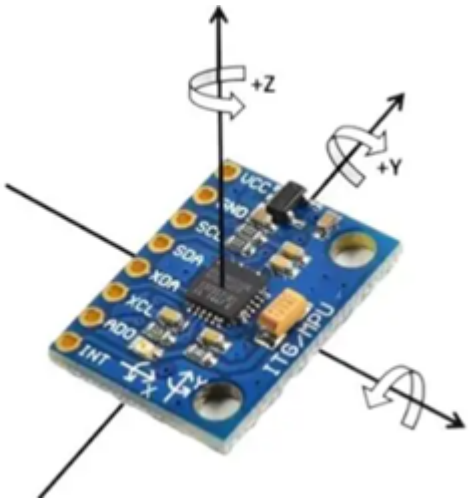
1 关于MPU6050数据采集

1.1 硬件部分



MPU6050的管脚：

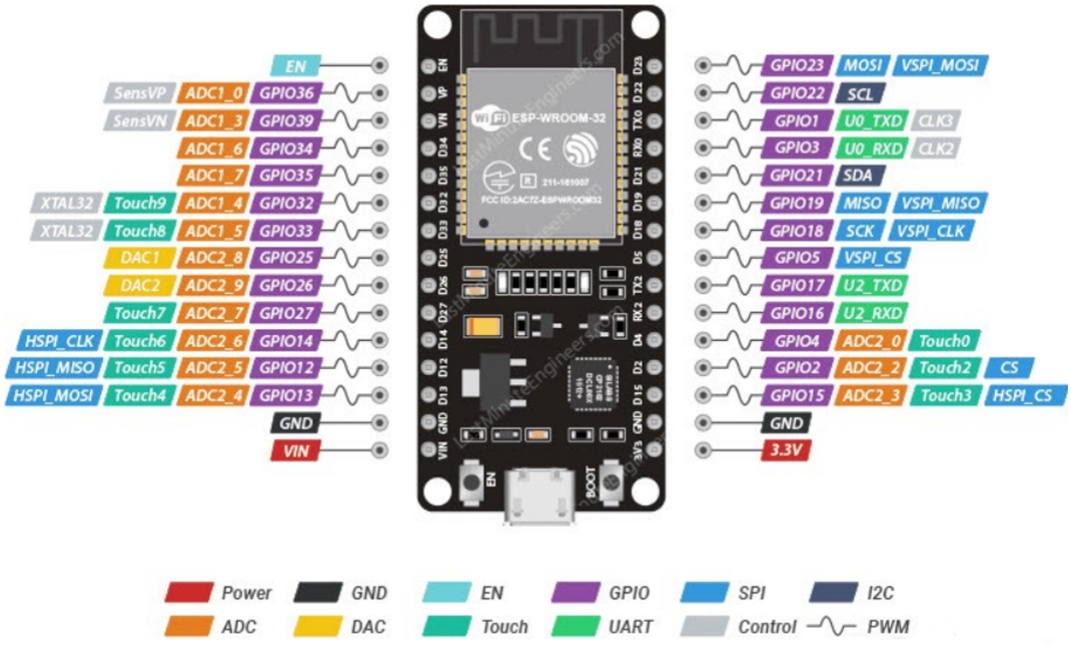
VCC：3.3V供电

GND：接地

I2C通信，串行通讯总线，

- 有串行数据线SDA，串行时钟线SCL
- 所有为开漏输出(OD)
- 总线上所有设备通过软件寻址有唯一地址
- 只存在主从关系，由数据传输方向决定

IIC在SCL处于高时拉低SDA为开始信号，SCL处于高时拉高SDA为结束信号；传输数据时SDA在SCL为低电平时改变数据，在SCL高电平时保持数据，每个SCL脉冲的高电平传递一位数据



esp32只有两个管脚可用于I2C通信，直接连上就行。

1.2 代码

依旧头文件

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
```

都是预下载好的库

用这个库可以调用MPU6050内置的DMP姿态解算单元，可以大大缩减esp32的MCU的运算负担

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

MPU6050 mpu;
uint8_t fifoBuffer[64];//内部缓存
Quaternion q;//四元数
VectorFloat gravity;//重力分量
VectorInt16 acc_raw;//三轴原始加速度
VectorInt16 lineacc;//去除重力的线性加速度
VectorInt16 worldacc;//世界坐标系下的
float ypr[3];//三轴角
float acc[3];//最终加速度
void setup()
{
  Wire.begin();
  Wire.setClock(400000);//I2C通信速率
  Serial.begin(115200);//波特率
  mpu.initialize();
  mpu.dmpInitialize();
  mpu.CalibrateAccel(6);
  mpu.CalibrateGyro(6);
  mpu.PrintActiveOffsets();
```

```
  mpu.setDMPEnabled(true);
}

void loop()
{
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))//读缓存
  {
    mpu.dmpGetQuaternion(&q,fifoBuffer);//读四元数
    mpu.dmpGetGravity(&gravity,&q);//读重力分量
    mpu.dmpGetAccel(&acc_raw,fifoBuffer);//读原始加速度
    acc_raw.x=acc_raw.x/2;
    acc_raw.y=acc_raw.y/2;
    acc_raw.z=acc_raw.z/2;
    mpu.dmpGetLinearAccel(&lineacc,&acc_raw,&gravity);//去除重力
    mpu.dmpGetYawPitchRoll(ypr,&q,&gravity);
    mpu.dmpConvertToWorldFrame(&worldacc,&lineacc,&q);//转世界坐标系
    acc[0]=worldacc.x/8192.0f;//归一化
    acc[1]=worldacc.y/8192.0f;
    acc[2]=worldacc.z/8192.0f;
    //Serial.print("yaw:");
    //Serial.println(ypr[0] * 180 / M_PI);
    //Serial.print(",pitch:");
    //Serial.println(ypr[1] * 180 / M_PI);
    //Serial.print(",roll:");
    //Serial.println(ypr[2] * 180 / M_PI);
    Serial.print("grax:");
    Serial.println(gravity.x);
    Serial.print(",gray:");
    Serial.println(gravity.y);
    Serial.print(",graz:");
    Serial.println(gravity.z);
    Serial.print("x:");
    Serial.println(acc[0]);
    Serial.print(",y:");
    Serial.println(acc[1]);
    Serial.print(",z:");
    Serial.println(acc[2]);
  }
}
```

中间有一段要除以二，是因为在其原始函数中，去除重力分量的定义是这样的：

```
uint8_t MPU6050_6Axis_MotionApps20::dmpGetLinearAccel(VectorInt16 *v, VectorInt16
*vRaw, VectorFloat *gravity) {
    // get rid of the gravity component (+1g = +8192 in standard DMP FIFO packet,
sensitivity is 2g)
    v -> x = vRaw -> x - gravity -> x*8192;
    v -> y = vRaw -> y - gravity -> y*8192;
    v -> z = vRaw -> z - gravity -> z*8192;
    return 0;
}
```

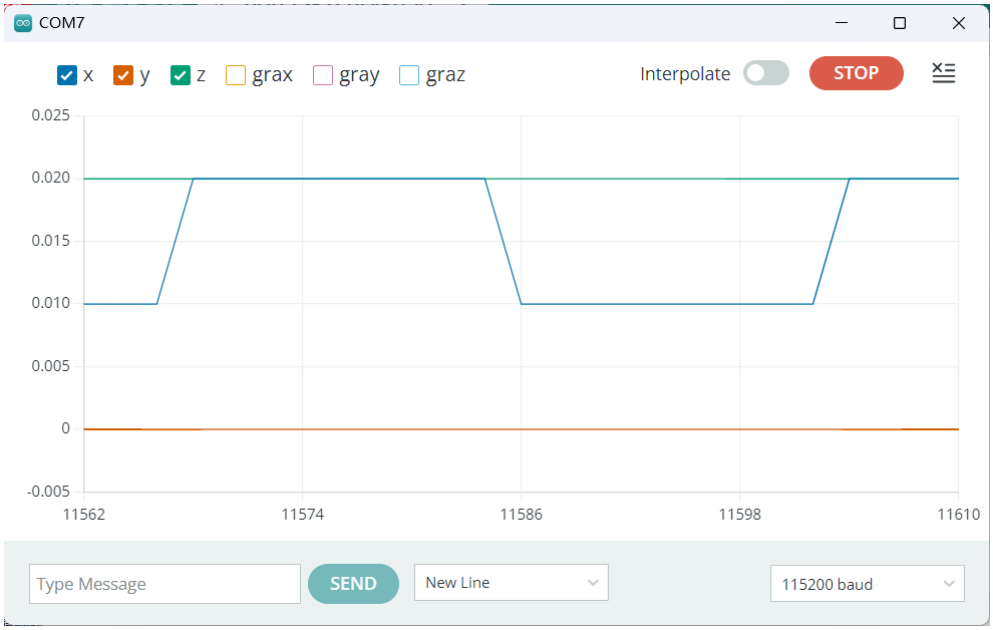
所有加速度得到的初始值单位是LSB，也就是数模转换的灵敏度，这个取决量程范围，对±2g，灵敏度为16384 LSB/g，对±4g，灵敏度为8192 LSB/g。

但是上述函数的原理，是将重力分量乘8192再减的，但是默认量程为2g，这就导致了灵敏度换算出现差异，于是原始灵敏度要先除以二再进行运算。

最终得到的结果有非常严重的零点漂移，理论上静止时三轴加速度都为0，但是实际上均不为0，而且随着姿态不同，偏移量也有变化，最大甚至0.3g。

（这个神奇的零偏现象在第二天的中午我再次插上USB后神奇的消失了，但是当我再次烧录进相同的代码时零飘又出现了，我本以为是长时间通电引起的温度升高导致的差异，结果我再也没有触发过这个情况，我们曾短暂的到达过未来）

重力分量是通过四元数运算得到的，三轴角是通过重力分量和四元数得到的，三轴加速度是直接从缓存中读出来的，去除重力的函数逻辑很简单就是直接减。可能这导致了静止时重力分量于三轴加速度不相等，而且这个差值和静止的姿态有关，和初始化时传感器的姿态也有关，且只有z轴会飘，初步判断与yaw角不准有很大关系。但是DMP的算法都是内部封装的，只能适应。



1.3 对抗零飘

z轴的加速度很重要，因为z轴积分代表了运动轨迹的上下，假如静止时偏移量为向上的0.3g，而你运动的实际加速度向下且小于0.3g，此时记录下来的加速度变化数据方向全部向上，这样就会把实际上向下的运动轨迹解析为向上的。

1.3.1 偏置项

记录开始时的一瞬间状态是静止的，可以将此时的z轴的值视为偏置项，在整个记录过程中减去这个偏置项。

这种解决方法的缺点在于当你运动时姿态是会变化的，这就导致零偏也会变，偏置项可能并不完全等于应该减去的偏移量。优点是基本上能够解决方向问题，方向不至于解析错掉，最多就积分出来短一点。

1.3.2 不用DMP库

Adafruit的MPU6050库读取出来的三轴加速度相当准确，可以手算重力的三轴分量来获得去除重力分量的三轴加速度，这需要三轴角yaw、pitch、roll角，这些角该怎么获得呢，用DMP肯定是不合适的，一方面DMP需要初始化MPU，Adafruit库也需要初始化MPU，会冲突；令一方面DMP库的三轴角就是通过重力分量解算出来的，再用他算回去重力分量没意义。

于是找到了Madgwick库，只需上传三轴加速度和三轴角速度，其运用互补滤波算法便可以求出三轴角。代码如下：

```
#include "mpu6050.h"
#include "MadgwickAHRS.h"
Madgwick filter;
float roll,pitch,yaw;
unsigned long lastTime=0;

void setup(){
  Serial.begin(115200);
  Init_mpu6050();
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  filter.begin(100);
  lastTime=micros();
}

void loop() {
  ReadMPU6050();

  unsigned long now=micros();
  float dt=(now-lastTime)/1000000.0f;
  lastTime=now;

  float gx=mpu6050_data.Angle_Velocity_R;
  float gy=mpu6050_data.Angle_Velocity_P;
  float gz=mpu6050_data.Angle_Velocity_Y;
  filter.updateIMU(
    gx,gy,gz,
    mpu6050_data.Acc_X,mpu6050_data.Acc_Y,mpu6050_data.Acc_Z
  );
  roll=filter.getRoll();
  pitch=filter.getPitch();
  yaw=filter.getYaw();

  Serial.print("yaw:");
  Serial.print(yaw);
  Serial.print(",");
  Serial.print("pitch:");
  Serial.print(pitch);
  Serial.print(",");
  Serial.print("roll:");
  Serial.println(roll);

  //Serial.print("Acc_x:");
  //Serial.print(mpu6050_data.Acc_X);
  //Serial.print(",");
  //Serial.print("Acc_Y:");
  //Serial.print(mpu6050_data.Acc_Y);
  //Serial.print(",");
  //Serial.print("Acc_Z:");
  //Serial.println(mpu6050_data.Acc_Z);

  //Serial.print("Angle_velocity_R:");
  //Serial.println(mpu6050_data.Angle_Velocity_R);
  //Serial.print(",");
  //Serial.print("Angle_velocity_P:");
  //Serial.print(mpu6050_data.Angle_Velocity_P);
  //Serial.print(",");
```

```
  //Serial.print("Angle_velocity_Y:");
  //Serial.println(mpu6050_data.Angle_Velocity_Y);
  delay(10);
}
```

