智能车实验报告

自动化2104班 马茂原 学号：2216113438

1. 硬件和软件环境配置
2. 硬件配置

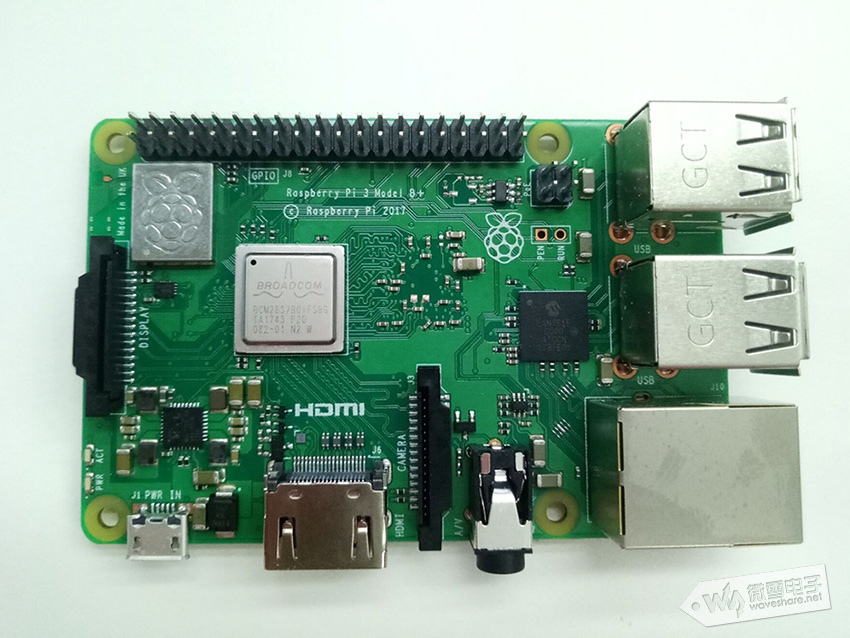


图1

树莓派3B

一个摄像头

两个PWM电机

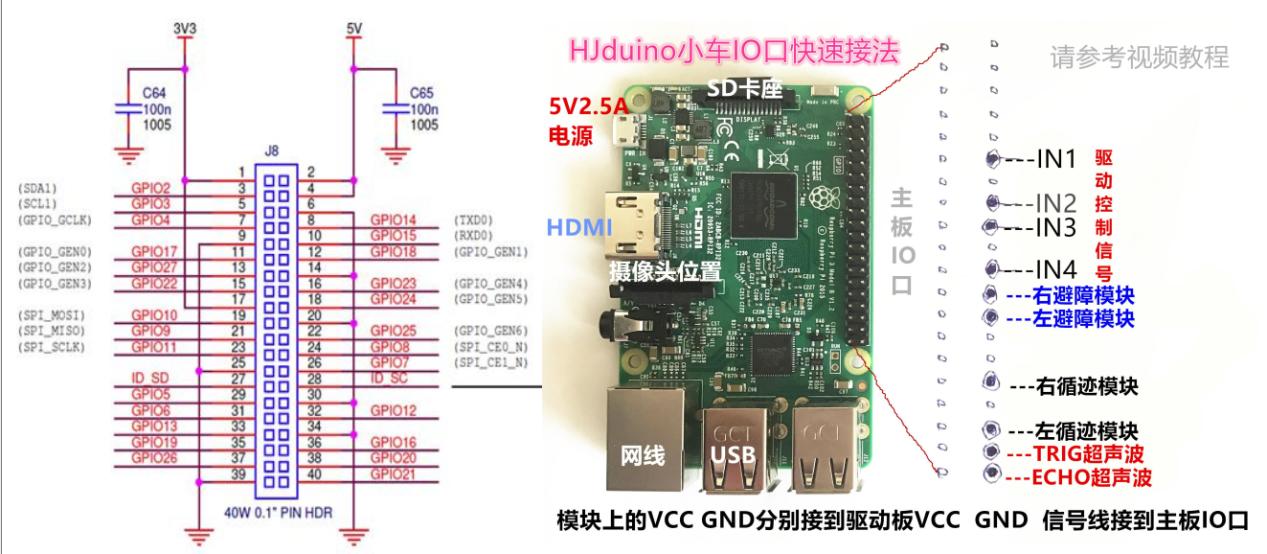


图2

引脚分配见后续代码。

1. 软件配置

使用树莓派镜像烧录器



图3

下载镜像，并选择自定义镜像。



图4

并设置手机热点作为智能车的网络。

1. 下载opencv

执行命令： sudo nano /etc/apt/sources.list

进入文件后注释掉所有行，在开头添加：

deb http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/bullseye main non-free contrib rpi deb-src http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/bullseye main non-free contrib rpi ctrl+o ctrl +x 保存退出

