项 目 报 告

课程名称：嵌入式系统与机器人

课程编号：ME432

学　　号：

姓　　名：

专　　业：机器人工程

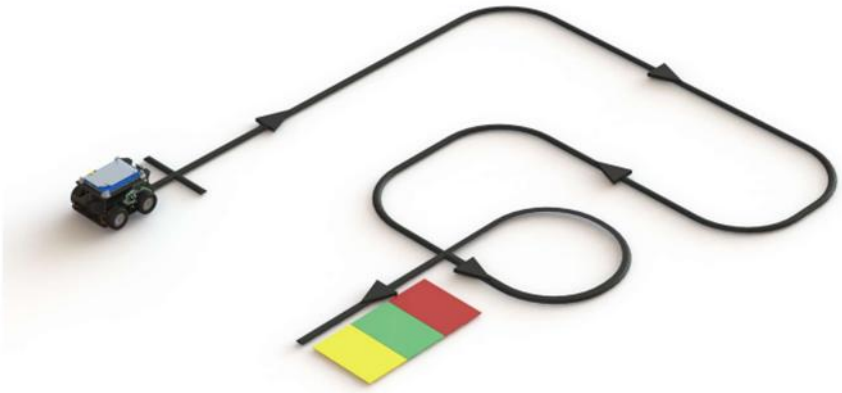
指导教师：

2022 年 6 月 5 日

目录

一、项目目标

嵌入式系统与机器人本课程的期末项目是智能循迹小车。我们需要运用课上所学知识以及自主查找资料，最后使得以 STM32F767 板控制的智能小车，利用传感器感知路线和颜色变化，自主进行判断以及相执行相应的动作，实现循迹，避障，和颜色判断的功能。任务目标分为两点，如下图所示：



首先需要进行循迹，即小车要绕着黑色路线前进，在十字路口以及圆周形路径上可以正确选择，其次要进行颜色识别，小车通过底部的颜色传感器识别颜色，并在指定的颜色上停止前进。

二、项目背景

嵌入式系统（Embedded System）是一种特殊的计算机系统 ，也可以是可定制、多样性、专用的操作系统。它包括硬件和软件的完整的计算机系统 ，其定义是“嵌入式系统是以应用为中心 ，以计算机技术为基础 ，并且软硬件可剪裁 ，适用于应用系统对功能、可靠性、成本、体积和功耗有严格要求的专用计算机系统”。嵌入式的特点不仅表现在系统内核小巧精简 ，多任务操作系统 ，具备高速固化的软硬件性能和标准 ，更表现在在高专用性、高实时性、高软硬件切合度以及应用的开发随市场导向和发展所需的的快速的更新迭代性。

