



计算机视觉

南区计软328

深圳大学
计算机与软件学院

实验内容

序号	实验主题	实验内容	实验要求	实验时数	每组人数	实验类型
1	图像处理应用实验	1. 熟悉图像的表示及基本元素、通道操作; 2. 掌握基本图像增强方法; 3. 掌握OpenCV计算机视觉库;	必做	6	1	讲授+实验
2	图像特征提取及综合应用实践	1. 熟悉图像处理基本操作; 2. 掌握图像边缘检测原理; 3. 掌握图像基本特征抽取以及在实际问题中的应用;	必做	6	1	讲授+实验
3	计算机视觉系统实践	1. 熟悉计算机视觉分类任务; 2. 掌握数据集的准备及模型训练过程; 3. 培养应用计算机视觉解决问题的能力;	必做	6	1	讲授+实验

涉及学科

- 数字图像处理
- 计算机视觉
- 模式识别
- 程序设计

实验考察形式

- 三次实验报告（在Blackboard发布与提交）
- 平时编程练习及表现
- 实验汇报

实验考察内容

- 计算机视觉与模式识别基础知识掌握能力
- 算法设计能力
- 编程及系统架构能力
- 论文写作及表达能力

● 实验一

实验一：图像增强应用实践

实验目的：

1. 熟悉图像表示及基本元素、通道操作；
2. 掌握基本图像增强方法；
3. 掌握OpenCV计算机视觉库；

实验要求：

1. 实验提交文件为实验报告和相关程序代码，以压缩包的形式提交，实验报告命名规则为“计算机视觉-学号-姓名-实验报告1.doc”，其他文件打包成压缩文件，命名为“计算机视觉-学号-姓名-实验报告1-其他.zip”；
2. 所有素材和参考材料需列明出处，实验报告中的图片和程序代码建议标注个人水印或标识信息：姓名，班级，学号信息；

二、实验内容：

- (1) 提取给定图像RGB和HSV颜色空间各通道下的图像；完成给定RGB、HSV颜色空间下各通道图像的合并，生成一张彩色图像；提取与合并操作需给出关键核心代码；
- (2) 不调用库函数，自己动手编程实现图像增强相关方法，并与OpenCV的库函数进行效果对比分析；

三、实验时间

实验报告统一命名为：计算机视觉-学号-姓名-实验报告1.doc

如有提交其他文件或程序，统一命名为：计算机视觉-学号-姓名-实验报告1-其他.zip

实验时间：2024年9月2日至2024年10月13日（注意按时提交，不要延期） 实验报告提交时间：
2024年10月13日下午5:00



OpenCV中关于图像操作（python实现）

- (1) OpenCV读取图像
- (2) 遍历图像上所有像素（如把图像左右翻转）
- (3) 图像缩放
- (4) 图像镜像
- (5) 仿射变换（目标校正）
- (6) 对一幅灰度图像进行直方图均衡化（作业）

(1) OpenCV读取图像

```
import cv2

# 读取图像
image = cv2.imread('image_path.jpg')

# 检查图像是否成功加载
if image is None:
    print("图像加载失败")
else:
    # 显示图像
    cv2.imshow('Image', image)

    # 等待按键
    cv2.waitKey(0)

    # 关闭所有窗口
    cv2.destroyAllWindows()
```

cv2.imread(filename, flags):

- 第一个参数：文件路径。
- 第二个参数（可选）：指定如何读取图像，默认是 `cv2.IMREAD_COLOR`，可以是以下几种：
 - （1）`cv2.IMREAD_COLOR`：以彩色模式读取图像（默认）。
 - （2）`cv2.IMREAD_GRAYSCALE`：以灰度模式读取图像。
 - （3）`cv2.IMREAD_UNCHANGED`：读取图像并保持其原始的透明度（Alpha通道）。

(2) 遍历图像上所有像素 (如把图像左右翻转)



(2) 遍历图像上所有像素 (如把图像左右翻转)

```
image = cv2.imread('image_path.jpg')

# 获取图像的行数（高度）和列数（宽度）
height, width = image.shape[:2]

# 创建一个空图像用于存放翻转后的数据
flipped_image = np.zeros_like(image)

# 遍历图像的每一个像素，并把它放到新的位置
for i in range(height):
    for j in range(width):
        # 将每一行的像素左右翻转，即把j位置的像素放到(width - 1 - j)位置
        flipped_image[i, j] = image[i, width - 1 - j]

# 显示原始图像和翻转后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Flipped Image', flipped_image)

# 等待按键关闭窗口
cv2.waitKey(0)
cv2.destroyAllWindows()
```

手动遍历图像的所有像素，并实现图像的左右翻转效果。

for i in range(height) 和 for j in range(width):

遍历图像的每一个像素位置，i 代表行，j 代表列。

flipped_image[i, j] = image[i, width - 1 - j]:

把图像中的像素从原来的列位置 j 移动到新的列位置 width - 1 - j，实现左右翻转。

(3) 图像缩放

cv2.resize(src, dsize, fx, fy, interpolation):

- **src**: 输入的图像。
- **dsize**: 输出图像的尺寸。它是一个元组，表示 (width, height)，即目标图像的宽度和高度。若使用缩放比例 **fx** 和 **fy** 进行缩放则可以设置为 **None**。
- **fx**: 水平方向上的缩放因子（宽度）。如果设置了 **dsize**，可以忽略此参数。
- **fy**: 垂直方向上的缩放因子（高度）。如果设置了 **dsize**，可以忽略此参数。
- **interpolation**（可选）：插值方法，决定如何计算新像素的值。常见的插值方法包括：
 - (1) **cv2.INTER_NEAREST**: 最近邻插值法，速度快但质量较低。
 - (2) **cv2.INTER_LINEAR**（默认）：双线性插值，通常用于缩小图像。
 - (3) **cv2.INTER_CUBIC**: 三次插值，效果较好，但速度较慢，通常用于放大图像。
 - (4) **cv2.INTER_LANCZOS4**: Lanczos 插值，效果最好，适用于图像缩放。