结果当然是出问题了，不仅响应速度慢，而且非常不准，如果说DMP的结果只是有一点偏差，那Madgwick库就是完全扯淡（也可能是我操作的问题）

```
yaw:178.43,pitch:-1.53,roll:108.95
yaw:178.43,pitch:-1.54,roll:108.87
yaw:178.43,pitch:-1.65,roll:108.83
yaw:178.43,pitch:-1.54,roll:108.80
yaw:178.43,pitch:-1.52,roll:108.88
yaw:178.43,pitch:-1.50,roll:108.80
yaw:178.43,pitch:-1.51,roll:108.72
yaw:178.43,pitch:-1.61,roll:108.76
yaw:178.43,pitch:-1.62,roll:108.84
yaw:178.43,pitch:-1.50,roll:108.84
yaw:178.43,pitch:-1.57,roll:108.
```

总之第二种方法在此夭折。

1.4 数据采集逻辑

1.4.1 按键

双端按键，一端接地，一端接GPIO13，因为GPIO13有内置上拉电阻，如果接没有上拉电阻的管脚，平时浮空输入电平会高低不定。输入设置为

```
  pinMode(buttonPin,INPUT_PULLUP);
```

按钮默认常断，因为3.3V的给传感器供电了，剩下的那个5V会烧芯片，因此接地。所以不按按钮时为HIGH，按下后为LOW

1.4.2 数据采集

定义三个数组分别存储xyz轴加速度

基本逻辑为：

1. 按下按钮开始记录，每次循环中将本次数据记录到数组中；在首次按下按钮时初始化一系列变量：包括下标，时间，flag2(开始计时标志)，偏置项
2. 松开按钮或者时间计时为2s后停止记录并进行积分运算得到轨迹。同时更新flag1，flag2关闭记录通道和积分运算通道。

其中为了解决当计时2s已到但是按钮未松开导使得一直在记录导致的数组溢出，定义flag1作为进入记录的标志；其中当按钮为松开状态时flag1为0，即只有按钮松开后才能够再次按下开始记录

为了避免平时即使没有记录也进入到积分环节，定义flag2，开始计时时才开始，积分结束后再关闭。

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

MPU6050 mpu;
uint8_t fifoBuffer[64];//缓存
Quaternion q;//四元数
VectorFloat gravity;//重力分量
```

```
VectorInt16 acc_raw;//原始加速度分量
VectorInt16 lineacc;//去除重力的加速度分量
VectorInt16 worldacc;//世界坐标系下去除重力的加速度分量
float ypr[3];//三轴角
float acc[3];//加速度
const int buttonPin=13;//按钮GPIO13
int buttonstate=0;//按钮状态
int lastbuttonstate=0;//上一次按钮状态
float z_offset=0;//偏执
int i=0;//下标
float x_acc[50];
float y_acc[50];
float z_acc[50];
unsigned long starttime=0;//开始时间
int flag1=0;//为了标志2s已到但是按钮未松开的情况的flag
int flag2=1;//为了标志开始计时的flag

void setup()
{
  pinMode(buttonPin,INPUT_PULLUP);
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(115200);
  mpu.initialize();
  mpu.dmpInitialize();
  mpu.CalibrateAccel(6);
  mpu.CalibrateGyro(6);
  mpu.PrintActiveOffsets();
  mpu.setDMPEnabled(true);
}

void loop()
{
  buttonstate=digitalRead(buttonPin);
  //获得世界坐标系下去除重力的加速度分量
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
  {
    mpu.dmpGetQuaternion(&q,fifoBuffer);
    mpu.dmpGetGravity(&gravity,&q);
    mpu.dmpGetAccel(&acc_raw,fifoBuffer);
    acc_raw.x=acc_raw.x/2;
    acc_raw.y=acc_raw.y/2;
    acc_raw.z=acc_raw.z/2;
    mpu.dmpGetLinearAccel(&lineacc,&acc_raw,&gravity);
    mpu.dmpGetYawPitchRoll(ypr,&q,&gravity);
    mpu.dmpConvertToWorldFrame(&worldacc,&lineacc,&q);
    acc[0]=worldacc.x/8192.0f;
    acc[1]=worldacc.y/8192.0f;
    acc[2]=worldacc.z/8192.0f;
    //Serial.print("yaw:");
    //Serial.println(ypr[0] * 180 / M_PI);
    //Serial.print(",pitch:");
    //Serial.println(ypr[1] * 180 / M_PI);
    //Serial.print(",roll:");
    //Serial.println(ypr[2] * 180 / M_PI);
  }
```

```
  Serial.print(flag1);
  if(buttonstate==LOW&&flag1==0){//如果按下按钮
    if(lastbuttonstate==HIGH){//首次按下按钮
      z_offset=acc[2];//偏置项更新
      i=0;//下标更新
      starttime=millis();//起始时间更新
      flag2=0;//开始计时标志
      Serial.print("首次按下");
    }
    x_acc[i]=acc[0]*9.8;//更新加速度
    y_acc[i]=acc[1]*9.8;
    z_acc[i]=(acc[2]-z_offset)*9.8;
    i++;
  }

  if((millis()-starttime>=2000||
(lastbuttonstate==LOW&&buttonstate==HIGH))&&flag2==0){
    flag1=1;//关闭记录通道
    flag2=1;//关闭积分运算通道
    //积分运算
    Serial.print("进入积分");
  }
  if(buttonstate==HIGH){flag1=0;}//如果按钮是松开的允许再按记录，如果是按到时间到了结束，
不触发flag1变化，不进记录
  lastbuttonstate=buttonstate;//记录上一次按钮状态
  delay(50);
}
```

)000首次按下-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010-0.010进入积分00000

)0.0400.0400.0400.0400.0400.0400.0400.0400.0400.0400.0400.0400.04进入积分111111111111111

效果非常ok，能够实现两种结束记录的方式。

1.4.3 积分环节

因为对运动轨迹的判断只需要相对轨迹，而非绝对位置的轨迹，因此可以令初始xyz坐标为0，初始xyz速度为0

每次循环的时间间隔，用micros()函数大致求了一下，267μs大约，非常小，因此可以直接每次循环delay(50)，这样可以近似看作每次循环间隔为50ms。

这样理论上2s内最多可以记录40个数据，实际跑一下，其实是35个，也差不多，为以防万一，可以定义数组长度为50

)000首次按下10203040506070809010011012013014015016017018019020021022023024025026027028029030031032033034035进入积分111

因为得到的数据是一个数组，那么其积分我们可以看作对一个阶梯函数求积分，积分的第i项就是对原数组图象前i项的面积和，本质上就是求数组的累加和，再乘上每次循环间隔的时间。

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

MPU6050 mpu;
```

```
uint8_t fifoBuffer[64];//缓存
Quaternion q;//四元数
VectorFloat gravity;//重力分量
VectorInt16 acc_raw;//原始加速度分量
VectorInt16 lineacc;//去除重力的加速度分量
VectorInt16 worldacc;//世界坐标系下去除重力的加速度分量
float ypr[3];//三轴角
float acc[3];//加速度
const int buttonPin=13;//按钮GPIO13
int buttonstate=0;//按钮状态
int lastbuttonstate=0;//上一次按钮状态
float z_offset=0;//偏执
int i=0;//下标
float x_acc[50];//加速度
float y_acc[50];
float z_acc[50];
float x_v[50];//速度
float y_v[50];
float z_v[50];
float x_x[50];//位移
float y_x[50];
float z_x[50];
int k;//工具变量

float sumwhat(int i,float tool[50]){//辅助函数求tool数组的前i项和
  int j;
  float sum=0.0;
  for (j=0;j<=i;j++){
    sum+=tool[j]*0.05;
  }
  return sum;
}

unsigned long starttime=0;//开始时间
int flag1=0;//为了标志2s已到但是按钮未松开的情况的flag
int flag2=1;//为了标志开始计时的flag

void setup()
{
  pinMode(buttonPin,INPUT_PULLUP);
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(115200);
  mpu.initialize();
  mpu.dmpInitialize();
  mpu.CalibrateAccel(6);
  mpu.CalibrateGyro(6);
  mpu.PrintActiveOffsets();
  mpu.setDMPEnabled(true);
}

void loop()
{
  buttonstate=digitalRead(buttonPin);
  //获得世界坐标系下去除重力的加速度分量
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
```

```
  {
    mpu.dmpGetQuaternion(&q,fifoBuffer);
    mpu.dmpGetGravity(&gravity,&q);
    mpu.dmpGetAccel(&acc_raw,fifoBuffer);
    acc_raw.x=acc_raw.x/2;
    acc_raw.y=acc_raw.y/2;
    acc_raw.z=acc_raw.z/2;
    mpu.dmpGetLinearAccel(&lineacc,&acc_raw,&gravity);
    mpu.dmpGetYawPitchRoll(ypr,&q,&gravity);
    mpu.dmpConvertToWorldFrame(&worldacc,&lineacc,&q);
    acc[0]=worldacc.x/8192.0f;
    acc[1]=worldacc.y/8192.0f;
    acc[2]=worldacc.z/8192.0f;
    //Serial.print("yaw:");
    //Serial.println(ypr[0] * 180 / M_PI);
    //Serial.print(",pitch:");
    //Serial.println(ypr[1] * 180 / M_PI);
    //Serial.print(",roll:");
    //Serial.println(ypr[2] * 180 / M_PI);
  }
  if(buttonstate==LOW&&flag1==0){//如果按下按钮
    if(lastbuttonstate==HIGH){//首次按下按钮
      z_offset=acc[2];//偏置项更新
      i=0;//下标更新
      starttime=millis();//起始时间更新
      flag2=0;//开始计时标志
      Serial.print("首次按下");
    }
    x_acc[i]=acc[0]*9.8;//更新加速度
    y_acc[i]=acc[1]*9.8;
    z_acc[i]=(acc[2]-z_offset)*9.8;
    i++;
  }

  if((millis()-starttime>=2000||
(lastbuttonstate==LOW&&buttonstate==HIGH))&&flag2==0){
    flag1=1;//关闭记录通道
    flag2=1;//关闭积分运算通道
    //积分运算
    Serial.print("进入积分");
    for (k=0;k<i;k++){
      x_v[k]=sumwhat(k,x_acc);
      y_v[k]=sumwhat(k,y_acc);
      z_v[k]=sumwhat(k,z_acc);
    }
    for (k=0;k<i;k++){
      x_x[k]=sumwhat(k,x_v);
      y_x[k]=sumwhat(k,y_v);
      z_x[k]=sumwhat(k,z_v);
    }
    for (k=0;k<i;k++){
      Serial.print("x:");
      Serial.print(x_x[k]);
      Serial.print(",y:");
      Serial.print(y_x[k]);
      Serial.print(",z:");
```

```
      Serial.println(z_x[k]);
      }
  }
  if(buttonstate==HIGH){flag1=0;}//如果按钮是松开的允许再按记录，如果是按到时间到了结束，
不触发flag1变化，不进记录
  lastbuttonstate=buttonstate;//记录上一次按钮状态
  delay(50);
}
```

由此计算出来的运动轨迹有：

```
x:0.36,y:-0.03,z:-0.01  x:0.00,y:0.32,z:0.00   x:-0.05,y:0.36,z:-0.09
x:0.42,y:-0.04,z:-0.01  x:0.01,y:0.37,z:-0.00  x:-0.04,y:0.34,z:-0.14
x:0.49,y:-0.05,z:-0.02  x:0.01,y:0.41,z:-0.00  x:-0.02,y:0.30,z:-0.19
x:0.55,y:-0.05,z:-0.03  x:0.02,y:0.46,z:-0.01  x:0.01,y:0.24,z:-0.25
x:0.62,y:-0.05,z:-0.03  x:0.03,y:0.51,z:-0.01  x:0.04,y:0.16,z:-0.32
x:0.69,y:-0.05,z:-0.04  x:0.04,y:0.57,z:-0.01  x:0.08,y:0.06,z:-0.40
```