执行命令： sudo nano /etc/apt/sources.list.d/raspi.list

同样注释后添加：

deb http://mirrors.tuna.tsinghua.edu.cn/raspberrypi/bullseye/main

更新软件源列表，同时也可以检查我们的换源编辑是否正确，在终端运行如下命令：

sudo apt-get update

运行命令安装opencv:

Pip install python3-opencv

1. 配置智能车摄像头

sudo raspi-config

找到Interface Options --> Camera -->打开摄像头选择YES，可以在终端中查看device0是否存在

1. 下载VNC Viewer

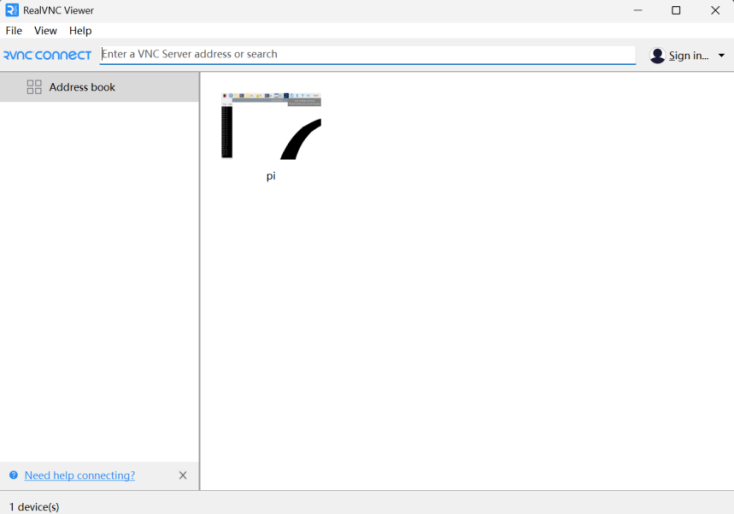


图5

并添加智能车的网络，VNC View用于查看智能车摄像头的实时画面[1]。

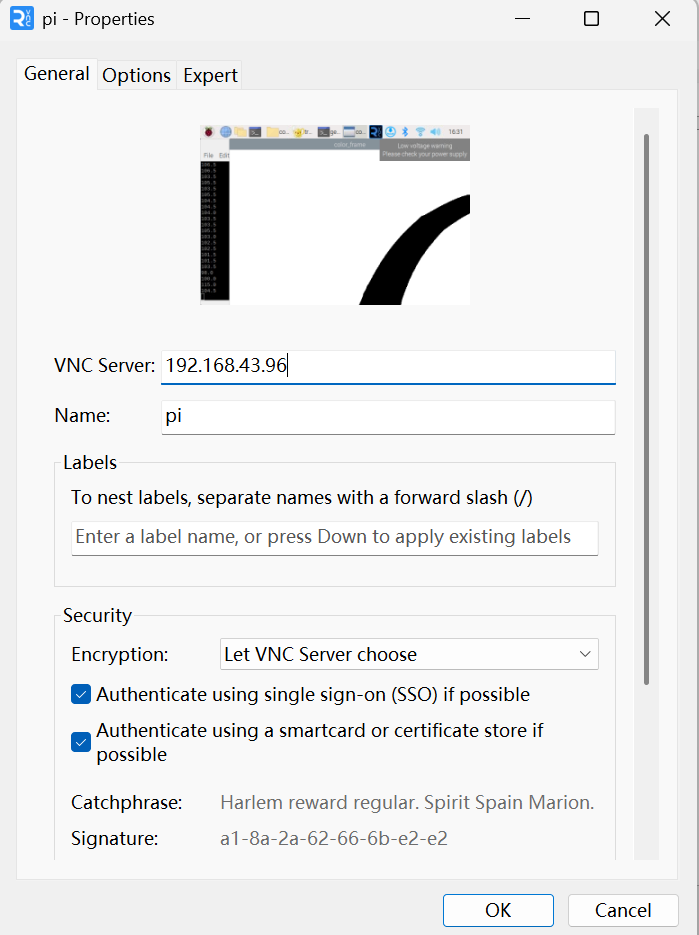


图6

1. 八字循迹

要求：在八字线上随机放一个黑色十字架、一个红十字架，遇到黑十字架车停一秒，遇到红十字架车停三秒，三分钟之内完成八字循迹。

本代码是一个基于树莓派的智能车自动循迹控制程序，使用摄像头模块来识别路径并根据检测结果控制小车前进、后退、左转和右转。

1. 导入库和设置常量

import RPi.GPIO as GPIO

import time

import cv2

import numpy as np

LINE0 = 150

LINE1 = 350

LINE2 = 380

BLACK\_HIGH = 25

RED\_LOW = 40

RED\_HIGH = 70

LEFT\_COL\_PX = 50

RIGHT\_COL\_PX = 590

RED\_FLAG = True

BLACK\_FLAG = True

常量定义了图像中用于检测颜色的行位置和阈值。

2. 设置GPIO引脚

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD) # 物理引脚BOARD编码

IN\_List = [12, 16, 18, 22]

ENA = 38

ENB = 40

OUTPUT\_GPIO = [12, 16 ,18, 22, 38, 40]

for item in OUTPUT\_GPIO:

GPIO.setup(item, GPIO.OUT)

PWM1 = GPIO.PWM(ENA, 50) # right

PWM2 = GPIO.PWM(ENB, 50) # left

PWM1.start(0)

PWM2.start(0)

设置GPIO模式为物理引脚编号。

初始化输出引脚列表，并将它们设为输出模式。

创建两个PWM对象分别控制左右电机的速度。

3. 定义小车运动函数

def move\_forward():

pwm\_value\_right = 30.0

pwm\_value\_left = 30.0

GPIO.output(IN\_List[0], GPIO.HIGH)

GPIO.output(IN\_List[1], GPIO.LOW)

GPIO.output(IN\_List[2], GPIO.LOW)

