智能小车实验报告

171250506
江辉

南京大学软件工程学院

目录

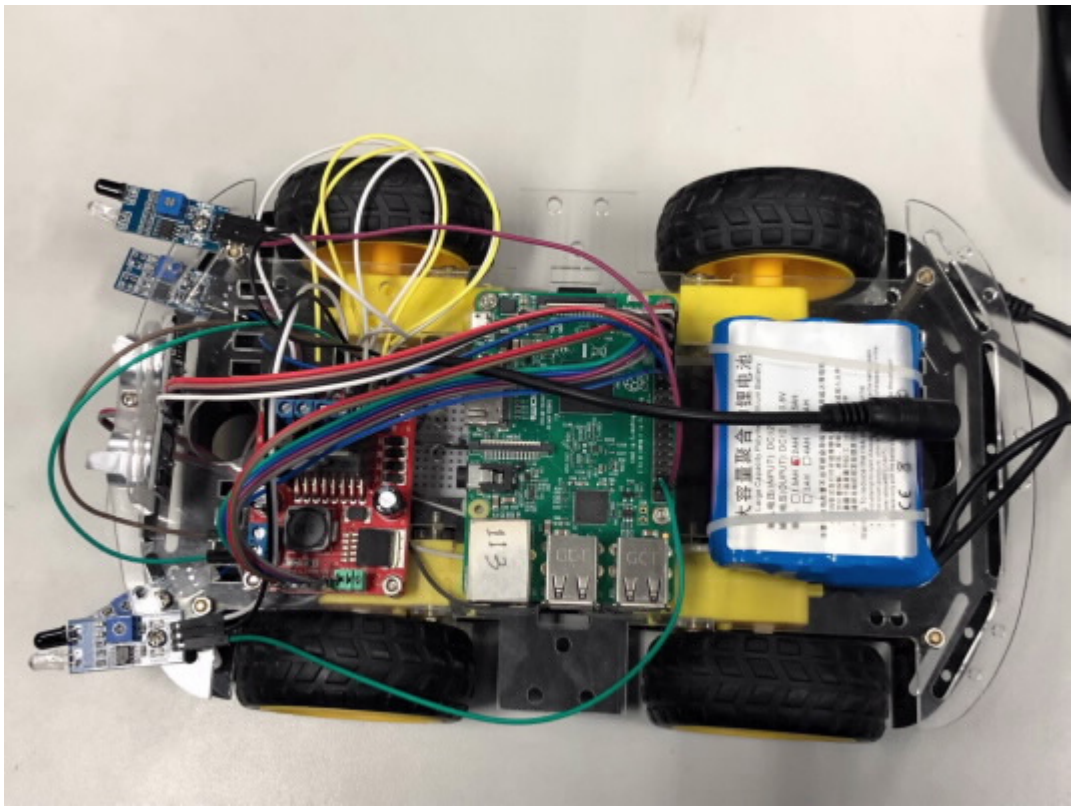
一、前言

此报告将详细描述我利用树莓派制作智能小车的相关事项，具体包括课程项目介绍、工具介绍、小车组装、功能介绍、具体代码、小组工作、实验总结等内容。在这个学期内学习制作这个智能小车，我学到了很多新的知识，比如如何通过编写程序来控制硬件和如何将硬件通过电路逻辑连接在一起协调工作，作为软件工程专业的学生，我在此之前都只是编写一些即写即跑的程序，从未接触过硬件与软件相结合的工作；同时，我也在小组工作中认识到了一些有趣的朋友。通过在网上查询资料和与同学讨论，解决了过程中遇到的很多问题，虽然最后做出来的小车功能很简单，只有手动操控、测距、自动循迹和自动避障，也还存在一些难以解决的bug，但是总体来说整个实验过程还是很有趣的，也令我受益匪浅。

