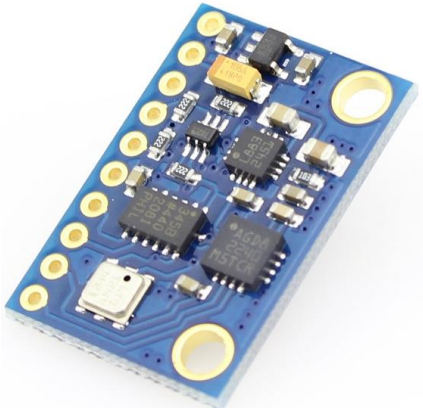
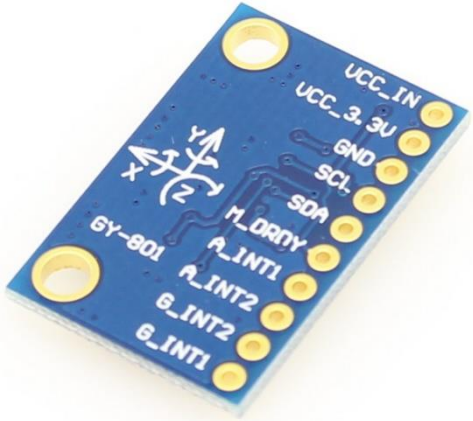
Outline

- 嵌入式應用: 人體活動偵測
  - 加速度、陀螺儀、電子羅盤、氣壓計…等
  - 感測數值分析



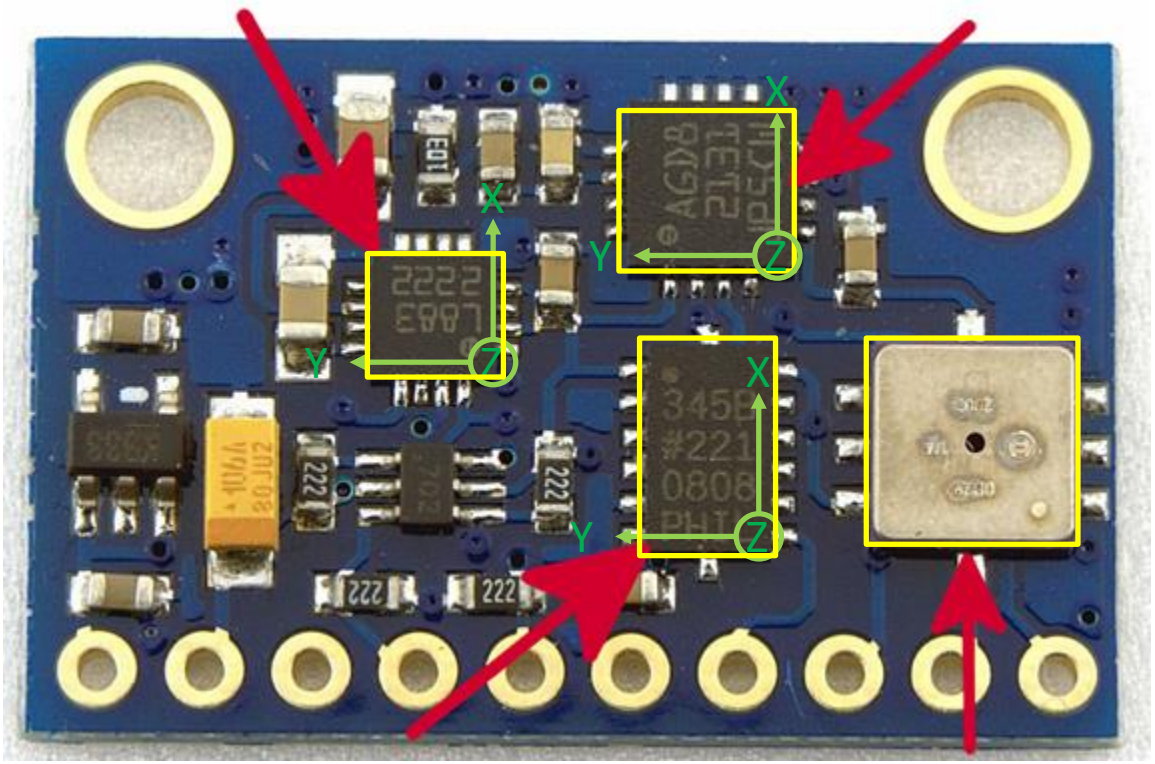
- GY801 (I2C sensor)
  - 3-axis Accelerometer, Gyroscope, magnetometer and pressure
    1. ADXL345：Accelerometer
    2. L3G4200：Gyroscope
    3. HMC5883：Magnetometer
    4. BMP085：Pressure

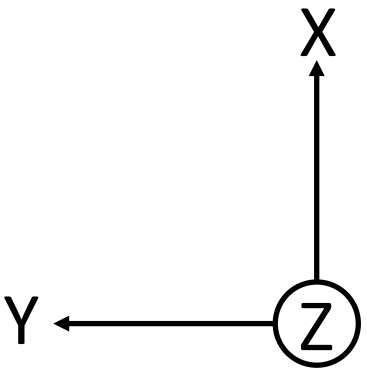


GY-801 (10 DoF sensor)

Compass (HMC5883L)

Gyro (L3G4200D)



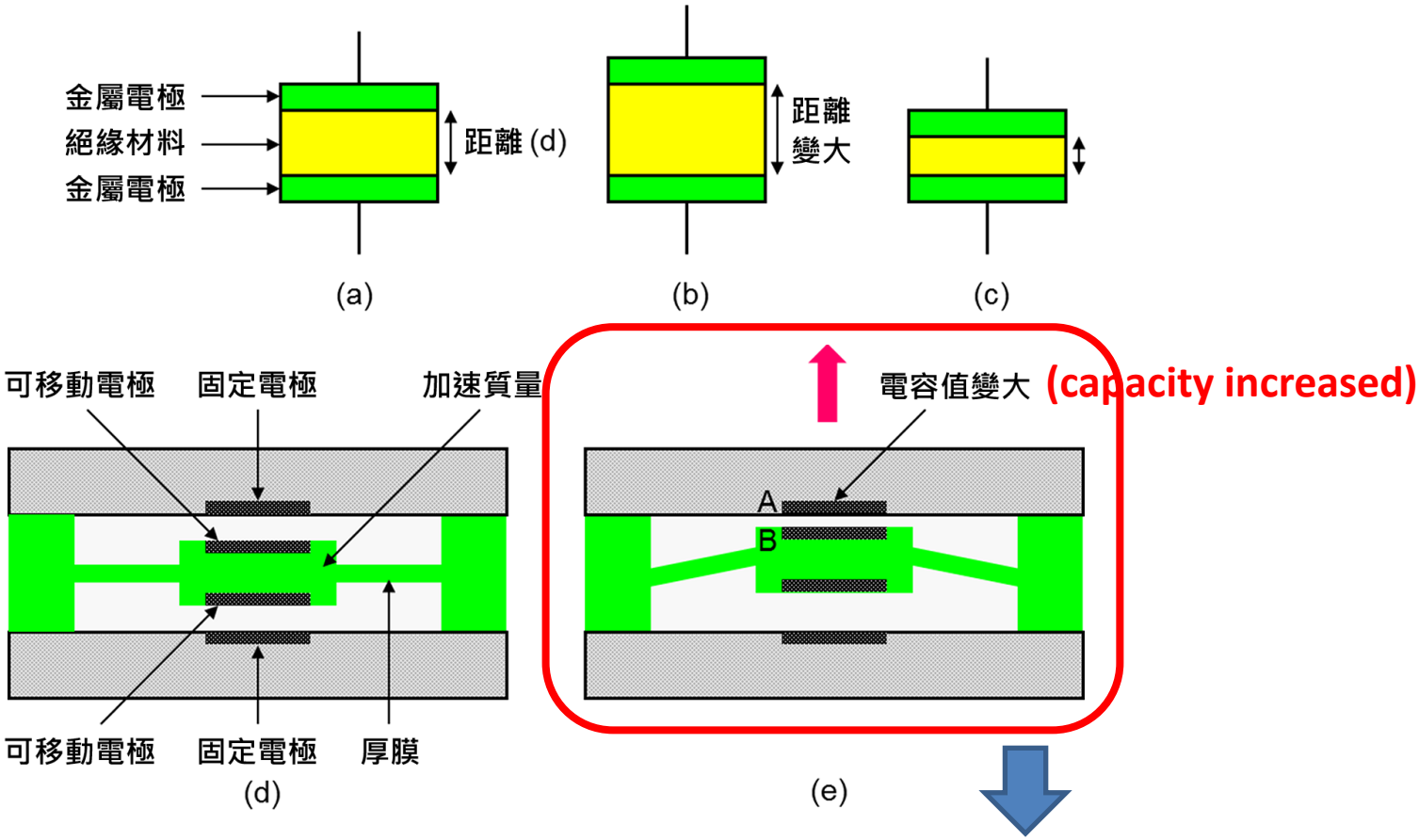


G-sensor (ADXL345)

Pressure (BMP085)

電容微加速度器的構造與原理
http://scimonth.blogspot.tw/2014/07/blog-post_94.html

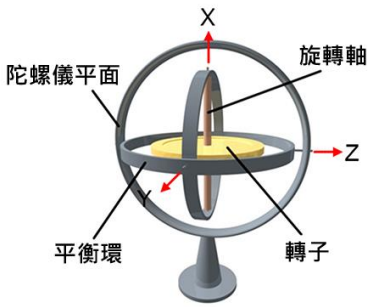
Sensor - Accelerometer



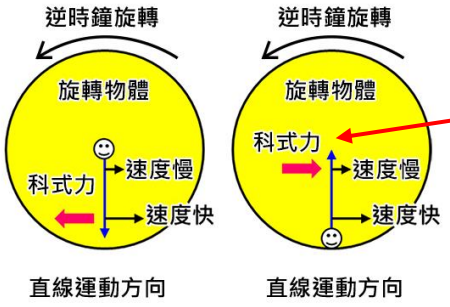
When you move the sensor, the distance (**capacity**) is changed.
We **use capacity to calculate acceleration**.

