=M4T;
        elif (ratio <= K5T):
            b=B5T; m=M5T;

```

```

        elif (ratio <= K6T):
            b=B6T; m=M6T;
        elif (ratio <= K7T):
            b=B7T; m=M7T;
        elif (ratio > K8T):
            b=B8T; m=M8T;
    elif packageType == 1: # CS package
        if ((ratio >= 0) and (ratio <= K1C)):
            b=B1C; m=M1C;
        elif (ratio <= K2C):
            b=B2C; m=M2C;
        elif (ratio <= K3C):
            b=B3C; m=M3C;
        elif (ratio <= K4C):
            b=B4C; m=M4C;
        elif (ratio <= K5C):
            b=B5C; m=M5C;
        elif (ratio <= K6C):
            b=B6C; m=M6C;
        elif (ratio <= K7C):
            b=B7C; m=M7C;

    temp = ((schannel0*b)-(schannel1*m))
    if temp < 0:
        temp = 0;
    temp += (1<<(LUX_SCALE-1))
    # strip off fractional portion
    lux = temp>>LUX_SCALE
    sleep(cooldown_time)
    if debug:
        print("TSL2561.calculateLux: %i" % lux)

    return lux

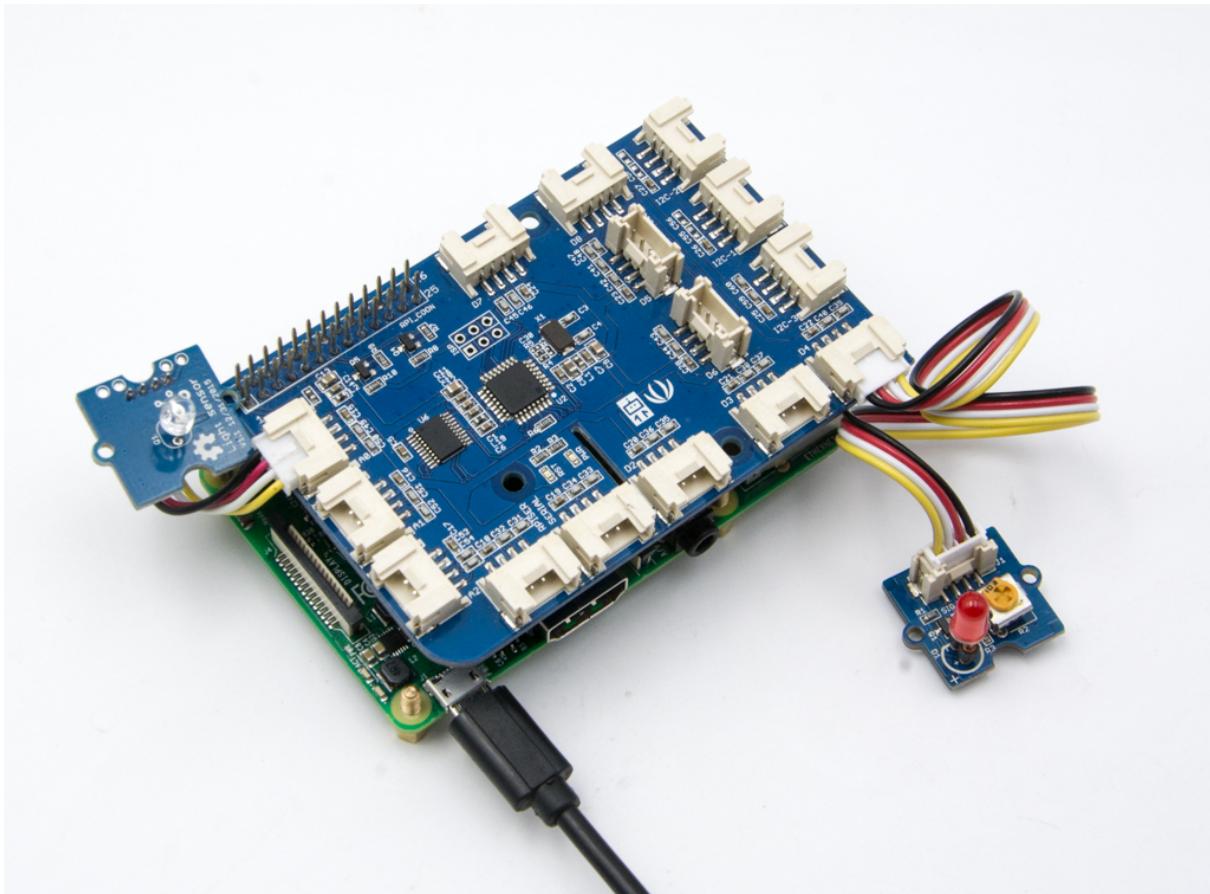
def init():
    powerUp()
    setTintAndGain()
    writeRegister(TSL2561_Interrupt, 0x00)
    powerDown()

def main():
    init()
    while (True):
        print("Lux: %i [Vis+IR=%i, IR=%i @ Gain=%ix, Timing=%.1fms]" %
(readVisibleLux(), channel0, channel1$
        sleep(1)

if __name__ == "__main__":
    main()

```

3.10 Capteur de lumière.



Le capteur Grove - Light intègre une photorésistance (résistance dépendante de la lumière) pour détecter l'intensité de la lumière. La résistance de la photorésistance diminue lorsque l'intensité de la lumière augmente. Une double puce OpAmp LM358 à bord produit une tension correspondant à l'intensité de la lumière (c'est-à-dire en fonction de la valeur de résistance). Le signal de sortie est une valeur analogique, plus la lumière est brillante, plus la valeur est élevée.

Ce module peut être utilisé pour construire un interrupteur commandé par la lumière, c'est-à-dire éteindre les lumières pendant la journée et allumer les lumières pendant la nuit.

Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python
```

Exécuter le programme python **grove_light_sensor.py** avec la commande :
python grove_light_sensor.py