(3) 图像缩放

1、使用固定尺寸进行缩放 (300, 300)

```
import cv2

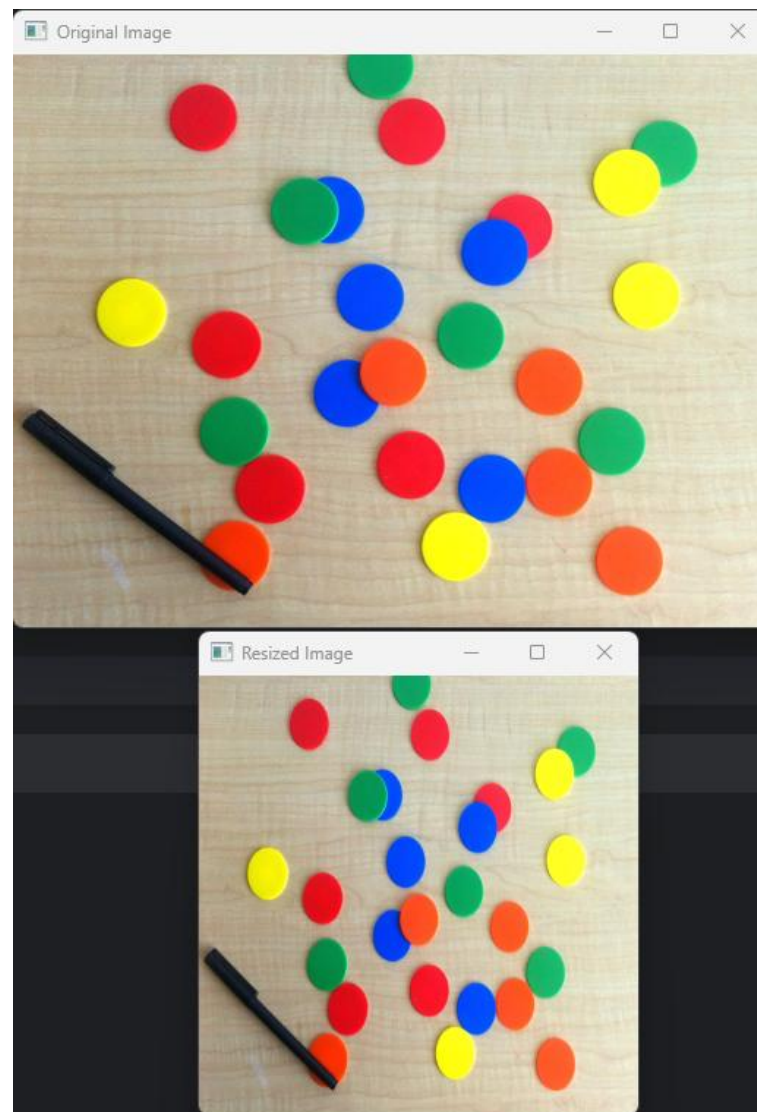
# 读取图像
image = cv2.imread('image_path.jpg')

# 设定缩放后的尺寸 (宽度, 高度)
new_size = (300, 300)

# 缩放图像到指定尺寸
resized_image = cv2.resize(image, new_size)

# 显示原始图像和缩放后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Resized Image', resized_image)

# 等待按键关闭窗口
cv2.waitKey(0)
cv2.destroyAllWindows()
```



(3) 图像缩放

2、使用缩放比例进行缩放 (0.5, 0.5)

```
import cv2

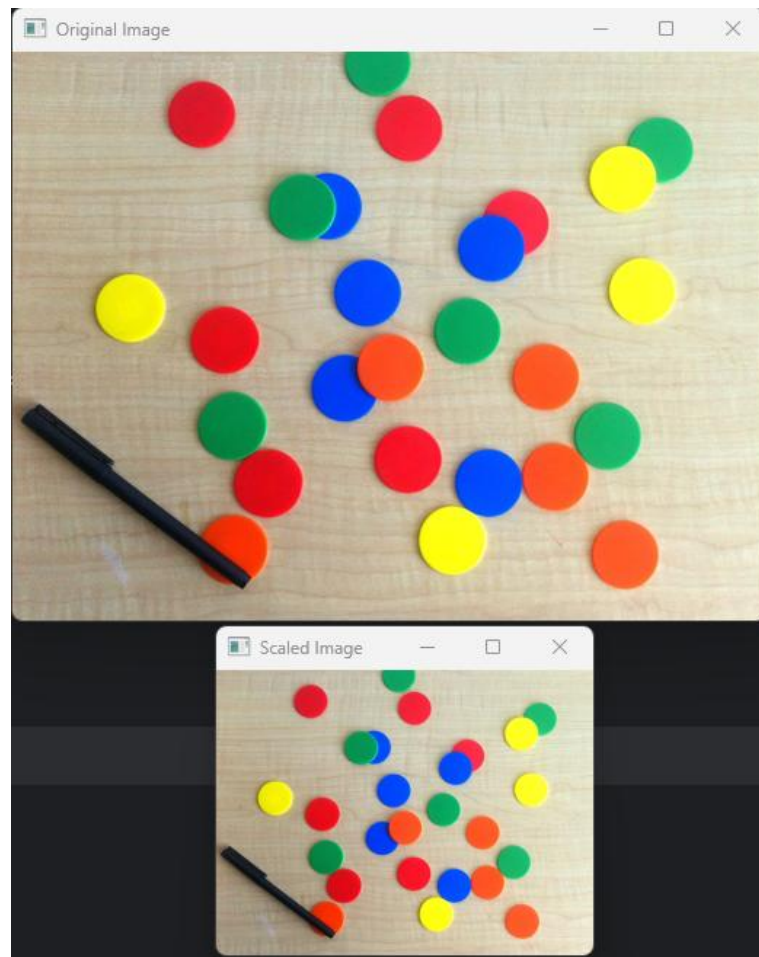
# 读取图像
image = cv2.imread('image_path.jpg')

# 设置缩放比例
fx = 0.5 # 宽度缩小一半
fy = 0.5 # 高度缩小一半

# 按比例缩放图像
resized_image = cv2.resize(image, None, fx=fx,
                             fy=fy)

# 显示原始图像和缩放后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Scaled Image', resized_image)

# 等待按键关闭窗口
cv2.waitKey(0)
cv2.destroyAllWindows()
```



