2022 年全国大学生电子设计竞赛

具有自动泊车功能的电动车（B 题）

参赛队号：20220139

2022 年 10 月 16 日

具有自动泊车功能的电动车

摘要

该系统以正点原子 STM32F407VET6 作为核心板，由 OpenMV 模块、循迹检测模块、蜂鸣器模块和前轮转向式四轮车构成。该自动泊车系统的设计采用了模块化思想，完成了色块识别模块及自动泊车模块的设计，达到集中检测车库、自动泊车于一体的目的，实现了自动泊车功能。

系统主要运用色块识别方法以及自动泊车模块完成自动泊车功能。色块识别方法是对摄像头采集的图像中的感兴趣区域（ROI）中的黑色像素数进行检测和统计，当像素数大于设定的阈值，发送高电平信号，标志检测完成；自动泊车模块是利用 STM32 的定时器输出 PWM 波控制电机速度以及舵机旋转角度，通过编码器确定小车运行轨迹，当车身进入车库内通过循迹避障模块检测地面库线，待小车全部进入车库内，最终实现自动泊车功能。

关键词：自动泊车，色块识别，STM32

目录

1.引言

题目要求设计一个具有自动泊车功能的电动车，具备以下功能：（1）能够在相邻车库有车的情况下完成倒车入库；（2）能够在相邻车库有车的情况下完成侧方位入库；（3）能够在邻库无车的情况下完成倒车入库；（4）能够在邻库无车的情况下完成侧方位入库；（5）能够连续完成（1）、（2）两项功能；（6）能够连续完成（3）、（4）两项功能。

为了设计实现该功能的电动车，本设计使用 STM32F407 作为核心处理器，利用色块识别算法和自动泊车算法，通过 OpenMV 利用色块检测方法识别车库，检测到车库之后，声光显示，然后通过自动泊车算法进行倒车，倒车过程中通过循迹避障模块检测地面边缘库线，直至小车完整入库。

2.方案论证

2.1 供电方案比较与选择

方案一、电源直接给系统供电

由于电源直接接入系统会一直放电，电量损耗快，影响电动车的稳定行驶，导致测试结果出现偏差，同时也无法实时监测电源的输出电压，容易造成电源过放。

方案二、电源接入稳压模块给系统供电

将电源接入稳压模块，通过稳压模块给系统供电，保证了电源电压的输出稳定，同时模块上板载的数码管模块可以实时显示电源输出电压，方便及时充电保护。

经过综合考虑，最终选用方案二。

2.2 电机驱动方案比较与选择

方案一、使用继电器

该方案电路简单可靠，但不容易实现精细控制。

方案二、使用分立元件构成电机驱动电路

该方案结构简单，价格低廉，在实际中应用广泛，缺点是该方案工作性能不够稳定，不满足比赛的需要。

方案三、使用 DRV8701E 作为电机驱动芯片

DRV8701E 具有抗电磁干扰，驱动力强，响应频率高，体积较小等优良性能，同时带有使能控制端，方便使用。同时控制电机正反转无需两路 PWM，只需一路 PWM 加高低电平即可实现正反转，节省单片机 PWM 资源。使用该芯片作为电机驱动，操作方便，稳定性好，性能优良。

结合比赛精度需求的考虑，最终选用方案三。

2.3 摄像头方案比较与选择

方案一、使用 MT9V034 神眼摄像头

MT9V034 具有全局快门、高动态成像等优点，使用过程中能以 60fps 的形式输出。根据车库编写特征识别函数，让电动车在行驶过程中通过摄像头识别车库相应特征，做出相应动作。但是特征识别函数编写工作量大，同时摄像头在工作时容易产生图像畸变，导致无法识别或误识别情况发生，识别精度低。

方案二、使用 OpenMV

OpenMV 以 STM32F427CPU 为核心，集成了 OV7725 摄像头芯片，能够高效的实现核心机器视觉算法。可以通过摄像头对车库边缘色块的检测，在设定好的检测区域内，一旦符合条件的像素点个数达到设定值，即可认定为检测到车库，检测精度高。

综合考虑系统设计的需要，本系统选用方案二。

3.理论分析与计算

本系统将小车自动泊车任务分解为小车驱动以及车库识别两个模块。通过输出 PWM 波驱动直流电机与舵机控制小车运动，小车行进过程中进行车库识别。输出 PWM 波以及车库识别的分析与计算过程如下。

3.1 电机与舵机 PWM 波配置

为了驱动电机以及舵机转动需要配置 PWM 波，该系统使用 STM32F407 的定时器 2 以及定时器 3 调制 PWM 波，调整 PWM 波占空比控制电机转速以及舵机旋转角度。定时器 3 和定时器 2 主频为 84MHZ，PWM 周期=(psc+1)*（arr+1）/主频，其中，psc 为定时器计数值，arr 为分频系数。定时器 2 的 psc 为 19999，arr 为 83，因此，其 PWM 波周期为 0.02s；定时器 3 的 psc 为 9999，arr 为 83，因此其 PWM 波周期为 0.01s。

3.2 OpenMV 检测车库方法

为了精确检测到车库位置，对摄像头采集到的视频进行实时的色块检测。对视频图像进行二值化处理并设定一个 30×140 的 ROI 区域，小车前进过程中不断检测并统计该区域中黑色像素点个数，当像素数达到设定阈值即表示检测到了车库交接处，通过 IO 口发送高电平信号给单片机，完成车库检测。车库检测方法示意图，如图 3.1 所示。

