一、实验内容与要求

1.1 实验内容

智能巡逻小狗在房屋周围道路上巡逻，当其发现前方有障碍物出现，将使用摄像头进行人脸识别。若没有识别到人脸，则说明为一般障碍物，小狗将自动绕过障碍物并回到原来道路上继续巡逻。若识别到人脸，将再辨别是否是主人归来，如果识别出是陌生人，小狗会把陌生人的照片拍照上传到主人邮箱中，提醒主人有陌生人来访，并发出警告声。若识别到主人归来，小狗将打开风扇为主人吹风降温，并接受主人命令。当主人下达蓝色命令，小狗将点亮彩灯并播放音频欢迎主人；当主人下达绿色命令，小狗将跟随主人前进；当主人下达红色命令，小狗将自动沿道路返回狗窝。

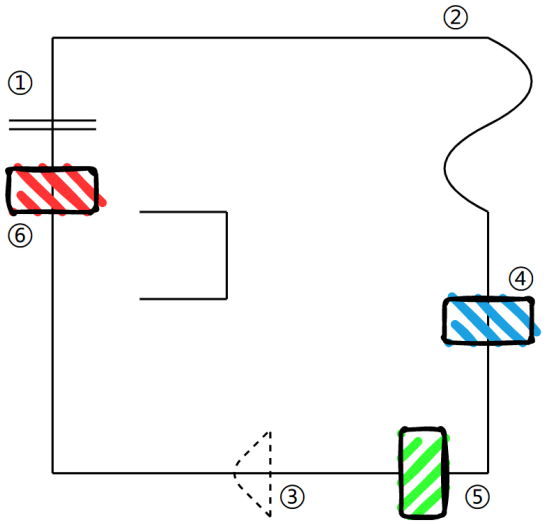

图 1 场地设计示意图

① 倒车入库线 ② S 型弯路 ③ 障碍物 ④ 蓝色命令牌 ⑤ 绿色命令牌 ⑥ 红色命令牌

1.2 实验要求

(1) 使用尽量多的模块（至少要有 4 个）；

(2) 小狗能稳定地沿道路巡逻，能处理复杂的道路形状；

(3) 在遇到非人类的普通障碍物后懂得及时绕障并在最后回到原定线路；

(4) 准确识别出主人与陌生人；

(5) 精准地辨别三种颜色命令，并正常执行命令任务；

(6) 小狗各个动作之间越连贯越好；

(7) 自行发挥的部分要丰富，有创造性。

1.3 实验目的

1. 小狗通过按键实现启停；
2. 小狗检测复杂弯曲的黑线，能够智能循迹，按照设计的路线来自动行驶，完成无间断行驶；
3. 小狗遇到障碍物自动避障，当按照规定的线路行驶时遇到非人类的普通障碍物，能够实现自动避障，在绕过障碍物后，继续行驶而不受障碍物影响；
4. 小狗接受返回车库（狗窝）命令后，能自动检测车库线，并自动停入车库；
5. 小狗有彩灯和音频模块，接受命令后点亮彩灯播放音频；
6. 小狗能进行红外跟随，跟随主人前进；
7. 小狗有人脸识别模块，分辨主人与陌生人；
8. 小狗有颜色识别模块，接收主人不同的颜色命令。

除了希望小狗能良好地模拟线路之外，该实验要求小组的成员进行分工合作，各司其职，齐心协力完成该工程问题，旨在培养同学们的团队协作能力、问题的解决能力、实际与理论的结合能力。

二、实验原理与硬件连线

2.1 实验原理

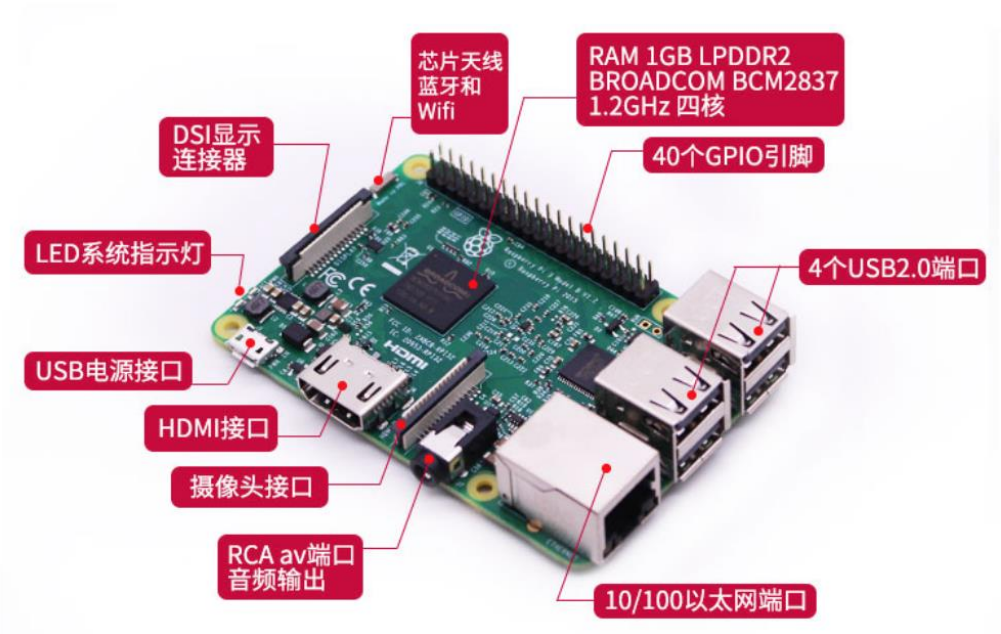

图 2 树莓派 4B 主控板

巡线模块实验原理：

红外传感器巡线的基本原理是利用物体的反射性质。本次实验是巡黑线行驶。当红外线发射到黑线上时会被黑线吸收掉，发射到其它颜色的材料上会有反射到红外的接收管上。我们根据反射的不同编写相应的代码完成小车的巡线功能。

检测中黑线时，模块的指示灯亮，输出低电平；否则，指示灯灭，输出高电平。

本次实验采用的四路红外传感器如图 3 所示，它的引脚分别连接在树莓派主控板上 BCM 编码为 3、5、4 和 18 的口上。

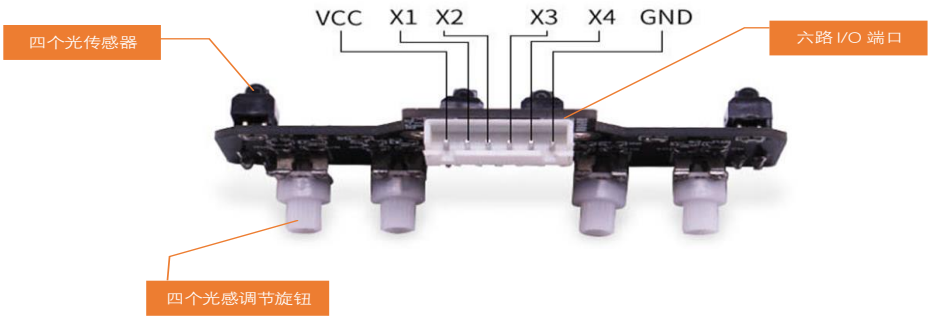

图 3 四路寻迹模块

红外避障模块实验原理：

红外传感器具有一对红外发射与接收管，发射管发射出一定频率的红外线，当检测方向遇到跟随物时，红外线红外线反射回来被接收管接收。它常用于安装在小车上，判断前方是否有跟随物。可通过电位器设置阀值。正前方有障碍时绿灯亮起，OUT 引脚为低电平，反之为高电平。

红外传感器跟随的基本原理是利用物体的反射性质，在一定范围内，如果没有跟随物，发射出去的红外线，因为传播的距离越来越远而逐渐减弱，最后消失。如果有跟随物，红外线遇到跟随物，被反射到达传感器接收头。我们本次实验采用的是两路红外传感器分别连接在树莓派主控板上的 BCM 编码的 12，17 口上。我们通过检测两端口的电平来判断跟随物的情况，并做出相应的跟随动作。红外跟随模块如图 4 所示。

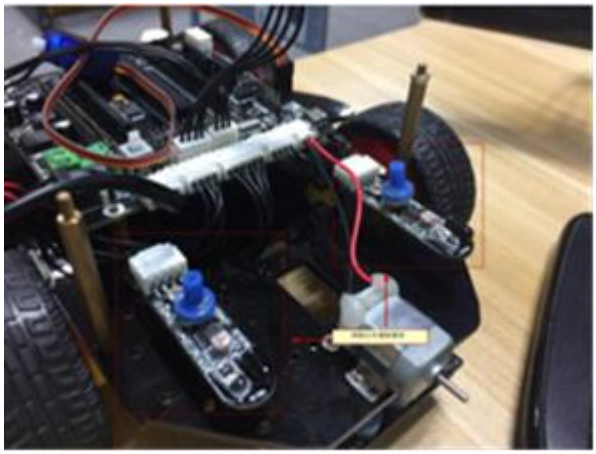

图 4 红外跟随模块

超声波模块实验原理：

超声波模块是利用超声波的特性检测距离的传感器，其带有两个超声波探头，分别作为发射和接收超声波。其测量的范围是 0-500cm。超声波模块实物图如图 5 所示。

图 5 超声波模块实物图

超声波原理：先向超声波 Trig 引脚输入至少 10us 的高电平信号，触发超声波模块的测距功能。接着 Trig 引脚会自动发出 8 个 40khz 的超声波脉冲，并自动将 Echo 脚的电平拉高，一旦检测到回波信号则将 Echo 脚的电平拉低。高电平持续的时间就是超声波从发射到返回的时间。此时即可计算出实际的距离：距离=高电平时间*声速(340M/S)/2。超声波模块引脚如图 6 所示，超声波发射与接受示意图如图 7 所示。

引脚名称	说　明
Vcc	电源 5 V
Trig	触发引脚
Echo	回馈引脚
Gnd	地

图 6 超声波模块引脚

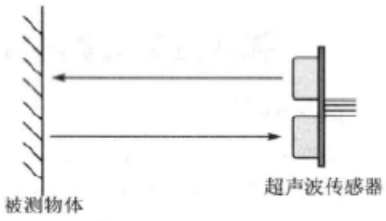

图 7 超声波发射与接收示意图

七彩灯模块实验原理：

RGB 三色灯和普通的 LED 灯其实没有什么不同，只是在封装上，RGB 灯内封装了三个 LED（红，绿，蓝），通过控制三种 LED 亮度（256 种亮度级别可选），可以混合出不同的颜色（256*256*256）。

本次实验的七彩灯探照灯模块实物图如图 8 所示，本实验中采用的 RGB LED 灯是共阴 LED，一个引脚接地，其余的三个 RGB 引脚分别接在树莓派主控板上的 BCM 编码 22、27、24 引脚上；同时每个 LED 灯需要串联一个 220 欧的电阻作为限流电阻，我们只需在树莓派主控板上控制相应的引脚为高电平，即可点亮相应的 LED。