二、实验简介

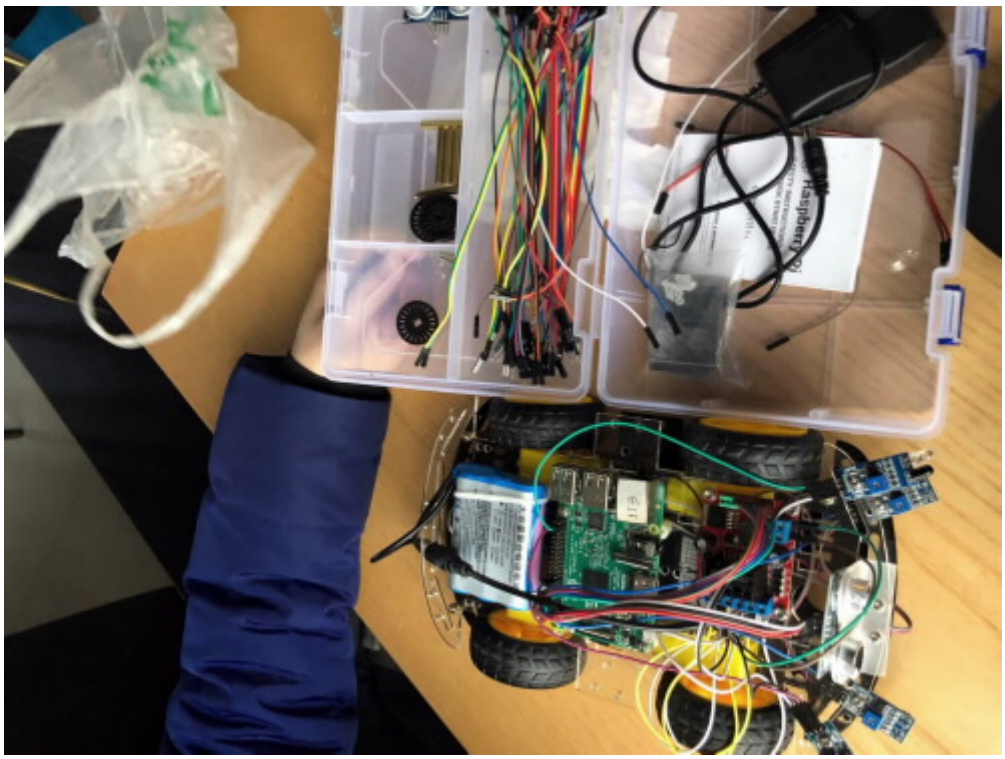
本次实验主要目的是根据课程需求制作树莓派智能小车，包括将小车零件组装成完整的小车和将树莓派等电路硬件组装到小车上，通过Linux系统编写简单的Python程序来调用树莓派的GPIO接口，来实现小车的手动操控、测距、自动循迹和自动避障功能。

下图为智能小车的最终成品展览图



三、实验工具

(1)硬件工具



a.小车材料：Pirobot小车

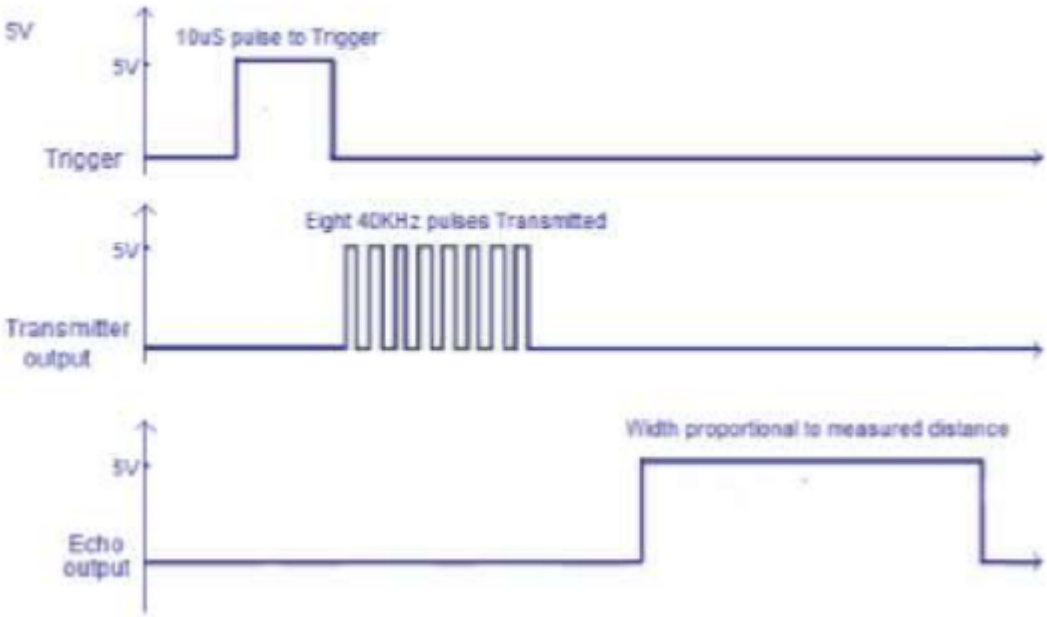
底板*2
数据线*N
车轮*4
驱动器*4
螺丝螺母*N
面包板*1
电源*1
变压器*1

