轮式机器人避障系统

设计说明书

单　位：西北工业大学
联 系 人：闫奕岐
联系电话：15902973834
地　址：西北工业大学友谊校区
邮　编：710072

目录

1 引言

1.1 软件编写目的

轮式机器人避障系统是以传统的轮式机器人避障竞赛项目为契机开发的通用系统。通过该系统，机器人将由 RGB 彩色摄像机转换为 HSV 空间图像，探测终点位置，同时根据当前避障状态进行自主决策，在保证顺利避障的前提下自主行进到终点线位置。为了方便使用，将全部功能集成到 GUI 界面上，在界面上提供用户自定义色标、自定义指令集、串口配置等功能接口。

1.2 技术特色

- 基于 HSV 色彩空间的目标检测

本系统利用 QT 中的 C++平台和 OpenCV 视觉处理库，设计出一整套改进的目标检测算法：通过色值转换公式，将 RGB 图像转换为 HSV 图像，再通过对 S 通道进行阈值处理排除光照影响，然后使用基于 CamShift 的目标追踪的算法检测终点线位置。目标检测的结果将作为下一步决策的重要参考信息。

- 基于有限状态机的策略设计

有限状态机（Finite-state machine, FSM），又称有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。使用有限状态机进行建模优势明显：各个状态的细节互相独立地进行设计互不干扰；然后通过状态转移条件将各个状态互相联通。在改进策略时，只需要调整状态转移函数（也就是修改状态转移条件），就可以方便的测试多种不同的策略，极大提高了开发效率。

1.3 运行环境

1）硬件环境

CPU>=1.0GHz;

内存>=1G;

硬盘>=16G;

2）软件环境

- 软件开发的软件环境

QT 运行库;

OpenCV 环境;

- 软件运行的软件环境

  QT Creator 软件平台;

  MinGw3.0.1 编译平台;

