

“联迪商用杯” 颜色识别 结题报告

CRCam 队

董皓，沈琳，檀锦彬

摘要

颜色识别是图像处理的重要组成部分，在当代社会中有着重要的应用。在本项目中，我们通过任务分解，将模型简化为一个个可实现的小目标，进行多次迭代，逐步实现图像颜色和内容的识别。首先，在 Visual Studio2015 开发平台上，配置 OpenCV 和 Qt，用 k-means 聚类方法对图像进行颜色分区，在用户界面上动态生成彩色标签，根据用户需求可显示不同色块。其次，作出彩色直方图，用以直观地表现每一种颜色在整幅图像中所占的比例。最后，在颜色识别的基础上，加入了人脸识别、数字识别和形状识别，拓宽了代码的广度和完备性，使得本项目更加具有实际应用意义。

关键词：OpenCV 颜色识别 数字识别 人脸识别

1 开发环境

1.1 开发环境的选择

开发环境 visual studio 2015 + Qt5.6 + OpenCV3.0

1.2 选择选择 Qt 和 OpenCV 的原因

为什么选择 Qt 和 OpenCV?

Qt 是基于 C/C++ 的一个跨平台的 GUI 库，OpenCV 也是基于 C/C++ 的一个计算机视觉库，它实现了很多图像处理的算法。两者搭配使用能非常快捷地开发与计算机视觉、人工智能、图像处理等相关的 GUI 应用。

1.3 环境配置

Qt 的安装和配置环境花了大概一天半的时间，因为要和 opencv 3.0 一起搭配使用，并且不是在官方开发工具 QtCreator 里而是 Visual Studio 里，故需要一定的配置步骤。

下面是安装和配置的参考教程：

Opencv3.0: <http://tieba.baidu.com/p/3931710438>

Qt5.6: <http://www.docin.com/p-1496848449.html>

我们在 win8.1 系统下，Visual Studio 2015 中配置成功

2 工作进程

2.1 开发任务分解

2.1.1 前端

网上虽然有很多 Qt 的教程，但把整个教程都学一遍需要较多的精力，时间上不被允许，由于队伍中成员有开发 web 前端的经验，因此粗略地将开发 GUI 分解成四个部分来学习：



之后选择了教程中的几节重点看，例如教程的第 5 节 Qt 布局管理器，第 15 节-绘制图片，第 44 节-信号与槽等等，看懂了教程里的 examples 后就照猫画虎往程序里加自己的代码。总体的开发过程就是看教程，模仿例子，遇到问题百度谷歌+Qt 的官方 A 文档。

2.1.2 后端

在项目开始时，我们预设的目标是能够实现对图像中颜色的分析，包括图像中有哪些颜色，每一颜色占整幅图像的比例是多少，并分区显示图像中不同颜色的色块。这是一个比较复杂的过程，经过讨论，我们决定将模型简化，从最简单的黑白识别做起，一步步迭代，直到完成目标。

在颜色识别完成之后，我们希望能够对图像实现进一步的分析，比如图像中的轮廓，图像内容（人脸、数字、形状）等。基本规划如下：



2.2 使用的算法说明

2.2.1 颜色识别

(1) 数据源

要对图像进行处理，首先要确定数据源，并将数据源数字化，使数据转化为程序可以操控，分析的格式。

① 本项目中数据源：任意图片；图片打开方式：文件选择—>打开。

② 在 `void get_RGB(IplImage* img, CvMat* samples)` 函数中调用 OpenCV 中 `CvScalar` `cvGet2D(const CvArr* arr, int idx0, int idx1)` 函数，获取图像各个像素点的三通道值 (RGB)，

③ 将像素点三通道的值按顺序排入样本矩阵。

具体实现如下：

```
//获取图像的RGB值，并存储到samples里面
void get_RGB(IplImage* img, CvMat* samples)
{
    int k = 0;
    for (int i = 0; i < img->width; i++)
    {
        for (int j = 0; j < img->height; j++)
        {
            CvScalar s;
            //获取图像各个像素点的三通道值 (RGB), s.val[0] 代表src图像BGR中的B通道的值~
            s = cvGet2D(img, j, i);
            cvSet2D(samples, k++, 0, s); //将像素点三通道的值按顺序排入样本矩阵
        }
    }
}
```

(2) 颜色聚类

通过上面的工作，可以实现简单的单颜色识别，但是对于颜色构成复杂的图像，要用聚类的方法分区判别图像颜色。步骤为：

① 我们用 OpenCV 中自带的 `k-means` 函数进行颜色聚类

```
cvKMeans2(samples, nClusters, clusters, cvTermCriteria(CV_TERMCRIT_EPS,
100, 1.0), 1, 0, 0, centers, 0);
```

② 获取每个聚类中心的 RGB 值，在前端动态生成彩色标签供用户选择

具体实现如下：

```

////////////////////////////////////
//获取每个聚类中心的RGB值，产生颜色按钮
void center_RGB(int nClusters, CvMat* centers)
{
    double out[3];
    double H, S, V;
    char *color;
    color = (char*)malloc(nClusters * sizeof(char));
    //for (int i = 0; i < nClusters; i++)
    int k = 0;
    for (int i = 0; i < nClusters * 3; i++)
    {
        out[2] = centers->data.fl[i++]; //centers里面第一个存的是B值
        out[1] = centers->data.fl[i++]; //第二个存的是G值
        out[0] = centers->data.fl[i++]; //第三个存的是R值
        //将每个聚类中心的RGB值转换成HSV值，方便判断其颜色

        /*Rgb2Hsv(out[0], out[1], out[2], H, S, V);
        color = judge_color(H, S, V);
        fout << "第" << k << "类: " << out[0] << " " << out[1] << " " << out[2] << " " << color << endl;
        fout << "第" << k++ << "类: " << H << " " << S << " " << V << " " << color << endl;*/
    }
}

