嵌入式系統設計概論與實作

曾煜棋、吳昆儒、張凌燕

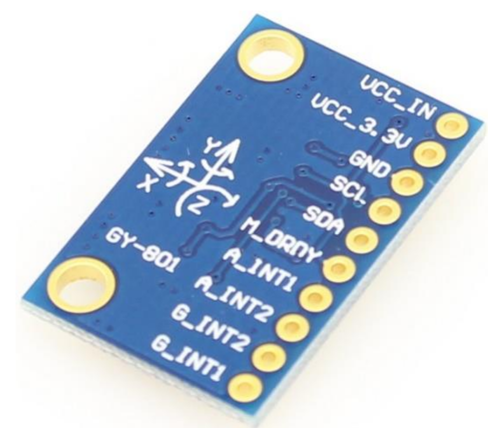
National Chiao Tung University

Outline

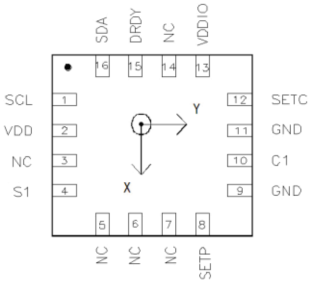
- 嵌入式應用: 人體活動偵測
  - 加速度、陀螺儀…等

- GY801 (I2C sensor)
  - 3-axis Accelerometer, Gyroscope, magnetometer and pressure

1. ADXL345：Accelerometer
2. L3G4200：Gyroscope
3. HMC5883：Magnetometer
4. BMP085：Pressure

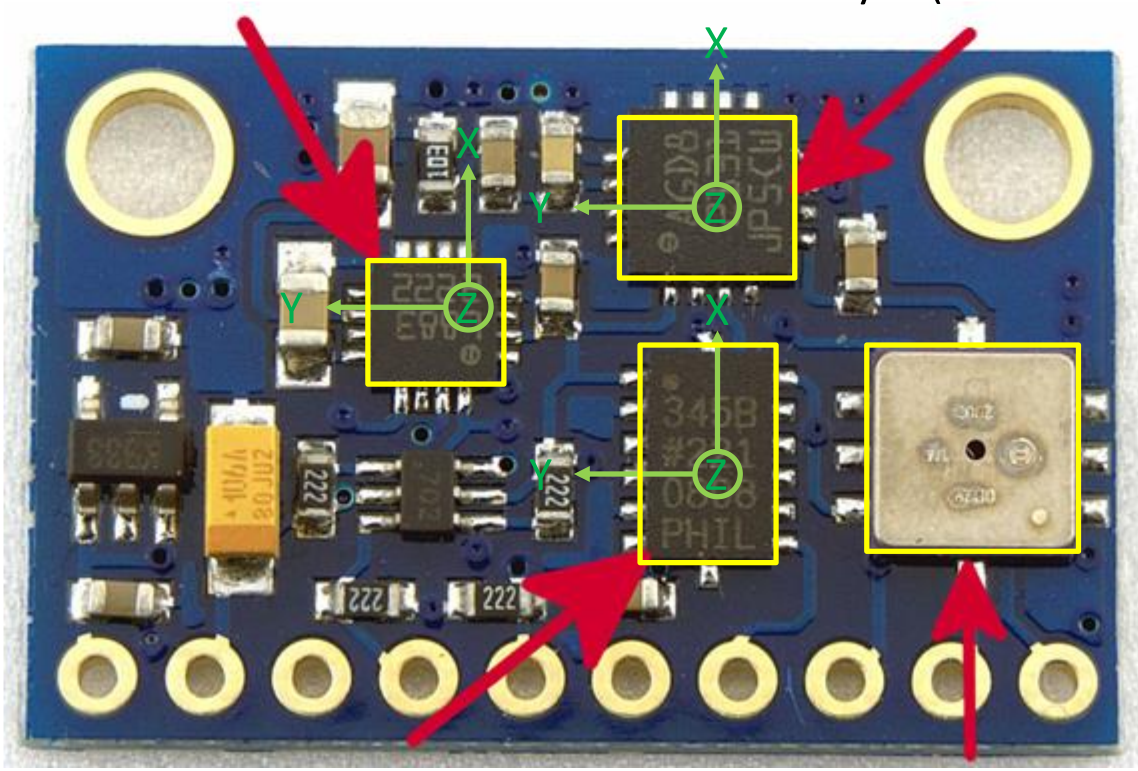


GY-801 (10 DoF sensor)



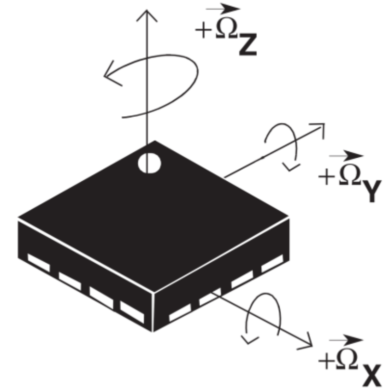
Compass (HMC5883L)

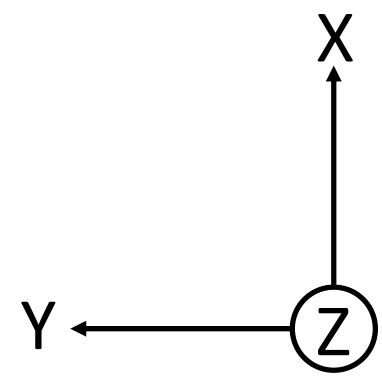
Gyro (L3G4200D)

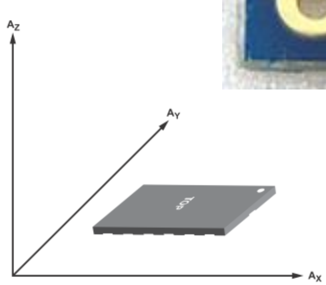


G-sensor (ADXL345)

Pressure (BMP085)

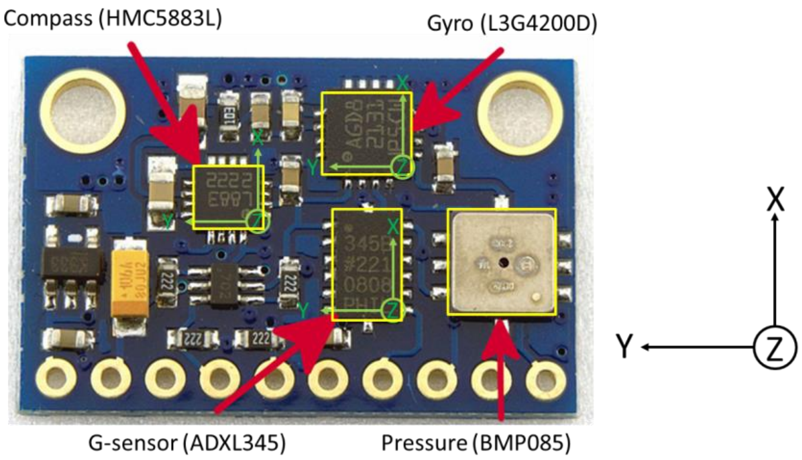


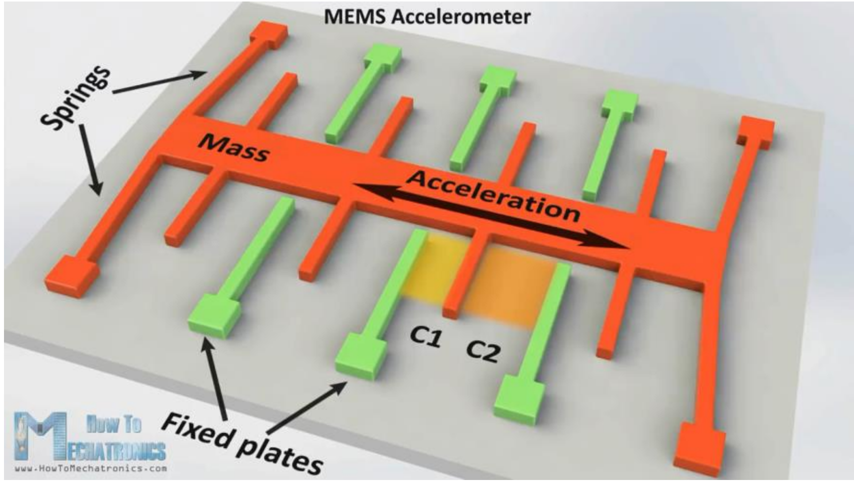


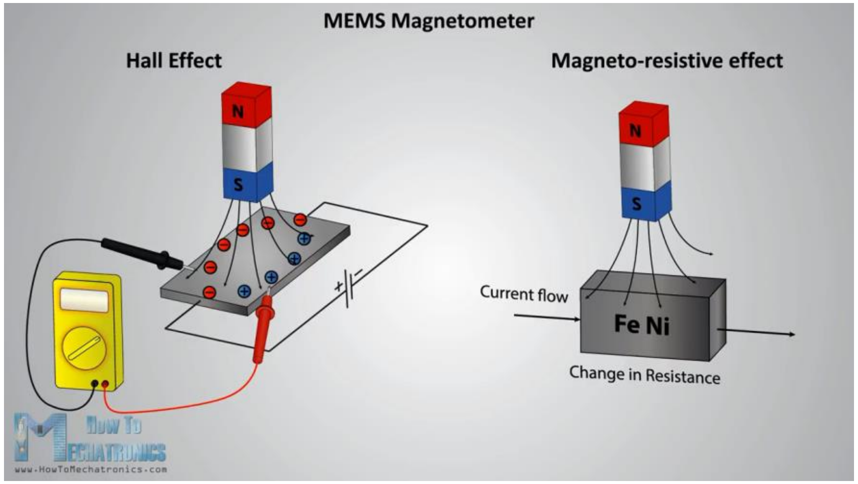


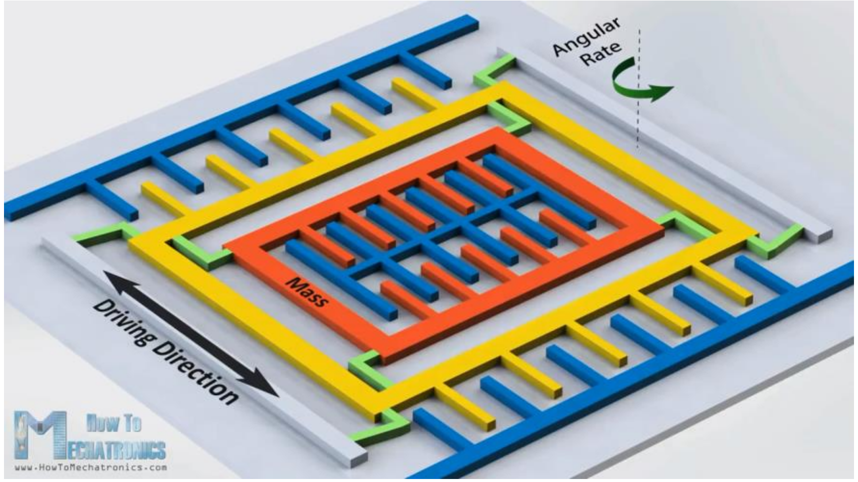
How MEMS Accelerometer, Gyroscope, Magnetometer Work