GPIO.output(IN\_List[3], GPIO.HIGH)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def move\_backward():

pwm\_value\_right = 30.0

pwm\_value\_left = 30.0

GPIO.output(IN\_List[0], GPIO.LOW)

GPIO.output(IN\_List[1], GPIO.HIGH)

GPIO.output(IN\_List[2], GPIO.HIGH)

GPIO.output(IN\_List[3], GPIO.LOW)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def turn\_right():

pwm\_value\_left = 60.0

pwm\_value\_right = 70.0

GPIO.output(IN\_List[0], GPIO.HIGH)

GPIO.output(IN\_List[1], GPIO.LOW)

GPIO.output(IN\_List[2], GPIO.HIGH)

GPIO.output(IN\_List[3], GPIO.LOW)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def turn\_left():

pwm\_value\_left = 70.0

pwm\_value\_right = 60.0

GPIO.output(IN\_List[0], GPIO.LOW)

GPIO.output(IN\_List[1], GPIO.HIGH)

GPIO.output(IN\_List[2], GPIO.LOW)

GPIO.output(IN\_List[3], GPIO.HIGH)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def stop():

for i in range(len(IN\_List)):

GPIO.output(IN\_List[i], GPIO.LOW)

4. 摄像头初始化和图像处理

cap = cv2.VideoCapture(0)

cap.set(3, 640)

cap.set(4, 480)

while (1):

ret, frame = cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

dst = gray

detect = gray[LINE0:LINE1]

left\_col = gray[:, LEFT\_COL\_PX]

right\_col = gray[:, RIGHT\_COL\_PX]

black\_num = len(np.where(detect <= BLACK\_HIGH)[0])

red\_num = len(np.where((detect >= RED\_LOW) & (detect <= RED\_HIGH))[0])

if red\_num >= 5000 and RED\_FLAG:

stop()

print("===== RED ===== STOP =====")