b.树莓派

树莓派是英国一个非盈利机构树莓派基金会开发的一种卡片式计算机，最初的目的是用于对少年儿童进行计算机普及教育。第一代树莓派产品发布于 2012 年 2 月。目前最新的版本树莓派 3-B+ 于 2018 年 3 月 14 日（这一天被叫做 π 日）发布，它采用博通公司的处理器芯片 BCM2837 作为核心处理器，内核是 Cortex-A53 架构，是一个 4 核的 64 位ARM处理器，主频可以达到 1.4GHz，此外还带有 VideoCore-IV 的图像处理单元 GPU。板上支持无线网络和蓝牙，并可以通过 HDMI 接口输出高清视频。 由于树莓派的成功，其他一些计算机开发商也仿照树莓派开发了类似的卡片式计算机产品。 树莓派这类计算机结构简单、体积小、耗电低，却拥有与普通计算机几乎相同的功能和性能，可以很方便地植入各种应用系统中。这种具有计算机的基本结构但又不具备普通计算机形态的计算机，业界称为 “嵌入式计算机”。所谓的 “嵌入式” 是指它是嵌在产品中的，是面向产品的专用计算机。人们看到的只是产品本身，看不到计算机。目前 98% 的计算机产品都属于嵌入式计算机。家用冰箱、微波炉、洗衣机、空调都有它们的存在。

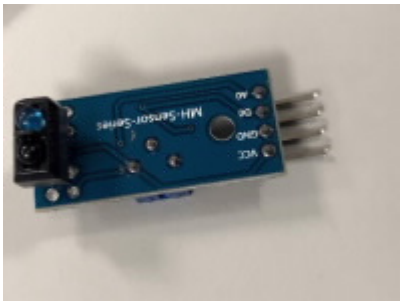
c. HC - SR04 超声波传感器

功能：发射超声波和接收回声的距离测试
组成：包括超声波发射器、接收器与控制电路。
基本工作原理：超声波测距模块一触发信号后发射超声波，当超声波投射到物体而反射回来时， 模块输出一回响信号， 以触发信号和回响信号间的时间差，来判定物体的距离。
特点：敏感范围大，无视觉盲区，不受障碍物干扰
工作原理：
①采用 I/O 触发测距，给至少 10us 的高电平信号；
②模块自动发送 8 个 40kHZ 的方波，自动检测是否有信号返回；
③有信号返回，I/O输出一高电平，高电平持续的时间就是超声波从发射到返回的时间。
测试距离=(高电平时间*声速(340M/S))/2



d. 红外线探测器

该传感器模块对环境光线适应能力强，其具有一对红外线发射与接收管，发射管发射出一定频率的红外线，当检测方向遇到障碍物（反射面）时，红外线反射回来被接 收管接收，经过比较器电路处理之后，绿色指示灯会亮起，同时信号输出接口输出数字信号（一个低电平 信号），可通过电位器旋钮调节检测距离，有效距离范围 2～30cm，工作电压为3.3V-5V。 该传感器的探测距离可以通过电位器调节、具有干扰小、便于装配、使用方便等特点，可以广泛应用于机器人避障、避障小车、流水线计数及黑白线循迹等众多场合。



(2)软件工具

a.开发环境

windows 10专业版操作系统

Windows 10是美国微软公司研发的跨平台及设备应用的操作系统。是微软发布的最后一个独立Windows版本。Windows 10共有7个发行版本，分别面向不同用户和设备。