https://www.youtube.com/watch?v=eqZgxR6eRjo



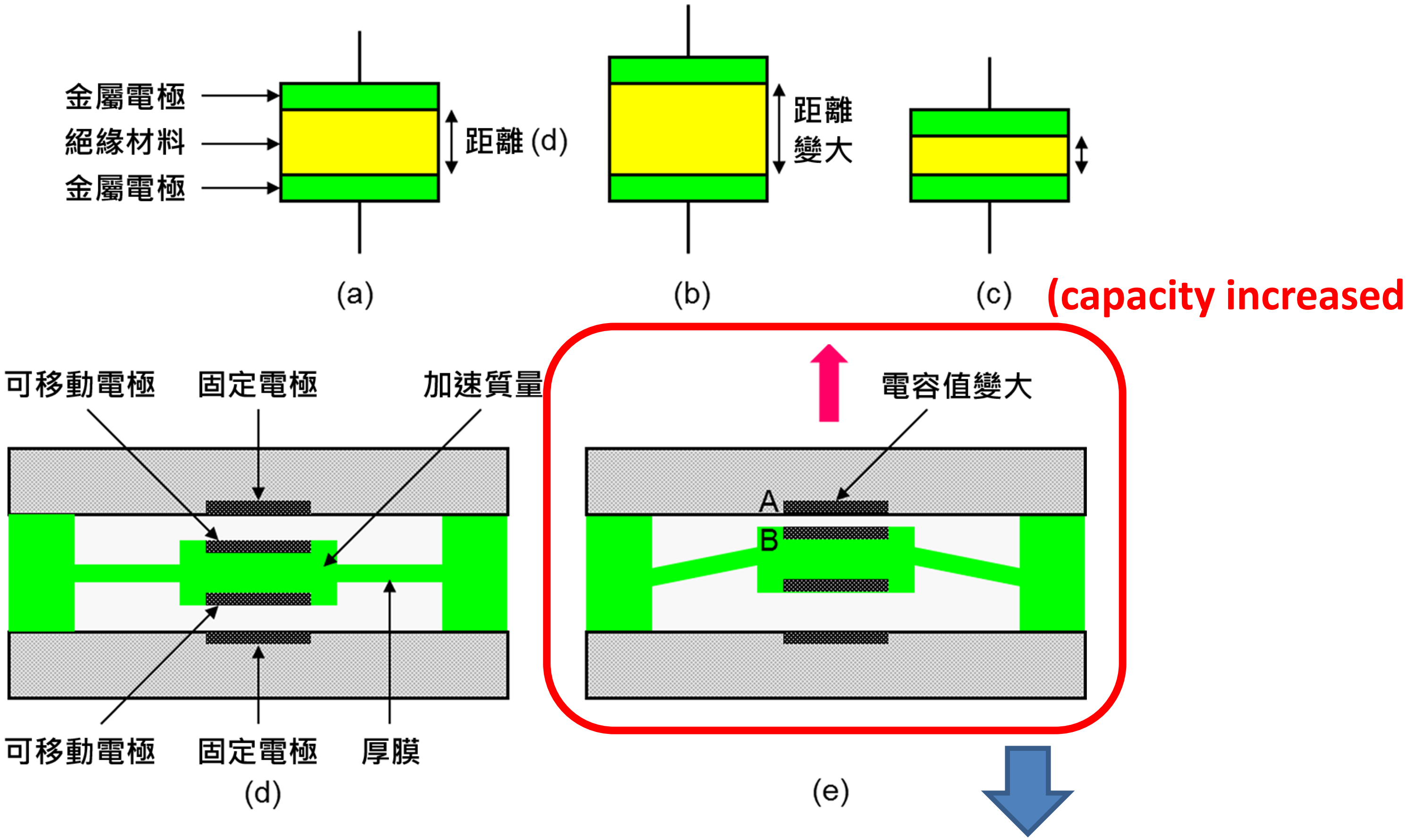






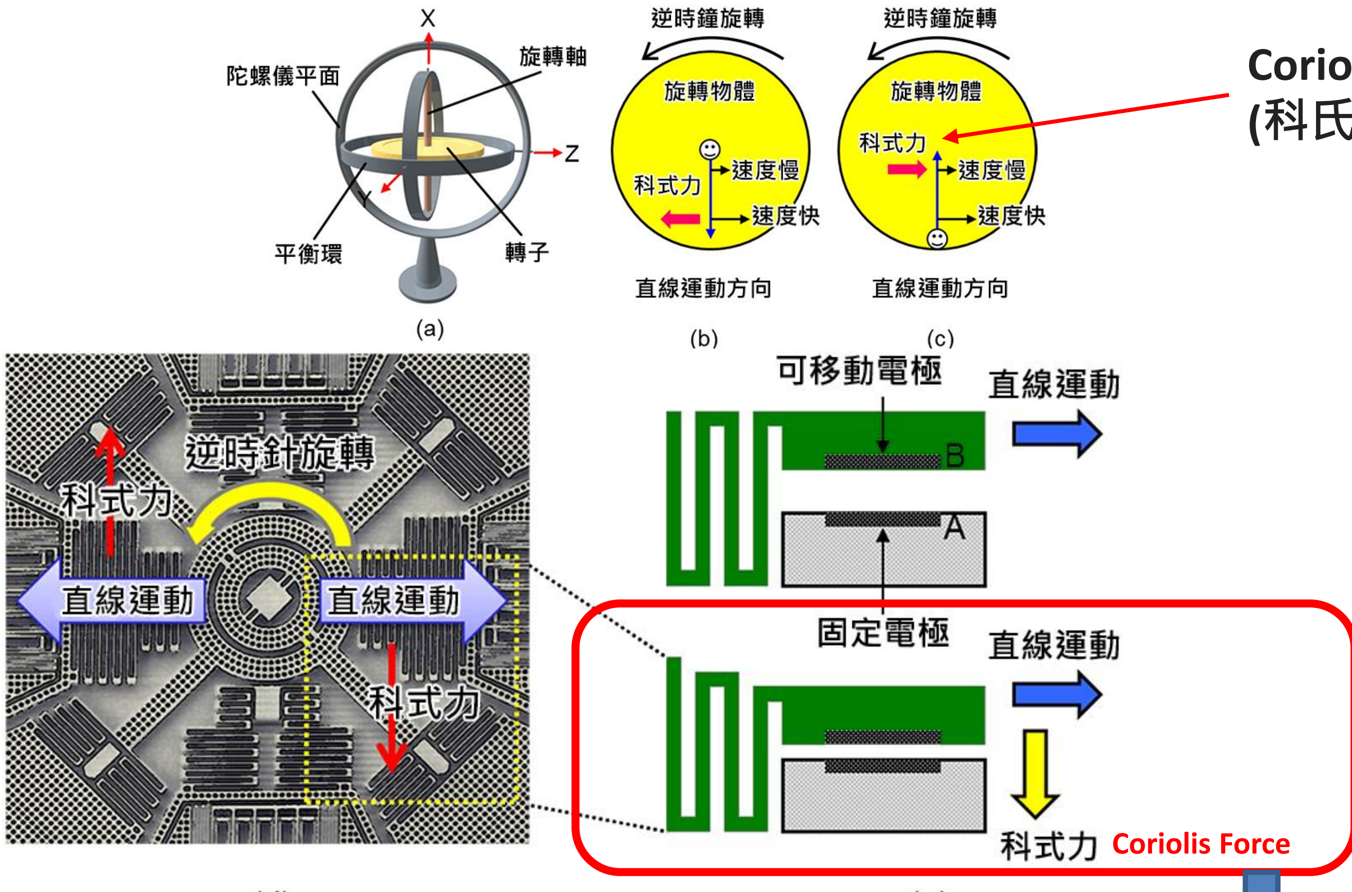
IMU. How MEMS Accelerometer Gyroscope Magnetometer Work & Arduino Tutorial

Sensor - Accelerometer



When you move the sensor, the distance (**capacity**) is changed.
We **use capacity to calculate acceleration**.

Sensor - Gyro



(a) (b) (c)

(d) (e)

Coriolis Force (科氏力)

When you rotate the sensor, Coriolis Force change the distance (**capacity**).