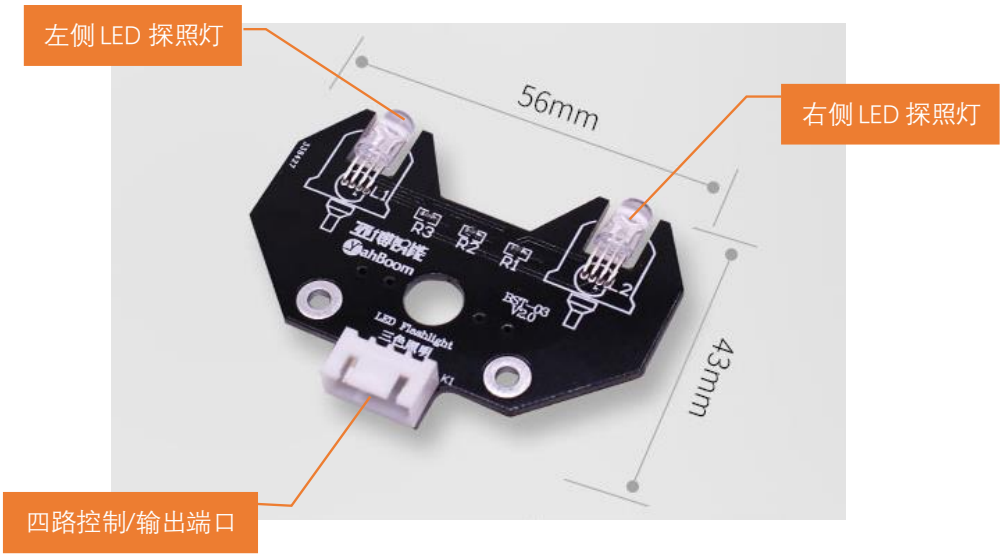

图 8 七彩探照灯模块

电机模块实验原理：

4 路直流减速电机的实物图如图 9 所示，它的控制采用的是 TB6612FNG 驱动芯片。通过控制驱动芯片的 AIN1、AIN2、BIN1、BIN2、PWMA、PWMB 信号的电平高低来控制电机的正转、反转和停止。本次实验主要是控制 AIN1、AIN2、BIN1 和 BIN2 的电平状态，进而通过控制 PWMA 和 PWMB 在 0-255 之间控制小车的速度,一路 PWM 控制小车一侧电机的速度。电机模块和控制器部分如图 10 所示。

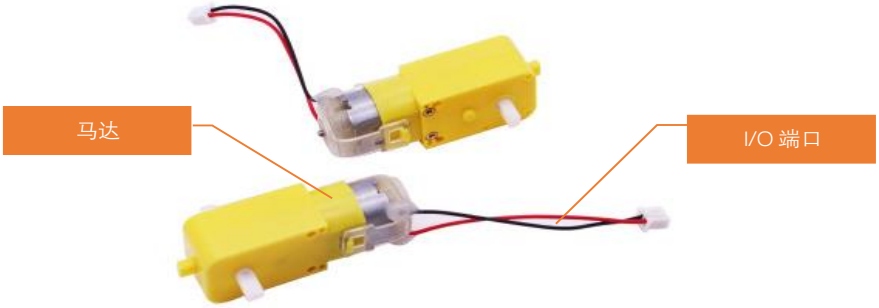

图 9 直流减速电机

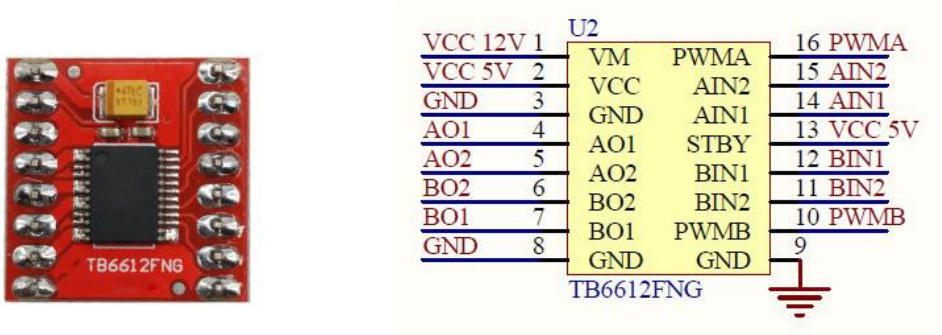

图 10 TB6612FNG 驱动芯片及其管脚示意图

按钮模块消抖原理：

按键消抖：通常我们的按键开关一般都是机械弹性开关。当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关子在闭合时不会马上就能稳定地接通，在断开时也不会一下子彻底断开，而是在闭合和断开时会伴随着一连串的抖动。按键抖动的状态图如图 11 所示。

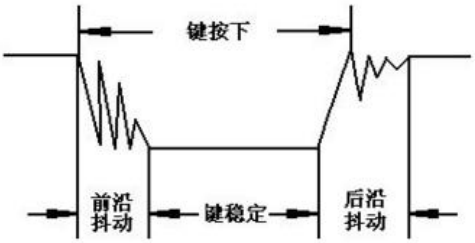

图 11 按键抖动状态图

抖动时间一般都是由按键的机械特性决定的，一般都会在 10ms 以内，为了确保程序对按键的一次闭合和一次断开只响应一次，必须进行按键的消抖处理，有硬件消抖和软件消抖。

其中，软件消抖是指检测出键闭合后执行一个延时程序，产生 5ms~10ms 的延时，让前沿抖动消失后再一次检测键的状态。如果仍保持闭合状态电平，则确认为真正有键按下。当检测到按键释放后，也要进行 5ms~10ms 延时，待后沿抖动消失后才能转入该键的处理程序。

硬件消抖是在开关两端接一个 0.1uf 的电容。本次实验我们采取软件延时去抖方法。

人脸检测和识别模块原理：

在进行人脸识别前，需要先进行人脸检测（Face Detection）。人脸检测是人脸识别最基本的任务，最常见的方法是使用基于 Haar 特征（如图 12 所示）的级联分类器，这是 Paul Viola 和 Michael Jones 于 2001 年在他们的论文《Rapid Object Detection using a Boosted Cascade of Simple Features》中提出的。这是一种基于机器学习的方法，它从许多正面和负面的图像中训练级联函数，然后用于检测其他图像中的对象。由于用来训练的时间有限，自己训练的模型精度也可能达不到要求，我们直接使用了 OpenCV 官方的预训练面部分类器 haarcascade_frontalface_default.xml，调用 CascadeClassifier 函数进行加载，并通过其子函数 detectMultiScale 获得面部信息。

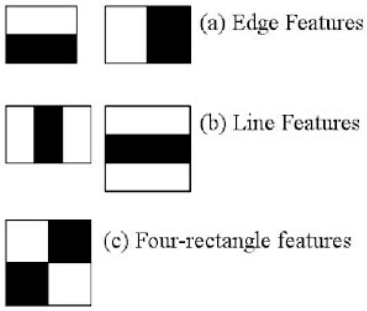

图 12 Haar 特征

为减小干扰，我们设置面部长度和宽度大于 100 像素时为检测到人脸，此时程序进入人脸识别阶段。由于在实际操作中遇到了 opencv-contrib 库无法安装的问题，我们无法使用 OpenCV 的识别器进行人脸识别，因此我们转而使用腾讯云 API 实现这一功能。将访客照片进行 base64 编码后传入腾讯云的人脸验证 API，发出查询请求后便可得知该访客是否在人员库中。

颜色识别模块原理：

通过 OpenCV+python 实现，主要是以下步骤：

1. 寻找 HSV 数据，确定颜色色域的阈值没，HSV(hue,saturation,value)颜色空间的模型对应于圆柱坐标系中的一个圆锥形子集，圆锥的顶面对应于 V=1. 它包含 RGB 模型中的 R=1，G=1，B=1 三个面，所代表的颜色较亮。色彩 H 由绕 V 轴的旋转角给定。红色对应于角度 0°，绿色对应于角度 120°，蓝色对应于角度 240°。在 HSV 颜色模型中，每一种颜色和它的补色相差 180°。饱和度 S 取值从 0 到 1，所以圆锥顶面的半径为 1 。HSV 颜色模型所代表的颜色域是 CIE 色度图的一个子集，这个模型中饱和度为百分之百的颜色，其纯度一般小于百分之百。在圆锥的顶点(即原点)处，V=0,H 和 S 无定义，代表黑色。圆锥的顶面中心处 S=0，V=1,H 无定义，代表白色。从该点到原点代表亮度渐暗的灰色，即具有不同灰度的灰色。对于这些点，S=0,H 的值无定义。可以说，HSV 模型中的 V 轴对应于 RGB 颜色空间中的主对角线。在圆锥顶面的圆周上的颜色，V=1，S=1,这种颜色是纯色。HSV 模型对应于画家配色的方法。画家用改变色浓和色深的方法从某种纯色获得不同色调的颜色，在一种纯色中加入白色以改变色浓，加入黑色以改变色深，同时加入不同比例的白色，黑色即可获得各种不同的色调。各颜色的 HSV 颜色空间参数值如图 13。

	黑	灰	白	红		橙	黄	绿	青	蓝	紫
hmin	0	0	0	0	156	11	26	35	78	100	125
hmax	180	180	180	10	180	25	34；	77	99	124	155
smin	0	0	0	43		43	43	43	43	43	43
smax	255	43	30	255		255	255	255	255	255	255
vmin	0	46	221	46		46	46	46	46	46	46
vmax	46	220	255	255		255	255	255	255	255	255

图 13 HSV 颜色空间图

2. 本次实验需识别红、绿、蓝三种颜色，将这三种颜色添加进颜色库，设定好对应的色域范围。

3. 通过 cv2 库的 videocapture 函数调用树莓派的摄像头进行拍照，将获取的图片通过 cv2 的 inRange 函数进行二值化操作，主要是将在两个阈值内的像素值设为白色（255），而不在阈值区间内的像素值设置为黑色（0）。

4. 对上一步处理好的图像通过 cv2 的 erode 函数进行腐蚀操作，将内核 B 划过图像,将内核 B 覆盖区域的最小相素值提取，并代替锚点位置的相素。

5. 对上一步处理好的图像通过 cv2 的 GaussianBlur 函数进行高斯滤波，去除噪声。

6. 对上一步处理好的图像通过 cv2 的 findContours 函数进行寻找轮廓操作。
7. 对上一步处理好的图像通过 cv2 的 countArea 函数计算图像轮廓的面积。
8. 依次遍历颜色库，比较得出图像轮廓面积最大的一种颜色即为所识别出的颜色。

2.2 硬件连线

将各模块的部件安装好，对应表查找相应部件的接口引脚（见图 14），在扩展板上相应位置接线（见图 15）。将顶板固定好并确保个部件安装牢固。树莓派的 GPIO 引脚与拓展板的对应位置相连，蓝牙模块接到树莓派上，通过拓展板的接口，连接上四个电机驱动四个车轮，通过接口控制两个红外寻光避障模块以及前方的风扇马达，另外车前方的超声波模块和舵机的旋转、七彩灯的变换都是通过拓展版的接口进行控制。

通过 wifi 热点的连接，计算机把程序在树莓派上编译并且执行，通过树莓派的 GPIO 接口传递电压信号给拓展板，拓展板根据不同的命令控制相应的模块实现对应的功能.