2 功能实现主架构

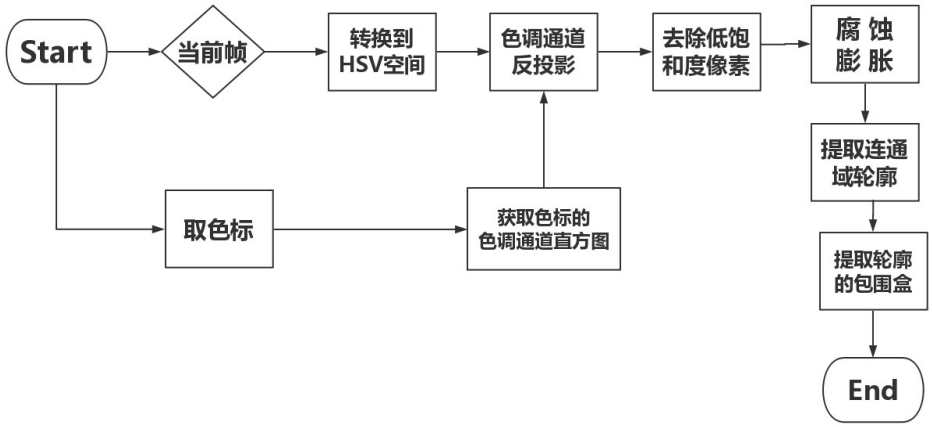


图 2-1 处理一帧图像

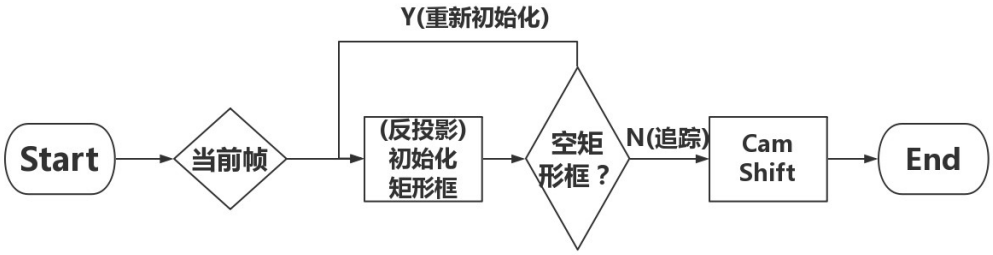


图 2-2 目标追踪算法框架

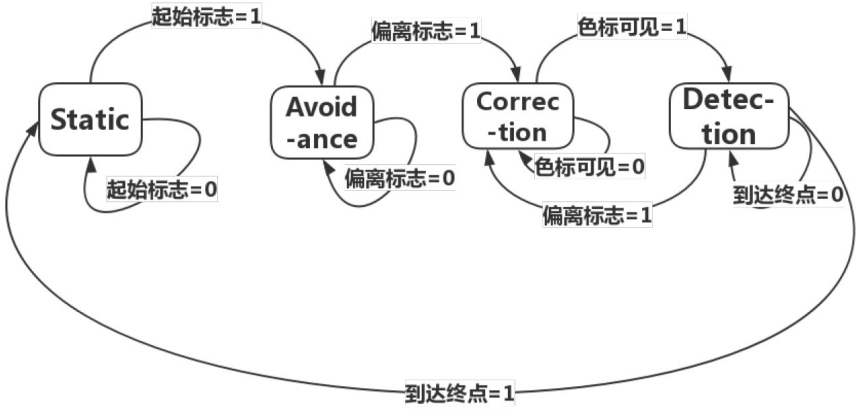


图 2-3 有限状态机图示

注：一共定义了四种状态，Static(机器人静止)、Avoidance(机器人正在避障)、Correction(机器人修正自身水平位置)、Detection(探测终点)。状态转移条件由各个标志位组成，标志位的值在程序运行过程中动态更新

3 代码核心算法说明

3.1 改进的颜色识别算法

1) 色彩模型

所谓色彩模型就是指某个三维颜色空间中的一个可见光子集，它包含某个颜色域的所有颜色。例如，RGB 颜色模型就是三维直角坐标颜色系统的一个单位正方体。颜色模型的用途是在某个颜色域内方便的指定颜色，由于每一个颜色域都是可见光的子集，所以任何一个颜色模型都无法包含所有的可见光。在大多数的彩色图形显示设备一般都是使用红、绿、蓝三原色，真实感图形学中的主要的颜色模型也是 RGB 模型，但是红、绿、蓝颜色模型用起来不太方便，它与直观的颜色概念如色调、饱和度和亮度等没有直接的联系。

下面比较两种主流色彩模型，并通过比较给出选择 HSV 的原因。

（1）HSV 色彩模型

在 HSV 模型中，每一种颜色都是由色相(Hue，H)，饱和度(Saturation，S)和色明度(Value，V)所表示的，可以用倒圆锥模型来表示(图 3-1)。

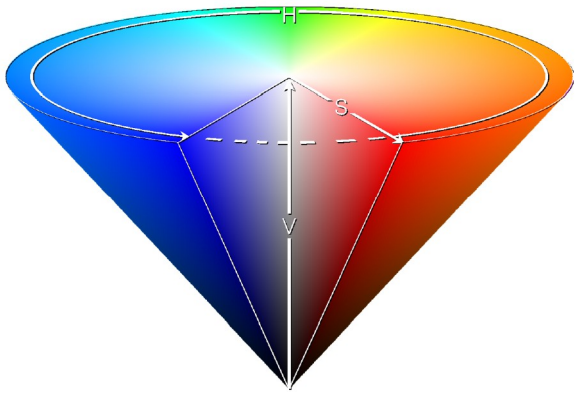


图 3-1 HSV 色彩模型

色相是由可见光光谱中各分量成分的波长来确定的，它是颜色光的基本特性，也就说色相分量是区分“不同”颜色的重要参考。饱和度反映了彩色的浓淡程度，取决于彩色光中白光含量，掺入白光越多，饱和度越高，因此饱和度分量

的处理对抑制光照影响至关重要。

（2）RGB 色彩模型

RGB(Red, Green, Blue)颜色模型采用笛卡尔（Cartesian）坐标系，图 3-7 显示了 RGB 空间单位立方体，在原点(0,0,0)处，三个分量均为最小值状态，表示为黑色，在立方体的对角顶点(1,1,1)处三分量为最大值，相应表示为白色。处于单位立方体主对角线上的三分量被称为灰度对角线，对应于每一个像素的亮度分量。其中红、绿、蓝相应位于立方体的三主轴的端点处，而其他颜色则是它们的线性组合。结构如图 4-4 所示。