We **use capacity to calculate Angular velocity (角速度).**

微地磁計的構造與原理
http://scimonth.blogspot.tw/2014/07/blog-post_94.html

Sensor - Magnetomet

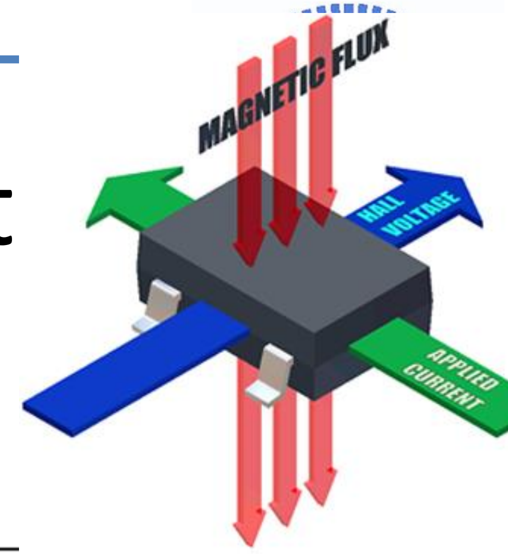
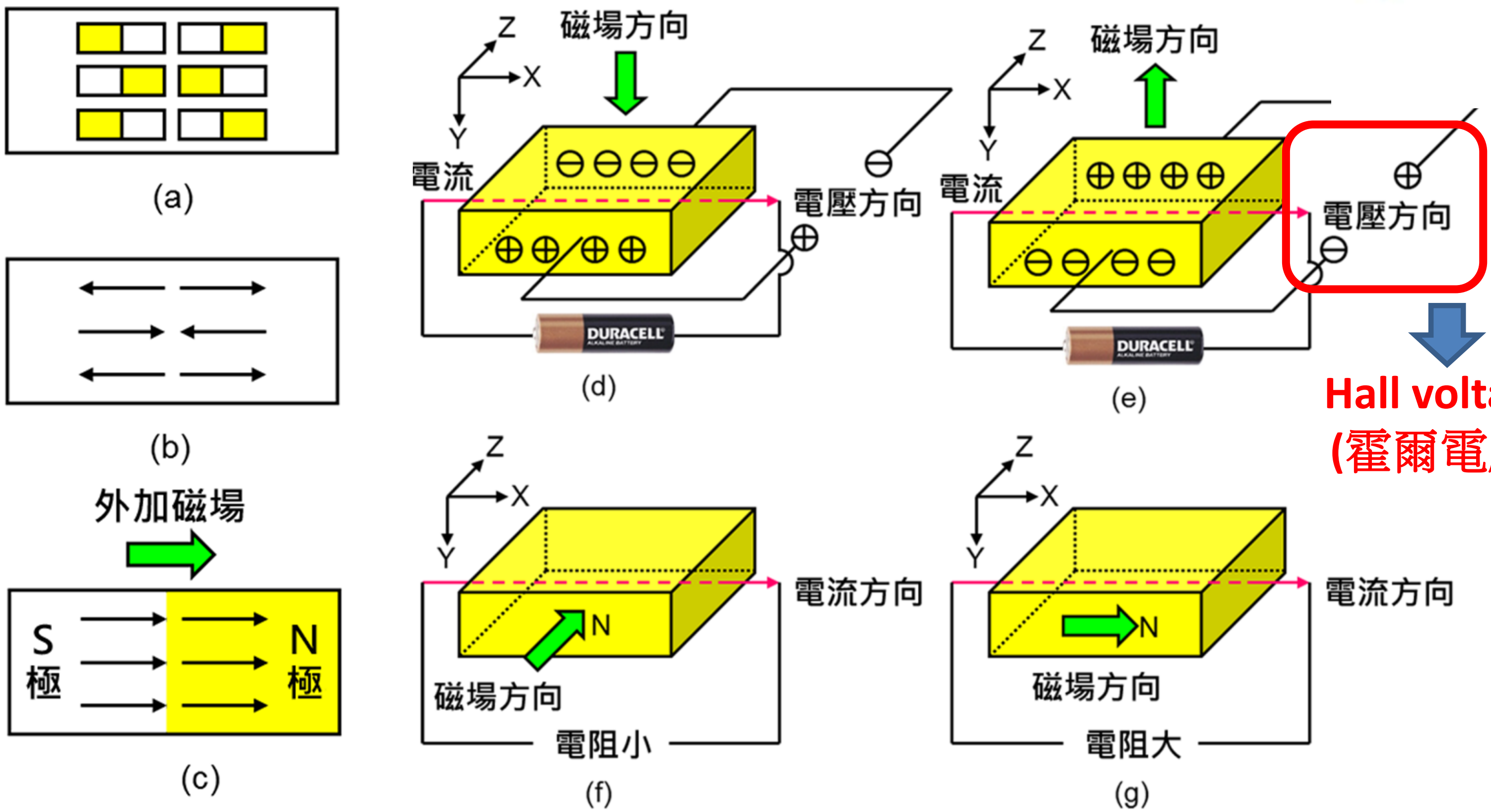


Fig: https://www.ecnmag.com/article/2009/03/sensor-zone-april-2009

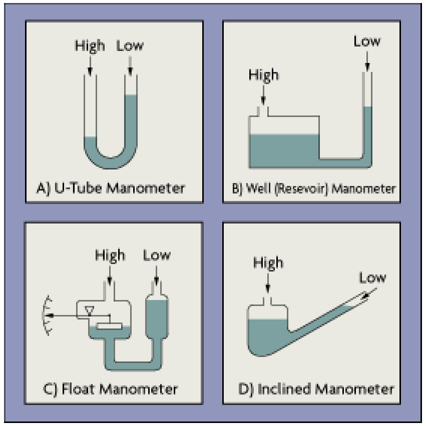


Hall voltage (霍爾電壓)

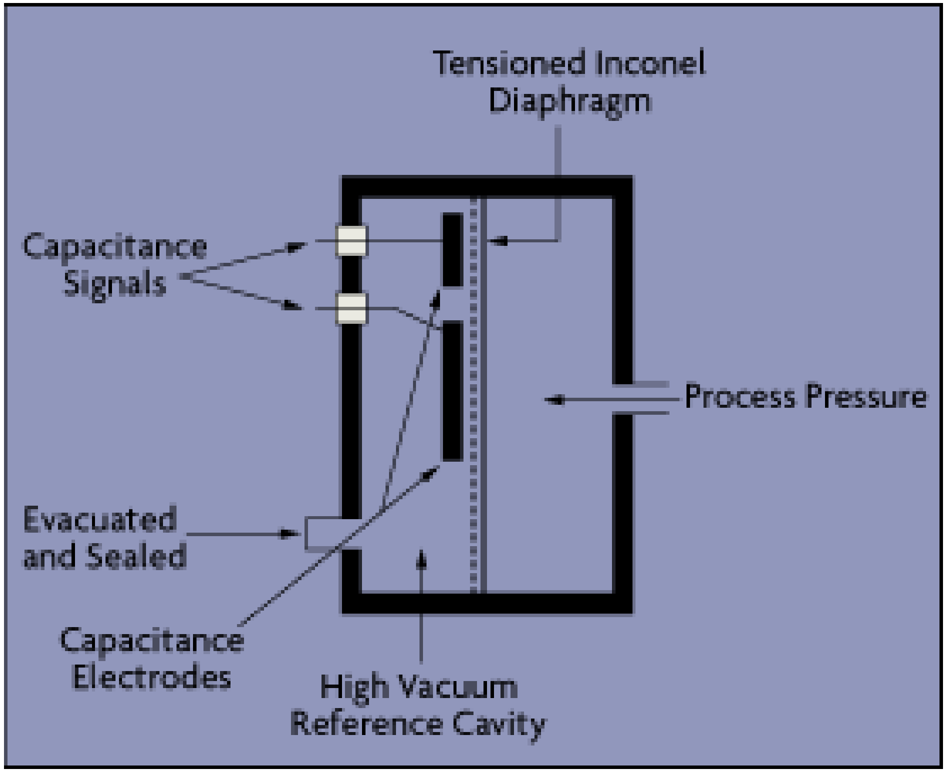
Measure the absolute **magnetic intensity** based on **Hall voltage (霍爾電壓)**

https://www.omega.com/literature/transactions/volume3/high3.html
https://www.electronicspecifier.com/sensors/a-quick-guide-to-choosing-a-pressure-sensor

Sensor - Pressure

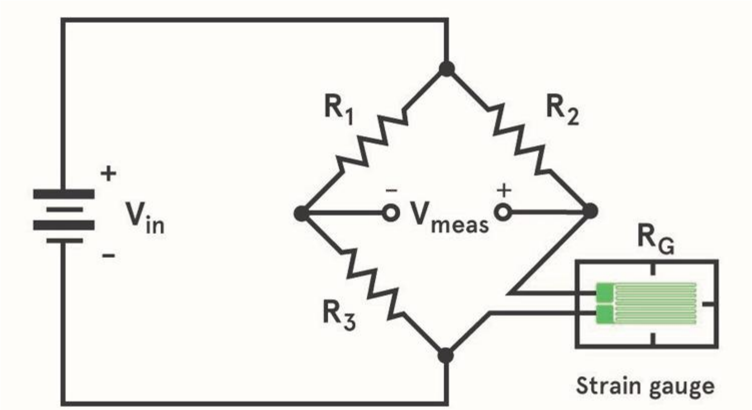


Manometer



Capacitance Vacuum Manometer

Measure the **varying capacity** based on **pressure**.

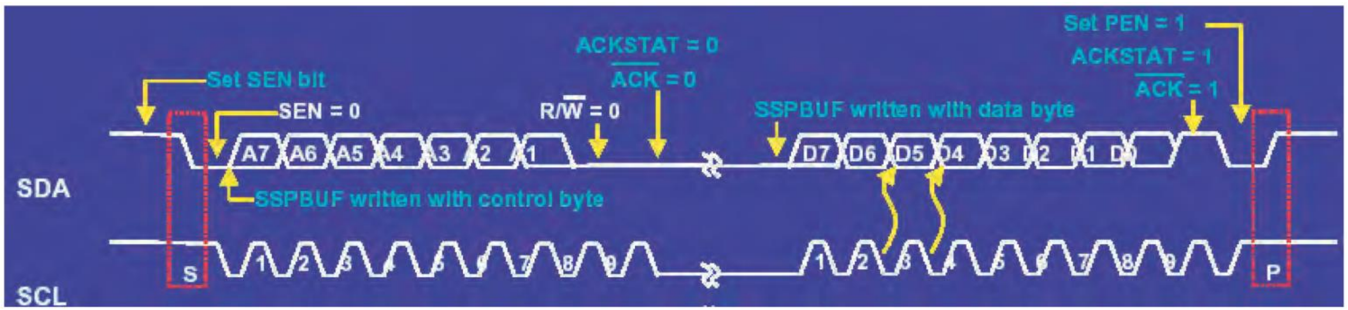


Use Wheatstone bridge to measure the resistance.

這個念square喔!!