微陀螺儀的構造與原理
http://scimonth.blogspot.tw/2014/07/blog-post_94.html

Sensor - Gyro

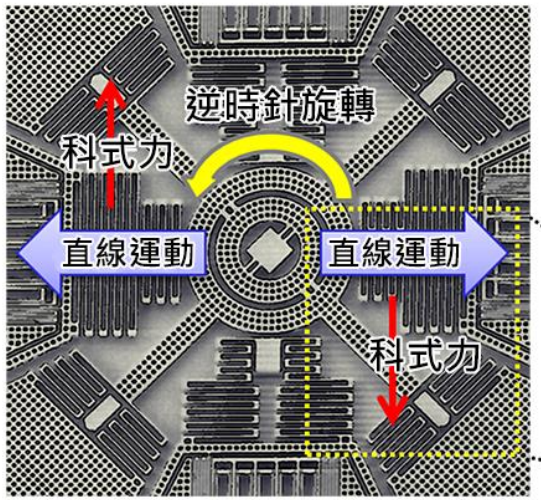


(a)

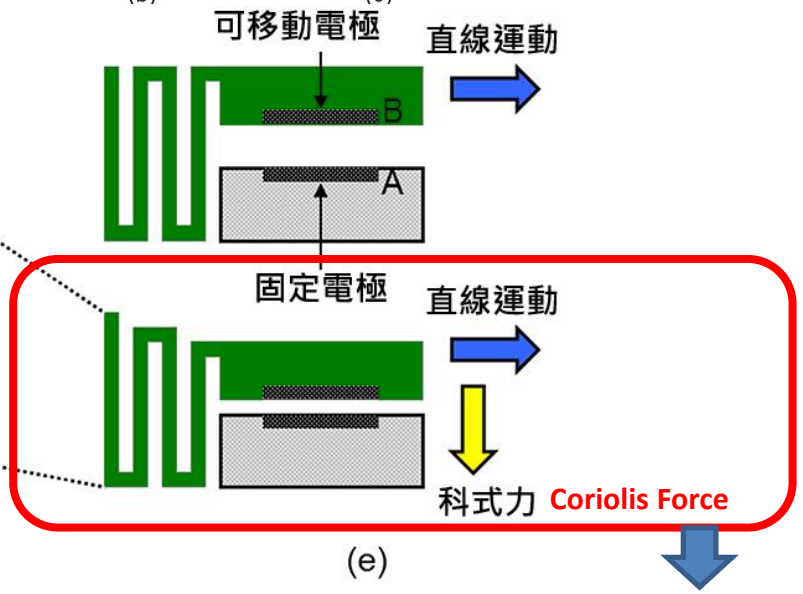


(b) (c)

Coriolis Force (科氏力)



(d)



(e)

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

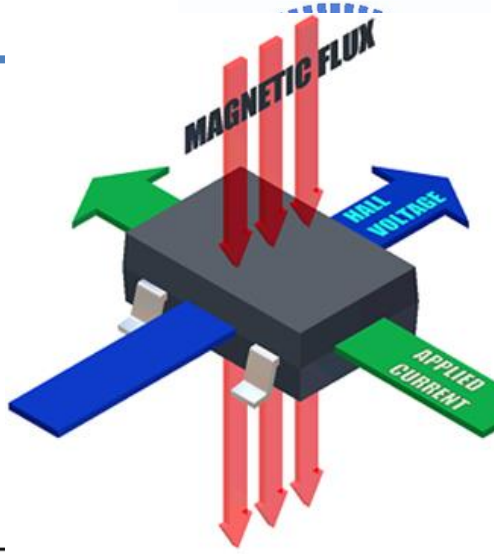
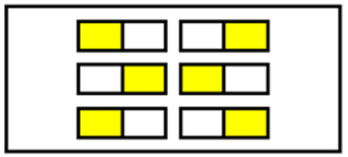
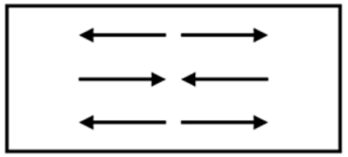
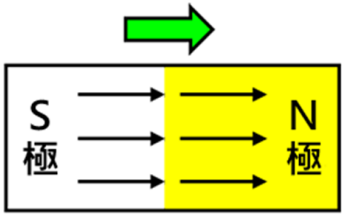


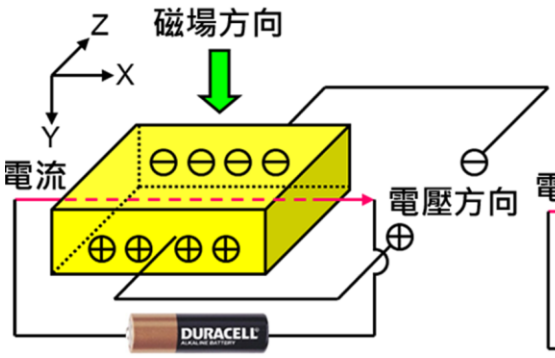
Fig: https://www.ecnmag.com/article/2009/03/sensor-zone-april-2009


(a)

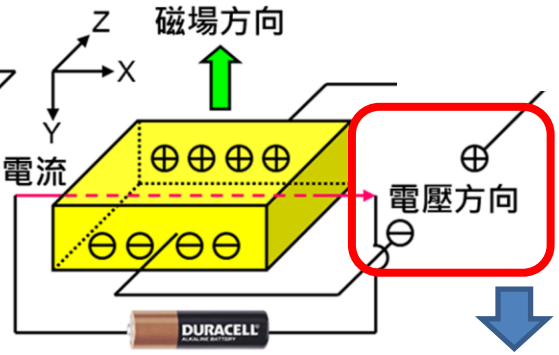

(b)



(c)

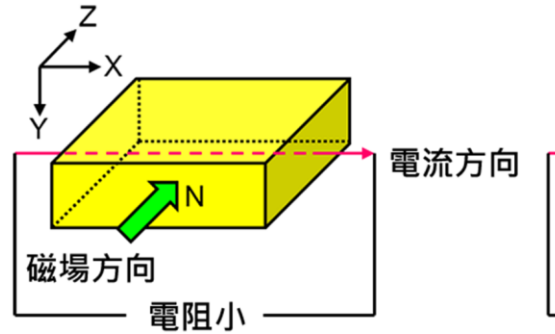


(d)

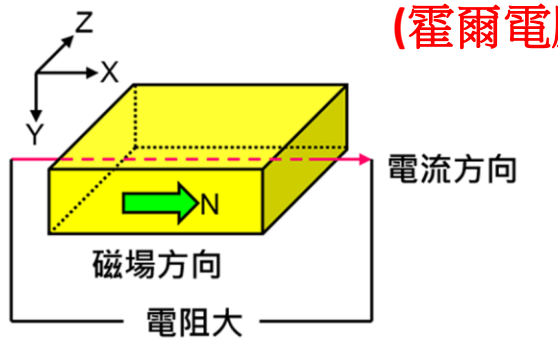


(e)

Hall voltage (霍爾電壓)



(f)

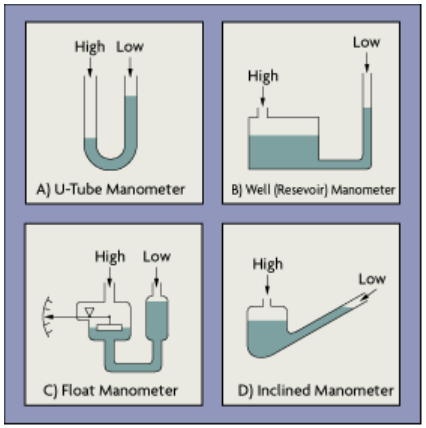


(g)

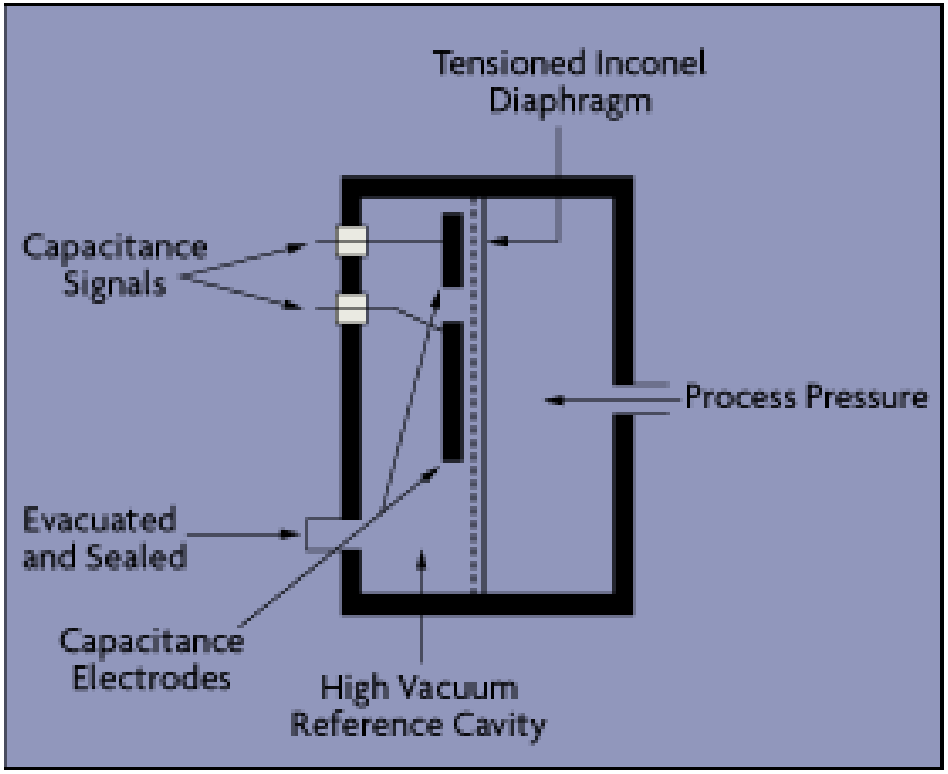
Measure the absolute **magnetic intensity** based on **Hall voltage (霍爾電壓)**