```

(3) 颜色直方图

为了直观地表示每一种颜色在正幅图像中所占的比例，我们作出了彩色直方图。步骤为：

① 将输入图像转换到 HSV 颜色空间，根据 H, S 两个平面数据统计直方图，获取直方图统计的最大值，用于动态显示；

② 再获得当前直方图代表的颜色，转换成 RGB 用于绘制。

具体实现如下：

```

////////////////////////////////////
//彩色直方图
void histogram(IplImage *img)
{
    IplImage* hsv = cvCreateImage(cvGetSize(img), 8, 3);
    IplImage* h_plane = cvCreateImage(cvGetSize(img), 8, 1);
    IplImage* s_plane = cvCreateImage(cvGetSize(img), 8, 1);
    IplImage* v_plane = cvCreateImage(cvGetSize(img), 8, 1);
    IplImage* planes[] = { h_plane, s_plane };

    /** H 分量划分为16个等级，s分量划分为8个等级 */
    int h_bins = 16, s_bins = 8;
    int hist_size[] = { h_bins, s_bins };

    /** H 分量的变化范围 */
    float h_ranges[] = { 0, 180 };

    /** s 分量的变化范围 */
    float s_ranges[] = { 0, 255 };
    float* ranges[] = { h_ranges, s_ranges };

    /** 输入图像转换到HSV颜色空间 */
    cvCvtColor(img, hsv, CV_BGR2HSV);
    cvCvtPixToPlane(hsv, h_plane, s_plane, v_plane, 0);

    /** 创建直方图，二维，每个维度上均分 */
    CvHistogram * hist = cvCreateHist(2, hist_size, CV_HIST_ARRAY, ranges, 1);
    /** 根据H,s两个平面数据统计直方图 */
    cvCalcHist(planes, hist, 0, 0);

    /** 获取直方图统计的最大值，用于动态显示直方图 */
    float max_value;
    cvGetMinMaxHistValue(hist, 0, &max_value, 0, 0);
}

```

```

/** 设置直方图显示图像 */
int height = 240;
int width = (h_bins*s_bins * 6);
IplImage* hist_img = cvCreateImage(cvSize(width, height), 8, 3);
cvZero(hist_img);
/** 用来进行HSV到RGB颜色转换的临时单位图像 */
IplImage * hsv_color = cvCreateImage(cvSize(1, 1), 8, 3);
IplImage * rgb_color = cvCreateImage(cvSize(1, 1), 8, 3);
int bin_w = width / (h_bins * s_bins);
for (int h = 0; h < h_bins; h++)
{
    for (int s = 0; s < s_bins; s++)
    {
        int i = h*s_bins + s;
        /** 获得直方图中的统计次数，计算显示在图像中的高度 */
        float bin_val = cvQueryHistValue_2D(hist, h, s);
        int intensity = cvRound(bin_val*height / max_value);

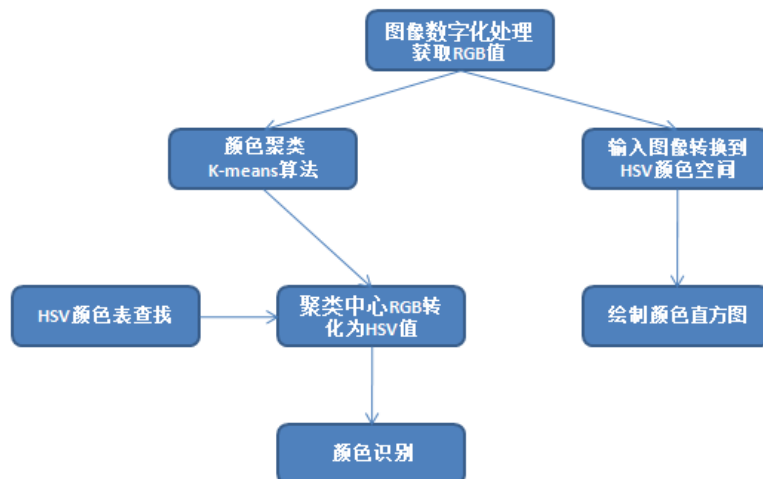
        /** 获得当前直方图代表的颜色，转换成RGB用于绘制 */
        cvSet2D(hsv_color, 0, 0, cvScalar(h*180.f / h_bins, s*255.f / s_bins, 255, 0));
        cvCvtColor(hsv_color, rgb_color, CV_HSV2BGR);
        CvScalar color = cvGet2D(rgb_color, 0, 0);

        cvRectangle(hist_img, cvPoint(i*bin_w, height),
                    cvPoint((i + 1)*bin_w, height - intensity),
                    color, -1, 8, 0);
    }
}

cvNamedWindow("H-S Histogram", 1);
cvShowImage("H-S Histogram", hist_img);
}

```

颜色识别实现流程图：



2.2.2 人脸识别

① 加载Haar特征检测分类器，haarcascade_frontalface_alt.xml是OpenCV自带的分类器；

② 用cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1)函数将需要人脸识别的图片转换成灰度图像；

③ 利用cvHaarDetectObjects(pGrayImage, pHaarCascade, pcvMStorage)函数对灰度图像做人脸识别，存入CvSeq *pcvSeqFaces，进行对pcvSeqFaces的圆形标记。

