电子系统导论实验报告

实验9 定时计数

指导教师：万景

学生姓名：彭堃	学生姓名：吴磊	学生姓名：徐洋
学 号：22307110109	学 号：22307130218	学 号：20300290037
专　业：保密技术	专　业：技科	专　业：计算机

日　期：2024.04. 28

一、实验目的：

- 了解霍尔码盘的基本原理
- 掌握树莓派定时计数的方法

二、实验原理：

(一) 霍尔效应

- 霍尔效应是电磁效应的一种，是美国物理学家霍尔于1879年在研究金属的导电机制时发现的。
- 当电流垂直于外磁场通过半导体时，载流子发生偏转，垂直于电流和磁场的方向会产生一附加电场，从而在半导体的两端产生电势差，这一现象就是霍尔效应，这个电势差也被称为霍尔电势差。
- 打个比方：好比一条路， 本来大家是均匀的分布在路面上, 往前移动。当有磁场时， 大家可能会被推到靠路的一边行走，导致路的两侧形成流量差。故路 (导体) 的两侧，就会产生电压差。
- 霍尔效应产生的电子流动方向使用左手定则判断。

(二) 霍尔编码器电机

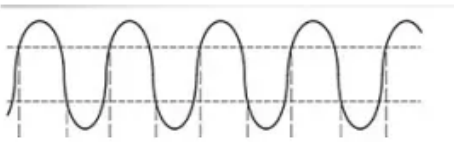
参数：

- 额定电压：DC 6V
- 工作电压：DC 5–13V
- 工作电流：390mA
- 传感器类型：霍尔式
- 减速比：1:45 （电机转45圈，车轮转1圈）
- 分辨率：585脉冲/车轮转1圈

(三) 霍尔编码器测速原理

- 当电机转动时，电机后部的磁体跟随电机一起转动，根据霍尔效应，磁场变化将引起霍尔传感器电压的高低变化
- 该电压变化经过整形后，变成连续的高低电平变化——即方波信号
- 车轮旋转一圈，将产生585个高低交替的脉冲 （右图红框为一个脉冲）
- **可以用手动转一圈来大致确认脉冲数量**
- 通过计算时间*t*内监测到的上升沿数*n*，可以算出车速：$v=\frac{pi*d*n}{585*t}$ （*d*为车轮直径）

霍尔编码器信号



整形后信号

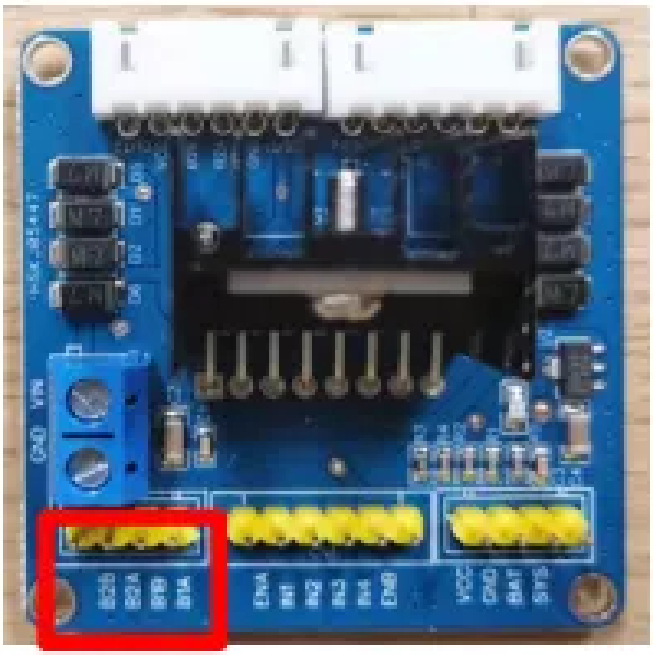
(四) 驱动板引脚补充介绍

引脚介绍

– 和编码器相关的引脚如图红框所示

– 其中B1A、B1B是一个电机的两相输出，B2A、B2B是另一个电机的两相输出，每组中任选一个就可以获取转速。

从左至右为：B2B，B2A，B1B，B1A



- **注意！**
  - B2A、B2B则对应被ENA、IN1、IN2三个管脚控制的电机；
  - 而B1A、B1B是对应ENB、IN3、IN4三个管脚控制的电机速度数据的输出口；
  - 明确对应关系，才能自由调节代码和硬件接线的对应关系。