嵌入式系统的应用包括工业控制嵌入式设备、消费电子、机器人与人工智能以及物联网等，随着嵌入式系统的发展及其广泛普及,人们越

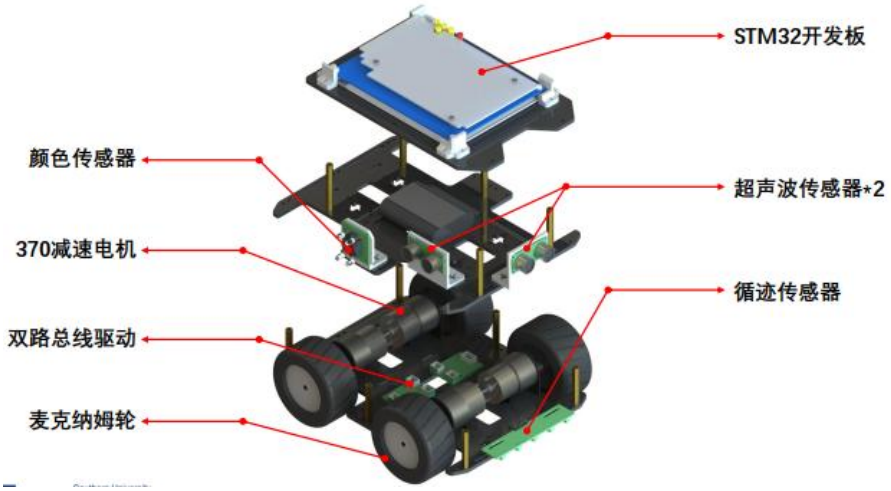
来越意识到嵌入式系统的重要性，目前嵌入式系统市场竞争力强，普及面广，并且越来越智能化，这都决定了嵌入式系统的发展前景不可估量。嵌入式系统的首要发展体现在开发工具的进一步强化和操作系统的进一步升级上，因为网络资源和信息愈加丰富，对信息资源的共享要求越来越高，并且目前的因特网技术已经非常成熟，不能够满足未来更大程度上的需要，所以对开发工具进一步强化和对操作系统进一步升级势在必行。除此之外，未来还需要对网络接口进行统一，还需要注意嵌入式系统的用户友好性，嵌入式系统发展的最终目的是为了让人们更好地应用，让人们的工作和生活更加便利，所以注意嵌入式系统的人机交互功能是非常重要的，是在未来的发展中不能忽视。

对于本实验来讲，我们需要完成以 STM32F767 开发板控制的小车进行循迹以及颜色识别的任务，这将训练我们对于软件和硬件之间的配合能力、加深对于理解关于嵌入式的基本知识以及串口配合、硬件控制等的能力，提高问题分析能力和解决能力。

三、硬件设计

整体结构：

智能循迹避障小车的硬件部分主要由如下几部分组成，如下图所示：

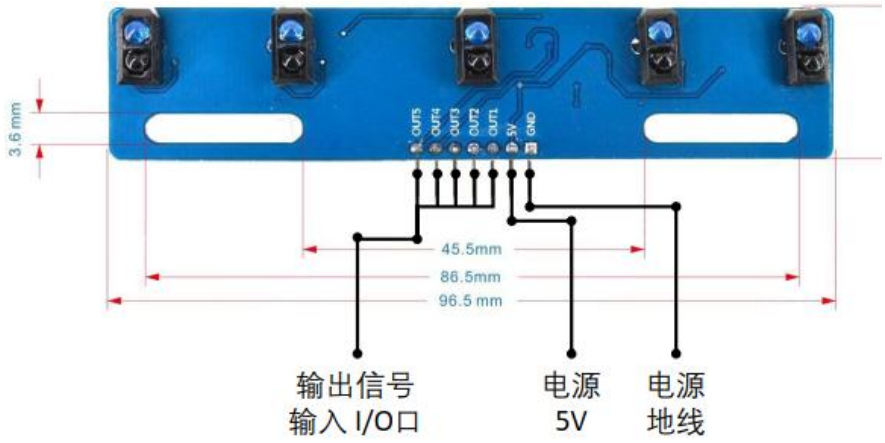


包括 STM32F767 开发板，一个循迹传感器、一个颜色传感器、四个 370 减速电机、麦克纳姆轮，双路总线驱动以及拼装小车所需要的连接件零件等。（本次实验由于降低实验内容并没有超声传感模块）