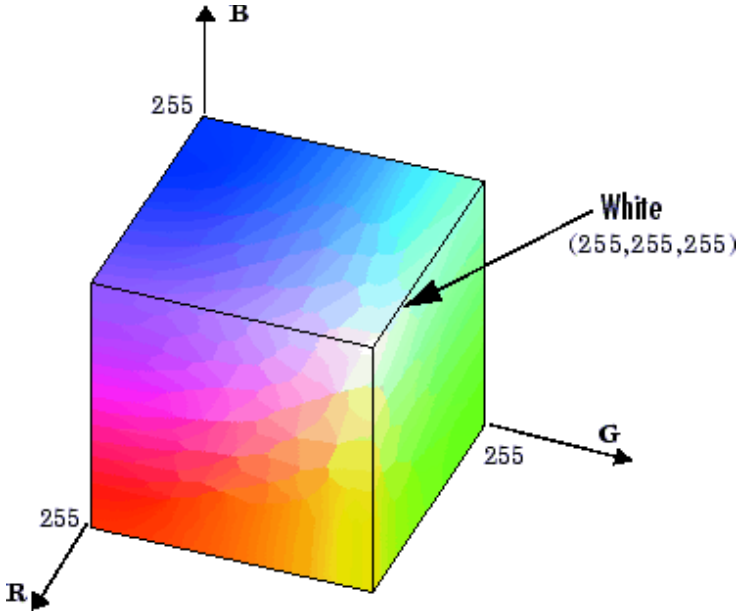


图 3-2 RGB 色彩模型

因为 RGB 并不是基于色彩描述的色彩空间，即 RGB 空间中的“笛卡尔距离”和颜色的实际差距没有一一对应的关系，不同程度的颜色差距可能对应着相同的笛卡尔距离，因此使用 RGB 模型进行色彩匹配不是一个好的选择。而变换到 HSV 空间就很好的解决了这个问题。

（3）RGB 图像转换为 HSV 图像

由于 HSV 色彩空间和 RGB 色彩空间只是同一物理量的不同表示法，因而它们之间存在着转换关系。

$$H=\begin{cases} undefined & if\ \max=\min \\ 60^\circ\times\dfrac{g-b}{\max-\min}+0^\circ, & if\ \max=r \quad and \quad g\geq b \\ 60^\circ\times\dfrac{g-b}{\max-\min}+360^\circ, & if\ \max=r \quad and \quad g<b \\ 60^\circ\times\dfrac{b-r}{\max-\min}+120^\circ, & if\ \max=g \\ 60^\circ\times\dfrac{r-g}{\max-\min}+240^\circ, & if\ \max=b \end{cases} \tag{3-1}$$

$$S=\begin{cases} 0 & if\ \max=0 \\ \dfrac{\max-\min}{\max}=1-\dfrac{\min}{\max}, & otherwise \end{cases} \tag{3-2}$$

$$V=\max \tag{3-3}$$

式中，max 为 RGB 三者最大值；

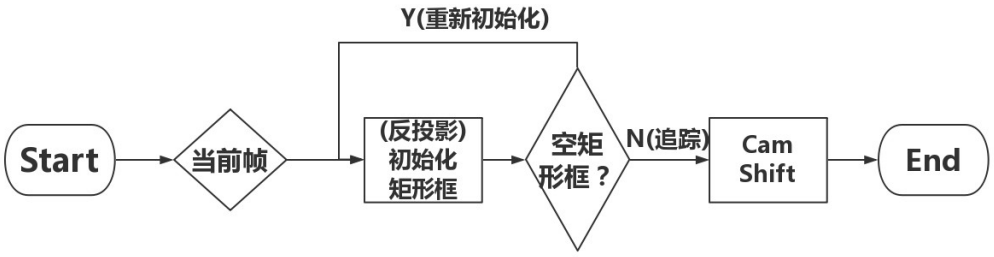
min 为 RGB 三者最小值。

通过该转换算法将彩色相机获取的 RGB 图像转化为 HSV 图像，从而可以减轻光照等因素对物体识别的影响。

2) 彩色图像目标追踪

对于多数场景色彩复杂度低，利用颜色跟踪即可有效对目标有效的分割识别，根据机器人运动特性 Camshift 算法可以对色标有效跟踪识别，其速率、准确度均能达到需求。