Code :

```
import time
import grovepi

# Connect the Grove Light Sensor to analog port A0
# SIG,NC,VCC,GND
light_sensor = 0
```

```

# Connect the LED to digital port D4
# SIG,NC,VCC,GND
led = 4

# Turn on LED once sensor exceeds threshold resistance
threshold = 10

grovepi.pinMode(light_sensor,"INPUT")
grovepi.pinMode(led,"OUTPUT")

while True:
    try:
        # Get sensor value
        sensor_value = grovepi.analogRead(light_sensor)

        # Calculate resistance of sensor in K
        resistance = (float)(1023 - sensor_value) * 10 / sensor_value

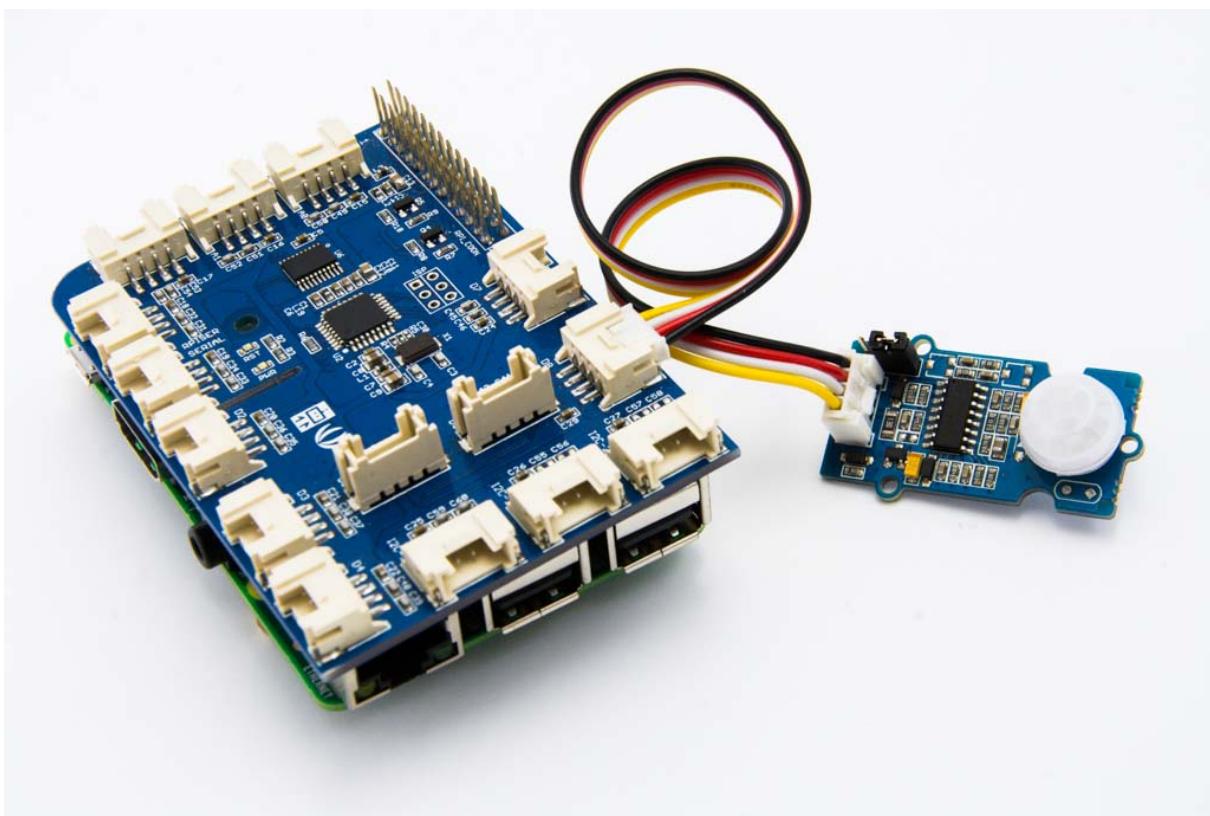
        if resistance > threshold:
            # Send HIGH to switch on LED
            grovepi.digitalWrite(led,1)
        else:
            # Send LOW to switch off LED
            grovepi.digitalWrite(led,0)

        print("sensor_value = %d resistance = %.2f" %(sensor_value, resistance))
        time.sleep(.5)

    except IOError:
        print ("Error")

```

3.11 DéTECTEUR de mouvement.



Ce capteur vous permet de détecter le mouvement, généralement le mouvement humain dans sa plage. Connectez-le simplement à Grove - Base Shield et programmez-le, lorsque quelqu'un se déplace dans sa plage de détection, le capteur sortira HAUT sur sa broche SIG.

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_pir_motion_sensor.py** avec la commande :

python grove_pir_motion_sensor.py

Code :

```
import time
import grovepi

# Connect the Grove PIR Motion Sensor to digital port D8
# NOTE: Some PIR sensors come with the SIG line connected to the yellow wire and some
# with the SIG line connected to the white wire.
# If the example does not work on the first run, try changing the pin number
# For example, for port D8, if pin 8 does not work below, change it to pin 7, since each port
# has 2 digital pins.
# For port 4, this would be pin 3 and 4

pir_sensor = 8
motion=0
grovepi.pinMode(pir_sensor,"INPUT")
```

```

while True:
    try:
        # Sense motion, usually human, within the target range
        motion=grovepi.digitalRead(pir_sensor)
        if motion==0 or motion==1:    # check if reads were 0 or 1 it can be 255 also
because of IO Errors so remove those values
        if motion==1:
            print ('Motion Detected')
        else:
            print ('-')

        # if your hold time is less than this, you might not see as many detections
        time.sleep(.2)

    except IOError:
        print ("Error")

```

Modifier le programme pour allumer une led en cas de détection d'un mouvement.