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

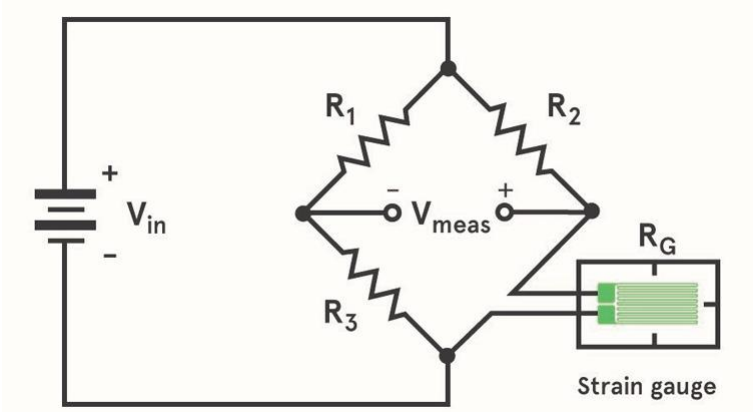
Sensor - Pressure



Manometer



Capacitance Vacuum Manometer



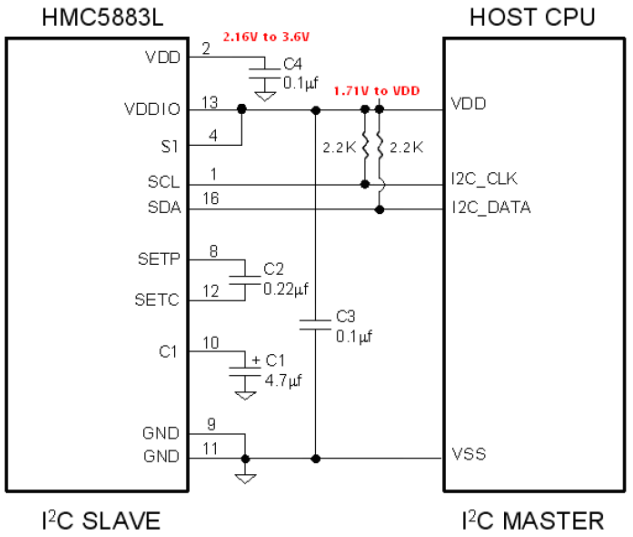
Use Wheatstone bridge to measure the resistance.

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

https://sites.google.com/site/playrobotquickstart/home/hmc583l-dian-zi-luo-pan
https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf
http://web.sensor-ic.com:8000/ZLXIAZAI/20130122/HMC5883L中文规格书-Feb%202011.pdf

3. Magnetometer (HMC5883L)

- HMC5883L
  - 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package
  - 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in ±8 Gauss Fields
  - Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 μ A)
  - I2C Digital Interface
  - Wide Magnetic Field Range (+/-8 Oe)

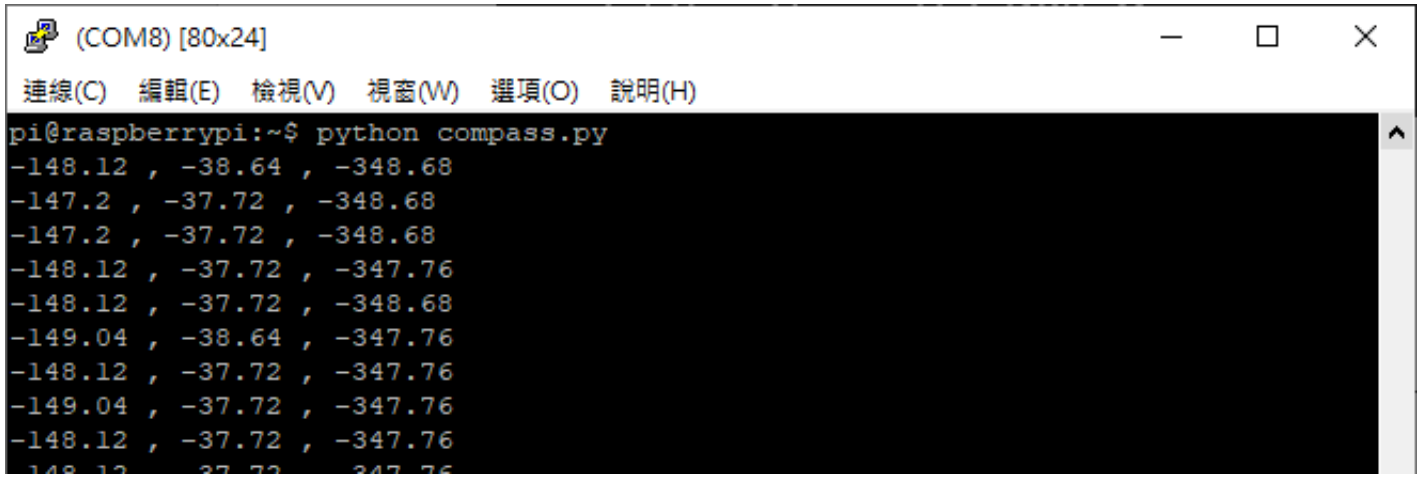


3. Magnetometer code

- Sample code:

```
wget  https://goo.gl/c6iD08 -O compass.py
python compass.py
```

- Execution



3. Magnetometer code

```python
import sys, os, math, time, thread, smbus, random, requests

bus = smbus.SMBus(1)
addrHMC = 0x1e

def read_word(address, adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr + 1)
    val = (high << 8) + low
    return val

def read_word_2c(address, adr):
    val = read_word(address, adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def main():

    bus.write_byte_data(addrHMC, 0, 0b01110000)  # Set to 8 samples @ 15Hz
    bus.write_byte_data(addrHMC, 1, 0b00100000)  # 1.3 gain LSb / Gauss 1090 (default)
    bus.write_byte_data(addrHMC, 2, 0b00000000)  # Continuous sampling

    while True:
        x = read_word_2c(addrHMC, 3)
        y = read_word_2c(addrHMC, 7)
        z = read_word_2c(addrHMC, 5)

        print x,",",y,",",z
        time.sleep(0.1)

if __name__ == "__main__":
    main()
```

Read register value

2's complement

Read mx/my/mz

Display sensing value

Download sample code：**wget https://goo.gl/c6iD08 -O compass.py**

3. HMC5883L Datasheet

- Set register address based on datasheet

Register Access

This section describes the process of reading from and writing to this device. The devices uses an address pointer to indicate which register location is to be read from or written to. These pointer locations are sent from the master to this slave device and succeed the 7-bit address (0x1E) plus 1 bit read/write identifier, i.e. 0x3D for read and 0x3C for write.

```
bus = smbus.SMBus(1)
addrHMC = 0x1e

def init_imu():
  # HMC setting
  bus.write_byte_data(addrHMC, 0, 0b01110000) # Set to 8 samples @ 15Hz
  bus.write_byte_data(addrHMC, 1, 0b00100000) # 1.3 gain LSb / Gauss 1090 (default)
  bus.write_byte_data(addrHMC, 2, 0b00000000) # Continuous sampling
```

3. HMC5883L Datasheet

- Set register address

```
bus.write_byte_data(addrHMC, 0, 0b01110000)
bus.write_byte_data(addrHMC, 1, 0b00100000)
bus.write_byte_data(addrHMC, 2, 0b00000000)
```

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

```
read_word_2c(addrHMC, 3)
read_word_2c(addrHMC, 5)
read_word_2c(addrHMC, 7)
```

Read mx/my/mz

Table2: Register List

3. HMC5883L Datasheet

- Set operation mode (Configuration Register A)

```
bus.write_byte_data(addrHMC, 0, 0b01110000) # Set to 8 samples @ 15Hz
```

CRA7	CRA6	CRA5	CRA4	CRA3	CRA2	CRA1	CRA0
(0)	MA1(0)	MA0(0)	DO2 (1)	DO1 (0)	DO0 (0)	MS1 (0)	MS0 (0)

Table 3: Configuration Register A

Location	Name	Description
CRA7	CRA7	Bit CRA7 is reserved for future function. Set to 0 when configuring CRA.
CRA6 to CRA5	MA1 to MA0	Select number of samples averaged (1 to 8) per measurement output. 00 = 1(Default); 01 = 2; 10 = 4; 11 = 8

8 samples

DO2	DO1	DO0	Typical Data Output Rate (Hz)
0	0	0	0.75
0	0	1	1.5
0	1	0	3
0	1	1	7.5
1	0	0	15 (Default)
1	0	1	30
1	1	0	75
1	1	1	Reserved

15 Hz

Table 5: Data Output Rates

3. HMC5883L Datasheet

- Set operation mode (Configuration Register B)

```
bus.write_byte_data(addrHMC, 1, 0b00100000) # 1.3 gain LSb / Gauss 1090 (default)
```