(4) 图像镜像

```
import cv2

# 读取图像
image = cv2.imread('./img/boldt.jpg')

# 使用 OpenCV 的内置函数进行左右翻转
flipped_image = cv2.flip(image, 1)

# 显示原始图像和翻转后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Flipped Image', flipped_image)

# 等待按键关闭窗口
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OpenCV 提供了 `cv2.flip()` 函数。该函数可以更高效地实现图像翻转操作。

`cv2.flip(src, flipCode)`

- **src**: 输入图像，必须是一个有效的图像数组，可以是灰度图像或彩色图。
- **flipCode**: 指定翻转的方式。它的取值决定了翻转的方向：
 - (1) **1**: 水平翻转，即左右翻转（沿着垂直轴翻转）。
 - (2) **0**: 垂直翻转，即上下翻转（沿着水平轴翻转）。
 - (3) **-1**: 同时进行水平和垂直翻转，即对角翻转。



(5) 仿射变换

在 OpenCV 中，仿射变换是一种线性变换，用于改变图像的几何形状。

仿射变换能够对图像进行**旋转、缩放、平移**以及**剪切**等操作，同时保持图像的直线性质。仿射变换通过一个 2×3 的变换矩阵来实现，可以通过 OpenCV 的 `cv2.warpAffine()` 函数来应用仿射变换。

仿射变换的主要步骤如下：

1. 确定三个点：选取原图像上的三个点及其在目标图像中的对应点。
2. 计算仿射变换矩阵：通过 `cv2.getAffineTransform()`
3. 计算仿射变换矩阵。应用仿射变换：通过 `cv2.warpAffine()` 应用仿射变换。



1、平移变换

```
import cv2
import numpy as np

# 读取图像
image = cv2.imread('image_path.jpg')

# 平移矩阵: 平移 (x方向, y方向)
M = np.float32([[1, 0, 50], [0, 1, 100]])

# 应用仿射变换
translated_image = cv2.warpAffine(image,
M, (image.shape[1], image.shape[0]))

# 显示原始图像和平移后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Translated Image',
translated_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

平移是将图像中的所有像素移动一个固定的距离。

`np.float32([[1, 0, 50], [0, 1, 100]])` 是一个 NumPy 数组，用于表示二维仿射变换中的平移矩阵。

具体来说，这是一个 **2x3** 的矩阵，用于将图像的所有像素平移。

1、平移变换

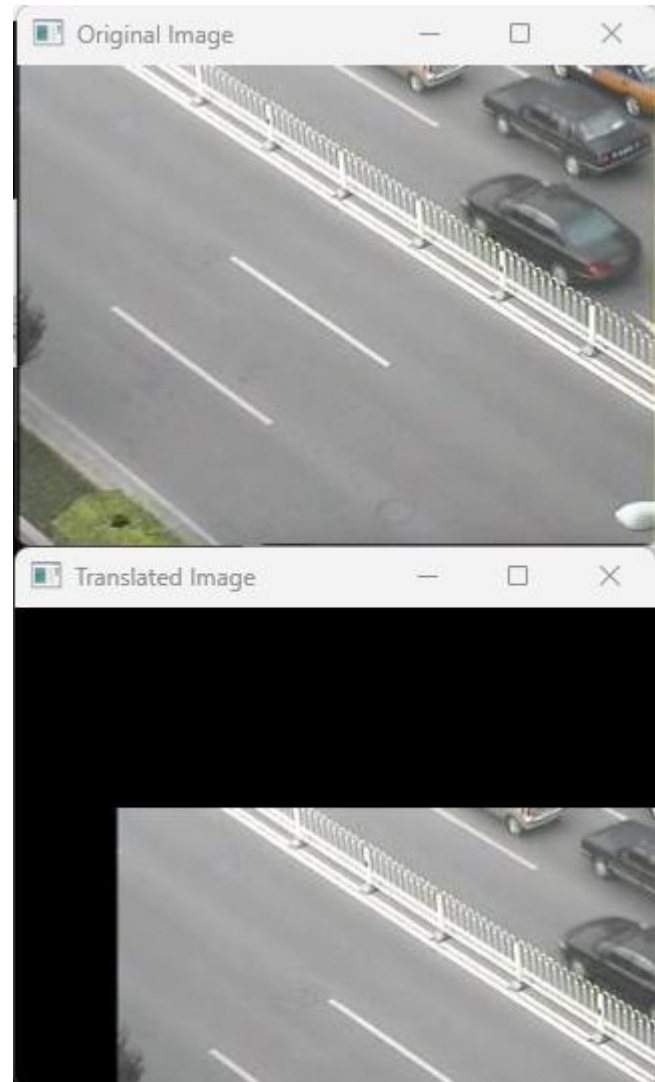
平移矩阵: 平移 (x方向, y方向)

```
M = np.float32([[1, 0, 50], [0, 1, 100]])
```

$$\begin{pmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \end{pmatrix}$$

第一行 $[1, 0, 50]$: 1 和 0 分别对应的是 x 方向的缩放和剪切（在这里没有进行缩放或剪切）。50 表示在 x 方向上的平移距离，即图像中的每个像素都会向右平移 50 个像素。

第二行 $[0, 1, 100]$: 0 和 1 分别对应的是 y 方向的缩放和剪切（在这里没有进行缩放或剪切）。100 表示在 y 方向上的平移距离，即图像中的每个像素都会向下平移 100 个像素。



2、旋转变换

```
# 读取图像
image = cv2.imread('image_path.jpg')

# 获取图像中心
(h, w) = image.shape[:2]
center = (w // 2, h // 2)

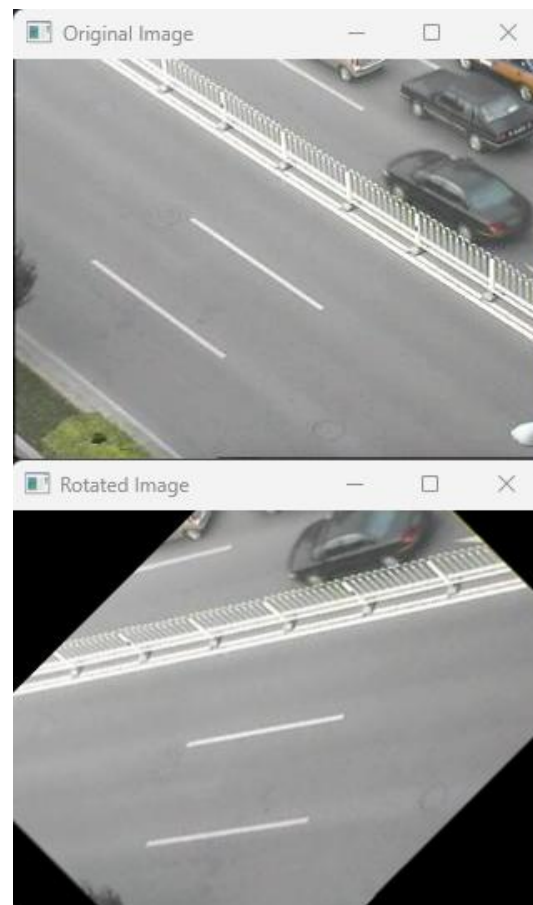
# 旋转矩阵: 旋转角度为 45 度, 缩放因子为 1
M = cv2.getRotationMatrix2D(center, 45, 1.0)

# 应用仿射变换
rotated_image = cv2.warpAffine(image, M,
(w, h))

# 显示原始图像和旋转后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

通过 `cv2.getRotationMatrix2D()` 来生成旋转矩阵, 并使用 `cv2.warpAffine()` 进行图像旋转。



3、仿射变换（变换三个点）