偶尔能够成功，但是仍然存在很大问题：就是他妈的世界坐标系不准，就是可能我向左挥，加速度变得不是x而是y，向上挥可能不止有z变化，还有xy的事。后续整体的算法思路都没有问题，有问题全出在一开始的三轴加速度采集上，DMP就是构式。

后续再修。

1.5 后续工作

修好工作轨迹，先用简单的上下左右轨迹，做轨迹分类，然后研究该怎么部署模型。

三轴加速度采集竟然是最难的，算法简单但是受限于库和硬件的抽象问题，导致出现很多非算法的问题，这种问题是最难调的。

2 新的方案

之前直接调用DMP库得到的结果有目共睹，零漂非常严重，而且姿态不同漂移的量还不同，导致不能用简单的偏置项来处理。因此我们决定换算法。

2.1 硬件部分

依旧不变

2.2 初始量获取

其实用DMP库时，初始获得的三轴加速度与三轴角速度就感觉别扭，得到的数据无论是响应速度还是精度还是数据的噪声，都不如Adafruit库的，究其原因是因为adafruit库是直接从传感器里面读出来的，而DMP是计算出来的，会误差累积。

因此我们依旧用Adafruit库获得最原始的三轴加速度与三轴角速度，只要依靠这些值获得三个欧拉角，便可构造旋转矩阵，从而从传感器坐标系转到世界坐标系。

2.3 欧拉角

2.3.1 互补滤波算法

在静止状态下仅靠重力加速度在三轴分量可以解算出roll和pitch角

假设 a_x a_y a_z 为三轴重力分量，那么可以认为是向量[0, 0 ,g]经过三轴旋转而得，因此满足：

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \mathrm{M}_x \cdot \mathrm{M}_y \cdot \mathrm{M}_z \cdot \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos r & \sin r \\ 0 & -\sin r & \cos r \end{bmatrix} \cdot \begin{bmatrix} \cos p & 0 & -\sin p \\ 0 & 1 & 0 \\ \sin p & 0 & \cos p \end{bmatrix} \cdot \begin{bmatrix} \cos y & \sin y & 0 \\ -\sin y & \cos y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$= \begin{bmatrix} \cos p \cdot \cos y & \cos p \cdot \sin y & -\sin p \\ \cos y \cdot \sin p \cdot \sin r - \cos r \cdot \sin y & \cos r \cdot \cos y + \sin p \cdot \sin r \cdot \sin y & \cos p \cdot \sin r \\ \sin r \cdot \sin y + \cos r \cdot \cos y \cdot \sin p & \cos r \cdot \sin p \cdot \sin y - \cos y \cdot \sin r & \cos p \cdot \cos r \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$= \begin{bmatrix} -\sin p \\ \cos p \cdot \sin r \\ \cos p \cdot \cos r \end{bmatrix} \cdot g$$

$$\begin{cases} roll = \arctan(\dfrac{a_y}{a_z}) \\ pitch = -\arctan(\dfrac{a_x}{\sqrt{{a_y}^2 + {a_z}^2}}) \end{cases}$$

因为在静止时精度好，在运动时不太适用，因此还需要互补滤波算法，这个作为一部分，另一部分来自于角速度积分，角速度积分会累积误差。

```
angle=α*(angle+gyro*dt)+(1-α)*accel
```

基本原理如上。

但是有一个致命缺点是，yaw角只能靠角速度积分获得，不能参与互补滤波，因此这个算法否定

2.3.2 Madgwick

这个算法是参考的github上 https://github.com/kriswiner/MPU6050 这位老哥的代码复现的，主要参考了其Madgwick四元数滤波器的算法，原理我是真的看不懂。

mpu6050.h

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;

struct MPU6050_DATA
{
  // 摄氏度
  float Temperature = 0.0;
```

```
  // 沿三轴的线加速度 米每二次方秒 m/s^2
  float Acc_X = 0.0;
  float Acc_Y = 0.0;
  float Acc_Z = 0.0;

  // 三轴角速度 弧度每秒 rad/s
  float Angle_Velocity_R = 0.0;
  float Angle_Velocity_P = 0.0;
  float Angle_Velocity_Y = 0.0;

}mpu6050_data;

void Init_mpu6050()
{
  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
  /*********************************************
    Available AccelerometerRange:
      MPU6050_RANGE_2_G
      MPU6050_RANGE_4_G
      MPU6050_RANGE_8_G
      MPU6050_RANGE_16_G

    API:
      getAccelerometerRange()
      setAccelerometerRange()
  *********************************************/
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  /*********************************************
    Available GyroRange:
      MPU6050_RANGE_250_DEG
      MPU6050_RANGE_500_DEG
      MPU6050_RANGE_1000_DEG
      MPU6050_RANGE_2000_DEG

    API:
      getGyroRange()
      setGyroRange()
  *********************************************/
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  /*********************************************
    Available Bandwidth:
      MPU6050_BAND_5_HZ
      MPU6050_BAND_10_HZ
      MPU6050_BAND_21_HZ
      MPU6050_BAND_44_HZ
      MPU6050_BAND_94_HZ
      MPU6050_BAND_184_HZ
```

```
      MPU6050_BAND_260_HZ

    API:
      getFilterBandwidth()
      setFilterBandwidth()
  ****************************************************/
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}


void ReadMPU6050()
{
  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  mpu6050_data.Temperature = temp.temperature;

  mpu6050_data.Acc_X = a.acceleration.x;
  mpu6050_data.Acc_Y = a.acceleration.y;
  mpu6050_data.Acc_Z = a.acceleration.z;

  mpu6050_data.Angle_Velocity_R = g.gyro.x;
  mpu6050_data.Angle_Velocity_P = g.gyro.y;
  mpu6050_data.Angle_Velocity_Y = g.gyro.z;
}
```

mozhang.ino

```
#include "mpu6050.h"
#include <math.h>
#define PI 3.1415926

uint32_t delt_t = 0; // used to control display output rate
uint32_t count = 0;  // used to control display output rate

float GyroMeasError = PI * (40.0f / 180.0f);     // gyroscope measurement error
in rads/s (start at 60 deg/s), then reduce after ~10 s to 3
float beta = sqrt(3.0f / 4.0f) * GyroMeasError;  // compute beta
float GyroMeasDrift = PI * (2.0f / 180.0f);      // gyroscope measurement drift
in rad/s/s (start at 0.0 deg/s/s)
float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;  // compute zeta, the other free
parameter in the Madgwick scheme usually set to a small or zero value
float deltat = 0.0f;                              // integration interval for
both filter schemes
uint32_t lastUpdate = 0, firstUpdate = 0;         // used to calculate
integration interval
uint32_t Now = 0;
float q[4] = {1.0f, 0.0f, 0.0f, 0.0f}; //四元数

float pitch, yaw, roll;//三轴角
float worldAccX,worldAccY,worldAccZ;//世界坐标系下三轴加速度
```

```
void MadgwickQuaternionUpdate(float ax, float ay, float az, float gyrox, float
gyroy, float gyroz)
{
    float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];         // short name local
variable for readability
    float norm;                                               // vector norm
    float f1, f2, f3;                                         // objetive
funcyion elements
    float J_11or24, J_12or23, J_13or22, J_14or21, J_32, J_33; // objective
function Jacobian elements
    float qDot1, qDot2, qDot3, qDot4;
    float hatDot1, hatDot2, hatDot3, hatDot4;
    float gerrx, gerry, gerrz, gbiasx, gbiasy, gbiasz;        // gyro bias error

    // Auxiliary variables to avoid repeated arithmetic
    float _halfq1 = 0.5f * q1;
    float _halfq2 = 0.5f * q2;
    float _halfq3 = 0.5f * q3;
    float _halfq4 = 0.5f * q4;
    float _2q1 = 2.0f * q1;
    float _2q2 = 2.0f * q2;
    float _2q3 = 2.0f * q3;
    float _2q4 = 2.0f * q4;
    float _2q1q3 = 2.0f * q1 * q3;
    float _2q3q4 = 2.0f * q3 * q4;

    // Normalise accelerometer measurement
    norm = sqrt(ax * ax + ay * ay + az * az);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f/norm;
    ax *= norm;
    ay *= norm;
    az *= norm;

    // Compute the objective function and Jacobian
    f1 = _2q2 * q4 - _2q1 * q3 - ax;
    f2 = _2q1 * q2 + _2q3 * q4 - ay;
    f3 = 1.0f - _2q2 * q2 - _2q3 * q3 - az;
    J_11or24 = _2q3;
    J_12or23 = _2q4;
    J_13or22 = _2q1;
    J_14or21 = _2q2;
    J_32 = 2.0f * J_14or21;
    J_33 = 2.0f * J_11or24;

    // Compute the gradient (matrix multiplication)
    hatDot1 = J_14or21 * f2 - J_11or24 * f1;
    hatDot2 = J_12or23 * f1 + J_13or22 * f2 - J_32 * f3;
    hatDot3 = J_12or23 * f2 - J_33 *f3 - J_13or22 * f1;
    hatDot4 = J_14or21 * f1 + J_11or24 * f2;

    // Normalize the gradient
    norm = sqrt(hatDot1 * hatDot1 + hatDot2 * hatDot2 + hatDot3 * hatDot3 +
hatDot4 * hatDot4);
    hatDot1 /= norm;
    hatDot2 /= norm;
```

```
    hatDot3 /= norm;
    hatDot4 /= norm;

    // Compute estimated gyroscope biases
    gerrx = _2q1 * hatDot2 - _2q2 * hatDot1 - _2q3 * hatDot4 + _2q4 * hatDot3;
    gerry = _2q1 * hatDot3 + _2q2 * hatDot4 - _2q3 * hatDot1 - _2q4 * hatDot2;
    gerrz = _2q1 * hatDot4 - _2q2 * hatDot3 + _2q3 * hatDot2 - _2q4 * hatDot1;

    // Compute and remove gyroscope biases
    gbiasx += gerrx * deltat * zeta;
    gbiasy += gerry * deltat * zeta;
    gbiasz += gerrz * deltat * zeta;
    gyrox -= gbiasx;
    gyroy -= gbiasy;
    gyroz -= gbiasz;

    // Compute the quaternion derivative
    qDot1 = -_halfq2 * gyrox - _halfq3 * gyroy - _halfq4 * gyroz;
    qDot2 =  _halfq1 * gyrox + _halfq3 * gyroz - _halfq4 * gyroy;
    qDot3 =  _halfq1 * gyroy - _halfq2 * gyroz + _halfq4 * gyrox;
    qDot4 =  _halfq1 * gyroz + _halfq2 * gyroy - _halfq3 * gyrox;

    // Compute then integrate estimated quaternion derivative
    q1 += (qDot1 -(beta * hatDot1)) * deltat;
    q2 += (qDot2 -(beta * hatDot2)) * deltat;
    q3 += (qDot3 -(beta * hatDot3)) * deltat;
    q4 += (qDot4 -(beta * hatDot4)) * deltat;

    // Normalize the quaternion
    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise
quaternion
    norm = 1.0f/norm;
    q[0] = q1 * norm;
    q[1] = q2 * norm;
    q[2] = q3 * norm;
    q[3] = q4 * norm;
}

void setup() {
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(115200);
  Init_mpu6050();
  pinMode(buttonPin,INPUT_PULLUP);
}

void loop() {
  buttonstate=digitalRead(buttonPin);
  ReadMPU6050();
  //可视化线加速度曲线
  //Serial.print("Acc_x:");
  //Serial.print(mpu6050_data.Acc_X);
  //Serial.print(",");
  //Serial.print("Acc_Y:");
  //Srial.print(mpu6050_data.Acc_Y);
  //Serial.print(",");
```

```
  //Serial.print("Acc_Z:");
  //Serial.println(mpu6050_data.Acc_Z);

  //可视化角速度曲线
  //Serial.print("Angle_velocity_R:");
  //Serial.println(mpu6050_data.Angle_Velocity_R);
  //Serial.print(",");
  //Serial.print("Angle_velocity_P:");
  //Serial.print(mpu6050_data.Angle_Velocity_P);
  //Serial.print(",");
  //Serial.print("Angle_velocity_Y:");
  //Serial.println(mpu6050_data.Angle_Velocity_Y);