功能	原理图编号	Arduino	STM32	51	树莓派	wiringPi	BCM	备注
左电机前	AIN2	8	PB9	P2.1	IO20	28	20	
左电机后	AIN1	7	PB8	P2.2	IO21	29	21	
右电机前	BIN2	2	PB4	P2.4	IO19	24	19	
右电机后	BIN1	4	PB5	P2.3	IO26	25	26	
左电机PWM	PWMA	6	PB7	P2.0	IO16	27	16	
右电机PWM	PWMB	5	PB6	P2.5	IO13	23	13	
左1	IN2	A2	PC14	P1.1	IO3	9	3	Arduino需要跳线，其他不用跳线
左2	IN1	A1	PC13	P1.0	IO5	21	5	
右1	IN3	A3	PC15	P1.2	IO4	7	4	
右2	IN4	A4	PB12	P1.3	IO18	1	18	
左	IN7	A3	PA6	P1.6	IO12	26	12	Arduino需要跳线，其他不用跳线
右	IN5	A1	PA4	P1.4	IO17	0	17	
左	IN8	A4	PB3	P1.7	IO7	11	7	
右	IN6	A2	PA5	P1.5	IO6	12	10	
灰度传感器	GS	A5	PA1	无	无	无	无	
按键	K2	A0	PA0	P2.7	IO8	10	8	
蜂鸣器	FM	A0	PA0	P2.7	IO8	10	8	
风扇	MOTOR	A5	PA1	P0.0	IO2	8	2	
红色	LED_R	11	PB1	P3.2/INT0	IO22	3	22	
绿色	LED_G	10	PB0	P3.3/INT1	IO27	2	27	
蓝色	LED_B	9	PA7	P3.4/T0	IO24	5	24	
云台	J1	3	PA11	P0.5	IO23	4	23	
发送	SCL_C	13	PA15	P3.7/RD	ID_SC	30	0	
接收	SDA_C	12	PA12	P3.6/WR	ID_SD	31	1	
红外遥控	IRN	A5	PA1	P3.5/T1	IO2	8	2	都需要跳线
电压检测	POWERC	A0	PA0	无	无	无	无	
MOSI	MOS	A3	PB15	P0.2	IO10	12	10	
MISO	MIS	A2	PB14	P0.3	IO9	13	9	
GND	GND	GND	GND	GND	GND	GND	GND	
VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	
CS	CS	A4	PA8	P0.1	IO25	6	25	
SCK	SCK	A1	PB13	P0.4	IO11	14	11	
时钟	SCL	SCL	PB10	P0.7	无	无	无	
数据	SDA	SDA	PB11	P0.6	无	无	无	
自由度1	J1	3	PA11	P0.5	IO23	4	23	
自由度2	J2	A1	PB13	P0.4	IO11	14	11	
自由度3	J3	A2	PB14	P0.3	IO9	13	9	
自由度4	J4	A3	PB15	P0.2	IO10	12	10	
自由度5	J5	A4	PA8	P0.1	IO25	6	25	
自由度6	J6	A5	PA1	P0.0	IO2	8	2	
RST	RST	A5	PA1	P0.0	无	无	无	
SCK	SCK	A1	PB13	P0.4	无	无	无	
MISI	MISI	A2	PB14	P0.3	无	无	无	
MOSI	MOSI	A3	PB15	P0.2	无	无	无	
CS	CS	A4	PA8	P0.1	无	无	无	
A0	A0	A1	PB13	无	无	无	无	
CLK	CLK	A3	PB15	无	无	无	无	
SI	SI	A2	PB14	无	无	无	无	
STM32： 1、左右是从车后面向前面看 2、PB4、PB3、PA15使用时需要关闭jtag								

表 14 模块的硬件接口

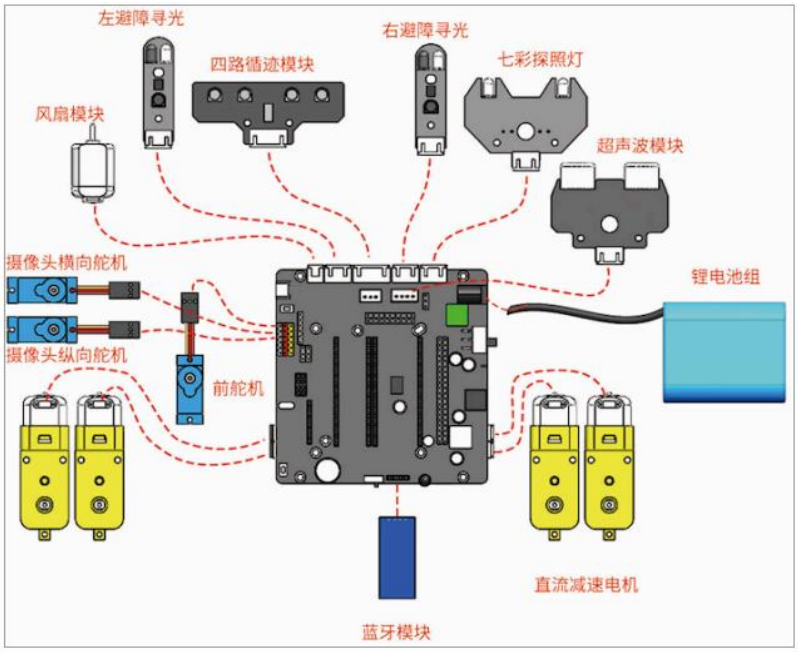

图 15 扩展板接线

三、设计思路、步骤和程序流程图

3.1 设计思路

树莓派是一种信用卡大小的掌上电脑，为编程学习者提供了一个强大的平台，其基于 Linux 系统，可以使用 Python2/3、Scratch、Java、C 等多种语言进行编程；同时树莓派还拥有多种接口和大量引脚，允许用户灵活地对其功能进行扩展。

在体验过树莓派智能小车的各种功能和控制方式之后，我们发现基于现有模块，可以实现循迹、避障、识别等功能，能够以此模拟看门狗的行为模式，因此展开了进一步的实验和研究。

本次实验共用到了风扇模块、七彩灯模块、电机模块、超声波模块、红外避障模块、四路循迹模块、摄像头模块和音频输出模块。

风扇模块在人脸识别通过后启动，表示在炎热的夏天，当主人回家时，智能巡逻小狗会为主人扇风降温。

七彩灯模块的功能有两个。一方面，七彩灯模块可以作为人脸识别是否通过的标志，通过时将闪两下绿灯，否则将闪两下红灯；另一方面，当人脸识别通过，进入颜色识别阶段时，主人出示蓝色卡牌，智能巡逻小狗便会调用七彩灯模块和音频输出模块，为主人闪烁彩灯并播放音频。

电机模块主要供应小车行驶中的动力，通过调节左右电机的转速和转向，可以完成小车的左右转和前后行。

超声波模块负责检测到前方障碍物的距离，智能巡逻小狗将根据到障碍物的距离，在高速巡线、

低速巡线以及停车并开始识别这三种行为中选择一种执行。

红外避障模块负责检测前方近距离是否有障碍物，根据两侧红外避障模块的返回值进行电机控制，以此实现红外跟随功能。

四路循迹模块负责检测正确的线路走向，通过返回值进行电机控制，可以矫正小车的行驶方向，以此保证小车一直行驶在正确的道路上。

摄像头模块负责采集图像信息，在信息输入函数后便可进行人脸的检测和识别。

音频输出模块主要用于在人脸识别通过并出示蓝色卡牌时，为主人播放音频。

3.2 实验步骤

(1) 调试小车的四路循迹模块，调整传感器，使其能够以恰到好处的灵敏度对黑线作出反应；
(2) 完成小车的巡线功能，要求小车能够正确以规定的速度按照所布置的黑线行驶，能够正确地完成左右转，处理曲线、折线等不同的转弯情况；
(3) 测试小车的超声波模块，使其能够正确反映小车和前方的障碍物的距离；
(4) 完成小车的避障功能，要求小车检测到前方近距离有障碍物时，能够正确地绕过障碍物并回到正轨；
(5) 完成小车的倒车入库功能，要求小车能正确地调整好行驶方向，倒车进入指定位置；
(6) 调试小车的红外避障模块，调整传感器，使其能够以恰到好处的灵敏度对前方障碍作出反应；
(7) 完成小车的红外跟随功能，要求小车能在检测到障碍时，对前方障碍进行跟随；
(8) 完成小车的闪灯、播放音频和警报的功能；
(9) 正确配置好视觉相关功能的所需环境，测试小车的摄像头及其云台舵机，使舵机能按指定角度转动，同时解决树莓派自带的拍摄进程和 OpenCV 冲突的问题；
(10) 完成小车的颜色识别功能，要求小车能正确识别图像中的主要颜色，并输出结果；
(11) 完成小车的人脸检测功能，要求小车能正确检测人脸，输出图像中是否有人脸的信息；
(12) 完成小车的人脸识别功能，要求小车能拍照保存当前识别的人脸图像，并正确识别出在库中的人脸，输出是否识别通过的信息；
(13) 完成小车的邮件发送功能，要求小车能将人脸图像作为附件，发送邮件给指定邮箱；
(14) 联立各个功能，将（2）（4）（5）（7）（8）（10）（11）（12）（13）完成的功能以合适的逻辑串联起来，完成设计目标。

