哈尔滨工业大学（深圳）

数字图像处理课程设计

题　　目　基于广义霍夫变换的自主停泊系统

院　　(系)　机电工程与自动化学院

日　　期　2020.12.20

报 告 人　肖书奇　刘瑞铭

基于广义霍夫变换的自主停泊系统

目录

1 研究目标

1. 远距离时，利用 HSV 彩色分割实现对目标“H”的颜色识别；
2. 近距离时，利用广义霍夫变换完成对目标“H”的轮廓识别；
3. DashBot 自主运动到“H”后停止，完成停泊任务。

2 研究内容

2.1 颜色分割

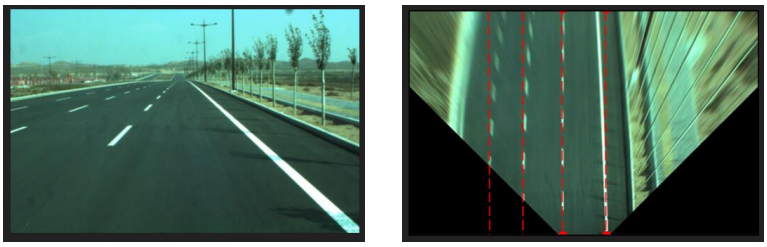
原理介绍 在图像处理中，最常用的颜色空间是 RGB 模型，常用于颜色显示和图像处理，其三维直角坐标的形式非常容易被理解。但是人眼对于这三种颜色分量的敏感程度是不一样的，在单色中，人眼对红色最不敏感，蓝色最敏感，所以 RGB 颜色空间是一种均匀性较差的颜色空间。如果颜色的相似性直接用欧氏距离来度量，其结果与人眼视觉会有较大的偏差。对于某一种颜色，我们很难推测出较为精确的三个分量数值来表示。且同一个物体在不同光照条件下，RGB 值可能差异很大。基于上述理由，在物体识别中使用较多的是 HSV 颜色空间，它比 RGB 更接近人们对彩色的感知经验，非常直观地表达颜色的色调、鲜艳程度和明暗程度，方便进行颜色的对比，且对外界光照要求更低。HSV 与 RGB 空间转换关系如下：

$$
\begin{aligned}
&\mathrm{V} \leftarrow \max(\mathrm{R}, \mathrm{G}, \mathrm{B}) \\
&\mathrm{S} \leftarrow \begin{cases} \dfrac{\mathrm{V} - \min(\mathrm{R}, \mathrm{G}, \mathrm{B})}{\mathrm{V}} & \text{if } \mathrm{V} \neq 0 \\ \text{otherwise} \end{cases} \\
&\mathrm{H} \leftarrow \begin{cases} 60(\mathrm{G} - \mathrm{B})/(\mathrm{V} - \min(\mathrm{R}, \mathrm{G}, \mathrm{B})) & \text{if } \mathrm{V} = \mathrm{R} \\ 120 + 60(\mathrm{B} - \mathrm{R})/(\mathrm{V} - \min(\mathrm{R}, \mathrm{G}, \mathrm{B})) & \text{if } \mathrm{V} = \mathrm{G} \\ 240 + 60(\mathrm{R} - \mathrm{G})/(\mathrm{V} - \min(\mathrm{R}, \mathrm{G}, \mathrm{B})) & \text{if } \mathrm{V} = \mathrm{B} \end{cases}
\end{aligned}
$$

(If $\mathrm{H} < 0$ then $\mathrm{H} \leftarrow \mathrm{H} + 360$. On output $0 \leq \mathrm{V} \leq 1, 0 \leq \mathrm{S} \leq 1, 0 \leq \mathrm{H} \leq 360$)

2.2 逆透视变换

本任务中机器人的 ZED 相机保持水平，目标“H”是地上的二维图案，这时接收到的图像会发生几何畸变。而随后的广义霍夫变换是基于轮廓的，所以必须预先对摄像头采集到的图像进行校正，得到鸟瞰视角。



(a) 原始图像　　(b) 变换后的图像

图 1: 逆透视变换举例

原理介绍　透射变换具有单应性，是三维空间的线性变换，有八个自由度。若知道变换前后四组对应点的坐标，即可求解出该变换对应的单应矩阵，从而实现几何畸变校正。

原本计划利用霍夫线检测得到图像中的消失点，自适应地完成逆透视变换，但时间有限，尚未实现。后仅基于试验数据，对该机器人的固定视角采取逆透视变换，尽管探测范围较窄（约 1.2 米），但可以满足后续广义霍夫变换的要求。

2.3　广义霍夫变换

广义霍夫变换是经典霍夫变换的推广，该算法可以在图像中识别出任意的非解析图案，并且可以识别该图案的缩放倍数和旋转角度。

原理介绍　广义霍夫变换主要分为两个步骤。第一步是基于模板图案，构造其参数空间 $\phi - table$；第二步是遍历原图像，利用 $\phi - table$ 选出最有可能是目标图案中心的点。

2.3.1　制作模板的 $\phi - table$

首先指定模板（提取轮廓后）的中心点，以其为原点建立极坐标系。从中心点扫描一周，记录采集到的点集的三项信息：该点处的梯度角 ϕ；该点到原点的距离 r；该点到原点的辐角 α。

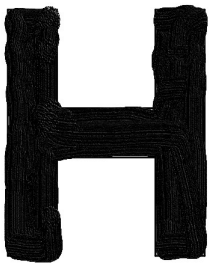


图 2: 鼠绘的 H 图案作为模板

ϕ	r	α
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮

$\phi - table$

2.3.2 投票

遍历原图像的所有点。首先判断该点处的梯度角是否在 $\phi - table$ 中的第一列里出现过——若从未出现过，则说明该点不可能在“H”的轮廓上，该点不具备投票资格。对于具有投票资格的点而言，先在 $\phi - table$ 中找到 ϕ 值匹配的 r 值与 α 值，之后将该点设定为极坐标系原点，将 r, α 代入极坐标公式，便可以恢复出“该点认为的图案中心”的位置。遍历完成之后，在“票数累加器”中找到最大值的位置，即为原图像中最接近模板之处。

