Introduction to Embedded Systems Design and Implementation HW4

學號: 0711282 姓名: 邱頎霖

Discussion 1
a. Based on the requirements, set correct value in the sample code
requirements:
8 samples per measurement
Data Output Rate = 15Hz
Gain=1090(LSb/Gauss)
Convert LSB to Gauss (by using self.scale)
b. Continuously measurement (infinite loop)
c. Calibrate your sensor

a.

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

Table 7: Configuration B Register

Data Output Rate = 15Hz 因此利用 datasheet 中的格式對應是 {CRA4 CRA3 CRA2} = 100
8 samples per measurement 因此設定 {CR6 CR5} = 11
因此整體 configuration register A 8 bit 為 0b01110000

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

Gain=1090(LSb/Gauss) 因此按照對應為 {CRA6 CRA5} = 0b001
因此整體 configuration register B 8 bit 為 0b00100000

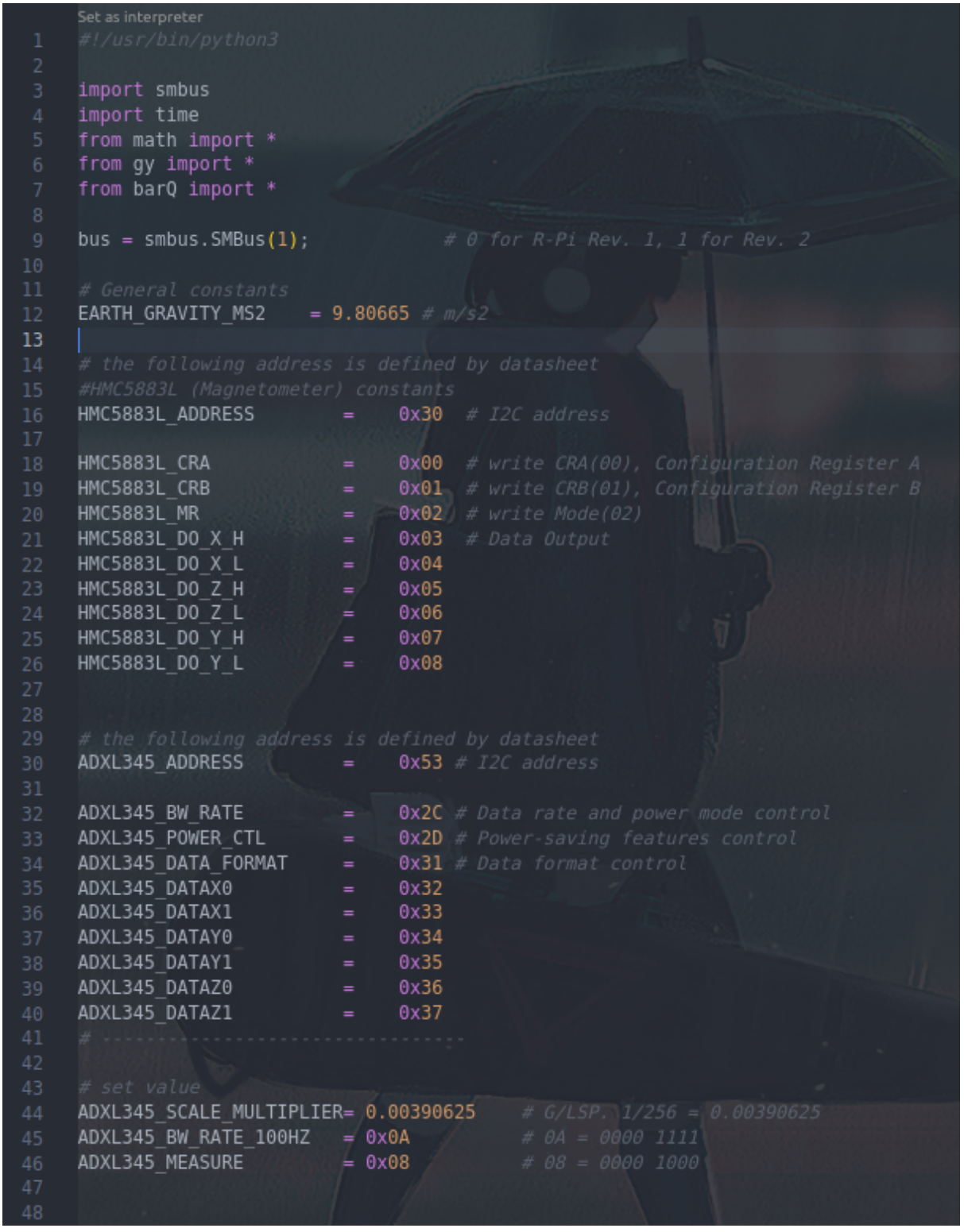
Operating modes 的部份則設定為 Continuous-Measurement mode
因此整體 mode register 8 bit 為 0b00000000

```
        self.scale = 0.92 # convert bit value(LSB) to gauss.
```