CRB7	CRB6	CRB5	CRB4	CRB3	CRB2	CRB1	CRB0
GN2 (0)	GN1 (0)	GN0 (1)	(0)	(0)	(0)	(0)	(0)

Table 7: Configuration B Register

GN2	GN1	GN0	Recommended Sensor Field Range	Gain (LSb/ Gauss)	Digital Resolution (mG/LSb)	Output Range
0	0	0	± 0.88 Ga	1370	0.73	0xF800–0x07FF (-2048–2047)
0	0	1	± 1.3 Ga	1090 (default)	0.92	0xF800–0x07FF (-2048–2047)

+/- 1.3Ga; Gain 1090

3. HMC5883L Datasheet

- Set operation mode (Mode Register)

```
bus.write_byte_data(addrHMC, 2, 0b00000000) # Continuous sampling
```

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
HS(0)	(0)	(0)	(0)	(0)	(0)	MD1 (0)	MD0 (1)

Table 10: Mode Register

Continuous Measurement Mode

MD1	MD0	Operating Mode
0	0	Continuous-Measurement Mode. In continuous-measurement mode, the device continuously performs measurements and places the result in the data register. RDY goes high when new data is placed in all three registers. After a power-on or a write to the mode or configuration register, the first measurement set is available from all three data output registers after a period of $2/f_{DO}$ and subsequent measurements are available at a frequency of f_{DO}, where f_{DO} is the frequency of data output.
0	1	Single-Measurement Mode (Default). When single-measurement mode is selected, device performs a single measurement, sets RDY high and returned to idle mode. Mode register returns to idle mode bit values. The measurement remains in the data output register and RDY remains high until the data output register is read or another measurement is performed.

You can also change operation mode based on your need and datasheet.
The code is already set default for you.

3. HMC5883L Datasheet

- In the code, we also obtain raw data (bit value, LSB). How to convert it to gauss?

$$m_{gauss} = m_{raw} * \mathbf{DigitalResolution}$$

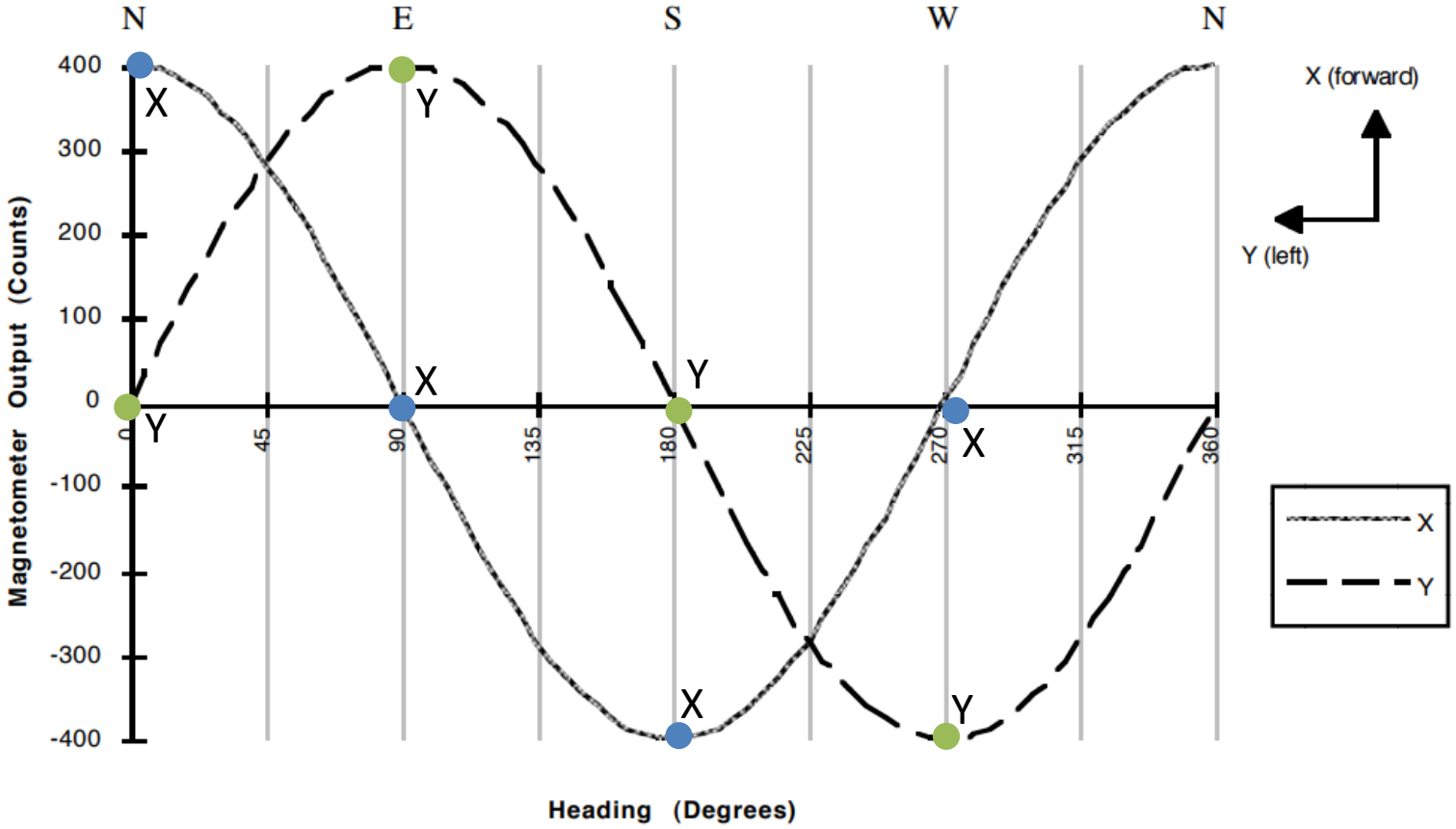
GN2	GN1	GN0	Recommended Sensor Field Range	Gain (LSb/ Gauss)	Digital Resolution (mG/LSb)	Output Range
0	0	0	± 0.88 Ga	1370	0.73	0xF800–0x07FF (-2048–2047)
0	0	1	± 1.3 Ga	1090 (default)	0.92	0xF800–0x07FF (-2048–2047)

  - When it report -4096, the overflow occurs.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

https://aerospace.honeywell.com/en/~/media/aerospace/files/application-note/an203_compass_heading_using_magnetometers.pdf

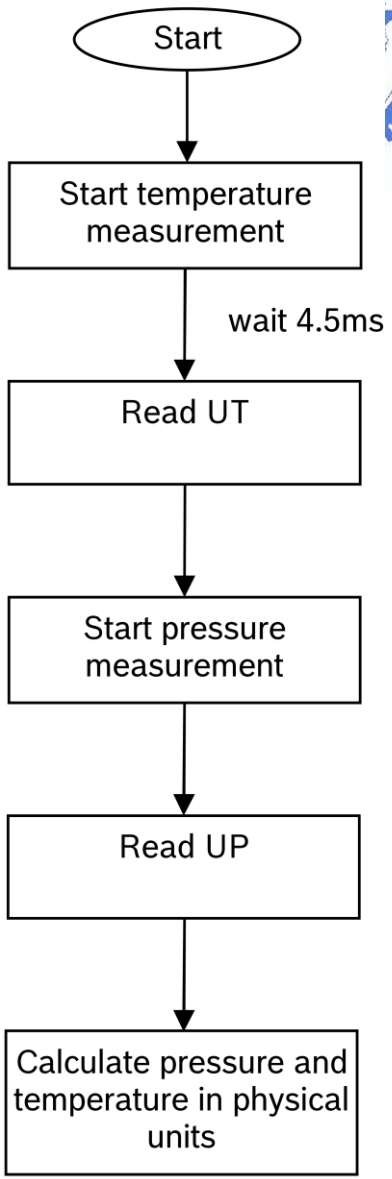
Magnetometer reading



Hx and Hy Magnetometer Readings for Different Compass Headings

4. Altitude (BMP085)

- Pressure sensing range: 300-1100 hPa
  - (9000m to -500m above sea level)
- Up to 0.03hPa / 0.25m resolution
- -40 to +85°C operational range
- +-2°C temperature accuracy
  - Temperature measurement included
- 2-pin i2c interface on chip



https://www.sparkfun.com/datasheets/Components/General/BST-BMP085-DS000-05.pdf
https://www.sparkfun.com/tutorials/253

4. Altitude (BMP085)

- BMP085 sample code:
  - https://github.com/adafruit/Adafruit_Python_BMP

```
sudo apt-get install git build-essential python-dev
git clone https://github.com/adafruit/Adafruit_Python_BMP
cd Adafruit_Python_BMP
sudo python setup.py install

cd examples/
sudo python simplectest.py
```

- Execution

```
pi@raspberrypi:~/gy80/4.bmp/examples$ sudo python simpletest.py
Temp = 25.50 *C
Pressure = 100536.00 Pa
Altitude = 65.90 m
Sealevel Pressure = 100536.00 Pa
```

4. Altitude code (BMP085)

- Source code: examples/simpletest.py

```
#!/usr/bin/python
import Adafruit_BMP.BMP085 as BMP085
sensor = BMP085.BMP085()

print('Temp = {0:0.2f} *C'.format(sensor.read_temperature()))
print('Pressure = {0:0.2f} Pa'.format(sensor.read_pressure()))
print('Altitude = {0:0.2f} m'.format(sensor.read_altitude()))
print('Sealevel Pressure = {0:0.2f} Pa'.format(sensor.read_sealevel_pressure()))
```