programme python **grove_pir_motion_sensor_led.py** avec la commande :
python grove_pir_motion_sensor_led.py

```

import time
import grovepi

# Connect the Grove PIR Motion Sensor to digital port D8
# NOTE: Some PIR sensors come with the SIG line connected to the yellow wire and some
with the SIG line connected to the white wire.
# If the example does not work on the first run, try changing the pin number
# For example, for port D8, if pin 8 does not work below, change it to pin 7, since each port
has 2 digital pins.
# For port 4, this would pin 3 and 4

# Connect the Grove LED to digital port D4
led = 4

grovepi.pinMode(led,"OUTPUT")
time.sleep(1)

pir_sensor = 8
motion=0
grovepi.pinMode(pir_sensor,"INPUT")

while True:
    try:
        # Sense motion, usually human, within the target range
        motion=grovepi.digitalRead(pir_sensor)
        if motion==0 or motion==1:    # check if reads were 0 or 1 it can be 255 also
because of IO Errors so remove those values
        if motion==1:

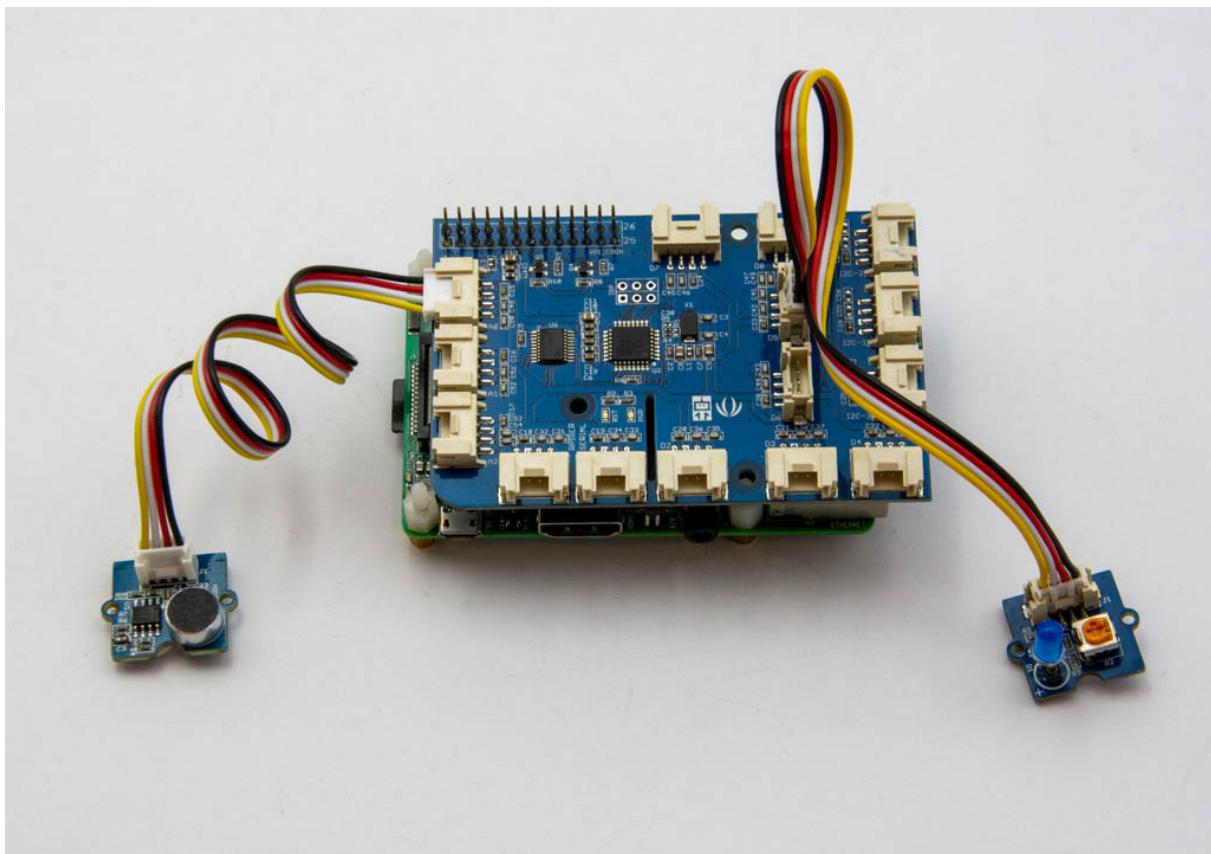
```

```
#Blink the LED
grovepi.digitalWrite(led,1) # Send HIGH to switch on LED
print ("LED ON!")
time.sleep(1)
# print ('Motion Detected')
else:
    grovepi.digitalWrite(led,0) # Send LOW to switch off LED
    print ("LED OFF!")
    time.sleep(1)
    #print ('-')

    # if your hold time is less than this, you might not see as many detections
    time.sleep(.2)

except IOError:
    print ("Error")
```

3.12 Capteur de son.



Grove - Sound Sensor peut détecter l'intensité sonore de l'environnement. Le composant principal du module est un simple microphone, basé sur l'amplificateur LM386 et un microphone à électret. La sortie de ce module est analogique.

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_sound_sensor.py** avec la commande :
python grove_sound_sensor.py

Code :

```
import time
import grovepi

# Connect the Grove Sound Sensor to analog port A0
# SIG,NC,VCC,GND
sound_sensor = 0

# Connect the Grove LED to digital port D5
# SIG,NC,VCC,GND
led = 5
```

```

grovepi.pinMode(sound_sensor,"INPUT")
grovepi.pinMode(led,"OUTPUT")

# The threshold to turn the led on 400.00 * 5 / 1024 = 1.95v
threshold_value = 400

while True:
    try:
        # Read the sound level
        sensor_value = grovepi.analogRead(sound_sensor)

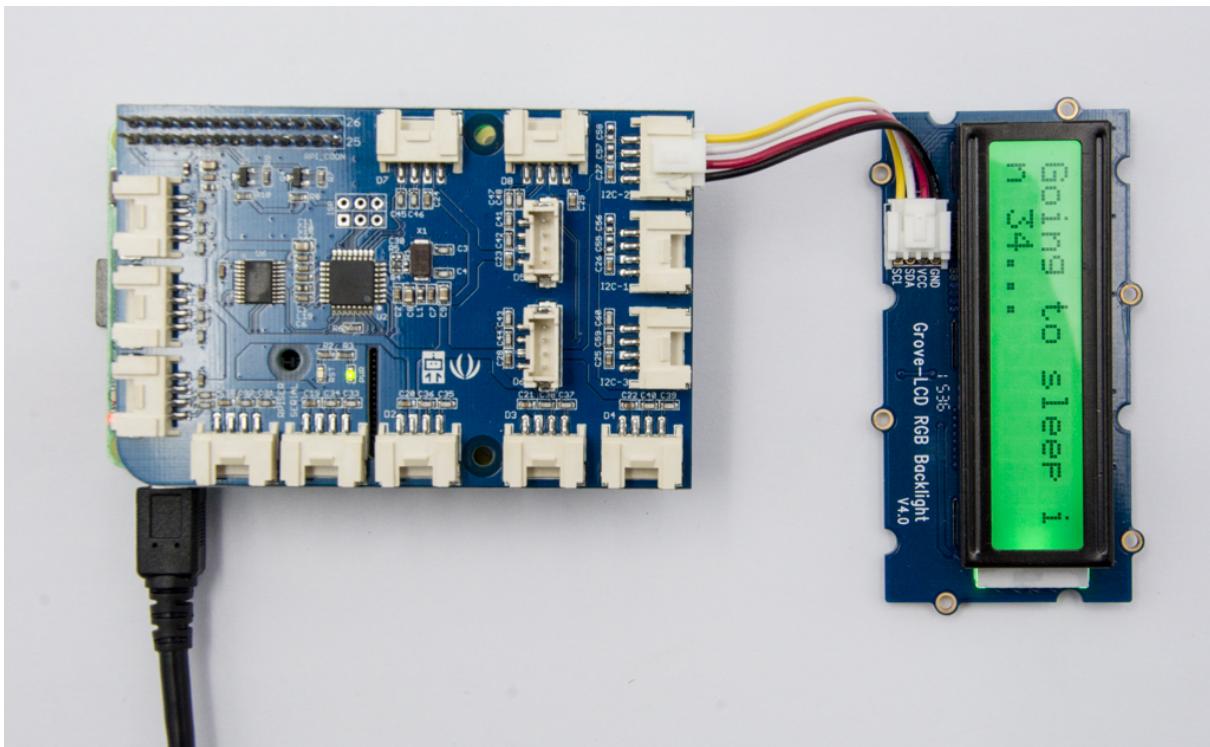
        # If loud, illuminate LED, otherwise dim
        if sensor_value > threshold_value:
            grovepi.digitalWrite(led,1)
        else:
            grovepi.digitalWrite(led,0)

        print("sensor_value = %d" %sensor_value)
        time.sleep(.5)

    except IOError:
        print ("Error")

```

3.13 Écran LCD.