I2C

- I2C全名為Inter-IC，它是一種半雙工同步多組設備匯流排，只需要兩條信號線：串列資料線 (SDA) 及串列時脈線 (SCL)。

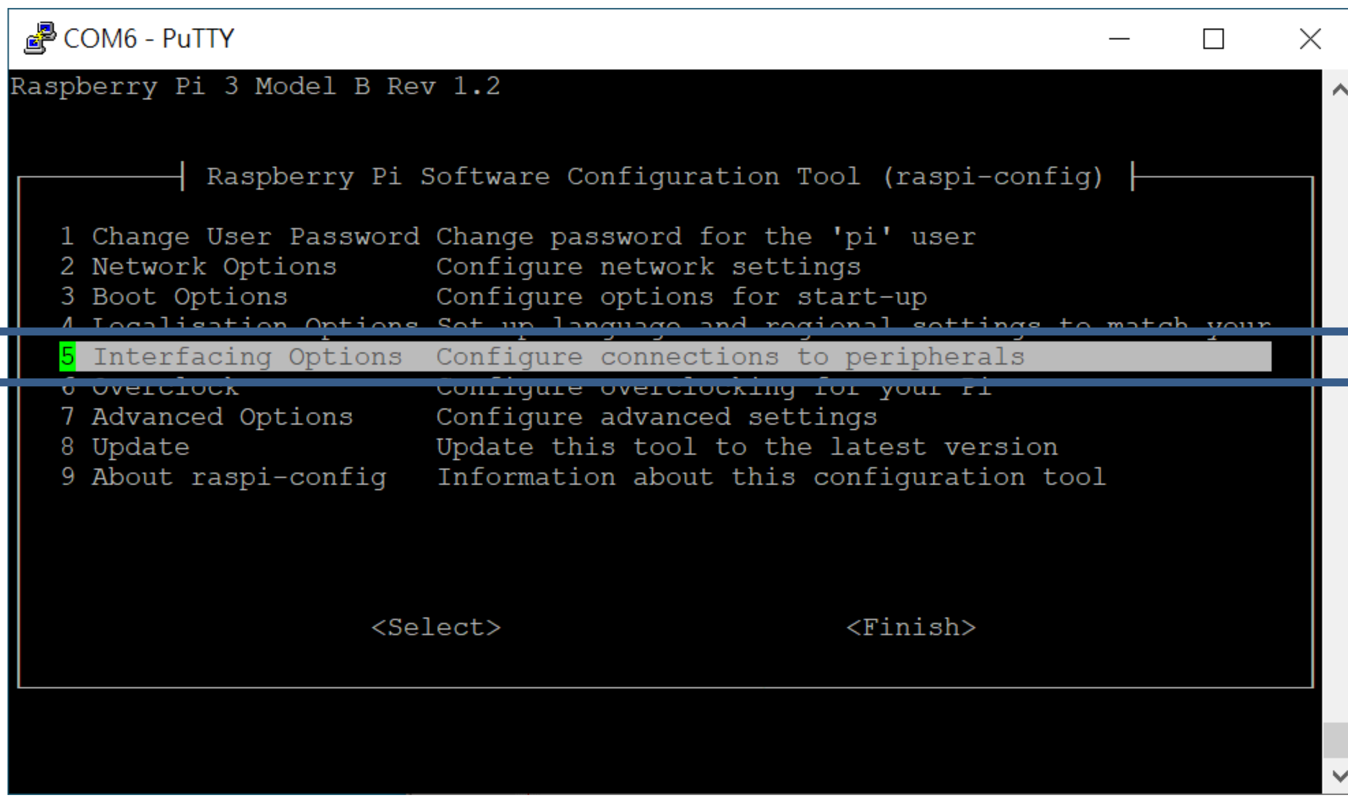


- 在傳送資料過程中共有三種類型信號，分別是：開始信號、結束信號和應答信號。
  - 開始信號：SCL為高電位時，SDA由高電位降為低電位，開始傳送資料。
  - 結束信號：SCL為高電位時，SDA由低電位升為高電位，結束傳送資料。
  - 應答信號：收到 8bit 資料後，向發送資料的IC發出特定的低電位脈衝
- 資料讀取方式：
  - 當 SCL 由低電位升為高電位時，讀取 SDA 的資料。