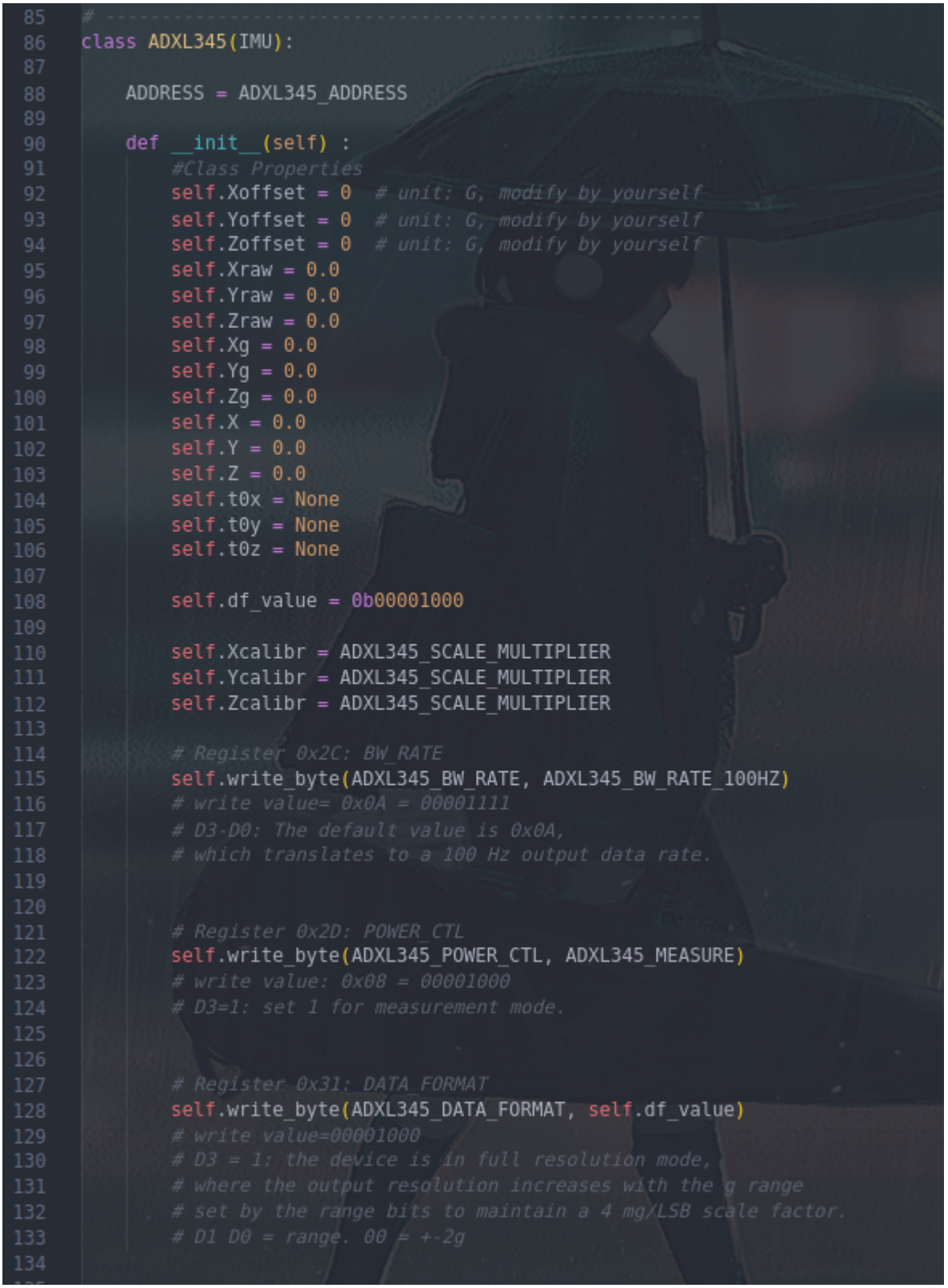
```
        # Configuration Register A
        self.write_byte(HMC5883L_CRA, 0b01110000)

        # Configuration Register B
        self.write_byte(HMC5883L_CRB, 0b00100000)

        # Mode Register
```

```
        self.write_byte(HMC5883L_MR,   0b00000000)
```