L'écran vous permet de régler la couleur à votre guise via l'interface simple et concise de Grove. Il utilise I2C comme méthode de communication avec votre microcontrôleur. Ainsi, le nombre de broches nécessaires pour l'échange de données et le contrôle du rétroéclairage passe de 10 à 2, ce qui soulage les E/S pour d'autres tâches difficiles. En outre, Grove - LCD RGB Backlight prend en charge les caractères définis par l'utilisateur.

Connaître l'adresse i2c :

```
sudo i2cdetect -y 1
```

```
pi@rasp40:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          03 04 -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3e --
40: -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- 62 -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- --
pi@rasp40:~ $
```

Installation :

```
cd ~
git clone https://github.com/DexterInd/GrovePi.git
```

Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python/grove_rgb_lcd
```

Exécuter le programme python **grove_rgb_lcd.py** avec la commande :

```
python grove_rgb_lcd.py
```

Code :

```
import time,sys

if sys.platform == 'uwp':
    import winrt_smbus as smbus
    bus = smbus.SMBus(1)
else:
    import smbus
    import RPi.GPIO as GPIO
    rev = GPIO.RPI_REVISION
    if rev == 2 or rev == 3:
        bus = smbus.SMBus(1)
    else:
        bus = smbus.SMBus(0)

# this device has two I2C addresses
DISPLAY_RGB_ADDR = 0x62
DISPLAY_TEXT_ADDR = 0x3e

# set backlight to (R,G,B) (values from 0..255 for each)
def setRGB(r,g,b):
    bus.write_byte_data(DISPLAY_RGB_ADDR,0,0)
    bus.write_byte_data(DISPLAY_RGB_ADDR,1,0)
    bus.write_byte_data(DISPLAY_RGB_ADDR,0x08,0xaa)
    bus.write_byte_data(DISPLAY_RGB_ADDR,4,r)
    bus.write_byte_data(DISPLAY_RGB_ADDR,3,g)
    bus.write_byte_data(DISPLAY_RGB_ADDR,2,b)

# send command to display (no need for external use)
def textCommand(cmd):
    bus.write_byte_data(DISPLAY_TEXT_ADDR,0x80,cmd)

# set display text \n for second line(or auto wrap)
def setText(text):
    textCommand(0x01) # clear display
    time.sleep(.05)
    textCommand(0x08 | 0x04) # display on, no cursor
    textCommand(0x28) # 2 lines
    time.sleep(.05)
    count = 0
    row = 0
    for c in text:
```

```

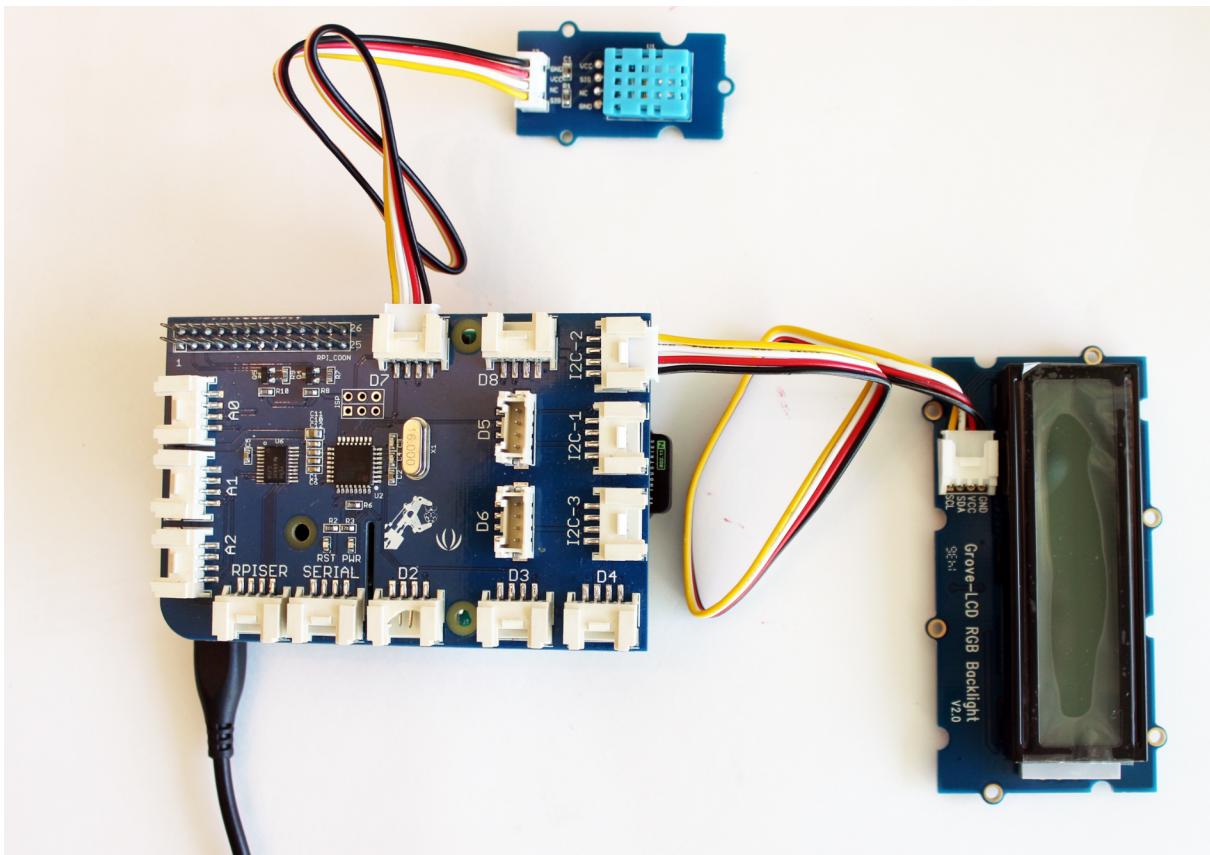
if c == '\n' or count == 16:
    count = 0
    row += 1
    if row == 2:
        break
    textCommand(0xc0)
    if c == '\n':
        continue
    count += 1
    bus.write_byte_data(DISPLAY_TEXT_ADDR,0x40,ord(c))