http://www.leaptronix.com/PDF/introducing_of_I2C.pdf

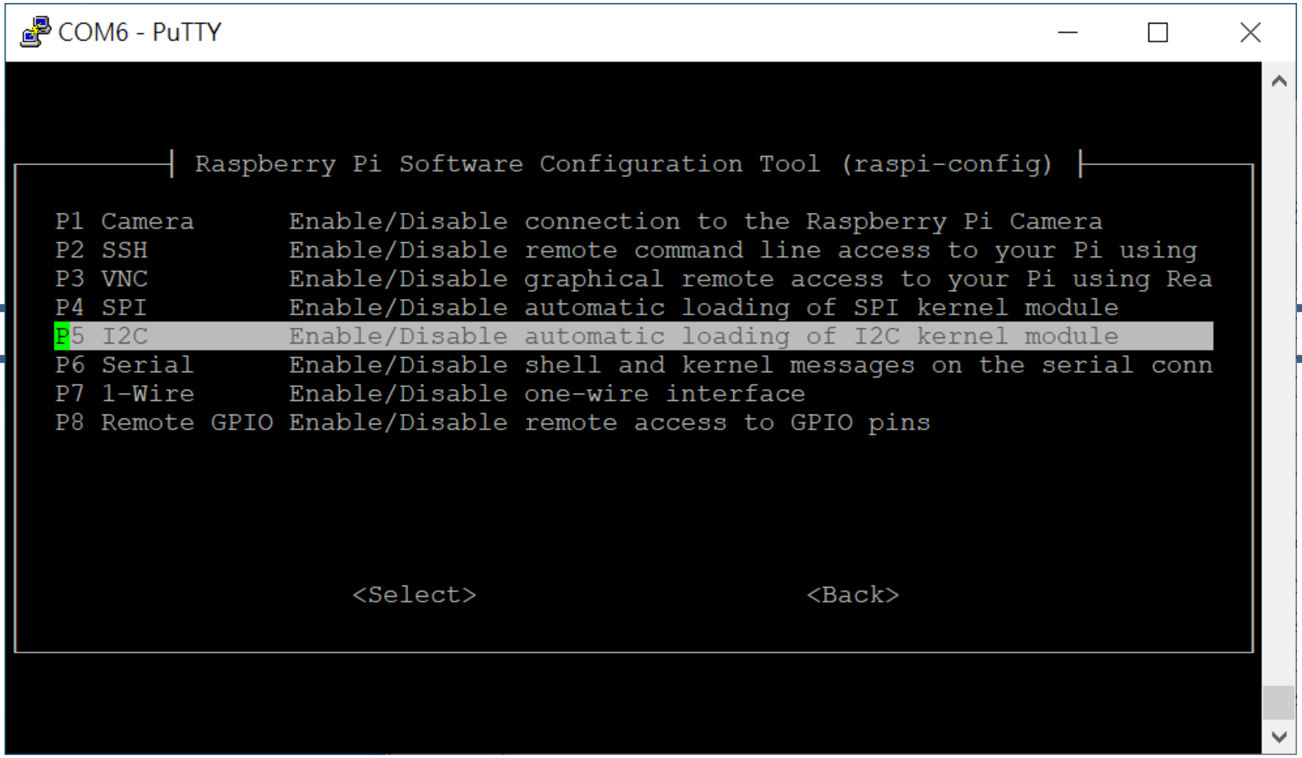
Enable I2C on Raspberry PI

- Enter the command：**sudo raspi-config**
  - Select **5 Interfacing Options**



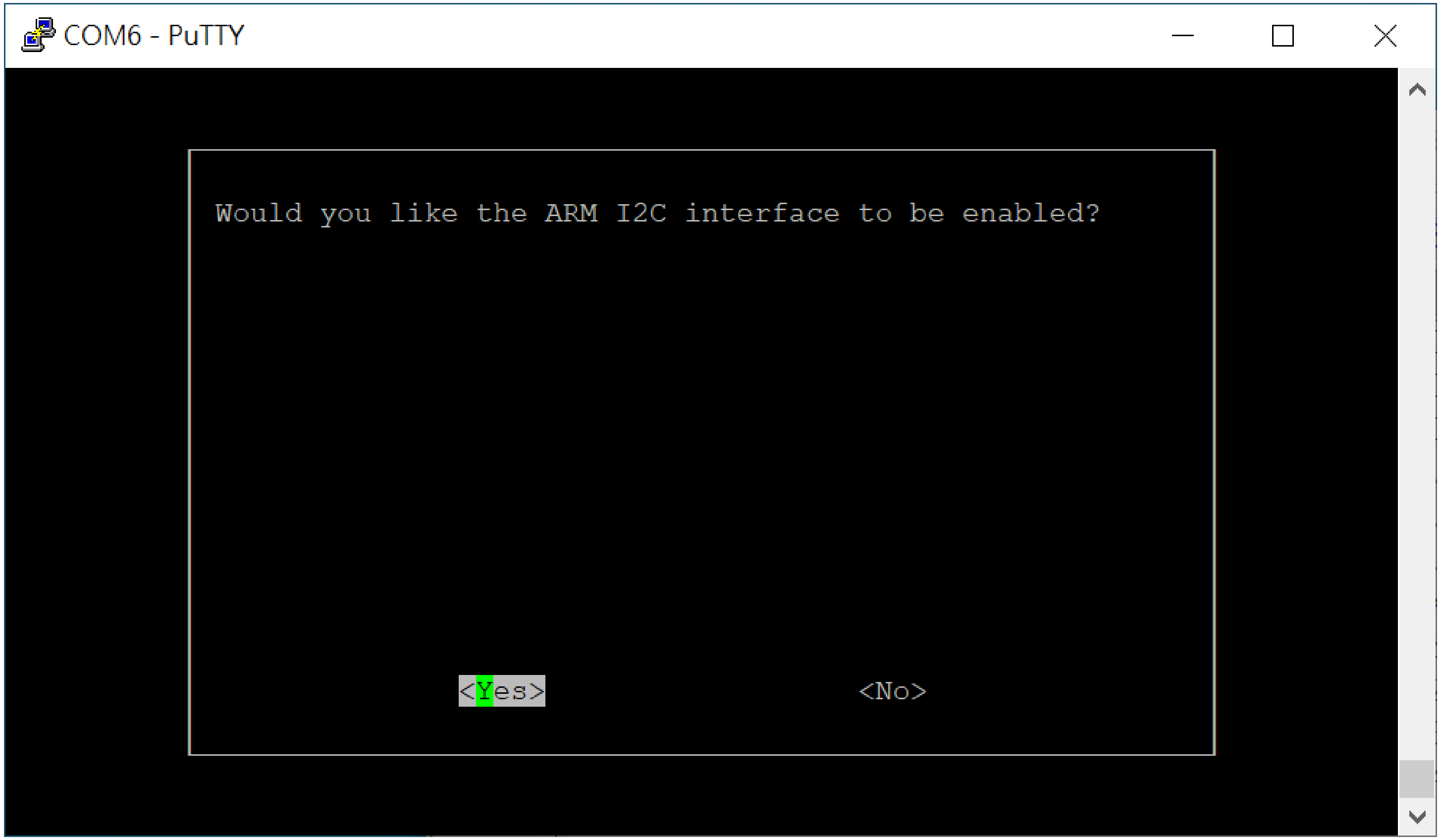
Enable I2C on RaspPI

- Select **P5 I2C**



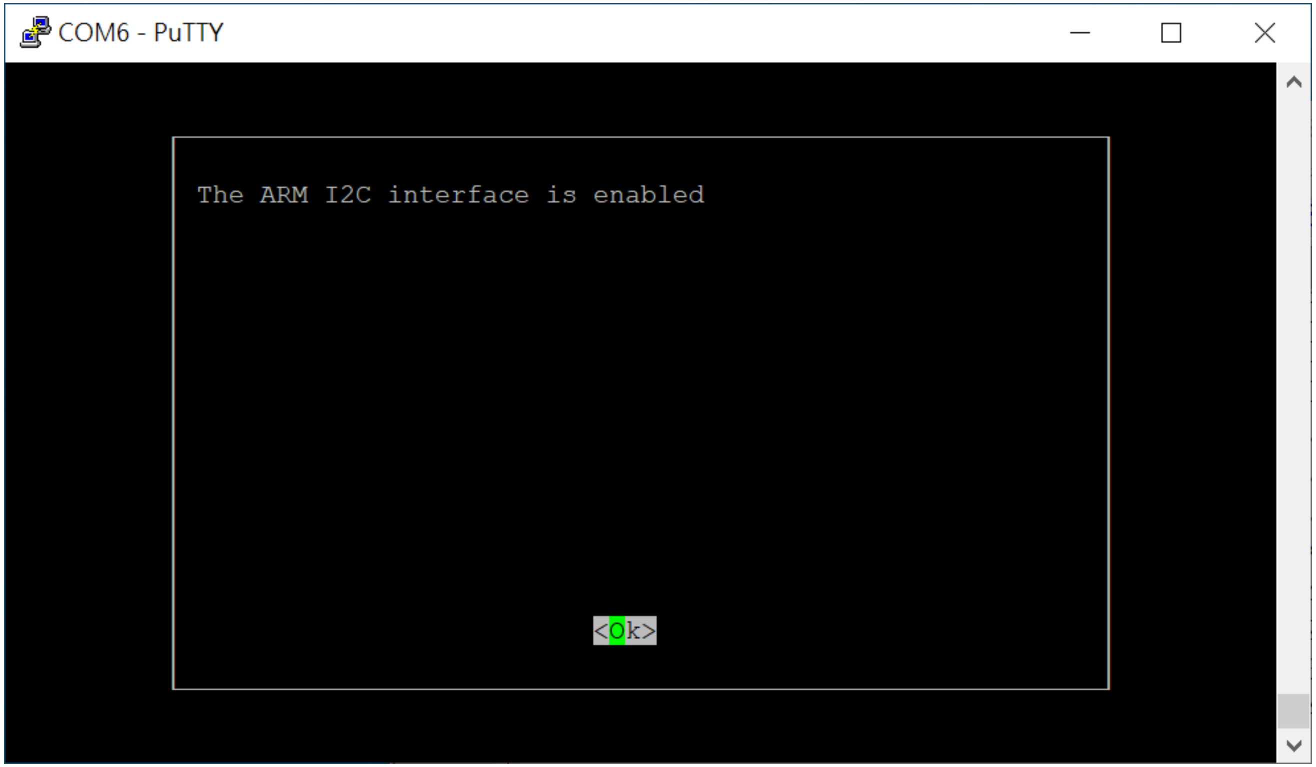
Enable I2C on RaspPI

- Choose **Yes**



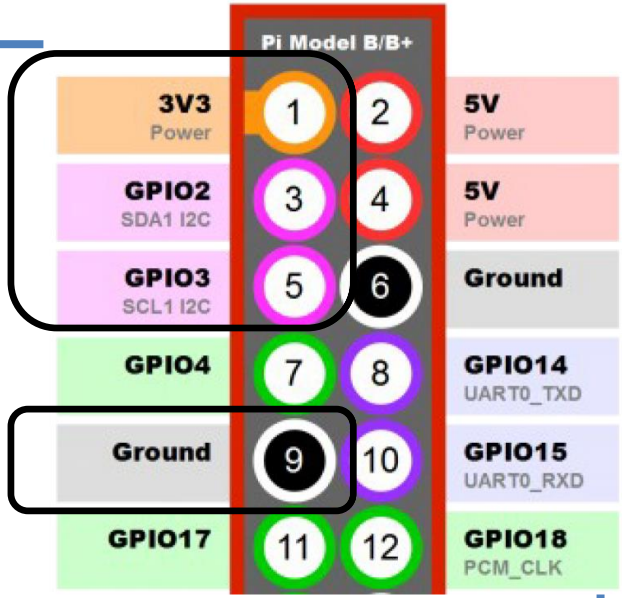
Enable I2C on RaspPI

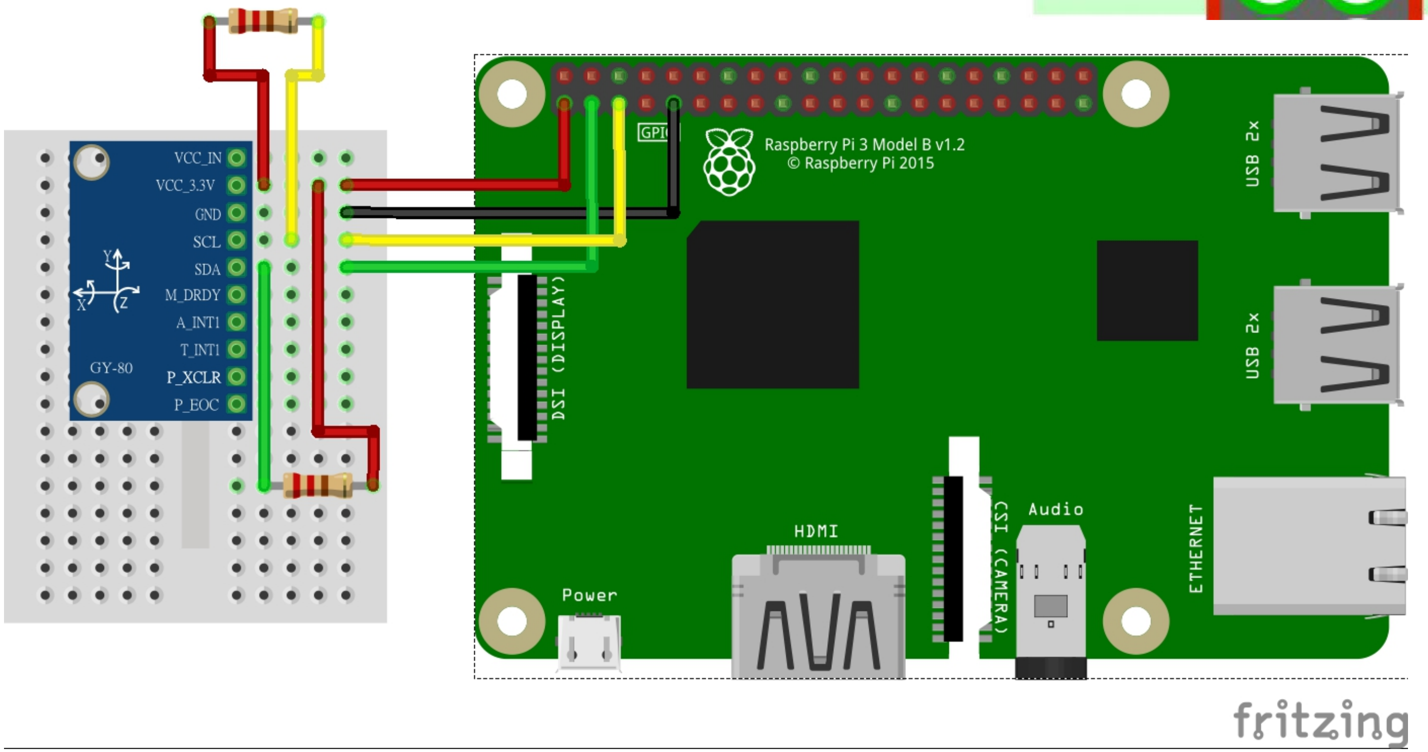
- I2C is enabled



Connect GY801

VCC	**Pin 1 (3.3V), Red line**
GND	**Pin 9 (Ground), Black line**
SCL	**Pin 5 (SCL), Yellow line**
SDA	**Pin 3 (SDA), Green line**





The resistors are build-in on circuit. (you can skip them)

After connecting GY801

- Install I2C tool to check the state:
  - **sudo apt-get install i2c-tools**
  - Command 1
    - **sudo ls -al /dev/*i2c***
  - Command 2
    - **sudo i2cdetect -y 1**

```
pi@raspberrypi:~$ sudo ls -al /dev/*i2c*
crw-rw---- 1 root i2c 89, 1 Mar 7 03:39 /dev/i2c-1
```

Show the I2C address

```
pi@raspberrypi:~$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

After connecting GY801

- If no I2C device
  - Command 1
    - sudo ls -al /dev/*i2c*

```
pi@raspberrypi:~$ ls -al /dev/*i2c*
ls: cannot access /dev/*i2c*: No such file or directory
```

  - Command 2
    - sudo i2cdetect -y 1

```
pi@raspberrypi:~/gy801$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf
http://www.analog.com/media/cn/technical-documentation/data-sheets/ADXL345_cn.pdf

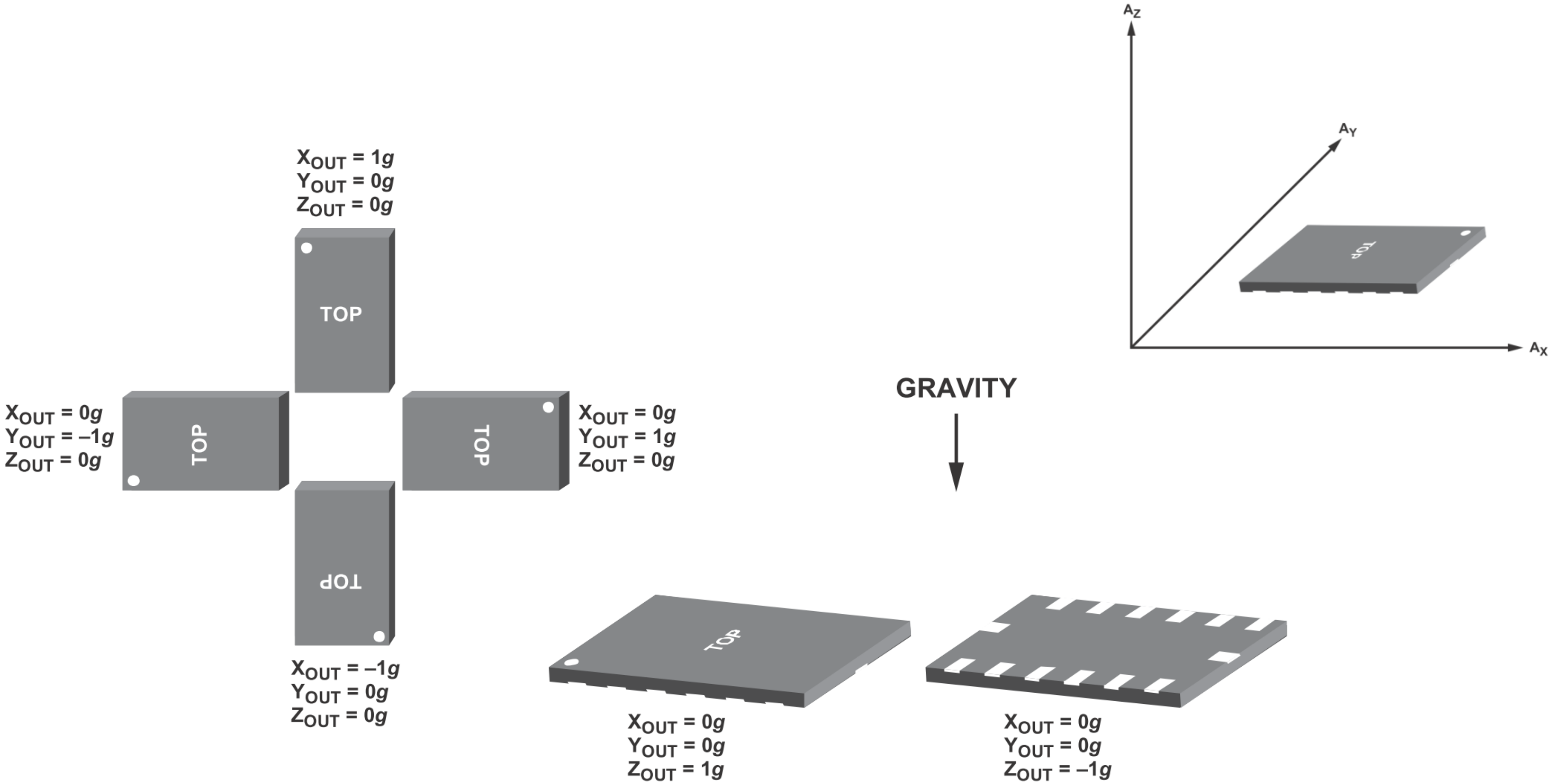
1. Accelerometer (ADXL345)