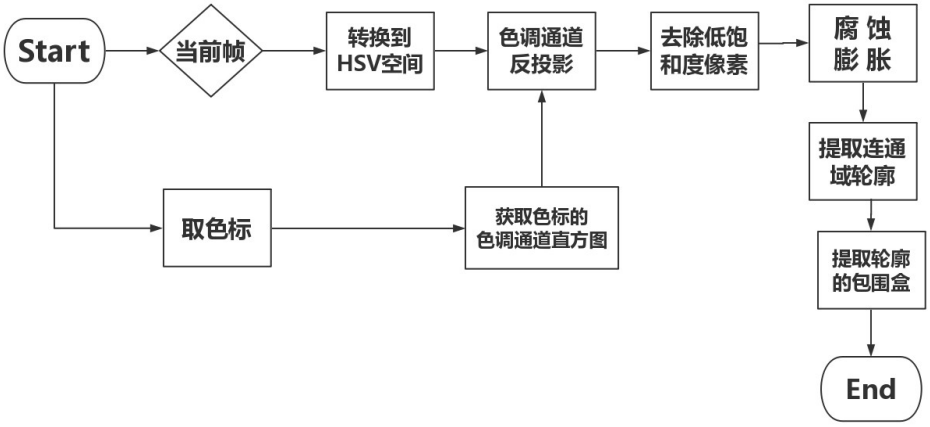
算法框架如下：



算法主要为两个部分：

(1) 直方图反投影初始化目标框

直方图反投影的算法流程如下

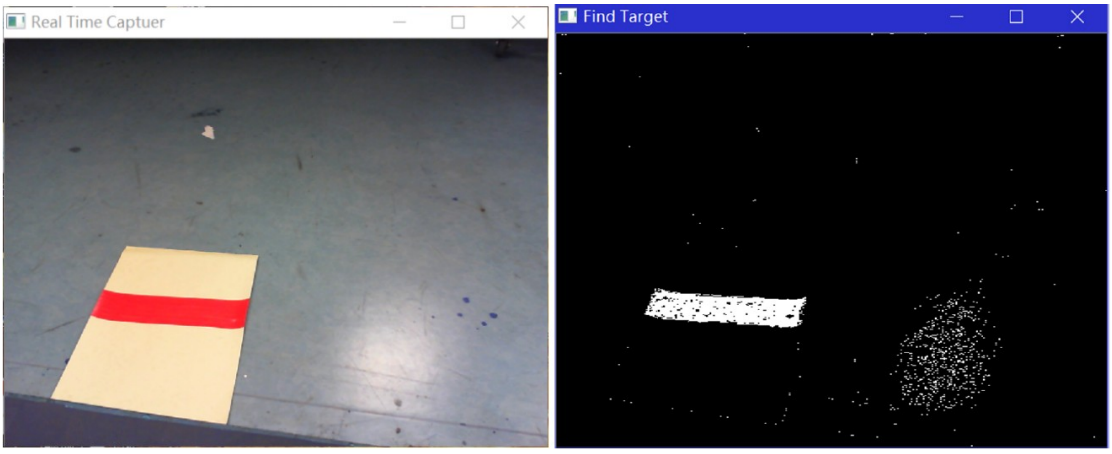


将 RGB 图像转换至 HSV 空间，对各目标色标模板 H 分量做 1D 直方图，在直方图中表示不同 H 分量值出现的概率或者像素个数，即为颜色概率查找表。将原始图像中每个像素值的值用颜色出现的概率做替换，得到颜色概率分布图，颜色概率分布图为灰度图像，该过程即为反向投影。