Load library

Print sensing value

More information: Adafruit_BMP/BMP085.py

4. BMP085 Datasheet

- Define address in Adafruit_BMP/BMP085.py

```
# BMP085 default address.
BMP085_I2CADDR          = 0x77

# Operating Modes
BMP085_ULTRALOWPOWER    = 0
BMP085_STANDARD         = 1
BMP085_HIGHRES          = 2
BMP085_ULTRAHIGHRES     = 3
```

The operation mode is set to BMP085_STANDARD. (refer to datasheet)

Mode	Parameter *oversampling_setting*	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [µA]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

4. BMP085 Datasheet

- Define Calibration coefficients from datasheet

	BMP085 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

```
# BMP085 Registers
BMP085_CAL_AC1      = 0xAA  # R   Calibration data (16 bits)
BMP085_CAL_AC2      = 0xAC  # R   Calibration data (16 bits)
BMP085_CAL_AC3      = 0xAE  # R   Calibration data (16 bits)
BMP085_CAL_AC4      = 0xB0  # R   Calibration data (16 bits)
BMP085_CAL_AC5      = 0xB2  # R   Calibration data (16 bits)
BMP085_CAL_AC6      = 0xB4  # R   Calibration data (16 bits)
BMP085_CAL_B1       = 0xB6  # R   Calibration data (16 bits)
BMP085_CAL_B2       = 0xB8  # R   Calibration data (16 bits)
BMP085_CAL_MB       = 0xBA  # R   Calibration data (16 bits)
BMP085_CAL_MC       = 0xBC  # R   Calibration data (16 bits)
BMP085_CAL_MD       = 0xBE  # R   Calibration data (16 bits)
```

4. BMP085 Datasheet

- How to read temperature and pressure referred by datasheet

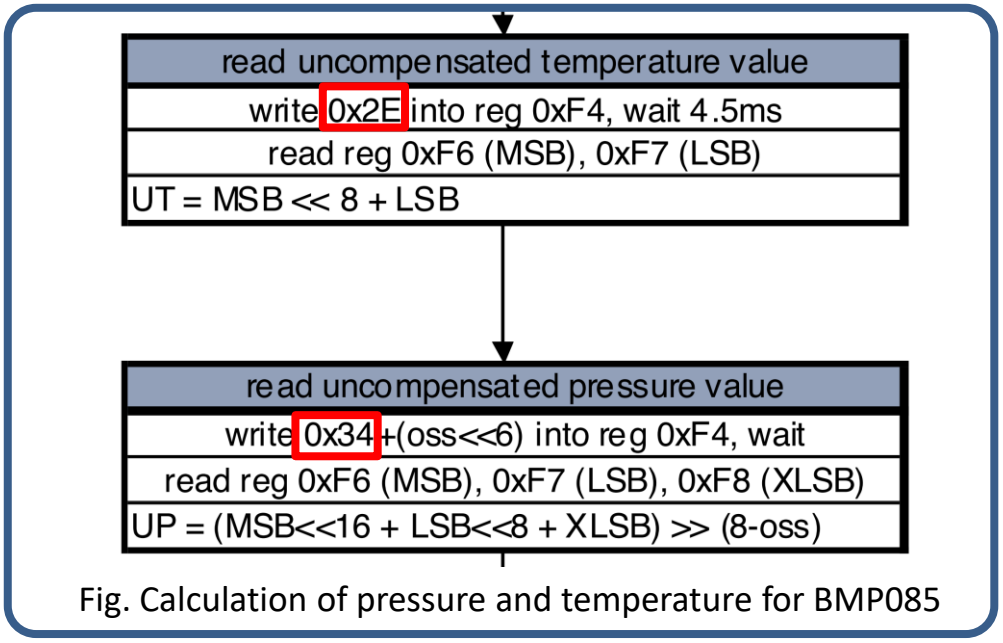


Fig. Calculation of pressure and temperature for BMP085

```
# Commands
BMP085_READTEMPCMD      = 0x2E
BMP085_READPRESSURECMD  = 0x34
```

```
# BMP085 Registers
BMP085_CONTROL          = 0xF4
BMP085_TEMPDATA         = 0xF6
BMP085_PRESSUREDATA     = 0xF6
```

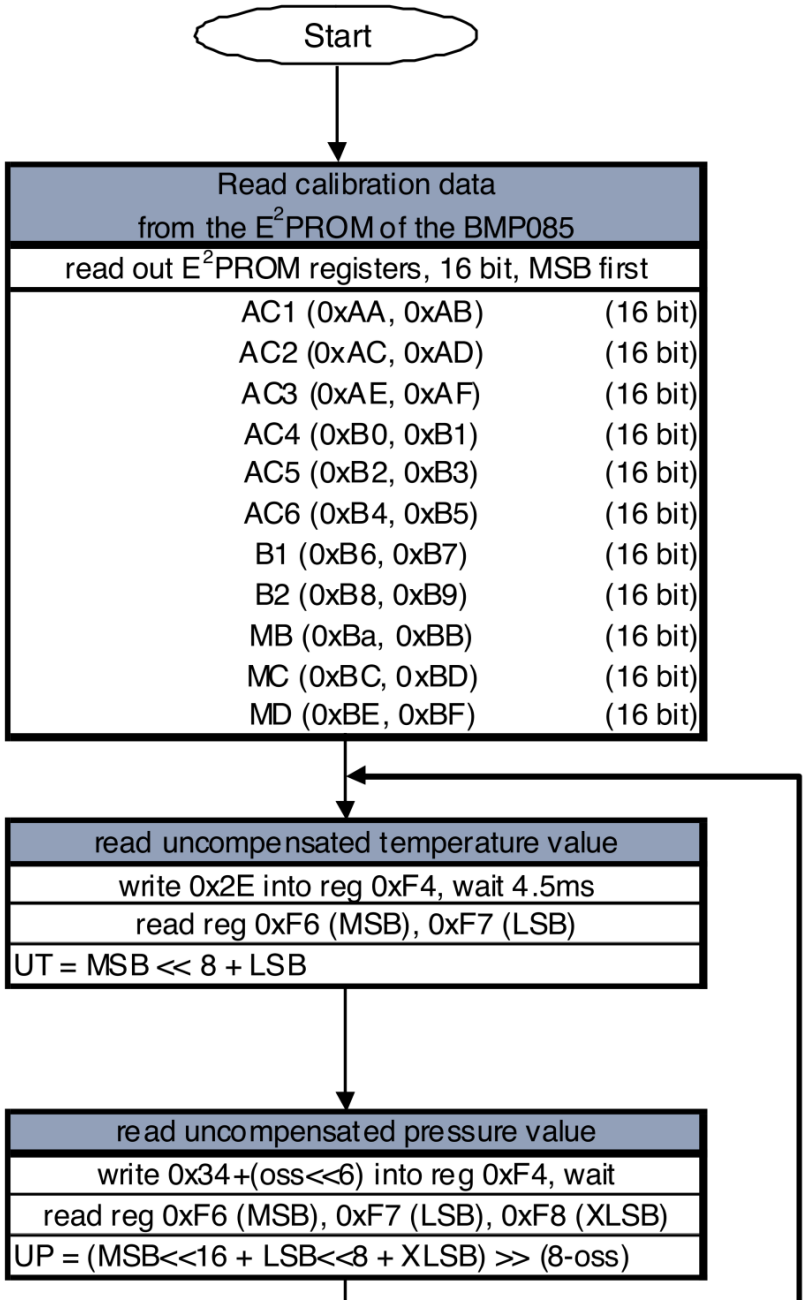
4. BMP085 Datasheet

- Debug coefficients (datasheet)

```
def _load_datasheet_calibration(self):
    # Set calibration from values
    # in the datasheet example.
    # Useful for debugging the
    # temp and pressure calculation accuracy.
    self.cal_AC1 = 408
    self.cal_AC2 = -72
    self.cal_AC3 = -14383
    self.cal_AC4 = 32741
    self.cal_AC5 = 32757
    self.cal_AC6 = 23153
    self.cal_B1 = 6190
    self.cal_B2 = 4
    self.cal_MB = -32767
    self.cal_MC = -8711
    self.cal_MD = 2868
```