b. 多加 while 迴圈, 程式碼連結 (這裡有 import discussion2 的 file(barQ, 其內容寫在 discussion2), discussion 1 與 2 的主程式碼寫在同一個 file 裡面)
截圖:

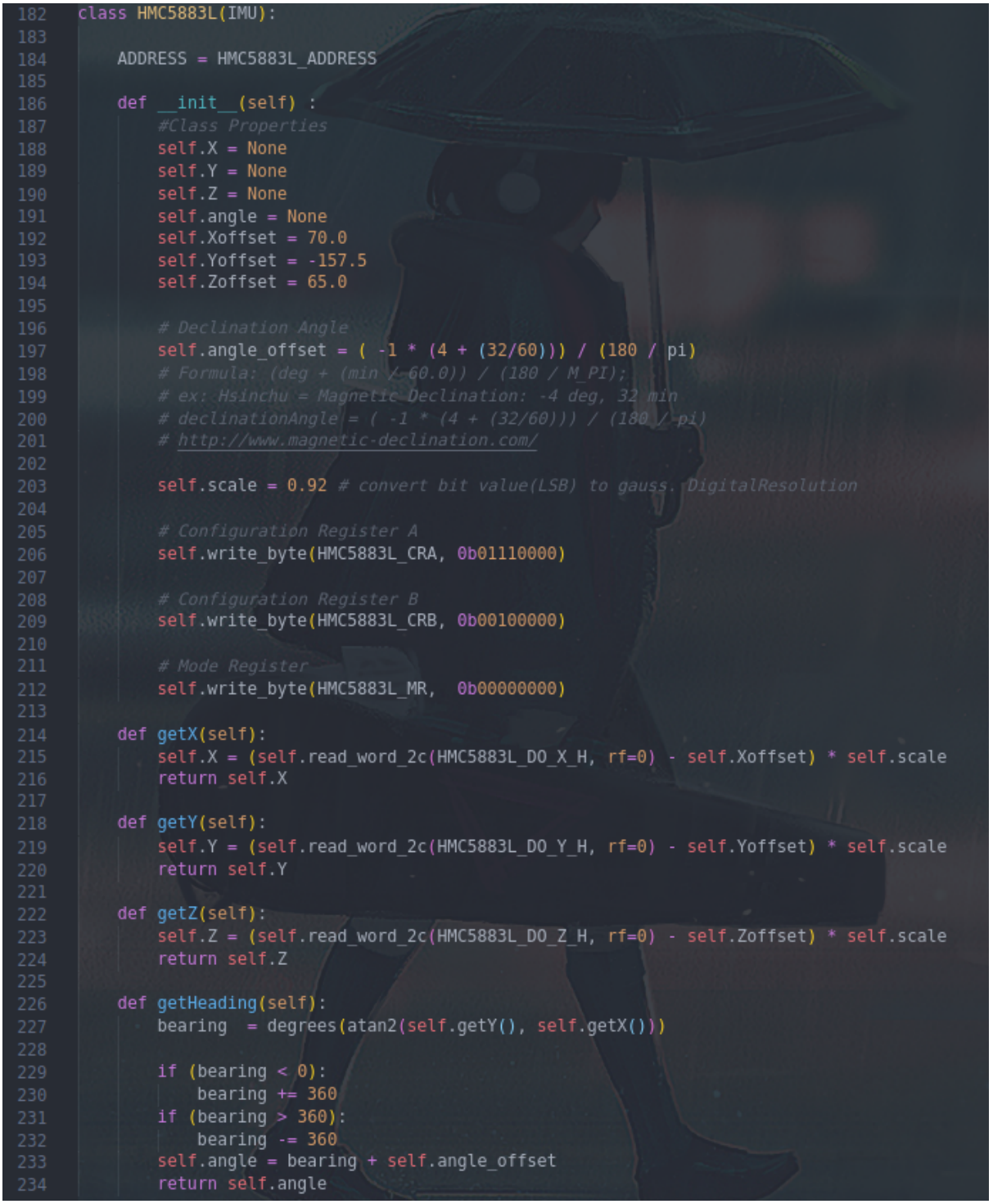


```
class IMU(object):

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

    def read_byte(self,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):
            # acc, gyro
            low = self.read_byte(adr)
            high = self.read_byte(adr+1)
        else:
            # compass
            high = self.read_byte(adr)
            low = self.read_byte(adr+1)
        val = (high << 8) + low
        return val

    def read_word_2c(self,adr,rf=1):
        val = self.read_word(adr,rf)
        if(val & (1 << 16 - 1)):
            return val - (1<<16)
        else:
            return val

class gy801(object):
    def __init__(self) :
        self.compass = HMC5883L()
        self.accel = ADXL345()