```
image = cv2.imread('image_path.jpg')

# 原始图像中的三个点
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

# 变换后的图像中的对应点
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

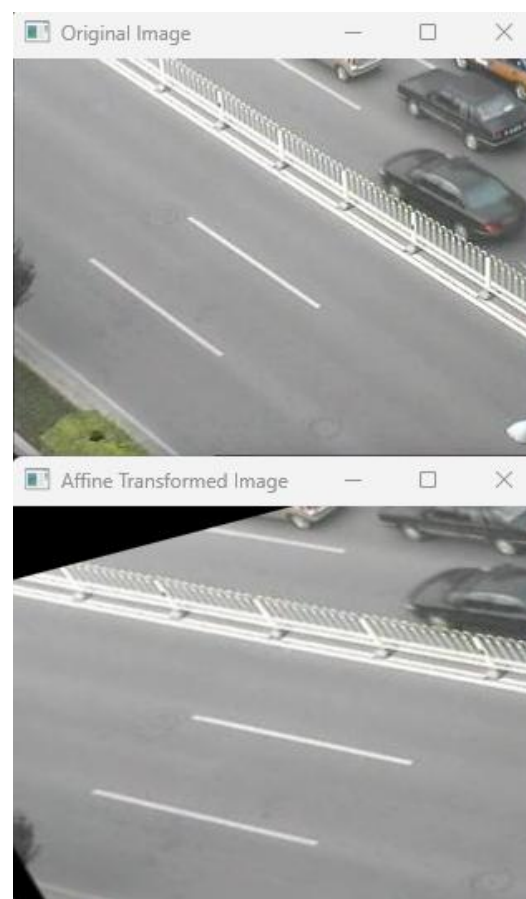
# 获取仿射变换矩阵
M = cv2.getAffineTransform(pts1, pts2)

# 应用仿射变换
affine_image = cv2.warpAffine(image, M,
                               (image.shape[1], image.shape[0]))

# 显示原始图像和仿射变换后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Affine Transformed Image', affine_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

通过变换三个点来控制图像的形变。



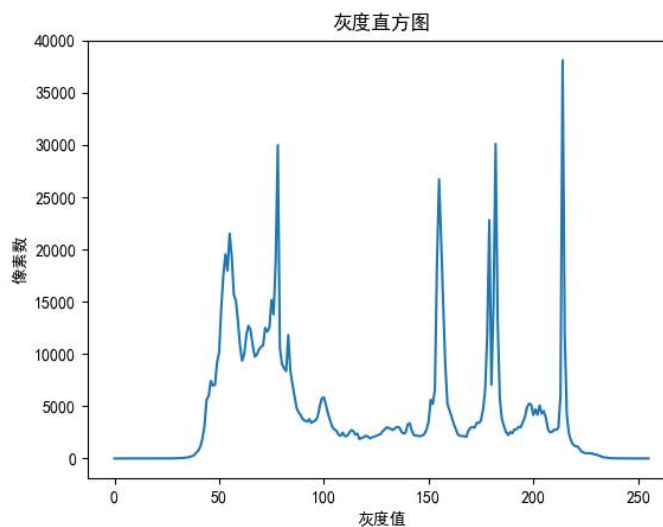
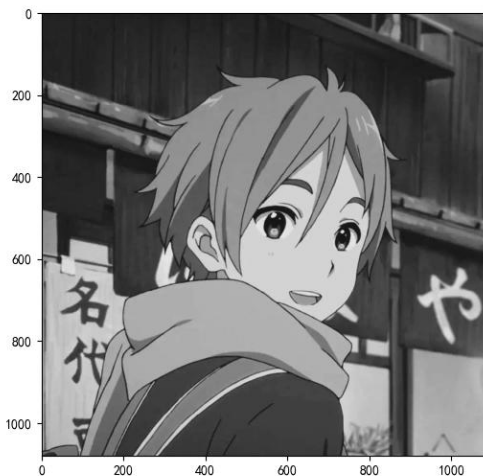
(6) 对一幅灰度图像进行直方图均衡化 (作业)

1、计算一幅单通道图像的直方图

在 OpenCV 中，可以使用 **cv2.calcHist()** 函数来计算图像的直方图。对于单通道图像（通常是灰度图像），直方图表示图像中每个灰度级别（0-255）出现的**像素数量**。

`cv2.calcHist([image], [0], None, [256], [0, 256]):`

- `[image]`: 需要计算直方图的图像。
- `[0]`: 通道索引，灰度图像只有一个通道，所以使用通道 0。
- `None`: 没有使用掩膜。
- `[256]`: 直方图的大小，即灰度级的数量（从 0 到 255，共 256 个灰度值）。
- `[0, 256]`: 像素值的范围，从 0 到 255。



(6) 对一幅灰度图像进行直方图均衡化 (作业)

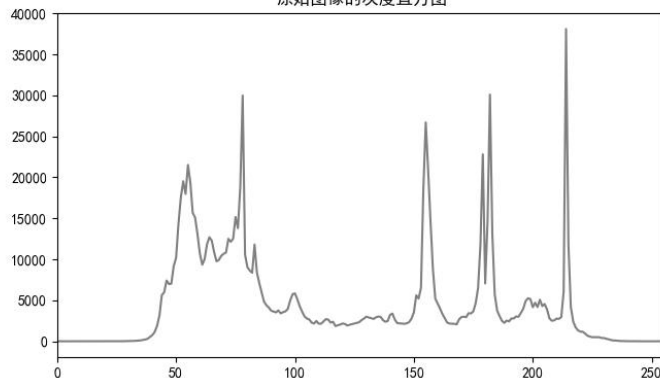
2、对一幅灰度图像进行直方图均衡化

在 OpenCV 中，直方图均衡化是一种增强图像对比度的技术，特别是对于灰度图像，它可以使亮度分布更加均匀。OpenCV 提供了一个非常简单的函数 `cv2.equalizeHist()` 来对灰度图像进行直方图均衡化。

