```python
import sys
import wiringpi as wi
import time
import datetime
import os



class MPU9250:
    # coefficient
    gyro_range = 1000
    accel_range = 8
    mag_range = 4912

    REG_PWR_MGMT_1 = 0x6B
    REG_INT_PIN_CFG = 0x37
    REG_ACCEL_CONFIG_1 = 0x1C
    REG_ACCEL_CONFIG_2 = 0x1D
    REG_GYRO_CONFIG = 0x1B

    MAG_MODE_POWER_DOWN = 0
    MAG_MODE_SERIAL_1 = 1
    MAG_MODE_SERIAL_2 = 2
    MAG_MODE_SINGLE = 3
    MAG_MODE_EX_TRIGGER = 4
    MAG_MODE_SELF_TEST = 5
    MAG_ACCESS = False
    MAG_MODE = 0
    MAG_BIT = 16

    offset_accel_x = 0
    offset_accel_y = 0
    offset_accel_z = 0
    offset_gyro_x = 0
    offset_gyro_y = 0
    offset_gyro_z = 0

    def __init__(self, sampling_time, mpu9250_address, ak8963_address):
        self.sampling_time = sampling_time
        self.mpu9250_address = mpu9250_address
        self.ak8963_address = ak8963_address

        wi.wiringPiSetup()
        self.i2c = wi.I2C()

        self.mpu9250 = self.i2c.setup(mpu9250_address)
        self.ak8963 = self.i2c.setup(ak8963_address)

        self.reset_register()
        self.power_wake_up()

        self.gyro_coefficient = self.gyro_range / float(0x8000)
        self.accel_coefficient = self.accel_range / float(0x8000)
        self.mag_coefficient_16 = self.mag_range / 32760.0
        self.mag_coefficient_14 = self.mag_range / 8190.0

    def reset_register(self):
        if self.MAG_ACCESS:
            self.i2c.writeReg8(self.ak8963, 0x0B, 0x01)
        self.i2c.writeReg8(self.mpu9250, 0x6B, 0x80)
        self.MAG_ACCESS = False
        time.sleep(0.1)

    def power_wake_up(self):
        self.i2c.writeReg8(self.mpu9250, self.REG_PWR_MGMT_1, 0x00)
        time.sleep(0.1)
        self.i2c.writeReg8(self.mpu9250, self.REG_INT_PIN_CFG, 0x02)
        self.MAG_ACCESS = True
        time.sleep(0.1)


    def set_accel_range(self, val=8, _calibration=False):
        if val == 16:
            self.accel_range = 16
            _data = 0x18
        elif val == 8:
            self.accel_range = 8
            _data = 0x10
        elif val == 4:
            self.accel_range = 4
```



```python
            _data = 0x08
        else:
            self.accel_range = 2
            _data = 0x00
        self.i2c.writeReg8(self.mpu9250, self.REG_ACCEL_CONFIG_1, _data)
        self.accel_coefficient = self.accel_range / float(0x8000)
        time.sleep(0.1)

        self.offset_accel_x = 0
        self.offset_accel_y = 0
        self.offset_accel_z = 0

        if _calibration:
            self.calibration_accel(1000)
        return

    def set_gyro_range(self, val, _calibration=False):
        if val == 2000:
            self.gyro_range = 2000
            _data = 0x18
        elif val == 1000:
            self.gyro_range = 1000
            _data = 0x10
        elif val == 500:
            self.gyro_range = 500
            _data = 0x08
        else:
            self.gyro_range = 250
            _data = 0x00
        self.i2c.writeReg8(self.mpu9250, self.REG_GYRO_CONFIG, _data)
        self.gyro_coefficient = self.gyro_range / float(0x8000)
        time.sleep(0.1)

        self.offset_gyro_x = 0
        self.offset_gyro_y = 0
        self.offset_gyro_z = 0

        if _calibration:
            self.calibration_gyro(1000)
        return

    def set_mag_register(self, _mode, _bit):
        if not self.MAG_ACCESS:
            raise Exception('001 Access to a sensor is invalid.')