#Update the display without erasing the display
def setText_norefresh(text):
    textCommand(0x02) # return home
    time.sleep(.05)
    textCommand(0x08 | 0x04) # display on, no cursor
    textCommand(0x28) # 2 lines
    time.sleep(.05)
    count = 0
    row = 0
    while len(text) < 32: #clears the rest of the screen
        text += ''
    for c in text:
        if c == '\n' or count == 16:
            count = 0
            row += 1
            if row == 2:
                break
            textCommand(0xc0)
            if c == '\n':
                continue
            count += 1
            bus.write_byte_data(DISPLAY_TEXT_ADDR,0x40,ord(c))

# example code
if __name__=="__main__":
    setText("Hello world\nThis is an LCD test")
    setRGB(0,128,64)
    time.sleep(2)
    for c in range(0,255):
        setText_norefresh("Going to sleep in {}...".format(str(c)))
        setRGB(c,255-c,0)
        time.sleep(0.1)
    setRGB(0,255,0)
    setText("Bye bye, this should wrap onto next line")

```

3.14 Écran LCD avec capteur de température et d'humidité Pro.



Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Projects/Advanced_RGB_LCD_TempAndHumidity

Exécuter le programme python **grovepi_lcd_dht.py** avec la commande :

python grovepi_lcd_dht.py

Deux modifications dans le fichier sont nécessaires pour afficher la température en degré Celcius.

```
fimport decimal
from grovepi import *
from grove_rgb_lcd import *

dht_sensor_port = 7    # Connect the DHT sensor to port 7
lastTemp = 0.1          # initialize a floating point temp variable
lastHum = 0.1            # initialize a floating Point humidity variable
tooLow = 62.0           # Lower limit in fahrenheit
justRight = 68.0         # Perfect Temp in fahrenheit
tooHigh = 74.0           # Temp Too high
```

```
# Function Definitions
def CtoF( tempc ):
    "This converts celcius to fahrenheit"
    tempf = round((tempc * 1.8) + 32, 2);
```

```

return tempf;

def FtoC( tempf ):
    "This converts fahrenheit to celcius"
    tempc = round((tempf - 32) / 1.8, 2)
    return tempc;

def calcColorAdj(variance):    # Calc the adjustment value of the background color
    "Because there is 6 degrees mapping to 255 values, 42.5 is the factor for 12 degree spread"
    factor = 42.5;
    adj = abs(int(factor * variance));
    if adj > 255:
        adj = 255;
    return adj;

def calcBG(ftemp):
    "This calculates the color value for the background"
    variance = ftemp - justRight;  # Calculate the variance
    adj = calcColorAdj(variance);  # Scale it to 8 bit int
    bgList = [0,0,0]              # initialize the color array
    if(variance < 0):
        bgR = 0;                  # too cold, no red
        bgB = adj;                 # green and blue slide equally with adj
        bgG = 255 - adj;
    elif(variance == 0):          # perfect, all on green
        bgR = 0;
        bgB = 0;
        bgG = 255;
    elif(variance > 0):          #too hot - no blue
        bgB = 0;
        bgR = adj;                # Red and Green slide equally with Adj
        bgG = 255 - adj;

    bgList = [bgR,bgG,bgB]       #build list of color values to return
    return bgList;

while True:

    try:
        temp = 0.01
        hum = 0.01
        [ temp,hum ] = dht(dht_sensor_port,1)      #Get the temperature and Humidity from the
DHT sensor
                                                #Change the second parameter to 0 when using DHT (instead
of DHT Pro)
                                                #You will get very large number values if you don't!
        if (CtoF(temp) != lastTemp) and (hum != lastHum) and not math.isnan(temp) and not
math.isnan(hum):

```

```

        print("lowC : ",FtoC(tooLow),"C\t\t","rightC : ", FtoC(justRight),"C\t\t","highC : "
",FtoC(tooHigh),"C") # comment these three lines
        print("lowF : ",tooLow,"F\t\tjustRight : ",justRight,"F\t\ttooHigh : ",tooHigh,"F")
# if no monitor display
        print("tempC : ", temp, "C\t\ttempF : ",CtoF(temp),"F\t\tHumidity =", hum,"%\r\n")

    lastHum = hum      # save temp & humidity values so that there is no update to
the RGB LCD
    ftemp = CtoF(temp)  # unless the value changes
    lastTemp = ftemp    # this reduces the flashing of the display
    # print "ftemp = ",ftemp," temp = ",temp  # this was just for test and debug