原始灰度图像



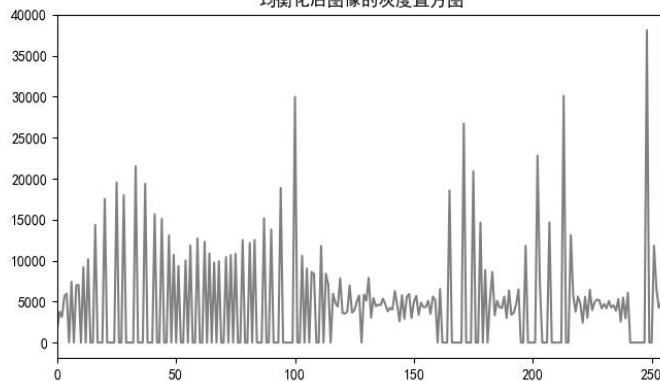
原始图像的灰度直方图



直方图均衡化后的图像



均衡化后图像的灰度直方图



● OpenCV中关于图像操作 (C++实现)

获取二值图像轮廓

`findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point());`

第一个参数: **image**表示输入的原图像，是一个单通道的二值图像

第二个参数: **contours**表示输出的轮廓集合，是`vector<vector<Point>>`类型的，每一个轮廓都用一系列的像素点`point (col,row)`表示出来

第三个参数: **hierarchy**表示返回的各个轮廓的拓扑结构，就是上面那个树形结构，遍历**contours**的时候就是靠着这个树形结构实现的，相当于先序遍历这个树，因此如果一个轮廓有内嵌轮廓，那么在遍历过当前轮廓之后，下一个遍历的轮廓一定是当前轮廓的内嵌轮廓，这也是二维码定位用这个函数实现的基础。数据格式`vector<vec4i>`,0~4分别存储后一个轮廓，前一个轮廓，父轮廓和内嵌轮廓的索引号，如果不存在就是-1。

- OpenCV中关于图像操作

图像缩放

`void cv::resize` (InputArray src, OutputArray dst, Size dsize, double fx = 0, double fy = 0, int interpolation =INTER_LINEAR)

图像镜像

`void cv::flip`(InputArray src, OutputArray dst, int flipCode)

$$\text{dst}_{ij} = \begin{cases} \text{src}_{\text{src.rows}-i-1,j} & \text{if } \text{flipCode} = 0 \\ \text{src}_{i,\text{src.cols}-j-1} & \text{if } \text{flipCode} > 0 \\ \text{src}_{\text{src.rows}-i-1,\text{src.cols}-j-1} & \text{if } \text{flipCode} < 0 \end{cases}$$

- OpenCV中关于图像操作

仿射变换 (目标校正)

`cv::warpAffine()`

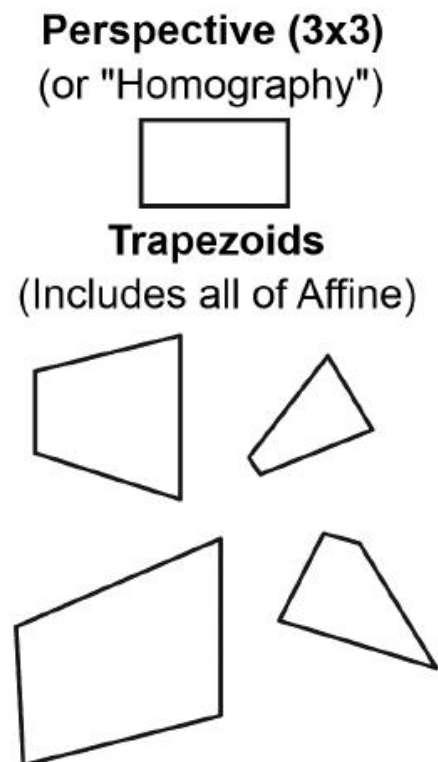
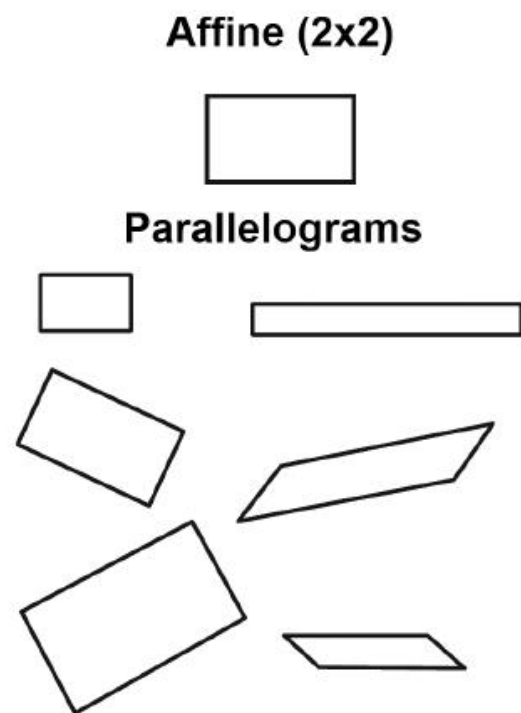


Figure 11-3. Affine and perspective transformations

● OpenCV中关于图像操作

仿射变换

```
void cv::warpAffine ( InputArray  src, OutputArray  dst, InputArray  M, Size  
dsize, int  flags = INTER_LINEAR, int  borderMode = BORDER_CONSTANT, const  
Scalar &  borderValue = Scalar() )
```

旋转矩阵

```
Mat cv::getRotationMatrix2D (Point2f center, double angle, double scale)
```