- ADXL345
  - Chipset：ADXL345
  - Power：3 ~ 5V
  - Protocol：I2C/SPI
  - Range：± 2g、± 4g、± 8g、± 16g

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
OUTPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*
	±4 *g*, 10-bit resolution	115	128	141	LSB/*g*
	±8 *g*, 10-bit resolution	57	64	71	LSB/*g*
	±16 *g*, 10-bit resolution	29	32	35	LSB/*g*
Sensitivity Deviation from Ideal	All *g*-ranges		±1.0		%
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	3.5	3.9	4.3	mg/LSB
	±2 *g*, 10-bit resolution	3.5	3.9	4.3	mg/LSB
	±4 *g*, 10-bit resolution	7.1	7.8	8.7	mg/LSB
	±8 *g*, 10-bit resolution	14.1	15.6	17.5	mg/LSB
	±16 *g*, 10-bit resolution	28.6	31.2	34.5	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C

1. Accelerometer (ADXL345)

- Output Response and Orientation



1. Accelerometer code

- Sample code results:

```
pi@raspberrypi:~/gy801$ python 1acc.py
ACC:
x = 0.498 m/s2
y = -0.306 m/s2
z = -10.343 m/s2
x = 0.051G
y = -0.031G
z = -1.055G
x = 0.000
y = -8.000
z = -270.000
```

1. Accelerometer code: define address

```
bus = smbus.SMBus(1); # 0 for R-Pi Rev. 1, 1 for Rev. 2

EARTH_GRAVITY_MS2     = 9.80665 # m/s2

# the following address is defined by datasheet
ADXL345_ADDRESS          =     0x53  # I2C address

ADXL345_BW_RATE          =     0x2C # Data rate and power mode control
ADXL345_POWER_CTL        =     0x2D # Power-saving features control
ADXL345_DATA_FORMAT      =     0x31 # Data format control
ADXL345_DATAX0           =     0x32
ADXL345_DATAX1           =     0x33
ADXL345_DATAY0           =     0x34
ADXL345_DATAY1           =     0x35
ADXL345_DATAZ0           =     0x36
ADXL345_DATAZ1           =     0x37
# ---------------------------------

# set value
ADXL345_SCALE_MULTIPLIER= 0.00390625     # G/LSP. 1/256 = 0.00390625
ADXL345_BW_RATE_100HZ    = 0x0A          # 0A = 0000 1111
ADXL345_MEASURE          = 0x08          # 08 = 0000 1000
```

When accessing I2C sensor, we have to write code based on datasheet.

1. Accelerometer spec.

REGISTER MAP

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1

Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)
These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

1. Accelerometer code: read/write byte data

```python
def write_byte(self,adr, value):
    bus.write_byte_data(self.ADDRESS, adr, value)

def read_byte(self,adr):
    # read_byte_data: read one byte from "self.ADDRESS", offset "adr"
    return bus.read_byte_data(self.ADDRESS, adr)

def read_word(self,adr,rf=1):
    # rf=1 Little Endian Format, rf=0 Big Endian Format
    if (rf == 1):
        # read two byte data. ex: addr 50 and 51
        low = self.read_byte(adr)
        high = self.read_byte(adr+1)
    else:
        high = self.read_byte(adr)
        low = self.read_byte(adr+1)

    # combine "high" and "low" byte together.
    # shift "high" to left by 8 bits, then put "low"
    # like this: HHHH HHHH  LLLL LLLL
    val = (high << 8) + low

    return val

def read_word_2c(self,adr,rf=1):
    # adr = register address. ex: set "0x32", then "adr" is equal to 50
    val = self.read_word(adr,rf)

    if(val & (1 << 16 - 1)):
        return val - (1<<16)
    else:
        return val
```

1. Accelerometer code: write value to specific address

```
# Register 0x2C: BW_RATE
self.write_byte(ADXL345_BW_RATE, ADXL345_BW_RATE_100HZ)
# write value= 0x0A = 00001111
# D3-D0: The default value is 0x0A,
# which translates to a 100 Hz output data rate.


# Register 0x2D: POWER_CTL
self.write_byte(ADXL345_POWER_CTL, ADXL345_MEASURE)
# write value: 0x08 = 00001000
# D3=1: set 1 for measurement mode.


# Register 0x31: DATA_FORMAT
self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
# write value=00001000
# D3 = 1: the device is in full resolution mode,
# where the output resolution increases with the g range
# set by the range bits to maintain a 4 mg/LSB scale factor.
# D1 D0 = range. 00 = +-2g
```

1. Accelerometer code: read X/Y/Z raw data

LSB

```
# RAW readings in LPS
# Register 0x32 to Register 0x37:
# DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)
def getRawX(self) :
    self.Xraw = self.read_word_2c(ADXL345_DATAX0)
    return self.Xraw

def getRawY(self) :
    self.Yraw = self.read_word_2c(ADXL345_DATAY0)
    return self.Yraw

def getRawZ(self) :
    self.Zraw = self.read_word_2c(ADXL345_DATAZ0)
    return -1*self.Zraw
```

1. Accelerometer code: convert unit

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*

SCALE_MULTIPLIER = 1/SENSITIVITY

```
ADXL345_SCALE_MULTIPLIER= 0.00390625    # G/LSP.
self.Xcalibr = ADXL345_SCALE_MULTIPLIER

# Register 0x31: DATA_FORMAT with write value=00001000
self.write_byte(ADXL345_DATA_FORMAT, self.df_value)
# D1 D0 = range. 00 = +-2g

# RAW readings in LPS
# Register 0x32 to Register 0x37:
# DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)
def getRawX(self) :
    self.Xraw = self.read_word_2c(ADXL345_DATAX0)
    return self.Xraw

# G related readings in g
def getXg(self,plf = 1.0) :
    self.Xg = (self.getRawX() * self.Xcalibr - self.Xoffset) * plf \
    + (1.0 - plf) * self.Xg
    return self.Xg

# Absolute reading in m/s2
def getX(self,plf = 1.0) :
    self.X = self.getXg(plf) * EARTH_GRAVITY_MS2
    return self.X
```

https://theccontinuum.com/2012/09/24/arduino-imu-pitch-roll-from-accelerometer/

Discussion 1

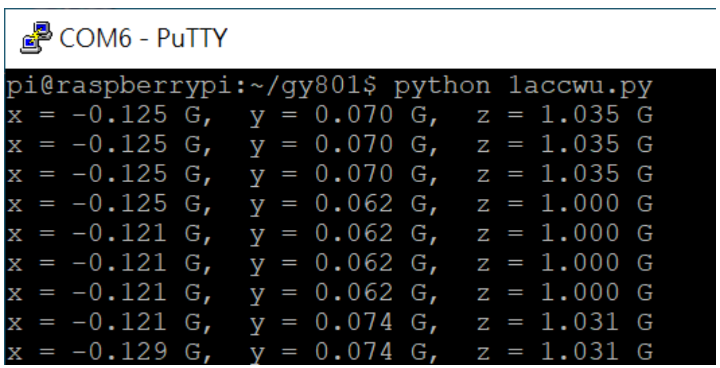
- 1. Based on the requirements, set correct value in the sample code
  - Output data rate: 100Hz
  - Range: +-2g
  - Convert LSB to G (by using SCALE_MULTIPLIER)

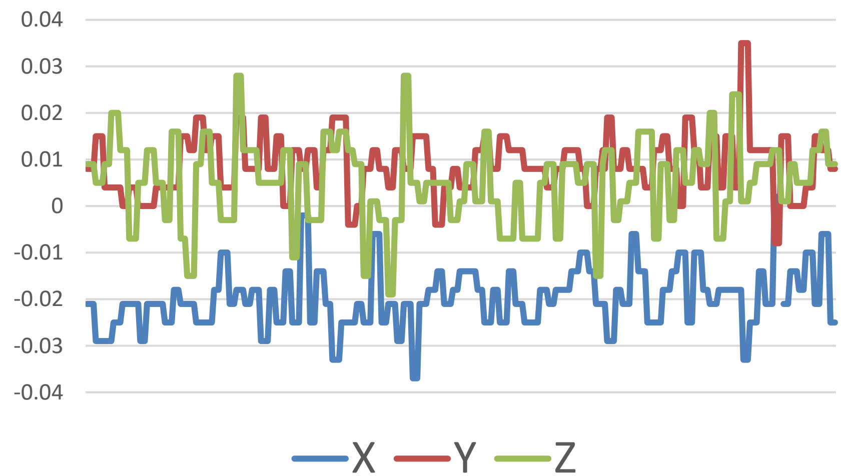
```
# set value
ADXL345_SCALE_MULTIPLIER= ???
ADXL345_BW_RATE_100HZ    = 0x???
ADXL345_MEASURE          = 0x???
```

- 2. Continuously measurement(infinite loop)
- 3. Calibrate your sensor (see next page)

Discussion 1 (cont.)