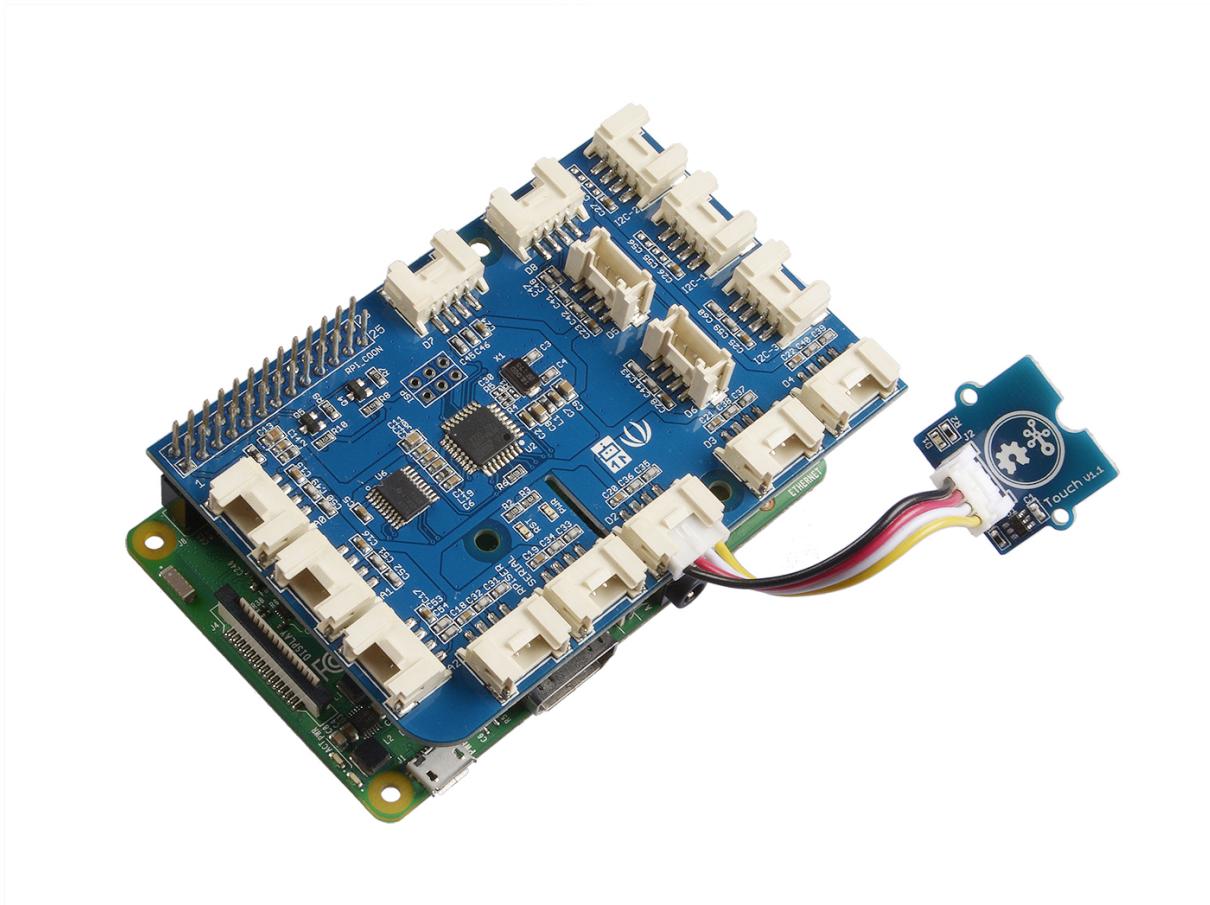
    bgList = calcBG(ftemp)      # Calculate background colors

t = str(temp)  # "stringify" the display values
# t = str(ftemp)  # "stringify" the display values
    h = str(hum)
    # print "(",bgList[0],",",bgList[1],",",bgList[2],")"  # this was to test and debug
color value list
    setRGB(bgList[0],bgList[1],bgList[2])  # parse our list into the color settings
    setText("Temp:" + t + "\n" + "Humidity :" + h + "%") # update the RGB LCD
display

except (IOError,TypeError) as e:
    print("Error" + str(e))

```

3.15 Capteur tactile.



Grove - Touch Sensor peut détecter le changement de capacité lorsqu'un doigt est à proximité. Cela signifie que peu importe que votre doigt touche directement le pad ou reste juste près du pad, Grove - Touch Sensor émet également HIGH.

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python

Exécuter le programme python **grove_touch_sensor.py** avec la commande :
python grove_touch_sensor.py

Code :

```
import time
import grovepi

# Connect the Grove Touch Sensor to digital port D4
# SIG,NC,VCC,GND
touch_sensor = 4

grovepi.pinMode(touch_sensor,"INPUT")

while True:
    try:
```

```

print(grovepi.digitalRead(touch_sensor))
time.sleep(.5)

except IOError:
    print ("Error")

```

Modification du programme pour allumer une led lors de l'appui

Programme python **grove_touch_sensor_led.py** avec la commande :
python grove_touch_sensor_led.py

Code :

```

import time
import grovepi

# Connect the Grove Touch Sensor to digital port D4
# SIG,NC,VCC,GND
touch_sensor = 4

grovepi.pinMode(touch_sensor,"INPUT")

# Connect the Grove LED to digital port D5
led = 5

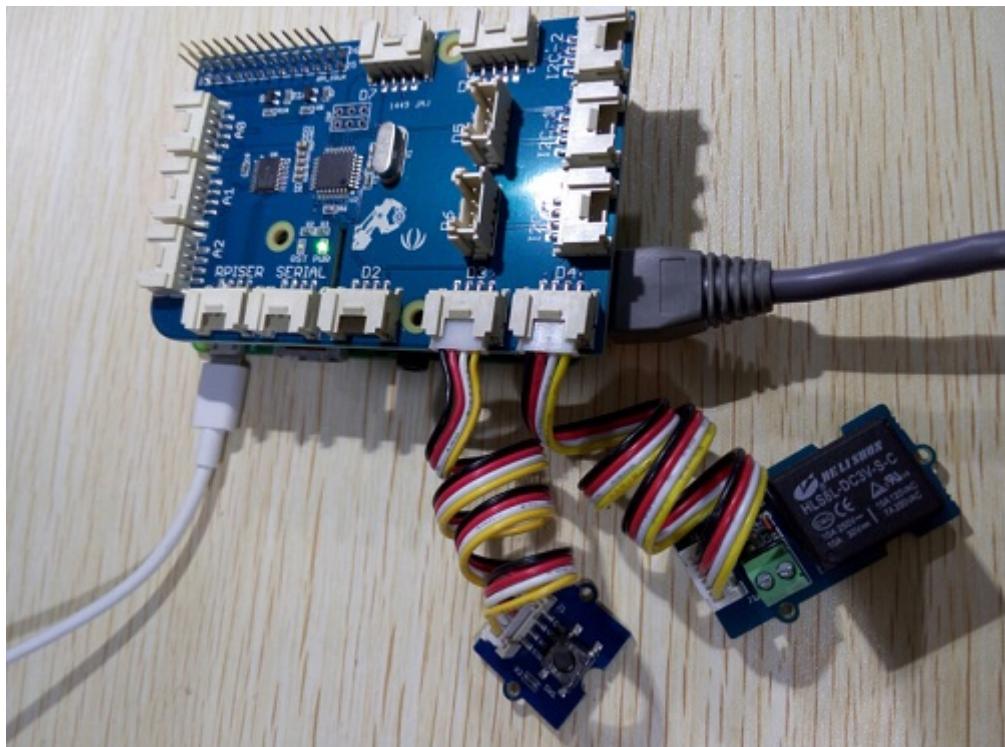
grovepi.pinMode(led,"OUTPUT")
time.sleep(1)

while True:
    try:
        touch = grovepi.digitalRead(touch_sensor)
        if touch == 1:
            #Blink the LED
            grovepi.digitalWrite(led,1)          # Send HIGH to switch on LED
            print ("LED ON!")
            time.sleep(1)
        else:
            grovepi.digitalWrite(led,0)          # Send LOW to switch off LED
            print ("LED OFF!")
            time.sleep(1)

    except IOError:
        print ("Error")

```

3.16 Relais.



Le module Grove-Relay est un commutateur numérique normalement ouvert. Grâce à lui, vous pouvez contrôler le circuit de haute tension avec basse tension, disons 5V sur le contrôleur. Il y a un voyant LED sur la carte, qui s'allume lorsque les terminaux contrôlés se ferment.

Changer de répertoire courant :