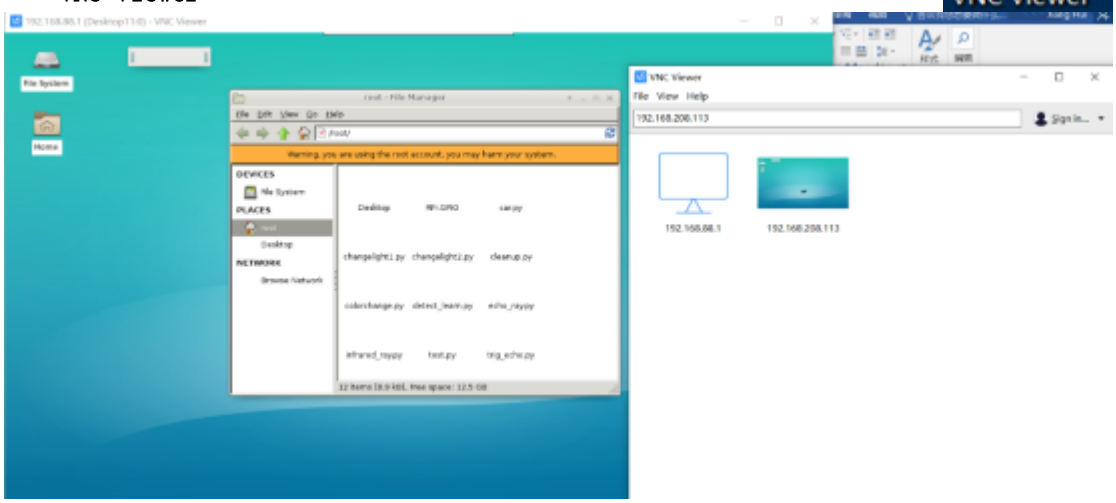
以window 10家庭版为基础，增添了管理设备和应用，保护敏感的企业数据，支持远程和移动办公，使用云计算技术。另外，它还带有Windows Update for Business，微软承诺该功能可以降低管理成本、控制更新部署，让用户更快地获得安全补丁软件。

Windows 10系统成为了智能手机、PC、平板、Xbox One、物联网和其他各种办公设备的心脏，使设备之间提供无缝的操作体验。

Windows 10操作系统在易用性和安全性方面有了极大的提升，除了针对云服务、智能移动设备、自然人机交互等新技术进行融台外，还对固态硬盘、生物识别、高分辨率屏幕等硬件进行了优化完善与支持。从技术角度来讲，Windows 10操作系统是一款优秀的消费级别操作系统。（摘自百度百科）

b.远程桌面操控

VNC Viewer



VNC (Virtual Network Console)是虚拟网络控制台的缩写。它是一款优秀的远程控制工具软件，由著名的 AT&T 的欧洲研究实验室开发的。VNC 是在基于 UNIX 和 Linux 操作系统

的免费的开源软件，远程控制能力强大，高效实用，其性能可以和 Windows 和 MAC 中的任何远程控制软件媲美。 在 Linux 中，VNC 包括以下四个命
令：vncserver，vncviewer，vncpasswd，和 vncconnect。大多数情况下用户只需要其中的两个命令：vncserver 和 vncviewer。
VNC基本上是由两部分组成：一部分是客户端的应用程序(vncviewer)；另外一部分是服务器端的应用程序(vncserver)。VNC的基本运行原理和一些Windows下的远程控制软件很相像。VNC的服务器端应用程序在UNIX和Linux操作系统中适应性很强，图形用户界面十分友好，看上去和Windows下的软件界面也很类似。在任何安装了客户端的应用程
序(vncviewer)的Linux平台的计算机都能十分方便地和安装了服务器端的应用程
序(vncserver)的计算机相互连接。另外，服务器端 (vncserver)还内建了Java Web接口，这样用户通过服务器端对其他计算机的操作就能通过Netscape显示出来了，这样的操作过程和显示方式比较直观方便。（摘自百度百科）
在本项目中使用的是VNC Viewer

c.编程语言

Python

python是一种动态的、面向对象的脚本语言，最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越来越多被用于独立的、大型项目的开发。
Python在设计上坚持了清晰划一的风格，这使得Python成为一门易读、易维护，并且被大量用户所欢迎的、用途广泛的语言。
设计者开发时总的指导思想是，对于一个特定的问题，只要有一种最好的方法来解决就好了。这在由Tim Peters写的Python格言（称为The Zen of Python）里面表述为：There should be one-- and preferably only one --obvious way to do it. 这正好和Perl语言（另一种功能类似的高级动态语言）的中心思想TMTOWTDI（There's More Than One Way To Do It）完全相反。
Python的作者有意的设计限制性很强的语法，使得不好的编程习惯（例如if语句的下一行不向右缩进）都不能通过编译。其中很重要的一项就是Python的缩进规则。
一个和其他大多数语言（如C）的区别就是，一个模块的界限，完全是由每行的首字符在这一行的位置来决定的（而C语言是用一对花括号{}来明确的定出模块的边界的，与字符的位置毫无关系）。这一点曾经引起过争议。因为自从C这类的语言诞生后，语言的语法含义与字符的排列方式分离开来，曾经被认为是一种程序语言的进步。不过不可否认的是，通过强制程序员们缩进（包括if，for和函数定义等所有需要使用模块的地方），Python确实使得程序更加清晰和美观。（摘自百度百科）