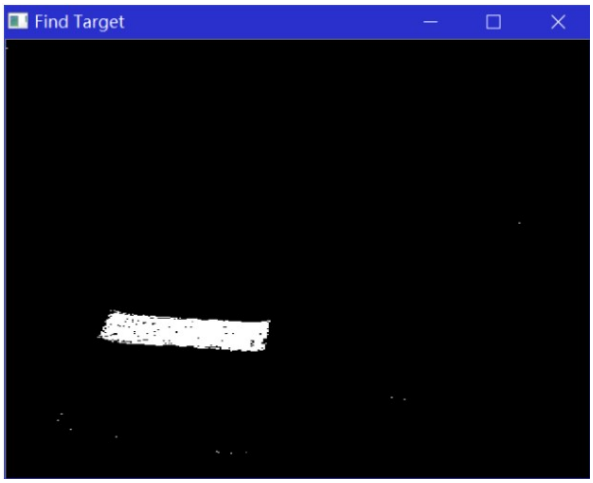
以下算法细节的改进能够使算法鲁棒性显著增强：

- 处理高饱和度像素

当像素饱和度偏高(光照较强)时，色相的区别会被大大削弱(各个色相都向“白色”偏移)，因此忽略高饱和度像素可以使颜色匹配的准确度大大提高。例如下图：地板的反光使得颜色匹配出现部分错误的结果



当进行高饱和度抑制处理后，匹配效果得到明显改善



➢ 形态学运算

直方图反投影的结果是二值图像，由于噪声影响，二值图中会出现噪点(微小的白色区域)。使用形态学闭运算可以将这些噪点去除，使得后续目标追踪更加准确。

(2) Meanshift

Camshift 是在 Meanshift 的基础上改进而来的。首先介绍 Meanshift 算法

Meanshift 算法是一种基于密度函数梯度估计的非参数方法，通过迭代寻找概率分布的极值来定位目标。算法过程为：

a) 随机在颜色概率分布图中选取搜索窗 W

b) 计算零阶、一阶距

$$M_{00} = \sum_x \sum_y I(\mathrm{x},\mathrm{y})$$

$$M_{10} = \sum_x \sum_y xI(\mathrm{x},\mathrm{y})$$

$$\mathrm{M}_{01} = \sum_x \sum_y yI(\mathrm{x},\mathrm{y})$$

c) 计算搜索窗的质心，调整搜索窗大小

$$X_c = \frac{M_{10}}{M_{00}}$$

$$Y_c = \frac{M_{01}}{M_{00}}$$

$$S = \sqrt{\frac{M_{00}}{256}}$$

d) 移动搜索窗的中心到质心，如果移动距离大于预设的固定阈值，则重复前三步，直到搜索窗的中心与质心间的移动小于预设的固定阈值或者循环迭代次数达到最大值，则停止计算。

将 Meanshift 算法扩展到连续图像序列，对视频所有帧均做 Meanshift 运算，并将上一帧结果，即为搜索窗的大小和中心，作为下一帧 Meanshift 算法的初始值，如此迭代实现对目标跟踪，算法过程：