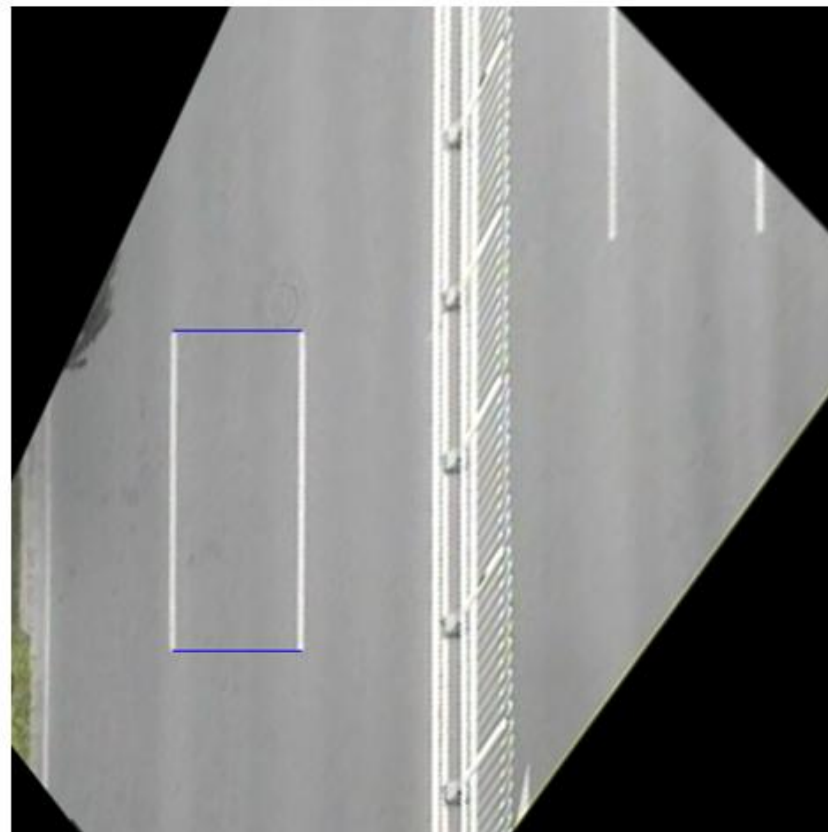
Ex1:

```
cv::Mat rot_mat = cv::getRotationMatrix2D(center, rotate_degree, 1.0);  
cv::warpAffine(img_src, img_rotate, rot_mat, src_sz);
```

Ex2:

```
cv::Rect2f bbox = cv::RotatedRect(cv::Point2f(), img_src.size(),  
rotate_degree).boundingRect();  
// adjust transformation matrix--translation transformation  
rot_mat.at<double>(0, 2) += bbox.width / 2.0 - img_src.cols / 2.0;  
rot_mat.at<double>(1, 2) += bbox.height / 2.0 - img_src.rows / 2.0;  
cv::warpAffine(img_src, img_urotate, rot_mat, bbox.size());
```

- OpenCV中关于图像操作



OpenCV图像读取示例

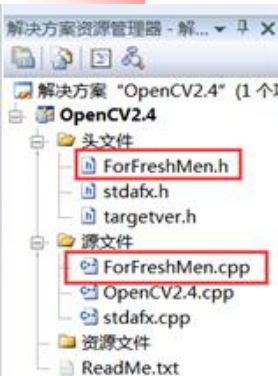
```
#include "stdafx.h"
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;

void Read_Show():
int _tmain(int argc, _TCHAR* argv[])
{
    Read_Show();
    return 0;
}

void Read_Show()
{
    const char* imagename = "boldt.jpg";
    //从文件中读入图像
    Mat img = imread(imagename);
    //如果读入图像失败
    if(img.empty())
    {
        fprintf(stderr, "Can not load image %s\n", imagename);
        exit(0);
    }
    //显示图像
    namedWindow("image", 1);
    cout<<"函数功能：读入并显示和保存一张图像"<<endl;
    imwrite("save.jpg", img);
    imshow("image", img);
    //此函数等待按键，按键盘任意键就返回
    waitKey(0);
}
```



OpenCV下采用类+结构体的框架进行编程



```

// OpenCV2.4.cpp : 定义控制台应用程序
//

#include "stdafx.h"
#include <opencv2/opencv.hpp>
#include "ForFreshMen.h"

using namespace std;
using namespace cv;

int main(int argc, char* argv[])
{
    CForFreshMen VarDemo;

    //函数功能：读入并显示和保存一张图像
    VarDemo.Read_Show();

    return 0;
}
    
```

```

OpenCV2.4.cpp* ForFreshMen.cpp ForFreshMen.h*
CForFreshMen class CForFreshMen
CForFreshMen

#ifndef _FORFRESHMEN_H_
#define _FORFRESHMEN_H_
#include <opencv2/opencv.hpp>
#include <string>

using namespace std;
using namespace cv;

class CForFreshMen
{
public:
    CForFreshMen(void);
    ~CForFreshMen(void);

    void Read_Show();
};
#endif
    
```

```

OpenCV2.4.cpp* ForFreshMen.cpp ForFreshMen.h*
ForFreshMen.cpp d:\OpenCV2.4\OpenCV2.4\F
(全局范围)

#include "ForFreshMen.h"

CForFreshMen::CForFreshMen()
{
}

CForFreshMen::~CForFreshMen()
{
}
    
```


OpenCV下采用类+结构体的框架进行编程

```
void CForFreshMen::Read_Show()  
{  
    const char* imagename = "boldt.jpg";  
    //从文件中读入图像  
    Mat img = imread(imagename);  
    //如果读入图像失败  
    if(img.empty())  
    {  
        fprintf(stderr, "Can not load image %s\n", imagename);  
        exit(0);  
    }  
    //显示图像  
    namedWindow("image",1);  
    cout<<"函数功能：读入并显示和保存一张图像"<<endl;  
    imwrite("save.jpg",img);  
    imshow("image", img);  
    //此函数等待按键，按键盘任意键就返回  
    waitKey(0);  
}
```