d. 树莓派 I/O 模块

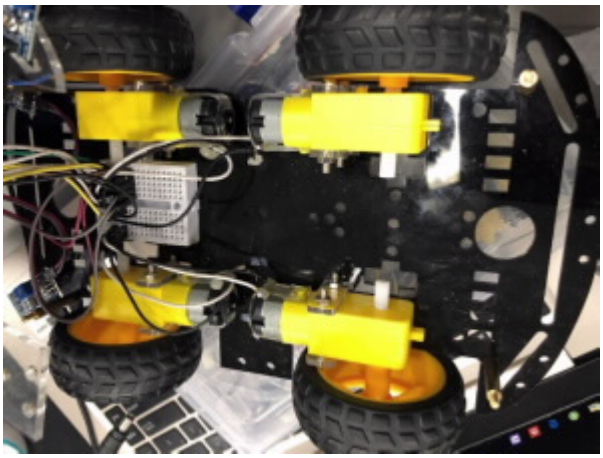
python GPIO模块

为树莓派 GPIO 开发的 Python 模块名为 RPi.GPIO
树莓派可以直接控制电子元件，是因为它提供了GPIO区，所谓GPIO 就是可以自由配置引脚的模式，比如可以配置成输出模式从而输出高低电平，或者可以配置成输入模式接收电平信号，正因为树莓派的 GPIO 功能，让它看起来似乎更加底层化了。
右图为树莓派上GPIO引脚的编码，本次实验中我全程采用物理引脚号

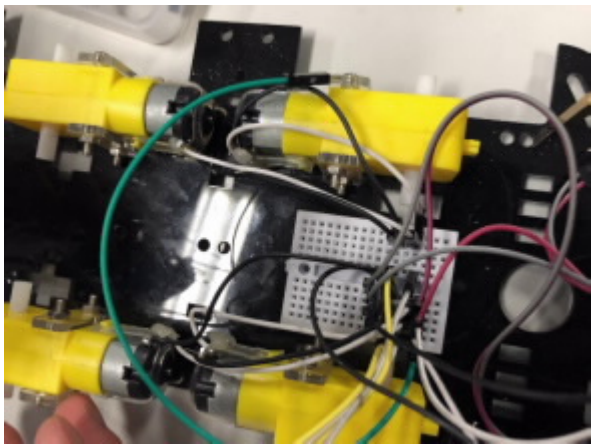
P1: The Main GPIO connector							
WiringPi	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi
		3.3v	1	2	5v		
8	GPIO2	SDA1	3	4	5v		
9	GPIO3	SCL1	5	6	GND		
7	GPIO4	GCLK	7	8	TxD	GPIO14	15
		GND	9	10	RxD	GPIO15	16
0	GPIO17	GEN0	11	12	GEN1	GPIO18	1
2	GPIO27	GEN2	13	14	GND		
3	GPIO22	GEN3	15	16	GEN4	GPIO23	4
		3.3v	17	18	GEN5	GPIO24	5
12	GPIO10	MOSI	19	20	GND		
13	GPIO9	MISO	21	22	GEN6	GPIO25	6
14	GPIO11	SCLK	23	24	CE0	GPIO8	10
		GND	25	26	CE1	GPIO7	11
	ID-SD		27	28		ID-SC	
	GPIO5		29	30	GND		
	GPIO6		31	32		GPIO12	
	GPIO13		33	34	GND		
	GPIO19		35	36		GPIO16	
	GPIO26		37	38		GPIO20	
		GND	39	40		GPIO21	
WiringPi	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi

四、小车组装

1. 底盘结构

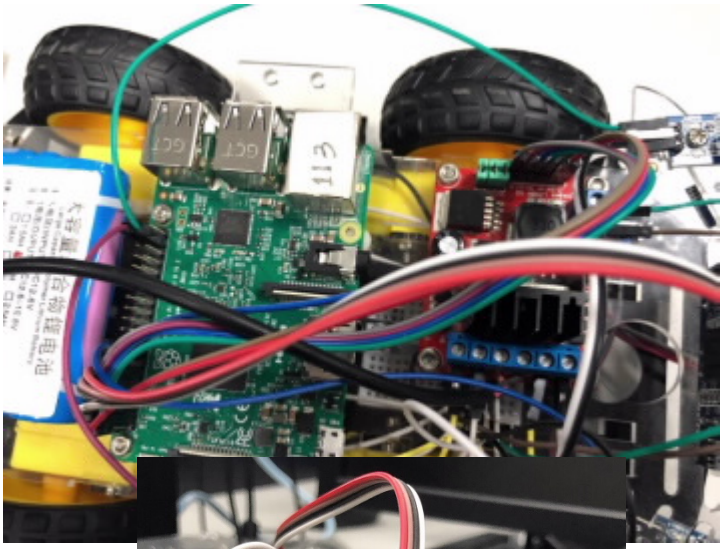


为保证同一侧车轮的转向相同，马达的相应接口并联到一起，并通过两对导线接入顶盘的变压器上面使得树莓派能够直接控制电压



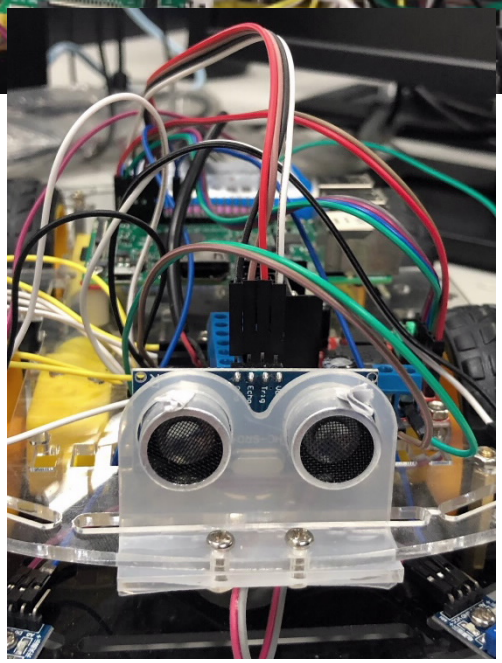
2. 树莓派连接

通过变压器给树莓派供电，并将树莓派相应的GPIO接口连接到变压器上用以控制底盘车轮的电压，同时接入控制其他外设的导线



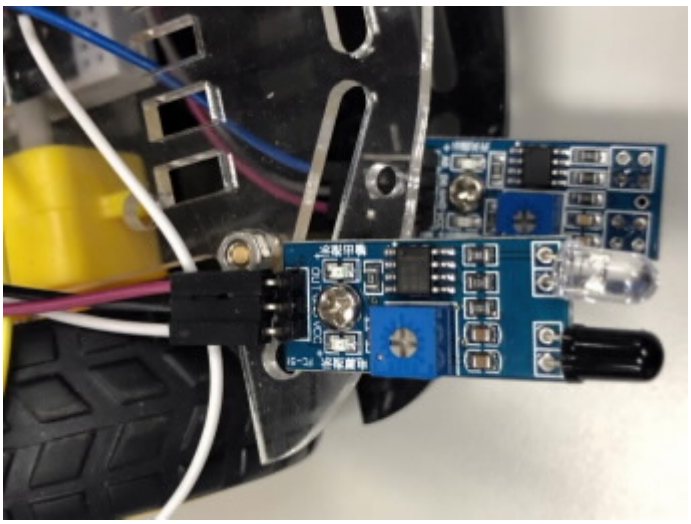
3. 超声波探测器连接

共有4根导线
vcc接入树莓派的5V供电接口
gnd接入树莓派的地线接口
trig和echo为输入输出接口，接入树莓派的GPIO接口



4. 红外线探测器连接

分为上下两组红外线探测器
顶盘的用于自动避障
底盘的用于自动循迹
各自接入3根导线
vcc接入树莓派5v供电接口
gnd接入变压器提供的地线接口
out为输出接口，接入GPIO给树莓派传输信息