具体实现如下：

```
//////////////////////////////////////
void facerecog(IplImage *img)
{
    // 加载Haar特征检测分类器
    // haarcascade_frontalface_alt.xml系OpenCV自带的分类器
    const char *pstrCascadeFileName = "C:\\opencv\\sources\\data\\haarcascades_GPU\\haarcascade_frontalface_alt.xml";
    CvHaarClassifierCascade *pHaarCascade = NULL;
    pHaarCascade = (CvHaarClassifierCascade*)cvLoad(pstrCascadeFileName);

    IplImage *pGrayImage = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    cvCvtColor(img, pGrayImage, CV_BGR2GRAY);
    // 人脸识别与标记
    if (pHaarCascade != NULL)
    {
        CvScalar FaceCirclecolors[] =
        {
            { 0, 0, 255 },
            { 0, 128, 255 },
            { 0, 255, 255 },
            { 0, 255, 0 },
            { 255, 128, 0 },
            { 255, 255, 0 },
            { 255, 0, 0 },
            { 255, 0, 255 }
        };

        CvMemStorage *pcvMStorage = cvCreateMemStorage(0);
        cvClearMemStorage(pcvMStorage);
        //识别
        CvSeq *pcvSeqFaces = cvHaarDetectObjects(pGrayImage, pHaarCascade, pcvMStorage);
        // 标记

        // 标记
        for (int i = 0; i < pcvSeqFaces->total; i++)
        {
            CvRect* r = (CvRect*)cvGetSeqElem(pcvSeqFaces, i);
            CvPoint center;
            int radius;
            center.x = cvRound((r->x + r->width * 0.5));
            center.y = cvRound((r->y + r->height * 0.5));
            radius = cvRound((r->width + r->height) * 0.25);
            cvCircle(img, center, radius, FaceCirclecolors[i % 8], 2);
        }
        cvReleaseMemStorage(&pcvMStorage);
    }

    const char *pstrWindowsTitle = "人脸识别 ";
    cvNamedWindow(pstrWindowsTitle, CV_WINDOW_AUTOSIZE);
    cvShowImage(pstrWindowsTitle, img);
    cvWaitKey(0);
    cvDestroyWindow(pstrWindowsTitle);
    cvReleaseImage(&img);
    cvReleaseImage(&pGrayImage);
}
}
```

2.2.3 数字识别

① 初始化数字模板：将数字图片 i 二值化后利用 cvFindContours 检测出轮廓，对应存入 CvSeq * pic[i]数组中

```

//数字图片轮廓计算
void Init(void)
{
    IplImage *src0;
    int i;
    //创建空间
    CvMemStorage* mem = cvCreateMemStorage(0);

    for (i = 0; i<N; i++)
    {
        src0 = cvLoadImage(picture[i].c_str(), CV_LOAD_IMAGE_GRAYSCALE);
        IplImage*src = cvCreateImage(cvGetSize(src0), src0->depth, 1);
        if (!src0)
        {
            printf("Couldn't load %s\n", picture[i]);
            exit(1);
        }
        //二值化图像
        cvThreshold(src0, src0, thres, 255, CV_THRESH_BINARY_INV);
        cvCopy(src0, src);

        //cvCanny(src0, src,110, 110);
        cvFindContours(src, mem, &pic[i], sizeof(CvContour), CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE, cvPoint(0, 0))
    }
}

```

② 对源图片进行相同的工作，即轮廓检测，并将轮廓存入 `CvSeq * contours` 中。

对每个轮廓进行以下操作：

I、匹配：调用 `ReadNumber` 函数，利用 `cvMatchShapes` 函数判断出与此轮廓最匹配的数字轮廓：

```

int ReadNumber(CvSeq* contoursTemp,int moshi)
{
    int i;
    double tmp = 5;
    mint = 5;
    int num = -1;
    for (i = 0; i < N; i++)
    {
        //匹配
        tmp = cvMatchShapes(contoursTemp, pic[i], moshi);
        {
            mint = tmp;
            num = i;
        }
        //printf("%d: %f\n", i, tmp);
    }
    return num;
}

```

如果匹配值<1，将其值、横纵坐标记录在 `numList` 中

II、在原图上绘制轮廓外矩形：

利用 `cvBoundingRect` 函数获得轮廓最外面（up-right）矩形边界 `rect`，用 `cvRectangle` 画出轮廓外矩形

III、提取外轮廓 上的坐标点，画出轮廓：

获取 `contours` 中的每个坐标，用 `cvSetReal2D(contoursImage, pt->y, pt->x, 255.0)` 将轮廓画入 `contoursImage` 中

```

if (mint <= 1)
{
    //计算矩形顶点
    pt1.x = rect.x;
    pt1.y = rect.y;
    pt2.x = rect.x + rect.width;
    pt2.y = rect.y + rect.height;
    if (num >= 0)
    {
        numList[count][0] = num;           //一维存数字
        numList[count][1] = rect.x;        //二维存数字横坐标
        numList[count][2] = rect.y;
    }
    //在原图上绘制轮廓外矩形
    cvRectangle(imgColor, pt1, pt2, CV_RGB(0, 255, 0), 2);
    //提取外轮廓 上的坐标点
    for (i = 0; i < contoursTemp->total; i++)
    {
        CvPoint * pt = (CvPoint*)cvGetSeqElem(contoursTemp, i); // 读出第i个点。
        cvSetReal2D(contoursImage, pt->y, pt->x, 255.0);
        //cvSet2D(imgColor, pt->y, pt->x, cvScalar(0, 0, 255, 0));
    }
    count++;    //数字轮廓+1
}
}