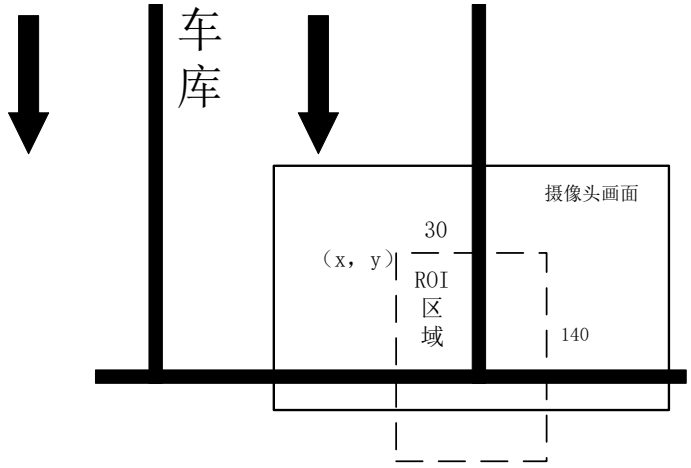


图 3.1 车库检测示意图

4.电路与程序设计

4.1 系统总体设计

本系统以 STM32F407 作为核心处理器，主要由 OpenMV 模块、循迹检测模块、蜂鸣器模块、直流电机、舵机等部分构成。该系统总体设计图以及系统设计清单如下图所示：

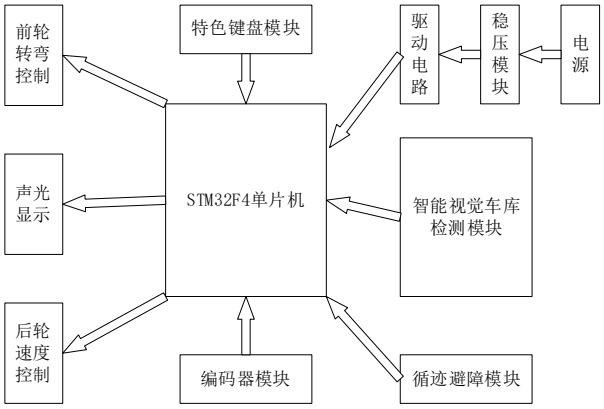


图 4.1 系统总体设计图

表 4.1 系统设计元件清单

元件	数量	元件	数量	元件	数量
四位独立按键	1	星瞳 OpenMV	1	有源蜂鸣器	1
电源	1	红外发射传感器	2	STM32F4 芯片	1
稳压模块	1	编码器	1	单片机最小系统板	1
直流电机	1	舵机	1		

4.2 单元电路设计

4.2.1 单片机最小系统设计

微处理器选择 STM32F407 单片机，其运算速度非常快，具有丰富的外设接口，适合电动车的运行，其主要性能指标如下：

（1） 工作频率：168MHZ

（2） 时钟配置： STM32F407 的 SYSCLK 最高频率是 168MHz; HCLK 最高频率为 168MHz；内部 RC 振荡器，可产生 16MHz 的时钟信号；

（3） 14 个定时器，包含：高级定时器 TIM1、TIM8 ；通用定时器 TIM2、TIM5、TIM3、TIM4，TIM9-TIM14；基本定时器 TIM6、TIM7 。

（4） 6 个 UART（串口），可灵活地与外部设备全双工数据交换。

（5） 丰富的 IO 口，方便连接外设。

STM32F407 单片机核心板电路图如附录所示，其中包括 MCU、晶振和复位电路以及外接设备模块连接情况。

4.2.2 电机驱动模块设计

该驱动模块采用 DRV8701E 芯片，该芯片具有驱动力强，响应频率高，体积较小等优良性能，同时带有使能控制端。同时控制电机正反转无需两路 PWM，只需一路 PWM 加高低电平即可实现正反转，节省单片机 PWM 资源。驱动模块电路原理图如图 4.2 所示。

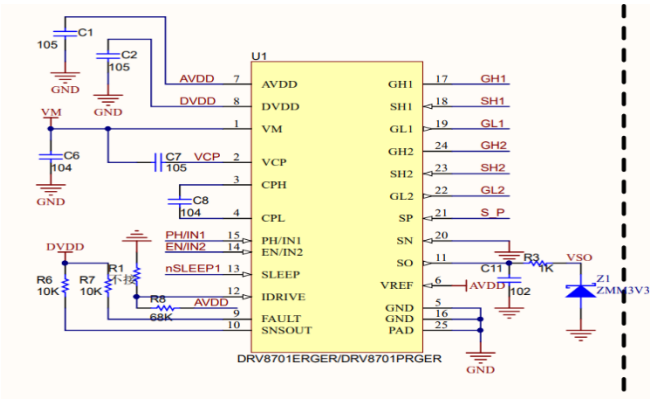


图 4.2 驱动模块原理图

4.2.3 独立按键模块设计

四位独立按键一端接地，另一端接单片机 IO 口，将单片机 IO 输入上拉，当按键按下时 IO 口被接地，IO 口电平反翻，用于按键扫描检测。该模块原理图如图 4.3 所示。

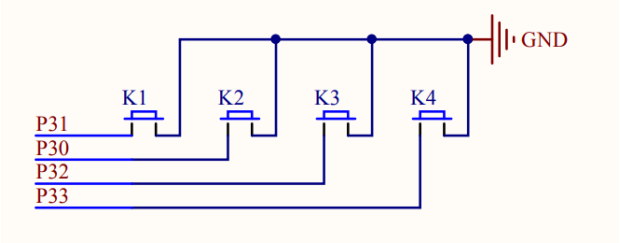


图 4.3 独立按键模块原理图

4.2.4 巡线避障模块设计

通过该模块检测车库边缘库线。使用 TCRT5000 传感器，当检测到黑线时反射回的红外线强度不够大，红外接收管关闭，输出端为高电平，指示灯熄灭。该模块电路原理图如图 4.4 所示。