若在投票过程中引入缩放因数 s 与旋转角度 θ，即可处理缩放与旋转识别。数学表述如下：

$$\begin{cases} x_c = x_i + r_k^i s\cos(\alpha_k^i + \theta) \\ y_c = y_i + r_k^i s\sin(\alpha_k^i + \theta) \end{cases} \quad , \quad A[x_c, y_c, s, \theta] + +$$

2.4 机器人运动

调用`roscpp`中`geometry_msgs::Twist`发布机器人的速度信息。

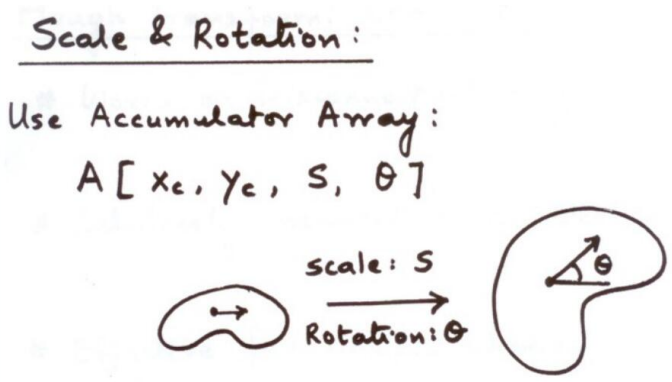


图 3: CMU《计算机视觉》讲义截图

3　方案设计

3.1　远距离识别：颜色分割

1. 色度空间转换
   `cv::cvtColor(...);`
2. 目标颜色分割并进行二值化
   `cv::inRange(...);`
3. 进行高斯滤波
   `cv::GaussianBlur(...);`
4. 轮廓提取，目标图像和模板进行匹配
   ```
   cv::findContours(...);
   double cv::cvMatchShapes(...);
   ```
5. 用最小矩形框出轮廓，利用矩形中心求目标点
   ```
   cv::minAreaRect(...);
   std::vector<Point2f> center_point;
   center_point.push_back(minRect[i].center);
   ```

3.2　近距离识别：逆透视变换 + 广义霍夫变换

离线

1. 获取单应矩阵
   ```
   Mat InversePerspectiveMapping(const int (*src_points)[2],
   const int (*dst_points)[2]);
   ```

2. 预处理后的模板图片

   ```
   Mat PreprocessTemplate(Mat img_origin);
   ```

3. 模板

   ```
   int ConstructRTable(Mat img_origin, float (*r_table)[3]);
   ```

在线

1. 逆透视变换

   ```
   cv::warpPerspective(Mat src,Mat dst,Mat m_ipm, cv::Size(src
   .cols,src.rows));
   ```

2. 广义霍夫变换完成目标识别

   ```
   int GeneralHoughTransform(Mat img_origin, float (*r_table)
   [3], int row_num_r_table, int& x_target, int& y_target);
   ```

3. 机器人运动

   ```
   void Navigate(int x_target, int y_target, int cols, int rows
   , int poll, bool& flag, float& linear_vel, float& angular_vel
   );
   ```

4. 停泊

   ```
   if (flag == 1)...
   ```

3.3 机器人运动

速度控制 角速度采用 P 控制，偏差为目标与画面中轴线的距离。

线速度采用分段的开关控制，设定了两个阈值。若目标的票数（可信度）低于阈值一，机器人原地不动；若目标的票数介于阈值一、二之间，机器人缓慢前进；若目标的票数高于阈值二，机器人快速前进。