3.3 程序流程图

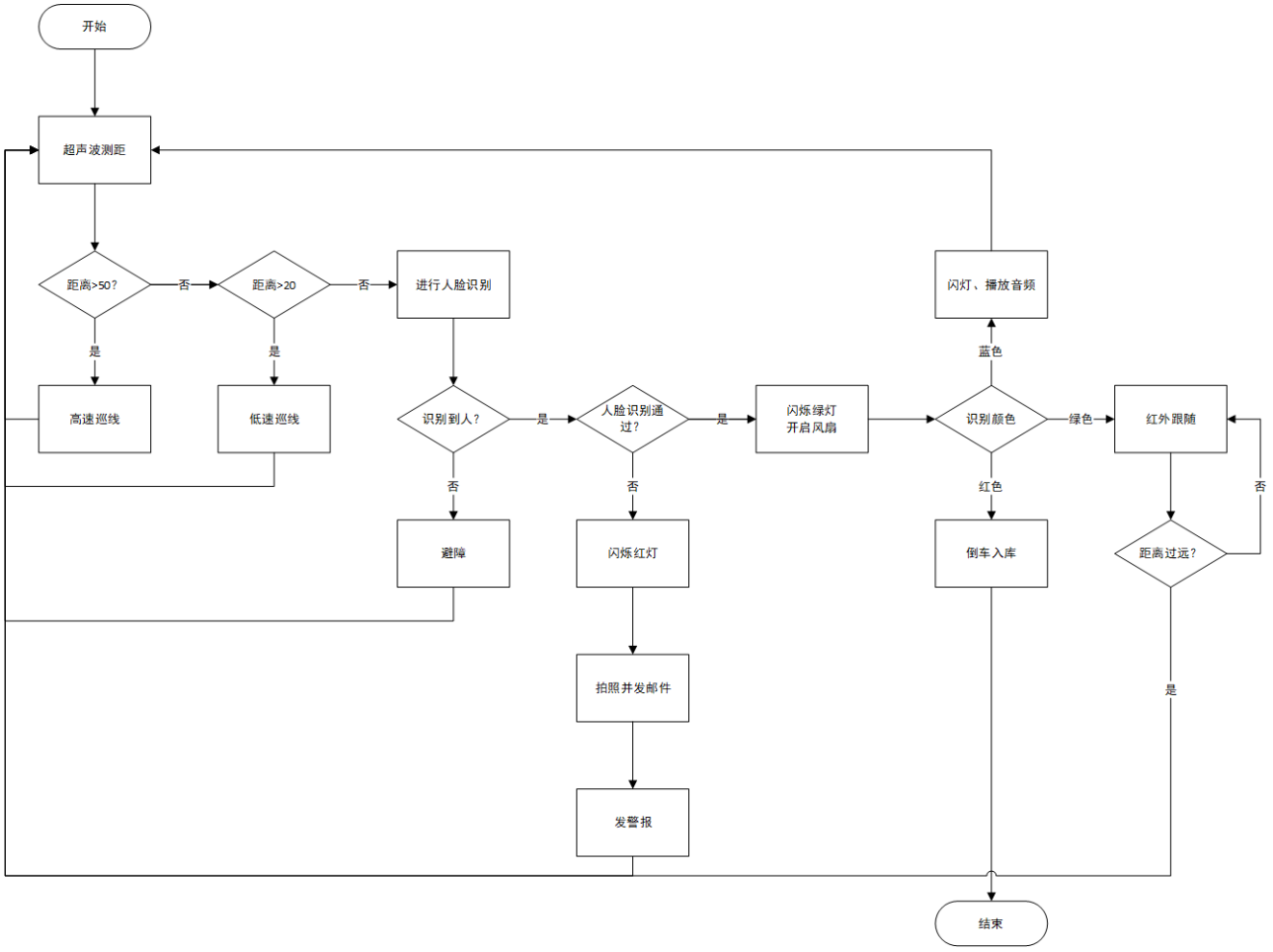

图 16 程序流程图

四、程序清单与执行结果

4.1 程序清单

```
# -*- coding:UTF-8 -*-
import RPi.GPIO as GPIO
import time
import cv2
import collections
import sys
import numpy as np
import base64
import json
```

```
import os
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException
from tencentcloud.iai.v20200303 import iai_client, models

# 小车电机引脚定义
IN1 = 20  # 前左电机是否开启
IN2 = 21  # 后左电机是否开启
IN3 = 19  # 前右电机是否开启
IN4 = 26  # 后右电机是否开启
ENA = 16  # 左侧电机控速
ENB = 13  # 右侧电机控速

# 小车按键定义
key = 8

# RGB 三色灯引脚定义
LED_R = 22
LED_G = 27
LED_B = 24

# 蜂鸣器接口定义
alarm = 8

# 风扇引脚设置
fan = 2

# 循迹红外引脚定义
```

```
# TrackSensorLeftPin1 TrackSensorLeftPin2 TrackSensorRightPin1 TrackSensorRightPin2
#         3                    5                    4                    18
TrackSensorLeftPin1 = 3   # 定义左边第一个循迹红外传感器引脚为 3 口
TrackSensorLeftPin2 = 5   # 定义左边第二个循迹红外传感器引脚为 5 口
TrackSensorRightPin1 = 4   # 定义右边第一个循迹红外传感器引脚为 4 口
TrackSensorRightPin2 = 18   # 定义右边第二个循迹红外传感器引脚为 18 口

# 超声波引脚定义
EchoPin = 0   # 回声脚
TrigPin = 1   # 触发脚

# 红外跟随模块引脚定义
FollowSensorLeft = 12
FollowSensorRight = 17

# 小车舵机定义
enSERVOUP = 3
enSERVODOWN = 4
enSERVOUPDOWNINIT = 5
enSERVOSTOP = 8

# 初始化上下左右角度为 90 度
ServoUpDownPos = 90

# 舵机引脚定义
ServoUpDownPin = 9

# 设置 GPIO 口为 BCM 编码方式
GPIO.setmode(GPIO.BCM)

# 忽略警告信息
GPIO.setwarnings(False)
```

```
# 电机引脚初始化为输出模式
# 按键引脚初始化为输入模式
# 循迹引脚初始化为输入模式
def init():
    global pwm_ENA
    global pwm_ENB
    global pwm_UpDownServo
    GPIO.setup(ENA, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(ENB, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(key, GPIO.IN)
    # 蜂鸣器
    GPIO.setup(alarm, GPIO.OUT, initial=GPIO.HIGH)
    # 风扇
    GPIO.setup(fan, GPIO.OUT, initial=GPIO.HIGH)
    # 循迹红外引脚
    GPIO.setup(TrackSensorLeftPin1, GPIO.IN)
    GPIO.setup(TrackSensorLeftPin2, GPIO.IN)
    GPIO.setup(TrackSensorRightPin1, GPIO.IN)
    GPIO.setup(TrackSensorRightPin2, GPIO.IN)
    GPIO.setup(EchoPin, GPIO.IN)
    GPIO.setup(TrigPin, GPIO.OUT)
    GPIO.setup(FollowSensorLeft, GPIO.IN)
    GPIO.setup(FollowSensorRight, GPIO.IN)
    # RGB 三色灯设置为输出模式
    GPIO.setup(LED_R, GPIO.OUT)
    GPIO.setup(LED_G, GPIO.OUT)
    GPIO.setup(LED_B, GPIO.OUT)
    # 舵机
    GPIO.setup(ServoUpDownPin, GPIO.OUT)
    # 设置 pwm 引脚和频率为 2000hz
```

```
    pwm_ENA = GPIO.PWM(ENA, 2000)
    pwm_ENB = GPIO.PWM(ENB, 2000)
    pwm_ENA.start(0)
    pwm_ENB.start(0)
    # 设置舵机的频率和起始占空比
    pwm_UpDownServo = GPIO.PWM(ServoUpDownPin, 50)
    pwm_UpDownServo.start(0)


# 摄像头舵机上下旋转到指定角度
def updownservo_appointed_detection(pos):
    for i in range(1):
        pwm_UpDownServo.ChangeDutyCycle(2.5 + 10 * pos / 180)
        time.sleep(0.02)  # 等待 20ms 周期结束
        # pwm_UpDownServo.ChangeDutyCycle(0)  #归零信号


# 摄像头舵机上下归位
def servo_updown_init():
    updownservo_appointed_detection(0)


# 舵机停止
def servo_stop():
    pwm_UpDownServo.ChangeDutyCycle(0)  # 归零信号


# 小车前进
def run(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)
    pwm_ENA.ChangeDutyCycle(leftspeed)
```

```
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车后退
def back(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.HIGH)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车向左倒车
def bleft(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.HIGH)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车向右倒车
def bright(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车左转
```

```
def left(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车右转
def right(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车原地左转
def spin_left(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)


# 小车原地右转
def spin_right(leftspeed, rightspeed):
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
```

```
    GPIO.output(IN4, GPIO.HIGH)
    pwm_ENA.ChangeDutyCycle(leftspeed)
    pwm_ENB.ChangeDutyCycle(rightspeed)



# 小车停止
def brake():
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)



# 按键检测
def key_scan():
    while GPIO.input(key):
        pass
    while not GPIO.input(key):
        time.sleep(0.01)
        if not GPIO.input(key):
            time.sleep(0.01)
            while not GPIO.input(key):
                pass



# 超声波函数
def Distance_test():
    GPIO.output(TrigPin, GPIO.HIGH)
    time.sleep(0.000015)
    GPIO.output(TrigPin, GPIO.LOW)
    while not GPIO.input(EchoPin):
        pass
    t1 = time.time()
    while GPIO.input(EchoPin):
```

```
            pass
        t2 = time.time()
        print("distance is %d " % (((t2 - t1) * 340 / 2) * 100))
        time.sleep(0.01)
        return ((t2 - t1) * 340 / 2) * 100



# 巡线
def tracking(leftspeed, rightspeed):
        # 检测到黑线时循迹模块相应的指示灯亮，端口电平为 LOW
        # 未检测到黑线时循迹模块相应的指示灯灭，端口电平为 HIGH
        TrackSensorLeftValue1 = GPIO.input(TrackSensorLeftPin1)
        TrackSensorLeftValue2 = GPIO.input(TrackSensorLeftPin2)
        TrackSensorRightValue1 = GPIO.input(TrackSensorRightPin1)
        TrackSensorRightValue2 = GPIO.input(TrackSensorRightPin2)

        # 四路循迹引脚电平状态
        # 0 0 0 0
        # 黑色横线，维持上次动作
        if TrackSensorLeftValue1 == False and TrackSensorLeftValue2 == False and TrackSensorRightValue1
== False and TrackSensorRightValue2 == False:
            run(leftspeed, rightspeed)

        # 四路循迹引脚电平状态
        # 0 0 X 0
        # 1 0 X 0
        # 0 1 X 0
        # 以上 6 种电平状态时小车原地右转
        # 处理右锐角和右直角的转动
        elif (TrackSensorLeftValue1 == False or TrackSensorLeftValue2
              == False) and TrackSensorRightValue2 == False:
            spin_right(40, 40)
            time.sleep(0.08)
```

```
    # 四路循迹引脚电平状态
    # 0 X 0 0
    # 0 X 0 1
    # 0 X 1 0
    # 处理左锐角和左直角的转动
    elif TrackSensorLeftValue1 == False and (TrackSensorRightValue1 == False or
                                             TrackSensorRightValue2 == False):
        spin_left(40, 40)
        time.sleep(0.08)

    # 0 X X X
    # 最左边检测到
    elif TrackSensorLeftValue1 == False:
        spin_left(30, 30)

    # X X X 0
    # 最右边检测到
    elif TrackSensorRightValue2 == False:
        spin_right(30, 30)
    # 四路循迹引脚电平状态
    # X 0 1 X
    # 处理左小弯
    elif TrackSensorLeftValue2 == False and TrackSensorRightValue1 == True:
        left(0, 40)

    # 四路循迹引脚电平状态
    # X 1 0 X
    # 处理右小弯
    elif TrackSensorLeftValue2 == True and TrackSensorRightValue1 == False:
        right(40, 0)
    # 四路循迹引脚电平状态
    # X 0 0 X
    # 处理直线
    elif TrackSensorLeftValue2 == False and TrackSensorRightValue1 == False:
```

```
            run(leftspeed, rightspeed)

        # 当为 1 1 1 1 时小车保持上一个小车运行状态



# 人脸识别
def takePhoto(picpath):
        """拍照
        """
        ret, img = cv2.VideoCapture(0).read()
        cv2.imwrite('/home/pi/Desktop/' + picpath, img)
        print('Photo is token.')



def result(res):
        """判断人脸识别结果
        """
        x = res['Score']
        threshold = 85

        if x > threshold:
                return True
        return False



def checkFace(picpath, personid):
        """人脸识别
        """
        # 先对拍摄到的图片进行 base64 编码
        with open(picpath, 'rb') as f:
                base64_data = base64.b64encode(f.read())
        s = base64_data.decode()
        pic_b64 = '' + s    # 将图片解码
```

```
    SecretId = "AKIDFJ7fy1nIWVEgsqrc7ML3UfZDaiaCYdT2"    # id
    SecretKey = "KNAgvuc3D6j6C3ey2rQkpdARRChvsYy7"    #  密钥

    try:
        # 实例化一个认证对象
        cred = credential.Credential(
            SecretId, SecretKey)   # 入参需要传入腾讯云账户 secretId，secretKey

        # 实例化一个 http 选项
        httpProfile = HttpProfile()
        httpProfile.endpoint = "iai.tencentcloudapi.com"    # 指定接入地域域名(默认就近接入)

        # 实例化一个 client 选项
        clientProfile = ClientProfile()
        clientProfile.httpProfile = httpProfile
        # 实例化要请求产品的 client 对象
        client = iai_client.IaiClient(cred, "ap-shanghai", clientProfile)

        # 实例化一个实例信息查询请求对象 req
        req = models.VerifyPersonRequest()
        params = {"Image": pic_b64, "PersonId": personid, "QualityControl": 2}
        req.from_json_string(json.dumps(params))

        resp = client.VerifyFace(req)
        return True, json.loads(resp.to_json_string())

    except TencentCloudSDKException as err:
        print(err)
        return False, err


def faceDetection():
    """人脸检测
    """
```

```
faceCascade = cv2.CascadeClassifier(
    '/home/pi/Desktop/haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)
cap.set(3, 640)  # 宽
cap.set(4, 480)  # 高

count = 0
wi = 0
hi = 0
while True:
    ret, img = cap.read()
    # img = cv2.flip(img, -1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.2,
                                         minNeighbors=5,
                                         minSize=(20, 20))

    for (x, y, w, h) in faces:
        # 框出人脸范围
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]
        wi = w
        hi = h

    cv2.imshow('video', img)

    k = cv2.waitKey(30) & 0xff
    if k == 27 or count == 30:  # 按'ESC'结束或等待时间截止
        break
    count += 1
```

```
    time.sleep(2)
    flag = False
    # 太小的不能算人脸
    if wi < 100 or hi < 100:
        flag = False
    else:
        flag = True
    # 摄像头调用完毕
    cap.release()
    cv2.destroyAllWindows()
    return flag


def checking(picpath='tmp.jpg', personid='1'):
    """识别总调用
    """
    i = 0
    flag = 0
    # 摄像头纵向舵机抬至 45 度，方便人脸识别
    updownservo_appointed_detection(45)
    time.sleep(1)
    servo_stop()
    # 检测是否有人脸
    havePer = faceDetection()
    if not havePer:
        # 没有人脸，返回-1
        servo_updown_init()
        return -1

    # 识别失败则再次识别，最多重复 3 次
    while i < 3:
        takePhoto(picpath)
        # 调用腾讯云 API 得到人脸验证结果
        rflag, info = checkFace(picpath, personid)
```

```
        if rflag:
            # 根据验证分数得到结果
            if result(info):
                # 验证通过，闪绿灯
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.HIGH)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.HIGH)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                print('好耶')
                flag = 1
                # 验证通过，返回 1
                servo_updown_init()
                return flag
            else:
                # 验证未通过，闪红灯
                GPIO.output(LED_R, GPIO.HIGH)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
```

```
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.HIGH)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                GPIO.output(LED_R, GPIO.LOW)