a） 初始化搜索窗；

b） 计算搜索窗的颜色分配概率（反向投影）；

c） 运行 Meanshift 算法，获取搜索窗新的大小和位置；

d） 在下一帧视频图像中用 3 的值重新初始化搜索窗的大小和位置，在跳转到 2 继续进行。

目标跟踪识别框架如图 3-3 所示。

Camshift 特点：

a） 有效解决目标变形和遮挡问题。

b） 对系统资源要求不高。

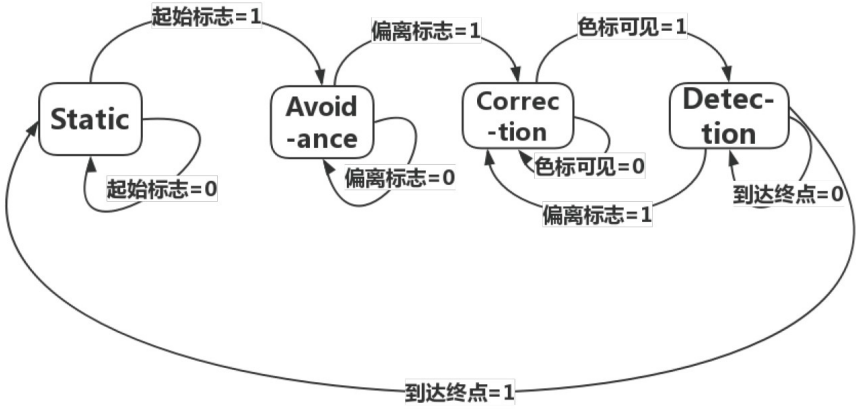
c） 时间复杂度低，在简单背景下能够取得良好的跟踪效果。

d） 忽略了目标的空间分布特性，背景复杂有干扰的情况下，会导致跟踪失。

3.2 有限状态机建模

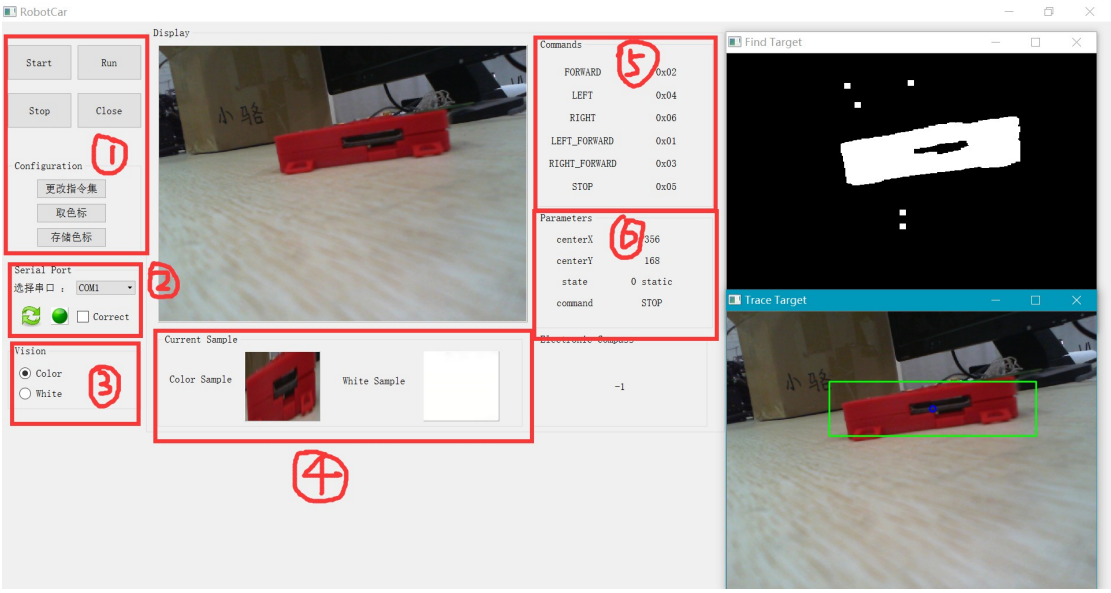
有限状态机（finite-state machine，FSM）又称有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。状态存储关于过去的信息，就是说：它反映从系统开始到现在时刻的输入变化。转移指示状态变更，并且用必须满足确使转移发生的条件来描述它。动作是在给定时刻要进行的活动的描述。

程序中一共定义了四种状态，Static(机器人静止)、Avoidance(机器人正在避障)、Correction(机器人修正自身水平位置)、Detection(探测终点)。状态转移条件由各个标志位组成，标志位的值在程序运行过程中动态更新。状态转移图如下：