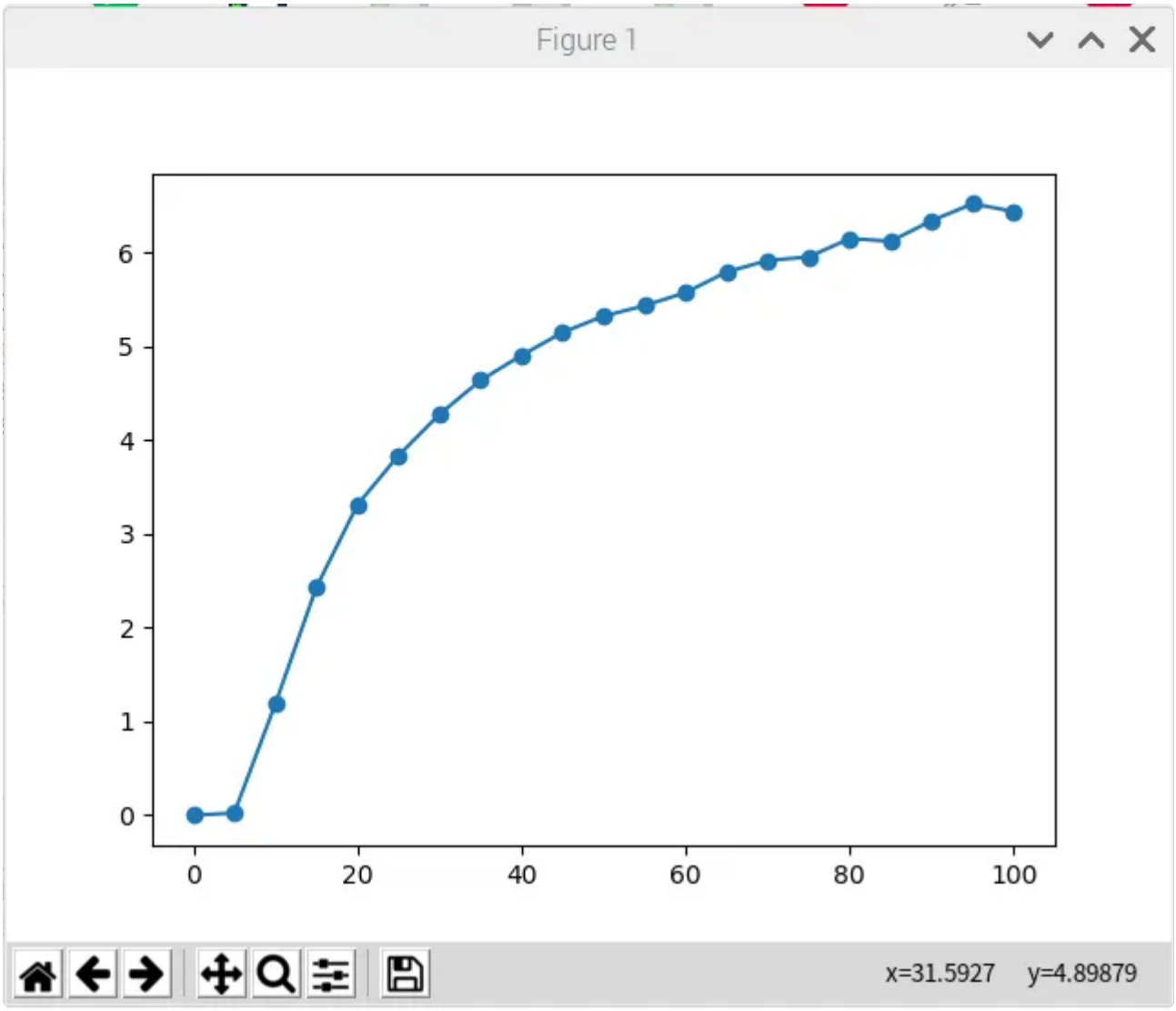
三、实验内容：

(一) 根据参考连接方式连线

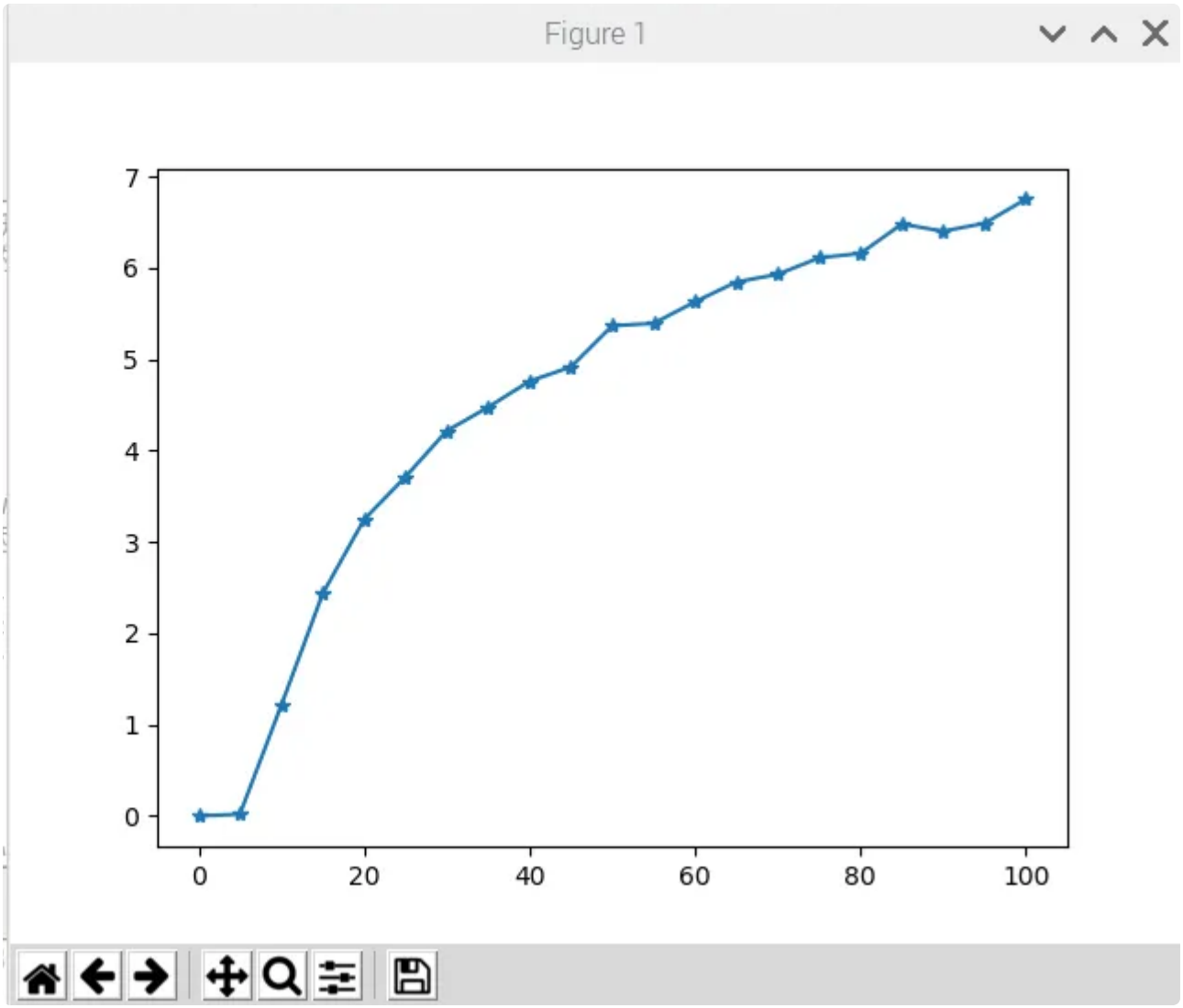
连接B2A、B1A引脚和树莓派GPIO6、GPIO12引脚

(二) 单侧测速绘图

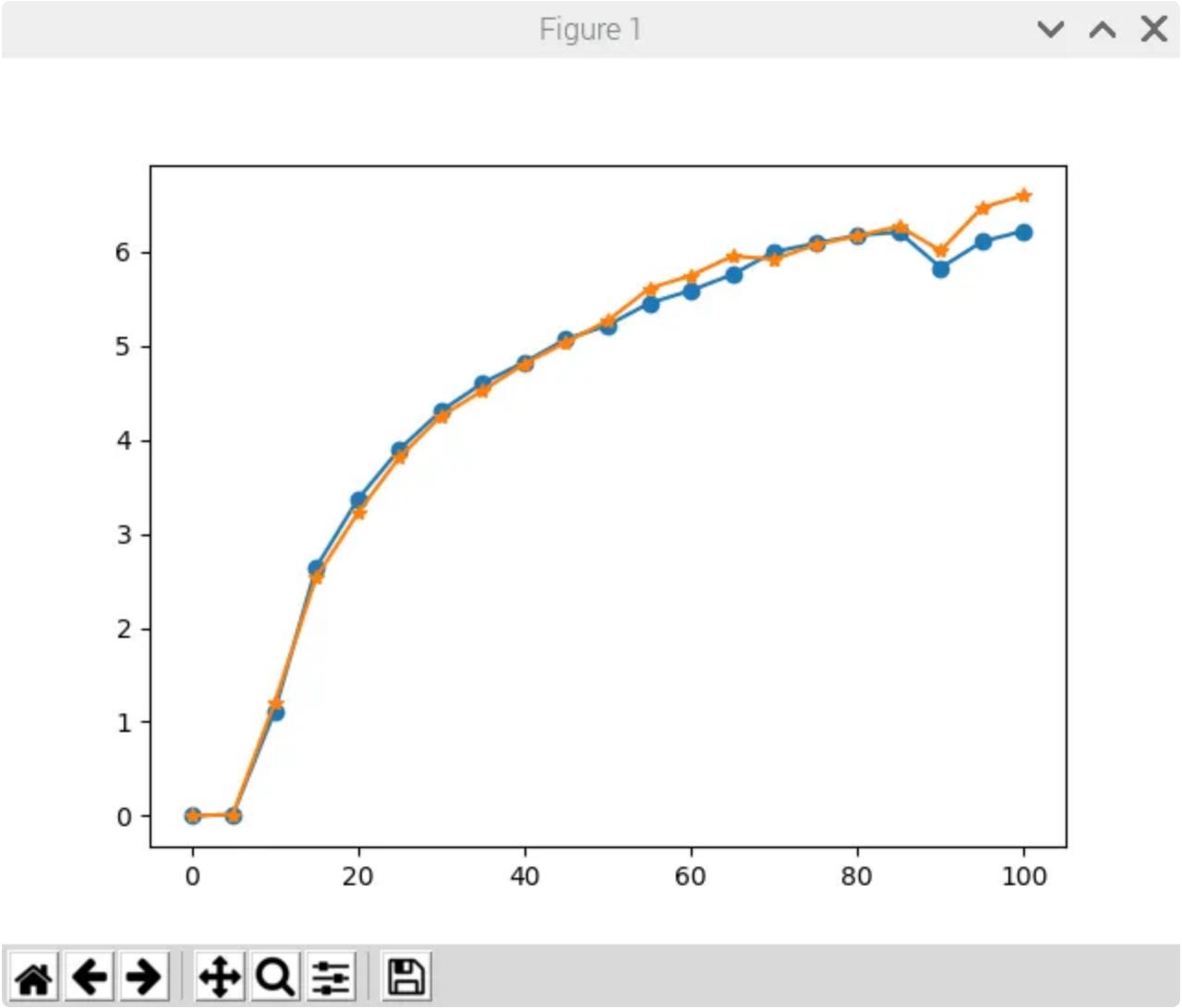
1. 左轮



2. 右轮