                GPIO.output(LED_G, GPIO.LOW)
                GPIO.output(LED_B, GPIO.LOW)
                time.sleep(0.05)
                print('你谁啊')

        # 验证不通过，返回 0
        i += 1

    servo_updown_init()
    return flag



# 闪灯
def ColorLED():
    # 循环显示 7 种不同的颜色
    i = 10
    while i > 0:
        GPIO.output(LED_R, GPIO.HIGH)   # 红
        GPIO.output(LED_G, GPIO.LOW)
        GPIO.output(LED_B, GPIO.LOW)
        time.sleep(0.05)
        GPIO.output(LED_R, GPIO.LOW)   # 绿
        GPIO.output(LED_G, GPIO.HIGH)
        GPIO.output(LED_B, GPIO.LOW)
        time.sleep(0.05)
        GPIO.output(LED_R, GPIO.LOW)   # 蓝
        GPIO.output(LED_G, GPIO.LOW)
        GPIO.output(LED_B, GPIO.HIGH)
```

```
            time.sleep(0.05)
            GPIO.output(LED_R, GPIO.HIGH)   # 红绿
            GPIO.output(LED_G, GPIO.HIGH)
            GPIO.output(LED_B, GPIO.LOW)
            time.sleep(0.05)
            GPIO.output(LED_R, GPIO.HIGH)   # 红蓝
            GPIO.output(LED_G, GPIO.LOW)
            GPIO.output(LED_B, GPIO.HIGH)
            time.sleep(0.05)
            GPIO.output(LED_R, GPIO.LOW)   # 绿蓝
            GPIO.output(LED_G, GPIO.HIGH)
            GPIO.output(LED_B, GPIO.HIGH)
            time.sleep(0.05)
            GPIO.output(LED_R, GPIO.LOW)   # 灭
            GPIO.output(LED_G, GPIO.LOW)
            GPIO.output(LED_B, GPIO.LOW)
            time.sleep(0.05)
            i -= 1


# 蜂鸣器报警
def alarming():
    GPIO.setup(8, GPIO.OUT)   # 将 BCM12 号引脚设置为输出模式
    pwm = GPIO.PWM(8, 1000)   # 设置 BCM12 号引脚, 设置 pwm 频率为 1000HZ
    pwm.start(50)   # 设置初始占空比（范围：0.0 <= dc <= 100.0)
    # pwm.ChangeFrequency(freq)     # freq 为设置的新频率，单位为 Hz
    # pwm.ChangeDutyCycle(dc)       # dc 为设置的新的占空比 范围：0.0-100.0
    i = 1000   # 频率
    dirs = 1   # 频率递增方向，1 为正，-1 为负
    try:
        count5 = 0
        while count5 < 300:   # 蜂鸣持续 1.5 秒
            pwm.ChangeFrequency(i)   # 为设置的新频率，单位为 Hz
            count5 += 1
```

```
            i = i + 100 * dirs    # 当前频率加上要增加的频率(10)乘以方向
            time.sleep(0.05)    # 延时 0.05 秒
            print('pwm 当前频率为: %d ' % i)    # 控制台打印当前的频率
            if i >= 2000:    # 如果当前频率大于 2000hz，方向改为负
                dirs = -1
            elif i <= 1000:    # 如果当前频率小于 1000hz，方向改为正
                dirs = 1
        GPIO.setup(alarm, GPIO.OUT, initial=GPIO.HIGH)    # 初始化停止蜂鸣
    finally:
        pass


# 对主人启动风扇
def fanBlow():
    GPIO.output(fan, not GPIO.input(fan))    # 风扇持续 4 秒
    time.sleep(4)
    GPIO.setup(fan, GPIO.OUT, initial=GPIO.HIGH)


# 播放音频
def sound():
    path = "test.mp3"    # 后续更改路径
    os.system('mplayer %s' % path)


# 发送邮件
def sendEmail():
    # 设置服务器所需信息
    fromaddr = '1137528068@qq.com'
    password = 'rmlkulkncwgbgbfd'    # qq 邮箱授权码
    toaddrs = ['757560948@qq.com', '1405257838@qq.com']

    # 设置 email 信息
    content = 'hello, this is email content.'
```

```
    textApart = MIMEText(content)

    imageFile = '/home/pi/Desktop/tmp.jpg'
    imageApart = MIMEImage(
        open(imageFile, 'rb').read(),
        imageFile.split('.')[-1])
    imageApart.add_header('Content-Disposition',
                          'attachment',
                          filename=imageFile)

    m = MIMEMultipart()
    m.attach(textApart)
    m.attach(imageApart)
    m['Subject'] = 'title'

    # 登录并发送邮件
    try:
        server = smtplib.SMTP('smtp.qq.com')   # qq 邮箱服务器地址
        server.login(fromaddr, password)
        server.sendmail(fromaddr, toaddrs, m.as_string())
        print('success')
        server.quit()

    except smtplib.SMTPException as e:
        print('error', e)   # 打印错误


# 红外跟随
def infrared_follow():
    # 遇到跟随物,红外跟随模块的指示灯亮,端口电平为 LOW
    # 未遇到跟随物,红外跟随模块的指示灯灭,端口电平为 HIGH
    LeftSensorValue = GPIO.input(FollowSensorLeft)
    RightSensorValue = GPIO.input(FollowSensorRight)
```

```
    if LeftSensorValue == False and RightSensorValue == False:
        run(13, 13)  # 当两侧均检测到跟随物时调用前进函数
    elif LeftSensorValue == False and RightSensorValue == True:
        spin_left(60, 60)  # 左边探测到有跟随物，有信号返回，原地向左转
        time.sleep(0.002)
    elif RightSensorValue == False and LeftSensorValue == True:
        spin_right(60, 60)  # 右边探测到有跟随物，有信号返回，原地向右转
        time.sleep(0.002)
    elif RightSensorValue == True and LeftSensorValue == True:
        brake()  # 当两侧均未检测到跟随物时停止
        return False  # 返回是否有跟随物
    return True


# 倒车入库
def back_into_garage(leftspeed, rightspeed):
    print("daocheing")
    # 检测到黑线时循迹模块相应的指示灯亮，端口电平为 LOW
    # 未检测到黑线时循迹模块相应的指示灯灭，端口电平为 HIGH
    TrackSensorLeftValue1 = GPIO.input(TrackSensorLeftPin1)
    TrackSensorLeftValue2 = GPIO.input(TrackSensorLeftPin2)
    TrackSensorRightValue1 = GPIO.input(TrackSensorRightPin1)
    TrackSensorRightValue2 = GPIO.input(TrackSensorRightPin2)

    # 四路循迹引脚电平状态
    # 0 0 0 0
    # 遇到黑色横线，倒车入库
    if TrackSensorLeftValue1 == False and TrackSensorLeftValue2 == False and TrackSensorRightValue1
== False and TrackSensorRightValue2 == False:
        back(10, 10)
        time.sleep(1.5)
        spin_left(80, 80)
        time.sleep(1.02)
        back(10, 10)
```

```
        time.sleep(1.45)
        sys.exit(0)

    # 四路循迹引脚电平状态
    # 0 0 X 0
    # 1 0 X 0
    # 0 1 X 0
    # 以上 6 种电平状态时小车原地右转
    # 处理右锐角和右直角的转动
    elif (TrackSensorLeftValue1 == False or TrackSensorLeftValue2
            == False) and TrackSensorRightValue2 == False:
        spin_right(40, 40)
        time.sleep(0.08)

    # 四路循迹引脚电平状态
    # 0 X 0 0
    # 0 X 0 1
    # 0 X 1 0
    # 处理左锐角和左直角的转动
    elif TrackSensorLeftValue1 == False and (TrackSensorRightValue1 == False or
                                             TrackSensorRightValue2 == False):
        spin_left(40, 40)
        time.sleep(0.08)

    # 0 X X X
    # 最左边检测到
    elif TrackSensorLeftValue1 == False:
        spin_left(30, 30)

    # X X X 0
    # 最右边检测到
    elif TrackSensorRightValue2 == False:
        spin_right(30, 30)
```

```
    # 四路循迹引脚电平状态
    # X 0 1 X
    # 处理左小弯
    elif TrackSensorLeftValue2 == False and TrackSensorRightValue1 == True:
        left(0, 40)

    # 四路循迹引脚电平状态
    # X 1 0 X
    # 处理右小弯
    elif TrackSensorLeftValue2 == True and TrackSensorRightValue1 == False:
        right(40, 0)

    # 四路循迹引脚电平状态
    # X 0 0 X
    # 处理直线
    elif TrackSensorLeftValue2 == False and TrackSensorRightValue1 == False:
        run(leftspeed, rightspeed)

    # 当为 1 1 1 1 时小车保持上一个小车运行状态


# 颜色识别组合三种操作
# 颜色库
def getColorList():
    dict = collections.defaultdict(list)

    # red
    lower_red = np.array([0, 43, 46])  # 红色色域低值
    upper_red = np.array([10, 255, 255])  # 红色色域高值
    color_list_red = []
    color_list_red.append(lower_red)
    color_list_red.append(upper_red)
    dict['red'] = color_list_red
```

```
    # green
    lower_green = np.array([35, 43, 46])  # 绿色色域低值
    upper_green = np.array([77, 255, 255])  # 绿色色域高值
    color_list_green = []
    color_list_green.append(lower_green)
    color_list_green.append(upper_green)
    dict['green'] = color_list_green

    # blue
    lower_blue = np.array([100, 43, 46])  # 蓝色色域低值
    upper_blue = np.array([124, 255, 255])  # 蓝色色域高值
    color_list_blue = []
    color_list_blue.append(lower_blue)
    color_list_blue.append(upper_blue)
    dict['blue'] = color_list_blue

    return dict


def get_color(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)  # 转化为 HSV 颜色空间
    maxsum = 0
    color = None
    color_dict = getColorList()  # 获取颜色库
    for d in color_dict:
        # 二值化功能，主要是将在两个阈值内的像素值设置为白色（255），而不在阈值区间内的像素值设置为黑色（0）
        mask = cv2.inRange(hsv, color_dict[d][0], color_dict[d][1])
        mask = cv2.erode(mask, None, iterations=2)  # 腐蚀函数
        mask = cv2.GaussianBlur(mask, (3, 3), 0)  # 对图像进行高斯滤波，去除噪声
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)[-2]  # 寻找轮廓
        sum = 0
        for c in cnts:
```

```
                sum += cv2.contourArea(c)  # 计算图像轮廓的面积
            if sum > maxsum:  # 选出色域面积最大的颜色即为当前所识别出的颜色
                maxsum = sum
                color = d

    return color


# 红绿蓝卡牌颜色命令
def videox_color():
    vix = cv2.VideoCapture(0)  # 调用摄像头
    tu, frame = vix.read()
    frame = cv2.GaussianBlur(frame, (5, 5), 0)  # 图像进行高斯滤波，去除噪声
    for i in range(30):
        cv2.imshow("video", frame)  # 显示调取画面
        cv2.waitKey(30)
    color = get_color(frame)
    # 摄像头调用结束
    vix.release()
    cv2.destroyAllWindows()
    print(color)
    return color  # 返回识别出的颜色


def colorcontrol():
    color = videox_color()  # 识别颜色
    if color == "blue":  # 识别出蓝色，点亮彩灯，2 秒后播放音频
        ColorLED()
        time.sleep(2)
        sound()
    elif color == "green":  # 识别出绿色，进行红外跟随
        while infrared_follow():
            pass
        print("end follow")
```

```
            time.sleep(3)
        elif color == "red":   # 识别出红色，进行倒车入库
            print("daoche")
            while True:
                back_into_garage(20, 20)


# 小车行驶中遇到障碍物进行绕开操作
def moveAway():
    spin_left(18, 18)   # 原地左转 0.7s
    time.sleep(0.7)
    brake()
    run(10, 10)   # 前进 1s
    time.sleep(1)
    spin_right(15, 15)   # 原地右转 0.8s
    time.sleep(0.8)
    run(10, 10)   # 前进 1.9s
    time.sleep(1.9)
    spin_right(20, 20)   # 原地右转 0.5s
    time.sleep(0.5)
    run(10, 10)   # 前进 1.2s
    time.sleep(1.2)
    spin_left(20, 20)   # 原地左转 0.8s
    time.sleep(0.8)


# 延时 2s
time.sleep(2)

try:
    init()   # 初始化
    key_scan()   # 按键检测
    while True:
        distance = Distance_test()   # 超声波实时测距
```

```
            if distance > 50:
                tracking(20, 20)   # 当距离障碍物较远时高速巡线前进
            elif 20 <= distance <= 50:
                tracking(10, 10)   # 当快靠近障碍物时低速巡线前进
            elif distance < 20:   # 当靠近障碍物
                brake()   # 停车 2 秒
                time.sleep(2)
                recognition = checking()   # 进行人脸识别
                if recognition == 1:   # 识别到主人，主人可用红绿蓝三色卡牌控制小狗
                    fanBlow()   # 给主人吹风
                    time.sleep(2)
                    count = 0
                    print("举起卡牌")
                    colorcontrol()   # 进行颜色识别
                elif recognition == 0:   # 识别到陌生人
                    sendEmail()   # 发送邮件至陌生人照片主人邮箱
                    time.sleep(1)
                    alarming()   # 发送完邮件 1 秒后蜂鸣报警
                elif recognition == -1:   # 未识别到人，判断为障碍物
                    moveAway()   # 执行避障操作

except KeyboardInterrupt:
    pass
pwm_ENA.stop()   # 左电机停止运行
pwm_ENB.stop()   # 右电机停止运行
GPIO.cleanup()   # 管脚清零
```