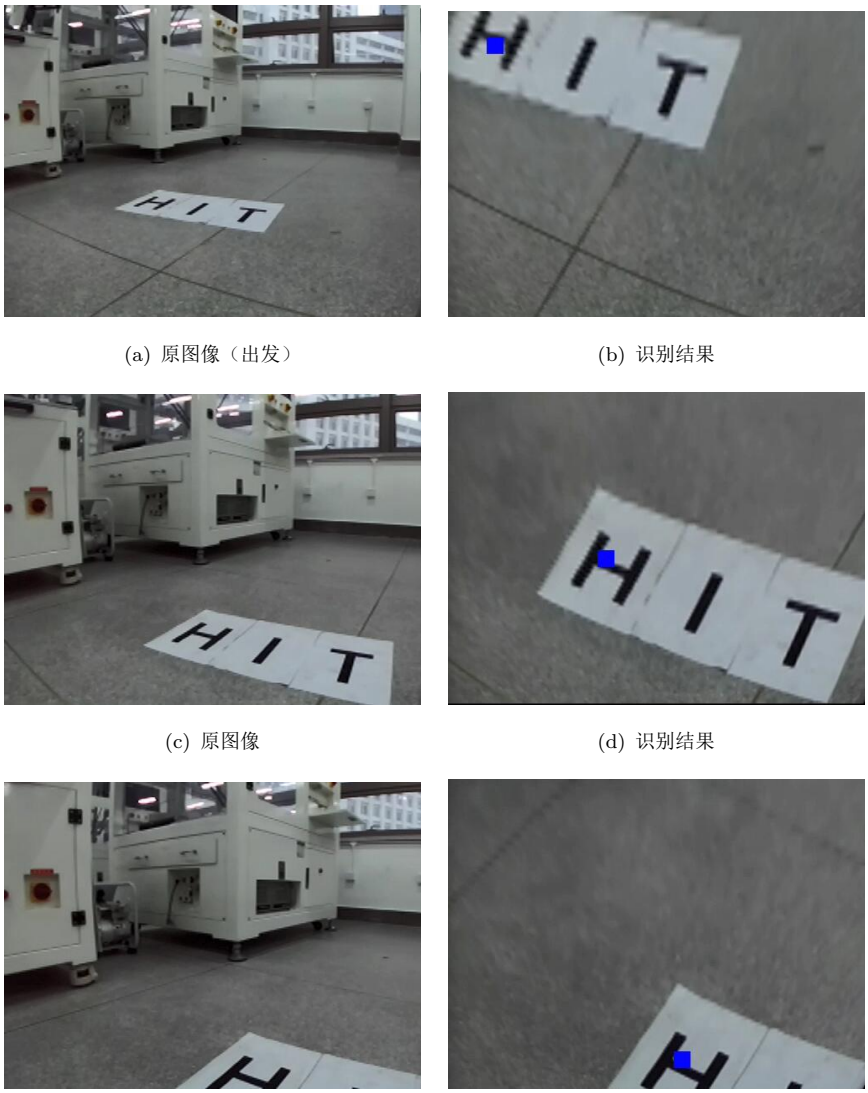
停止条件 同时满足如下三个条件后，机器人运行 1 秒后停车。

- 目标得到票数足够大。（足够可信)
- 目标与画面中轴线距离足够小。（角度正确）
- 目标足够靠近画面的下沿。（距离足够近）

4 实验验证

成功完成了“远距离且无强光、颜色干扰时利用颜色识别目标”;“近距离采用精确的广义霍夫变换识别目标”;“移动到目标位置后自动停止”三个目标。

图 4: 利用颜色识别到达目标点



(a) 原图像（出发） (b) 识别结果

(c) 原图像 (d) 识别结果

(e) 原图像（触发停车指令） (f) 识别结果

图 5: 逆透视变换 + 广义霍夫变换的运动过程截图

5 结论

基于 Dashbot 移动机器人平台，实现了 HSV 颜色分割、逆透视变换、广义霍夫变换、移动机器人的运动与停泊策略，实现了预期目标。

5.1 优缺点

优点

- 因地制宜，准备了两种方案；
- 颜色识别准确且快速；
- 广义霍夫变换的鲁棒性好；（尽管模板是鼠标绘制的，比较粗糙，但可以用来识别各种“H”。）
- 广义霍夫变换的灵敏度高；（调高灵敏度后，甚至可以把眼镜识别为横着放的“H”。）
- 广义霍夫变换可以处理遮盖问题。（盖上目标的一部分，仍能有效识别。）

缺点

- 颜色识别对外界环境要求高；
- 固定视角下的逆透视变换导致视角变小；
- 广义霍夫变换的计算复杂度高；
- 广义霍夫变换具体参数、阈值的设定只能凭借经验。

5.2 有待改进之处

- 使用改进后的颜色分割算法，从而降低对环境的要求；
- 双目结合，扩大视野，估计目标深度；
- 使用自适应的逆透视变换算法；
- 优化广义霍夫变换的计算复杂度；
- 优化机器人的控制方法；规划机器人的运动路线。

6 参考文献

[1] Gonzalez, Rafael C., Richard Eugene Woods, and Steven L. Eddins. Digital image processing using MATLAB. Pearson Education India, 2004.

[2] Ioannis (Yannis) Gkioulekas, 16-385 Computer Vision. Carnegie Mellon University, Spring 2020.