```



```python
# ----------------------------------------------------
class ADXL345(IMU):

    ADDRESS = ADXL345_ADDRESS

    def __init__(self) :
        #Class Properties
        self.Xoffset = 0  # unit: G, modify by yourself
        self.Yoffset = 0  # unit: G, modify by yourself
        self.Zoffset = 0  # unit: G, modify by yourself
        self.Xraw = 0.0
        self.Yraw = 0.0
        self.Zraw = 0.0
        self.Xg = 0.0
        self.Yg = 0.0
        self.Zg = 0.0
        self.X = 0.0
        self.Y = 0.0
        self.Z = 0.0
        self.t0x = None
        self.t0y = None
        self.t0z = None

        self.df_value = 0b00001000

        self.Xcalibr = ADXL345_SCALE_MULTIPLIER
        self.Ycalibr = ADXL345_SCALE_MULTIPLIER
        self.Zcalibr = ADXL345_SCALE_MULTIPLIER

        # 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

```

```
135
136     # RAW readings in LPS
137     # Register 0x32 to Register 0x37:
138     # DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)
139     def getRawX(self) :
140         self.Xraw = self.read_word_2c(ADXL345_DATAX0)
141         return self.Xraw
142 
143     def getRawY(self) :
144         self.Yraw = self.read_word_2c(ADXL345_DATAY0)
145         return self.Yraw
146 
147     def getRawZ(self) :
148         self.Zraw = self.read_word_2c(ADXL345_DATAZ0)
149         return self.Zraw
150 
151     # G related readings in g
152     # similar to filter. combine current value with previous one.
153     # plf = 1 means it only uses "current reading"
154     def getXg(self,plf = 1.0) :
155         self.Xg = (self.getRawX() * self.Xcalibr - self.Xoffset) * plf + (1.0 - plf) * self.Xg
156         return self.Xg
157 
158     def getYg(self,plf = 1.0) :
159         self.Yg = (self.getRawY() * self.Ycalibr - self.Yoffset) * plf + (1.0 - plf) * self.Yg
160         return self.Yg
161 
162     def getZg(self,plf = 1.0) :
163         self.Zg = (self.getRawZ() * self.Zcalibr - self.Zoffset) * plf + (1.0 - plf) * self.Zg
164         return self.Zg
165 
166     # Absolute reading in m/s2
167     def getX(self,plf = 1.0) :
168         self.X = self.getXg(plf) * EARTH_GRAVITY_MS2
169         return self.X
170 
171     def getY(self,plf = 1.0) :
172         self.Y = self.getYg(plf) * EARTH_GRAVITY_MS2
173         return self.Y
174 
175     def getZ(self,plf = 1.0) :
176         self.Z = self.getZg(plf) * EARTH_GRAVITY_MS2
177         return self.Z
178 
179
```



```
class HMC5883L(IMU):

    ADDRESS = HMC5883L_ADDRESS

    def __init__(self) :
        #Class Properties
        self.X = None
        self.Y = None
        self.Z = None
        self.angle = None
        self.Xoffset = 70.0
        self.Yoffset = -157.5
        self.Zoffset = 65.0

        # Declination Angle
        self.angle_offset = ( -1 * (4 + (32/60))) / (180 / pi)
        # Formula: (deg + (min / 60.0)) / (180 / M_PI);
        # ex: Hsinchu = Magnetic Declination: -4 deg, 32 min
        # declinationAngle = ( -1 * (4 + (32/60))) / (180 / pi)
        # http://www.magnetic-declination.com/

        self.scale = 0.92 # convert bit value(LSB) to gauss. DigitalResolution

        # Configuration Register A
        self.write_byte(HMC5883L_CRA, 0b01110000)

        # Configuration Register B
        self.write_byte(HMC5883L_CRB, 0b00100000)

        # Mode Register
        self.write_byte(HMC5883L_MR,  0b00000000)

    def getX(self):
        self.X = (self.read_word_2c(HMC5883L_DO_X_H, rf=0) - self.Xoffset) * self.scale
        return self.X

    def getY(self):
        self.Y = (self.read_word_2c(HMC5883L_DO_Y_H, rf=0) - self.Yoffset) * self.scale
        return self.Y

    def getZ(self):
        self.Z = (self.read_word_2c(HMC5883L_DO_Z_H, rf=0) - self.Zoffset) * self.scale
        return self.Z

    def getHeading(self):
        bearing  = degrees(atan2(self.getY(), self.getX()))

        if (bearing < 0):
            bearing += 360
        if (bearing > 360):
            bearing -= 360