4.2 执行结果

智能巡逻小狗整体行进轨迹图 17 所示。

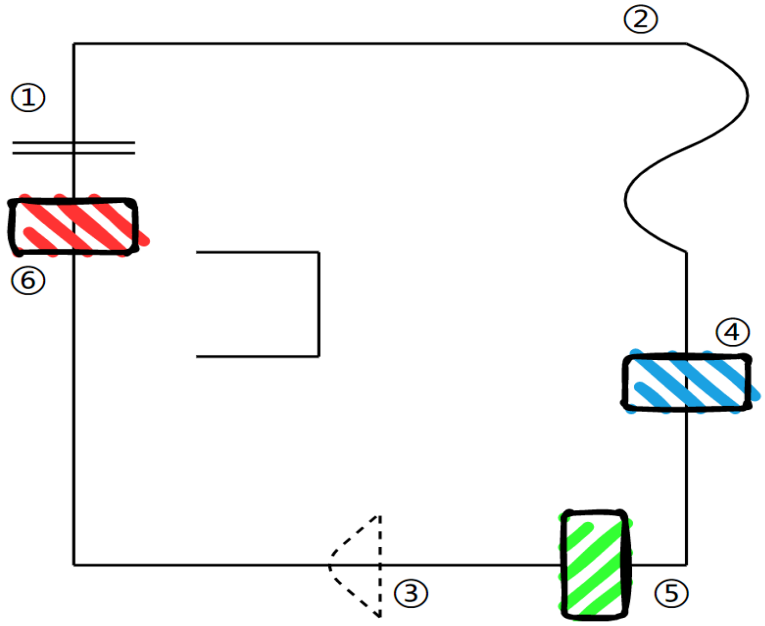

图 17 小狗巡逻图

按下 key 键后，小狗接到巡逻信号，从 1 号点出发顺时针沿着住宅外围巡逻，如图 18 所示。

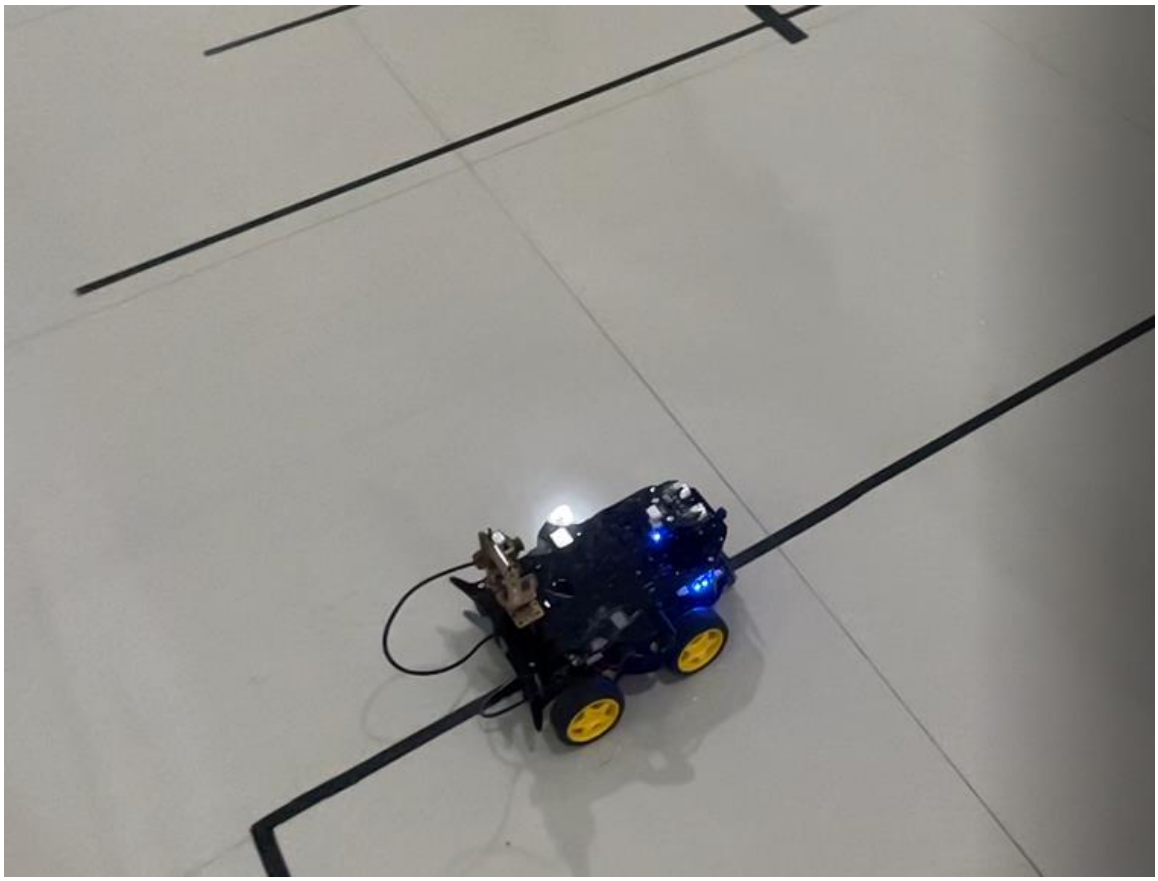

图 18 开始巡逻

2 号点小狗沿 S 型轨迹循迹，如图 19 所示。

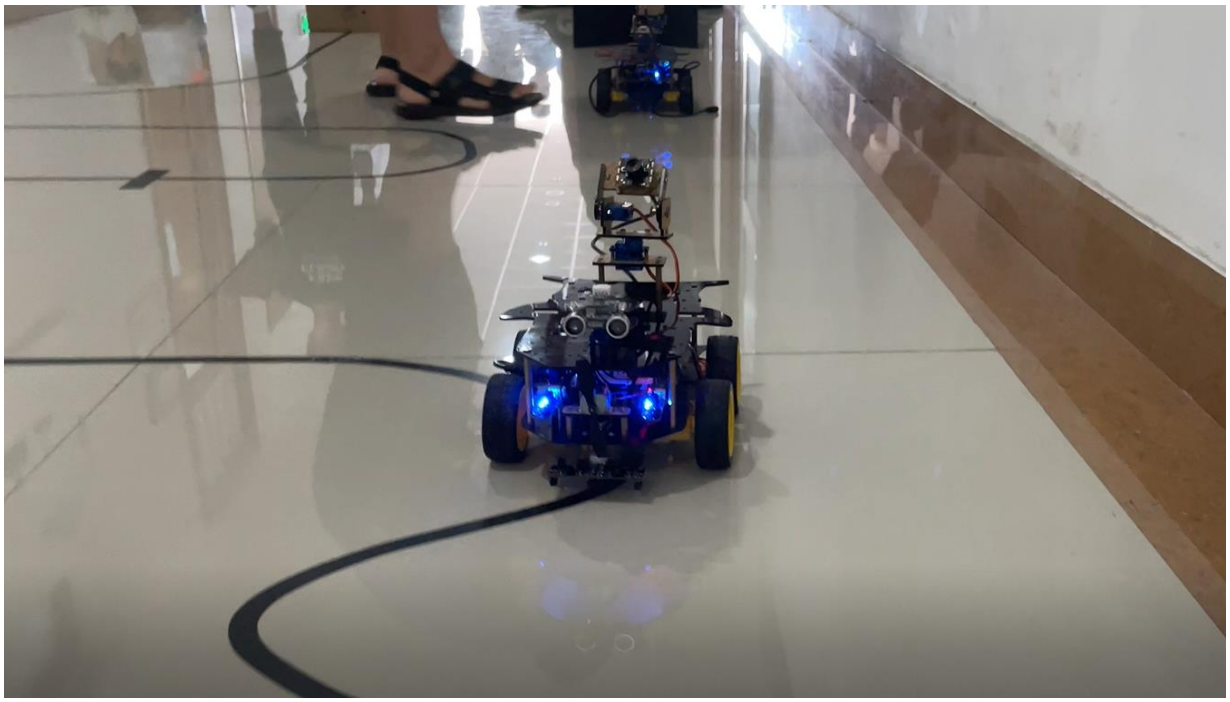

图 19 S 型循迹

3 号点小狗巡逻路上遇到障碍物，先减速，进一步靠近后停车并进行人脸检测，未检测到人脸则判断为障碍物，从而进行执行超声波避障，如图 20 所示。

图 20 开始避障

3 号点避障结束后，小狗继续巡逻，如图 21 所示。

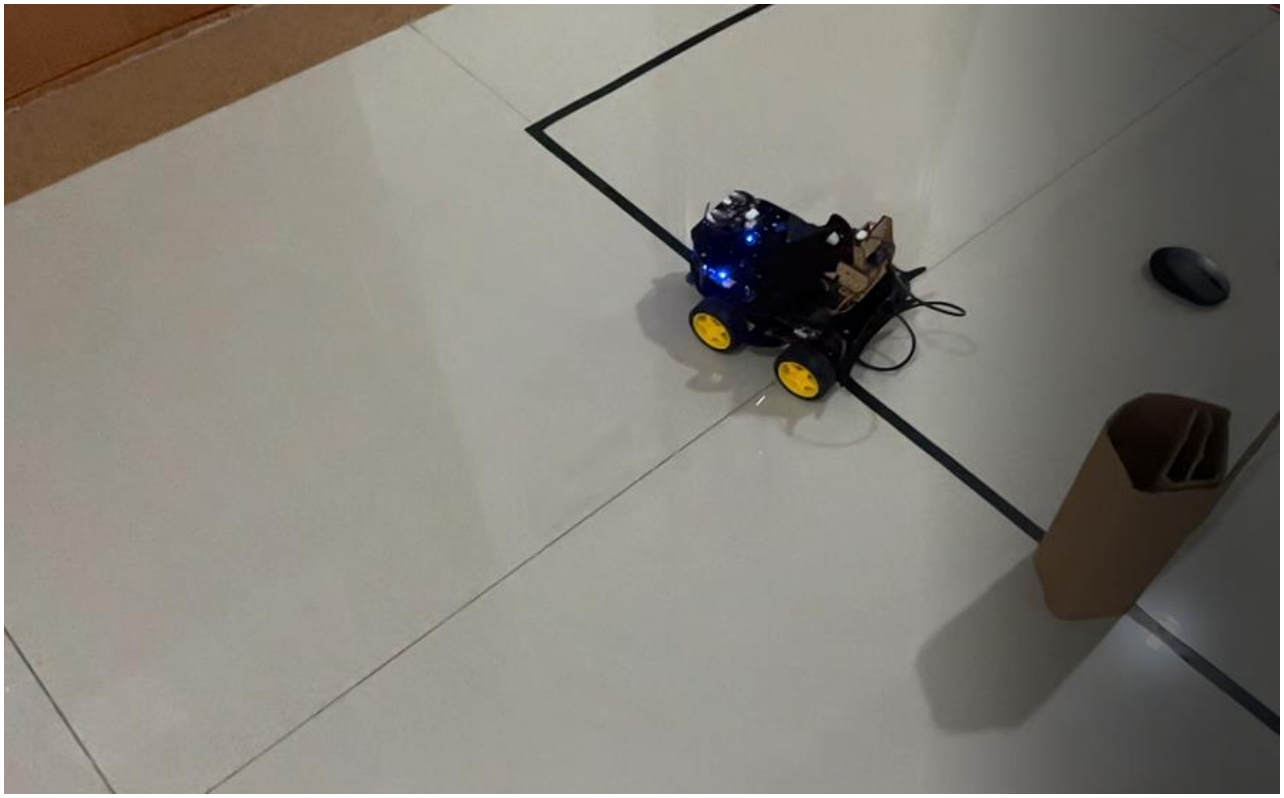

图 21 避障后继续巡逻

3 号点小狗遇到障碍物，抬起摄像头进行人脸检测，判断为人后进一步开始人脸识别，判断不是主人，则对其拍照并发送至主人邮箱，如图 22 所示。

图 22 陌生人照片传至邮箱

4 号点小狗遇到障碍物，抬起摄像头进行人脸识别，判断为主人后对主人吹风，亮灯，随后进行颜色识别，当识别蓝牌时，如图 23 所示。

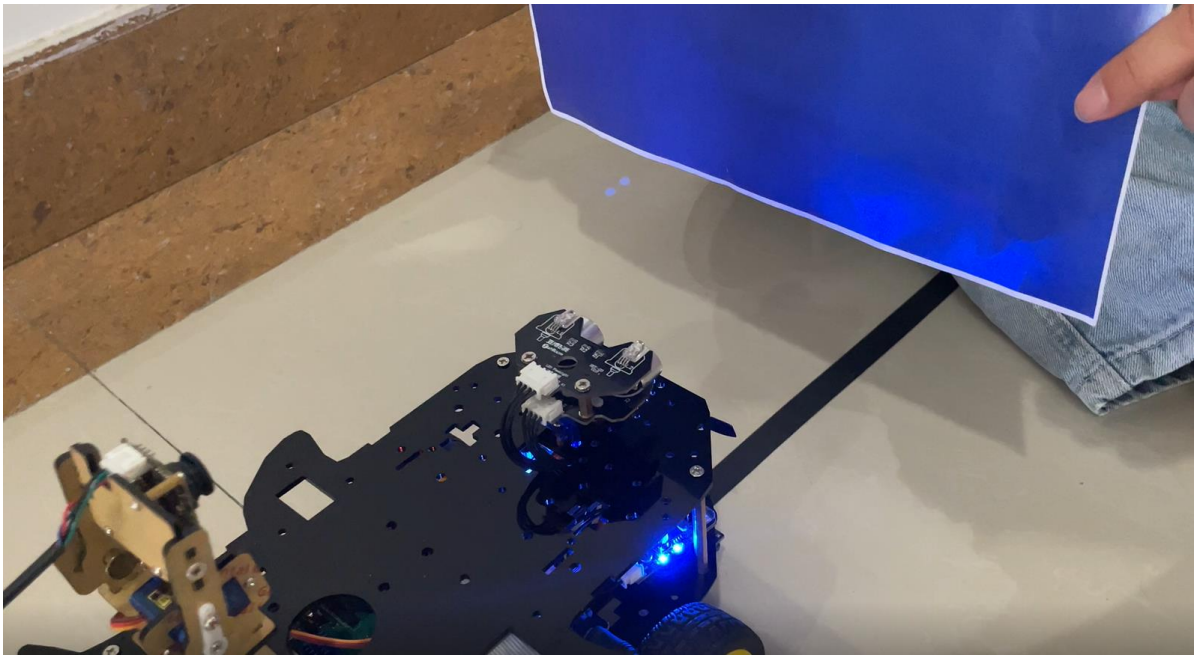

图 23 识别蓝色

4 号点识别出蓝色后，先亮彩灯，随后播放一段音频，如图 24 所示，图中 A 表示 Audio，即音频输出，A 后的数字则表示当前已播放的时间和音频的总时长。

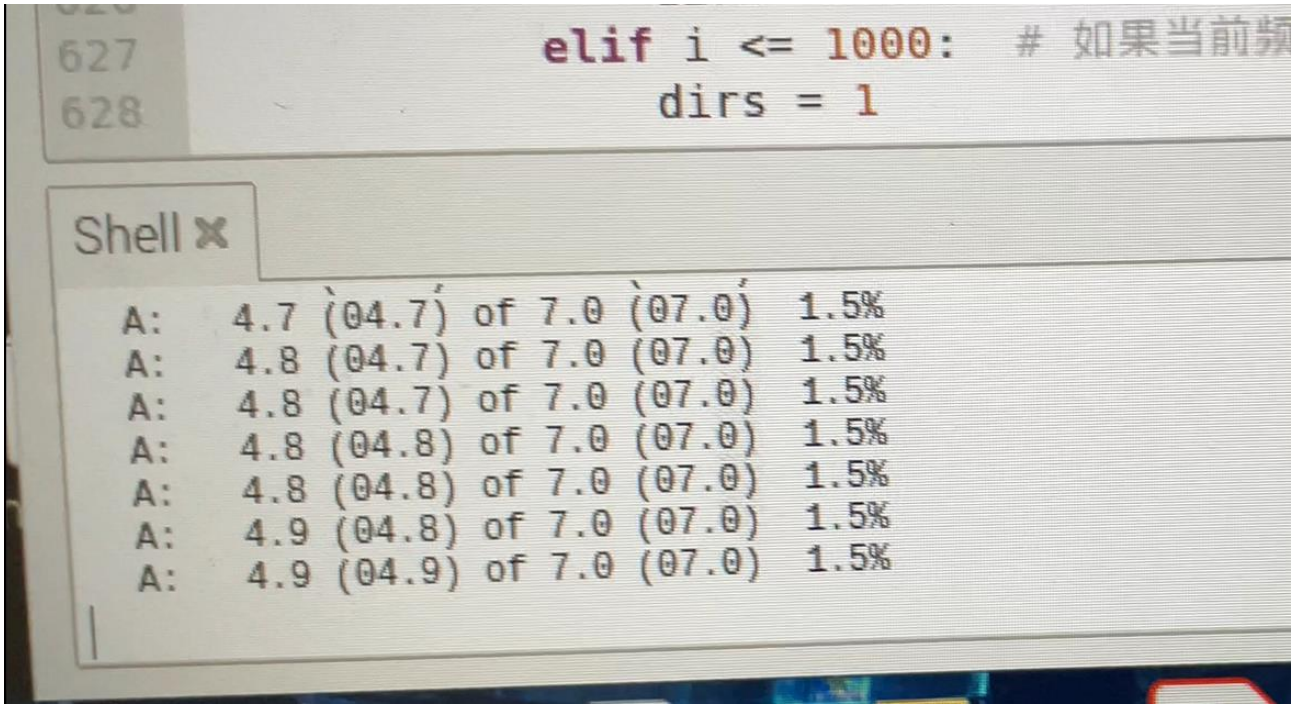

图 24 播放音频

5 号点小狗遇到障碍物，停车，人脸识别出主人后，颜色检测出绿色，开始红外跟随，将小狗引

出赛道后引回到赛道上后继续沿线巡逻，如图 25 所示。

图 25 红外跟随图

6 号点小狗遇到障碍物，人脸识别后，识别出颜色为红色，继续巡逻至横向黑线后开始倒车入库，即小狗进窝，如图 26 所示。

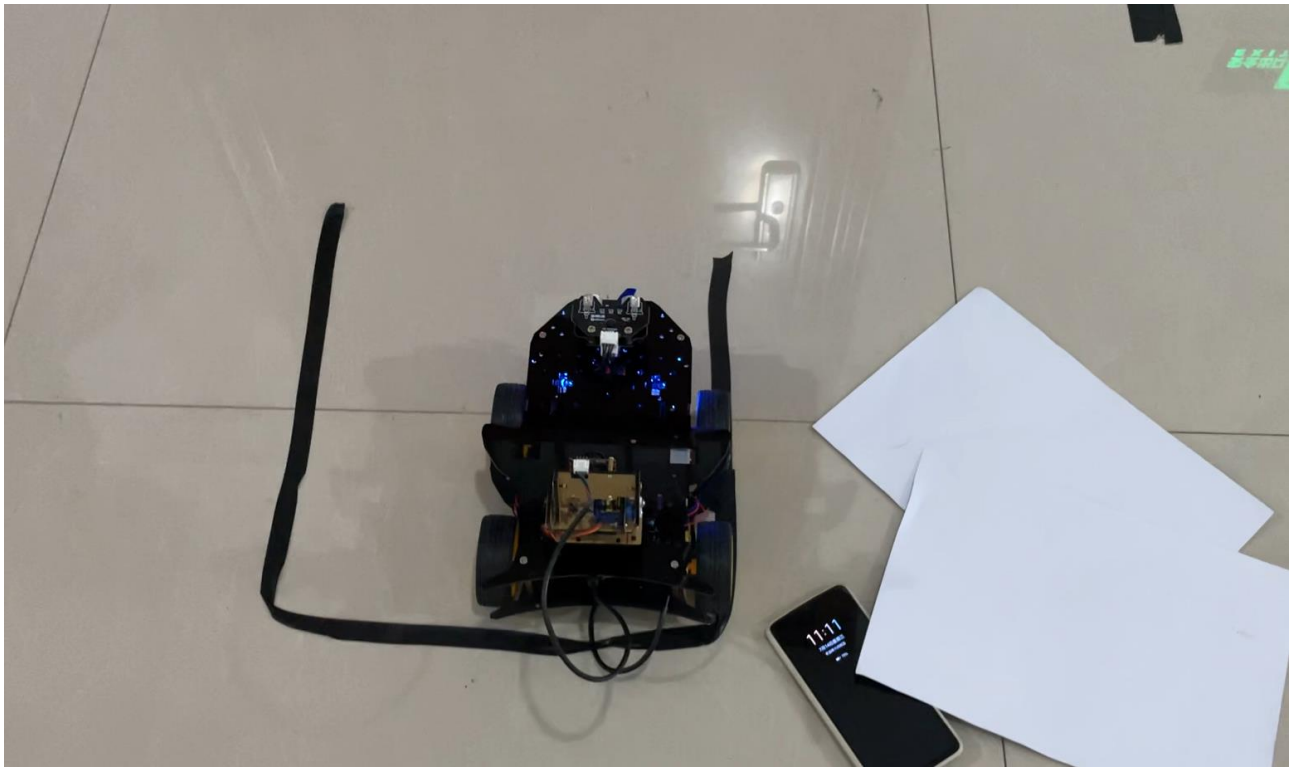

图 26 倒车入库图

五、程序调试说明和实验感想

5.1 调试说明

1. **小车在巡线过程中，遇到用来做倒车标志的黑色横线时，无法保持直行。**