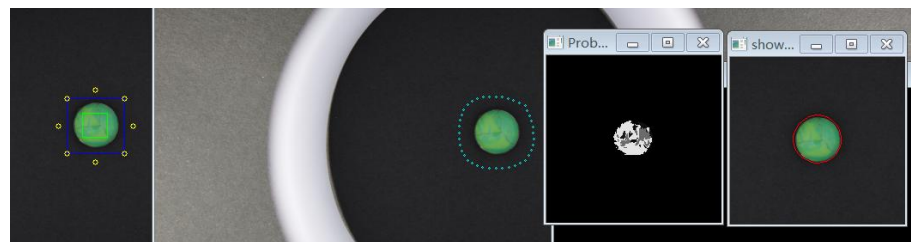
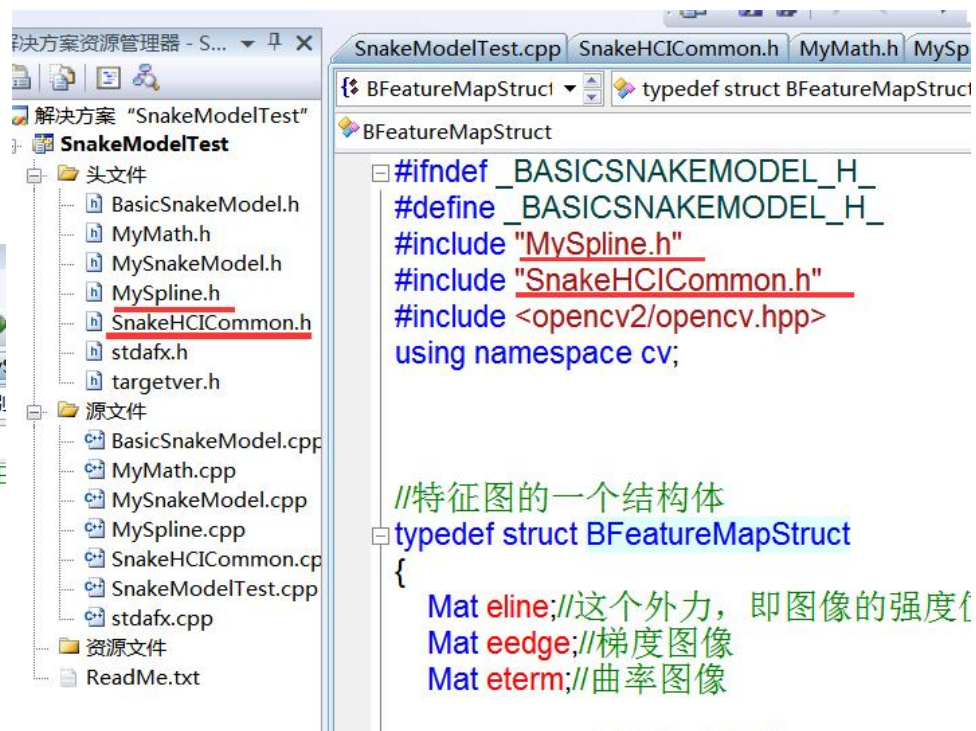
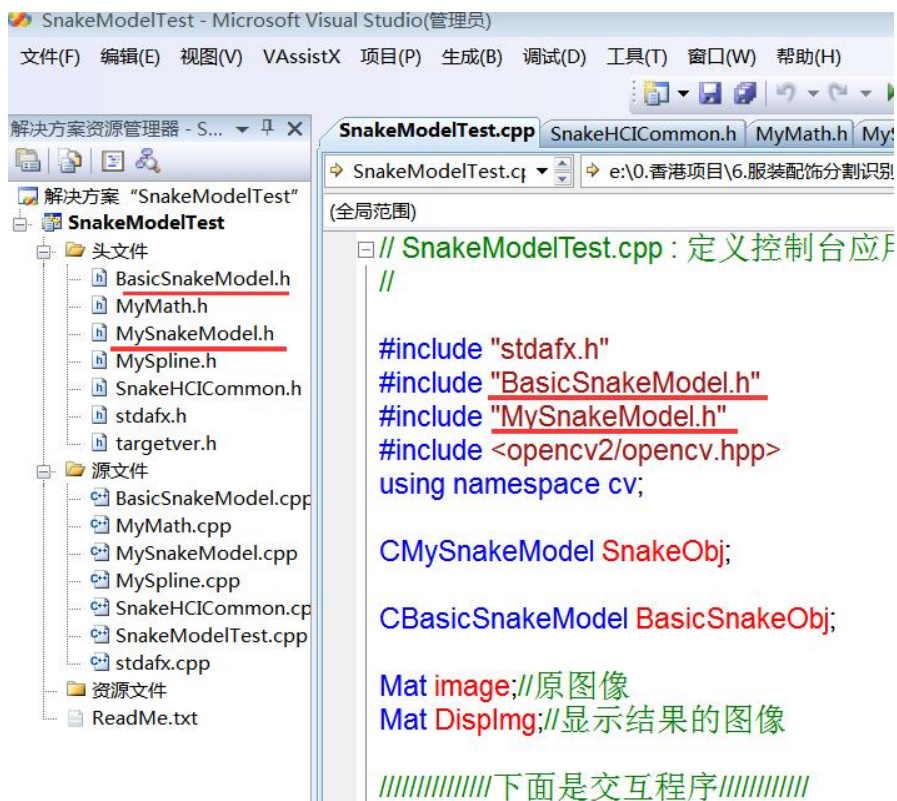


OpenCV下采用类+结构体的框架进行编程

International Journal of Computer Vision, 321-331 (1988)
© 1987 Kluwer Academic Publishers, Boston, Manufactured in The Netherlands

Snakes: Active Contour Models

MICHAEL KASS, ANDREW WITKIN, and DEMETRI TERZOPOULOS
Schlumberger Palo Alto Research, 3340 Hillview Ave., Palo Alto, CA 94304



一个蛇模型算法框架实例

遍历图像上所有像素（例如把图像左右翻转）

访问图像中像素的几种方法

1. 使用Mat_模板子类访问图像元素

```
Mat_<uchar> img=Mat::zeros(cvSize(300,200),CV_8U);
for(int j=0;j<img.rows/2;j++)
{
    for(int i=0;i<img.cols/2;i++)
    {
        img(j,i)=255;
    }
}
cout<<"//函数功能：使用Mat_模板子类更简洁访问图";
imshow("show",img);
cvWaitKey(0);
```

2. 以.at<>方式访问

```
cout<<"函数功能：给图片添加椒盐噪声"<<endl;
for(int k=0;k<n;k++)
{
    int i=rand()%image.cols;
    int j=rand()%image.rows;
    if(image.channels()==1)//灰度图
    {
        image.at<uchar>(j,i)=255;
    }
    else if(image.channels()==3)//彩色图
    {
        image.at<Vec3b>(j,i)[0]=255;
        image.at<Vec3b>(j,i)[1]=255;
        image.at<Vec3b>(j,i)[2]=255;
    }
}
```