RED\_FLAG = False

BLACK\_FLAG = True

time.sleep(3.0)

if black\_num >= 20000 and BLACK\_FLAG:

left\_num = len(np.where(left\_col <= BLACK\_HIGH + 50)[0])

right\_num = len(np.where(right\_col <= BLACK\_HIGH + 50)[0])

if not (left\_num > 20 and right\_num > 20):

stop()

print("===== BLACK ===== STOP =====")

RED\_FLAG = True

BLACK\_FLAG = False

time.sleep(1.0)

orientation = dst[LINE2]

blacks = np.where(orientation <= 20)

if blacks is not None:

center = np.mean(blacks)

else:

center = 320

direction = center - 320

cv2.imshow('color\_frame', dst)

threshold = 100

if direction <= -threshold:

turn\_left()

elif direction >= threshold:

turn\_right()

else:

move\_forward()

if cv2.waitKey(1) & 0xFF == ord('q'):

break

打开摄像头并设置分辨率为640x480。

在循环中不断读取摄像头帧并转换为灰度图像。

提取特定行的颜色信息用于检测红黑线。

根据检测到的黑色像素数量判断是否停止或继续行驶。

计算目标中点与标准中点（320）的偏移量，根据偏移量调整小车的方向。

5. 清理工作

cap.release()

cv2.destroyAllWindows()

PWM1.stop()

PWM2.stop()

GPIO.cleanup()