   原因：最初的循迹函数中，并未定义四灯全亮或全灭时的行为，因此这时小车将保持上一次循环的行为。而在小车行进到黑色横线时，常常会由于车头不正或循迹传感器灵敏度不一致，将黑色横线错误地识别为转角而开始执行转弯操作；在全部循迹传感器进入黑色横线后，又可能会因保持了上次循环的动作而导致转角过大，掉出轨道，从而开始原地旋转。

   解决：①定义循迹函数中四灯全亮（即检测到黑色横线）时的动作为直线行驶；

   ②对巡线功能的代码进行调整，使小车能及时摆正自身行驶方向。

2. **小车在巡线过程中会受到其它因素干扰从而偏离轨道**

   原因：循迹传感器过于灵敏，在识别到砖缝甚至阴影时，经常会误判为黑色轨道。

   解决：①调节循迹传感器灵敏度，确保在遇到砖缝时无反应，遇见黑线时灯亮；

   ②开灯，保证光线良好。

3. **opencv-contrib 库无法安装**

   原因：①下载时报错为 MemoryError，表示树莓派内存不足；

   ②成功下载后，报错 Command … failed with error code 1 in …，此类错误在网上并没有找到确切的原因和有效的解决方案，由于之前尝试编译.c 的例程也会报出相似的错误，推测是无法编译 opencv-contrib 库中的 c 语言文件。

   解决：①针对 MemoryError，扩展树莓派虚拟内存即可；

   ②针对 Command … failed with error code 1 in …，没有找到有效的解决方案，放弃使用 opencv-contrib 库，相关功能转用腾讯云 API 实现。

4. **OpenCV 无法打开摄像头**

   原因：OpenCV 和树莓派自身的 mjpg_stream 视频流进程有冲突，不能同时使用。

   解决：如图 27 使用 top 命令查看进程号，再使用 sudo kill -9 进程号的命令杀死树莓派自身的 mjpg_stream 进程即可。

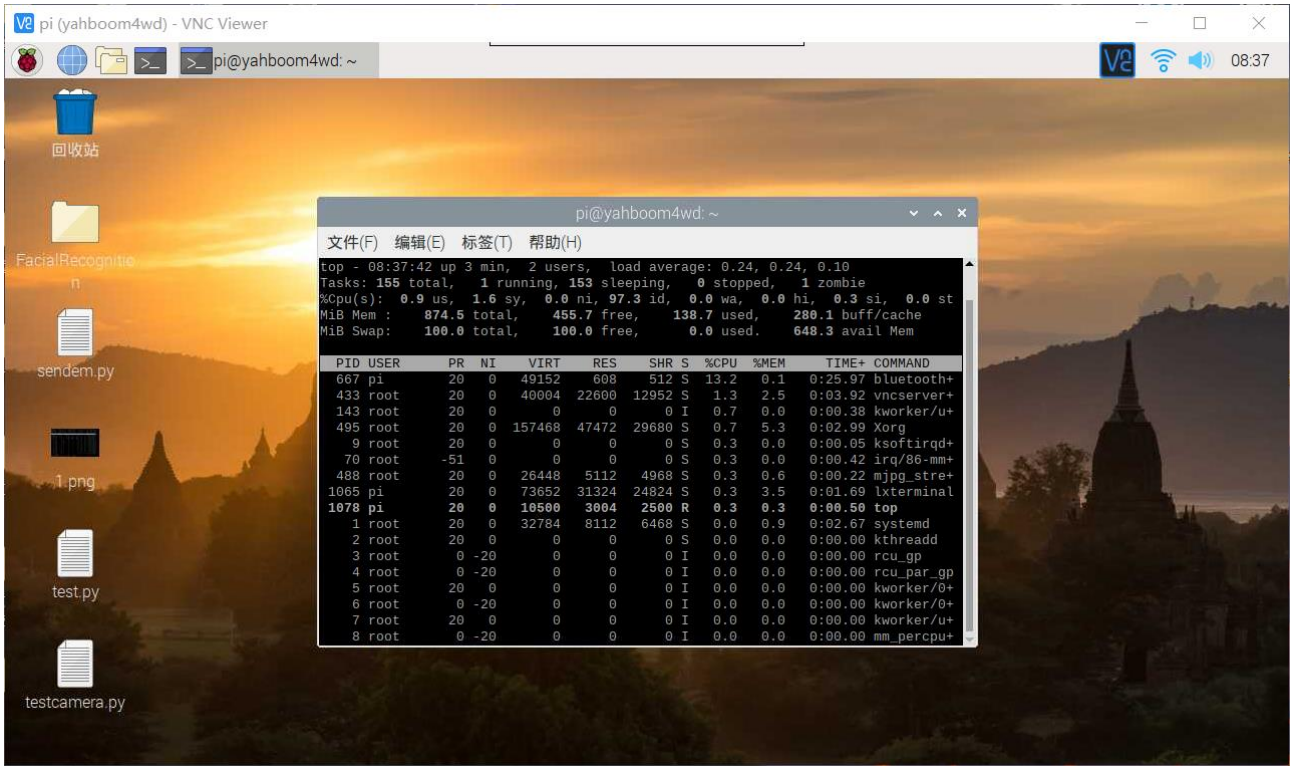

图 27 查看进程号

5.2 实验特色

1. 巡线时根据超声波所测距离对小车进行变速，在保证整体运行速度的同时避免了因撞到障碍导致的损坏；
2. 在遇到障碍时先进行人脸检测判断是障碍物还是人，若是人再进行人脸识别，这样的策略使小车能够应对不同的障碍情况；
3. 进行人脸检测前，摄像头舵机会抬升，方便看到人脸；人脸识别结束后舵机下降，重新看向正前方；
4. 人脸识别后彩灯闪烁，便于查看识别结果；