```
cd /home/pi/Dexter/GrovePi/Software/Python/
```

Exécuter le programme python **grove_relay.py** avec la commande :
python grove_relay.py 4

code :

```
import time
import grovepi

# Connect the Grove Relay to digital port D4
# SIG,NC,VCC,GND
relay = 4

grovepi.pinMode(relay,"OUTPUT")

while True:
    try:
        # switch on for 5 seconds
        grovepi.digitalWrite(relay,1)
        print ("on")
        time.sleep(5)
```

```

# switch off for 5 seconds
grovepi.digitalWrite(relay,0)
print ("off")
time.sleep(5)

except KeyboardInterrupt:
    grovepi.digitalWrite(relay,0)
    break
except IOError:
    print ("Error")

```

Modification du programme avec un bouton pour déclencher le relais :

```

# Raspberry Pi + Grove Switch + Grove Relay

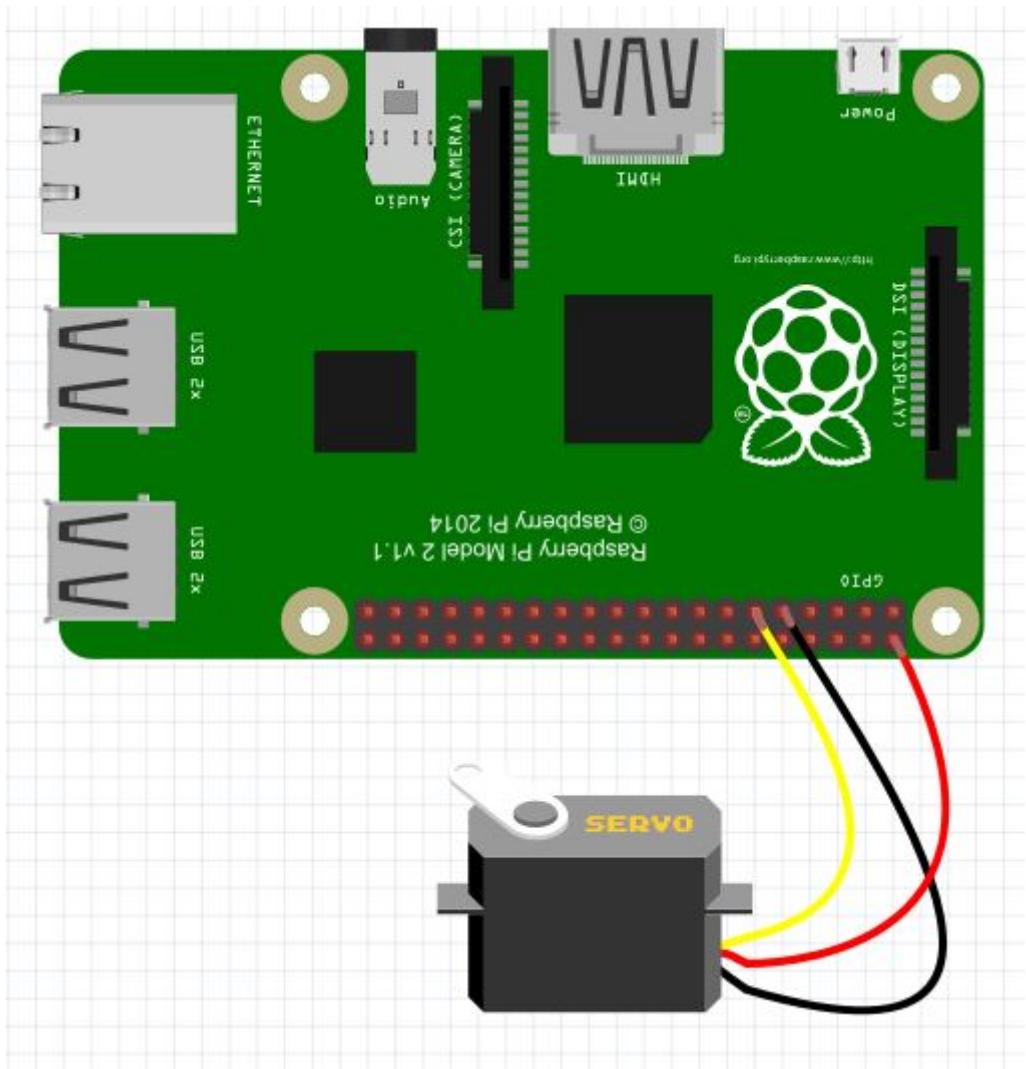
import time
import grovepi
# Connect the Grove Switch to digital port D3
# SIG,NC,VCC,GND

switch = 3
# Connect the Grove Relay to digital port D4
# SIG,NC,VCC,GND

relay = 4
grovepi.pinMode(switch,"INPUT")
grovepi.pinMode(relay,"OUTPUT")
while True:
    try:
        if grovepi.digitalRead(switch):
            grovepi.digitalWrite(relay,1)
        else:
            grovepi.digitalWrite(relay,0)
            time.sleep(.05)
    except KeyboardInterrupt:
        grovepi.digitalWrite(relay,0)
        break
    except IOError:
        print "Error"

```

3.17 Analog Servo.



Grove - Servo est un moteur à courant continu avec système d'engrenage et de rétroaction. Il est utilisé dans le mécanisme d'entraînement des robots.

télécharger le fichier exemple :

git clone https://github.com/Seeed-Studio/grove.py

Changer de répertoire courant :

cd /home/pi/Dexter/GrovePi/Software/Python/

Créer le programme python **grove_analog_servo.py** avec la commande :

nano grove_analog_servo.py 11

Exécuter le programme python **grove_analog_servo.py** avec la commande :

python grove_analog_servo.py 11

Code :