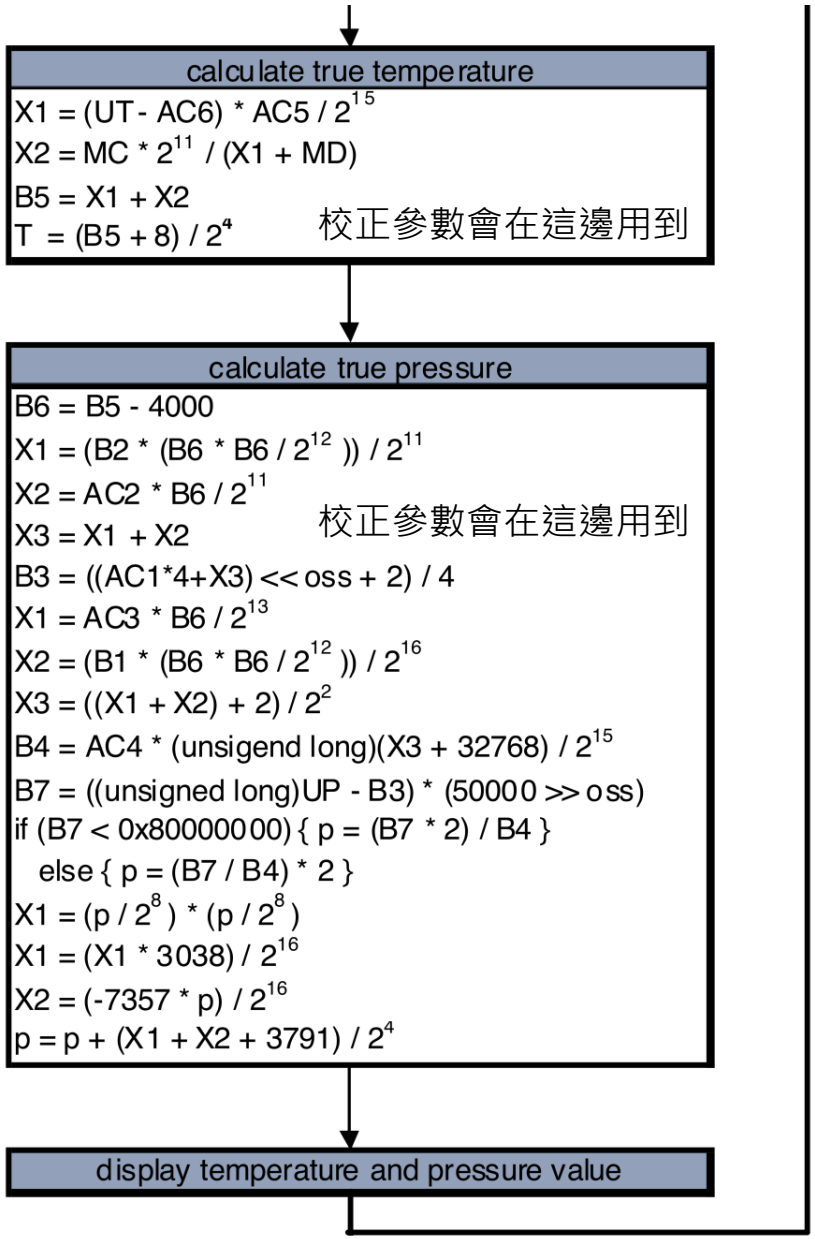
AC1 =	408
AC2 =	-72
AC3 =	-14383
AC4 =	32741
AC5 =	32757
AC6 =	23153
B1 =	6190
B2 =	4
MB =	-32768
MC =	-8711
MD =	2868

Datasheet example



Measurement flowchart

	example:	C code function:	type:
		bmp085_get_cal_param	
AC1 =	408		short
AC2 =	-72		short
AC3 =	-14383		short
AC4 =	32741		unsigned short
AC5 =	32757		unsigned short
AC6 =	23153		unsigned short
B1 =	6190		short
B2 =	4		short
MB =	-32768		short
MC =	-8711		short
MD =	2868		short
		bmp085_get_ut	
UT =	**27898**		long
oss =	0		short (0 .. 3)
= oversampling_setting			
(ultra low power mode)		bmp085_get_up	
UP =	**23843**		long



bmp085_get_temperature

X1 =	4743		long
X2 =	-2344		long
B5 =	2399		long
T =	**150**	***temp in 0.1℃***	long

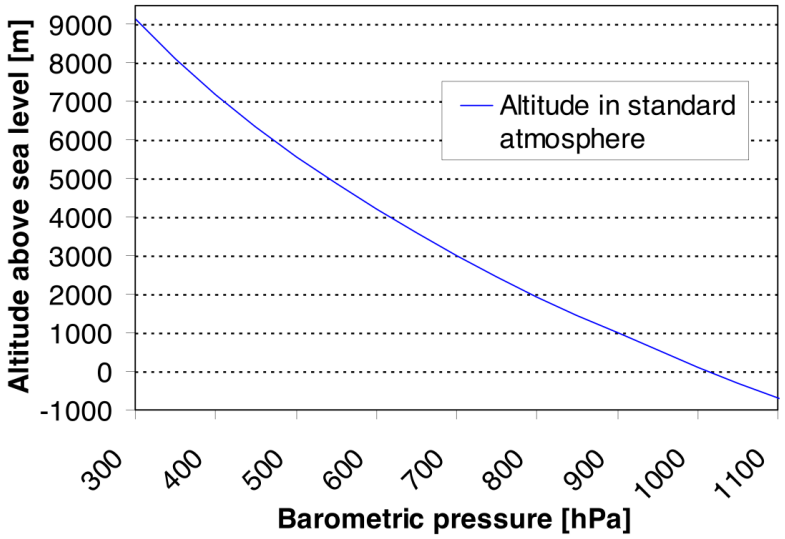
BMP085_calpressure

B6 =	-1601		long
X1 =	1		long
X2 =	56		long
X3 =	57		long
B3 =	422		long
X1 =	2810		long
X2 =	59		long
X3 =	717		long
B4 =	33457		unsigned long
B7 =	1171050000		unsigned long
p =	70003		long
			long
X1 =	74529		long
X1 =	3454		long
X2 =	-7859		long
p =	**69964**	***press. in Pa***	long

Measurement flowchart

4. Altitude (BMP085)

- How to calculate altitude?



$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

```
print('Altitude = {0:0.2f} m'.format(sensor.read_altitude()))
```

The code is already done for you.

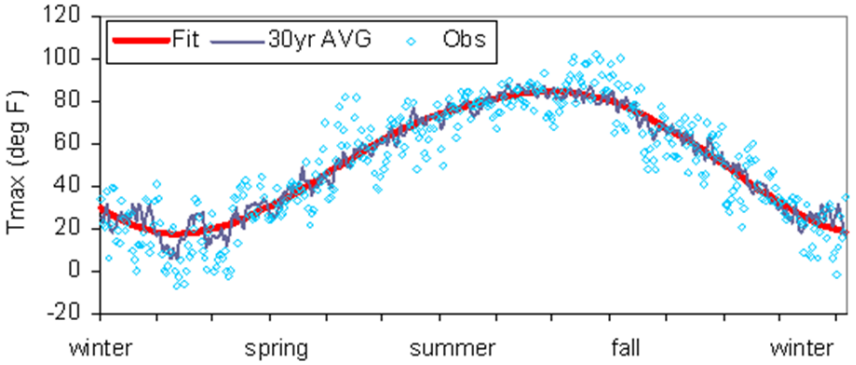
Todo: Integrate code

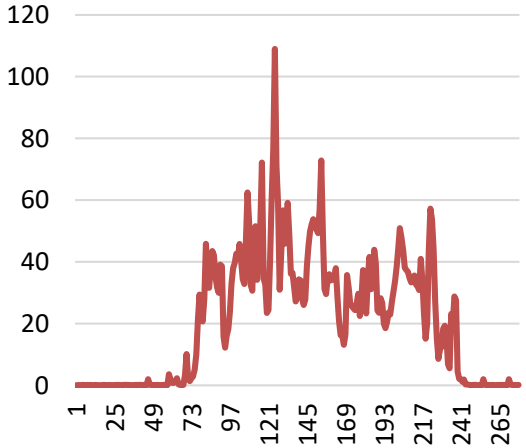
- GY801 can provide 3-axis Accelerometer, Gyroscope, magnetometer and pressure.
- **Integrate the codes, print accelerometer, gyroscope, magnetometer and altitude at the same time.**
  - The sensing value should be converted to corresponding unit.
  - Ex: accel = +/-2G

ACC: x: ? ,y: ? ,z: ?; GYRO: x: ? ,y ?,z: ?; MAG: x: ?, y: ?, z: ?; Alti: ?.

After obtain sensing value

- How to smooth the sensing data? -> Use filter
  1. Moving average
  2. Low-pass filter
  3. Kalman filter
  4. Complementary filter





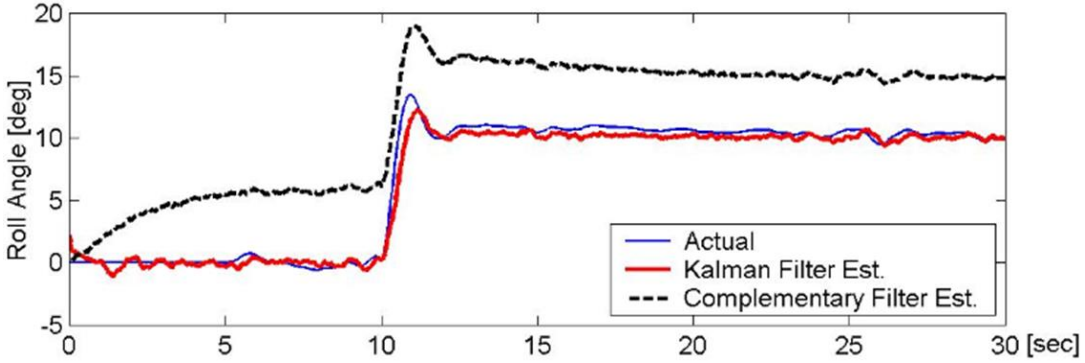
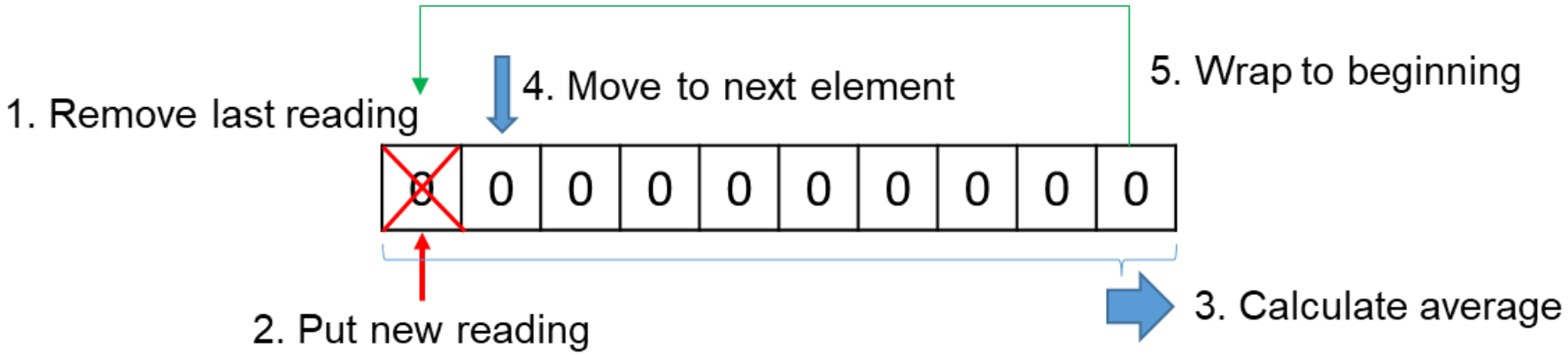


