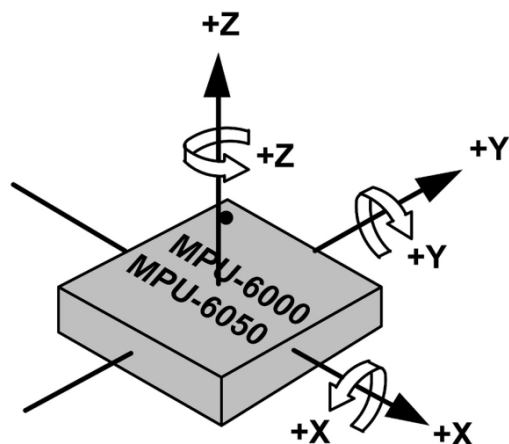
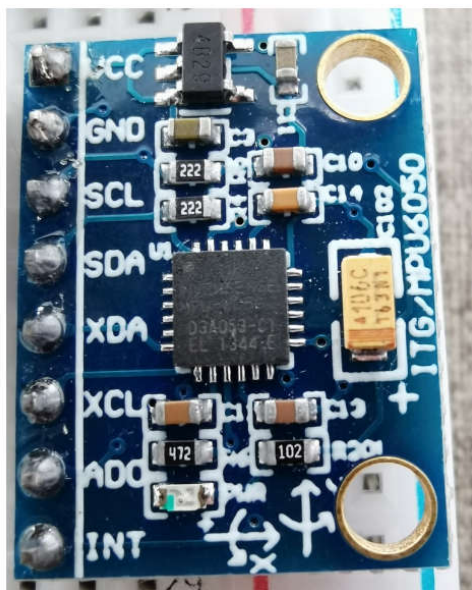


智能系统与控制

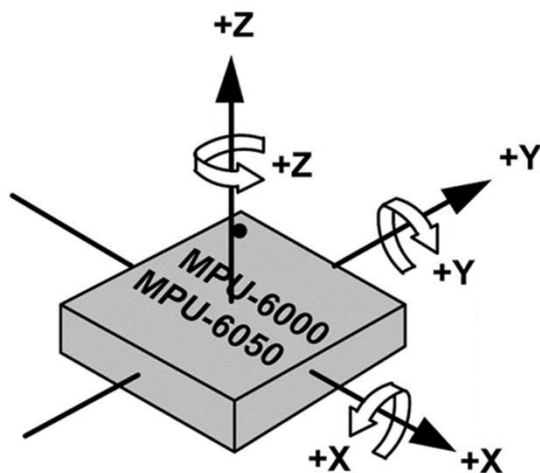
树莓派：GPIO-陀螺仪-MPU6050



于泓
鲁东大学
信息与电气工程学院
2021.11.8

MPU6050

MPU-6050 是全球首款也是唯一一款为智能手表、平板电脑和可穿戴设备提供运动追踪功能的低成本 6 轴运动传感器。它具有低功耗、低成本、高性能等特点被广泛的应用于平衡车、自平稳云台、动作捕捉器等产品的设计中。



获取三个轴的加速度和角速度

MPU6050 分别用三个 **16 位的 ADC** 测量

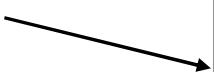
x、y、z 三个方向的**加速度**（加速度计，图中的黑色箭头所示），
绕 x、y、z 三个轴转动的**角速度**（陀螺仪，如图中的白色箭头所示），
将测量的模拟量转换为可输出的数字量后通过 **I2C 接口** 进行输出。

