电子系统导论实验报告

期末考核 1 折线竞速

指导教师：冯辉

学生姓名：李航成　学生姓名：卢睿健　学生姓名：姚宇轩

学　　号：22307130449　学　　号：22307130091　学　　号：22307130097

专　　业：技术科学试验班　专　　业：技术科学试验班　专　　业：技术科学试验班

日　　期：2023 年 6 月 4 日

一、 实验目的：

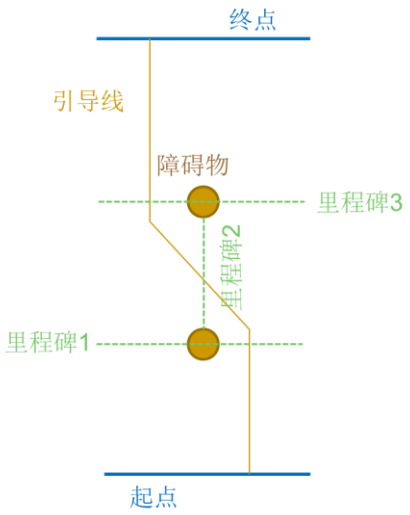
基于树莓派、小车，综合运用本学期所学知识，紧密联合硬件与软件，自行设计算法，完成期末考核的“折线竞速”项目。并经充分调试寻求最佳参数，不断提高小车前行速度、提高小车前行时的稳定性，减少“折线竞速”总用时。

从而加深对电子系统的认识，提升基础电子系统的设计能力。同时培养发现问题、分析问题、解决问题的能力与团队合作的能力与精神。

“折线竞速”规则和示意图如下：

在赛道中间设置障碍物，提供地面引导线。通过超声、摄像头等传感器完成绕桩前行，不允许人工干预小车前进。

要求：静止状态从起点线出发，经过第一个障碍物右侧，第二个障碍物左侧，最终到达终点线（不一定需要严格循线走）



二、 实验原理：

(一)　实现使小车“巡线而行”的基本处理思想

1. 方案一：“定时转向”

本实验的评判准则为使小车以尽可能少的时间从起点绕过障碍物到达终点。因此，若可基于小车自身性能，精准把握好两个“转弯时刻”，便可取得很好的效果。

2. 方案二：“图像判断”

考虑让使用摄像头实时捕获图像，利用图像处理技术**判断小车是否偏离给出的“引导线”，从而控制小车的转向，达到“巡线而行”的效果**。

当然，经实践，我们选择了“图像判断”的方案二（具体请见后文）。

(二)　用作实现核心的课程知识

- 第八次实验（4/24）自动控制中所学的 **PID 控制方法**，可控制小车在加速与高速行进时保持直线行进，也可控制转向。（当然，这一处理本身建立在第六

次实验（4/10）PWM 与直流电机驱动和第七次实验（4/18）定时与计数基础上）

- 第九次实验（5/8）图像与视频接口中所学的 **OpenCV 图像处理**，可借以获得图像不同色彩空间（RGB, YCrCb）的像素情况，并可对图像进行二值化、滤波等处理。

三、 实验内容：

(一)　“定时转向”加 PID 控制的尝试

在最开始的阶段，我们的思路是定时转向，直线段速度拉满往前冲，完全不需要用到图像处理，最关键的就是让小车直线段跑直。

我们在 PID 控制程序的基础上完成了 PID 计时折线跑程序的初版代码，并且进行了两个夜晚的长时间调试。**但是出现了一些难以消除，甚至不可控的问题**：

1. 地毯不能做到完全平整，小车的直线行驶会受到许多外界因素的影响，并且难以用 PID 弥补，直线行驶效果不可控。这是最最主要的原因。
2. 使用 PID 控制转向，但是我们发现 PID 控制转向会附带一些不希望的效果，其中最主要的就是在控制一边转速降低完成转向之后，PID 调整转速回升到直线行驶转速时，会带来一个**过调效果**。直接导致小车转向到下一段直线时会出现“往回打一把”的**回正现象**。