五、功能实现

0、概述

在python 中导入GPIO模块后，通过对GPIO的接口的调用，设置输入与输出模式。树莓派通过GPIO接口与外设进行信息交互，以此达到控制小车的目的。交互的信息为1和0代表的高电平和低电平，根据获得的信号外设会执行相应的操作，或者外设接收到外界信号后输入或输出高低电平给树莓派。主要外设有超声波探测器、红外线探测器、车轮马达。

```
def start():
    G.setmode(G.BOARD)

    #right wheels
    G.setup(R1,G.OUT)
    G.setup(R2,G.OUT)
    G.output(R1,0)
    G.output(R2,0)

    #left wheels
    G.setup(L1,G.OUT)
    G.setup(L2,G.OUT)
    G.output(L1,0)
    G.output(L2,0)

    #echo
    G.setup(Trig_rec,G.IN)
    G.setup(Trig,G.OUT)
    G.output(Trig,0)

    #ray
    G.setup(R_ray,G.IN)
    G.setup(L_ray,G.IN)
    G.setup(R_up_ray,G.IN)
    G.setup(L_up_ray,G.IN)

    print("the car is ready")
```

1、手动操控

a.原理

通过GPIO输出高低电压差的方向来控制车轮马达转动的方向。需要前进或后退时，将

所有车轮全部设置为某一侧高；需要左右转弯时，将其中一侧车轮设为静止，以左右速度差来进行原地转向；需要静止时，将马达两侧输出设为0；主程序用比较低效的轮询来

b.代码实现

```
def go_straight():
    G.output(R1,1)
    G.output(R2,0)
    G.output(L1,1)
    G.output(L2,0)
    #print("going strghtly")

def go_back():
    G.output(R1,0)
    G.output(R2,1)
    G.output(L1,0)
    G.output(L2,1)
    #print("going back")

def turn_left():
    G.output(R1,1)
    G.output(R2,0)
    G.output(L1,0)
    G.output(L2,0)
    #print("turning left")

def turn_right():
    G.output(R1,0)
    G.output(R2,0)
    G.output(L1,1)
    G.output(L2,0)
    #print("turning right")

def wait():
    G.output(R1,0)
    G.output(R2,0)
    G.output(L1,0)
    G.output(L2,0)
    #print("waiting instruction")


while True:
        print("please enter nest instruction:",end = "")
        order = input()
        if order == "w":
            go_straight()
```

```
        elif order == "a":
            turn_left()
        elif order == "d":
            turn_right()
        elif order == "s":
            go_back()
        elif order == "q":
            wait()
        elif order == "c":
            calculate_distance()
        elif order == "auto":
            auto()
        elif order == "p":
            auto_p()
        elif order == "stop":
            break
        else:
            print("no such instruction")
```

2、测距

a.原理

使用超声波探测器，超声波探测器共有4个接口

trig：接收GPIO的输出，当输出高电平时发送超声波

echo：向GPIO输入信号，在发射超声波之后变为高电平接收超声波的回声

vcc：电源，5V

gnd：地线

通过GPIO输出 20us 的高电平信号给超声波探测仪的Trig接口，模块自发送一连串超声波信号，echo接口的输出会变成高电平，并自动检测是否有信号返回。当接收到返回信号时，echo接口会从高电平信号变为低电平，所以通过GPIO读取高电平持续的时间即可超声波从发射到返回的时间。距离=34000*t/2

b.代码实现

```
def calculate_distance():
    G.output(Trig,G.HIGH)
    time.sleep(0.00002)
    G.output(Trig,G.LOW)
    while G.input(Trig_rec) == 0:
        pass
    starttime = time.time()
    while G.input(Trig_rec) == 1:
        pass
    stoptime = time.time()
```

```
    totaltime = stoptime - starttime
    distance = (totaltime * 34000)/2
    print('distance is : ',distance)
    return distance
```