7　附录

7.1　逆透视变换 + 广义霍夫变换程序代码

```
/* Project: Self-parking System Based on Color Segmentation and General Hough Transformation
*
* Author: Shuqi Xiao, Ruiming Liu
*
* Date: 2020.12.12
*/

#include <stdlib.h>
#include <iostream>
#include <string>
#include <math.h>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Bool.h"
#include "std_msgs/Float32.h"
#include <geometry_msgs/Twist.h>
#define PI 3.1415

using namespace std;
using namespace cv;

// 图像基础尺寸
const int kRowNumLowResImg = 200;            //纵坐标最大值
const int kColNumLowResImg = 200;            //横坐标最大值

// 文件路径
// const string kFilenameTest = "/home/zdh/xsq_ws/img/input/test_alphabet.jpg";
// const string kFilenameOutput = "/home/zdh/xsq_ws/img/output";

Mat HsvSegment(Mat img_origin)
{
  Mat img_hsv;
  cvtColor(img_origin, img_hsv, COLOR_BGR2HSV);
  // Green
  // inRange(img_hsv, Scalar(35, 43, 46), Scalar(77, 255, 255), img_hsv);
  // Black
  inRange(img_hsv, Scalar(0, 0, 0), Scalar(180, 255, 46), img_hsv);
  return img_hsv;
}

Mat PreprocessTemplate(Mat img_origin)
{
  /* 功能：广义霍夫变换的图片预处理
  * 输入：
  * 输出：预处理后的图像
  *操作：低分辨率处理；彩色转灰度；高斯滤波；Canny 边沿检测
  */
  // 图像预处理采用的参数
  const float kGaussianKernSigma = 1.5;      //高斯卷积核方差
  const float kGaussianKernSize = 5;         //高斯卷积核尺寸
  const float kCannyThresh1 = 70;            //Canny双阈值处理
  const float kCannyThresh2 = 150;           //Canny双阈值处理

  Mat img_low_res;
  resize(img_origin,img_low_res,cv::Size(kColNumLowResImg, kRowNumLowResImg),CV_INTER_AREA);
  Mat img_gray;
  cvtColor(img_low_res, img_gray, CV_BGR2GRAY);
  Mat img_gaussian_blur;
```

```
63    GaussianBlur(img_gray, img_gaussian_blur, cv::Size(kGaussianKernSize,kGaussianKernSize),
          kGaussianKernSigma);
64    Mat img_canny;
65    Canny(img_gaussian_blur, img_canny, kCannyThresh1, kCannyThresh2, 3, false);
66    imshow("Canny",img_canny);
67    return img_canny;
68  }
69
70  Mat PreprocessImg(Mat img_origin)
71  {
72    /* 功能: 广义霍夫变换的图片预处理
73    * 输入:
74    * 输出: 预处理后的图像
75    * 操作: 低分辨率处理; 彩色转灰度; 高斯滤波; Canny 边沿检测
76      */
77    // 图像预处理采用的参数
78    const float kGaussianKernSigma = 1.5;       //高斯卷积核方差
79    const float kGaussianKernSize = 5;          //高斯卷积核尺寸
80    const float kCannyThresh1 = 70;             //Canny双阈值处理
81    const float kCannyThresh2 = 150;            //Canny双阈值处理
82
83    Mat img_low_res;
84    resize(img_origin,img_low_res,cv::Size(kColNumLowResImg, kRowNumLowResImg),CV_INTER_AREA);
85    Mat img_gaussian_blur;
86    GaussianBlur(img_low_res, img_gaussian_blur, cv::Size(kGaussianKernSize,kGaussianKernSize),
          kGaussianKernSigma);
87    Mat img_canny;
88    Canny(img_gaussian_blur, img_canny, kCannyThresh1, kCannyThresh2, 3, false);
89    imshow("Canny",img_canny);
90    return img_canny;
91  }
92
93  Mat InversePerspectiveMapping(const int (*src_points)[2], const int (*dst_points)[2])
94  {
95    /* 功能: 利用单次逆透射变换完成相机的几何畸变校正
96    * 输入: 参考四边形顶点坐标（二维数组行指针）; 变换后矩形四顶点坐标（二维数组行指针）
97    * 输出: 单应矩阵
98    */
99    vector<Point2f> p,q;
100
101   p.push_back(Point2f(src_points[0][0],src_points[0][1]));
102   q.push_back(Point2f(dst_points[0][0],dst_points[0][1]));
103
104   p.push_back(Point2f(src_points[1][0],src_points[1][1]));
105   q.push_back(Point2f(dst_points[1][0],dst_points[1][1]));
106
107   p.push_back(Point2f(src_points[2][0],src_points[2][1]));
108   q.push_back(Point2f(dst_points[2][0],dst_points[2][1]));
109
110   p.push_back(Point2f(src_points[3][0],src_points[3][1]));
111   q.push_back(Point2f(dst_points[3][0],dst_points[3][1]));
112
113   Mat matrix_perspective_transform = getPerspectiveTransform(p,q);
114
115   return matrix_perspective_transform;
116 }
117
118 int ConstructRTable(Mat img_origin, float (*r_table)[3])
119 {
120   /* 功能: 建立模板的r_table
121   * 输入: 模板图片; r_table二维数组的行指针
122   * 输出: r_table的行数
123   */
124   // 广义霍夫变换之r-table的采样率与阈值
125   const float kAngleSamplingNum = 50;        //角采样率
126   const int kThreshTemplateGrayScale = 100;  //灰度阈值
127   // Preprocess
128   Mat img_after_preprocessing;
129   img_after_preprocessing = PreprocessTemplate(img_origin);
```

```
130   // Gradient
131   Mat grad_x, grad_y, grad_mag, grad_ang;                                    //x方向梯度，y方向梯度，梯度
          模长，梯度角度
132   Sobel(img_after_preprocessing, grad_x, CV_32FC1, 1, 0); //x方向梯度