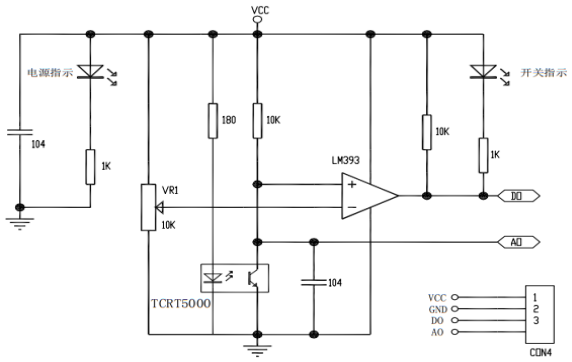


图 4.4 巡线避障模块原理图

4.3 系统程序设计

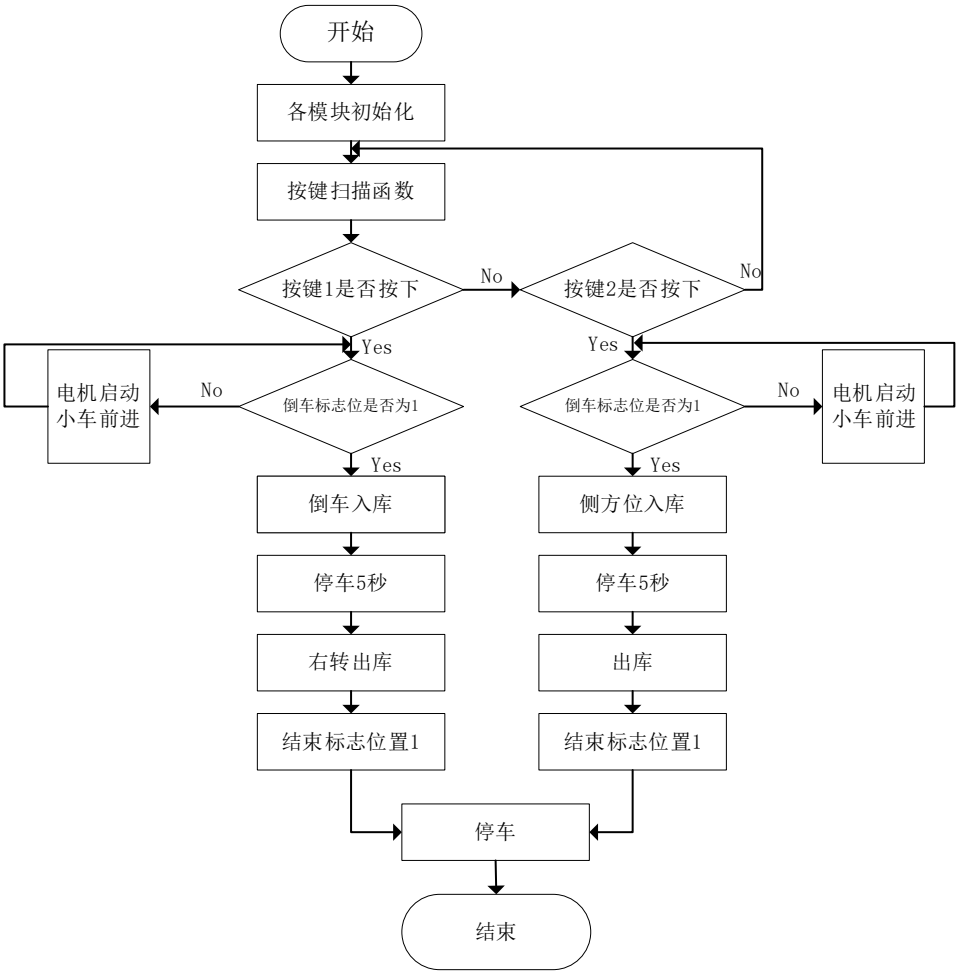


图 4.5 系统软件总体流程图

5.测试方案与测试结果

5.1 测试仪器

电动车、秒表、比赛赛道。

5.2 测试方案

5.2.1“邻库有车”情况停车测试

将库 1、库 3、库 4、库 6 分别放入车辆，分三次将电动车分别置于距起点 5cm、15cm、30cm 处，启动电动车，小车蜂鸣器第一次响起（即开始倒车）时按下秒表，待小车蜂鸣器第二次响起（即倒车完成）停止计时，记录时间。分别观

察小车倒车入库以及侧方位入库的完成情况，记录小车停止后与车库边的夹角。

5.2.2“邻库无车”情况停车测试

移除库 1、库 3、库 4、库 6 中的车辆，分三次将电动车分别置于距起点 5cm、15cm、30cm 处，启动电动车，小车蜂鸣器第一次响起（即开始倒车）时按下秒表，待小车蜂鸣器第二次响起（即倒车完成）停止计时，记录时间。分别观察小车倒车入库以及侧方位入库的完成情况，记录小车停止后与车库边的夹角。

5.3 测试结果

表 5.1 电动车“邻库有车”倒车入库情况测试结果

小车距起点距离（cm）	是否完成倒车	是否压线	倒车所用时间（s）	小车停止后与库边夹角（°）
5	是	否	2.87	7
15	是	否	2.45	9
30	是	否	2.73	10

表 5.2 电动车“邻库有车”侧方位入库情况测试结果

小车距起点距离（cm）	是否完成倒车	是否压线	倒车所用时间（s）	小车停止后与库边夹角（°）
5	是	否	2.25	9
15	是	否	2.46	11
30	是	否	2.15	10