释放摄像头资源，关闭OpenCV窗口，停止PWM信号，清理GPIO设置。

整个程序的核心思想是利用摄像头捕捉图像并通过图像处理算法确定路径的位置，然后根据路径的位置调整小车的行驶方向，从而实现自动循迹。

其核心代码需要区分以下四种情况：

情况一：智能车在普通的黑色道路上运行，智能车需要保持运动状态。

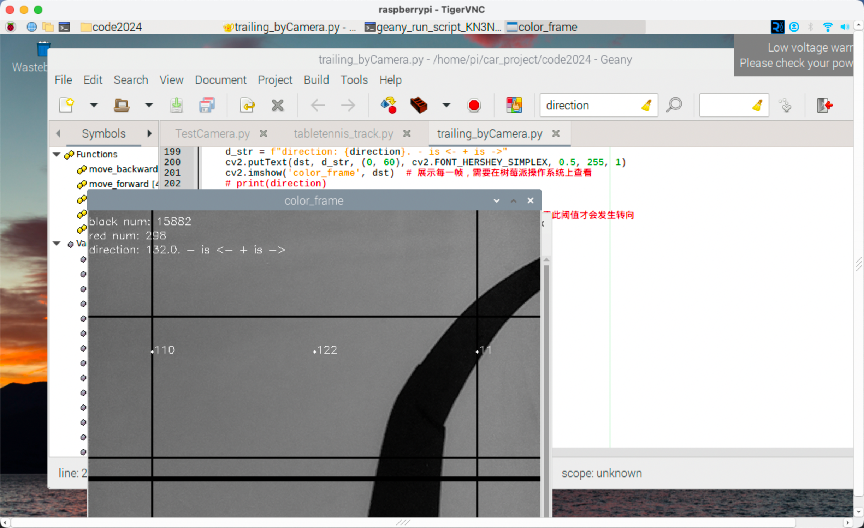


图7

情况二：智能车遇到黑色十字架，智能车需要停止运行1秒。

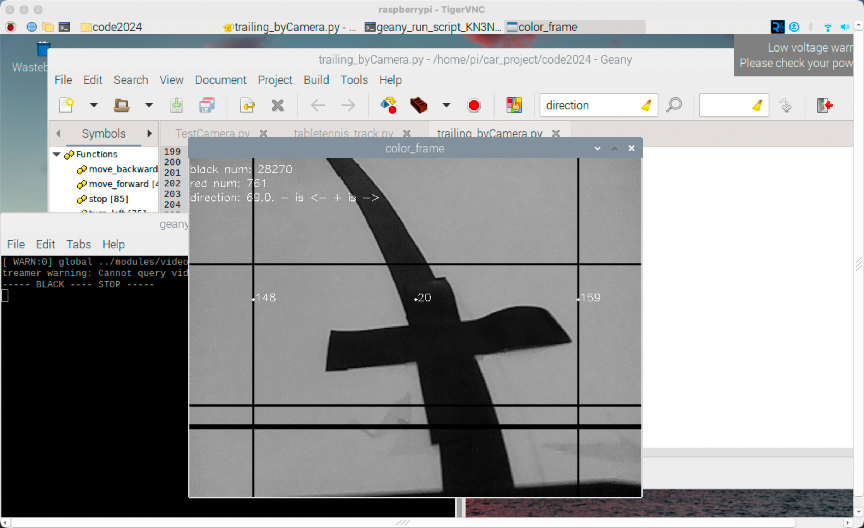


图8

情况三：智能车遇到红色十字架，智能车需要停止运行3秒。

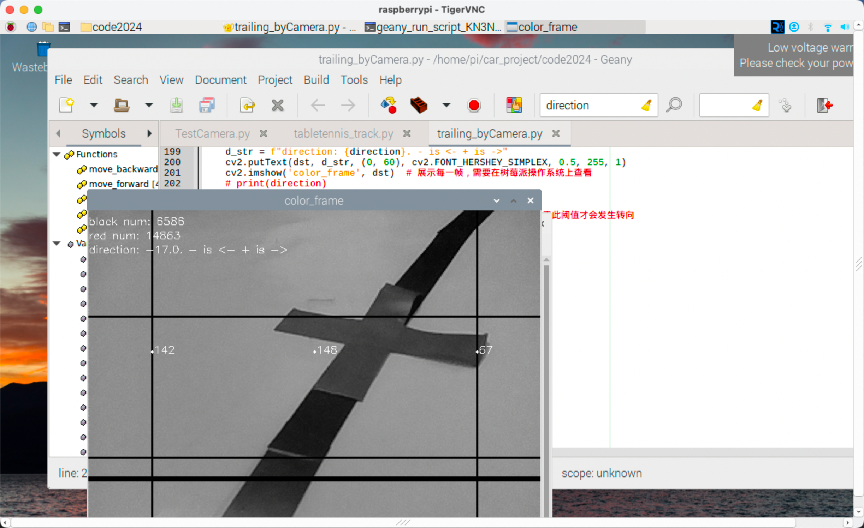


图9

情况四：智能车遇到“8”字黑色道路的交叉处，智能车需要保持运动状态。

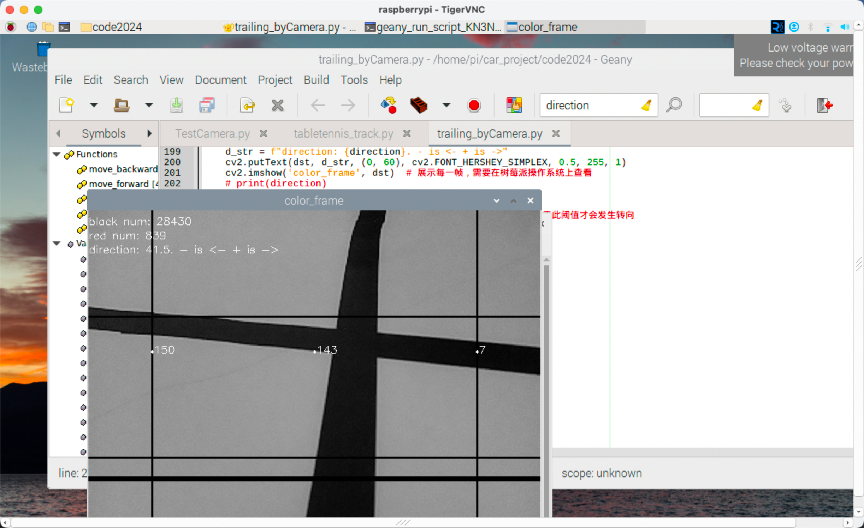


图10

a）首先，区分情况三与其他情况。

获取灰度图像的一部分连续横行作为特征提取的范围（经过测试为150-350），求出其灰度

值位于红色灰度值范围（经过测试为40-70）的数量，当其超过某一阈值时，智能车停止运

行3秒。又通过设置红色十字暂停标志RED\_FLAG，只有当RED\_FLAG为True时，智能车才

有可能因为遇到红色十字而停止，从而避免了多次停止运行的情况。

b）其次，区分情况一与情况二、情况四

由图7可知，和情况二、情况三相比，特征提取的范围内（经过测试为150-350）的黑色像

素（经过测试为0-20）数量较少，因此，当区域内的黑色像素数量小于某一阈值时（经过

测试为2000），认为情况一发生，智能车继续保持运动状态。只有当黑色像素数量大于该

阈值时，认为情况二或情况三发生。

c）最后，区分情况二和情况四

由图8和10所示，情况二的黑色横线较短，在两边区域存在很多白色像素；情况四的黑色

横线较长，在两边区域的白色像素相比与情况二较少。因此，在b）情况的基础上（黑色像

素数量大于该阈值），提取图像两侧的特定列像素（经过测试为50和590）。如果这两列

像素的黑色像素值均大于某个值，则认为是情况四，智能车保持运动状态；如果这两列像素

的黑色像素值没有均大于某个值，则认为是情况二，智能车暂停1秒。又通过设置黑色十字

暂停标志Black\_FLAG，只有当Black\_FLAG为True时，智能车才有可能因为遇到黑色十字而

停止，从而避免了多次停止运行的情况。



图11

1. 乒乓球跟踪
2. 导入库：

import cv2

import numpy as np

import RPi.GPIO as GPIO

设置全局变量和常量：

imshow = True

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD) # 物理引脚BOARD编码

# 引脚12、16、18、22【IN1 - IN4】

IN\_List = [12, 16, 18, 22]

ENA = 38

ENB = 40

# 输出引脚

OUTPUT\_GPIO = [12, 16, 18, 22, 38, 40]

for item in OUTPUT\_GPIO:

GPIO.setup(item, GPIO.OUT)

# 设置PWM波,频率为50Hz

PWM1 = GPIO.PWM(ENA, 50) # right

PWM2 = GPIO.PWM(ENB, 50) # left