Fig source: https://github.com/snowpuppy/augreality/wiki/Alec-green-notebook
Fig source: http://www.nws.noaa.gov/om/csd/pds/PCU2/statistics/Stats/part2/Filter_LP.htm

1. Moving average

- Basic concept:



2. Low-pass Filter

- Features
  - letting the low frequencies pass and attenuating the high frequencies
  - Simple to implement

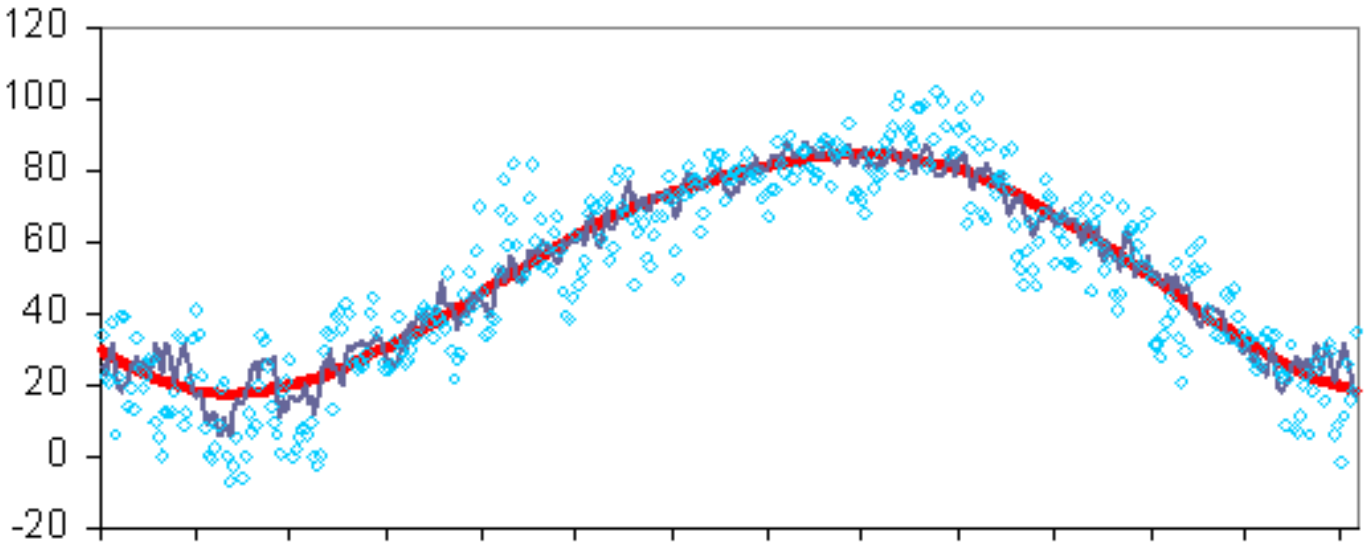


Fig source: http://www.nws.noaa.gov/om/csd/pds/PCU2/statistics/Stats/part2/Filter_LP.htm

2. Low-pass Filter

- Implement low-pass filter in the code
  - x[]: current reading
  - y[]: output value
  - α: smoothing factor (0-1)

y[t] := α * x[t] + (1-α) * y[t-1]

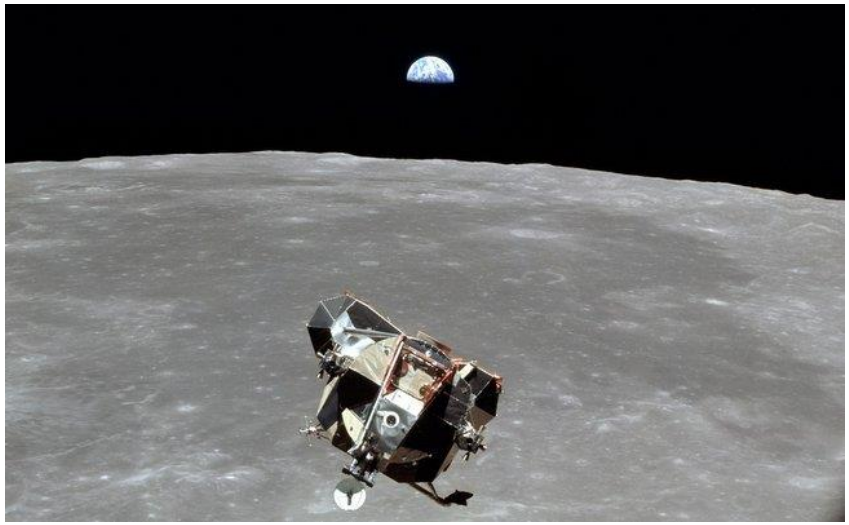
```
// Pseudocode algorithm
// Return RC low-pass filter output samples, given input samples,
// time interval dt, and time constant RC
function lowpass(real[0..n] x, real dt, real RC)
  var real[0..n] y
  var real α := dt / (RC + dt)
  y[0] := x[0]
  for i from 1 to n
     y[i] := α * x[i] + (1-α) * y[i-1]
  return y
```

https://en.wikipedia.org/wiki/Low-pass_filter

3. Kalman filter

- Applications
  - Determination of planet orbit parameters from limited earth observations

A Kalman filter was used to assist with navigation in the Apollo 11 moon landing.



https://stanford.edu/class/ee363/lectures/kf.pdf
http://www.iceira.ntu.edu.tw/project-plans/200-iii

3. Kalman filter

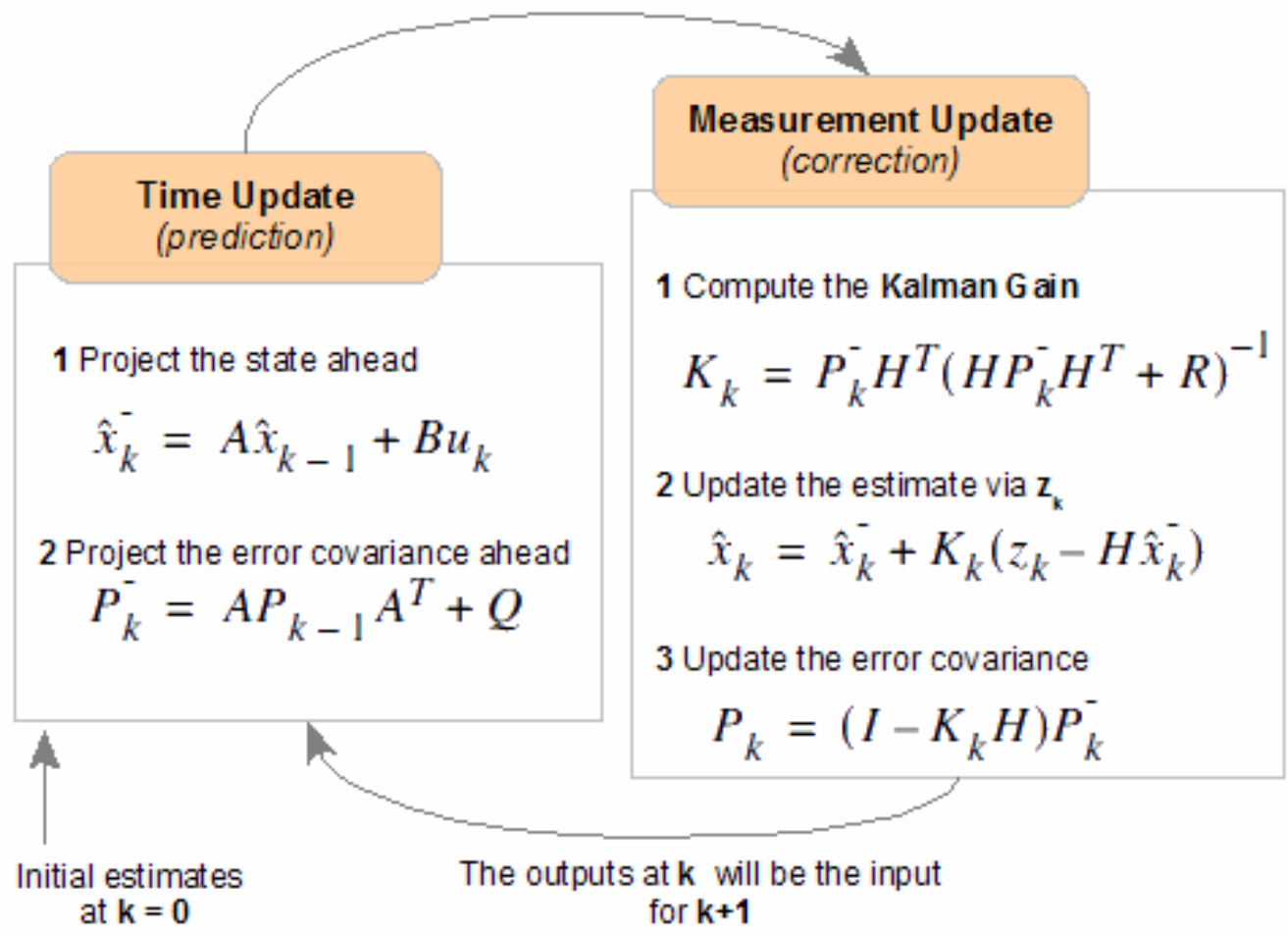
- Kalman filter is an optimal estimator
  - infer parameters of interest from indirect, inaccurate and uncertain observations
  - produce estimates of unknown variables that tend to be more accurate