4 程序运行效果及界面功能说明

4.1 主界面



- 区域①：用户控制区

用户操作系统的运行、停止，或者调出色标配置、指令集配置的子功能

- 区域②：串口配置区

自动检测并显示当前可用串口，显示当前串口连接状态

- 区域③：选择模式

如果目标是白色，选择“White”，否则选择“Color”。

- 区域④：色标预览

显示当前有效的色标；对应两种模式，色标也有两个

- 区域⑤：指令集预览

显示当前指令集；指令集可以由用户配置，详见子窗口介绍

- 区域⑥：状态预览区

检测并显示包括检测到的目标坐标、状态机当前状态等信息

- 图像显示区域

左侧是摄像头图像的实时预览；右侧是目标检测的结果

4.2 子界面

点击主界面中的“更改指令集”按钮即可调出子界面

在此界面中，用户可以更改不同指令对应的 16 进制指令码，方便与下位机指令系统向配合。

5 关键程序框架

5.1 基于直方图反投影和 **Camshift** 的目标追踪算法

```
//识别彩色
if (ui->radioButton_color->isChecked())
{
    ColorHistogram hc;
    ContentFinder finder;
    image_detect=image.clone();
    hc.getHueHistogram(ROI_Color,minSaturation);
    cv::cvtColor(image_detect,image_detect,CV_BGR2HSV);
    cv::split(image_detect,v);  //v[1]是饱和度分量
    cv::threshold(v[1],v[1],minSaturation,255,cv::THRESH_BINARY);
    finder.setHistogram(hc.hist);
    finder.setThreshold(finder_th);
    finder.findHueContent(image_detect,0.0f,180.0f,1);
    cv::bitwise_and(finder.result,v[1],finder.result); //去除低饱和度像素
    //形态学运算
```

```
    cv::erode(finder.result,finder.result,cv::Mat());
    cv::dilate(finder.result,finder.result,element2);
    //初始化矩形框
    if (!flag_init)
    {
       image_detect=finder.result.clone();
       cv::findContours(image_detect,contours,
                 CV_RETR_EXTERNAL,
                 CV_CHAIN_APPROX_NONE);
       boundRect=std::vector<cv::Rect>(contours.size());
       for(uint i=0;i<contours.size();i++)
       boundRect[i]=cv::boundingRect(cv::Mat(contours[i]));
       if(contours.size()!=0)
       {
          for(uint i=0,pos_max=0; i<contours.size(); i++)
          {
             if(boundRect[pos_max].area() <= boundRect[i].area())
             {
                pos_max=i;
                rect=boundRect[pos_max];
             }
          }
       }
       flag_init=true;
    }
    //CamShift 算法
    if (rect.area()>minArea)
    {
       image_detect=finder.result.clone();
       cv::CamShift(image_detect,rect,criteria);
    }
      else
         flag_init=false;
    image_detect=image.clone();
    if (rect.area()>minArea)
    {
        cv::rectangle(image_detect,rect,cv::Scalar(0,255,0),2); //绿色矩
形框
       cv::circle(image_detect,
             cv::Point(rect.x+rect.width/2,rect.y+rect.height/2),
             5,cv::Scalar(255,0,0),2);
       //更新参数
       centerX = rect.x+rect.width/2;
       centerY = rect.y+rect.height/2;
```

```
    }
    else
    {
        cv::rectangle(image_detect,rect,cv::Scalar(0,0,255),2);  //红色矩
形框
        cv::circle(image_detect,
              cv::Point(rect.x+rect.width/2,rect.y+rect.height/2),
              5,cv::Scalar(255,0,0),2);
        centerX = -1;
        centerY = -1;
    }
    //更新标识符
    if(!flag_init)
        flag_visible = false;
    else
        flag_visible = true;
```

5.2 状态转移函数

```
void RobotCar::UpdateState()
{
    switch(state)
    {
    case STATIC    :
        ui->label_state_2->setText("0 static");
        if(flag_run)
            state = AVOIDANCE;
        else
            state = STATIC;
        break;
    case AVOIDANCE :
        ui->label_state_2->setText("1 avoidance");
        if(flag_avoid)
        {
            if(flag_deviation==1)
            {
                SetCommand(RIGHT);
                state = CORRECTION;
            }
            else if(flag_deviation==2)
            {
                SetCommand(LEFT);
                state = CORRECTION;
            }
            else
```

```
            state = AVOIDANCE;
        }
        else
            state = AVOIDANCE;
        break;
    case CORRECTION:
        ui->label_state_2->setText("2 correction");
        if(flag_visible)
        {
            state = DETECTION;
            round = 0;
        }
        else
            state = CORRECTION;
        if (round == 30) //平移仍然找不到色标，可能是下位机发错信息了
        {
            if (command == RIGHT)
                SetCommand(LEFT);
            else
                SetCommand(RIGHT);
            state = CORRECTION;
        }
        round++;
        break;
    case DETECTION :
        ui->label_state_2->setText("3 detection");
             if(centerX == -1 || centerY == -1) //跑偏了，需要返回
CORRECTION 状态
        {
            flag_visible = false;
            round = 0;
            if(flag_deviation==1)
            {
                SetCommand(RIGHT);
                state = CORRECTION;
            }
            else if(flag_deviation==2)
            {
                SetCommand(LEFT);
                state = CORRECTION;
            }
            else
                state = AVOIDANCE;
        }
```

```
        else
        {
            if(flag_reach)
            {
                StopRun();
                state = STATIC;
            }
            else
            {
                //执行算法
                RunToEnd();
                state = DETECTION;
            }
        }
        break;
    }
}
```