3、自动循迹

a.原理

使用红外线探测器*2，使用红外线探测器的3个接口：

vcc：电源，3.3-5V

gnd：地线

out：输出接口，接收到遇到障碍物反射的红外线时输出一个低电平信号

通过在小车底盘前方两侧安装两个红外线探测器，用较为低效的轮询方式检测地面的黑线，当红外线被黑线吸收时，out输出高电平，说明黑线不在小车中间，根据左或右的高电平信号进行相应的转弯，使黑线保持在小车中间

```
#每2秒检测一次前方距离，当前方障碍物距离低于安全距离时，让小车停止
def cal_dist_auto():
    global IsSafe
    dist = calculate_distance()
    while dist >= Safe_dist:
        dist = calculate_distance()
        time.sleep(2)
    IsSafe = False
    wait()
    time.sleep(2)
    IsSafe = True
    print("unsafe distance, waiting for new instruction")

#这个函数用来强制让小车停止
def control():
    command = input()
    while True:
        if command == "q":
            stop()
            exit("the car has stopped,please restart the program")


def auto():
    print("auto mode")
    global IsSafe
    thread.start_new_thread(control,()) #这个线程用来监测是否要让小车停止运行
    thread.start_new_thread(cal_dist_auto,())   #这个线程用来自动测距保证前方安全
    go_straight()
```

```
    while IsSafe:       #IsSafe是自动测距的全局变量
        right_ray = G.input(R_ray)
        left_ray = G.input(L_ray)
        if right_ray == 0 and left_ray ==0:
            go_straight()
            continue
        if left_ray == 0:
            turn_left()
            continue
        if right_ray == 0:
            turn_right()
            continue
    wait()
    time.sleep(2)
    return
```

4、自动避障

a.原理

利用超声波探测器和红外线探测器*2

在小车正前方安装一个超声波探测器，并在小车前方两侧安装红外线探测器，开启一个自动测距线程每0.6秒自动测距一次，并在主循坏不断获取两侧红外线探测器的信号，如果前方距离小于安全距离或者两侧红外装置探测到障碍物，小车就会先后退一小段距离，再根据两侧红外探测的信号决定往哪一侧拐弯

```
def cal_dist_auto_p():
    global IsSafe
    dist = calculate_distance()
    while True:
        if dist >= Safe_dist:
            dist = calculate_distance()
            IsSafe = False
        else:
            IsSafe = True
        time.sleep(0.6)


def auto_p():
    print("auto mode of preventing barrier")
    global IsSafe
    thread.start_new_thread(control,())
#    thread.start_new_thread(cal_dist_auto_p,())
    go_straight()
    while True:
```

```
dist = calculate_distance()
right_up_ray = G.input(R_up_ray)
left_up_ray = G.input(L_up_ray)
if dist >= Safe_dist and right_up_ray == 1 and left_up_ray == 1:
    IsSafe = True
else:
    IsSafe = False
if IsSafe:
    go_straight()
    print("safe")
else:
    right_up_ray = G.input(R_up_ray)
    left_up_ray = G.input(L_up_ray)
    go_back()
    print("unsafe")
    time.sleep(0.5)
    if right_up_ray == 1:
        turn_right()
    else:
        turn_left()
    time.sleep(0.7)
right_up_ray = 0
left_up_ray = 0
```

六、实验总结

1.经过一个学期的学习和尝试，整体上实现了智能小车，完成了大部分的预期目标，比如手动操控、测距和大体上完整的自动循迹和自动避障。在学习这门课程之前，作为软件工程专业的学生，此前我对硬件几乎没有一点概念，都是在IDE里写单纯的代码，这门课让我第一次接触到软硬结合的实现，对硬件层次有了更深的理解，也体会到了软件操控硬件的不易，尤其是在刚开始的阶段，完全不知所措，不知道要用什么语言和什么环境去使用树莓派，对各种硬件的名词也是完全没有了解过，导致初期学习的时候到处碰壁，解决一个小接口的问题就要花费好久的时间，理解一些概念也需要很久的时间。小车的制作过

程中认识了几个同学，一起讨论一起努力，都基本完成了各自的预期目标。坚持了一个学期下来，各方面能力都有提升，也锻炼了自己思考问题并付与实践的能力。

2.还有一些没能解决的问题，比如没有完成小车车速调控的功能，循迹和避障方面因此在一些情况下会因为车速过快而导致和预期出现一些偏差。还因为不太熟悉python的一些用法导致在多线程的情况下出现不同线程的一些冲突。另外，代码实现中采取了较为低效的轮询方式，不仅使得运行效率大大降低，也失去了一些实现更灵活操作的机会。没能使用更加高效的方式，是一大遗憾。

171250506

江辉

2019年1月11日