表 5.1 电动车“邻库无车”倒车入库情况测试结果

小车距起点距离（cm）	是否完成倒车	是否压线	倒车所用时间（s）	小车停止后与库边夹角（°）
5	是	否	2.45	9
15	是	否	2.96	9
30	是	否	2.28	10

表 5.2 电动车“邻库无车”侧方位入库情况测试结果

小车距起点距离（cm）	是否完成倒车	是否压线	倒车所用时间（s）	小车停止后与库边夹角（°）
5	是	否	2.64	8
15	是	否	2.27	10
30	是	否	2.84	11

5.4 测试结果分析

根据“邻库有车”情况停车测试结果，可以看出电动车在任意位置发车，能够识别到空车库，并完成倒车入库和侧方位入库，车身位置与库边夹角在 10°左右；倒车所用时间均在 30s 以内。

根据“邻库无车”情况停车测试结果，可以看出电动车在任意位置发车，能够识别到目标车库，并完成倒车入库和侧方位入库，车身位置与库边夹角在 10°左右；倒车所用时间均在 30s 以内。

综上，该系统能够完成题目所给任务。

6.结论

该系统以正点原子 STM32F407VET6 作为核心板，由 OpenMV 模块、循迹检测模块、蜂鸣器模块和前轮转向式四轮车构成。该自动泊车系统的设计采用了模块化思想，完成了色块识别模块及自动泊车模块的设计，达到集中检测车库、自动泊车于一体的目的，实现了自动泊车功能，较好的完成了设计要求。

通过测试方案检验，本系统在邻库有车和邻库无车的情况下，均能完成自动泊车功能。

附录 1.部分程序代码

1. 1 主程序代码

```
#include "stm32f4xx.h"
#include "usart.h"
#include "delay.h"
#include "sys.h"
#include "usart.h"
#include "pwm.h"
#include "timer.h"//OPENMV_OUT() C4
#include "led.h"
#include "key.h"//A1 A2 A3 A4
#include "sensor.h"////PC1  Sensor1()  PC2   Sensor2()
#include "beer.h"//C3
#include "speed.h"//DIR A7  PWM A6
#include "ste.h"//A0
#include "Encoder.h" //E0
int Flag_finish=0;
int count_j=0;
int main(void)
{
 LED_Init();
 LED_Close();
  delay_init(168);
 delay_ms(7000);//等待openmv 完成初始化
 TIM5_Int_Init(2000,8399);
 uart_init(115200);
 Beer_Init();
 speed_Init();
 Sensor_Init();
 ste_Init();
 KEY_Init();
 TIM_ETR_Init();
 LED_Open();//各初始化完成标志
 while(1)
 {
  if(KEY_Num==0)
  {
   KEY_Num=KEY_Scan(0);
   count_j=0;
```

```
38.   }
39.   if(KEY_Num!=0&&count_j==0)//有输入时延时2s
40.   {
41.    if(KEY_Num==1)
42.    {
43.     LED_Open();
44.     Beer_Open();
45.     delay_ms(500);
46.     Beer_Close();
47.     delay_ms(1500);
48.     count_j++;
49.    }
50.    else if(KEY_Num==2)
51.    {
52.     Beer_Close();
53.     Beer_Open();
54.     delay_ms(500);
55.     Beer_Close();
56.     delay_ms(1500);
57.     count_j++;
58.    }
59.
60.   }
61.   //前进识别
62.   if((KEY_Num==1||KEY_Num==2)&&Flag_back==0)
63.   {
64.    Forward_PWM(3000);
65.   }
66.
67.
68.   if(KEY_Num==1&&Flag_back==1)//倒车入库
69.   {
70.
71.    Backward_PWM(0);
72.    delay_ms(100);
73.    STE_PWM(0);
74.    TIM_SetCounter(TIM4,0);
75.    Backward_PWM(3000);
76.    while(TIM_GetCounter(TIM4)<170);
77.    TIM_SetCounter(TIM4,0);//向后
78.
79.    STE_PWM(-300);
```

```
80.    Backward_PWM(0);
81.    delay_ms(100);
82.       Forward_PWM(4000);
83.
84.
85.    while(TIM_GetCounter(TIM4)<400);
86.    Forward_PWM(0);
87.    STE_PWM(400);
88.    delay_ms(100);
89.    Backward_PWM(4000);
90.    Beer_Open();
91.    delay_ms(500);
92.    Beer_Close();
93.    while(Sensor1()==0);
94.    while(sensor2()==0);
95.    delay_ms(200);
96.    STE_PWM(0);
97.    TIM_SetCounter(TIM4,0);
98.    while(TIM_GetCounter(TIM4)<280);
99.    Forward_PWM(0);
100.      TIM_SetCounter(TIM4,0);//已入库
101.
102.      Beer_Open();
103.      delay_ms(500);
104.      Beer_Close();
105.      delay_ms(4500);
106.
107.      Forward_PWM(4000);//出库
108.  //    while(Sensor1()==0&&sensor2()==0);
109.      TIM_SetCounter(TIM4,0);
110.      while(TIM_GetCounter(TIM4)<250);
111.      STE_PWM(400);
112.      TIM_SetCounter(TIM4,0);
113.      while(TIM_GetCounter(TIM4)<1050);
114.      //进入完成阶段标志
115.      Flag_finish=1;
116.    }
117.
118.    else if(KEY_Num==2&&Flag_back==1)//侧方位
119.    {
120.     TIM_SetCounter(TIM4,0);
121.     Forward_PWM(3000);
```

```
122.      while(TIM_GetCounter(TIM4)<400);
123.      TIM_SetCounter(TIM4,0);
124.      Forward_PWM(0);
125.      Beer_Open();
126.      STE_PWM(200);
127.      Backward_PWM(4000);
128.      delay_ms(500);
129.      Beer_Close();
130.      while(Sensor1()==0);
131.      STE_PWM(-325);
132.      TIM_SetCounter(TIM4,0);
133.      while(TIM_GetCounter(TIM4)<600);//已入库
134.      Backward_PWM(0);
135.      Beer_Open();
136.      delay_ms(500);
137.      Beer_Close();
138.      delay_ms(4500);
139.
140.      Forward_PWM(4000);//出库
141.      while(sensor2()==0);
142.      delay_ms(400);
143.      STE_PWM(325);
144.      TIM_SetCounter(TIM4,0);
145.      while(TIM_GetCounter(TIM4)<770);
146.
147.      //进入完成阶段标志
148.      Flag_finish=1;
149.     }
150.
151.     if(Flag_finish==1&&Flag_back==1)//完成阶段标志
152.     {
153.      STE_PWM(0);
154.      TIM5_Int_Init(2000,8399);
155.      TIM_SetCounter(TIM4,0);
156.      Forward_PWM(3000);
157.      while(TIM_GetCounter(TIM4)<500);
158.      Forward_PWM(0);
159.      TIM_SetCounter(TIM4,0);
160.      Flag_finish=0;
161.      Flag_back=0;
162.      KEY_Num=0;
163.
```

```
164.     }
165.    }
166.   }
```