  Now = micros();
  deltat = ((Now - lastUpdate)/1000000.0f); // set integration time by time
elapsed since last filter update
  lastUpdate = Now;

  MadgwickQuaternionUpdate(mpu6050_data.Acc_X/9.8065, mpu6050_data.Acc_Y/9.8065,
mpu6050_data.Acc_Z/9.8065, mpu6050_data.Angle_Velocity_R,
mpu6050_data.Angle_Velocity_P, mpu6050_data.Angle_Velocity_Y);
  delt_t = millis() - count;
  if (delt_t > 50) { // update LCD once per half-second independent of read rate


  yaw=atan2(2.0f * (q[1] * q[2] + q[0] * q[3]), q[0] * q[0] + q[1] * q[1] - q[2]
* q[2] - q[3] * q[3]);
  pitch=-asin(2.0f * (q[1] * q[3] - q[0] * q[2]));
  roll=atan2(2.0f * (q[0] * q[1] + q[2] * q[3]), q[0] * q[0] - q[1] * q[1] - q[2]
* q[2] + q[3] * q[3]);

  //Serial.print(yaw, 2);
  //Serial.print(", ");
  //Serial.print(pitch, 2);
  //Serial.print(", ");
  //Serial.println(roll, 2);

  float R[3][3];
  R[0][0]=cos(pitch)*cos(yaw);
  R[0][1]=cos(yaw)*sin(roll)*sin(pitch)-cos(roll)*sin(yaw);
  R[0][2]=sin(roll)*sin(yaw)+cos(roll)*cos(yaw)*sin(pitch);

  R[1][0]=cos(pitch)*sin(yaw);
  R[1][1]=cos(roll)*cos(yaw)+sin(roll)*sin(pitch)*sin(yaw);
  R[1][2]=cos(roll)*sin(pitch)*sin(yaw)-cos(yaw)*sin(roll);

  R[2][0]=-sin(pitch);
  R[2][1]=cos(pitch)*sin(roll);
  R[2][2]=cos(roll)*cos(pitch);

  worldAccX=R[0][0]*mpu6050_data.Acc_X+R[0][1]*mpu6050_data.Acc_Y+R[0]
[2]*mpu6050_data.Acc_Z;
  worldAccY=R[1][0]*mpu6050_data.Acc_X+R[1][1]*mpu6050_data.Acc_Y+R[1]
[2]*mpu6050_data.Acc_Z;
  worldAccZ=R[2][0]*mpu6050_data.Acc_X+R[2][1]*mpu6050_data.Acc_Y+R[2]
[2]*mpu6050_data.Acc_Z;
```

```
  // 减去重力
  worldAccZ-=9.8065;

  //Serial.print(worldAccX, 2);
  //Serial.print(", ");
  //Serial.print(worldAccY, 2);
  //Serial.print(", ");
  //Serial.println(worldAccZ, 2);

  count = millis();

}
}
```

这里主要注意一个单位换算，Adafruit读出的加速度值单位是m/s^2，角速度单位是rad/s。而madgwick函数输入值的单位分别为g和rad/s，最后减去重力时单位仍然为m/s^2

获得欧拉角后旋转矩阵为：

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z(\gamma)\,R_y(\beta)\,R_x(\alpha)$$

$$R = \begin{bmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{bmatrix}$$

最终得到的去掉重力分量后的世界坐标系下的三轴加速度为：

突然向上

```
0.06,  0.04,  0.10
0.01,  0.01,  0.08
0.03,  -0.02,  -0.13
0.20,  0.26,  0.13
-1.30,  1.46,  9.93
-1.99,  -3.03,  11.95
2.99,  -0.38,  -6.46
-0.50,  1.28,  -11.45
0.60,  -0.79,  -5.50
1.25,  -2.06,  0.51
```

突然向下

```
-0.05,  -0.24,  0.06
1.13,  -0.95,  -0.30
1.61,  2.68,  -11.10
-0.70,  8.24,  -13.79
2.35,  -10.26,  11.88
-2.54,  -0.93,  9.98
-1.87,  2.33,  1.01
-0.06,  2.35,  -1.28
-0.39,  -1.27,  1.49
```

突然向左

```
0.02,  0.01,  0.21
0.04,  0.01,  -0.03
-0.03,  -0.08,  0.03
-4.27,  -2.45,  1.33
-21.16,  -8.98,  -1.21
8.99,  -0.21,  -0.70
13.69,  10.59,  1.13
3.54,  5.20,  -0.78
0.73,  1.25,  -0.80
```

突然向右

```
-0.00,  -0.00,  -0.14
-0.46,  -0.52,  -0.10
-0.72,  -0.41,  -0.44
10.09,  16.70,  0.38
12.08,  10.99,  -0.33
-13.13,  -16.07,  -3.11
-2.80,  -6.69,  2.06
2.22,  1.23,  1.28
-0.64,  0.82,  0.45
-0.69,  -0.45,  -0.62
0.90,  0.68,  0.25
```

我们获得的三个欧拉角是相当准确的，只是yaw角会缓慢漂移，这个是MPU6050的不可避免的硬件漏洞，只能靠安装磁力计并且与利用磁力计校准才不会漂移，但是这个漂移其实无伤大雅，大约5分钟累积10°

2.4 数据获取逻辑以及积分

```
#include "mpu6050.h"
#include <math.h>
#define PI 3.1415926

uint32_t delt_t = 0; // used to control display output rate
uint32_t count = 0;  // used to control display output rate

float GyroMeasError = PI * (40.0f / 180.0f);     // gyroscope measurement error
in rads/s (start at 60 deg/s), then reduce after ~10 s to 3
float beta = sqrt(3.0f / 4.0f) * GyroMeasError;  // compute beta
float GyroMeasDrift = PI * (2.0f / 180.0f);      // gyroscope measurement drift
in rad/s/s (start at 0.0 deg/s/s)
float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;  // compute zeta, the other free
parameter in the Madgwick scheme usually set to a small or zero value
float deltat = 0.0f;                              // integration interval for
both filter schemes
uint32_t lastUpdate = 0, firstUpdate = 0;         // used to calculate
integration interval
uint32_t Now = 0;
float q[4] = {1.0f, 0.0f, 0.0f, 0.0f}; //四元数

float pitch, yaw, roll;//三轴角
float worldAccX,worldAccY,worldAccZ;//世界坐标系下三轴加速度

const int buttonPin=13;//按钮GPIO13
int buttonstate=0;//按钮状态
int lastbuttonstate=0;//上一次按钮状态
int i=0;//下标
float x_acc[50];//加速度
float y_acc[50];
float z_acc[50];
float x_v[50];//速度
float y_v[50];
float z_v[50];
float x_x[50];//位移
float y_x[50];
float z_x[50];
int k;//工具变量

float sumwhat(int i,float tool[50]){//辅助函数求tool数组的前i项和
  int j;
  float sum=0.0;
  for (j=0;j<=i;j++){
    sum+=tool[j]*0.05;
  }
  return sum;
}

unsigned long starttime=0;//开始时间
int flag1=0;//为了标志2s已到但是按钮未松开的情况的flag
int flag2=1;//为了标志开始计时的flag
```

```
void MadgwickQuaternionUpdate(float ax, float ay, float az, float gyrox, float
gyroy, float gyroz)
{
    float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];         // short name local
variable for readability
    float norm;                                               // vector norm
    float f1, f2, f3;                                         // objetive
funcyion elements
    float J_11or24, J_12or23, J_13or22, J_14or21, J_32, J_33; // objective
function Jacobian elements
    float qDot1, qDot2, qDot3, qDot4;
    float hatDot1, hatDot2, hatDot3, hatDot4;
    float gerrx, gerry, gerrz, gbiasx, gbiasy, gbiasz;        // gyro bias error

    // Auxiliary variables to avoid repeated arithmetic
    float _halfq1 = 0.5f * q1;
    float _halfq2 = 0.5f * q2;
    float _halfq3 = 0.5f * q3;
    float _halfq4 = 0.5f * q4;
    float _2q1 = 2.0f * q1;
    float _2q2 = 2.0f * q2;
    float _2q3 = 2.0f * q3;
    float _2q4 = 2.0f * q4;
    float _2q1q3 = 2.0f * q1 * q3;
    float _2q3q4 = 2.0f * q3 * q4;

    // Normalise accelerometer measurement
    norm = sqrt(ax * ax + ay * ay + az * az);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f/norm;
    ax *= norm;
    ay *= norm;
    az *= norm;

    // Compute the objective function and Jacobian
    f1 = _2q2 * q4 - _2q1 * q3 - ax;
    f2 = _2q1 * q2 + _2q3 * q4 - ay;
    f3 = 1.0f - _2q2 * q2 - _2q3 * q3 - az;
    J_11or24 = _2q3;
    J_12or23 = _2q4;
    J_13or22 = _2q1;
    J_14or21 = _2q2;
    J_32 = 2.0f * J_14or21;
    J_33 = 2.0f * J_11or24;

    // Compute the gradient (matrix multiplication)
    hatDot1 = J_14or21 * f2 - J_11or24 * f1;
    hatDot2 = J_12or23 * f1 + J_13or22 * f2 - J_32 * f3;
    hatDot3 = J_12or23 * f2 - J_33 *f3 - J_13or22 * f1;
    hatDot4 = J_14or21 * f1 + J_11or24 * f2;

    // Normalize the gradient
    norm = sqrt(hatDot1 * hatDot1 + hatDot2 * hatDot2 + hatDot3 * hatDot3 +
hatDot4 * hatDot4);
    hatDot1 /= norm;
    hatDot2 /= norm;
```

```
    hatDot3 /= norm;
    hatDot4 /= norm;

    // Compute estimated gyroscope biases
    gerrx = _2q1 * hatDot2 - _2q2 * hatDot1 - _2q3 * hatDot4 + _2q4 * hatDot3;
    gerry = _2q1 * hatDot3 + _2q2 * hatDot4 - _2q3 * hatDot1 - _2q4 * hatDot2;
    gerrz = _2q1 * hatDot4 - _2q2 * hatDot3 + _2q3 * hatDot2 - _2q4 * hatDot1;

    // Compute and remove gyroscope biases
    gbiasx += gerrx * deltat * zeta;
    gbiasy += gerry * deltat * zeta;
    gbiasz += gerrz * deltat * zeta;
    gyrox -= gbiasx;
    gyroy -= gbiasy;
    gyroz -= gbiasz;

    // Compute the quaternion derivative
    qDot1 = -_halfq2 * gyrox - _halfq3 * gyroy - _halfq4 * gyroz;
    qDot2 =  _halfq1 * gyrox + _halfq3 * gyroz - _halfq4 * gyroy;
    qDot3 =  _halfq1 * gyroy - _halfq2 * gyroz + _halfq4 * gyrox;
    qDot4 =  _halfq1 * gyroz + _halfq2 * gyroy - _halfq3 * gyrox;

    // Compute then integrate estimated quaternion derivative
    q1 += (qDot1 -(beta * hatDot1)) * deltat;
    q2 += (qDot2 -(beta * hatDot2)) * deltat;
    q3 += (qDot3 -(beta * hatDot3)) * deltat;
    q4 += (qDot4 -(beta * hatDot4)) * deltat;

    // Normalize the quaternion
    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise
quaternion
    norm = 1.0f/norm;
    q[0] = q1 * norm;
    q[1] = q2 * norm;
    q[2] = q3 * norm;
    q[3] = q4 * norm;
}

void setup() {
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(115200);
  Init_mpu6050();
  pinMode(buttonPin,INPUT_PULLUP);
}

void loop() {
  buttonstate=digitalRead(buttonPin);
  ReadMPU6050();
  //可视化线加速度曲线
  //Serial.print("Acc_x:");
  //Serial.print(mpu6050_data.Acc_X);
  //Serial.print(",");
  //Serial.print("Acc_Y:");
  //Srial.print(mpu6050_data.Acc_Y);
  //Serial.print(",");
```

```
  //Serial.print("Acc_Z:");
  //Serial.println(mpu6050_data.Acc_Z);

  //可视化角速度曲线
  //Serial.print("Angle_velocity_R:");
  //Serial.println(mpu6050_data.Angle_Velocity_R);
  //Serial.print(",");
  //Serial.print("Angle_velocity_P:");
  //Serial.print(mpu6050_data.Angle_Velocity_P);
  //Serial.print(",");
  //Serial.print("Angle_velocity_Y:");
  //Serial.println(mpu6050_data.Angle_Velocity_Y);