        self.angle = bearing + self.angle_offset
        return self.angle
```

```
pitcharr = []
rollarr = []
pitcharr.append(0)
pitcharr.append(0)
rollarr.append(0)
rollarr.append(0)

try:
    sensors = gy801()
    compass = sensors.compass
    adxl345 = sensors.accel

    # ################ LAB3 ######################
    sensor2 = gy801_2()
    gyro = sensor2.gyro
    # ################ LAB3 ######################

    sensors3 = gy801_3()
    barometer = sensors3.baro

    while True:
        magx = compass.getX()
        magy = compass.getY()
        magz = compass.getZ()

        # -------------------------------------------------
        # calculate pitch, roll, tilt
        aX = adxl345.getX()
        aY = adxl345.getY()
        aZ = adxl345.getZ()

        roll = atan2(aY,aZ)
        pitch = -1 * atan2(-aX,sqrt(aY*aY+aZ*aZ))
        # -------------------------------------------------


        # -------------------------------------------------
        # Heading
        bearing1  = degrees(atan2(magy, magx))

        if (bearing1 < 0):
            bearing1 += 360
        if (bearing1 > 360):
            bearing1 -= 360
        bearing1 = bearing1 + compass.angle_offset

        # Tilt compensate
        compx = magx * cos(pitch) + magz * sin(pitch)
        compy = magx * sin(roll) * sin(pitch) \
                + magy * cos(roll) \
                - magz * sin(roll) * cos(pitch)
```

```
        # -----------------------------------------------
        # Heading
        bearing1  = degrees(atan2(magy, magx))

        if (bearing1 < 0):
            bearing1 += 360
        if (bearing1 > 360):
            bearing1 -= 360
        bearing1 = bearing1 + compass.angle_offset

        # Tilt compensate
        compx = magx * cos(pitch) + magz * sin(pitch)
        compy = magx * sin(roll) * sin(pitch) \
                + magy * cos(roll) \
                - magz * sin(roll) * cos(pitch)

        bearing2  = degrees(atan2(compy, compx))
        if (bearing2 < 0):
            bearing2 += 360
        if (bearing2 > 360):
            bearing2 -= 360
        bearing2 = bearing2 + compass.angle_offset
        # -----------------------------------------------

        # ################ LAB3 ####################
        tmpPitch = (pitcharr[0]+gyro.getXangle())*0.98+pitch*0.02
        tmpRoll = (rollarr[0]+gyro.getYangle())*0.98+roll*0.02
        pitcharr[0] = pitcharr[1]
        pitcharr[1] = tmpPitch
        rollarr[0] = rollarr[1]
        rollarr[1] = tmpRoll
        # print("pitch:", pitcharr[1], "roll:", rollarr[1])
        # ################ LAB3 ####################

#        print ("Compass: " )
#        print ("X = %d ," % ( magx )),
#        print ("Y = %d ," % ( magy )),
#        print ("Z = %d (gauss)" % ( magz ))
#        print ("tiltX = %.3f ," % ( compx )),
#        print ("tiltY = %.3f ," % ( compy )),

        xh = magx*cos(pitcharr[1])+magz*sin(pitcharr[1])
        yh = magx*sin(rollarr[1])*sin(pitcharr[1])+magy*cos(rollarr[1])-magz*sin(rollarr[1])*cos(pitcharr[1])

        # print(xh," ",yh)
        print("Pitch: ",pitcharr[1], " Roll: ", rollarr[1])
        print("Heading", atan2(yh,xh)*180/pi)

#        print ("Angle offset = %.3f deg" % ( compass.angle_offset ))
        print ("Original Heading = %.3f deg, " % ( bearing1 )),
        print ("Tilt Heading = %.3f deg, " % ( bearing2 ))

        tempC = barometer.getTempC()
        tempF = barometer.getTempF()
        press = barometer.getPress()
        altitude = barometer.getAltitude()
```

```
            tempC = barometer.getTempC()
            tempF = barometer.getTempF()
            press = barometer.getPress()
            altitude = barometer.getAltitude()

            # print("Barometer:")
            # print("    Temp: %f C (%f F)" % (tempC, tempF))
            # print("    Press: %f (hPa)" % (press))
            print("    Altitude: %f m s.l.m" % (altitude))
            print("")
            time.sleep(1)

    except KeyboardInterrupt:
        print("Cleanup")
```