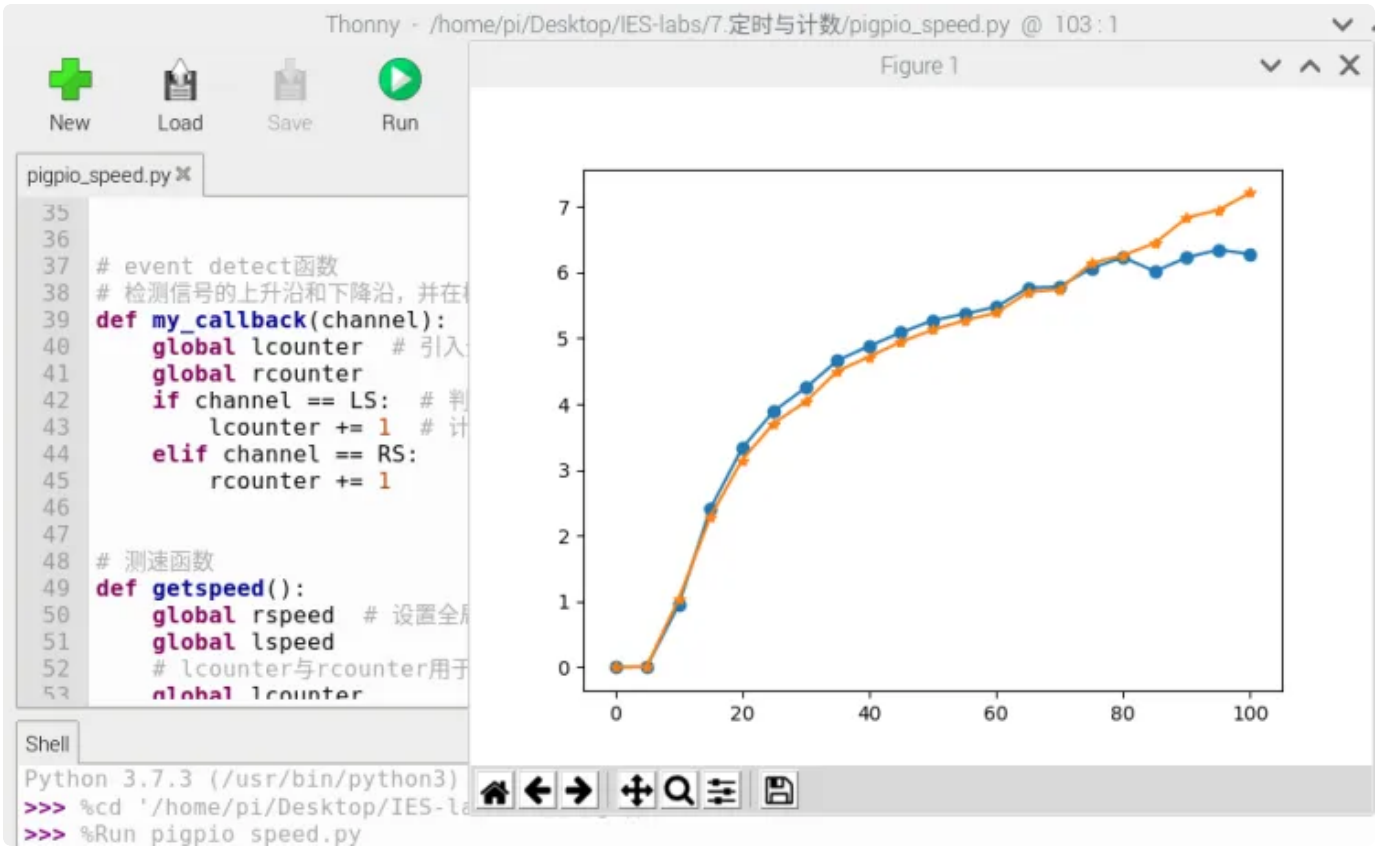


(三) 双侧测速绘图



(四) 使用pigpio改写的代码测试

代码修改逻辑与之前相同，源码附在实验报告zip里


Thonny - /home/pi/Desktop/IES-labs/7.定时与计数/pigpio_speed.py @ 103 : 1
New
Load
Save
Run
pigpio_speed.py
35
36
37 # event detect函数
38 # 检测信号的上升沿和下降沿，并在
39 def my_callback(channel):
40 global lcounter # 引入
41 global rcounter
42 if channel == LS: # 判
43 lcounter += 1 # 计
44 elif channel == RS:
45 rcounter += 1
46
47
48 # 测速函数
49 def getspeed():
50 global rspeed # 设置全
51 global lspeed
52 # lcounter与rcounter用于
53 global lcounter
Shell
Python 3.7.3 (/usr/bin/python3)
>>> %cd '/home/pi/Desktop/IES-la
>>> %Run pigpio_speed.py
Figure 1
0
1
2
3
4
5
6
7
0
20
40
60
80
100