        _write_data = 0x00
        if _mode == '8Hz':
            _write_data = 0x02
            self.MAG_MODE = self.MAG_MODE_SERIAL_1
        elif _mode == '100Hz':
            _write_data = 0x06
            self.MAG_MODE = self.MAG_MODE_SERIAL_2
        elif _mode == 'POWER_DOWN':
            _write_data = 0x00
            self.MAG_MODE = self.MAG_MODE_POWER_DOWN
        elif _mode == 'EX_TRIGGER':
            _write_data = 0x04
            self.MAG_MODE = self.MAG_MODE_EX_TRIGGER
        elif _mode == 'SELF_TEST':
            _write_data = 0x08
            self.MAG_MODE = self.MAG_MODE_SELF_TEST
        elif _mode == 'SINGLE':
            _write_data = 0x01
            self.MAG_MODE = self.MAG_MODE_SINGLE
        else:
            raise Exception('002 set_mag_register write data "%s" is not defined' % _mode)

        if _bit == '14bit':
            _write_data = _write_data | 0x00
            self.MAG_BIT = 14
        elif _bit == '16bit':
            self.MAG_BIT = 16
        else:
            raise Exception('003 set_mag_register _bit "%s" is not defined' % _bit)

        self.i2c.writeReg8(self.ak8963, 0x0A, _write_data)


    def u2s(self, unsigned_data):
        if unsigned_data & (0x01 << 15):
```

```
            return -1 * ((unsigned_data ^ 0xffff) + 1)
        return unsigned_data


    def get_accel(self):
        accel_x_high = self.i2c.readReg8(self.mpu9250, 0x3B)
        accel_x_low = self.i2c.readReg8(self.mpu9250, 0x3C)
        accel_y_high = self.i2c.readReg8(self.mpu9250, 0x3D)
        accel_y_low = self.i2c.readReg8(self.mpu9250, 0x3E)
        accel_z_high = self.i2c.readReg8(self.mpu9250, 0x3F)
        accel_z_low = self.i2c.readReg8(self.mpu9250, 0x40)
        raw_x = self.accel_coefficient * self.u2s(accel_x_high << 8 | accel_x_low) + self.offset_accel_x
        raw_y = self.accel_coefficient * self.u2s(accel_y_high << 8 | accel_y_low) + self.offset_accel_y
        raw_z = self.accel_coefficient * self.u2s(accel_z_high << 8 | accel_z_low) + self.offset_accel_z
        return raw_x, raw_y, raw_z


    def get_gyro(self):
        gyro_x_high = self.i2c.readReg8(self.mpu9250, 0x43)
        gyro_x_low = self.i2c.readReg8(self.mpu9250, 0x44)
        gyro_y_high = self.i2c.readReg8(self.mpu9250, 0x45)
        gyro_y_low = self.i2c.readReg8(self.mpu9250, 0x46)
        gyro_z_high = self.i2c.readReg8(self.mpu9250, 0x47)
        gyro_z_low = self.i2c.readReg8(self.mpu9250, 0x48)
        raw_x = self.gyro_coefficient * self.u2s(gyro_x_high << 8 | gyro_x_low) + self.offset_gyro_x
        raw_y = self.gyro_coefficient * self.u2s(gyro_y_high << 8 | gyro_y_low) + self.offset_gyro_y
        raw_z = self.gyro_coefficient * self.u2s(gyro_z_high << 8 | gyro_z_low) + self.offset_gyro_z
        return raw_x, raw_y, raw_z

    def get_mag(self):
        if not self.MAG_ACCESS:
            raise Exception('004 access to a mag sensor denied.')