1.2 定时器代码

```
1. void TIM3_PWM_Init(u32 arr,u32 psc)
2. {
3.  //此部分需手动修改 IO 口设置
4.
5.  GPIO_InitTypeDef GPIO_InitStructure;
6.  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
7.  TIM_OCInitTypeDef  TIM_OCInitStructure;
8.
9.  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);   //TIM3 时钟使
   能
10.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);  //使能
   PORTA 时钟
11.
12.     GPIO_PinAFConfig(GPIOA,GPIO_PinSource6,GPIO_AF_TIM3); //GPIOA6 复
   用为定时器 3
13.
14.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;            //GPIOFA
15.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;         //复用功能
16.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //速度 100MHz
17.     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;       //推挽复用输
   出
18.     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;         //上拉
19.     GPIO_Init(GPIOA,&GPIO_InitStructure);               //初始化 PA6
20.
21.     TIM_TimeBaseStructure.TIM_Prescaler=psc;  //定时器分频
22.     TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上
   计数模式
23.     TIM_TimeBaseStructure.TIM_Period=arr;   //自动重装载值
24.     TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
25.
26.     TIM_TimeBaseInit(TIM3,&TIM_TimeBaseStructure);//初始化定时器 3
27.
28.     //初始化 TIM14 Channel1 PWM 模式
29.     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模
   式:TIM 脉冲宽度调制模式 2
30.      TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; /
   /比较输出使能
```

```
31.     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //输出极性:TIM 输出比较极性低
32.     TIM_OC1Init(TIM3, &TIM_OCInitStructure);  //根据 T 指定的参数初始化外设 TIM1 4OC1
33.
34.     TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);  //使能 TIM3 在 CCR1 上的预装载寄存器
35.
36.      TIM_ARRPreloadConfig(TIM3,ENABLE);//ARPE 使能
37.
38.     TIM_Cmd(TIM3, ENABLE);  //使能 TIM3
39.
40.
41.    }
42.
43.    //A0
44.    void TIM2_PWM_Init(u32 arr,u32 psc)
45.    {
46.     GPIO_InitTypeDef GPIO_InitStructure;
47.     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
48.     TIM_OCInitTypeDef TIM_OCInitStructure;
49.
50.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);//使能定时器时钟
51.
52.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);//使能 IO 口时钟
53.
54.     GPIO_PinAFConfig(GPIOA,GPIO_PinSource0,GPIO_AF_TIM2);//PA0 复用为定时器 2
55.
56.
57.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
58.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;          //复用功能
59.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //速度 100MHz
60.     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;        //推挽复用输出
61.     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;          //上拉
62.     GPIO_Init(GPIOA,&GPIO_InitStructure);                 //初始化 PA0
63.
64.
```

```
65.     TIM_TimeBaseStructure.TIM_Prescaler=psc;                    //定时器分频
66.     TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//向上计数模式
67.     TIM_TimeBaseStructure.TIM_Period=arr;                   //自动重装载值
68.     TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
69.     TIM_TimeBaseInit(TIM2,&TIM_TimeBaseStructure);          //初始化定时器
70.
71.
72.     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;           //选择PWM模式
73.     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;//比较输出使能
74.     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;   //输出极性低
75.     TIM_OC1Init(TIM2, &TIM_OCInitStructure);                    //初始化定时器通道
76.
77.     /*定时器主模式设置*/
78.     TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);
79.        TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_OC1Ref);
80.
81.     TIM_Cmd(TIM2, ENABLE);
82.
83.     TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);//使能定时器在ccr上的预装载寄存器
84.
85.     TIM_ARRPreloadConfig(TIM2,ENABLE);//ARPE使能
86.     TIM_SetCompare1(TIM2,1000);
87.    }
```

