# 第五章 系统软件设计和仿真分析

前面的章节已经完成了捡球小车的控制方案设计、硬件选型和硬件电路的设计，本章节将在前面工作已经完成的基础上，开始设计捡球小车的运行程序。本章节将围绕以下？个主要功能进行程序的设计和讲解。代码篇幅适中的会直接在论文里展示，而代码篇幅较长的在论文中只会展示核心代码，全部代码会以附录的形式附在论文最后。

## 5.1 捡球小车与客户端通信设计

捡球小车的这个功能是通过套接字（socket）实现的，能够让用户使用手机APP控制捡球小车的运动并且能够通过手机屏幕实时监控捡球小车前方车况。socket是计算机之间一种通信的方式，通过socket能够实现不同计算机之间的信息发送与接收。在Python中，socket已经被集成为了库函数，方便开发者直接调用，大大节约程序的开发时间。要想实现两台设备之间的通信就需要设计两个py文件，一个作为客户端，一个作为服务端。这里选择手机作为客户端，向捡球小车发送指令；捡球小车作为服务端。

首先设计作为服务端的捡球小车。

**import** threading

**import** socket

socket\_port = 1314

socket\_get\_ok = 0

socket\_receive\_buf = ""

引入了threading和socket库。将1314端口设置为socket通信端口，并且将接收到的信息存放在socket\_receive\_buf变量里，变量socket\_get\_ok则是用于判断是否有数据接收。一般情况下，一个进程最少拥有一个线程，当存在多个线程的情况下，就需要设计一个主线程。同一个进程的所有线程共用系统资源，但是每个线程都拥有属于自己的栈和寄存器。为了完成捡球小车的多线程设计，所以引入了Python的多线程模块——threading。捡球小车与手机APP的通信引入多线程可以大大提高系统的处理效率，减少服务端和客户端数据收发的时延。

**def** SocketEvent():

**global** socket\_receive\_buf, socket\_get\_ok, socket\_port

    serverSocket = socket.socket()

    hostIP = "192.168.12.1"

**print**('hostIP:', hostIP)

    # data port

    dataPort = socket\_port

    address = (hostIP, dataPort)

    serverSocket.bind(address)

serverSocket.listen(5)

**print**('socket waitting......')

    serverConnect, addr = serverSocket.accept()

**print**('socket connected!')

 mode = 0

这里定义了一个名为SocketEvent的函数。函数开头先创建了一个套接字，将网络地址设置为192.168.121。serverSocket.listen(5)将超过限制数量的连接之前，捡球小车可以挂起的最大连接数量设置为5，可以简单的理解为捡球小车的最多只可以连接5个手机APP。mode0是捡球小车的一种工作模式。

**while** True:

    data = str(serverConnect.recv(1024))

    data = data.split('\'')[1]

**if** len(data):

**print**(data)

socket\_receive\_buf = data

**if** socket\_get\_ok == 0:

            uart\_receive\_buf = socket\_receive\_buf

            uart\_get\_ok = 0

**if** mode == 0:

**if** uart\_receive\_buf.find('{') >= 0:

                    mode = 1

**elif** uart\_receive\_buf.find('$') >= 0:

                    mode = 2

**elif** uart\_receive\_buf.find('#') >= 0:

                    mode = 3

**if** mode == 1:

**if** uart\_receive\_buf.find('}') >= 0:

                    uart\_get\_ok = 1

                    mode = 0

**elif** mode == 2:

**if** uart\_receive\_buf.find('!') >= 0:

                    uart\_get\_ok = 2

                    mode = 0

**elif** mode == 3:

**if** uart\_receive\_buf.find('!') >= 0:

                    uart\_get\_ok = 3

                    mode = 0

            socket\_get\_ok = uart\_get\_ok

**else**:

**print**('socket disconnected.')

        serverSocket.listen(5)

**print**('socket waitting......')

        serverConnect, addr = serverSocket.accept()

**print**('socket connected!')

et\_thread = threading.Thread(target=socketEvent)

这段程序是作用是将捡球小车收到的信息进行处理，用到的方法有str、split和find。首先将捡球小车收到的信息使用str方法将其转换为字符串类型，在按照“\’”将其分割成多个字符串，这些字符串会保存在一个列表中，在传递给socket\_receive\_buf存储。之后会进行一系列的判断过程，通过find查找到的关键字来选择捡球小车的工作模式。