```

IV、将数字按横坐标从左往右排列

调用 Sort 子函数，利用冒泡算法进行横坐标的排序，使其能够按照顺序输出。

```

void Sort(int numList[1000][3], int count)    //将数字按横坐标从左往右排列
{
    int i, j; //循环
    int n = 0; //融合后的数字个数
    int tem;
    int head = 0, tail = 1;
    int width = 0; //两数字间的距离
    int newList[1000] = { 0 }; //数字融合后的新序列
    for (i = 0; i < count - 1; i++)
    {
        for (j = i + 1; j < count; j++)
        {
            if (*numList[i][2] > numList[j][2] || /* numList[i][1] > numList[j][1]
            {
                //交换坐标
                tem = numList[i][1]; numList[i][1] = numList[j][1]; numList[j][1] = tem;
                //交换数字
                tem = numList[i][0]; numList[i][0] = numList[j][0]; numList[j][0] = tem;
            }
        }
    }
    if (count == 0)
    {
        printf("no number!");
    }
    else
    {
        for (i = 0; i < count; i++)
        {
            printf("%d\t", numList[i][0]);
        }
    }
}

```

2.2.4 圆的识别

(1) 对象模型

圆具有典型的几何特征：①任意两条弦的中垂线交于圆心；②圆上的点到圆心的距离均为半径。因此一个圆可抽象为一个三元组 $\langle x, y, r \rangle$ ， (x, y) 为圆心坐标， r 为半径。有了圆的模型，程序总体流程如下：



(2) 算法原理

圆形识别程序的核心基于 RANSAC 算法¹,该算法最早由 Fischler 和 Bolles 于 1981 年提出²。算法的基本思想是通过不断尝试不同的目标空间参数,使得目标函数值最大化的过程。该过程是随机、数据驱动的过程。通过反复的随机选择数据集的子空间来产生一个模型估计,然后利用估计出来的模型,使用数据集剩余的点进行测试,获得一个得分,最终返回一个得分最高的模型估计作为整个数据集的模型。算法的伪代码如下:

```
while iteration < k
    random select a subSet of dataSet
    generate a maybeModel suitable for subSet
    use remainSet test maybeModel to get a score
    update bestScore and bestModel
return bestModel
```

本程序基于该算法,但最终不是返回一个最优解模型,而是返回适合于数据集的所有圆的模型,然后才能将它们绘制出来。其中涉及到阈值的适当选择,即模型被判定为足够好,可适合于数据集的阈值。

(3) 程序实现

边缘化生成观测数据集

```

Mat edges;
Canny(image, edges, MAX(canny_threshold / 2, 1), canny_threshold);
std::vector<Point2d> points; // Create point set
for (int r = 0; r < edges.rows; r++)
{
    for (int c = 0; c < edges.cols; c++)
    {
        if (edges.at<unsigned char>(r, c) == 255)
            points.push_back(cv::Point2d(c, r));
    }
}

```

随机选取一个子集（为什么是 4 个点）

```

// get 4 random points
pointA = points[rng.uniform((int)0, (int)points.size())];
pointB = points[rng.uniform((int)0, (int)points.size())];
pointC = points[rng.uniform((int)0, (int)points.size())];
pointD = points[rng.uniform((int)0, (int)points.size())];

```

生成模型(计算弦方程 -> 中垂线的交点 -> 半径)

```

//AB
m_AB = (pointB.y - pointA.y) / (pointB.x - pointA.x + 0.000000001); //
b_AB = pointB.y - m_AB*pointB.x;
//BC
m_BC = (pointC.y - pointB.y) / (pointC.x - pointB.x + 0.000000001); //
b_BC = pointC.y - m_BC*pointC.x;

//find perpendicular bisector
//AB
XmidPoint_AB = (pointB.x + pointA.x) / 2.0; //midpoint
YmidPoint_AB = m_AB * XmidPoint_AB + b_AB;
m2_AB = -1.0 / m_AB;
b2_AB = YmidPoint_AB - m2_AB*XmidPoint_AB;
//BC
XmidPoint_BC = (pointC.x + pointB.x) / 2.0; //midpoint
YmidPoint_BC = m_BC * XmidPoint_BC + b_BC;
m2_BC = -1.0 / m_BC;
b2_BC = YmidPoint_BC - m2_BC*XmidPoint_BC;

//find intersection = circle center
x = (b2_AB - b2_BC) / (m2_BC - m2_AB);
y = m2_AB * x + b2_AB;
center = Point2d(x, y);
radius = cv::norm(center - pointB);

```

剩余点测试得到的模型，适合则加入 vote 集合（这里为什么没有体现剩余集合）

```
std::vector<int> votes;
std::vector<int> no_votes;
for (int i = 0; i < (int)points.size(); i++)
{
    double vote_radius = norm(points[i] - center);
    if (abs(vote_radius - radius) < radius_tolerance)
        votes.push_back(i);
    else
        no_votes.push_back(i);
}
```

vote 集合中元素大于设定阈值，判定模型足够好，适合于数据集

```
if ((float)votes.size() / (2.0*CV_PI*radius) >= circle_threshold)
{
    circles.push_back(Vec3f(x, y, radius));
}
```

绘制算法得到的圆

```
cvtColor(image, image, CV_GRAY2RGB);
for (int i = 0; i < (int)circles.size(); i++)
{
    int x = circles[i][0];
    int y = circles[i][1];
    float rad = circles[i][2];
    circle(image2, Point(x, y), rad, Scalar(0, 0, 255), 3);
}
```

(4) 分析改进

由于本程序最终目的不是返回一个最优解，而是判定足够好的解，即匹配度足够高的圆形形状，因此可以在算法基础上做一些改进来提高速度。例如：

① 原算法为了得到最优解，假定每次选取的点都是独立的，即本次局内点下一次仍然可能被选中。而本程序只需要得到判定形状足够匹配的圆，因此当得到的模型足够好时，就将数据集缩小为圆以外的点来减少观测量。

```
std::vector<Point2d> new_points;
for (int i = 0; i < (int)no_votes.size(); i++)
{
    new_points.push_back(points[no_votes[i]]);
}
points.clear();
points = new_points;
```