```
import RPi.GPIO as IO  
import sys  
import time
```

```

from numpy import interp

IO.setwarnings(False)
IO.setmode(IO.BCM)

class GroveServo:
    MIN_DEGREE = 0
    MAX_DEGREE = 180
    INIT_DUTY = 2.5

    def __init__(self, channel):
        IO.setup(channel, IO.OUT)
        self.pwm = IO.PWM(channel, 50)
        self.pwm.start(GroveServo.INIT_DUTY)

    def __del__(self):
        self.pwm.stop()

    def setAngle(self, angle):
        # Map angle from range 0 ~ 180 to range 25 ~ 125
        angle = max(min(angle, GroveServo.MAX_DEGREE), GroveServo.MIN_DEGREE)
        tmp = interp(angle, [0, 180], [25, 125])
        self.pwm.ChangeDutyCycle(round(tmp/10.0, 1))

Grove = GroveServo

def main():
    if len(sys.argv) < 2:
        print('Usage: {} servo_channel'.format(sys.argv[0]))
        sys.exit(1)

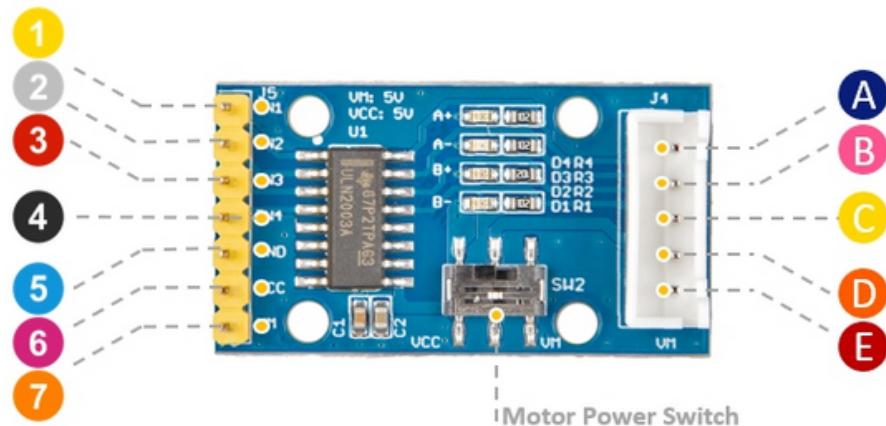
    servo = GroveServo(int(sys.argv[1]))

    while True:
        for x in range(0, 180):
            print x, "degree"
            servo.setAngle(x)
            time.sleep(0.05)
        for x in range(180, 0, -1):
            print x, "degree"
            servo.setAngle(x)
            time.sleep(0.05)

if __name__ == '__main__':
    main()

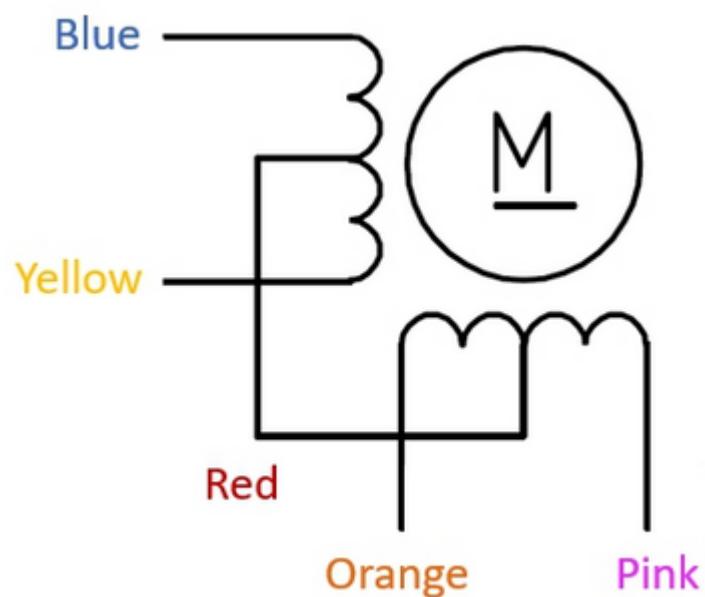
```

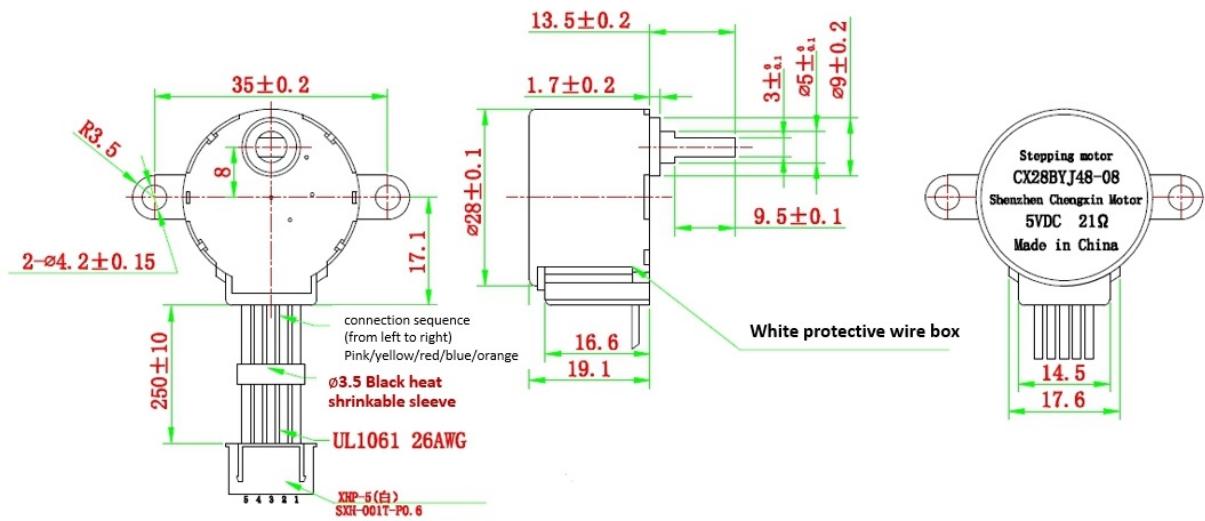
3.17 Stepping motor.



- ① IN1: input 1 connect to phrase 1
- ② IN2: input 2 connect to phrase 2
- ③ IN3: input 3 connect to phrase 3
- ④ IN4: input 4 connect to phrase 4
- ⑤ GND: connect this module to the system GND
- ⑥ VCC: we use 5v for this module
- ⑦ VM: optional power for Motor

- A connect D4 to Blue coil
- B connect D3 to Pink coil
- C connect D2 to Yellow coil
- D connect D1 to Orange coil
- E Connect Power switch pin to Red coil





Le pack pilote de moteur pas à pas comprend un moteur pas à pas et une carte de commande de moteur. Il s'agit d'un moteur pas à pas à quatre phases à huit phases, et vous pouvez facilement contrôler ce moteur pas à pas via la carte d'entraînement.