**def** setup\_socket(port):

**global** socket\_thread, socket\_port

    socket\_port = port

    # socket接收线程

    socket\_thread.start()

**def** loop\_socket():

**global** socket\_get\_ok, socket\_receive\_buf

**if** socket\_get\_ok:

**print**(socket\_receive\_buf)

        socket\_get\_ok = 0

**if** \_\_name\_\_ == '\_\_main\_\_':

    setup\_socket(2020)

**try**:

**while** True:

            loop\_socket()

**except** KeyboardInterrupt:

**if** serverSocket != None:

  serverSocket.close()

这段程序则是开始接收线程，循环接收线程。

手机端APP设计则是使用了Android进行开发，通过socket与捡球小车通信,surfaceview读取视频流捡球小车摄像头的视频流。由于工程文件太多，这里只展示手机端APP的最后实现效果。图5-1展示了手机APP的连接配置页面。



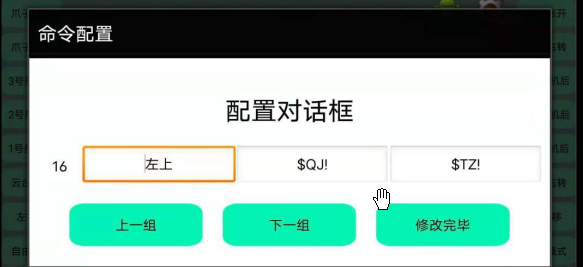
**图5-1手机端APP连接配置界面**

首先将手机连接到捡球小车发出的热点，APP默认配置好了能与捡球小车通信的正确网络地址、视频端口和数据端口。



**图5-2 运行界面展示**

用户还可以通过程序右上角的安卓图标更改按键配置，方便用户设置为自己的习惯按键位置。



**图5-3按键位置更改**

图5-4用程序流程图演示手机APP如何通过socket与捡球小车进行通信，进行实时图传与控制。



**图5-4 socket通信流程图**

## 5.2 乒乓球识别程序

乒乓球识别使用到了NumPy和OpenCV库。Numpy为Python科学计算中的基本工具，可用于矩阵的存储与处理，而在此程序中用于颜色阈值的设定;OpenCV为计算机视觉中常用的库，并提供Python接口，可使开发者用Python语言来调用C/C++,既能保证运行效率又能提高程序的阅读性。

Open CV为Hough Circles()提供了一种检测图像圆形的方法，也称为霍夫圆环变换（Holf Transfer Transform）。 霍夫圆环变换在检测过程中会有两次检测，第一次检测发现可能为圆形的圆心坐标,第二次检测计算这些圆心坐标的半径长度,最后将两者储存于圆心坐标与半径长度所组成的浮点型数组中。 霍夫圆形变换为开发者提供了大量的参数设置，尽管开发者使用起来会很方便，但随之而来的坏处就是局限性大，无法为捡球小车提供适合不同环境下场地的快速调参。 而且很难迅速、准确地获得这些数据。 如果要让捡球能够准确地识别到乒乓球并定位它就必须解决这个问题。

为此，本论文提出了一种新方法，不仅可以做到识别率高，而且能够让无任何编程基础的用户快速进行参数，达到最优的识别效果。

这个新方法的核心是通过先对图像预处理降低输入数据量， 然后利用HSV颜色模型及边缘检测技术对特征进行匹配。

图5-5展示了乒乓球识别程序的流程图。在流程图中可以看到乒乓球识别程序有两大分支，一个是对摄像头捕捉到的画面进行预处理，进行调参；另一个是利用前一个程序得到的参数，在进行空间色彩变换、边缘检测等操作识别到乒乓球。



图5- 5 乒乓球识别程序流程图

首先设计一个名为hsv.py程序，此程序的功能是对图像进行预处理，调试各种参数，达到最优的识别效果。此程序的优点在于用户只用拖动滑条即可完成调参。

**import** cv2

**import** numpy as np

frameWidth = 960

frameHeight = 540

cap = cv2.VideoCapture(0)

cap.set(3, frameWidth)

cap.set(4, frameHeight)

cap.set(10, 50)

pulse\_ms = 30

lower = np.array([13, 96, 119])

upper = np.array([23, 255, 255])