PWM1.start(0)

PWM2.start(0)

1. 定义小车运动函数

def move\_forward():

pwm\_value\_right = 35.0 # 值越小，速度越小

pwm\_value\_left = 35.0

GPIO.output(IN\_List[0], GPIO.HIGH)

GPIO.output(IN\_List[1], GPIO.LOW)

GPIO.output(IN\_List[2], GPIO.LOW)

GPIO.output(IN\_List[3], GPIO.HIGH)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def move\_backward():

pwm\_value\_right = 35.0 # 值越小，速度越小

pwm\_value\_left = 35.0

GPIO.output(IN\_List[0], GPIO.LOW)

GPIO.output(IN\_List[1], GPIO.HIGH)

GPIO.output(IN\_List[2], GPIO.HIGH)

GPIO.output(IN\_List[3], GPIO.LOW)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def turn\_right():

pwm\_value\_left = 37.0

pwm\_value\_right = 37.0

GPIO.output(IN\_List[0], GPIO.HIGH)

GPIO.output(IN\_List[1], GPIO.LOW)

GPIO.output(IN\_List[2], GPIO.HIGH)

GPIO.output(IN\_List[3], GPIO.LOW)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def turn\_left():

pwm\_value\_left = 37.0

pwm\_value\_right = 37.0

GPIO.output(IN\_List[0], GPIO.LOW)

GPIO.output(IN\_List[1], GPIO.HIGH)

GPIO.output(IN\_List[2], GPIO.LOW)

GPIO.output(IN\_List[3], GPIO.HIGH)

PWM1.ChangeDutyCycle(pwm\_value\_right)

PWM2.ChangeDutyCycle(pwm\_value\_left)

def stop():

for i in range(len(IN\_List)):

GPIO.output(IN\_List[i], GPIO.LOW)

1. 绘制方向箭头

def draw\_direction(img, lx, ly, nx, ny):

dx = nx - lx

dy = ny - ly

if abs(dx) < 4 and abs(dy) < 4:

dx = 0

dy = 0

else:

r = (dx \*\* 2 + dy \*\* 2) \*\* 0.5

dx = int(dx / r \* 40)

dy = int(dy / r \* 40)

cv2.arrowedLine(img, (60, 100), (60 + dx, 100 + dy), (0, 255, 0), 2)