   我们反复调整 PID 参数，并且尝试了转向预减速、直线预加速等过渡措施，也难以完全消除这种回正的问题，而这会为下一段直线行驶的方向带来不可控的误差。

   （问题出现的具体原理见实验分析）
3. 因为前两段的直线和转向因为刚刚提到的两个问题，第三段直线的行驶方向会出现或大或小的偏差。这个偏差有一定概率导致小车上墙，并且这个偏差因为受外界因素影响，难以严格控制，这就为实验带来了不可忍受的不可控因素。

最终，经综合考虑，我们一致认为，PID 控制计时转向的策略存在的不可控问题不能被接受，这个策略不能作为第一选择。于是我们转向了对**图像处理巡线方法**的尝试。

（被放弃的计时策略程序可见本文最后及附件）

(二)　“图像判断”的算法设计和代码构建

我们同样在 PID 控制程序的 GPIO 设定基础上，添加了图像的获取和处理判断转向的算法，之后我们放弃了 PID 控制转向。而是直接调整占空比来改变转速，有效消除回正问题。

1. 核心处理过程一：图像偏离引导线程度的计算

（1） 基本思想

设法**识别出图像中的引导线**，并取其某一行的**中心位置，将其与整个图像的中心像素位置进行比较**，即可判断当前小车偏左还是偏右，**并将二者的差值作为偏离程度**。

```
            # 获取引导线中心
            cur_center = calculate_center(frame)
            # 计算偏离中心的程度，其中 sym_axis 是图像竖直对称轴位置
            delta = cur_center - sym_axis
```

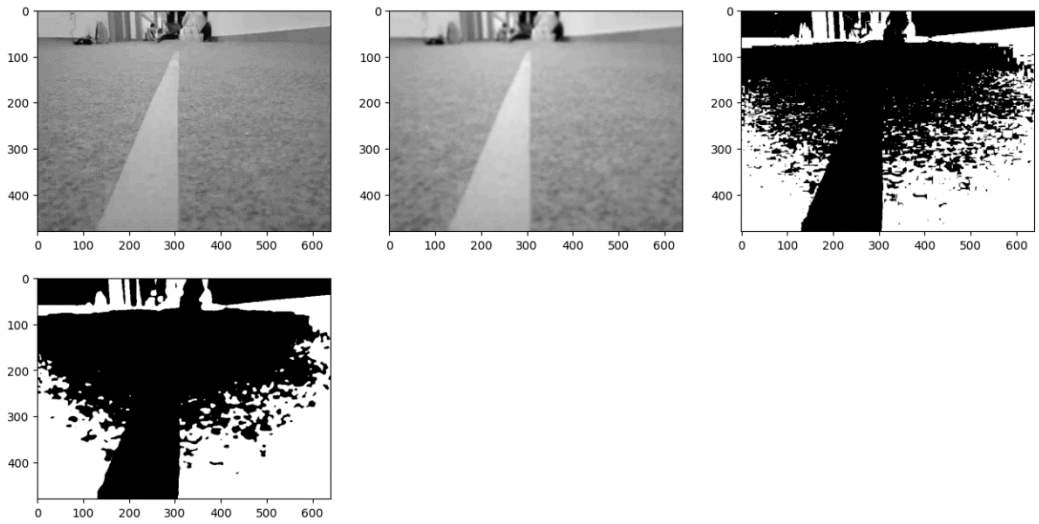
（2） 几个实现部分的代码构建和优化过程

a） 图像获取及黑白二值化

- 初始的尝试：直接对 BGR 色彩空间的图像取灰度

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gauss_gray = cv2.cvtColor(guass_k9, cv2.COLOR_BGR2GRAY)