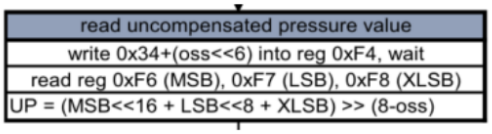
c. 校正

照著投影片的校正動作並同時跑 **"3calibrate-hmc5883l.py"** 之後會輸出 **x, y, z offset** 的數值, 我的結果為:

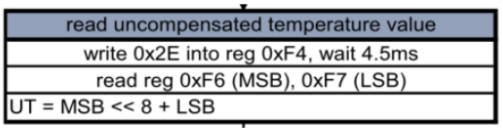
```
        self.Xoffset = 70.0
        self.Yoffset = -157.5
        self.Zoffset = 65.0
```

Discussion 2

- 1. Based on the datasheet, set correct value in the sample code
  - Standard mode



read uncompensated pressure value
write 0x34+(oss<<6) into reg 0xF4, wait
read reg 0xF6 (MSB), 0xF7 (LSB), 0xF8 (XLSB)
UP = (MSB<<16 + LSB<<8 + XLSB) >> (8-oss)



read uncompensated temperature value
write 0x2E into reg 0xF4, wait 4.5ms
read reg 0xF6 (MSB), 0xF7 (LSB)
UT = MSB << 8 + LSB

```
def getPress(self) :
    # print ("Calculating temperature...")
    self.write_byte(0xF4, 0x??)
    time.sleep(0.005)
```

```
# read uncompensated temperature value
def getTempC(self) :
    # print ("Calculating temperature...")
    self.write_byte(0xF4, 0x??)
    time.sleep(0.005)
```

- 2. Continuously measurement(infinite loop)