为了实现精确的跟踪和快速和慢速的运动，可以通过编程
控制传感器的测量范围。

陀螺仪的可测范围为 ± 250 、 ± 500 、 ± 1000 、 ± 2000 $^{\circ}/s$ ，

加速度计的可测范围为 ± 2 、 ± 4 、 ± 8 、 $\pm 16g$ （g 表示重力加速度）。

地址68H



```
pi@raspberrypi:~$ 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: -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

$$\text{Sample Rate} = \text{Gyroscope Output Rate} / (1 + \text{SMPLRT_DIV})$$

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

AFS_SEL	Full Scale Range
0	± 2g
1	± 4g
2	± 8g
3	± 16g

	Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		
时钟	19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]								00H	
是否外接 同步信号	1A	26	CONFIG	R/W	-	-	EXT_SYNC_SET[2:0]		DLPF_CFG[2:0]				00H	
	1B	27	GYRO_CONFIG	R/W	-	-	-	FS_SEL [1:0]		-	-	-	量程 选择	
	1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]						
工作方式	6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]				00H
加速度	3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]									
	3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]									
	3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]									
	3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]									
	3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]									
	40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]									
角速度	43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]									
	44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]									
	45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]									
	46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]									
	47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]									
	48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]									

时钟

是否外接
同步信号

工作方式

加
速
度角
速
度

00H

00H

量程
选择

00H

基本驱动代码：

```
import smbus
import math
import time
# MPU6050 I2C 物理地址
MPU6050_ADDR=0x68

# MPU6050 采样率的设置寄存器
MPU6050_SMPLRT_DIV=0x19

# MPU6050配置寄存器用来设置是否需要外置同步帧
# 设置对3轴加速度输出以及3轴陀螺仪输出进行数字低通滤波的方式
MPU6050_CONFIG=0x1a

#设置3轴陀螺仪输出量程的寄存器
MPU6050_GYRO_CONFIG=0x1b

# 设置3轴加速度输出量程的寄存器
MPU6050_ACCEL_CONFIG=0x1c

# 设置MPU6050的工作方式的寄存器 睡眠/循环/ 内部时钟。。。
MPU6050_PWR_MGMT_1=0x6b

# MPU6050 3轴加速计的数据地址
MPU6050_ACCX_DATA=0x3b
MPU6050_ACCY_DATA=0x3d
MPU6050_ACCZ_DATA=0x3f

# MPU6050 3轴陀螺仪的数据地址
MPU6050_GYROX_DATA=0x43
MPU6050_GYROY_DATA=0x45
MPU6050_GYROZ_DATA=0x47
```

2021/11/14

```
class MPU6050(object):
    def __init__(self, address = MPU6050_ADDR, bus = 1,scal_acc=4,scal_gyro=1000):
        self.bus = smbus.SMBus(bus)
        self.address = address

        # 设置工作在默认的8kHz下
        self.bus.write_byte_data(self.address,MPU6050_SMPLRT_DIV,0x00)
        # 不需要外置帧同步、滤波器工作在方式0
        self.bus.write_byte_data(self.address,MPU6050_CONFIG,0x00)
        # 设置陀螺仪的工作量程为 +- 500°/s
        self.bus.write_byte_data(self.address,MPU6050_GYRO_CONFIG,0x08)
        # 设置加速度计的量程为 +- 2g
        self.bus.write_byte_data(self.address,MPU6050_ACCEL_CONFIG,0x00)
        # 设置MPU6050的工作方式为采用内部8k时钟
        self.bus.write_byte_data(self.address,MPU6050_PWR_MGMT_1,0x00)

        self.scal_acc = 65536.0/scal_acc/9.8
        self.scal_gyro = 65536.0/scal_gyro
```

```
def red_word_2c(self,address):
    high = self.bus.read_byte_data(self.address,address)
    low = self.bus.read_byte_data(self.address,address+1)
    val = (high<<8)+low
    if (val>=0x8000):
        return -((65535-val)+1)
    else:
        return val

def get_rawAcc(self):
    rawAccX = self.red_word_2c(MPU6050_ACCX_DATA);
    rawAccY = self.red_word_2c(MPU6050_ACCY_DATA);
    rawAccZ = self.red_word_2c(MPU6050_ACCZ_DATA);
    return rawAccX,rawAccY,rawAccZ

def get_ACC(self):
    rawAccX,rawAccY,rawAccZ = self.get_rawAcc()
    accX = rawAccX/self.scal_acc
    accY = rawAccY/self.scal_acc
    accZ = rawAccZ/self.scal_acc
    return accX,accY,accZ

def get_rawGyro(self):
    rawGyroX = self.red_word_2c(MPU6050_GYROX_DATA);
    rawGyroY = self.red_word_2c(MPU6050_GYROY_DATA);
    rawGyroZ = self.red_word_2c(MPU6050_GYROZ_DATA);
    return rawGyroX,rawGyroY,rawGyroZ

def get_Gyro(self):
    rawGyroX,rawGyroY,rawGyroZ =self.get_rawGyro()
    GyroX = rawGyroX/self.scal_gyro
    GyroY = rawGyroY/self.scal_gyro
    GyroZ = rawGyroZ/self.scal_gyro
    return GyroX,GyroY,GyroZ
```