    thres = 117 # to be changed when needed
    retval, bin = cv2.threshold(gray, thres, 255, cv2.THRESH_BINARY_INV)
    retval_g, bin_g = cv2.threshold(gauss_gray, thres, 255,
cv2.THRESH_BINARY_INV)
```

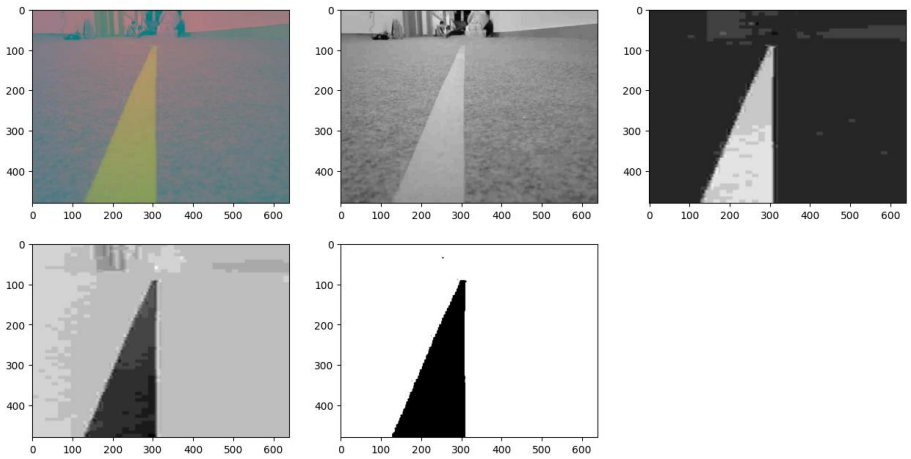


如图，在最开始，我们的图像处理效果非常差，难以得到预期的正确判断结果。图像有大量噪点，之后在回过头重读课程 PPT 时突然意识到，我们当前使用的是直接将 RGB 图像转换为灰度图像再进行二值化的方法，但如果先将 RGB 转换为 YCrCb 色彩

空间，再提取其中的 Cr 通道可能会改善图像效果。请见下面

- 在 YCrCb 色彩空间下取 Cr 通道的结果
  - 这样的优势：直接考察某一色彩通道，大幅减少噪点，且能减弱亮度的影响
  - 为什么提取 Cr 通道：引导线是橙红色的

```
# 转化至 YCrCb 色彩空间
imgYCrCB = cv2.cvtColor (img , cv2.COLOR_BGR2YCrCb)
# 提取 Cr 通道下的结果，据此进行黑白二值化
Y, Cr, Cb = cv2.split(imgYCrCB)
```

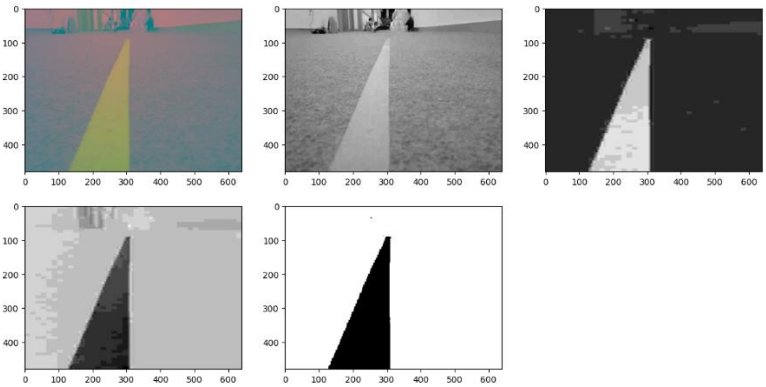


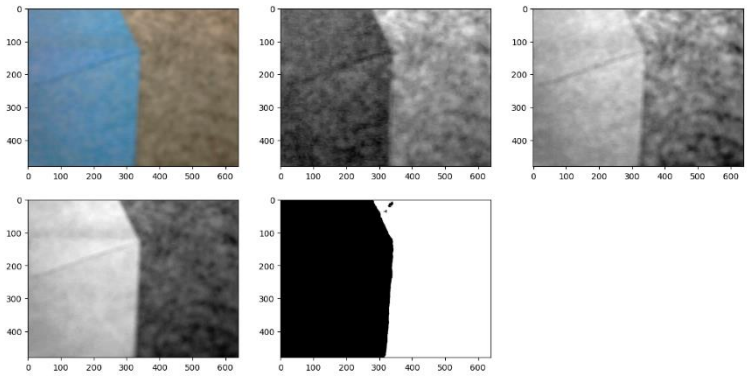
如图，结果出乎意料的好，二值化得到的直线十分光滑，边缘十分清晰，几乎没有噪点。

b） 二值化阈值的斟酌

```
thres = 133 # to be changed when needed
retval, binCr = cv2.threshold(Cr, thres, 255, cv2.THRESH_BINARY_INV)
```

（接以上代码）显然，二值化需要设定阈值，经我们仔细严谨的尝试，最终判断取 **133** 最佳（实际上区别不大，但正所谓“精益求精”）。





c） “扫描一行像素，计算中间点”的代码构建

- 初始想法：由于一开始的图像处理方法有噪点，因此考虑将这一行的所有黑点区间取出来，挑选其中最长的区间，将其视作引导线