4.1 各部分硬件简要说明

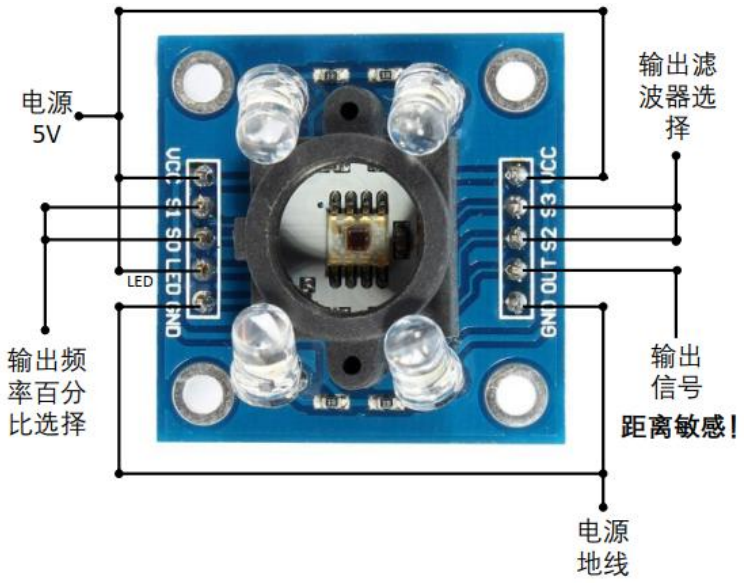
1） 循迹传感器

循迹传感使用 TCRT5000 传感模块，该模块利用红外检测黑、白路线。模块共 7 个接口，分别为两个供电接口（VCC、 GND），五个信号输出接口。当检测到黑线时，传感器对应位置 LED 等会熄灭，输出 0；反之会点亮，输出 1. 如下图所示：



2） 颜色传感器

颜色传感使用 TCS3200 颜色传感模块，模块可识别 RGB 三通道颜色值。通过调整模块滤波，可以控制通过传感的颜色。颜色传感使用之前需要进行白平衡，对接受的光波进行标准化，设定为（255，255， 255），在之后的识别，将捕获的光波乘以比例因子。模块共有 10 个引脚，四个供电接口（VCC、 GND），LED 控制接口（LED），输出频率百分比选择（S0、S1），以及输出滤波器选择（S2、S3）。如图 10 所示：



3） 电机

电机选择直流无刷电机，利用总线马达驱动模块进行驱动，一个模块具有两个电机接口以及两个总线通信接口，通信接口具有三个引脚，两个供电接口（VCC、 GND），一个通信接口（TX\RX）。其中供电接口与电池相连接，一个通信接口与 STM32 开发板的发送信号的端口相连接，下图即为本次实验所使用的电机：

3.2 各部分组件之间端口的连接

项目所涉及的端口连接的关系如下图所示

模块	模块接口	开发板接口	类型
总线马达驱动	RX\TX	PA2	USART
	VCC、GND	连接电池（5V）	
颜色传感模块	S0		GPIO
	S1		GPIO
	S2		GPIO
	S3		GPIO
	LED		GPIO

	OUT		TIM
	VCC、GND	开发板供电口（5V）	
循迹传感模块	OUT1		GPIO
	OUT2		GPIO
	OUT3		GPIO
	OUT4		GPIO
	OUT5		GPIO
	VCC、GND	开发板供电口（5V）	

四、软件设计

4.1 各部分的控制

1） 电机

电机利用 PWM 进行速度控制，我们小组采用的是直接通过命令“#idPpwmTtime”进行控制，其中，id 为电机编号，范围为 000~254，pwm 范围为 0500~2500，大于 1500 正传，小于 1500 反转，time 为电机旋转时间，范围为 0000~9999。由于其他控制电机方式的不稳定性，我们小组直接更改这个命令以此来调电机的速度，避开电机的死区；

2） 循迹模块

本次实验所使用的循迹模块，通过五个 OUT 口返回传感器所读到的黑白颜色信息，我们通过传感器返还的数值，进而对电机发布不同的指令，使四个电机配合可以完成循迹直行、转弯等动作；

3） 颜色模块

本次实验所使用的颜色模块，在使用时我们小组采用的策略是让颜色传感器首先进行白平衡，然后设置颜色范围，使颜色传感器在返还符合条件的值的时候然后进行相关命令，改变电机的速度，进而使小车达到识别不同颜色进行不同操作的结果；