1. 霍夫圆检测

使用霍夫变换在灰度图像中检测圆形，并在画布上绘制检测到的圆[2]：

def Hough\_circle(imgGray, canvas):

global Hough\_x, Hough\_y # 定义全局变量

img = cv2.medianBlur(imgGray, 3) # 中值滤波

img = cv2.GaussianBlur(img, (17, 19), 0) #高斯滤波

circles = cv2.HoughCircles(img, cv2.HOUGH\_GRADIENT, 1, 200, param1=20, param2=50, minRadius=30, maxRadius=70) # 霍夫曼圆检测

try:

circles = np.uint16(np.around(circles))

except:

pass #如果没有检测到圆，跳过

else:

for i in circles[0, :]:

cv2.circle(canvas, (i[0], i[1]), i[2], (255, 100, 0), 2)

cv2.circle(canvas, (i[0], i[1]), 2, (0, 0, 255), 3)

Hough\_x = i[0]

Hough\_y = i[1]

如果检测到圆，则绘制圆的外边缘和中心点。

1. 主循环参数设置

frameWidth = 640

frameHeight = 480

cap = cv2.VideoCapture(0)

cap.set(3, frameWidth)

cap.set(4, frameHeight)

cap.set(10, 80) # 设置亮度

pulse\_ms = 30

lower = np.array([4, 180, 156]) # 适用于橙色乒乓球4<=h<=32

upper = np.array([32, 255, 255])

targetPos\_x = 0 # 颜色检测得到的x坐标

targetPos\_y = 0 # 颜色检测得到的y坐标

lastPos\_x = 0 # 上一帧图像颜色检测得到的x坐标

lastPos\_y = 0 # 上一帧图像颜色检测得到的y坐标

lastarea = 0

Hough\_x = 0 # 霍夫圆检测得到的x坐标

Hough\_y = 0 # 霍夫圆检测得到的y坐标

1. 主循环：

6.1图像预处理：分离图像的BGR通道，计算红色通道减去蓝色通道的结果:

\_, img = cap.read()

b, g, r = cv2.split(img)

r = np.int16(r)

b = np.int16(b)

r\_minus\_b = r - b

r\_minus\_b = (r\_minus\_b + abs(r\_minus\_b)) / 2

r\_minus\_b = np.uint8(r\_minus\_b)

6.2复制图像并将原始图像转换为HSV颜色空间，以便更容易地进行颜色范围的筛选:

imgHough = img.copy()

imgHsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV) # 将BGR格式的图像转换为HSV格式。

颜色掩膜和轮廓检测，创建颜色掩膜，只保留符合条件的颜色区域:

imgMask = cv2.inRange(imgHsv, lower, upper)

返回一个二值图像 imgMask，其中像素值在指定范围内的位置为白色（255），其他位置为黑色（0）

contours,hierarchy=cv2.findContours(imgMask,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_NONE)

6.3绘制边界框和目标位置，遍历所有轮廓，计算每个轮廓的面积。如果面积大于300，则绘制矩形框和中心点，并更新最大面积及其对应的坐标：

x, y, w, h = 0, 0, 0, 0

max\_area = 0

targetPos\_x\_max = 0

targetPos\_y\_max = 0

for cnt in contours:

area = cv2.contourArea(cnt)

if area > 300:

x, y, w, h = cv2.boundingRect(cnt) # 获取当前轮廓的矩形边界

lastarea = area

lastPos\_x = targetPos\_x

lastPos\_y = targetPos\_y

targetPos\_x = int(x + w / 2)

targetPos\_y = int(y + h / 2)

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2) # 绘制矩形边界

cv2.circle(img, (targetPos\_x, targetPos\_y), 2, (0, 255, 0), 4) # 绘制圆心

if max\_area < area:

max\_area = area

targetPos\_x\_max = targetPos\_x

targetPos\_y\_max = targetPos\_y

cv2.putText(img, "({:0<2d}, {:0<2d}, {:0<2f})".format(targetPos\_x\_max, targetPos\_y\_max, max\_area), (20, 30), cv2.FONT\_HERSHEY\_PLAIN, 1, (0, 255, 0), 2)

draw\_direction(img, lastPos\_x, lastPos\_y, targetPos\_x, targetPos\_y)