5. 人脸识别失败则将访客照片发送到主人邮箱，提醒主人来访情况；
6. 人脸识别调用了线上的预训练模型，识别准确度高，同时方便对人员库进行管理；
7. 循迹功能可以适应不同转弯情况，小车行驶流畅，循迹效果好；
8. 倒车前先进行循迹操作，直到遇到倒车标志（即黑色横线）时才开始倒车，这样一来在任意位置下达倒车指令都可以正确实现倒车；
9. 倒车入库前通过让小车高速旋转数圈来调整方向，观赏性强；
10. 尽可能使用了较多模块，对小车功能进行了拓展。

5.3 总结与展望

这次工程实训让我们受益匪浅，一方面我们对使用程序控制硬件设备的方法有了初步的了解，另一方面我们也初步掌握了各种 API 在实际项目中的应用方法，同时我们的动手能力也得到了提高。我们总结了体会和经验如下。

1. 工程实训中动手操作的过程和以往我们使用软件仿真有很大的不同，在实际操作中往往会出现各种无法预料的干扰和问题，这极大地考验了我们的心态和应变能力；
2. 要成功完成目标，我们必须弄懂实验的原理，在吃透例程代码、会自己编写代码的同时，还要熟悉硬件的控制和操作。然而在小车的设计和调试过程中，很多环节都是陌生的，需要我们自己去学习、去提问。在这个过程中，我们的自信心和自学能力得到了提高，也体会到了研究和学习的乐趣；
3. 在实训过程中，我们只有尽量减少操作的盲目性，加强自身认识的完备性，才能真正提高实验效率。在第一次安装 OpenCV 时我们过于急躁，没有仔细阅读教程，甚至尝试直接拷贝别人安装好的文件，却因为部分文件没有操作权限而导致整个库无法使用和重装，最后只好重新烧写镜像，从头再来。
4. 我们认为，对于一个新的领域，如何通过自主学习，从无到有从 0 到 1 完成任务，是一项非常重要的能力。树莓派智能小车只是我们今后生涯的一个基础，但它给我们带来了能力的提高和视野的开阔，这是十分重要的，非常感谢这门课能给我们这个锻炼的机会！

如果有充裕的时间，我们还希望实现以下内容。

1. 加入访客的登入操作，在智能小狗端实现访客的登记，下次识别到访客便可验证通过；
2. 加入语音输入控制，利用语音信号对智能小狗发出指令；
3. 加强对视觉技术的研究，使智能小狗能识别到更多内容，如可将红外跟随升级为通过视觉信号跟随，令跟随操作更加自然便利；
4. 制作相应的网页端或 APP，使用户能够更加便利地对智能小狗进行控制，并在其中加入进行实时图像传输功能；
5. 深化对树莓派网络功能的应用，结合物联网技术让智能小狗融入智能家居的系统中；
6. 为树莓派智能小车增加更多模块（如机械臂等），使其能够实现更多功能。