4.2 各部分控制代码简单分析

1） 电机驱动

电机的控制可以通过串口发送指令实现：

指令	注释
#idPpwmTtime!	1）id范围是000~254，必须为三位数，不足补0。特别地，255为广播ID，所有在线的设备均会响应。 2）pwm范围是0500~2500，必须为四位数，不足补0。1500表示停止，大于1500正传，小于1500反转。与1500的差距越大，转速越快！另外，正转反转都有一定死区，每一个电机可能不一样，需要自行标定。 3）time表示旋转时间，必须为四位数。范围0000~9999，特殊地，0000代表循环执行。

小车驱动的具体思路是：通过麦轮的正确组合以及不同电机的正反转驱动可以实现小车的全向运动。

首先在 usart.c 文件中，定义各向运动对应的指令：

```
typedef uint8_t u8;

// left
u8 mess2_ne[] = "#002P0980T0500!";
u8 mess3_ne[] = "#003P1150T0500!";
u8 mess1_po[] = "#001P1200T0500!";
u8 mess4_po[] = "#004P1800T0500!";

// right
u8 mess1_ne_r[] = "#001P1750T0500!";
u8 mess4_ne_r[] = "#004P1090T0500!";
u8 mess2_po_r[] = "#002P1700T0500!";
u8 mess3_po_r[] = "#003P1700T0500!";

// straight
u8 mess1_po_s[] = "#001P1250T0500!";
u8 mess4_po_s[] = "#004P1700T0500!";
u8 mess2_po_s[] = "#002P1700T0500!";
u8 mess3_po_s[] = "#003P1700T0500!";

//stop
u8 mess1_0[] = "#001P1500T0000!";
u8 mess2_0[] = "#002P1500T0000!";
u8 mess3_0[] = "#003P1500T0000!";
u8 mess4_0[] = "#004P1500T0000!";
```

用 HAL_UART_Transmit 函数执行串口通信，直接把指令发送给电机。这里将不同指令的组合分别封装在不同的函数中（代表小车的不同行为），便于后续直接调用：

```
void forward(void) //quan bu zheng zhuan
{
    HAL_UART_Transmit(&huart2,mess1_po_s,sizeof(mess1_po_s),400);
    HAL_UART_Transmit(&huart2,mess2_po_s,sizeof(mess2_po_s),400);
    HAL_UART_Transmit(&huart2,mess3_po_s,sizeof(mess3_po_s),400);
    HAL_UART_Transmit(&huart2,mess4_po_s,sizeof(mess4_po_s),400);

}

void stop(void)
{
  HAL_UART_Transmit(&huart2,mess1_0,sizeof(mess1_0),400);
  HAL_UART_Transmit(&huart2,mess2_0,sizeof(mess2_0),400);
  HAL_UART_Transmit(&huart2,mess3_0,sizeof(mess3_0),400);
  HAL_UART_Transmit(&huart2,mess4_0,sizeof(mess4_0),400);
}

void left_rotate(void)
{
  HAL_UART_Transmit(&huart2,mess1_po,sizeof(mess1_po),400);
  HAL_UART_Transmit(&huart2,mess2_ne,sizeof(mess2_ne),400);
  HAL_UART_Transmit(&huart2,mess3_ne,sizeof(mess3_ne),400);
  HAL_UART_Transmit(&huart2,mess4_po,sizeof(mess4_po),400);
}

void right_rotate(void)
{
  HAL_UART_Transmit(&huart2,mess1_ne_r,sizeof(mess1_ne_r),400);
  HAL_UART_Transmit(&huart2,mess2_po_r,sizeof(mess2_po_r),400);
  HAL_UART_Transmit(&huart2,mess3_po_r,sizeof(mess3_po_r),400);
  HAL_UART_Transmit(&huart2,mess4_ne_r,sizeof(mess4_ne_r),400);
}
```

最后在 main.c 中声明上述函数，即可直接调用：

```
void left_rotate(void);
void right_rotate(void);
void forward(void);
void stop(void);
```