133   Sobel(img_after_preprocessing, grad_y, CV_32FC1, 0, 1); //y方向梯度
134   magnitude(grad_x, grad_y, grad_mag);            //梯度幅值
135   phase(grad_x, grad_y, grad_ang, false);         //梯度角（使用弧度制，若true则为角度制
136     // xy-table
137   int row_num_r_table = 0;
138   float xy_table[kRowNumLowResImg*kColNumLowResImg][3];
139   int x,y = 0;                                    //循环变量，作为横纵坐标
140   int y_reference_point = ceil(0.5 * kRowNumLowResImg); //模板参考点y坐标（中心点）
141   int x_reference_point = ceil(0.5 * kColNumLowResImg); //模板参考点x坐标（中心点）
142   for (int i = 0; i < kAngleSamplingNum; i++)   //遍历所有角度
143   {
144     for (int x = 0; x < kColNumLowResImg; x++)//遍历x坐标
145     {
146       // 求得该直线上对应y坐标的位置
147       y = ceil(tan(i*(2*PI)/kAngleSamplingNum)*(x-x_reference_point)+y_reference_point);
148       // 直线不能越出图片边界，忽略强度较小的边沿
149       if (y > 0 && y < kRowNumLowResImg && img_after_preprocessing.at<uchar>(y,x) >
              kThreshTemplateGrayScale)
150       {
151         xy_table[row_num_r_table][0] = i*(2*PI)/kAngleSamplingNum;
152         xy_table[row_num_r_table][1] = x;
153         xy_table[row_num_r_table][2] = y;
154         row_num_r_table ++;
155       }
156     }
157   }
158   row_num_r_table++;//得到r-table的行数（模板精细程度的指标，基本由“图像基础尺寸”与“角采样数”决定）
159   // r-table
160   for (int i = 0; i < row_num_r_table; i++)
161   {
162     r_table[i][0] = grad_ang.at<float>(xy_table[i][2],xy_table[i][1]);
163     r_table[i][1] = sqrt(pow((xy_table[i][1]-x_reference_point),2) + pow((xy_table[i][2]-
            y_reference_point),2) );
164     r_table[i][2] = xy_table[i][0]+PI;
165   }
166
167   return row_num_r_table;
168 }
169
170 int GeneralHoughTransform(Mat img_origin, float (*r_table)[3], int row_num_r_table, int& x_target, int&
        y_target)
171 {
172   /* 功能：利用广义霍夫变换完成物体识别
173   * 输入：待识别图片；模板的r-table；r_table数组的行数；目标x坐标的地址；目标y坐标的地址
174   * 输出：最高票数
175   */
176
177   // 广义霍夫变换之模板匹配
178   const float kThreshGradAngleDiff = 0.01;    //模板与目标图像梯度角的误差最大值
179   const float kMinZoomFactor = 0.3;           //最小缩放倍数
180   const float kMaxZoomFactor = 1.2;           //最大缩放倍数
181   const int kScaleSamplingNum = 3;            //尺寸采样数
182   const int kRotationSamplingNum = 2;         //旋转采样数
183   const int kRotationSymmetry = 2;            //H旋转180°后与自身重合，利用此对称性减少不必要的计算
184   // Preprocess
185   Mat img_after_preprocessing;
186   img_after_preprocessing = PreprocessImg(img_origin);
187   // Gradient
188   Mat grad_x, grad_y, grad_mag, grad_ang;                                    //x方向梯度，y方向梯度，梯度模
          长，梯度角度
189   Sobel(img_after_preprocessing, grad_x, CV_32FC1, 1, 0); // x方向梯度
190   Sobel(img_after_preprocessing, grad_y, CV_32FC1, 0, 1); // y方向梯度
191   magnitude(grad_x, grad_y, grad_mag);      //梯度幅值
192   phase(grad_x, grad_y, grad_ang, false);   //梯度角（使用弧度制，若true则为角度制
193   // Voting
```

```
194  int idx_scale, idx_rotation = 0;//循环变量，遍历模板的尺度与旋转
195  float zoom_factor, angle = 0;//对应的缩放倍数与旋转角度
196  int i,j = 0;//选民的横纵坐标
197  int x,y = 0;//候选人的横、纵坐标
198  int counter[kScaleSamplingNum][kRotationSamplingNum][kRowNumLowResImg][kColNumLowResImg] = {0};// 计数
       器
199  int maximum_counter = 0;                                                    //最大计数
200  float zoom_factor_maximum_counter = 0;
201  float angle_maximum_counter = 0;
202  int x_maximum_counter = 0;                                  //低分辨率图像中目标的横坐标
203  int y_maximum_counter = 0;                                  //低分辨率图像中目标的纵坐标
204    for (int idx_scale = 0; idx_scale < kScaleSamplingNum; idx_scale++)
205  {
206    zoom_factor = kMinZoomFactor + idx_scale*(kMaxZoomFactor - kMinZoomFactor)/kScaleSamplingNum;
207    for (int idx_rotation = 0; idx_rotation < kRotationSamplingNum; idx_rotation++)
208    {
209      angle = idx_rotation*2*PI/kRotationSymmetry/kRotationSamplingNum; // H is symmetric, so the
           coefficient of PI is one.
210      for (int j = 0; j < kRowNumLowResImg; j++)
211      {
212        for (int i = 0; i < kColNumLowResImg; i++)
213        {
214          for (int k = 0; k < row_num_r_table; k++)
215          {
216            //判断(i,j)像素是否有投票资格
217            //1. (i,j)像素的灰度值不为0（预筛选，提速）
218            //2. (i,j)像素处的梯度角在r-table名单第一列中出现过（细筛选）