1.3 电机控制代码

```
1. void speed_Init(void)
2. {
3.  GPIO_InitTypeDef  GPIO_InitStructure;
4.   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);//使能GPIOC时钟
5.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
6.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
7.  GPIO_InitStructure.GPIO_OType =GPIO_OType_PP;
```

```
8.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;//100M
9.   GPIO_InitStructure.GPIO_PuPd =  GPIO_PuPd_NOPULL;//
10.      GPIO_Init(GPIOA, &GPIO_InitStructure);//初始化
11.     TIM3_PWM_Init(9999,83);
12.     Forward_PWM(0);
13.    }
14.    void Forward_PWM(u16 PWM)
15.    {
16.
17.     GPIO_ResetBits(GPIOA,GPIO_Pin_7);
18.     TIM_SetCompare1(TIM3,PWM);
19.
20.    }
21.
22.
23.    void Backward_PWM(u16 PWM)
24.    {
25.     GPIO_SetBits(GPIOA,GPIO_Pin_7);
26.     TIM_SetCompare1(TIM3,PWM);
27.
28.    }
```

1.4 编码器代码

```
1.  void TIM_ETR_Init(void)
2.  {
3.  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
4.  GPIO_InitTypeDef GPIO_InitStructure;
5.
6.  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);
7.  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
8.
9.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
10. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
11. GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
12. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
13. GPIO_Init(GPIOE,&GPIO_InitStructure);
14.
15. GPIO_PinAFConfig(GPIOE,GPIO_PinSource0,GPIO_AF_TIM4);    //开启第二功
    能
16.
17. TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
18. TIM_TimeBaseStructure.TIM_Prescaler = 0;
```

```
19. TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
20. TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
21. TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
22.
23. TIM_ETRClockMode2Config(TIM4,TIM_ExtTRGPSC_OFF,TIM_ExtTRGPolarity_No
    nInverted,0);  //外部时钟源模式2
24. TIM_Cmd(TIM4, ENABLE);
25. TIM_GetCounter(TIM4);
26. TIM_SetCounter(TIM4,0);
27. }
```

1.5 舵机控制代码

```
1.  void ste_Init(void)
2.  {
3.   TIM2_PWM_Init(19999,83);
4.   TIM_SetCompare1(TIM2,STE_Median_PWM);
5.  }
6.  void STE_PWM(int PWM)
7.  {
8.   if(PWM<0)
9.   {
10.   PWM=PWM*1.62;
11.  }
12.  if(PWM>400)
13.  {
14.   PWM=400;
15.  }
16.  else if(PWM<-648)
17.  {
18.   PWM=-648;
19.  }
20.  TIM_SetCompare1(TIM2,PWM+STE_Median_PWM);
21. }
22.
```

1.6 色块识别代码

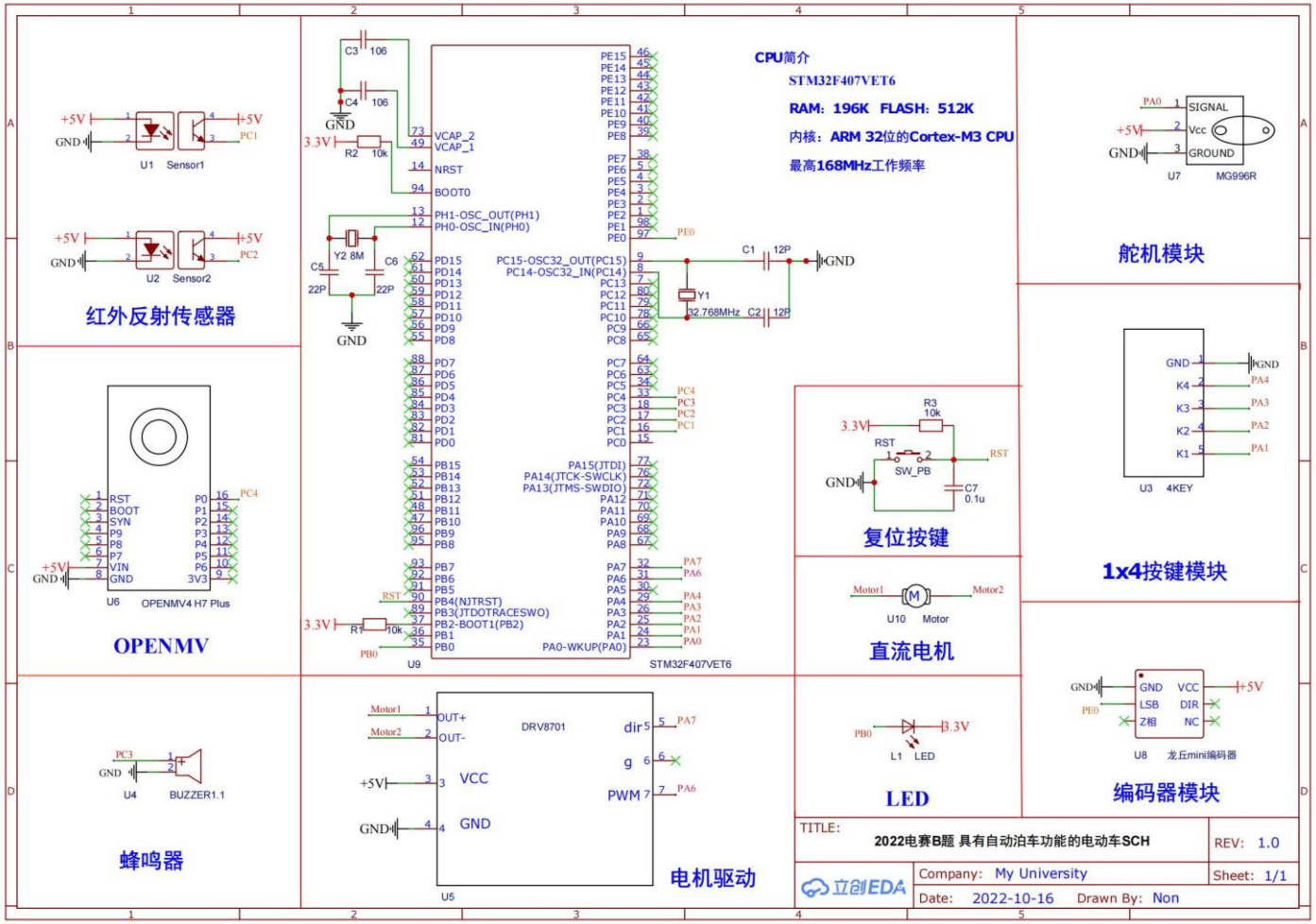
```
1.  # Single Color Grayscale Blob Tracking Example