首先导入需要用到的opencv和numpy库。frameWidth和frameHeight代表着显示窗口的行列像素，通俗说就是窗口的大小。使用VideoCapture方法初始化摄像头，0代表摄像头的索引。set中，这里的3，下面的4和10是类似于功能号的东西，数字的值没有实际意义。cap.set(10,50)则是设置了亮度。lower和upper则是参照了HSV色彩空间内黄色的颜色范围，初始化了一个预设值。

def empty(a):

pass

cv2.namedWindow("HSV")

cv2.resizeWindow("HSV", 640, 300)

cv2.createTrackbar("HUE Min", "HSV", 13, 179, empty)

cv2.createTrackbar("SAT Min", "HSV", 96, 255, empty)

cv2.createTrackbar("VALUE Min", "HSV", 119, 255, empty)

cv2.createTrackbar("HUE Max", "HSV", 23, 179, empty)

cv2.createTrackbar("SAT Max", "HSV", 255, 255, empty)

cv2.createTrackbar("VALUE Max", "HSV", 255, 255, empty)

这段代码的作用是产生控制滑条。使用namedWindows创建了一个叫HSV的窗口并将窗口尺寸设为640\*300。使用createTrackbar创建6个滑条分别代表着HUE Min、SAT Min、VALUE Min、HUE Max、SAT Max和VALUE Max。HUE代表着色调、SAT代表着饱和度、VALUE代表着明亮度。这里使用HSV而不是RGB的原因是，HSV是面向用户的，RGB是面向硬件的。createTrackbar的第五个参数是执行回调函数，回调函数始终具有默认参数，即轨迹栏位置。但是在捡球小车中，回调函数什么都不做，所以定义一个empty函数使用pass简单地通过即可。

h\_min = cv2.getTrackbarPos("HUE Min", "HSV")

h\_max = cv2.getTrackbarPos("HUE Max", "HSV")

s\_min = cv2.getTrackbarPos("SAT Min", "HSV")

s\_max = cv2.getTrackbarPos("SAT Max", "HSV")

v\_min = cv2.getTrackbarPos("VALUE Min", "HSV")

v\_max = cv2.getTrackbarPos("VALUE Max", "HSV")

lower = np.array([h\_min, s\_min, v\_min])

upper = np.array([h\_max, s\_max, v\_max])

这段程序的功能是获取滑条的位置，并且将这些值分别赋值给h\_min、h\_max、s\_min、s\_max、v\_min、v\_max，再将这六个值传递给lower和upper。这样就可以实现用过通过控制滑条来控制画面只显示乒乓球。

\_, img = cap.read()

imgHsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

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

imgOutput = cv2.bitwise\_and(img, img, mask=imgMask)

imgMask = cv2.cvtColor(imgMask, cv2.COLOR\_GRAY2BGR)

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

cv2.namedWindow("imgMask", 0)

cv2.resizeWindow("imgMask", 800, 400)

cv2.imshow('imgMask', imgStack)

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

**print**("Quit\n")

**break**

这段程序则是显示摄像头得到的画面进行处理和显示。cv2.inRange方法可以提取用户想要的的颜色，这里的效果是只显示用户调试好的最佳乒乓球识别颜色，其余的位置显示为纯黑色。为了用户在调试完毕后可以直接退出，在程序的最后设置了按键检测，只要用户按下英文输入下的“q”键就可以退出程序。



**图5-6 程序运行效果**

图5-6展示了hsv.py程序运行的实时画面。从图中可以看出用户只需简单的拖动滑条，就可以让摄像头里只展示乒乓球。

cnts = cv2.findContours(mask.copy(),cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)[-2]

x\_bias = 0

y\_bias = 0

area = 0

**if** len(cnts) >0 :

    cnt = max(cnts,key=cv2.contourArea)

    rect = cv2.minAreaRect(cnt)

    box = cv2.boxPoints(rect)

    c\_x, c\_y = rect[0]

    c\_h, c\_w = rect[1]

    c\_angle = rect[2]

**if** c\_angle<-45:

        c\_angle = -(90+c\_angle)