  Now = micros();
  deltat = ((Now - lastUpdate)/1000000.0f); // set integration time by time
elapsed since last filter update
  lastUpdate = Now;

  MadgwickQuaternionUpdate(mpu6050_data.Acc_X/9.8065, mpu6050_data.Acc_Y/9.8065,
mpu6050_data.Acc_Z/9.8065, mpu6050_data.Angle_Velocity_R,
mpu6050_data.Angle_Velocity_P, mpu6050_data.Angle_Velocity_Y);
  delt_t = millis() - count;
  if (delt_t > 50) { // update LCD once per half-second independent of read rate


  yaw=atan2(2.0f * (q[1] * q[2] + q[0] * q[3]), q[0] * q[0] + q[1] * q[1] - q[2]
* q[2] - q[3] * q[3]);
  pitch=-asin(2.0f * (q[1] * q[3] - q[0] * q[2]));
  roll=atan2(2.0f * (q[0] * q[1] + q[2] * q[3]), q[0] * q[0] - q[1] * q[1] - q[2]
* q[2] + q[3] * q[3]);

  //Serial.print(yaw, 2);
  //Serial.print(", ");
  //Serial.print(pitch, 2);
  //Serial.print(", ");
  //Serial.println(roll, 2);

  float R[3][3];
  R[0][0]=cos(pitch)*cos(yaw);
  R[0][1]=cos(yaw)*sin(roll)*sin(pitch)-cos(roll)*sin(yaw);
  R[0][2]=sin(roll)*sin(yaw)+cos(roll)*cos(yaw)*sin(pitch);

  R[1][0]=cos(pitch)*sin(yaw);
  R[1][1]=cos(roll)*cos(yaw)+sin(roll)*sin(pitch)*sin(yaw);
  R[1][2]=cos(roll)*sin(pitch)*sin(yaw)-cos(yaw)*sin(roll);

  R[2][0]=-sin(pitch);
  R[2][1]=cos(pitch)*sin(roll);
  R[2][2]=cos(roll)*cos(pitch);

  worldAccX=R[0][0]*mpu6050_data.Acc_X+R[0][1]*mpu6050_data.Acc_Y+R[0]
[2]*mpu6050_data.Acc_Z;
  worldAccY=R[1][0]*mpu6050_data.Acc_X+R[1][1]*mpu6050_data.Acc_Y+R[1]
[2]*mpu6050_data.Acc_Z;
  worldAccZ=R[2][0]*mpu6050_data.Acc_X+R[2][1]*mpu6050_data.Acc_Y+R[2]
[2]*mpu6050_data.Acc_Z;
```

```
  // 减去重力
  worldAccZ-=9.8065;

  //Serial.print(worldAccX, 2);
  //Serial.print(", ");
  //Serial.print(worldAccY, 2);
  //Serial.print(", ");
  //Serial.println(worldAccZ, 2);
  if(buttonstate==LOW&&flag1==0){//如果按下按钮
    if(lastbuttonstate==HIGH){//首次按下按钮
      i=0;//下标更新
      starttime=millis();//起始时间更新
      flag2=0;//开始计时标志
      Serial.print("首次按下");
    }
    x_acc[i]=worldAccX;//更新加速度
    y_acc[i]=worldAccY;
    z_acc[i]=worldAccZ;
    i++;
  }

  if((millis()-starttime>=2000||
(lastbuttonstate==LOW&&buttonstate==HIGH))&&flag2==0){
    flag1=1;//关闭记录通道
    flag2=1;//关闭积分运算通道
    //积分运算
    Serial.print("进入积分");
    for (k=0;k<i;k++){
      x_v[k]=sumwhat(k,x_acc);
      y_v[k]=sumwhat(k,y_acc);
      z_v[k]=sumwhat(k,z_acc);
    }
    for (k=0;k<i;k++){
      x_x[k]=sumwhat(k,x_v);
      y_x[k]=sumwhat(k,y_v);
      z_x[k]=sumwhat(k,z_v);
    }
    for (k=0;k<i;k++){
      Serial.print("x:");
      Serial.print(x_x[k]);
      Serial.print(",y:");
      Serial.print(y_x[k]);
      Serial.print(",z:");
      Serial.println(z_x[k]);
      }
  }
  if(buttonstate==HIGH){flag1=0;}//如果按钮是松开的允许再按记录，如果是按到时间到了结束，
不触发flag1变化，不进记录
  lastbuttonstate=buttonstate;//记录上一次按钮状态

  count = millis();


}
```

```
}
```

和之前的一样，这个获取的相当准确

向上运动

```
首次按下进入积分x:-0.00,y:-0.00,z:0.0(
x:-0.00,y:-0.00,z:0.00
x:-0.00,y:-0.00,z:0.01
x:-0.00,y:-0.00,z:0.01
x:-0.00,y:-0.01,z:0.01
x:-0.00,y:-0.01,z:0.01
x:-0.00,y:-0.01,z:0.01
x:0.00,y:-0.01,z:0.02
x:0.01,y:-0.02,z:0.07
x:0.00,y:-0.03,z:0.16
x:-0.01,y:-0.06,z:0.25
x:-0.03,y:-0.07,z:0.31
x:-0.04,y:-0.09,z:0.34
x:-0.06,y:-0.11,z:0.35
x:-0.07,y:-0.12,z:0.36
```

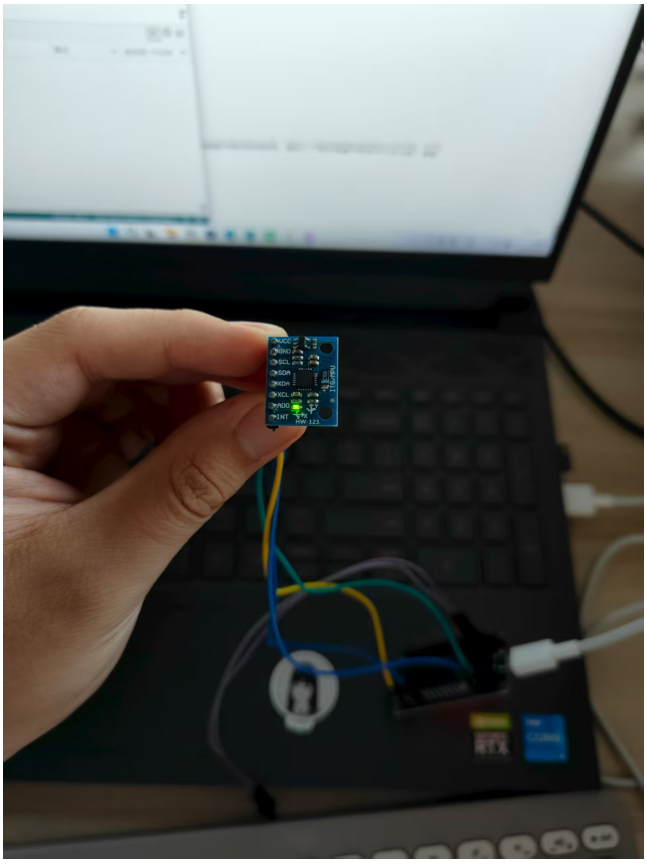
向下运动

```
首次按下进入积分x:0.00,y:0.00,z:0.00
x:0.00,y:0.00,z:0.00
x:0.00,y:0.00,z:0.00
x:0.00,y:0.00,z:0.01
x:0.00,y:0.00,z:0.01
x:0.00,y:0.00,z:0.01
x:0.00,y:0.00,z:0.02
x:0.00,y:0.00,z:0.02
x:0.00,y:0.00,z:0.02
x:-0.00,y:0.01,z:0.00
x:0.00,y:0.03,z:-0.05
x:0.01,y:0.03,z:-0.12
x:0.00,y:0.03,z:-0.16
x:0.00,y:0.03,z:-0.19
x:-0.00,y:0.03,z:-0.20
x:-0.01,y:0.03,z:-0.20
x:-0.01,y:0.03,z:-0.19
```

此时几乎只有z轴有响应。

至于左右动，有一个问题，就是其对世界坐标系的定义其实是三个欧拉角都为零时的坐标系，那边三个欧拉角都为零是怎么定义的，是取决于上电时的传感器姿态，即将上电时的传感器坐标系作为世界坐标系。

当传感器这样放置时（俯视图）：、



左移

右移

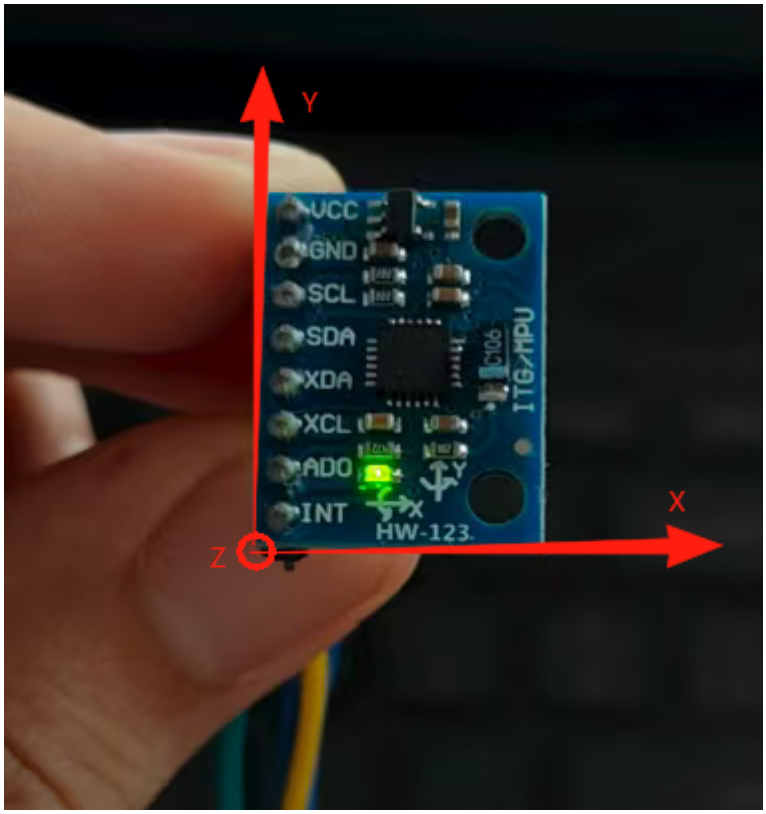
```
首次按下进入积分x:-0.00,y:-0.00,z:0.00
x:0.00,y:0.00,z:0.00
x:0.00,y:-0.00,z:0.00
x:0.00,y:-0.00,z:0.01
x:0.00,y:-0.00,z:0.01
x:0.00,y:-0.00,z:0.02
x:-0.02,y:0.01,z:0.02
x:-0.08,y:0.03,z:0.02
x:-0.14,y:0.04,z:0.03
x:-0.19,y:0.05,z:0.03
x:-0.22,y:0.06,z:0.02
x:-0.24,y:0.08,z:0.02
```