```
def calc_GyroOffsets(self):
    x = 0
    y = 0
    z = 0
    for i in range(3000):
        rx, ry, rz = self.get_Gyro()
        x = x + rx
        y = y + ry
        z = z + rz

    gyroXoffset = x/3000
    gyroYoffset = y/3000
    gyroZoffset = z/3000

    return gyroXoffset, gyroYoffset, gyroZoffset

def calc_angleAcc(self):
    accX, accY, accZ = self.get_ACC()

    angleAccX = math.degrees(math.atan2(accY, math.sqrt(accZ * accZ + accX * accX)))
    angleAccY = math.degrees(math.atan2(accX, math.sqrt(accZ * accZ + accY * accY)))

    return angleAccX, angleAccY
```

```
if __name__ == "__main__":  
    time.sleep(0.1)  
    print('开始计算Gyro偏置')  
    m_MPU = MPU6050(address = 0x68)  
    GyroOffsets=m_MPU.calc_GyroOffsets()  
    print('Gyro偏置为: \n x_offset = %.2f y_offset = %.2f z_offset = %.2f'%(GyroOffsets[0],GyroOffsets[1],GyroOffsets[2]))  
    try:  
        while True:  
            time.sleep(0.001)  
  
            acc_x,acc_y,acc_z=m_MPU.get_ACC()  
            print('acc_x = %.2f acc_y = %.2f acc_z = %.2f'%(acc_x,acc_y,acc_z))  
  
            gyro_x,gyro_y,gyro_z=m_MPU.get_Gyro()  
  
            print('gyro_x = %.2f gyro_y = %.2f gyro_z = %.2f'%(gyro_x,gyro_y,gyro_z))  
  
            angleAccX,angleAccY= m_MPU.calc_angleAcc()  
  
            print('angleAccX= %.2f,angleAccY=%.2f'%(angleAccX,angleAccY))  
  
    except KeyboardInterrupt:  
        print('\n Ctrl + C QUIT')
```

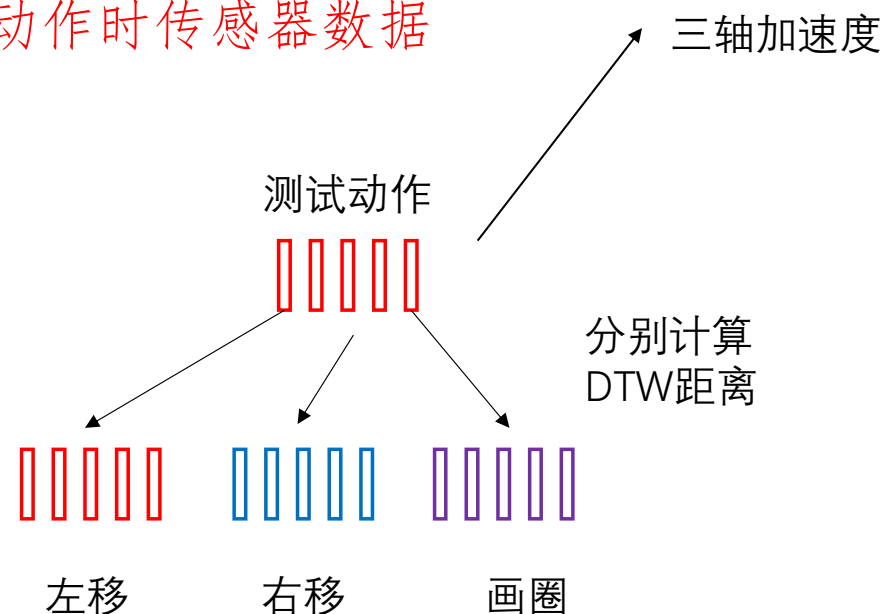
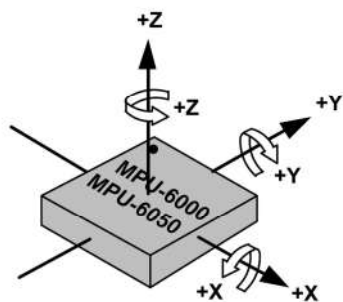
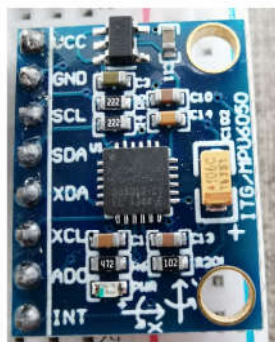

DTW的应用

- 计算两个序列之间的相似性（取dis）

动作识别：

A：测试动作时传感器数据

B：录制动作时传感器数据