② 不使用模型所需的最小子集而另加上一个测试点 D，如果这个点不能适用于该模型，则有较大概率上该模型不能适用于数据集，因为该模型也是随机生成的。结果证明，

一定概率上确实可以有效提高算法速度（可以注释掉该改进试试）。

```
//check if the 4 point is on the circle
if (abs(cv::norm(pointD - center) - radius) > radius_tolerance)
    continue;
```

由于算法的迭代次数是没有上限的，预先设定的 K 值可能大于实际需要的迭代数。可以设定一个剩余点数阈值，若数据集中点数已经不超过该值，则认定已经识别完成，无需继续迭代直接离开。

```
if ((int)points.size() < points_threshold)
    break;
```

2.2.5 多边形识别

多边形的识别采用的是直线检测法，利用 `cvHoughLine` 在二值化图像中检测直线，其中涉及到的技术是 `hough` 变换³，该过程在一个参数空间中通过计算累计结果的局部最大值得到一个符合该特定形状的集合作为霍夫变换结果。Hough 变换于 1962 年由 Paul Hough 首次提出，后于 1972 年由 Richard Duda 和 Peter Hart 推广使用]，经典霍夫变换用来检测图像中的直线，后来霍夫变换扩展到任意形状物体的识别。下图展示了 `hough` 变换检测直线并连接边缘的流程：

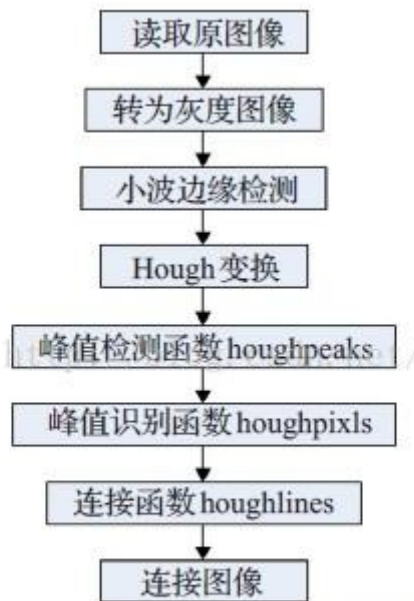


图 10 Hough 变换法边缘连接流程图

具体的程序实现为：

```

CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* lines = 0;
dst = cvCreateImage(cvGetSize(src), 8, 1);
cvCanny(src, dst, 50, 200, 3);
lines = cvHoughLines2(dst, storage, CV_HOUGH_PROBABILISTIC, 1, CV_PI / 180, 50, 50, 10);
IplImage* color_dst = cvLoadImage(filename, 1);
for (i = 0; i < lines->total; i++)
{
    CvPoint* line = (CvPoint*)cvGetSeqElem(lines, i);
    cvLine(color_dst, line[0], line[1], CV_RGB(0, 0, 255), 3, CV_AA, 0);
}

```

2.3 遇到的问题及解决方法

2.3.1 配置环境遇到的问题

① 在 VC++ 目录 -> 引用目录中也要添加

OpenCV 安装目录\opencv\build\x64\vc12\staticlib;

OpenCV 安装目录\opencv\build\x64\vc12\lib;

OpenCV 安装目录\opencv\build\x64\vc12\bin;

这三个文件（上面的教程中没有添加）

② 在 C/C++ -> 附加包含目录中包含\$(QTDIR)\include 以及 include 以下的子目录

③ 若要在 Qt 应用工程中，使用控制台帮助调试，就在链接器 -> 系统 -> 子系统中选择“控制台 (/SUBSYSTEM:CONSOLE)”即可调出控制台。

2.3.2 颜色识别过程中遇到的问题

① 获取聚类中心时，由于一开始不清楚 `cvCreateMat((img->width)*(img->height), 1, CV_32FC3)` 这个函数存储返回值的方式，走了一些弯路。解决办法：构造函数 `center_RGB(nClusters, centers)`，获取每个聚类中心的 RGB 值

② 在做颜色判别时，一开始想用 RGB 值直接判断，但是我们发现两种直观上看起来一样的颜色 RGB 值可能差异很大，没办法划定一个范围。所以要把 RGB 转换为 HSV 值，在转换过程中，因为不同资料给出的区间不一样，有的 S 值在 $[0, 1]$ ，有的则是在 $[0, 255]$ 。一开始以为是公式出了错误，思考了很久。发现之后乘以对应的系数就解决了。

2.3.3 数字识别过程中遇到的问题

①一开始没有二值化图片，直接利用 `cvFindContours()` 函数进行轮廓检测，导致出来的结果很不理想。

②对于 `cvMatchShapes()` 函数的参数没有很好的理解，比如其中第三个参数 `method`，表示用那种方法进行匹配。起初一直没有设置，结果很不理想，后来发现设置成 3 之后，结果变得准确很多。

③对于检测到数字之后的输出顺序问题没有注意。因为 `cvFindContours()` 函数并不是按照顺序检测轮廓的，所以需要加一个排序函数来输出。

2.3.4 圆形识别过程中遇到的问题

由于 RANSAC 算法没有迭代上限，也没有指定一些判定阈值，所以一开始使用该算法的时候，阈值和上限设置的不够恰当，导致程序运行的结果和预期相差较大。经过对程序的仔细分析，明白了阈值的设置其实与图片大小有很大的关系，如 `radius` 阈值的设置：