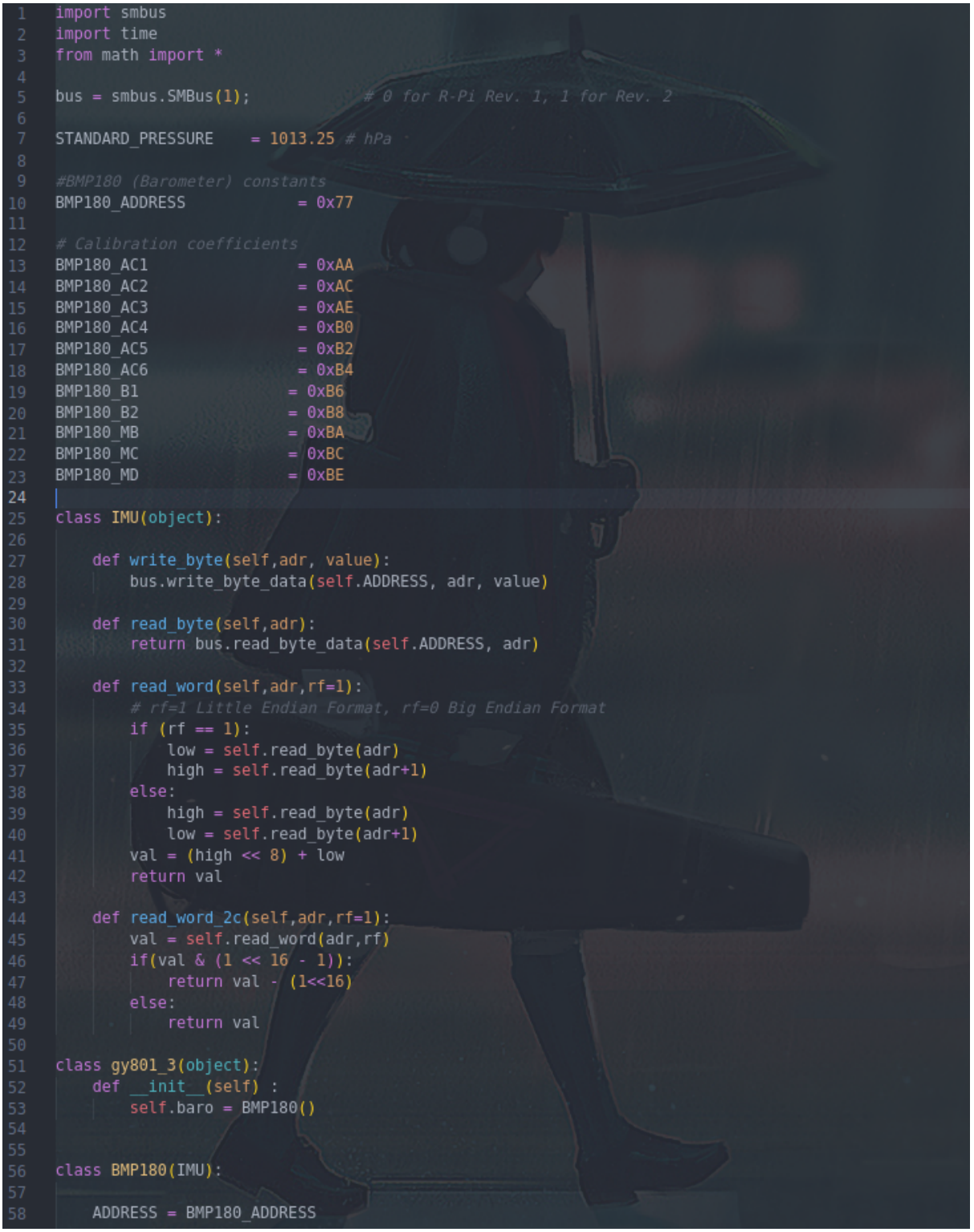
1.

```
    def getTempC(self) :
        # print ("Calculating temperature...")
        self.write_byte(0xF4, 0x2E)
        time.sleep(0.005)
```

```
    def getPress(self) :
        # print ("Calculating temperature...")
        self.write_byte(0xF4, 0x2E)
        time.sleep(0.005)
```

2.