```
首次按下进入积分x:-0.00,y:0.00,z:-0.00
x:-0.00,y:0.00,z:-0.00
x:0.00,y:-0.00,z:0.00
x:0.00,y:-0.00,z:0.00
x:-0.00,y:-0.00,z:0.00
x:0.00,y:-0.00,z:0.00
x:0.00,y:-0.00,z:0.01
x:0.04,y:0.00,z:0.03
x:0.14,y:-0.01,z:0.03
x:0.19,y:-0.01,z:0.04
x:0.20,y:-0.02,z:0.05
x:0.19,y:-0.03,z:0.06
x:0.18,y:-0.03,z:0.08
```

所动的是x且左负右正

水平旋转180°时所动的仍然是x只不过左正右负

说明其姿态与建立的坐标系的关系是这样的（虽然在板子上也有画）：

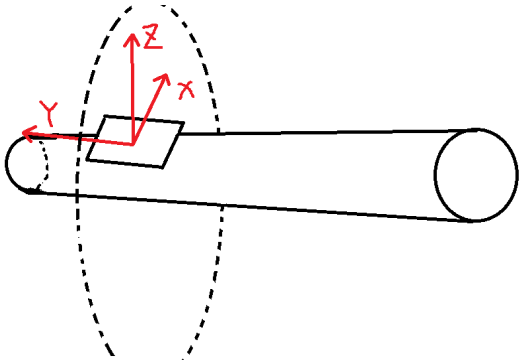


现在我们获得了非常准确与稳定的三轴角速度和能够准确反映运动轨迹的两次积分得到的位移数组，问题就出在我们世界坐标系的定义上，如果我们仅在一套坐标系下得到的数据训练模型，在其他坐标系下是不适用的。

比如在某个坐标系下左右移动是x数据变化，以此训练出后，实际应用时左右移动为y数据变化，这样先前的判断依据就不适用了。

2.5 世界坐标系定义问题

但是实际上我们再仔细想一下，我们所需要的一切运动规矩：无论是简单的上下左右，还是复杂的画圆、画Z字等，本质上都是一个二维图形，我们可以看作是在xOz平面上的投影，比如说向下运动时，我们的运动轨迹可能为既向前又向下，即Y为正、Z为负，但是用作分类时我们只需考虑这段三维轨迹在xOz平面上投影，即Z为负、X为0，于是我们可以考虑将传感器这样安装：



即传感器坐标系下的Y轴正方向始终指向魔杖的正方向，这样魔杖与xOz平面始终保持垂直，这样安装我们就无需考虑y轴的位移，并且在正常使用中也不会出现y轴对应左右的情况，y轴的情况对整体的轨迹判断无影响。

其实可以用一个补偿矩阵将每次上电时重新定义的世界坐标系校准到一个提前定义好的固定的世界坐标系，比如固定X轴指西、Y轴指南，但是这样明显是不行的，如果你的世界坐标系是一定的，当你面向不同的方位时，对同一个运动轨迹又有不同的三轴位移解算。比如当你面向西方左右挥舞，对应变化的坐标不是X轴而是Y轴。

对于上述的安装方案，仍然有一个棘手的问题是魔杖可以绕Y轴自由旋转，这样的话X和Z轴的解算情况就乱套了，但是只要能够实现：无论魔杖绕Y轴旋转多少度时上电，从魔杖根部看向尖端的视角下，都有Z轴向上\X轴向右的世界坐标系即可满足上下对应Z、左右对应X，并且在运动过程中无论怎么绕Y轴旋转都不影响（因为有旋转矩阵矫正，经过实验了）。换句话说就是无论上电时魔杖怎么转，所定义的世界坐标系都是图上的那个，而不随传感器位置变化而变化。

但是，正当研究怎么实现的时候，突然发现了一个问题，那就是我们去除重力分量是始终默认直接z轴加速度减去g即可的，当我们在不同姿态下上电时，静止时的三轴加速度始终为0，如果真的是世界坐标系定义为上电时的传感器坐标系时，那么当z轴不竖直时，重力加速度在三轴都有分量，静止时三轴加速度不可能稳定在0，**这就推翻了之前我们对世界坐标系的定义：此传感器定义的世界坐标系的z轴是始终竖直向上的，而X轴Y轴则是传感器坐标系的XY轴在垂直于Z轴的平面上的投影**（这可能由于获得三轴角时需要四元数与重力分量来计算）。**这样构建的三轴坐标系，无论以什么姿态上电，Z轴永远竖直向上，变化的只有X与Y的方位，而上述安装方案又将Y轴的方位确定，这样X轴方位也就固定了，因此只要挥舞时人面对的朝向不变，模型就有用。**

（这个世界坐标系的定义绝对正确，这是我在研究怎么固定Z轴朝向时突然想出来的，如果他不固定的话用 worldAccZ-=9.8065; 来消除重力分量不是纯扯淡吗，所以Z轴他必须是固定的，剩下的就不攻自破了，就算把Y轴朝天上电，左右前后移动时依然对应的是XZ轴。

左　　右　　上　　下

首次按下进入积分x:0.00, y:-0.00, z:-0.00
x:0.00, y:-0.00, z:-0.00
x:0.00, y:-0.00, z:0.00
x:0.00, y:-0.00, z:0.00
x:0.00, y:-0.00, z:0.00
x:-0.00, y:-0.00, z:0.00
x:-0.02, y:-0.00, z:-0.00
x:-0.07, y:-0.00, z:0.01
x:-0.14, y:-0.00, z:0.02
x:-0.19, y:-0.01, z:0.04
x:-0.21, y:-0.03, z:0.05
x:-0.22, y:-0.03, z:0.06

首次按下进入积分x:0.00, y:-0.00, z:-0.00
x:0.00, y:-0.00, z:-0.00
x:0.00, y:-0.00, z:0.00
x:0.00, y:-0.00, z:0.00
x:0.00, y:-0.00, z:0.00
x:-0.00, y:-0.00, z:0.00
x:-0.02, y:-0.00, z:-0.00
x:-0.07, y:-0.00, z:0.01
x:-0.14, y:-0.00, z:0.02
x:-0.19, y:-0.01, z:0.04
x:-0.21, y:-0.03, z:0.05
x:-0.22, y:-0.03, z:0.06

首次按下进入积分x:0.00, y:0.00, z:0.00
x:0.00, y:0.00, z:0.00
x:0.00, y:0.00, z:0.00
x:0.00, y:0.00, z:0.00
x:0.00, y:0.00, z:-0.00
x:-0.00, y:-0.00, z:-0.01
x:-0.00, y:0.00, z:-0.01
x:0.00, y:0.00, z:-0.01
x:0.00, y:-0.01, z:0.05
x:0.00, y:-0.04, z:0.12
x:0.01, y:-0.06, z:0.17
x:0.01, y:-0.07, z:0.19
x:-0.00, y:-0.07, z:0.19

首次按下进入积分x:-0.00, y:0.00, z:-0.00
x:-0.00, y:0.00, z:-0.00
x:-0.00, y:0.00, z:-0.00
x:-0.00, y:0.00, z:-0.00
x:0.00, y:0.00, z:-0.02
x:0.01, y:-0.01, z:-0.07
x:0.02, y:-0.01, z:-0.14
x:0.02, y:0.00, z:-0.18
x:0.02, y:0.01, z:-0.20
x:0.03, y:0.02, z:-0.21

现在关于运动轨迹解算为位移数组这一块已经解决了（唯一缺点是开机时间长了yaw角会飘，导致坐标系会缓慢绕z轴旋转，问题不大）

3 数据收集与处理

如果只是区分上下左右的话，可以只考虑XZ轴的位移，用简单的if语句来进行判断吗？

并不可以，因为实际情况并不能做到完美的竖直上下和水平左右移动，可以加一个阈值吗？也不可以，比如你向右运动时向下也有运动，这种情况应该判为向右，而这个阈值把握不准。

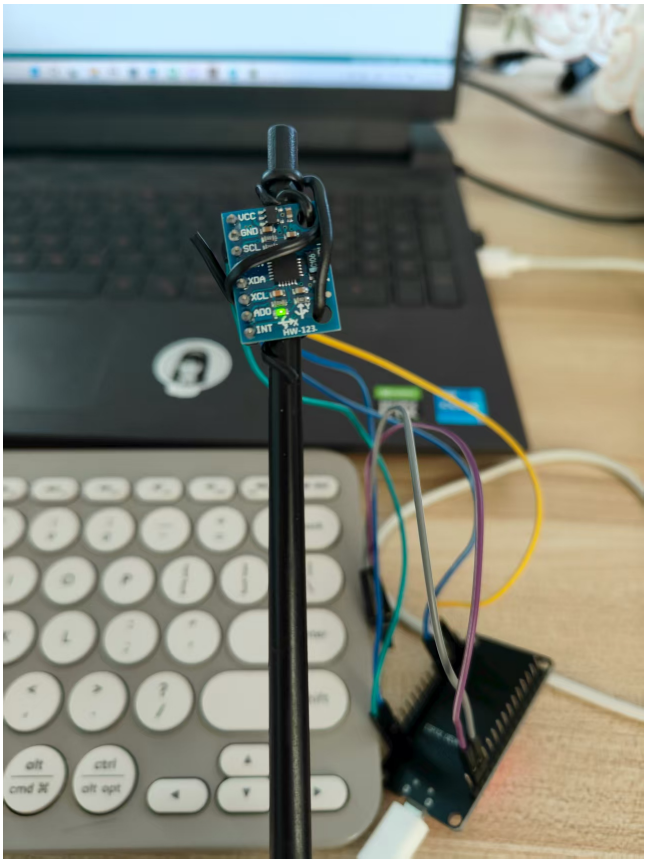
所以直接上机器学习。

3.1 数据收集

首先要获得大量的原始数据作为数据集，需要将每次运行得到的XYZ三轴位移三个数组打包起来并且写入一个txt文件存档（其实只需要XZ轴），每50ms输出一次数据，最多2s，经验证最多41个数据，为了保险定义长度50的数组。

按照人类的使用习惯，传感器安装在魔杖顶端，手握住末端，人手挥动时基本上只动手腕，因此不会出现纯上下左右平移的情况，而是会在Y轴上自然做出一些移动的，比如向上，人习惯向上移动时向内(-Y方向)挥动；向左，人也习惯向左时向内移动一些，因此我们忽略Y轴位移只看XOZ平面投影的方法是正确的。

为了模拟真实的魔杖挥舞场景，因为真正的魔杖还没有打印出来，我将传感器绑在筷子顶端，板子在末端，一手持按钮，模拟魔杖挥舞的场景以采集基础数据。如图



我们想要将每次按下按钮后得到的运动轨迹解算的位移数据保存下来作为训练模型的数据集，可以用python的**pyserial**库来通过串口传输数据，并且将其保存在txt文档中。

arduino内串口打印处代码加上开始与结束的标识符

```
...
    Serial.println("===START===");    // 一次动作开始
    for (k=0;k<i;k++){
      Serial.print("x:");
      Serial.print(x_x[k]);
      Serial.print(",y:");
      Serial.print(y_x[k]);
      Serial.print(",z:");
      Serial.println(z_x[k]);
      }
    Serial.println("===END===");      // 一次动作结束
...
```

python脚本如下

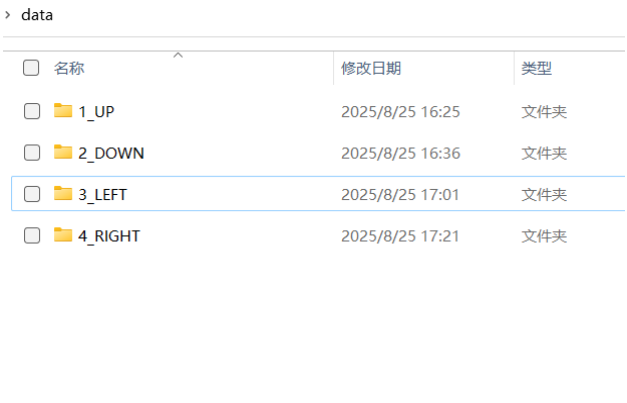
```
import serial
import time
import os

#串口参数
SERIAL_PORT = "COM7"
BAUD_RATE = 115200


#保存路径
SAVE_DIR = r"D:\新建文件夹 (5)\data\RIGHT"
os.makedirs(SAVE_DIR, exist_ok=True)

ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
time.sleep(2)  # 等待串口稳定

sample_idx = 1
recording = False
buffer = []

while True:
    line = ser.readline().decode("utf-8", errors="ignore").strip()
    if not line:
        continue

    if line == "===START===":
        print(f"开始采集样本 {sample_idx}")
        buffer = []
        recording = True
        continue

    if line == "===END===":
        if recording:
            filename = os.path.join(SAVE_DIR, f"sample_{sample_idx:03d}.txt")
            with open(filename, "w") as f:
                f.write("\n".join(buffer))
            print(f"样本 {sample_idx} 保存到 {filename}")
            sample_idx += 1
            recording = False
```

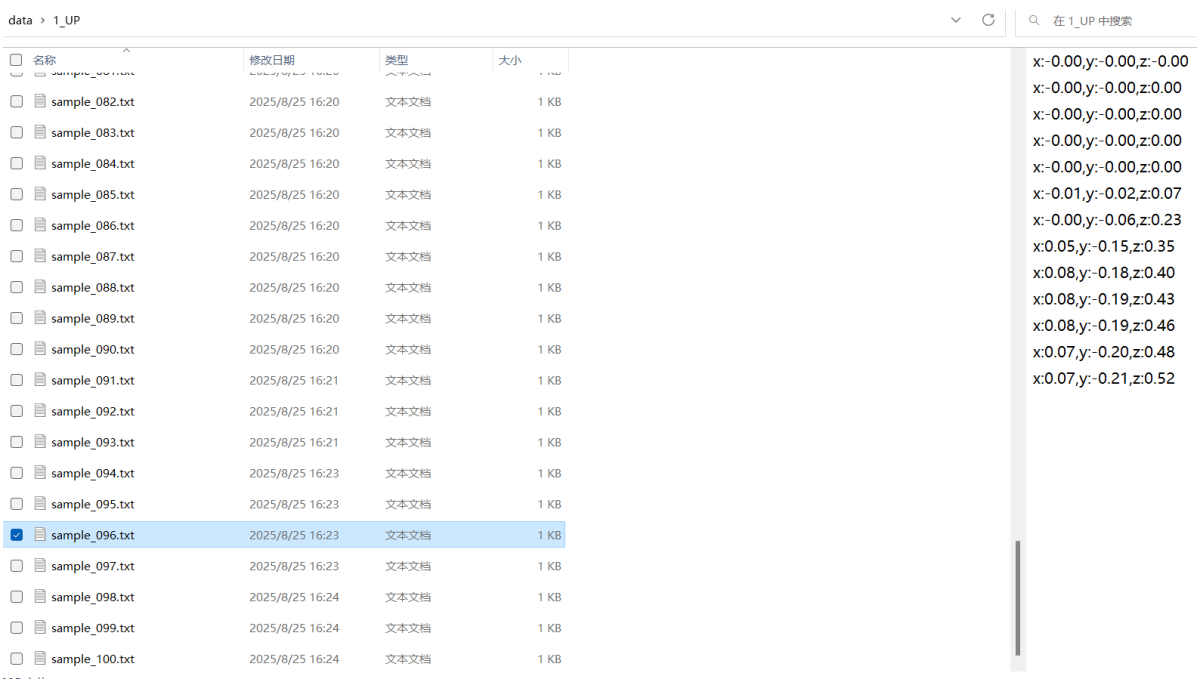
```
            continue

        if recording:
            buffer.append(line)

    ser.close()
```

在检测到串口输出有起始符"===START==="时，便可将每行数据读取出来并写入指定的txt文档，每个文档写一组数据，读取到终止符"===END==="时结束。因为用的是死循环，所以退出循环需要手动关闭，此代码会按顺序依次创建txt文档，一个串口只能同时被一个工作占用，即此代码、串口监视器、代码上传等只能同时运行一个。

共有四个文件夹依次对应上下左右的运动轨迹，每个轨迹收集了一百组数据。

```
开始采集样本 88
样本 88 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_088.txt
开始采集样本 89
样本 89 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_089.txt
开始采集样本 90
样本 90 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_090.txt
开始采集样本 91
样本 91 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_091.txt
开始采集样本 92
样本 92 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_092.txt
开始采集样本 93
样本 93 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_093.txt
开始采集样本 94
样本 94 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_094.txt
开始采集样本 95
样本 95 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_095.txt
开始采集样本 96
样本 96 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_096.txt
开始采集样本 97
样本 97 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_097.txt
开始采集样本 98
样本 98 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_098.txt
开始采集样本 99
样本 99 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_099.txt
开始采集样本 100
样本 100 保存到 D:\新建文件夹 (5)\data\RIGHT\sample_100.txt
```





3.2 数据处理

现在我们得到了若干txt文档，每个文档每一行都有xyz的位移数据，实际上我们只需要xz轴的。为了方便后续处理我们可以将每一组数据看为一个2×50的矩阵，第一行对应x轴数据，第二行对应z轴数据，每个轨迹对应的所有组数据整合为一个大100×2×50的矩阵。

对单个txt文档的处理，我们可以用正则表达式仅取x：和z：之后的数据。对批量文件的处理可以用glob库

```python
import numpy as np
import re
import glob

def process_txt(file_path, max_len=50):
    #处理单个txt文件，返回一个2x50的矩阵
    x_vals, z_vals=[],[]

    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            match=re.findall(r"x:([-+]?\d*\.?\d+),y:([-+]?\d*\.?\d+),z:([-+]?
\d*\.?\d+)", line)
            if match:
                x, _, z=match[0]
                x_vals.append(float(x))
                z_vals.append(float(z))

    #不够的补0
    x_vals=(x_vals+[0.0]*max_len)[:max_len]
    z_vals=(z_vals+[0.0]*max_len)[:max_len]

    return np.array([x_vals, z_vals])

# 批量读取文件
files_1=glob.glob(r"D:\新建文件夹 (5)\data\1_UP\*.txt")
files_2=glob.glob(r"D:\新建文件夹 (5)\data\2_DOWN\*.txt")
files_3=glob.glob(r"D:\新建文件夹 (5)\data\3_LEFT\*.txt")
files_4=glob.glob(r"D:\新建文件夹 (5)\data\4_RIGHT\*.txt")

matrices_1=[process_txt(f) for f in files_1]
matrices_2=[process_txt(f) for f in files_2]
matrices_3=[process_txt(f) for f in files_3]
matrices_4=[process_txt(f) for f in files_4]
#堆叠成三维数组 N×2×50

data_1= np.stack(matrices_1)
data_2= np.stack(matrices_2)
data_3= np.stack(matrices_3)
data_4= np.stack(matrices_4)
```

最终结果：得到了四个100×2×50的矩阵

```python
print(data_1.shape)
print(data_1[0])
```

```
(100, 2, 50)
[[-0.   -0.   -0.   -0.   -0.01 -0.02 -0.05 -0.11 -0.15 -0.16 -0.15 -0.14
  -0.12  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.  ]
 [ 0.    0.01  0.01  0.01  0.01  0.07  0.2   0.31  0.35  0.37  0.38  0.4
   0.41  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   0.    0.  ]]
```

3.3 模型训练

因为我们数据集非常小，而且数据特征其实很明显，我们可以将每组数据展开成一维并且将其堆叠起来成一个二维矩阵，然后用SVM支持向量机来拟合，SVM性能超级吊，四个轨迹的标签依次为1、2、3、4

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

#拼接成一个总数据集
X=np.concatenate([data_1,data_2,data_3,data_4],axis=0)
y=np.array([1]*100+[2]*100+[3]*100+[4]*100)

#展平每个样本
X=X.reshape(X.shape[0],-1)

#划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=41, stratify=y
)

#用SVM训练
clf = SVC(kernel='rbf', C=1, gamma='scale')
clf.fit(X_train, y_train)
#预测
y_pred = clf.predict(X_test)

#评估
print("准确率:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

得到的结果为：

```
准确率: 0.925
              precision    recall  f1-score   support

           1       1.00      0.87      0.93        30
           2       0.91      0.97      0.94        30
           3       0.88      0.97      0.92        30
           4       0.93      0.90      0.92        30

    accuracy                           0.93       120
   macro avg       0.93      0.93      0.92       120
weighted avg       0.93      0.93      0.92       120
```

将所有原数据处理后丢进模型中跑一遍得到的结果为：

```
准确率: 0.925
              precision    recall  f1-score   support

           1       1.00      0.87      0.93        30
           2       0.91      0.97      0.94        30
           3       0.88      0.97      0.92        30
           4       0.93      0.90      0.92        30

    accuracy                           0.93       120
   macro avg       0.93      0.93      0.92       120
weighted avg       0.93      0.93      0.92       120
```

效果还是很不错的

3.4 测试

想要实时监测一下魔杖得到的数据经过模型处理后预测的结果，很简单，只需要将前面的代码融合一下即可

串口接收数据写入txt——>process_txt函数转为矩阵——>矩阵展品预测——>输出预测结果

```
import serial
import time
import os

#串口参数
SERIAL_PORT = "COM7"   # 根据实际修改，比如 Linux 下 "/dev/ttyUSB0"
BAUD_RATE = 115200

#保存路径
SAVE_DIR = r"D:\新建文件夹 (5)\data"
os.makedirs(SAVE_DIR, exist_ok=True)

ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
time.sleep(2)  # 等待串口稳定

recording = False
buffer = []

while True:
    line = ser.readline().decode("utf-8", errors="ignore").strip()
    if not line:
        continue
    if line == "===START===":
        print("开始采集样本")
        buffer = []
        recording = True
        continue
    if line == "===END===":
        if recording:
            filename = os.path.join(SAVE_DIR, "test.txt")
            with open(filename, "w") as f:
                f.write("\n".join(buffer))
            print("样本已保存")
            recording = False
        array=[process_txt(r"D:\新建文件夹 (5)\data\test.txt")]
        test=np.stack(array)
        test=test.reshape(test.shape[0],-1)
        result=clf.predict(test)
        print(result)
        continue

    if recording:
        buffer.append(line)

ser.close()
```

```
开始采集样本
样本已保存
[1]
开始采集样本
样本已保存
[2]
开始采集样本
样本已保存
[3]
开始采集样本
样本已保存
[4]
```

效果非常好 此处有视频为证

3.5 模型部署到esp32上

现在模型仅存在于python脚本中，是电脑通过串口读取数据再在python脚本中进行数据处理和模型预测的，但是我们的项目是个独立的嵌入式个体，成品肯定不能依赖于电脑等上位机，于是我们需要将模型部署到开发板上。

所幸我们使用的是机器学习的小模型，而且我们的数据量也不大，算力应该是足够的。

python训练的模型，本质上是一个较为复杂的大函数，训练只是调整里面的参数，知道他的结构便可以用c语言复刻出来，不过这么大的模型手搓会累死的，还好已经有了micromlgen库，可以将简单的模型用c语言输出。注意此时训练的模型的参数需要改变一下

```
clf=SVC(kernel='rbf',C=1,gamma=1.0)
```

此时的gamma参数决定了rbf核的宽度，过大过小都会影响效果，这里的1.0是试出来的，对应准确率0.925。在正常训练时可以用字符串参数"scale"来自动调参，但是对于这些c语言模型转换库，gamma必须有一个确定的值。

```
from micromlgen import port
c_code=port(clf)
with open(r"D:\新建文件夹 (5)\data\classifier.h", "w") as f:
    f.write(c_code)
```

用micromlgen库成功导出模型的c语言版本后，得到的结果为：

classifier.h：

```
#pragma once
#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class SVM {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        float kernels[131] = { 0 };
                        float decisions[6] = { 0 };
                        int votes[4] = { 0 };
```

```
                        kernels[0] = compute_kernel(x,   0.0  , 0.0  , -0.0  ,
-0.0  , 0.1  , 0.18  , 0.19  , 0.19  , 0.19  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , -0.0  , -0.0  , 0.03  , 0.14  , 0.27  , 0.31  , 0.33  , 0.33  , 0.34  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0 );
                        ...//此处省略128个kernels

                        kernels[130] = compute_kernel(x,   -0.0  , -0.0  , -0.01
, -0.01  , -0.01  , -0.0  , 0.08  , 0.2  , 0.29  , 0.32  , 0.34  , 0.34  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , -0.0  , -0.0  , -0.0  , -0.01  , -0.01  , -0.01  , 0.02  , 0.04
, 0.05  , 0.03  , 0.04  , 0.03  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  ,
0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0
, 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0  , 0.0 );
                        decisions[0] = -0.206241160743
                        + kernels[0] * 0.06523232929
                        + kernels[6] * 0.60241767888
                        + kernels[7] * 0.239027242719
                        + kernels[8] * 0.452574668479
                        + kernels[9] * 0.31387810585
                        + kernels[11]
                        + kernels[15] * 0.786632371654
                        + kernels[17] * 0.353397278291
                        + kernels[18]
                        + kernels[19] * 0.162507380842
                        + kernels[20] * 0.252686093615
                        + kernels[22] * 0.122671373807
                        + kernels[25] * 0.785626220029
                        + kernels[26] * 0.279711296209
                        + kernels[27]
                        + kernels[29] * 0.492071881706
                        + kernels[33]
                        + kernels[34] * -0.65015214112
                        + kernels[35] * -0.252584644359
                        + kernels[36] * -0.395736043262
                        + kernels[38] * -0.227120165159
                        + kernels[39] * -0.582519956587
                        + kernels[40] * -0.111581988234
                        + kernels[41] * -0.036852170466
                        + kernels[43] * -0.503157877139
                        + kernels[44] * -0.617754403456
                        + kernels[45] * -0.421330550067
                        + kernels[46] * -0.526631498636
                        - kernels[47]
                        + kernels[50] * -0.764576548976
                        + kernels[51] * -0.083932009446
                        + kernels[52] * -0.312736772403
                        + kernels[53] * -0.022218182772
```