```
def get_longest_interval(row, aimcolor, size = 640):
    i = 0
    j = 0
    mxlen = 0
    resl = -1
    resr = -1
    while i != size:
        if row[i] == aimcolor:
            j = i
            while j != size and row[j] == aimcolor:
                j += 1
            # print(aimcolor, "interval: ", i, j - 1)
            if j - i > mxlen:
                mxlen = j - i
                resl, resr = i, j
            i = j
        else:
            i = i + 1
    return resl, resr
def calculate_center(img):
    # ……（前面略）
    # 取某一行的结果，扫一遍计算引导线的中心
    row = binCr[195]
    wl, wr = get_longest_interval(row, 0)

    if wl == -1:
        return sym_axis
    return (wl + wr) / 2
```

- 后来改进了图像处理，大幅减小噪点出现的可能。但显然上述算法仍适用，而且可应对一些“意外”。

2. **核心处理过程二：**基于以上结果控制小车转向

给出最终实现方式：（原因见下）

```
# 经实验，认为应直接将 speed 即占空比拉满，从而增加小车稳定性
speed = 100 # equivalent to PWM's duty
```

```
# ……（略）
# note that speed is 100
def TurnLeft(par):
    print("Turning Left!")
    nspeed = speed + par / 320.0 * sensibility
    if nspeed > 100.0:
        pwmb.ChangeDutyCycle(100.0)
        pwma.ChangeDutyCycle(speed*((320-par)/320))
    else:
        pwmb.ChangeDutyCycle(nspeed)
        pwma.ChangeDutyCycle(speed)

''' 右转指令 '''
def TurnRight(par):
    print("Turning Right!")
    nspeed = speed + par / 320.0 * sensibility
    if nspeed > 100.0:
        pwma.ChangeDutyCycle(100.0)
        pwmb.ChangeDutyCycle(speed*((320-par)/320))
    else:
        pwma.ChangeDutyCycle(nspeed)
        pwmb.ChangeDutyCycle(speed)
```

(三)　“图像判断”的实地运行调参

实地调参的过程中，我们对参数改进主要包括：转向判断阈值的优化、转向函数的算法优化以及图像处理所取行位置的优化。

1. 阈值的优化即为不断重复尝试，最终选择效果最优的阈值，不做赘述。
2. 关于转向函数的算法优化，我们总的来说进行了从占空比的增减式改变模式优化为比例系数相乘的改变模式。
3. 图像处理算法的优化主要是改变转向判断算法所选取的像素位置（即行数）。

（具体优化思想见实验分析相应部分）

四、 实验分析：

（一）PID 尝试中出现“方向回调”问题的可能原因

由于小车每一段直行由 P, I, D 三个参数控制，其中 P 代表直接的电机转速的调整。当小车经过一段差速转弯进入下一段直行时，两轮仍然处于差速状态，所以无论目标速度和之前转向时内侧轮还是外侧轮转速相近，都会使得另一侧轮子的当前速度和目标速度之间存在很大差异（以内侧轮减速转弯为例），那么 PID 的控制就会使得该侧电机转速突然增大并且大于另一侧电机转速（其中，P 控制直接的转速改变至大于目标速度，I 的积分控制则会使得接下来一小段时间内转向时的内侧轮速度一直大于目标速度），所以在这一段时间内，小车会出现方向回调的现象，且很难消除。

（二）图像巡线改进的分析

1. 转向阈值

转向阈值，即拍摄的图像经过处理后白色区域中心与图像中心差值超过多少时开始调节电机转速。

理论分析并经过实验验证后我们发现：阈值设置过大，则小车偏离原有行驶方向太多才开始调整，**往往调整幅度难以足够，会经常导致小车彻底离开线路**；阈值设置过小，则小车过于频繁地调整方向，**会造成小车前进时“抽搐”**，且严重影响整体速度，因为小车调节方向是通过一边电机减速实现的，过多调整使得电机整体速度大幅下降。

2. 转向函数的优化调整

我们尝试过一下几种调节转向使得小车能够巡线的方法：

（1）两轮占空比原始值固定，图像偏离中心超过阈值后把偏离的值乘上一个系数，加到一边电机的占空比上。存在的问题：要不有时占空比的调整值会大于 100，报错；要不就调整幅度过小，无法保持在线上。

（2）两轮占空比原来为 100，图像偏离中心超过阈值后把偏离的值乘上一个系数，减到一边电机的占空比上。存在的问题：要不有时占空比的调整值会小于 0，报错；要不就调整幅度过小，无法保持在线上。

（3）两轮占空比原来为 100，图像偏离中心超过阈值后，把偏离的值比上最大偏离值（320 像素），得到一个比例，对一边电机的占空比乘上这个比例，以减小该侧电机转速（如下）。之前的问题基本解决！