219            if (img_after_preprocessing.at<uchar>(j,i) != 0 && abs(grad_ang.at<float>(j,i) - r_table[k
                 ][0]) < kThreshGradAngleDiff)
220            {
221              //计算(i,j)像素的投票对象(x,y)
222              x = ceil(i+r_table[k][1]*zoom_factor*cos(r_table[k][2]+angle));
223              y = ceil(j+r_table[k][1]*zoom_factor*sin(r_table[k][2]+angle));
224              //确保(x,y)不能越界
225              if (x >= 0 && x < kColNumLowResImg && y >= 0 && y < kRowNumLowResImg)
226              {
227                  counter[idx_scale][idx_rotation][y][x]++;
228              }
229            }
230          }
231        }
232      }
233    }
234  }
235
236  // Sifting
237    // 利用 std::max_element 找到 counter 最大值
238  maximum_counter = *std::max_element(&counter[0][0][0][0], &counter[0][0][0][0]+kRowNumLowResImg*
       kColNumLowResImg*kScaleSamplingNum*kRotationSamplingNum);
239    // 四层循环找到最大值，找到即退出循环。（有待优化）
240    bool flag = false;
241    for (int idx_scale = 0; idx_scale < kScaleSamplingNum && flag == false; idx_scale++)
242    {
243      for (int idx_rotation = 0; idx_rotation < kRotationSamplingNum && flag == false; idx_rotation++)
244      {
245        for (int y = 0; y < kRowNumLowResImg && flag == false; y++)
246        {
247          for (int x = 0; x < kColNumLowResImg && flag == false; x++)
248          {
249            if (counter[idx_scale][idx_rotation][y][x] == maximum_counter)
250            {
251              zoom_factor_maximum_counter = kMinZoomFactor + idx_scale*(kMaxZoomFactor - kMinZoomFactor)
                     /kScaleSamplingNum;
252              angle_maximum_counter = 360/kRotationSymmetry*idx_rotation/kRotationSamplingNum;
253              y_maximum_counter = y;
254              x_maximum_counter = x;
255              flag = true;
256            }
257          }
```

```
258             }
259           }
260         }
261
262         // 得到原图中的目标点位置
263       x_target = round(x_maximum_counter*img_origin.size().width/kColNumLowResImg);
264       y_target = round(y_maximum_counter*img_origin.size().height/kRowNumLowResImg);
265       return maximum_counter;
266     }
267
268
269     Mat Mark(Mat img_origin, int x_target, int y_target)
270     {
271       /* 功能: 在给定像素位置标记彩色方块, 便于展示
272       * 输入: 待标记图片; 目标横坐标的值; 目标纵坐标的值
273       * 输出: 标记后的图片
274       */
275
276       const int kSizeDisplayBlock = 10;           // 输出图像上目标点处的正方形像素块的大小
277       Mat img_target_display = img_origin.clone();
278       //越界检查
279       if (x_target > kSizeDisplayBlock
280         && x_target + kSizeDisplayBlock < img_target_display.size().width
281         && y_target > kSizeDisplayBlock
282         && y_target + kSizeDisplayBlock < img_target_display.size().height)
283       {
284         for (int j = y_target - kSizeDisplayBlock; j < y_target + kSizeDisplayBlock; j++)
285         {
286           for (int i = x_target - kSizeDisplayBlock; i < x_target + kSizeDisplayBlock; i++)
287           {
288             img_target_display.at<Vec3b>(j,i)[0] = 255;
289             img_target_display.at<Vec3b>(j,i)[1] = 0;
290             img_target_display.at<Vec3b>(j,i)[2] = 0;
291           }
292         }
293       }
294       else
295       {
296         cout<<"Target is on the edge of the detection zone."<<endl;
297       }
298       return img_target_display;
299     }
300
301     void Navigate(int x_target, int y_target, int cols, int rows, int poll, bool& flag, float& linear_vel,
          float& angular_vel)
302     {
303       // 速度范围
304       const float kMinAngularVel = 0.1;
305       const float kMaxAngularVel = 0.25;
306       const float kMinLinearVel = 0.1;
307       const float kMaxLinearVel = 0.3;
308       // 线速度开关控制参数
309       const int kPollThresh1 = 10;// 轻视票数低于此阈值的目标
310       const int kPollThresh2 = 30;// 重视票数高于此阈值的目标
311       // 角速度P控制参数
312       const float kProportion = 0.0020;
313       // 中轴像素点范围
314       const int kCenterRange = 200;
315
316       int error = cols/2-x_target;  // 偏差
317
318       // 角速度: P控制, 偏差为目标到画面中轴线的距离
319
320       angular_vel = kProportion*error;
321       if (abs(angular_vel) > kMaxAngularVel)
322       {
323         if(angular_vel >= 0)     angular_vel = kMaxAngularVel;
324         if(angular_vel < 0)     angular_vel = -kMaxAngularVel;
```

```
325     }