四、实验分析：

(一) 代码分析

speed.py Python

```python
# 库引入
import RPi.GPIO as GPIO
import time # 引入时间库
import threading # 引入线程库
import numpy
import matplotlib.pyplot as plt
# 接口定义与初始化
# 设置各个GPIO口与pwm
EA, I2, I1, EB, I4, I3, LS, RS = (13, 19, 26, 16, 20, 21, 6, 12)
FREQUENCY = 50 # 50Hz
GPIO.setmode(GPIO.BCM)
GPIO.setup([EA, I2, I1, EB, I4, I3], GPIO.OUT)
GPIO.setup([LS, RS],GPIO.IN)
GPIO.output([EA, I2, EB, I3], GPIO.LOW)
GPIO.output([I1, I4], GPIO.HIGH)

pwma = GPIO.PWM(EA, FREQUENCY)
pwmb = GPIO.PWM(EB, FREQUENCY)
pwma.start(0) # 占空比为0
pwmb.start(0)

lspeed = 0
rspeed = 0
lcounter = 0
rcounter = 0

# event detect函数
# 检测信号的上升沿和下降沿，并在检测到边缘时执行线程回调函数
def my_callback(channel): # 定义回调函数
    global lcounter # 引入全局变量
    global rcounter
    if (channel==LS): # 判断是哪个通道触发了回调函数
        lcounter+=1 # 计数器加1
    elif(channel==RS):
        rcounter+=1

# 测速函数
def getspeed():
    global rspeed #设置全局变量lspeed、rspeed，用于向主函数传递电机速度
    global lspeed
    #lcounter与rcounter用于记录从上一次被清零开始，两个霍尔传感器收到了多少个方波
    global lcounter
    global rcounter
    # 添加两个边沿检测，并调回my_callback
    # GPIO.RISING 也可以使用GPIO.FALLING、GPIO.BOTH 对边缘进行检测
```

```
    GPIO.add_event_detect(LS,GPIO.RISING,callback=my_callback)
    GPIO.add_event_detect(RS,GPIO.RISING,callback=my_callback)
    while True:
        #每隔一秒读取一次counter值并转换成速度传递给相应的speed，然后将counter清零。
        rspeed=(rcounter/585.0) # “/585.0”是因为轮子转一圈会有585个脉冲，用“.0”是为了防止speed被自动取整
        lspeed=(lcounter/585.0)
        rcounter = 0
        lcounter = 0
        time.sleep(1)

thread1=threading.Thread(target=getspeed) # 创建新线程
thread1.start() # 启动线程
# 它会不停的统计光电门输入的上升沿，并每隔一秒把全局变量更新为前一秒的速度。单位：圈/秒
# threading没有提供停止线程的方法，关闭图像后可以使用^+z结束程序

i=0
x=[]
y1=[]
y2=[]
while i<=20:
    #主函数每隔3秒增加一次pwm的占空比（本例中步长为5%）。
    #并读取一次新占空比下的两个speed，存入两个数组中。
    x.append(5*i)
    pwma.ChangeDutyCycle(5*i)
    pwmb.ChangeDutyCycle(5*i)
    time.sleep(3)
    y1.append(lspeed)
    y2.append(rspeed)
    i=i+1

# 显示出lspeed与rspeed关于pwm的关系图像。
plt.plot(x,y1,'-o')
plt.plot(x,y2,'-*')
pwma.stop()
pwmb.stop()
GPIO.cleanup()
plt.show()
```

(二) 本次实验为什么需要采用多线程？能否只用单线程完成？

1. 中断法不可避免使用到多线程

首先来说add_event_detect调用回调函数就是在另一个线程里，多线程的作用已经无可替代，因为如果不使用这样的中断方法，而是使用轮询的方法来检测引脚电压变化，cpu的负载将会超出极限，最终很难得到正确的转速，主要的控制逻辑也无法同时执行，即使我们使用pigpio的硬件pwm波来防止软件pwm阻塞cpu，让cpu一心检测引脚电压变化，最后超过3000次变化每秒的速度也会让cpu无法正确得到结果，毕竟设置的轮询周期和实际执行的轮询周期完全没有关系，python的运行效率十分感人。

2. 区间测速和多线程导致的误差分析和取舍

因为计算机测速必然需要一定的时间间隔，比如原始的实现中，便是每隔1秒查看一次lcounter和rcounter，计算速度并置零，而lcounter和rcounter的增加是通过两个event_detect实现的，这两个event_detect将会不断开启新线程，每个线程都会尝试将其中一个counter加1，虽然说多线程可以在一定程度上让程序同时执行，但是无论如何cpu一个核只能运行一个机器码，树莓派的cpu就4个核，所以很难达到理想的counter反馈，更别说python不会使用这么多核，还有线程锁，所以在内存层面上，counter的增加是有滞后效应的，因为我们还需要执行主函数和其他部分的一些逻辑。同时我们还有一个计时逻辑，表面上计时结束，后面的语句就该执行了，但实际上每个线程都在争夺cpu的使用权，如果操作系统想要callback函数先运行，那么sleep的时间就会偏长，主函数被阻塞，如果操作系统先执行主函数，就会导致部分callback函数未执行，得到的结果偏小，如果在主线程等callback函数运行，就等同于主函数被阻塞，得不偿失。