```
        pwmb.ChangeDutyCycle(100.0)
        pwma.ChangeDutyCycle(speed*((320-par)/320))
```

3. 图像处理选取行数以及优化效率的想法

（1）处理图像后选取某一行进行分析，这个选取的行所在的位置是提升速度的关键。位置越靠上（即取值越小），越能提前判断偏移，但是对图像处理清晰度的要求就越高，因为白色可能只占很少的一部分；位置靠下，则处理比较稳定，但是判断较为迟缓。最终考试时，我们先用 70，再改为 50，最后改为 10，成绩逐次得到提升。

（2）原本的策略是先处理整个图像，在从中取一行分析，我们认为这样对树莓派的运算要求较高，所以有如下思路：先选定某一行，然后再只对这一行进行图像处理。这样，理论上运算量少了两个数量级，在实验中也得到了更快的小车反馈。

五、 考核结果：

- 最快用时：7.54（还是 7.55）秒

六、 总结与思考：

我们独立书写代码，尤其是最重要的“计算中心”函数，高质量解决图像处理算法问题。

同时，在完成这一项目的过程中，我们经历了**方法问题、两次电机故障、数次 GPIO 接口故障、多次调车未完成时便电压不足**等重重困难。

但是在数个夜晚之后，我们很荣幸地拿出了一台以巡线方法跑出理论极限速度的作品，我们认为之前的全身心的投入，乃至说“挣扎”都是值得的。

最后，非常感谢助教老师和老师和老师支持！

【附】代码

- “巡线”（最终使用的，含注释）

```
import RPi.GPIO as GPIO
import time
import threading
import numpy
import matplotlib.pyplot as plt
import cv2
import numpy as np
import math

''' GPIO 和 PWM 初始化 '''
EA, I2, I1, EB, I4, I3 = (16, 19, 26, 13, 20, 21) # changed
FREQUENCY = 100

GPIO.setmode(GPIO.BCM)
GPIO.setup([EA, I2, I1, EB, I4, I3], GPIO.OUT)
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)
pwmb.start(0)

''' 摄像头初始化 '''
cap = cv2.VideoCapture(0)

# 其中 sym_axis 是图像竖直对称轴位置，摄像头默认参数横向宽度是 640
sym_axis = 640 / 2
# 经实验，认为应直接将 speed 即占空比拉满，从而增加小车稳定性
speed = 100 # equivalent to PWM's duty
```

```
cmd = 0
ret, frame = cap.read()
# cv2.imshow('diplay', frame)
# cv2.imwrite("2.jpeg", frame) 调试代码，输出图片进行检查
print('Ready!')

''' 前进指令 '''
def Straight():
    pwma.ChangeDutyCycle(speed)
    pwmb.ChangeDutyCycle(speed)
    print("Go Straight")

# 决定敏感程度，可见后面两个函数
sensibility = 70

''' 左转指令 '''
# changed by lhc !
# suppose that pwma and pwmb is for the left and right respectively
# note that speed is 100
def TurnLeft(par):
    print("Turning Left!")
    nspeed = speed + par / 320.0 * sensibility
    if nspeed > 100.0:
        pwmb.ChangeDutyCycle(100.0)
        pwma.ChangeDutyCycle(speed*((320-par)/320))
    else:
        pwmb.ChangeDutyCycle(nspeed)
        pwma.ChangeDutyCycle(speed)