2） 颜色识别

首先根据main.c函数中合适的delay时间确定颜色传感器的计时器周期，总的来说，周期需要小于 main 中的循环的延迟时间。此处在循迹模块中的最优延迟时间为 30ms，因此将时钟设置更改如下：

```
htim1.Instance = TIM1;
htim1.Init.Prescaler = 19;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 50000;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
```

接着在 tim.c 中编写接收滤波器脉冲的代码，注意每一次循环完成后需将计 数器中的数值清零，以免脉冲数不断增加：

```
if(htim->Instance == TIM1){

  count = __HAL_TIM_GET_COUNTER(&htim2); //将TIM2统计的脉冲数存入count
  __HAL_TIM_SET_COUNTER(&htim2,0);  //TIM2清零，以便下一次重新计算脉冲数

  //printf("Updated!\r\n");
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_0);

  switch(flag){
  case 0:
      printf("---TCS START!---\r\n");
      TCS_Next(0, 0);
      break;
  case 1:
      printf("RED = %d\t", count); //打印10ms内的红色通过滤波器时，TCS3200输出的脉冲数
      cnt[0] = count; //储存到数组
      TCS_Next(1, 1); //下一次选择绿色光线通过滤波器的模式
      break;
  case 2:
      printf("GREEN = %d\t", count); //打印10ms内的绿色通过滤波器时，TCS3200输出的脉冲数
      cnt[1] = count; //储存到数组
      TCS_Next(0, 1); //下一次选择蓝色光线通过滤波器的模式
      break;
  case 3:
      printf("BLUE = %d\r\n", count); //打印10ms内的蓝色通过滤波器时，TCS3200输出的脉冲数
      printf("---TCS END!---\r\n");
      cnt[2] = count; //储存到数组
      TCS_Next(1, 0); //无滤波器的模式
      break;
  default:
      count = 0;  //计数器清零
      break;
```

在 main.c 中设置一个只运行一次的函数，完成白平衡：

```
LED_ON;   //打开四个白色LED，进行白平衡
HAL_Delay(1500);    //延时三秒，等待识别

//通过白平衡测试，计算得到白色物的RGB值255与0.5秒内三色光脉冲数的RGB比例因子
for(int i=0;i<3;i++)
{
  RGB_Scale[i] = 255.0/cnt[i];
  printf("%51f  \r\n", RGB_Scale[i]);
}
//红绿蓝三色光分别对应的0.5s内TCS3200输出脉冲数，乘以相应的比例因子就是我们所谓的RGB标准值
//打印被测物体的RGB值

for(int i=0; i<3; i++)
{
  printf("%d \r\n", (int) (cnt[i]*RGB_Scale[i]));
}
printf("White Balance Done!\r\n");
//白平衡结束
```

完成白平衡后，只需将滤波器的每个值与得到的白平衡数列的对应值相乘就可得到对应颜色的 RGB 值，再根据 RGB 值完成条件判定：

```
for(int i=0; i<3; i++)
{
  if(i==0)
    printf("RGB = (");
  if(i==2)
    printf("%d)\r\n",(int) (cnt[i]*RGB_Scale[i]));
  else
    printf("%d, ", (int) (cnt[i]*RGB_Scale[i]));
}
difference1 = cnt[2]*RGB_Scale[2]-cnt[1]*RGB_Scale[1];
difference2 = cnt[2]*RGB_Scale[2]-cnt[0]*RGB_Scale[0];
```

3） 循迹

循迹模块使用五个并排的红外线传感器，通过五个传感器检测到的黑色区域的数量和位置判断小车前方的路况，总体来说只需设置 GPIO 口与判断条件就可实现：

```
/*TRACKING*/
GPIO_InitStruct.Pin=TRACK_INFRARED_M_PIN  | TRACK_INFRARED_L2_PIN | TRACK_INFRARED_R2_PIN | TRACK_INFRARED_L1_PIN | TRACK_INFRARED_R1_PIN;
GPIO_InitStruct.Mode=GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
//GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
int L1 = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_4);
int L2 = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_5);
int M = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_6);
int R2 = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_7);
int R1 = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_8);

}else{
  printf("forward");
  if(L1+L2<R1+R2&&R1+R2-L1-L2>=2){

    left_rotate();

  }else if(L1+L2>R1+R2&&L1+L2-R1-R2>=2){
    right_rotate();

  }else{
    forward();
  }
```