**if** 1 < c\_h < 180 **and** 1 < c\_w < 180:

        cv2.drawContours(frame, [np.int0(box)], -1, (0, 255, 255), 2)

        x\_bias = c\_x - width/2

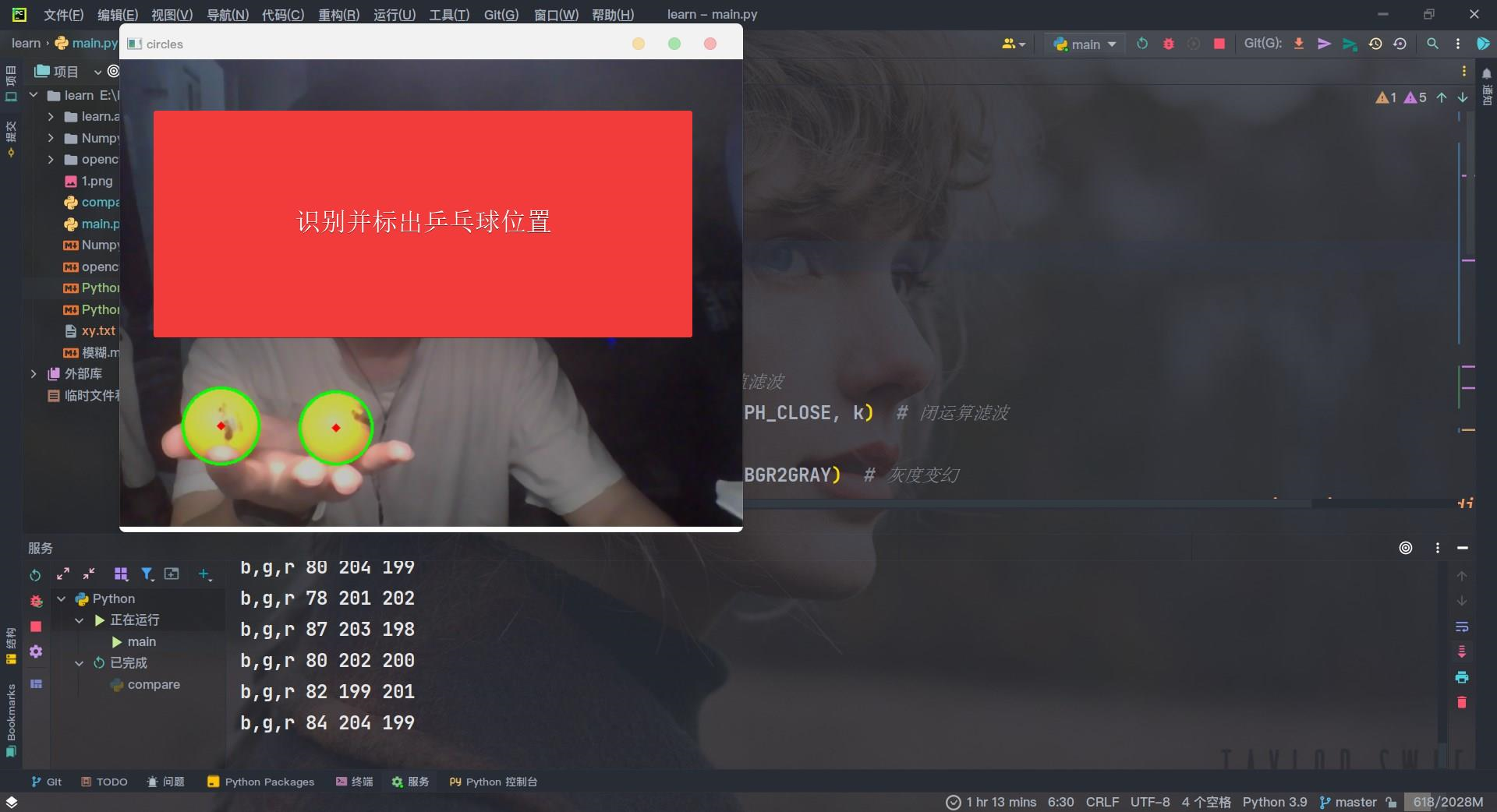
        y\_bias = c\_y - hight/2

        area = c\_h\*c\_w

cv2.imshow('frame',frame)

这段程序在前面调试好参数的情况下，通过cv2.findCountours()方法来计算图像梯度来判断出图像的边缘，再通过cv2.drawContours()方法来绘制出轮廓。在这段程序之前，会对图像进行预处理。通过cv2.erode()方法对图像进行腐蚀，强化图像的细节；cv2.GaussianBlur()方法进行高斯模糊，抑制图像中的噪声。