''' 右转指令 '''
def TurnRight(par):
    print("Turning Right!")
    nspeed = speed + par / 320.0 * sensibility
    if nspeed > 100.0:
        pwma.ChangeDutyCycle(100.0)
        pwmb.ChangeDutyCycle(speed*((320-par)/320))
    else:
        pwma.ChangeDutyCycle(nspeed)
        pwmb.ChangeDutyCycle(speed)

def Stop():
    pwma.stop()
    pwmb.stop()
    GPIO.cleanup()
    print('Out of the Way')

# 获取最长黑色区间，将其视作图像这一行的引导线位置
def get_longest_interval(row, aimcolor, size = 640):
    i = 0
    j = 0
    mxlen = 0
    resl = -1
    resr = -1
    while i != size:
        if row[i] == aimcolor:
            j = i
            while j != size and row[j] == aimcolor:
```

```
                j += 1
            # print(aimcolor, "interval: ", i, j - 1)
            if j - i > mxlen:
                mxlen = j - i
                resl, resr = i, j
            i = j
        else:
            i = i + 1
    return resl, resr

# img is 480*640 2D-array
def calculate_center(img):
    # 转化至 YCrCb 色彩空间
    imgYCrCB = cv2.cvtColor (img , cv2.COLOR_BGR2YCrCb)
    # 提取 Cr 通道下的结果，据此进行黑白二值化
    Y, Cr, Cb = cv2.split(imgYCrCB)

    # 二值化时针对像素的阈值，经实验认为 133 最佳
    thres = 133 # to be changed when needed
    retval, binCr = cv2.threshold(Cr, thres, 255, cv2.THRESH_BINARY_INV)

    # 取某一行的结果，扫一遍计算引导线的中心
    # 经实验取第 195 行最理想
    row = binCr[195]
    wl, wr = get_longest_interval(row, 0)

    if wl == -1: # 防“意外”
        return sym_axis
    # 返回区间中点
    return (wl + wr) / 2

# cur_center = calculate_center(frame)
# print(cur_center)

cmd = 0

try:
    while (1):
        if (cmd == 0):
            cmd = input()
        else:
            ret, frame = cap.read()  # read one fps
            # cv2.imshow("display", frame)
            # 获取引导线中心
            cur_center = calculate_center(frame)
            # 计算偏离中心的程度，其中 sym_axis 是图像竖直对称轴位置
            delta = cur_center - sym_axis
            print("cur_center=%d, delta=%d" % (cur_center, delta))

            # 对于偏离中心是否要转向的阈值，经实验认为取 10 最佳
            thres = 10
            # 若偏离中心在这一范围内，则不控制转向
            # 否则相应地向右或向左转向
            if (delta < thres and delta > -thres):
                Straight()
            elif (delta > thres):
                TurnRight(delta)
            elif (delta < -thres):
```

```
                TurnLeft(-delta)
            time.sleep(0)
except KeyboardInterrupt:
    pass

cap.release()  # 释放摄像头
cv2.destroyAllWindows()  # 关闭所有显示窗体
pwma.stop()
pwmb.stop()
GPIO.cleanup()
```

- 一开始尝试的“定时转向”：

```
import RPi.GPIO as GPIO
import time
import threading
import numpy
import matplotlib.pyplot as plt


EA, I2, I1, EB, I4, I3, LS, RS = (16, 19, 26, 13, 20, 21, 22, 27)
FREQUENCY = 100
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)
pwmb.start(0)

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

class PID:
    """PID Controller
    """

    def __init__(self, P=60, I=0, D=0, speed=0.4, duty=26):

        self.Kp = P
        self.Ki = I
        self.Kd = D
        self.err_pre = 0
```

```
        self.err_last = 0
        self.u = 0
        self.integral = 0
        self.ideal_speed = speed
        self.last_duty = duty
        self.pre_duty = duty

    def update(self,feedback_value):
        self.err_pre = self.ideal_speed - feedback_value
        self.integral+= self.err_pre
        self.u = self.Kp*self.err_pre + self.Ki*self.integral +
self.Kd*(self.err_pre-self.err_last)
        self.err_last = self.err_pre
        self.pre_duty = self.last_duty + self.u
        if self.pre_duty > 100:
            self.pre_duty = 100
        elif self.pre_duty < 0:
            self.pre_duty = 0
        self.last_duty = self.pre_duty
        return self.pre_duty

    def setKp(self, proportional_gain):
        """Determines how aggressively the PID reacts to the current
error with setting Proportional Gain"""
        self.Kp = proportional_gain

    def setKi(self, integral_gain):
        """Determines how aggressively the PID reacts to the current
error with setting Integral Gain"""
        self.Ki = integral_gain

    def setKd(self, derivative_gain):
        """Determines how aggressively the PID reacts to the current
error with setting Derivative Gain"""
        self.Kd = derivative_gain



def my_callback(channel):
    global lcounter
    global rcounter
    if (channel==LS):
        lcounter+=1
    elif(channel==RS):
        rcounter+=1
```