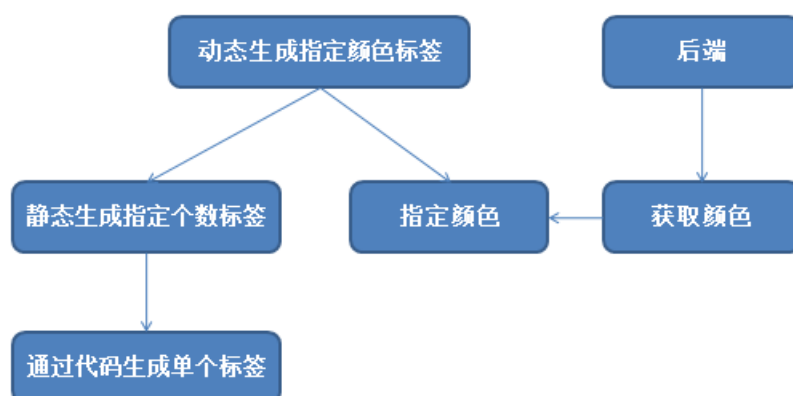
```
int radius_tolerance = 3;
```

它是以像素为单位的，若设置的太大，图片较小时会识别出过多无用的图形，而设置的过小，当图片较大时会识别不出正确的圆。未来将考虑把这些阈值改进，设置为与图片大小相关联，自适应自=地改变自身的大小。

2.3.4 Qt 开发过程中遇到的难点

①彩色标签的动态生成

分解：



QT 内提供了 designer mode，因此之前开发过程都是通过鼠标拖拽生成标签的，我查看控件的源代码之后仿照着生成标签，但是没有得到显示。在网上寻找帮助，查到通过代码生成控件用 new QLabel 和 setParent 是不行的。而应该是先指定布局，如下：

```
qvbox = new QVBoxLayout(widget)

for 1 to n

    label = new QLabel

    qvbox.addWidget(label).
```

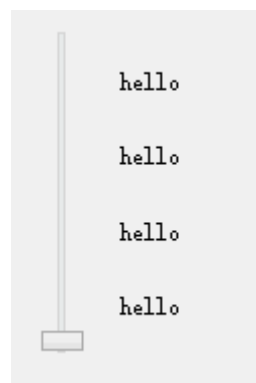
之后指定颜色可用 setStyleSheet 方法，与 css 语法兼容。

下面开发过程的实时记录：

(1) 生成一个标签使用 new QLabel 然后 setParent 是不行的，需要 new 一个 QVBoxLayout 或其他布局，在布局中 addWidget(label)。

(2) 生成多个标签，使用 new QLabel[10] 这样是不行的，我的实现方案是加一个循环来 new QLabel 和 addWidget，且 Layout 只能 new 一次，故应该在循环外。

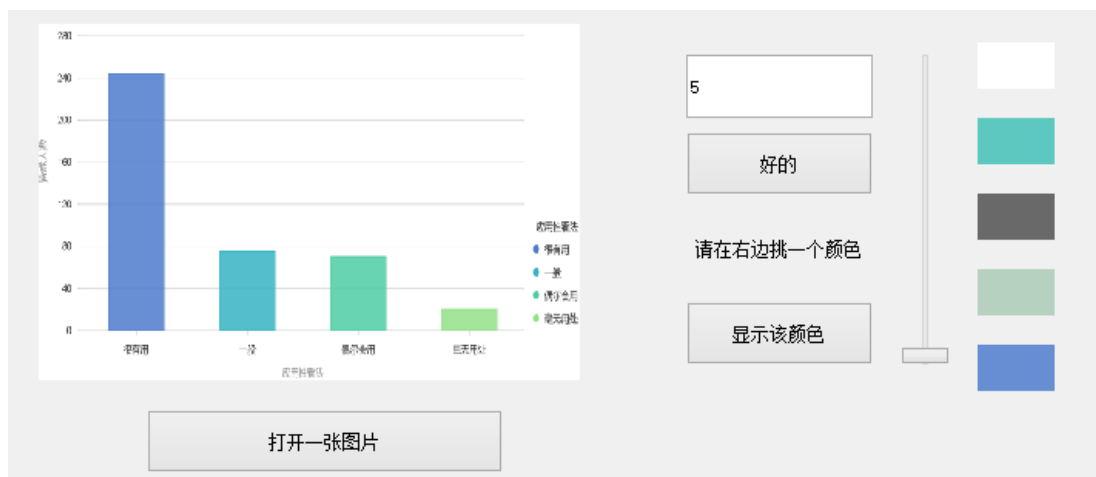
(3) 以上的 addWidget 若以 centerWidget（主面板）为放置对象，就是在整个页面内布局生成的 label，显然不是我的目的，我的实现方案是在滑动条旁放置一个等高的 label 当作布局面板，成功达到了初步的效果：



(4) 指定颜色，帮助文档中提到需要使用 setStyleSheet() 方法，参数格式与 CSS 相同，设置后的效果：



(5) 显然需要将 label 的尺寸重新设置，由于使用了布局，setGeometry() 方法不再生效，换了 resize() 方法也不见效果，最后使用 setMaximumHeight() 方法改变了高度，不能达到完美，但通过合理调整 labelPane 和 label 的尺寸，最后得到的效果满足需求：



上面还有一个小问题需要提一下，生成彩色块使用的 addWidget 是每次添加在末尾，而滑动条的值从下到上是 0-4，因此会出现颜色对应反序的问题，这里使用了一个栈来解决的。

② Qt 界面内图片的重绘

需要重绘的情况是：

- 用户重新选择图片或聚类数后，彩色标签需要重绘
- OpenCV 进行图像处理之后，图片需要重绘

第一种情况的重绘需要擦除控件再添加。前一天晚上参考网上的代码却一直擦除失败，找不到原因。于是单独新建一个工程来测试擦除部分的代码，后来发现原来是删除控件的函数声明成了成员方法而不是槽，自然不能响应事件。