326     if (abs(angular_vel) < kMinAngularVel)
327     {
328       if(angular_vel >= 0)     angular_vel = kMinAngularVel;
329       if(angular_vel < 0)     angular_vel = -kMinAngularVel;
330     }
331
332     // 线速度: 开关控制
333     if ( abs(error) <= kCenterRange )
334     {
335       if (poll > kPollThresh1 && poll < kPollThresh2)
336       {
337         linear_vel = kMinAngularVel;
338       }
339       if (poll >= kPollThresh2)
340       {
341         linear_vel = kMaxLinearVel;
342       }
343     }
344
345     if (rows - y_target < 0.1*rows && poll > kPollThresh1)
346     {
347       flag = 1;
348       angular_vel = 0;
349       linear_vel = 0.1;
350     }
351 }
352
353
354 int main(int argc, char **argv)
355 {
356     // 模板图片的路径
357     const string kFilenameTemplate = "/home/zdh/xsq_ws/img/template/template.jpg";
358     // 打开Dashbot摄像头为1, 打开笔记本摄像头为0
359     // const int kCamera = 0;
360     // 几何畸变校正的参考四边形坐标, (x,y)左上、右上、左下、右下
361     const int kQuadrangle[4][2] = {
362     {161,231},
363     {496,234},
364     {0,375},
365     {670,375}};
366     // 映射后的矩形坐标
367     const int kFrame[4][2] = {
368     {0,0},
369     {670,0},
370     {0,370},
371     {670,370}};
372
373     // 初始化
374     Mat img_template, img_origin, m_ipm, img_ipm, img_ipm_target_display;
375     float r_table[kRowNumLowResImg*kColNumLowResImg][3] = {0};
376     int maximum_counter, x_target_ipm, y_target_ipm = 0;
377     // 读取模板图片
378     img_template = imread(kFilenameTemplate);
379     // 建立模板的r_table
380     int row_num_r_table = ConstructRTable(img_template, r_table);
381     // 得到单应矩阵
382     m_ipm = InversePerspectiveMapping(kQuadrangle,kFrame);
383     // cout << "matrix_perspective_transform:  "<<endl << format(m_ipm, Formatter::FMT_C) << endl << endl;
384
385     // 初始化ROS节点
386     ROS_WARN("**********START**********");
387     ros::init(argc,argv,"self_parking");
388     ros::NodeHandle self_parking;          // 节点句柄
389     ros::Publisher pub=self_parking.advertise<geometry_msgs::Twist>("/smoother_cmd_vel", 5);
390     // 打开摄像头
391     VideoCapture capture;
392     capture.open(1);
```

```
393   waitKey(100);
394   if(!capture.isOpened())        // 摄像头异常处理
395   {
396     printf("摄像头图像读取失败。\n");
397     return 0;
398   }
399   // 控制量
400   float linear_vel = 0;
401   float angular_vel = 0;
402   // 停车标志：当flag == 1时，认为已经来到了目标面前。
403   bool flag = 0;
404   // 主循环
405   while(ros::ok())
406   {
407     // 截取双目摄像头的一目
408     capture.read(img_origin);
409     img_origin = img_origin(Range(0,img_origin.rows),Range(0,img_origin.cols/2-170));
410     imshow("src",img_origin);
411     // 颜色分割（非必要）
412     Mat img_hsv = HsvSegment(img_origin);
413     imshow("img_hsv",img_hsv);
414     // 逆透射变换
415     warpPerspective(img_hsv, img_ipm, m_ipm, cv::Size(img_origin.cols,img_origin.rows));
416     warpPerspective(img_origin, img_ipm_target_display, m_ipm, cv::Size(img_origin.cols,img_origin.rows)
            );
417     // 广义霍夫变换
418     maximum_counter = GeneralHoughTransform(img_ipm, r_table, row_num_r_table, x_target_ipm,
            y_target_ipm);
419     cout<<"maximum_counter = "<<maximum_counter<<endl;
420     // 展示识别结果
421     img_ipm_target_display = Mark(img_ipm_target_display, x_target_ipm, y_target_ipm);
422     imshow("Target Recognition: After IPM", img_ipm_target_display);
423     // 导航
424     Navigate(x_target_ipm, y_target_ipm, img_ipm.cols, img_ipm.rows, maximum_counter, flag, linear_vel,
            angular_vel);
425     cout<<"linear_vel = "<<linear_vel<<"     "<<"angular_vel"<<"     "<<angular_vel<<endl;
426     // 发布速度信息
427     ros::Publisher pub=self_parking.advertise<geometry_msgs::Twist>("/smoother_cmd_vel", 5);
428     geometry_msgs::Twist cmd_navigate;
429     cmd_navigate.linear.x = linear_vel;
430     cmd_navigate.linear.y = 0;
431     cmd_navigate.linear.z = 0;
432     cmd_navigate.angular.x = 0;
433     cmd_navigate.angular.y = 0;
434     cmd_navigate.angular.z = angular_vel;
435     pub.publish(cmd_navigate);
436     // 停车
437     if (flag == 1)
438     {
439       for (int i = 0; i < 200000; i++)
440       {
441         pub.publish(cmd_navigate);
442       }
443       break;
444     }
445
446     ros::spinOnce();
447     waitKey(5);
448   }
449   return 0;
450 }
```