```
                    - kernels[54]
                    + kernels[56] * -0.058190148545
                    + kernels[59] * -0.34135882074
                    - kernels[62]
                    ;
                    decisions[1] = 0.222983612937
                    + kernels[2] * 0.571709273039
                    + kernels[6] * 0.554642091523
                    + kernels[7] * 0.425930457234
                    + kernels[11]
                    + kernels[12]
                    + kernels[13]
                    + kernels[15]
                    + kernels[16]
                    + kernels[17]
                    + kernels[18]
                    + kernels[19]
                    + kernels[21]
                    + kernels[25] * 0.312308029541
                    + kernels[27]
                    + kernels[32] * 0.910092051203
                    + kernels[33]
                    - kernels[63]
                    - kernels[64]
                    - kernels[66]
                    - kernels[68]
                    - kernels[70]
                    + kernels[72] * -0.309169392721
                    + kernels[75] * -0.351535711493
                    - kernels[76]
                    + kernels[77] * -0.671477022822
                    - kernels[79]
                    + kernels[80] * -0.435690678723
                    + kernels[82] * -0.119057268052
                    + kernels[83] * -0.116169239717
                    - kernels[87]
                    - kernels[88]
                    + kernels[89] * -0.245569215807
                    + kernels[90] * -0.434668932959
                    - kernels[91]
                    + kernels[92] * -0.091344440245
                    - kernels[94]
                    ;
                    decisions[2] = 0.462716119431
                    + kernels[1]
                    + kernels[3]
                    + kernels[4] * 0.114498667676
                    + kernels[5]
                    + kernels[6] * 0.463964735455
                    + kernels[8]
                    + kernels[10]
                    + kernels[11]
                    + kernels[12]
                    + kernels[14] * 0.379362877075
                    + kernels[15]
                    + kernels[16]
```

```
                    + kernels[17]
                    + kernels[18]
                    + kernels[19]
                    + kernels[21]
                    + kernels[23]
                    + kernels[24] * 0.114003314881
                    + kernels[25] * 0.713514944915
                    + kernels[27]
                    + kernels[28] * 0.704354792881
                    + kernels[30]
                    + kernels[31] * 0.299028300538
                    + kernels[33]
                    - kernels[99]
                    - kernels[100]
                    - kernels[101]
                    - kernels[103]
                    - kernels[104]
                    + kernels[106] * -0.671257168502
                    - kernels[108]
                    - kernels[109]
                    - kernels[110]
                    - kernels[113]
                    - kernels[114]
                    - kernels[118]
                    - kernels[119]
                    - kernels[120]
                    - kernels[121]
                    + kernels[123] * -0.440455757257
                    + kernels[124] * -0.561007986563
                    + kernels[125] * -0.753243317389
                    + kernels[126] * -0.913043060342
                    - kernels[127]
                    + kernels[128] * -0.449720343369
                    - kernels[129]
                    ;
                    decisions[3] = 0.620657920277
                    + kernels[34]
                    + kernels[36] * 0.241334460517
                    + kernels[37] * 0.18409640726
                    + kernels[38]
                    + kernels[39] * 0.862077630331
                    + kernels[42]
                    + kernels[43] * 0.779049924353
                    + kernels[44] * 0.204581965894
                    + kernels[45]
                    + kernels[46] * 0.153514588274
                    + kernels[47]
                    + kernels[48]
                    + kernels[49]
                    + kernels[50]
                    + kernels[54] * 0.837980833533
                    + kernels[57] * 0.607789518975
                    + kernels[60] * 0.94737579014
                    + kernels[61]
                    + kernels[62]
                    - kernels[65]
```

```
                    + kernels[71] * -0.843096424242
                    - kernels[73]
                    - kernels[74]
                    + kernels[78] * -0.884037358106
                    + kernels[79] * -0.253398585608
                    - kernels[80]
                    + kernels[81] * -0.00020960447
                    + kernels[82] * -0.83556120158
                    - kernels[84]
                    + kernels[85] * -0.604221251822
                    - kernels[86]
                    + kernels[89] * -0.341850778301
                    - kernels[90]
                    + kernels[91] * -0.523361291848
                    - kernels[92]
                    - kernels[93]
                    - kernels[95]
                    + kernels[96] * -0.5320646233
                    ;
                    decisions[4] = 0.533032568937
                    + kernels[34] * 0.835943113652
                    + kernels[36] * 0.142526185553
                    + kernels[38] * 0.997424637593
                    + kernels[39] * 0.469338126518
                    + kernels[43] * 0.383211101475
                    + kernels[44] * 0.320672027948
                    + kernels[45]
                    + kernels[46] * 0.304469044224
                    + kernels[47]
                    + kernels[50]
                    + kernels[52] * 0.02309036362
                    + kernels[54]
                    + kernels[55] * 0.112453168738
                    + kernels[56] * 0.225162843498
                    + kernels[58] * 0.731773673715
                    + kernels[62]
                    - kernels[97]
                    + kernels[100] * -0.223078591785
                    + kernels[107] * -0.872112998545
                    - kernels[112]
                    - kernels[113]
                    - kernels[119]
                    - kernels[120]
                    - kernels[121]
                    + kernels[123] * -0.210897898371
                    + kernels[126] * -0.880960163315
                    - kernels[127]
                    + kernels[128] * -0.359014634516
                    ;
                    decisions[5] = 0.196913140229
                    + kernels[63]
                    + kernels[64]
                    + kernels[65]
                    + kernels[66]
                    + kernels[67]
                    + kernels[69]
```

```
                    + kernels[70]
                    + kernels[73]
                    + kernels[79]
                    + kernels[80]
                    + kernels[82]
                    + kernels[83]
                    + kernels[87]
                    + kernels[88]
                    + kernels[89] * 0.529620231206
                    + kernels[90]
                    + kernels[91]
                    + kernels[92]
                    + kernels[94] * 0.659512652205
                    - kernels[97]
                    - kernels[98]
                    + kernels[100] * -0.699939038292
                    - kernels[102]
                    - kernels[104]
                    - kernels[105]
                    + kernels[107] * -0.767029891777
                    + kernels[108] * -0.584158909511
                    - kernels[111]
                    - kernels[113]
                    - kernels[115]
                    - kernels[116]
                    - kernels[117]
                    - kernels[119]
                    - kernels[120]
                    - kernels[121]
                    - kernels[122]
                    + kernels[126] * -0.360383513919
                    - kernels[127]
                    + kernels[130] * -0.777621529913
                    ;
                    votes[decisions[0] > 0 ? 0 : 1] += 1;
                    votes[decisions[1] > 0 ? 0 : 2] += 1;
                    votes[decisions[2] > 0 ? 0 : 3] += 1;
                    votes[decisions[3] > 0 ? 1 : 2] += 1;
                    votes[decisions[4] > 0 ? 1 : 3] += 1;
                    votes[decisions[5] > 0 ? 2 : 3] += 1;
                    int val = votes[0];
                    int idx = 0;

                    for (int i = 1; i < 4; i++) {
                        if (votes[i] > val) {
                            val = votes[i];
                            idx = i;
                        }
                    }

                    return idx;
                }

            protected:
                /**
                * Compute kernel between feature vector and support vector.
```

```
                    * Kernel type: rbf
                    */
                    float compute_kernel(float *x, ...) {
                        va_list w;
                        va_start(w, 100);
                        float kernel = 0.0;

                        for (uint16_t i = 0; i < 100; i++) {
                            kernel += pow(x[i] - va_arg(w, double), 2);
                        }

                        return exp(-1.0 * kernel);
                    }
                };
            }
        }
    }
```

这便是SVC在rbf内核下的c++版本的结构，将其导入工程文件夹内。

mozhang.ino改动如下，导入了clf模型，添加了存放输入输出的变量

```
...
#include "classifier.h"
Eloquent::ML::Port::SVM clf;//实例化模型
...
float input[100]={0};//用于跑模型的变量
int pred;//预测结果
...
for (k=0;k<50;k++){
      input[k]=x_x[k];
      input[k+50]=z_x[k];
    }
...
pred=clf.predict(input);
Serial.println(pred);
...
```

```
entry 0x4008059c
MPU6050 Found!
首次按下
进入积分
===START===
0
===END===
首次按下
进入积分
===START===
1
===END===
首次按下
进入积分
===START===
2
===END===
首次按下
进入积分
===START===
3
===END===
```

上电测试一下发现效果太棒了，上下左右对应标签变成了0、1、2、3。不得不说esp32的算力太够了，零延迟跑完模型出预测结果，当然也可能是这个模型比较简单。 此处有视频为证

至此，python脚本可以下岗了。

4 蓝牙传输

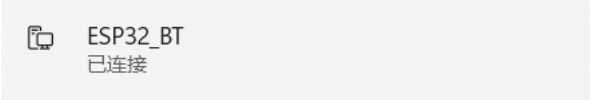
蓝牙传输非常简单，因为esp32自带蓝牙模块与蓝牙库。

4.1 发送端

mozhang.ino的改动如下，主要是引入头文件，初始化蓝牙模块，然后直接发送就行

```
#include "BluetoothSerial.h"
...
BluetoothSerial SerialBT;//定义
...
SerialBT.begin("ESP32_BT");//初始化
...
SerialBT.println(pred);//发送
```

首先以电脑为接收端做实验



电脑端可以发现蓝牙设备并且连接，此时电脑会自动分配一个端口给蓝牙来接收数据（我这里是CMO8）,蓝牙发送的数据默认都是字符串。于是我们可以返场之前的读串口的python脚本进行监测：

```
[*]: import serial
ser=serial.Serial("COM8", 115200)

while True:
    if ser.in_waiting:
        data = ser.readline().decode("utf-8").strip()
        print("收到ESP32数据:", data)
```

```
收到ESP32数据: 0
收到ESP32数据: 1
收到ESP32数据: 2
收到ESP32数据: 3
```

确实可以　　此处有视频为证

```
import serial
import os
ser=serial.Serial("COM8", 115200)

while True:
    if ser.in_waiting:
        data = ser.readline().decode("utf-8").strip()
        print(data)
        if data=="1":
            os.system("start cmd")
```

至此，数据线退休，仅靠电池供电即可完成所有运算与信号传输，魔杖端施工完毕。

4.2 接收端

两个esp32开发板的蓝牙连接与一个板子与电脑的连接是不同的，因为电脑可以可视化手动连接开发板的蓝牙，而开发板之间只能依靠名称或者mac地址，名称已经尝试过了，根本连不上，所以使用mac地址。

vehicle.ino (从机)

```
#include "BluetoothSerial.h"
#include "esp_bt_device.h"

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("vehicle");  //从机名称
  const uint8_t* mac = esp_bt_dev_get_address();
  Serial.printf("ESP32 Bluetooth MAC address: %02X:%02X:%02X:%02X:%02X:%02X\n",
                mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
}


void loop() {
    if (SerialBT.available()) {
      char msg = SerialBT.read();
      Serial.println(msg);
    }
  }
```

为了获取mac地址，调用了esp_bt_device.h库中的函数，因为BluetoothSerial.h里没有。得到从机开发板的mac地址为**28:56:2F:49:A0:C6**

此处因为我板子忘家里了又买了一块，所以mac地址变了

mozhang.ino (主机)

```
#include "BluetoothSerial.h"
#include "esp_bt_device.h"

const int LED_PIN=2;
...

void setup() {
  Wire.begin();
  Wire.setClock(400000);
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  Init_mpu6050();
  pinMode(buttonPin,INPUT_PULLUP);
  SerialBT.begin("ESP32_BT",true);//加true标志此为主机端
  uint8_t macAddr[6]={0x28,0x56,0x2F,0x49,0xA0,0xC6};//纠正为{0x84, 0x1F, 0xE8,
0x15, 0xBE, 0x4E}--9.1

  if (SerialBT.connect(macAddr)) {
    Serial.println("Connected to Slave!");
    for (k=0;k<3;k++) {//蓝牙连接成功提升:灯闪三下
      digitalWrite(LED_PIN, HIGH);
      delay(300);
      digitalWrite(LED_PIN, LOW);
      delay(300);
    }
  } else {
    Serial.println("Failed to connect to Slave.");
    digitalWrite(LED_PIN, HIGH);//蓝牙连接失败提示:灯一直亮
  }
}
...
```

双esp32的连接是主机连接从机，所以启动时应该先开从机的板子，再开主机的板子，为了方便监测是否连接成功，调用了一个led灯作为提示

* LOG（临时抱佛脚是对的）

1.1——1.5+ppt 完成于 8.4——8.10

2.1——2.3 完成于 8.22，8.23

2.4——2.5 完成于 8.24

3.1——3.4 完成于 8.25

3.5——4.2 完成于 8.26