- Theoretically: x,y,z report (0,0,1) when static
- Actually:



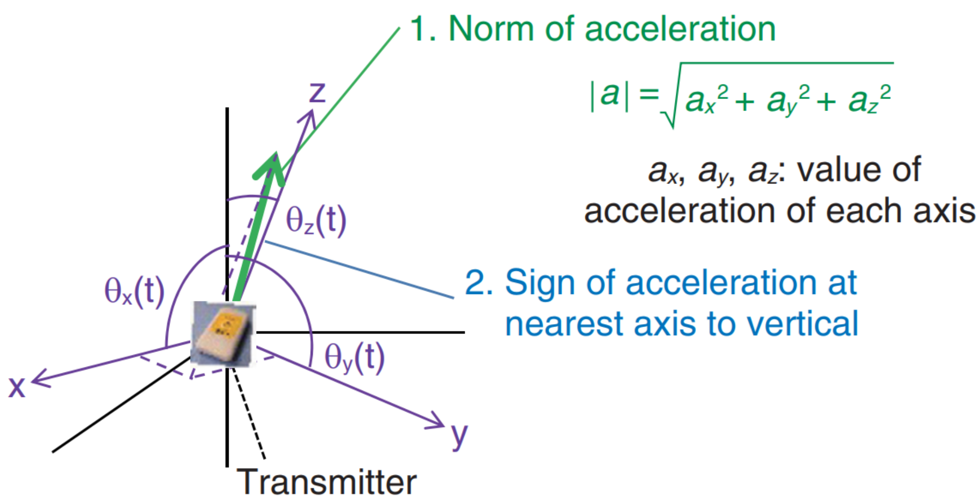


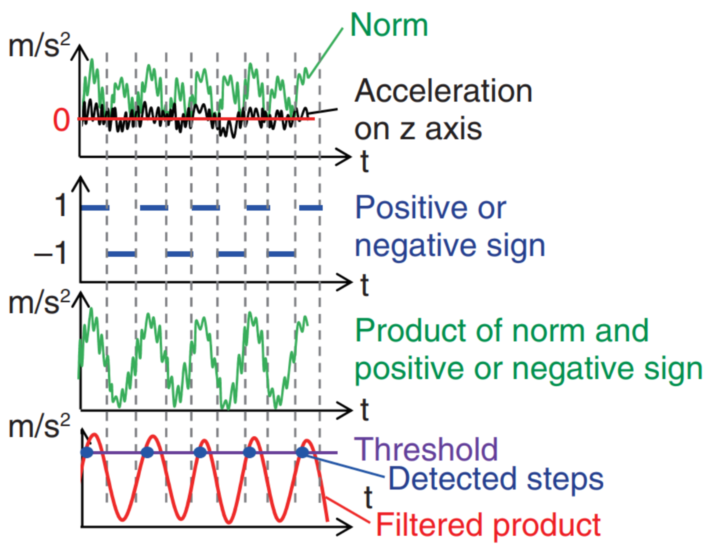
- Measure your static reading, then feedback to code

```
self.Xoffset = 0.0
self.Yoffset = 0.0
self.Zoffset = 0.0
```

Quiz 1

- Calculate the norm of acceleration |a|
  - Formula:
    |a| = sqrt(x^2 + y^2 + z^2)





Application: step detection

Gait Analysis Using a Wearable T-shirt Type Sensor
https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201604ra1.html

Quiz 1

- Use python to calculate pow and sqrt

```
import numpy
x = 4

y = pow(x,2)
# y=16

z = numpy.sqrt(x)
# z=2

print y
print z
```

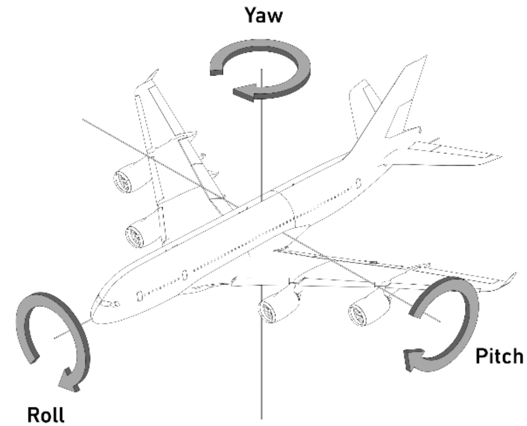
(COM4) [80x24]

連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

```
pi@raspberrypi:~$ python norm_a.py
16
2.0
pi@raspberrypi:~$
```

Quiz2: Roll, Pitch, Tilt

- Calculate posture:
  - Use normalized accelerometer reading G_p
  - Tilt, Roll, Pitch



(from AN3461.pdf)

Roll: $\tan \phi_{xyz} = \frac{G_{py}}{G_{pz}}$

Pitch: $\tan \theta_{xyz} = \frac{-G_{px}}{\sqrt{G_{py}^2 + G_{pz}^2}}$

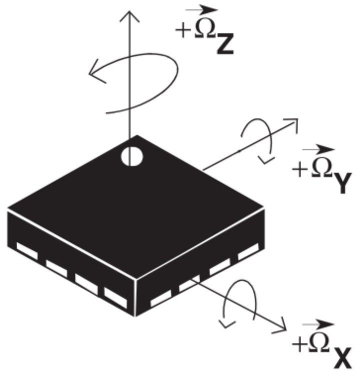
Figure 3. Definition of Coordinate System and Rotation Axes

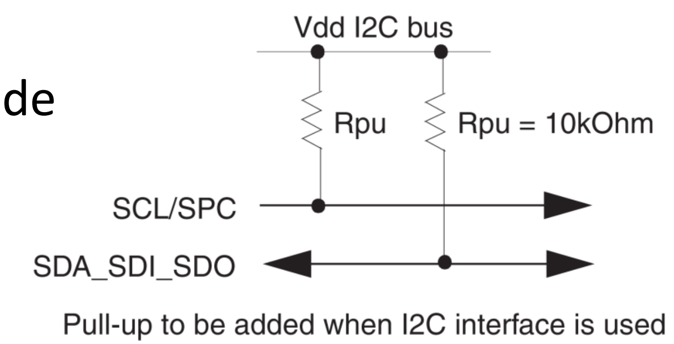
Tilt: $\cos \rho = \frac{G_{pz}}{\sqrt{G_{px}^2 + G_{py}^2 + G_{pz}^2}}$

Figure 7. Calculation of the Tilt Angle ρ from Vertical

2. Gyroscope (L3G4200D)

- Features
  - 3-Axis angular rate sensor
  - Supports I2C and SPI communications
  - Three selectable scales: 250/500/2000 degrees/sec (dps)
  - High shock survivability
  - Embedded temperature sensor -40 to +185 °F (-40 to + 85 °C)
  - Embedded power-down and sleep mode
  - 16 bit-rate value data output
  - 8-bit temperature data output

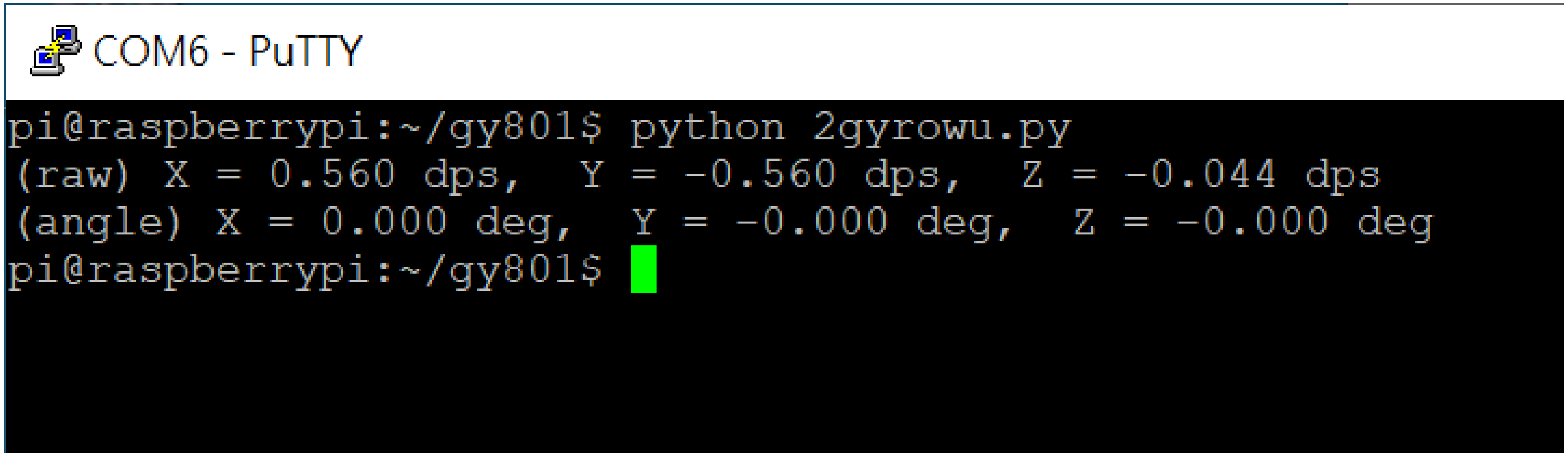




http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Gyros/3-Axis/17116.pdf
http://www.playrobot.com/gyro/284-sparkfun-tri-axis-gyro-breakout-l3g4200d.html

2. Gyroscope code

- Sample code results:



2. Gyroscope code: define address, convert unit

```
L3G4200D_ADDRESS        =     0x69
L3G4200D_CTRL_REG1      =     0x20 # CTRL_REG1 (20h)
L3G4200D_CTRL_REG4      =     0x23 # CTRL_REG4 (23h)
L3G4200D_OUT_X_L        =     0x28 # X-axis angular rate data.
L3G4200D_OUT_X_H        =     0x29 # The value is expressed as two's complement.
L3G4200D_OUT_Y_L        =     0x2A
L3G4200D_OUT_Y_H        =     0x2B
L3G4200D_OUT_Z_L        =     0x2C
L3G4200D_OUT_Z_H        =     0x2D
```

```
        # FS=250dps, Sensitivity = 8.75 mdps/digit
        self.gain_std = 0.00875

        # 0x20: CTRL_REG1, write 0x0F = 0000 1111
        self.write_byte(L3G4200D_CTRL_REG1, 0x0F)
        # DR1-DR0 = 00; BW1-BW0 = 00 -> 100Hz, Cut-off=12.5
        # PD, Zen, Yen, Xen = 1111

        # 0x23: CTRL_REG4, write 0x80 = 0000 1000
        self.write_byte(L3G4200D_CTRL_REG4, 0x80)
        # BDU=continous update; BLE=Data LSB
        # FS1-FS0=00 -> 250 dps

        self.setCalibration()

    def setCalibration(self) :
        gyr_r = self.read_byte(L3G4200D_CTRL_REG4)

        self.gain = 2 ** ( gyr_r & 48 >> 4) * self.gain_std
```

2. Gyroscope code: write value to specific address

```
        # 0x20: CTRL_REG1, write 0x0F = 0000 1111
        self.write_byte(L3G4200D_CTRL_REG1, 0x0F)
        # DR1-DR0 = 00; BW1-BW0 = 00 -> 100Hz, Cut-off=12.5
        # PD, Zen, Yen, Xen = 1111
```

CTRL_REG1 (20h)

Table 20. CTRL_REG1 register

DR1	DR0	BW1	BW0	PD	Zen	Yen	Xen

Table 21. CTRL_REG1 description

DR1-DR0	Output Data Rate selection. Refer to *Table 22*
BW1-BW0	Bandwidth selection. Refer to *Table 22*
PD	Power down mode enable. Default value: 0 (0: power down mode, 1: normal mode or sleep mode)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

2. Gyroscope code: write value to specific address

```
        # 0x23: CTRL_REG4, write 0x80 = 0000 1000
        self.write_byte(L3G4200D_CTRL_REG4, 0x80)
        # BDU=continous update; BLE=Data LSB
        # FS1-FS0=00 -> 250 dps
```

CTRL_REG4 (23h)

Table 30. CTRL_REG4 register

BDU	BLE	FS1	FS0	-	ST1	ST0	SIM

Table 31. CTRL_REG4 description

BDU	Block Data Update. Default value: 0 (0: continous update; 1: output registers not updated until MSB and LSB reading)
BLE	Big/Little Endian Data Selection. Default value 0. (0: Data LSB @ lower address; 1: Data MSB @ lower address)
FS1-FS0	Full Scale selection. Default value: 00 (00: 250 dps; 01: 500 dps; 10: 2000 dps; 11: 2000 dps)
ST1-ST0	Self Test Enable. Default value: 00 (00: Self Test Disabled; Other: See *Table*)
SIM	SPI Serial Interface Mode selection. Default value: 0 (0: 4-wire interface; 1: 3-wire interface).