7.2　颜色分割程序代码

```
#include <stdlib.h>
#include <cv.h>
#include <highgui.h>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Bool.h"
#include "std_msgs/Float32.h"
#include <geometry_msgs/Twist.h>
#include <vector>
#define PI 3.1415
#define zero 0.000001

using namespace std;
using namespace cv;

void Navigate(int x_target, int y_target, int cols, int rows, float& linear_vel, float& angular_vel)
{
  // 速度范围
  const float kMinAngularVel = 0.1;
  const float kMaxAngularVel = 0.3;
  const float kMinLinearVel = 0.1;
  const float kMaxLinearVel = 0.3;
  // 线速度开关控制参数
  const int kPollThresh1 = 10; // 轻视景数低于此阈值的目标
  const int kPollThresh2 = 30;// 重视景数高于此阈值的目标
  // 角速度PI控制参数
  const float kProportion = 0.0025;
  const float kIntegral = 0;
  int error = cols/2-x_target;  // 偏差

  linear_vel = 0.1;

  // 角速度: P控制, 偏差为目标到画面中轴线的距离
  angular_vel = kProportion*error;
  if (abs(angular_vel) > kMaxAngularVel)
  {
    if(angular_vel >= 0)    angular_vel = kMaxAngularVel;
    if(angular_vel < 0)    angular_vel = -kMaxAngularVel;
  }
  if (abs(angular_vel) < kMinAngularVel)
  {
    if(angular_vel >= 0)    angular_vel = kMinAngularVel;
    if(angular_vel < 0)    angular_vel = -kMinAngularVel;
  }
}


int main(int argc, char **argv)
{
  ROS_WARN("**********START**********");
  ros::init(argc,argv, "trafficLaneTrack"); //初始化ROS节点
  ros::NodeHandle n;

  VideoCapture capture;
  //capture.open(1);     //打开zed相机
  capture.open(1);

  waitKey(100);
  if(!capture.isOpened())
  {
    printf("摄像头没有正常打开, 请重新插拔\n");
    return 0;
  }
int frameWidth = capture.get(CV_CAP_PROP_FRAME_WIDTH);//图片宽
    int frameHeight = capture.get(CV_CAP_PROP_FRAME_HEIGHT);//图片高
```

```
68      int lowh = 26;        //目标颜色阈值
69      int lows = 44;
70      int lowv = 46;
71      int highh = 34;
72      int highs = 255;
73      int highv = 255;
74      static int b = 0;
75  #ifndef READIMAGE_ONLY
76    ros::Publisher pub=n.advertise<geometry_msgs::Twist>("/smoother_cmd_vel", 5);//定义dashgo机器人的速度
          发布器
77  #endif
78    Mat src_frame;    //目标图像
79    Mat model=imread("/home/lruiming/h_red.png");      //模板图像
80    while(ros::ok())
81    {
82      capture.read(src_frame);
83
84      src_frame = src_frame(Range(0,src_frame.rows),Range(src_frame.cols/2,src_frame.cols));
85      if (src_frame.empty())
86      {
87        break;
88      }
89          imshow("origin", src_frame);
90    imshow("origin_1", model);
91    /*Mat blur1;
92    cv::GaussianBlur(src_frame,blur1, cv::Size(5, 5), 3, 3);
93    imshow("GaussianBlur", blur1);*/
94    Mat src_frame_HSV;    //RGB空间转HSV空间
95    Mat model_HSV;
96    //cvtColor(blur1,src_frame_HSV,COLOR_BGR2HSV);
97    cvtColor(src_frame,src_frame_HSV,COLOR_BGR2HSV);
98    cvtColor(model,model_HSV,COLOR_BGR2HSV);
99    //Mat img;//二值图
100   /*inRange(src_frame_HSV,Scalar(lowh,lows,lowv),Scalar(highh,highs,highv),img);
101   imshow("twocount",img);*/
102   /*Mat mask1, mask2;//红色
103       inRange(src_frame_HSV, Scalar(0, 43, 46), Scalar(10, 255, 255), mask1);
104       inRange(src_frame_HSV, Scalar(156, 43, 46), Scalar(180, 255, 255), mask2);
105         Mat mask = mask1 | mask2;
106       imshow("Mask", mask);*/
107   Mat model_mask1,model_mask2;//模板分割颜色
108   inRange(model_HSV, Scalar(0, 43, 46), Scalar(10, 255, 255), model_mask1);    //颜色分割并二值化
109       inRange(model_HSV, Scalar(156, 43, 46), Scalar(180, 255, 255), model_mask2);
110         Mat model_mask = model_mask1 | model_mask2;
111   imshow("model_mask", model_mask);
112   Mat mask;       //目标图像分割颜色
113   inRange(src_frame_HSV,Scalar(lowh,lows,lowv),Scalar(highh,highs,highv),mask);
114   //Mat model_mask;
115
116
117   Mat blur;//除噪声
118   Mat open;
119   /*//开操作 (去除一些噪点)  如果二值化后图片干扰部分依然很多, 增大下面的size
120     Mat element = getStructuringElement(MORPH_RECT, Size(5, 5));
121     morphologyEx(mask, open, MORPH_OPEN, element);
122
123   //闭操作 (连接一些连通域)
124     morphologyEx(open, blur, MORPH_CLOSE, element);
125     imshow("zaosheng",blur);*/
126
127   //高斯滤波
128   Mat blur1;
129   cv::GaussianBlur(mask,blur1, cv::Size(3, 3), 3, 3);
130   imshow("GaussianBlur", blur1);
131
132   //闭操作 (连接一些连通域)
133   Mat element = getStructuringElement(MORPH_RECT, Size(5, 5));
134     morphologyEx(blur1, blur, MORPH_CLOSE, element);
135     imshow("zaosheng",blur);
```

```
136
137   vector<vector<Point> > contours,model_contours;//寻找边界
138   vector<Vec4i> hierarchy,model_hierarchy;
139   findContours(blur,contours,hierarchy,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
140   findContours(model_mask,model_contours,model_hierarchy,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
141   //cout<<model_contours.size()<<endl;
142   vector<vector<Point> > match;    //边界匹配
143   double match_T;
144   for(int i=0;i<contours.size();i++)
145   {
146     match_T=matchShapes(contours[i],model_contours[0],CV_CONTOURS_MATCH_I3,1.0);
147     if(match_T<10)
148     {
149       match.push_back(contours[i]);     //记录匹配的边界
150     }
151   }
152   //cout<<match.size()<<endl;
153   Mat lunkuo=src_frame.clone();     //用矩形框出轮廓的图像
154     /*vector<RotatedRect> minRect( contours.size() );
155   for (int i = 0; i<contours.size(); i++)
156   {
157       minRect[i] = minAreaRect( Mat(contours[i]) );
158
159   }
160     for( int i = 0; i< contours.size(); i++ )
161       {
162         //Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
163         for( int j = 0; j < 4; j++ )
164           {
165                   int area=minRect[i].size.area();
166       if(area>700)
167       {
168             Point2f rect_points[4]; minRect[i].points( rect_points );
169       line( lunkuo, rect_points[j], rect_points[(j+1)%4], Scalar(0, 0, 255), 1, 8 );
170       }
171     }
172       }*/
173   vector<RotatedRect> minRect( match.size() ); //最小矩形框住边界
174   for (int i = 0; i<match.size(); i++)
175   {
176       minRect[i] = minAreaRect( Mat(match[i]) );
177
178   }
179 vector<Point2f> center_point;
180     for( int i = 0; i< match.size(); i++ )   //绘制矩形
181       {
182         //Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
183
184         for( int j = 0; j < 4; j++ )
185           {
186                   int area=minRect[i].size.area();
187       if(area>500)
188       {
189             Point2f rect_points[4]; minRect[i].points( rect_points );
190
191
192       line( lunkuo, rect_points[j], rect_points[(j+1)%4], Scalar(0, 0, 255), 1, 8 );
193       center_point.push_back(minRect[i].center);    //记录中心点
194       }
195     }
196       }
197 cout<<center_point<<endl;
198   imshow("lunkuo",lunkuo);
199       //这里是自定义的求取形心函数，当然用连通域计算更好
200
201       //Point center;
202
203       //center = GetCenterPoint(imgThresholded);//获取二值化白色区域的形心
204   int x_target=0;
```

```
205   int y_target=0;
206   int center_point_size=1;
207   //center_point.at(i).x
208   for(int i=0;i<center_point.size();i++)      //中心点求均值计算目标点
209   {
210     x_target=x_target+center_point.at(i).x;
211     y_target=y_target+center_point.at(i).y;
212     center_point_size++;
213   }
214   x_target=x_target/center_point_size;
215   y_target=y_target/center_point_size;
216   float linear_vel, angular_vel = 0;
217   if(y_target<src_frame.rows/10&&y_target>=0)   //若已经看不见目标则停下
218   {
219     linear_vel=0;
220     angular_vel = 0;
221     //exit(0);
222   }
223     else
224     {
225     Navigate(x_target, y_target, src_frame.cols, src_frame.rows, linear_vel, angular_vel);
226     }
227       //circle(img, center, 100, Scalar(0,0,255), 5, 8, 0);//绘制目标位置
228
229       //im("end", img);
230
231 #ifndef READIMAGE_ONLY
232   //以下代码可设置机器人速度值，从而控制机器人运动
233     geometry_msgs::Twist cmd_red;
234     cmd_red.linear.x = linear_vel;
235     cmd_red.linear.y = 0;
236     cmd_red.linear.z = 0;
237     cmd_red.angular.x = 0;
238     cmd_red.angular.y = 0;
239     cmd_red.angular.z = angular_vel;
240     pub.publish(cmd_red);
241 #endif
242     ros::spinOnce();
243     waitKey(5);
244   }
245   return 0;
246 }
```