提供的代码便是优先主线程，因为测速逻辑在另一个线程，callback函数的线程也会优先阻塞另一个线程，主线程相对更独立。

3. 单线程的可能性

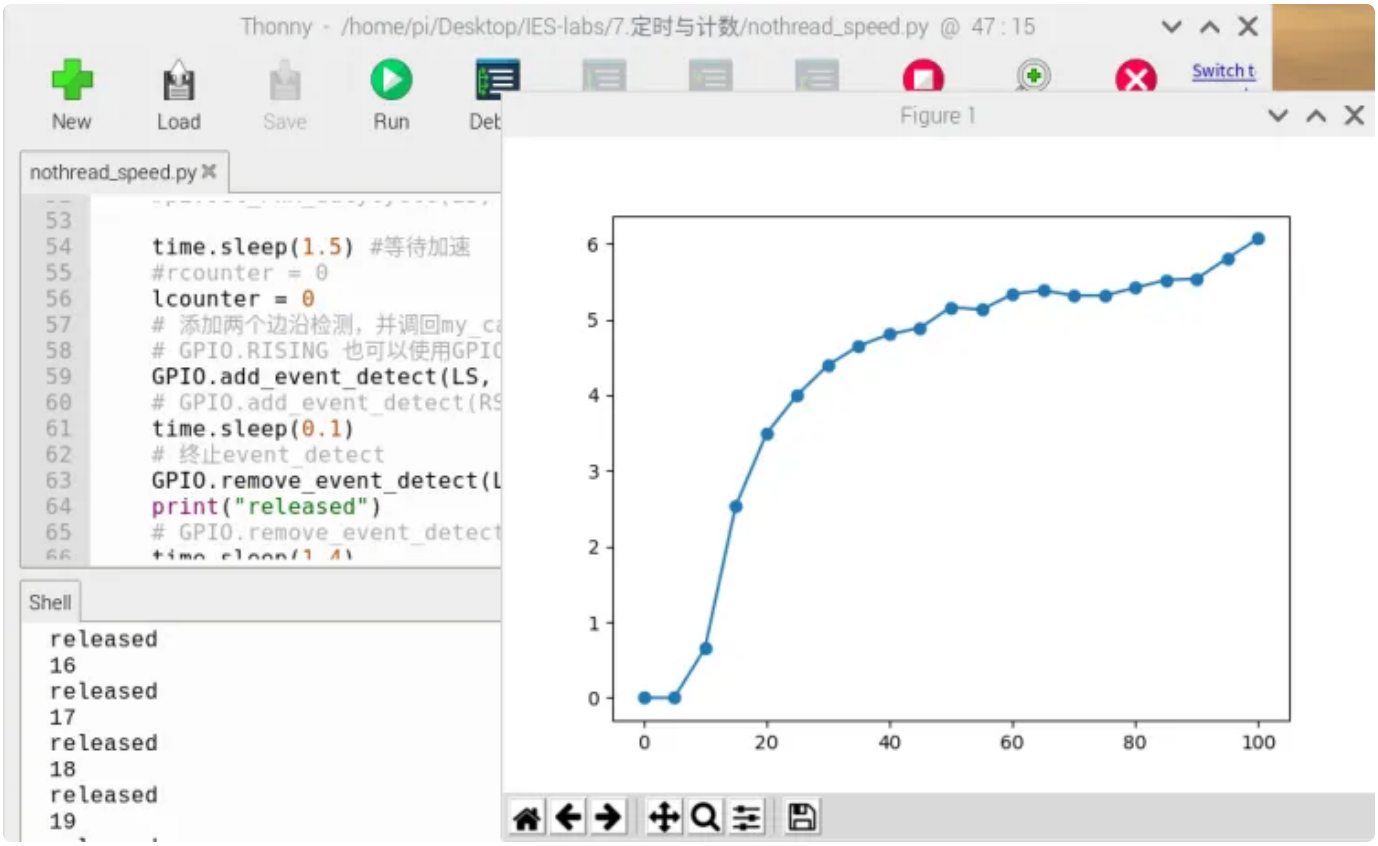
首先排除纯单线程（轮询法），如果追求纯粹的单线程肯定实现不了，我们暂定可以使用中断法的多线程。