6.4霍夫圆检测:

Hough\_circle(r\_minus\_b, imgHough)

if imshow:

cv2.imshow("R\_Minus\_B", r\_minus\_b)

cv2.putText(imgHough, "({:0<2d}, {:0<2d})".format(Hough\_x, Hough\_y), (20, 30),

cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 100, 0), 2)

进行霍夫圆检测并绘制霍夫圆检测得到的目标位置。

水平拼接原始图像和霍夫圆检测结果图像显示:

imgStack = np.hstack([img, imgHough])

if imshow:

cv2.imshow('Horizontal Stacking', imgStack)

6.5控制小车运动:

如果检测到的目标面积小于100，则停止小车。否则，根据目标位置相对于图像中心的位置调整小车的方向，如果目标位置偏左超过阈值，则左转。如果目标位置偏右超过阈值，则右转。否则，根据目标面积调整小车的移动：如果面积小于5000，则前进。如果面积介于5000和8000之间，则停止。如果面积大于8000，则后退。

if max\_area < 100:

stop()

else:

x\_delta = targetPos\_x - 320

threshold = 150

if x\_delta <= -threshold:

turn\_left()

elif x\_delta >= threshold:

turn\_right()

else:

if max\_area < 5000:

move\_forward()

elif max\_area < 8000:

stop()

else:

move\_backward()

6.6退出循环：

if cv2.waitKey(pulse\_ms) & 0xFF == ord('q'):

print("Quit\n")

break

1. 清理工作：

cap.release()

cv2.destroyAllWindows()

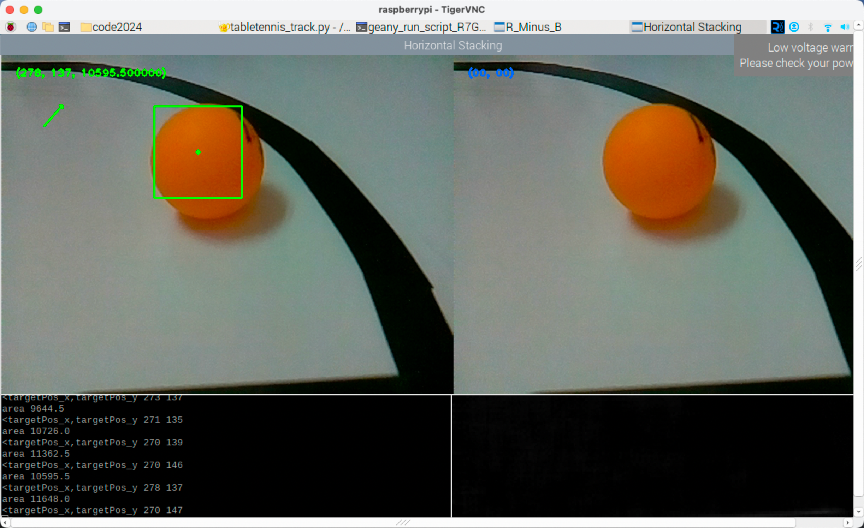


图12

代码的核心功能是通过摄像头捕捉图像，检测图像中的橙色乒乓球，并根据检测结果控制小车的运动，使其能够跟随乒乓球移动。

1. 实验总结

此次实验利用嵌入式平台结合计算机视觉技术实现智能控制。通过对硬件的选择与组装、软件环境的设置、图像处理算法的设计等一系列步骤，我们构建了一个具备自主导航能力的小型智能车系统。这不仅加深了对相关知识的理解，也为今后进一步探索更复杂的智能系统奠定了坚实的基础。此外，实验过程中遇到的问题及解决方案也为后续研究提供了宝贵的经验参考。

1. 参考文献

[1] “基于树莓派4B的OpenCV安装与简单应用（真速通版）\_树莓派opencv-CSDN博客.” Accessed: Jan. 04, 2025. [Online]. Available: https://blog.csdn.net/black\_sneak/article/details/131343797

[2] “opencv 十一 霍夫圆检测原理及高级使用案例（含优化步骤）-CSDN博客.” Accessed: Jan. 04, 2025. [Online]. Available: https://blog.csdn.net/m0\_74259636/article/details/132655935