五、实验过程以及结果分析

5.1 各部分问题解决的策略

1）电机调试问题

在调试过程中，发现 1 号电机的转向控制比较不同（发送正转指令会令其反转），而且各电机正转反转都存在死区，所以只能慢慢通过调参数，观察再继续调参数进而得到相对较优的结果。其中遇到的问题有：杜邦线的连接很容易松动导致的接触不良，使电机的速度不稳定，然后在全部更换杜邦线之后，电机速度的调试基本比较顺利；

2）颜色识别问题

颜色识别问题解决策略是首先以白色对该传感器进行白平衡（255，255，255），然后通过几种策略的对比，我们小组最后选择了以差值来判断小车所识别的颜色，当差值达到我们通过实验所设置的范围时，小车即会停止，达到了颜色识别的效果；

3）循迹策略

通过 5 个 OUT 口返回的数值，我们大致可以判断小车目前位于轨迹的那个位置（例如偏左、偏右、正中等），通过这个位置的判断，我们小组对小车发布左转、右转以及直行的命令。这个命令与颜色识别的命令的优先级更高，当循迹任务完成之后，循迹传感器接收到的信号将不满足预设的所有情况，即开始执行颜色识别策略，最终完成目标；

5.2 实验结果的简单分析

关于颜色识别和循迹的控制以及策略前面都已经提到过了，这部分将进一步分别说明这两者的调参过程：

1）关于颜色识别

我们小组在最开始进行颜色调节的时候，是先进行白平衡然后设置颜色范围，但是后面发现，这种方法每次白平衡之后，目标范围的颜色范围差别比较大，甚至差别有可能超过 100，所以后面我们改用差值的方法，通过计算三个 RGB 的数值的差值，来确定所识别的颜色，这样的方式每次出现的结果差不会太大，比较稳定，经过一系列的调节参数，我们发现在

$$difference1 = RGB\_G - RGB\_B;$$

$$difference2 = RGB\_G - RGB\_R;$$

只有当两个差值都大于 30 时，判断即为小车出现到了目标颜色区域即蓝色区域上，小车停止。

2）关于循迹

循迹的算法我们并没有复杂化。首先接受循迹传感器返回的数值（R2,R1,M,L1,L2 分别对应右 2，右 1，中间、左 1、左 2），当左边的传感器识别到黑线，即$L1 + L2 < R1 + R2$,则小车向左转，反之则向右转，如果数值均为零，说明小车正好位于轨迹的正中间，小车将直行；

六、总结（只能编这么点了，你们可以再加点）

本次项目我们小组实现了软件与硬件相结合的开发过程，在本次实验中，

我们更加深刻的学习到了嵌入式的基本原理、接口技术以及运动控制等，还在其中学习到了例如 STM32CubeMX 以及 Keil uVision 等工具软件的使用，进一步提高了我们小组各成员问题分析以及解决的能力，最终成功的实现了小车的电机协调控制、循迹前行以及颜色识别等任务。

诚然，虽然完成了项目的目标，但是我们小组未来可以探索进步的空间还很大，项目虽已画上句点，但是有关嵌入式知识的学习却是起点，未来我们将继续学习运用有关嵌入式的知识，在其他项目中使其得以应用。

最后，非常感谢老师、助教以及同学的帮助！

引用文献：

[1] 嵌入式系统的应用分析 许斌 科技创新导报 2017.NO27

[2] 浅谈嵌入式系统的应用 郑建南 ，曾 珍 科学家 2016 4.（9）