discussion 1 與 **2** 的主程式寫在同一個 **file(**在 **discussion 1** 的 **b),** 因此這裡僅附上 **import** 的 **file(barQ)** 內容**:**



```python
import smbus
import time
from math import *

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

STANDARD_PRESSURE    = 1013.25 # hPa

#BMP180 (Barometer) constants
BMP180_ADDRESS            = 0x77

# Calibration coefficients
BMP180_AC1                = 0xAA
BMP180_AC2                = 0xAC
BMP180_AC3                = 0xAE
BMP180_AC4                = 0xB0
BMP180_AC5                = 0xB2
BMP180_AC6                = 0xB4
BMP180_B1                 = 0xB6
BMP180_B2                 = 0xB8
BMP180_MB                 = 0xBA
BMP180_MC                 = 0xBC
BMP180_MD                 = 0xBE

class IMU(object):

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

    def read_byte(self,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):
            low = self.read_byte(adr)
            high = self.read_byte(adr+1)
        else:
            high = self.read_byte(adr)
            low = self.read_byte(adr+1)
        val = (high << 8) + low
        return val

    def read_word_2c(self,adr,rf=1):
        val = self.read_word(adr,rf)
        if(val & (1 << 16 - 1)):
            return val - (1<<16)
        else:
            return val

class gy801_3(object):
    def __init__(self) :
        self.baro = BMP180()


class BMP180(IMU):

    ADDRESS = BMP180_ADDRESS
```

```
class gy801_3(object):
    def __init__(self) :
        self.baro = BMP180()


class BMP180(IMU):

    ADDRESS = BMP180_ADDRESS

    def __init__(self) :
        #Class Properties
        self.tempC = None
        self.tempF = None
        self.press = None
        self.altitude = None

        self.oversampling = 0

        self._read_calibratio_params()

    # read calibration data
    def _read_calibratio_params(self) :
        self.ac1_val = self.read_word_2c(BMP180_AC1,0)
        self.ac2_val = self.read_word_2c(BMP180_AC2,0)
        self.ac3_val = self.read_word_2c(BMP180_AC3,0)
        self.ac4_val = self.read_word(BMP180_AC4,0)
        self.ac5_val = self.read_word(BMP180_AC5,0)
        self.ac6_val = self.read_word(BMP180_AC6,0)
        self.b1_val = self.read_word_2c(BMP180_B1,0)
        self.b2_val = self.read_word_2c(BMP180_B2,0)
        self.mc_val = self.read_word_2c(BMP180_MC,0)
        self.md_val = self.read_word_2c(BMP180_MD,0)

    # read uncompensated temperature value
    def getTempC(self) :
        # print ("Calculating temperature...")
        self.write_byte(0xF4, 0x2E)
        time.sleep(0.005)

        ut = self.read_word(0xF6,0)

        # calculate true temperature
        x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
        x2 = (self.mc_val << 11) // (x1 + self.md_val)
        b5 = x1 + x2
        self.tempC = ((b5 + 8) >> 4) / 10.0

        return self.tempC

    def getTempF(self) :
        #print ("Calculating temperature (Fahrenheit)...")
        self.tempF = self.getTempC() * 1.8 + 32

        return self.tempF

```

```
    def getTempF(self) :
        #print ("Calculating temperature (Fahrenheit)...")
        self.tempF = self.getTempC() * 1.8 + 32

        return self.tempF

    # read uncompensated pressure value
    def getPress(self) :
        # print ("Calculating temperature...")
        self.write_byte(0xF4, 0x2E)
        time.sleep(0.005)

        ut = self.read_word(0xF6,0)

        x1 = ((ut - self.ac6_val) * self.ac5_val) >> 15
        x2 = (self.mc_val << 11) // (x1 + self.md_val)
        b5 = x1 + x2

        #print ("Calculating pressure...")
        self.write_byte(0xF4, 0x34 + (self.oversampling << 6))
        time.sleep(0.04)

        msb = self.read_byte(0xF6)
        lsb = self.read_byte(0xF7)
        xsb = self.read_byte(0xF8)

        up = ((msb << 16) + (lsb << 8) + xsb) >> (8 - self.oversampling)

        # calculate true pressure
        b6 = b5 - 4000
        b62 = b6 * b6 >> 12
        x1 = (self.b2_val * b62) >> 11
        x2 = self.ac2_val * b6 >> 11
        x3 = x1 + x2
        b3 = (((self.ac1_val * 4 + x3) << self.oversampling) + 2) >> 2

        x1 = self.ac3_val * b6 >> 13
        x2 = (self.b1_val * b62) >> 16
        x3 = ((x1 + x2) + 2) >> 2
        b4 = (self.ac4_val * (x3 + 32768)) >> 15
        b7 = (up - b3) * (50000 >> self.oversampling)

        press = (b7 * 2) // b4
        #press = (b7 / b4) * 2

        x1 = (press >> 8) * (press >> 8)
        x1 = (x1 * 3038) >> 16
        x2 = (-7357 * press) >> 16
        self.press = ( press + ((x1 + x2 + 3791) >> 4) ) / 100.0

        return self.press
```

```
    def getAltitude(self) :
        #    print ("Calculating altitude...")
        self.altitude = 44330 * (1 - ((self.getPress() / STANDARD_PRESSURE) ** 0.1903))
        return self.altitude
```