        if self.MAG_MODE == self.MAG_MODE_SINGLE:
            if self.MAG_BIT == 14:
                _write_data = 0x01
            else:
                _write_data = 0x11
            self.i2c.writeReg8(self.ak8963, 0x0A, _write_data)
            time.sleep(0.01)
        elif self.MAG_MODE == self.MAG_MODE_SERIAL_1 or self.MAG_MODE == self.MAG_MODE_SERIAL_2:
            status = self.i2c.readReg8(self.ak8963, 0x02)
            if (status & 0x02) == 0x02:
                self.i2c.readReg8(self.ak8963, 0x09)
        elif self.MAG_MODE == self.MAG_MODE_EX_TRIGGER:
            # todo: 実装
            return
        elif self.MAG_MODE == self.MAG_MODE_POWER_DOWN:
            raise Exception('005 Mag sensor power down')
        status = self.i2c.readReg8(self.ak8963, 0x02)
        while (status & 0x01) != 0x01:
            time.sleep(0.01)
            status = self.i2c.readReg8(self.ak8963, 0x02)

        mag_x_high = self.i2c.readReg8(self.ak8963, 0x03)
        mag_x_low = self.i2c.readReg8(self.ak8963, 0x04)
        mag_y_high = self.i2c.readReg8(self.ak8963, 0x05)
        mag_y_low = self.i2c.readReg8(self.ak8963, 0x06)
        mag_z_high = self.i2c.readReg8(self.ak8963, 0x07)
        mag_z_low = self.i2c.readReg8(self.ak8963, 0x08)
        mag_off = self.i2c.readReg8(self.ak8963, 0x09)
        raw_x = self.gyro_coefficient * self.u2s(mag_x_high << 8 | mag_x_low)
        raw_y = self.gyro_coefficient * self.u2s(mag_y_high << 8 | mag_y_low)
        raw_z = self.gyro_coefficient * self.u2s(mag_z_high << 8 | mag_z_low)
        st2 = mag_off

        if (st2 & 0x08) == 0x08:
            raise Exception('006 Mag sensor over flow')

        if self.MAG_BIT == 16:
            raw_x = raw_x * self.mag_coefficient_16
            raw_y = raw_y * self.mag_coefficient_16
            raw_z = raw_z * self.mag_coefficient_16
        else:
            raw_x = raw_x * self.mag_coefficient_14
            raw_y = raw_y * self.mag_coefficient_14
            raw_z = raw_z * self.mag_coefficient_14

        return raw_x, raw_y, raw_z

    # def get_temperature(self):
```



```
    # def get_temperature(self):
    #     data = self.i2c.readReg8()


    def calibration_accel(self, _count=1000):
        print('Accel calibration start')
        _sum = [0, 0, 0]

        for i in range(_count):
            _data = self.get_accel()
            _sum[0] += _data[0]
            _sum[1] += _data[1]
            _sum[2] += _data[2]

        self.offset_accel_x = -1.0 * _sum[0] / _count
        self.offset_accel_y = -1.0 * _sum[1] / _count
        self.offset_accel_z = -1.0 * ((_sum[2] / _count) - 1.0)

        print('Accel calibration complete')
        return self.offset_accel_x, self.offset_accel_y, self.offset_accel_z


    def calibration_gyro(self, _count=1000):
        print('Gyro calibration start')
        _sum = [0, 0, 0]

        for i in range(_count):
            _data = self.get_gyro()
            _sum[0] += _data[0]
            _sum[1] += _data[1]
            _sum[2] += _data[2]

        self.offset_gyro_x = -1.0 * _sum[0] / _count
        self.offset_gyro_y = -1.0 * _sum[1] / _count
        self.offset_gyro_z = -1.0 * _sum[2] / _count

        print('Gyro calibration complete')
        return self.offset_gyro_x, self.offset_gyro_y, self.offset_gyro_z

    # def get_axis(self):
    #     import math
    #     accel = self.get_accel()
    #     mag = self.get_mag()
    #     roll = math.atan2(accel[1], accel[2])
    #     temp = math.sqrt(accel[2] ** 2 + accel[1] ** 2)
    #     pitch = math.atan2(accel[0], temp)
    #     yaw = math.atan2(mag[1], mag[0])
    #     return pitch, yaw, roll


if __name__ == '__main__':
    sensor = MPU9250(0.1, 0x68, 0x0c)
    sensor.reset_register()
    sensor.power_wake_up()
    sensor.set_accel_range(8, True)
    sensor.set_gyro_range(1000, True)
    sensor.set_mag_register('100Hz', '16bit')
    while True:
        now = time.time()
        accel = sensor.get_accel()
        gyro = sensor.get_gyro()
        mag = sensor.get_mag()
        print(mag)
        sleep_time = sensor.sampling_time - (time.time()-now)
        if sleep_time < 0.0:
            continue
        time.sleep(sleep_time)

```