图5-7展示了乒乓球识别程序的最终识别效果。从图中可以明显的看到，乒乓球识别程序不仅准确识别到了图像中的乒乓球，而且画出了乒乓球的轮廓和圆心。



**图5-7乒乓球识别最终效果**

## 5.3 捡球小车跟踪程序

捡球小车跟踪程序需要前面乒乓球识别程序的配合，依靠准确识别出的乒乓球轮廓来指引捡球小车行驶。

**def** car\_run(speed\_l1,speed\_r1,speed\_l2,speed\_r2):

    textSrt = '#006P{:0>4d}T0000!#007P{:0>4d}T0000!#008P{:0>4d}T0000!#009P{:0>4d}T0000!'.format(speed\_l1,speed\_r1,speed\_l2,speed\_r2)

**print**(textSrt)

    myUart.uart\_send\_str(textSrt)

**def** car\_go\_back(speed):

    car\_run(1500+speed,1500-speed,1500+speed,1500-speed)

**def** car\_left\_turn(speed):

    speedl = 1500+speed\*2//3

    speedr = 0

    car\_run(speedl,speedr,speedl,speedr)

**def** car\_right\_turn(speed):

    speedl = 0

    speedr = 1500+speed\*2//3

    car\_run(speedl,speedr,speedl,speedr)

**def** car\_stop():

    myUart.uart\_send\_str('#255P1500T1000!')

这段程序定义了四个函数，分别是car\_run、car\_go\_back、car\_left\_turn、car\_rignt\_turn、car\_stop，可以实现捡球小车前进、后退、左转、右转和停止。变量textSrt代表的是向总线双路驱动模块发送的指令，接收到特定的指令，总线双路驱动模块会执行特定的动作。总线双路驱动模块的通讯协议在第四章做了详细阐述。通过调用不同的函数，捡球小车会执行特定的动作。

图5-8展示了捡球小车追踪程序流程图，并按照流程图所给思路设计捡球小车追踪代码。



图5- 8 捡球小车跟踪程序工作流程

**def** car\_follow():

**global** systick\_ms\_bak,next\_time,x\_bias,y\_bias,area

**while** True:

**if** int((time.time() \* 1000))- systick\_ms\_bak >= int(next\_time):

            systick\_ms\_bak = int((time.time() \* 1000))

**if** abs(x\_bias) < 10 **and** area > 400:

                car\_go\_back(400)

                next\_time = 0

**elif** int(x\_bias) > 10:

                next\_time = 0

                interval\_time = x\_bias/100

                car\_left\_turn()

**elif** int(x\_bias) < -10:

                next\_time = 0

                interval\_time = -x\_bias/100

                car\_right\_turn()

**elif** area == 0:

                car\_left\_turn()

                time.sleep(0.5)

                car\_stop()

                time.sleep(1)

                car\_right\_turn()

                time.sleep(0.5)

                car\_stop()

time.sleep(1)

这段程序是捡球小车的运行控制逻辑。程序会通过比较识别到的乒乓球的圆心和屏幕中点坐标的横坐标得到差值并且赋值给变量x\_bias。在满足x\_bias绝对值小于10和area大于400的条件下，捡球小车才会驶向识别到的乒乓球；如果乒乓球圆心横坐标和屏幕中点横坐标超过10，捡球小车分别会左转和右转；如果area的值为0则表示摄像头没有识别到乒乓球，那么捡球小车会左转再进行识别。其中的time.sleep(n)则是代表程序等待n秒的时间再接着往下执行，等待的时间留给乒乓球识别程序进行识别。

## 5.4 避障程序

捡球小车的避障程序分为两个部分，分别是超声波避障程序和碰撞传感器避障程序。二者相互搭配实现捡球小车避障功能。

图5-9展示了捡球小车的避障流程图。从图中可以看出避障程序是一个中断程序，在发生碰撞的时候触发中断。



图5- 9 避障程序流程图

**def** setup():

    GPIO.setmode(GPIO.BCM)

    GPIO.setwarnings(False)

    GPIO.setup(TRIG, GPIO.OUT)

    GPIO.setup(ECHO, GPIO.IN, pull\_up\_down = GPIO.PUD\_UP)

**def** distance():

    err = 0

    GPIO.output(TRIG, 0)

    time.sleep(0.000002)

    GPIO.output(TRIG, 1)

    time.sleep(0.00001)