（这样的好处是我们可以抽象出一个测速函数，需要时就调用，耗费0.1s就能得到一个相对准确的速度值。）

于是我们可以构造这样一种（不纯的）单线程设计：

先让小车加速，等待小车加速完成后开启一个event_detect,一秒后将其关闭，然后等待所有callback线程执行完毕，最好计算速度并导入

这样的设计看起来很好，但实际上会出现一个暂时无法解决的内存问题，结果如下



可以看到基本逻辑是

- sleep（1.5）等待加速
- counter归0
- 启动event_detect，保持0.1秒后关闭
- 等待callback线程执行完毕，计算速度并导入

这里有两个细节让程序得以正常运行：

- 一是关闭了另一个轮子的所有逻辑，这样阻塞少一半
- 二是减短了event_detect的时间，这样callback线程少非常多

如果维持原本的callback函数量，将会很容易得到一个段错误（segementation fault），我个人认为有两个可能，一个是单纯的内存不够，因为关闭一个轮子的逻辑，程序就能多跑一点再出现错误，另一个是counter的更新可能有问题，debug下几乎只有在访问counter时程序才会崩溃，至于究竟如何就得引入一些工具或者查看底层逻辑了，但经过一番调整，我们也确实得到了一个结果图像。

不难发现在转速较低的条件下，这样的单线程设计几乎没有什么问题，转速较高的情况下误差其实也有限，主要问题还是程序会出现段错误，这个不引入特殊工具是debug不了的，不会显示在第几行出错，最后只能在一些区域一行加一个debug语句来实现粗略的定位，后续如果研究透了也许也挺有价值（挖个坑）

总得来说如果只是想要搞这么一个图像，这样的做法倒也不是不行，但是如果是要得到一个差不多的即时速度，这种方法还是别用了，对主线程影响过大，如果要追求纯单线程，我觉得还是洗洗睡吧

4. 多线程的优势

从提供的代码就能看出，我们采取每3秒增加一次转速，每1秒测定一次速度，就是为了在提取速度时能够得到相对正确的速度值，一般来说我们提高pwm波频率，转速还需要一段时间提升，所以前面一段肯定不能用，我们睡3秒在提取速度，由于全局速度总是上一秒的速度，所以我们大概率得到的是第二秒的速度，这样误差比较小。

当然更重要的是未来如果我们需要有一个长期的速度参照，肯定不能是包装成一个函数，放在主线程里面需要就去调用然后测一下，这样会阻塞主线程，专门设计一个线程检测和更新速度值对未来的整体性设计非常有益。其对cpu的消耗不算很大（之前的方法失败是因为内存安全性问题），但是对cpu的阻塞不可忽略，我们将其放在另一个线程里，对主线程也就是控制线程的影响相对更小，这样的整体逻辑更加合理。

(三) 如果发现轮子不转，如何分步排查故障?

我们先查看影响轮子转动的几个因素：

1.树莓派GPIO端口与电机连线是否稳固，接口有没有错接漏接；

2.代码的PWM与GPIO端口设置有误导致PWM并没有连接到正确的GPIO端口上，接受不到信号；

2.初始设置了轮子的占空比为0，代码后续每隔三秒增加占空比步长是否正常运行，可以加入一个函数用于控制台打印观察；

3.电机是否正常工作，观察电机有没有处于正常的工作状态有无异响。

(四) 如果发现两个轮子转速差异很大，如何分步确定原因?

影响轮子转速不同的几个因素：

1.代码是否正确设定两个电机的占空比同步改变，如果代码中出现设定一个电机占空比增长而另外一个电机没有增长会出现轮子转速差异较大，通过增加代码使用控制台打印信息观察两个电机占空比改变状态；

2.在相同占空比下，两个电机的转速功率也会有差异，若差异过大，导致其中一轮的转速过大，可以设定另一个电机占空比增大以实现两轮转速相近。

五、总结与思考：

1.本次实验运用了霍尔效应对霍尔编码型电机驱动下的轮子进行测速，解决了对小车速度测量的问题，为之后小车的自动控制速度和方向提供了速度的精确值。

2.在问题探讨中，尝试探讨了单线程方法的可行性，也给出了多线程方法的优点，即逻辑清晰、对主线程干扰小、执行效率高。