其实删除的步骤就是两步：

a: `layout->removeWidget(widget);`

b: `widget->deleteLater();`

很多个就要采用循环每个都做一遍。

第二种情况，最开始我们暂时抛开了它，采用的实现是将处理后的图片弹窗展示出来。但更友好方式应当是在 QT 页面内对原图重绘，但 Qt 库函数无法识别 OpenCV 处理后的图像格式。查阅网上资料，得知需要通过一系列转换步骤才能让 Qt 绘制出 OpenCV 的图像，其中涉及到一系列图像格式的基础知识，于是我们采用了另一种策略：将 OpenCV 处理后的图像先存为本地临时图片文件，再由 Qt 读取图片文件并绘制出来，这样在时间上并没有很明显的消耗，因此可作为临时解决方案来使用。

③中文路径的解决

Qt 对中文的兼容性较差，需要程序员亲自来转换编码，大量查阅网上的资料后，以下的方案可以解决某些问题：

解决中文字符串显示乱码：

```
QTextCodec::setCodecForTr(QTextCodec::codecForName("GB2312"));
```

```
setText(QString::fromLocal8Bit("正常显示中文"));
```

解决文件的中文路径问题

```
QString fileName = QFileDialog::getOpenFileName(this, tr("Open Image"), "", tr("Image  
File(*.bmp *.jpg *.jpeg *.png)"));
```

```
QTextCodec *code = QTextCodec::codecForName("utf8");
```

```
string name = code->fromUnicode(fileName).data();
```

```
fileName = QString::fromStdString(name); // Qt 读取
```

```
image = cv::imread(name); // OpenCV 读取
```