```
import numpy as np
import os
import uuid
from MPU6050 import MPU6050
import time
```

```
def MIN(a,b,c):
    return np.min([a,b,c])
```

```
def dis_abs(x, y):
    return abs(x-y)[0]
```

```
def NORM(x):
    return np.math.sqrt(np.dot(x,x))
```

```
def dis_ACC(x,y):
    scale = 0.5
    diff = x-y
    dis = np.dot(x,y)/( NORM(x) * NORM(y) + 1e-8)
```

```
    dis = (1-scale*dis) * NORM(diff)
```

```
    return dis
```

x, y 三维数据
表示3轴的加速度
Size: $N \times 3$

余弦距离 (越相似, 距离越大)

欧式距离 (越相似, 距离越小)

两种距离融合, 注意负号

每隔0.03秒
记录一个数据

```

if __name__ == "__main__":

    m_MPU = MPU6050(address = 0x68)
    data_path = "mov_record"
    try:
        while True:
            print('1 录制 2 测试 3 退出')
            opt = input()

            if opt == '1':
                str_mov = input('请输入动作名称')
                print('5 秒后开始录制动作 %s'%(str_mov))
                time.sleep(5)
                # 录制动作
                mov_data = mov_record(m_MPU)

                # 进行保存
                path = os.path.join(data_path,str_mov)
                os.makedirs(path,exist_ok = True)
                file_name = os.path.join(path,str(uuid.uuid4())+".npy")
                np.save(file_name,np.array(mov_data))
                continue

```

```

def mov_record(MPU):
    data_record = []
    print('请在1.5s内完成动作')
    for i in range(50):
        time.sleep(0.03)
        accs=MPU.get_ACC()
        data_record.append(accs)
    print('动作结束')
    return data_record

```

每隔0.03秒
记录一个数据

/home/pi/EXP-Raspberry/EXP_MPU6050/mov_record/			
Name	Size (KB)	Last modified	Owner
..			
sanjiao		2021-11-12 1...	pi
up		2021-11-12 1...	pi
yuan		2021-11-12 1...	pi

```
elif opt == '2':
    print('5 秒后开始录制测试动作')
    time.sleep(5)
    test_mov = mov_record(m_MPU)

    scores = []
    files, labs = get_mov_list(data_path)

    if len(files) < 1:
        print('没有找到存储动作，请先进行动作录制')
        continue

    # 计算捕捉的动作与存储动作的距离
    for file in files:
        # 加载动作数据
        data_train = np.load(file)
        # 计算测试动作与存储动作之间的距离
        score, _, _ = estimate_dtw(np.array(test_mov), data_train)
        scores.append(score)

    # 找到最小值的距离
    print(scores)
    index = np.where(scores == np.min(scores))[0][0]
    print('测试动作为 ' + labs[index])

    continue

elif opt == '3':
    break

except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')
```

```
def get_mov_list(path):
    list_files = []
    list_labs = []

    if os.path.exists(path):

        # 遍历文件夹，找到.npy文件
        for root, ds, fs in os.walk(path):
            for f in fs:
                if f.endswith('.npy'):
                    fullname = os.path.join(root, f)
                    list_files.append(fullname)

            for file in list_files:
                lab = os.path.split(os.path.split(file)[0])[-1]
                list_labs.append(lab)

    else:
        print('this path not exist')

    return list_files, list_labs
```