2.  #
3.  # This example shows off single color grayscale tracking using the O
    penMV Cam.
4.
```

```
import sensor, image, time, math
import pyb
from pyb import Pin
# Color Tracking Thresholds (Grayscale Min, Grayscale Max)
# The below grayscale threshold is set to only find extremely bright
 white areas.
thresholds = (0, 74)

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False) # must be turned off for color tracking
sensor.set_auto_whitebal(False) # must be turned off for color track
ing
sensor.set_vflip(1)
sensor.set_hmirror(1)
clock = time.clock()
PIN1=Pin('P0',Pin.OUT_PP)
PIN1.low()
led = pyb.LED(3)
led1 = pyb.LED(1)
led.off()
count_i=0
count_k=0
count_j=0
# Only blobs that with more pixels than "pixel_threshold" and more a
rea than "area_threshold" are
# returned by "find_blobs" below. Change "pixels_threshold" and "are
a_threshold" if you change the
# camera resolution. "merge=True" merges all overlapping blobs in th
e image.
while(True):
    clock.tick()
    img = sensor.snapshot()
    blobs = img.find_blobs([thresholds], roi= [145, 120, 30, 140],pi
xels_threshold=100, area_threshold=100, merge=True)
    for blob in blobs:
        count_k=count_k+1
    if count_k<2:
        count_j=1
        count_k=0
```

```
41.     else:
42.        count_k=0
43.        count_j=0
44.     for blob in blobs:
45.         # These values depend on the blob not being circular - other
    wise they will be shaky.
46.         if blob.elongation() > 0.5:
47.             img.draw_edges(blob.min_corners(), color=0)
48.             img.draw_line(blob.major_axis_line(), color=0)
49.             img.draw_line(blob.minor_axis_line(), color=0)
50.         # These values are stable all the time.
51.         img.draw_rectangle(blob.rect(), color=127)
52.         img.draw_cross(blob.cx(), blob.cy(), color=127)
53.         #print(blob.pixels())
54.         print(count_j)
55.         if blob.pixels()>1000 and count_j==1:
56.            count_i=count_i+1
57.            led.on()
58.            if count_i<=2:
59.              time.sleep_ms(500)
60.            led.off()
61.            if count_i==3:
62.              count_i=0
63.              PIN1.high()
64.              led1.on()
65.              time.sleep_ms(1000)
66.              PIN1.low()
67.              led1.off()
68.
69.
70.         #if blob.pixels()/last_num>2:
71.             #print(2)
72.         #print(blob.pixels())
```