```
def getspeed():
    global rspeed
    global lspeed
    global lcounter
    global rcounter
    GPIO.add_event_detect(LS,GPIO.RISING,callback=my_callback)
    GPIO.add_event_detect(RS,GPIO.RISING,callback=my_callback)
    while True:
        rspeed=(rcounter/585.0)
        lspeed=(lcounter/585.0)
        rcounter = 0
        lcounter = 0
        time.sleep(0.1)


thread1=threading.Thread(target=getspeed)
thread1.start()


l_origin_duty = 5
r_origin_duty = 5
pwma.start(l_origin_duty)
pwmb.start(r_origin_duty)


try:

#第一段

    #第一段直行
    i = 0
    #第一段直行速度控制
    speed = 0.55
    L_control = PID(60,0.06,80,speed,l_origin_duty)
    R_control = PID(60,0.06,80,speed,r_origin_duty)
    #第一段直行时间
    t = 3
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1

    #转弯 1
```

```
    #转弯预减速
    '''i = 0
    speed = 0.3
    L_control = PID(60,0.06,80,speed,l_origin_duty)
    R_control = PID(60,0.06,80,speed,r_origin_duty)
    #直行时间
    t = 0.5
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1
    i = 0'''

    #第一次转弯速度控制
    Lspeed = 0.55
    Rspeed = 1
    L_control = PID(60,0.06,80,Lspeed,l_origin_duty)
    R_control = PID(60,0.06,80,Rspeed,r_origin_duty)
    #第一次转弯持续时间
    t = 1
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1

#第二段

    #第二段直行预加速
    i = 0
    #第二段直行预加速速度控制
    speed = 0.55
    #L_control = PID(0,0.06,80,speed,l_origin_duty)
    R_control = PID(10,0.06,10,speed,r_origin_duty)
    #第二段直行预加速时间
    t = 0.5
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1

    #第二段直行
```

```
    i = 0
    #第二段直行速度控制
    speed = 0.55
    L_control = PID(60,0.06,80,speed,l_origin_duty)
    R_control = PID(60,0.06,80,speed,r_origin_duty)
    #第一段直行时间
    t = 3
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1


    #第二次转弯预减速
    '''i = 0
    speed = 0
    L_control = PID(60,0.06,80,speed,l_origin_duty)
    R_control = PID(60,0.06,80,speed,r_origin_duty)
    #直行时间
    t = 0.5
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1'''

    #第二次转弯速度控制
    i=0
    Lspeed = 1
    Rspeed = 0.55
    L_control = PID(60,0.06,80,Lspeed,l_origin_duty)
    #R_control = PID(60,0.06,80,Rspeed,r_origin_duty)
    #第二次转弯持续时间
    t = 1
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1

#第三段

    #第三段直行预加速
```

```
    i = 0
    #第三段直行预加速速度控制
    speed = 0.55
    L_control = PID(0,0.06,80,speed,l_origin_duty)
    #R_control = PID(10,0.06,10,speed,r_origin_duty)
    #第三段直行预加速时间
    t = 0.5
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1


     #第三段直行
    i = 0
    #第三段直行速度控制
    speed = 0.55
    L_control = PID(60,0.06,80,speed,l_origin_duty)
    R_control = PID(60,0.06,80,speed,r_origin_duty)
    #第三段直行时间
    t = 3
    while i< t:
        pwma.ChangeDutyCycle(L_control.update(lspeed))
        pwmb.ChangeDutyCycle(R_control.update(rspeed))
        time.sleep(0.1)
        i+=  0.1


except KeyboardInterrupt:
    pass
pwma.stop()
pwmb.stop()
GPIO.cleanup()
```