2. Gyroscope code: convert raw data to degree

```python
# read raw data
def getRawX(self):
    self.Xraw = self.read_word_2c(L3G4200D_OUT_X_L)
    return self.Xraw

# similar to filter. combine current value with previous one.
# plf = 1 means it only uses "current reading"
def getX(self,plf = 1.0):
    self.X = ( self.getRawX() * self.gain ) * plf + (1.0 - plf) * self.X
    return self.X

# convert dps to angle. LP = loop period.
# Degree per second * second = degree
def getXangle(self,plf = 1.0) :
    if self.t0x is None : self.t0x = time.time()
    t1x = time.time()
    LP = t1x - self.t0x
    self.t0x = t1x
    self.Xangle = self.getX(plf) * LP
    return self.Xangle
```

Discussion 2

- 1. Based on the requirements, set correct value in the sample code
  - Data rate: 100Hz, cut-off = 12.5
  - Full Scale selection = 250 dps
  - Set Sensitivity for 250 dps (by using SCALE_MULTIPLIER)

```
# set value
self.gain_std = ??     # dps/digit

self.write_byte(L3G4200D_CTRL_REG1, 0x??)
self.write_byte(L3G4200D_CTRL_REG4, 0x??)
```

- 2. Continuously measurement (infinite loop)

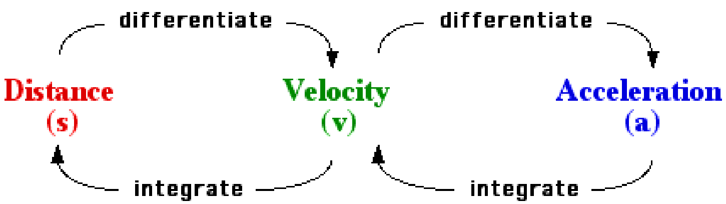
Quiz 2

- We can obtain ax/ay/az from accelerometer. How to calculate the distance?

$$\mathbf{s}(t) = \mathbf{s}_0 + \mathbf{v}_0 t + \tfrac{1}{2}\mathbf{a}t^2 = \mathbf{s}_0 + \frac{\mathbf{v}_0 + \mathbf{v}(t)}{2}t$$

$$\mathbf{v}(t) = \mathbf{v}_0 + \mathbf{a}t$$

$$v^2(t) = {v_0}^2 + 2\mathbf{a}\cdot[\mathbf{s}(t) - \mathbf{s}_0]$$



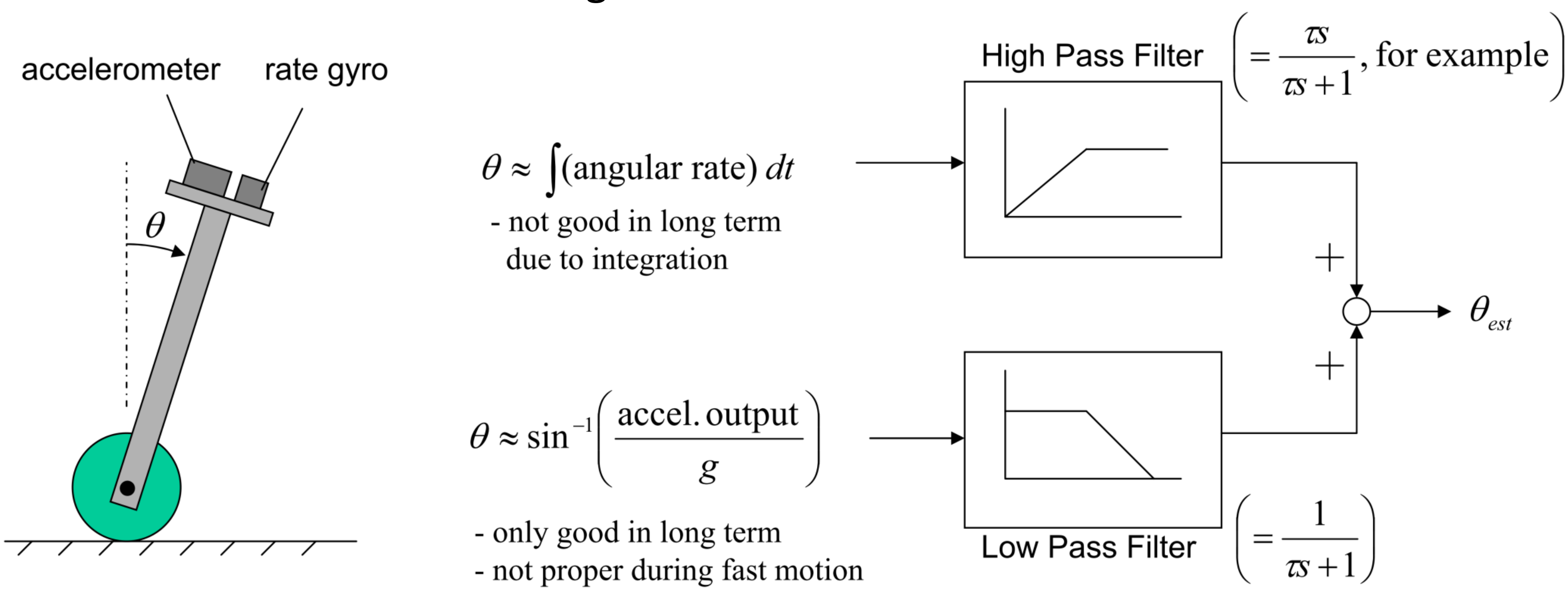

```
# convert dps to angle. LP = loop period.
# Degree per second * second = degree
def getXangle(self,plf = 1.0) :
    if self.t0x is None : self.t0x = time.time()
    t1x = time.time()
    LP = t1x - self.t0x
    self.t0x = t1x
    self.Xangle = self.getX(plf) * LP
    return self.Xangle
```

Hint: In Gyro sample code, it convert dps to angle.

https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-333-aircraft-stability-and-control-fall-2004/lecture-notes/lecture_15.pdf

Sensor fusion

- Accelerometer is good in long term.
- Gyroscope is good in short term.
  - Use Complementary filter to integrate them
  - Enhanced rotation angle

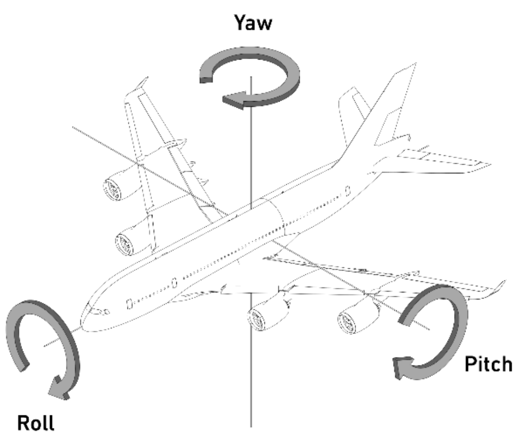


Quiz3: Sensor fusion

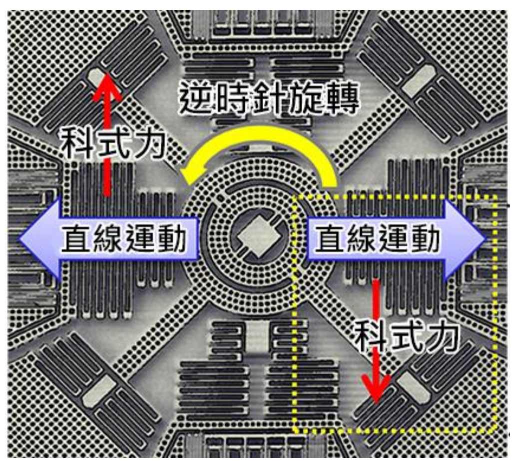
- Calculate angle based on accelerometer and gyroscope
  - Complementary filter: Enhanced rotation angle

Angle = 0.98 * (angle + gyroData * dt) + 0.02 * accData

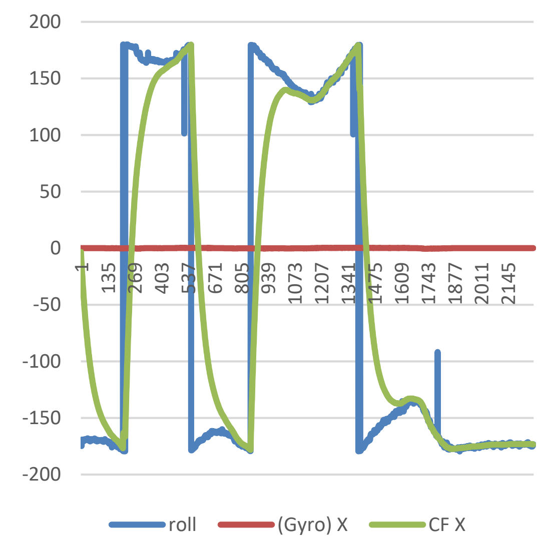
pitch = (pitch + gyroX * dt) * 0.98 + pitchAcc * 0.02;
roll = (roll + gyroY * dt) * 0.98 + rollACC * 0.02



We can get **Roll, Pitch** from accelerometer.



When rotating the sensor, we can get **Angular velocity.**



Summary

- Practice Lab (accelerometer, gyroscope)
- Write down the answer for discussion
  - Discussion1&2:
    - Based on the requirements, set correct value in the sample code
    - Deadline: Before ??

- Write code for Quiz 1 - 3, then demonstrate it to TAs
  - Quiz1: Calculate the norm of acceleration |a|
  - Quiz2: Roll, Pitch, Tilt
  - Quiz3: Sensor fusion
    - Deadline: Before ??
    - Late Demo: Before ??