附录 2.电路连接原理


CPU简介
STM32F407VET6
RAM：196K FLASH：512K
内核：ARM 32位的Cortex-M3 CPU
最高168MHz工作频率
红外反射传感器
OPENMV
蜂鸣器
电机驱动
复位按键
直流电机
LED
舵机模块
1x4按键模块
编码器模块
TITLE:
2022电赛B题 具有自动泊车功能的电动车SCH
REV: 1.0
Company: My University
Sheet: 1/1
Date: 2022-10-16
Drawn By: Non

附录 3.实验结果照片

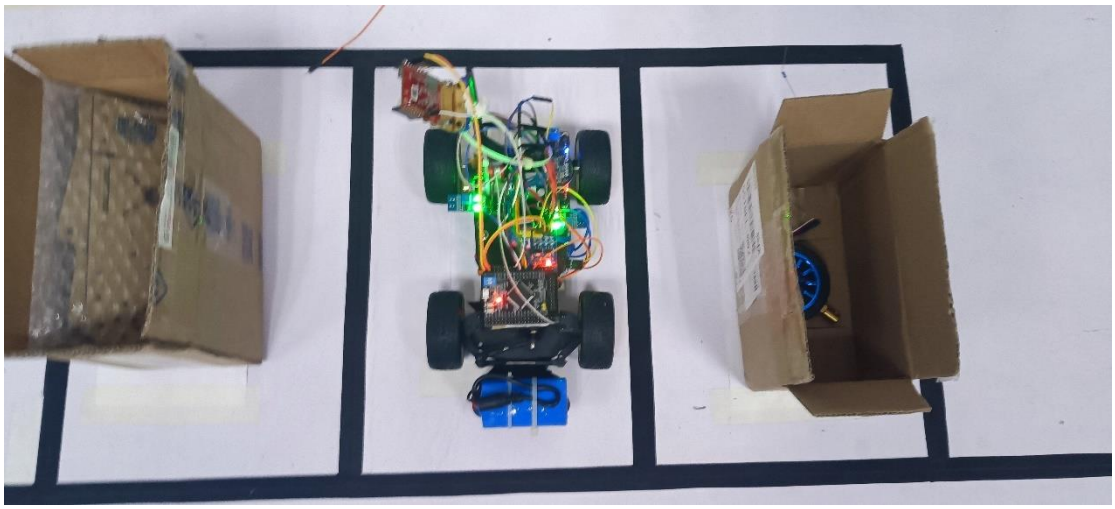


图 1. 邻库有车倒车入库测试结果图

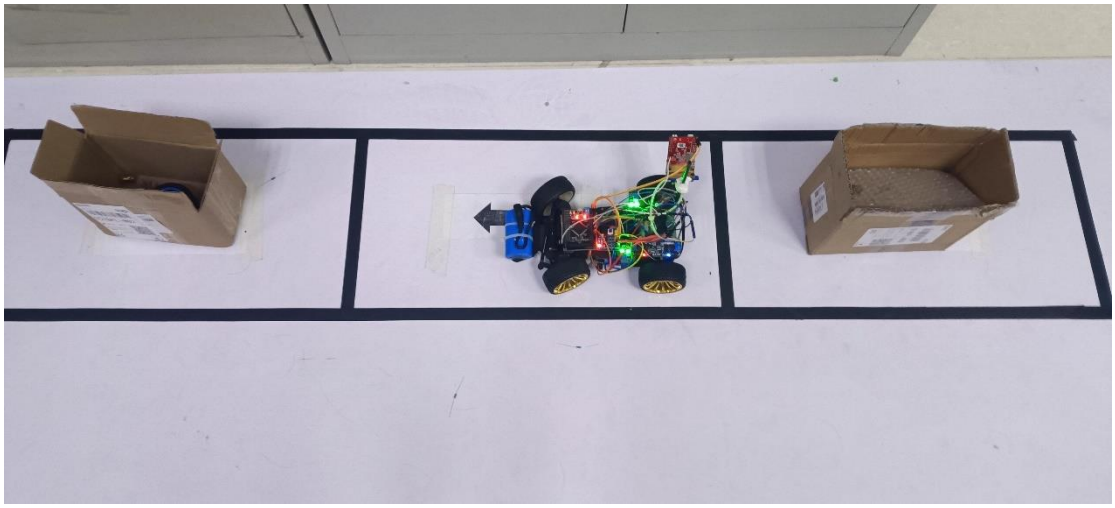


图 2. 邻库有车侧方位停车测试结果图

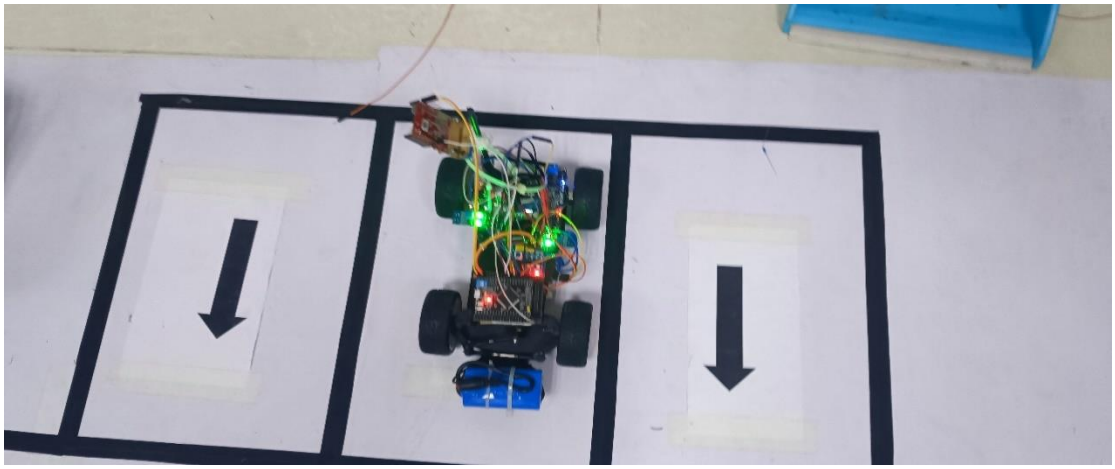


图 3. 邻库无车倒车入库测试结果图

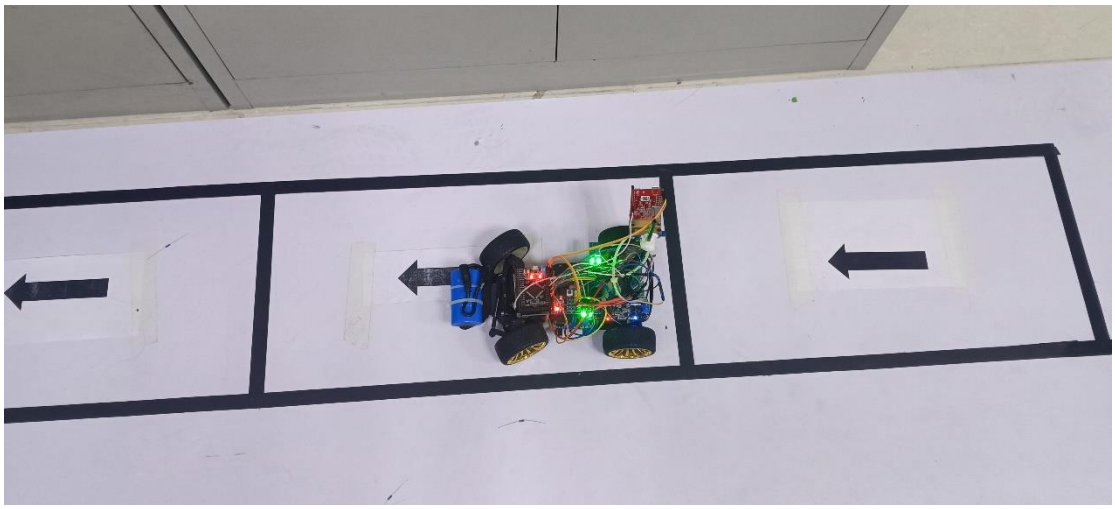


图 4. 邻库无车侧方位停车测试结果图