但图片文件名若包含中文字符，仍然无法正常显示，这一问题还未解决。

3 测试效果

3.1 颜色识别





打开图片

8

分析颜色

请在上方选择颜色

颜色识别

人脸识别

圆形/多边形





打开图片

8

分析颜色


请在上方选择颜色


颜色识别

人脸识别

圆形/多边形

3.2 人脸识别





打开图片

8

分析颜色

请在上方选择颜色

颜色识别

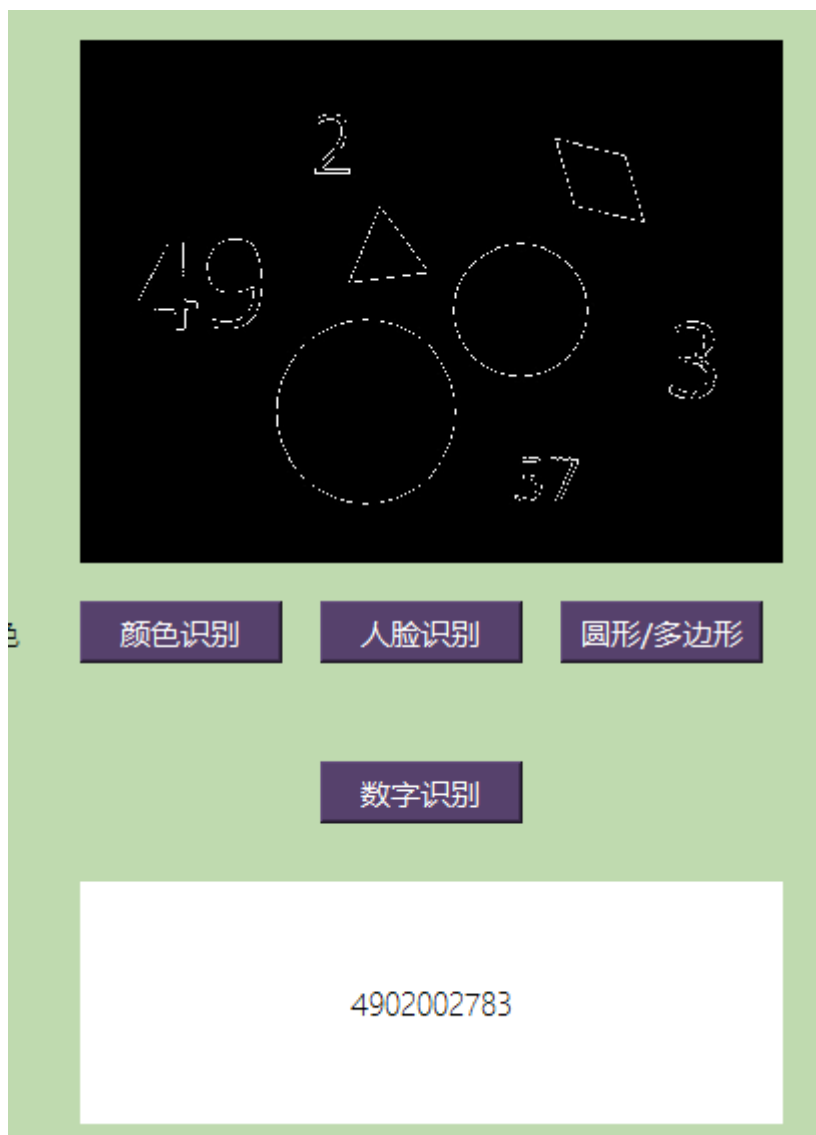
人脸识别

圆形/多边形

3.3 圆形/多边形识别



3.4 数字识别



3.5 综合：身份证的识别

姓名 奥巴马

性别 男 民族 肯尼亚卢欧族

出生 1961 年 8 月 4 日

住址 华盛顿市区中心宾夕法尼亚大街1600号

公民身份号码 123456196108041236

打开图片

6

分析颜色

请在上方选择颜色

颜色识别

人脸识别

圆形/多边形

姓名 奥巴马

性别 男 民族 肯尼亚卢欧族

出生 1961 年 8 月 4 日

住址 华盛顿市区中心宾夕法尼亚大街1600号

公民身份号码 123456196108041236

打开图片

6

分析颜色

请在上方选择颜色

颜色识别

人脸识别

圆形/多边形

姓名 奥巴马

性别 男 民族 肯尼亚卢欧族

出生 1961 年 8 月 4 日

住址 华盛顿市区中心宾夕法尼亚大街1600号

公民身份号码 123456196108041236

打开图片

6

分析颜色

请在上方选择颜色

颜色识别

人脸识别

圆形/多边形

颜色分布图

数字识别

121456196108041716

4 不足

4.1 颜色识别的不足

由于没有查找到范围定义明确详尽的颜色表，因此本项目中没有显示出确定的颜色，而是生成彩色标签来表示。

4.2 数字识别的不足

4.2.1 数字 6 和 9 只能识别一个

数字识别中是利用 `cvMatchShapes` 函数来判断与输入图片轮廓最匹配的数字轮廓。这个函数通过对轮廓上的点进行运算得到轮廓的矩，而矩对缩放，旋转，镜像映射具有不变性，这也就意味着数字旋转后无法正确匹配，因此数字 6 和 9 只能识别一个。

4.2.2 只能识别同一行的数字

由于图片中的轮廓是随机获取的，图像中的数字序列存储在二维数组 `numList[][]` 中（一维是数字，二维是数字所在横坐标），在输出时利用冒泡算法进行横坐标的排序，使其能够按顺序输出，因此不能识别两行及以上的数字。

5 改进

定义颜色表

要识别图片颜色，需要进行颜色数据采集，建立颜色表，然后查找颜色表，确定颜色。由于 RGB 三通道的范围都是 $[0, 255]$ ，如果不同 RGB 值定义为不同颜色，对于颜色构成复杂的图片，识别出的颜色种类数会非常多，这是既不现实的，也不是我们想要的结果。因此在定义颜色表时，要允许颜色容差，即在某个范围内为同一种颜色。

经过观察我们发现相近颜色的 RGB 值差别可能很大，难以总结出规律。那么有没有其他的定义可以确定颜色范围呢？通过查找资料，我们决定用 HSV 值来定义颜色表。颜色表如下：

	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

具体实现如下：

```
char* judge_color(double H, double S, double V)
{
    if (S >= 43)
    {
        if (V <= 46)
            return "黑色";
        else
        {
            if (H < 11 || (H >= 156 && H <= 180))
                return "红色";
            else if (H >= 11 && H < 26)
                return "橙色";
            else if (H >= 26 && H < 35)
                return "黄色";
            else if (H >= 35 && H < 78)
                return "绿色";
            else if (H >= 78 && H < 100)
                return "青色";
            else if (H >= 100 && H < 125)
                return "蓝色";
            else if (H >= 125 && H < 156)
                return "紫色";
            else
                return "未识别颜色";
        }
    }
    else
    {
        if (S > 30 && V >= 46 && V < 221)
            return "灰色";
        else if (S <= 30 && V >= 221 && V <= 256)
            return "白色";
    }
}
```

```

        else
            return "未识别颜色";
    }
}

```

由于我们读取图片得到的是 RGB 值, 因此要先将 RGB 值转换为 HSV 值, 转换公式为:

$$h = \begin{cases} \text{undefined} & \text{if } \max = \min \\ 60^\circ \times \frac{g-b}{\max-\min} + 0^\circ, & \text{if } \max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{\max-\min} + 360^\circ, & \text{if } \max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{\max-\min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \frac{r-g}{\max-\min} + 240^\circ, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max-\min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

具体实现如下:

//将RGB值转换成HSV值, 方便判断颜色

```

void Rgb2Hsv(double R, double G, double B, double& H, double& S, double&V)
{
    double max1, min1;
    R = R / 255;
    G = G / 255;
    B = B / 255;
    max1 = max(R, G, B);
    min1 = min(R, G, B);
    if (max1 == min1)
        H = 0.0;
    else if (R == max1 && G >= B)
        H = (G - B) / (max1 - min1);
    else if (R == max1 && G < B)
        H = 6 + (G - B) / (max1 - min1);
    else if (G == max1)
        H = 2 + (B - R) / (max1 - min1);
    else if (B == max1)
        H = 4 + (R - G) / (max1 - min1);
    H = H * 60;
    if (H < 0)
        H = H + 360;
    V = max1;
    if (max1 == 0)
        S = 0;
}

```

```

else
    S = (max1 - min1) / max1;
    H = H / 2;          //H:0~180
    S = S * 255; //S:0~255
    V = V * 255; //V:0~255

}

```

5 总结

在这次项目开发的过程中，我们熟悉了 Visual Studio 的开发环境，掌握 OpenCV 和 Qt 的基本使用，了解 OpenCV 中的相关函数，对图像处理有了一定程度的认识，学习了一些与图像处理相关的算法。

队伍成员的分工合作、沟通协调也是项目能够完成的重要原因之一，更重要的是我们学会了在开发过程中逐步分解任务的思想，将大目标转化为能够实现的小目标，同时锻炼了我们的资料检索能力、动手能力和工程思维能力。

参考资料

[1] <http://www.cnblogs.com/xrwang/archive/2011/03/09/ransac-1.html>

[2] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model

fitting with applications to image analysis and automated cartography. Communications

of the ACM, 24(6):381–395, 1981.

[3] <http://blog.csdn.net/augusdi/article/details/9022827>

[4] 安装, http://blog.csdn.net/poem_qianmo/article/details/19809337

[5] 彩色直方图, <http://blog.csdn.net/leixiaohua1020/article/details/13624999>

[6] 数字识别, <http://www.cnblogs.com/qg76211822/p/4718868.html>

[7] 人脸识别, <http://blog.csdn.net/morewindows/article/details/8426318>