- Why Kalman filter?
  - Good results in practice due to optimality and structure
  - Convenient form for online real time processing
  - Easy to formulate and implement given a basic understanding

http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

https://www.codeproject.com/Articles/865935/Object-Tracking-Kalman-Filter-with-Ease
http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies

3. Kalman filter

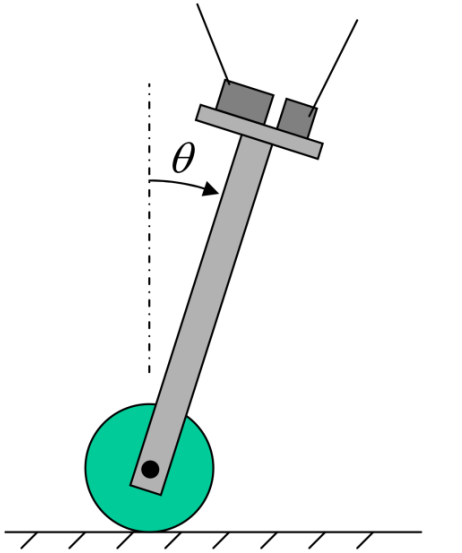


We can use Kalman filter to combine the accelerometer and gyro values.

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

4. Complementary filter

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

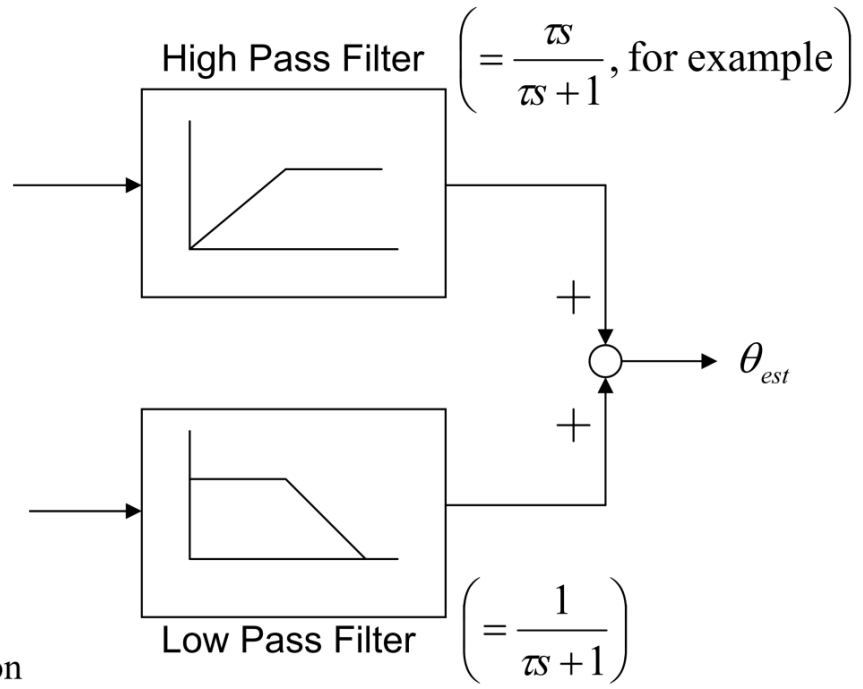


$\theta \approx \int (\text{angular rate})\, dt$

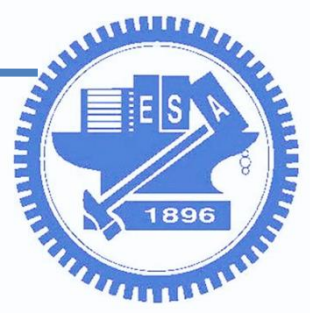
- not good in long term due to integration

$\theta \approx \sin^{-1}\left(\frac{\text{accel. output}}{g}\right)$

- only good in long term
- not proper during fast motion



We can use Complementary filter to combine the accelerometer and gyro values.



BerryIMU (for 3 and 4)

- The code currently performs **angle measurements** using the **gyroscope** and **accelerometer**, which are fused using a **complementary filter** and **kalman filter**.

```
#Complementary filter used to combine the accelerometer and gyro values.
CFangleX=AA*(CFangleX+rate_gyr_x*LP) +(1 - AA) * AccXangle
CFangleY=AA*(CFangleY+rate_gyr_y*LP) +(1 - AA) * AccYangle

#Kalman filter used to combine the accelerometer and gyro values.
kalmanY = kalmanFilterY(AccYangle, rate_gyr_y,LP)
kalmanX = kalmanFilterX(AccXangle, rate_gyr_x,LP)
```

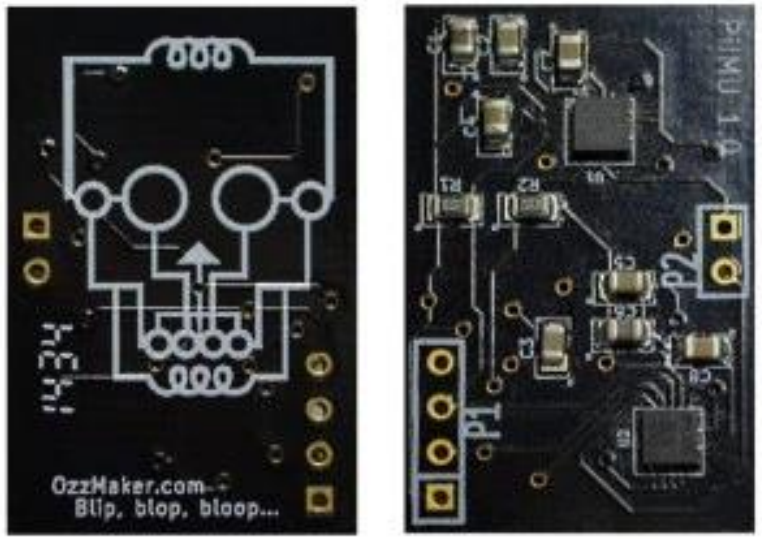
- Original source code:
  - https://github.com/mwilliams03/BerryIMU/blob/master/python-BerryIMU-gryo-accel-compass/berryIMU.py

Download code to your PI:

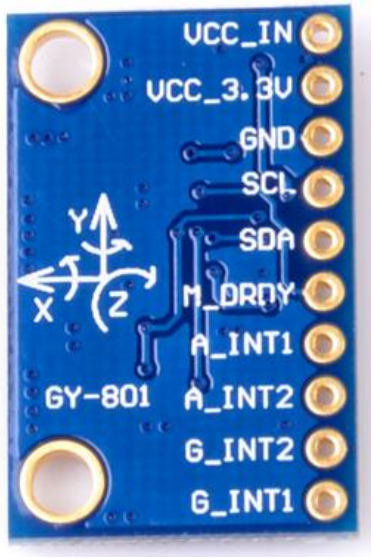
wget http://140.113.144.127/~wufish/12kalman_and_comp__blank.py

BerryIMU

- The code is based on BerryIMU
  - 3D accelerometer, 3D gyroscope, 3D magnetometer
  - Different sensor module, I2C address …



BerryIMU



GY801

BerryIMU

- Basic flowchart of code

Sensing value

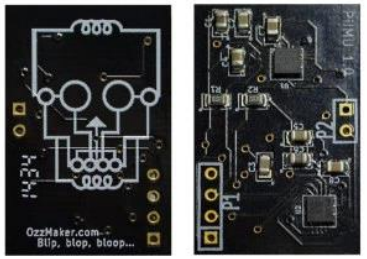
(Input)

Complementary Filter
Kalman Filter

Display data

(Output)

Original: Sensor data is from BerryIMU



Todo: Sensor data is from GY801

BerryIMU code

- Import libraries and define parameters

```
import Adafruit_ADXL345
import string
import sys, os, math, time, thread, smbus, random, requests
import Queue
from signal import signal, SIGPIPE, SIG_DFL
import smbus
import time
import math
import datetime
bus = smbus.SMBus(1)

RAD_TO_DEG = 57.29578
M_PI = 3.14159265358979323846
G_GAIN = 0.070  # [deg/s/LSB]  If you change the dps for gyro,
AA =  0.40      # Complementary filter constant
```

BerryIMU code

- Kalman filter

```
#Kalman filter variables
Q_angle = 0.02
Q_gyro = 0.0015
R_angle = 0.005
y_bias = 0.0
x_bias = 0.0
XP_00 = 0.0
XP_01 = 0.0
XP_10 = 0.0
XP_11 = 0.0
YP_00 = 0.0
YP_01 = 0.0
YP_10 = 0.0
YP_11 = 0.0
KFangleX = 0.0
KFangleY = 0.0
```

```
def kalmanFilterY ( accAngle, gyroRate, DT):
    …

def kalmanFilterX ( accAngle, gyroRate, DT):
    …

while True:
    # Read sensor data
    …

    # Kalman filter used to combine
    # the accelerometer and gyro values.
    kalmanY = kalmanFilterY(AccYangle, rate_gyr_y,LP)
    kalmanX = kalmanFilterX(AccXangle, rate_gyr_x,LP)
    …
```

BerryIMU code

- Complementary filter

```
…

while True:
    # Read sensor data
    …

    # Complementary filter used to combine
    # the accelerometer and gyro values.
    CFangleX=AA*(CFangleX+rate_gyr_x*LP) +(1 - AA) * AccXangle
    CFangleY=AA*(CFangleY+rate_gyr_y*LP) +(1 - AA) * AccYangle
    …
```