    GPIO.output(TRIG, 0)

    time0 = time.time()

**while** GPIO.input(ECHO) == 0 **and** time.time()-time0<0.1:

        a = 0

**if** time.time()-time0 >= 0.1:

        err = 1

**return** 0

    time1 = time.time()

**while** GPIO.input(ECHO) == 1  **and** time.time()-time1<0.1:

        a = 1

**if** time.time()-time1 >= 0.1:

        err = 1

**return** 0

    time2 = time.time()

    during = time2 - time1

**return** during \* 340 / 2 \* 100

这段程序用来控制超声波传感器，获得小车与后方障碍物的距离。获得的距离用来辅助碰撞程序判断。

**def** collision\_detection\_left():

**if**(devValue\_left() == 0):

        time.sleep(0.02) #qu dou dong

**if**(devValue\_left() == 0):

**print**(">Dev get Low")

            beep\_on()

**while**(devValue\_left() == 0):

                car\_left\_turn()

                dis = distance()

**print** ('dis:%d cm'%dis)

                time.sleep(0.1)

            beep\_off()

**print**(">>>Dev get High")

**def** collision\_detection\_right():

**if**(devValue\_right() == 0):

        time.sleep(0.02) #qu dou dong

**if**(devValue\_right() == 0):

**print**(">Dev get Low")

            beep\_on()

**while**(devValue\_right() == 0):

                car\_right\_turn()

                dis = distance()

**print** ('dis:%d cm'%dis)

                time.sleep(0.1)

            beep\_off()

**print**(">>>Dev get High")

setup\_dev()

这里定义里两个函数分别叫做collision\_detection\_left和collision\_detection\_right，对用的是捡球小车左边和右边的碰撞传感器。在函数中time.sleep()的功能是用用于来去抖动，防止碰撞传感器机械设计以及器件老化所带来的问题。如果碰撞传感器被触发（以左碰撞传感器为例），集成在拓展板上的蜂鸣器会发出哔哔声，同时捡球小车会逆时针旋转，确保不会再次发生碰撞。此外，在程序终端还会打印出后方障碍物的距离。

## 5.5 换点搜索程序

换点搜索程序可以对捡球小车的结果进行全局解，能够保证场馆内所有区域都会被捡球小车摄像头扫描到，不会遗漏被大型障碍物遮挡的后方区域。

换点搜索程序的实现十分简单，如果捡球小车旋转一圈都没有找到乒乓球，那么捡球小车就会寻找摆在场馆内部固定位置的红色方块。

**while**(~red\_block):

    collision\_detection\_left()

只要没搜索到红色方块就持续调用collision\_detection\_left()，直到找到为止。红色方块的识别程序原理和乒乓球识别程序原理相同，这里不重复赘述。图5-10展示了捡球小车的换点搜索程序流程图。



图5- 10换点搜索程序流程图

## 5.6 串口程序

串口程序（z\_uart）用来将前面程序产生的串口信息发送到双路总线驱动模块，只需要在前面的程序导包是导入z\_uart即可。由于串口程序篇幅太长，这里只展示核心代码。

**def** loop\_uart():

**global** uart\_get\_ok, uart\_receive\_buf

**if**(uart\_get\_ok):

**print**(int(time.time()\*1000))

        uart\_send\_str(uart\_receive\_buf)

**if** uart\_receive\_buf == '$LEDON!':

            myLed.on(1)

**elif** uart\_receive\_buf == '$LEDOFF!':

            myLed.off(1)

**elif** uart\_receive\_buf == '$BEEPON!':

            myBeep.on()

**elif** uart\_receive\_buf == '$BEEPOFF!':

            myBeep.off()

        uart\_receive\_buf = ''

        uart\_get\_ok = 0

**if** \_\_name\_\_ == '\_\_main\_\_':

    setup\_uart(115200)

    myLed.setup\_led()

    myBeep.setup\_beep()

    myBeep.beep(0.1)

    myBeep.beep(0.1)

    myBeep.beep(0.1)

**try**:

**while** True:

            loop\_uart()

**except** KeyboardInterrupt:

**if** ser != None:

            ser.close()

        myLed.off(1)

        myLed.off(2)

myBeep.off()

因为捡球小车在运行时会持续产生串口信息，所以串口程序的核心要义是不断接收和发送串口信息，所以程序里有大循环确保串口信息可以传递给双路总线驱动模块。