3. 通过访问图像每行以及以in-place的方式访问

```
int nl=image.rows;//行数
//每行的元素个数
int nc=image.cols*image.channels();
int k=0;
cout<<"//函数功能：通过访问图像每行";
for(int j=0;j<nl;j++)
{
    //得到第j行的首地址
    uchar* data=image.ptr<uchar>(j);
    for(int i=0;i<nc;i++)
    {
        if(k==0)
        {
            int tt=data[i];
            cout<<tt<<endl;
        }
        //处理每一个像素
        //data[i]=data[i]/div*div+div/2;//慢速,
        *data++=*data/div*div+div/2;//快速
    }
    if(k==0)
    {
        int tt=data[i];
        cout<<tt<<endl;
        k++;
    }
}
```

4. 通过迭代器方式访问

```
//得到初始位置的迭代器
Mat_<Vec3b>::iterator it=image.begin<Vec3b>();
//得到终止位置的迭代器
Mat_<Vec3b>::iterator itend=image.end<Vec3b>();
cout<<"//函数功能：通过访问图像每行，以迭代器";
//遍历所有像素
for(;it!=itend;++it)
{
    //处理每个像素
    (*it)[0]=(*it)[0]/div*div+div/2;
    (*it)[1]=(*it)[1]/div*div+div/2;
    (*it)[2]=(*it)[2]/div*div+div/2;
}
```

5. 通过三行指针联合访问

```
//如有必要则分配图像
cout<<"//函数功能：锐化图像-通过三行指针"<<endl;
result.create(img.size(),img.type());
for(int j=1;j<img.rows-1;j++)//处理除了第一行和最后一行之外的所有行
{
    const uchar*previous=img.ptr<const uchar>(j-1);//上一行
    const uchar*current=img.ptr<const uchar>(j);//当前行
    const uchar*next=img.ptr<const uchar>(j+1);//下一行
    uchar*output=result.ptr<uchar>(j);//输出行

    for(int i=1;i<img.cols-1;i++)
    {
        *output++=saturate_cast<uchar>((5*current[i]-current[i-1]-current[i+1]-previous[i]-next[i]));
    }
}
//将未处理的像素设置为0
result.row(0).setTo(Scalar(0));
result.row(result.rows-1).setTo(Scalar(0));
result.col(0).setTo(Scalar(0));
result.col(result.cols-1).setTo(Scalar(0));
```


STL模板库

Vector中的删除操作

```
#include <list>
#include <vector>
using namespace std;
```

```
int main(int argc, char* argv[])
{
    //删除指定vector元素
    vector<int> intervectorlist;
    for(int i=0;i<10;i++)
    {
        intervectorlist.push_back(i);
    }
    vector<int>::iterator it_vector;
    for(it_vector=intervectorlist.begin();it_vector!=intervectorlist.end();)
    {
        if(*it_vector==2||*it_vector==7)
        {
            printf("Delete=%d\n",(*it_vector));
            it_vector=intervectorlist.erase(it_vector); //删除元素, 返回值指向已删除元素的下一个位置
            printf("删除结束\n");
        }
        else
        {
            it_vector++;
        }
    }
    //输出
    printf("删除vector\n");
    for(int i=0;i<intervectorlist.size();i++)
    {
        printf("%d\n",intervectorlist[i]);
    }
    printf("删除全部\n");
    vector<int>().swap(intervectorlist);
    for(int i=0;i<intervectorlist.size();i++)
    {
        printf("%d\n",intervectorlist[i]);
    }
    printf("\n");
}
```

List中的删除操作

```
//删除指定list元素
list<int> interlist;
for(int i=0;i<10;i++)
{
    interlist.push_back(i);
}
list<int>::iterator it;
for(it=interlist.begin();it!=interlist.end();)
{
    if(*it==2||*it==7)
    {
        printf("Delete=%d\n",(*it));
        interlist.erase(it++); //删除元素, 返回值指向已删除元素的下一个位置
        printf("删除结束\n");
    }
    else
    {
        it++;
    }
}
printf("删除全部\n");
if(interlist.size()>0)
{
    list<int>::iterator it;
    for(it=interlist.begin();it!=interlist.end();)
    {
        it=interlist.erase(it++);
    }
}
```

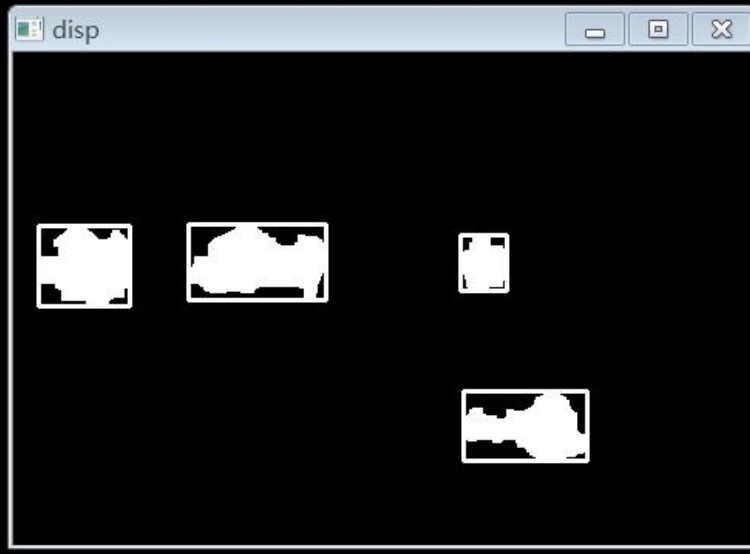
OpenCV获取图像中的轮廓，对图像中的目标进行计数



OpenCV获取图像中的轮廓，对图像中的目标进行计数

函数功能：求轮廓图，然后显示大小大于某阈值的那些图

```
i=1, size=42  
i=2, size=154  
i=3, size=3726  
i=4, size=54  
i=5, size=1147  
i=6, size=3180  
i=7, size=4500  
i=8, size=108  
i=9, size=42064
```



```
void CForFreshMen::ContourFunSomeSizeShow()
{
    cout<<"函数功能：求轮廓图，然后显示大小大于某阈值的那些图"<<endl;
    Mat image=imread("binaryGroup.bmp",0);
    Mat ShowImg (image.size(),CV_8U,Scalar(0));

    vector<vector<Point>>>contours;
    imshow("before",image);
    findContours(image, //注意，这个函数会修改image图像的内容
        contours, // a vector of contours
        CV_RETR_EXTERNAL, //retrieve the external contours 或者： CV_RETR_LIST //retrieve all contours
        CV_CHAIN_APPROX_NONE); // all pixels of each contours

    vector<vector<Point>>>contours_tmp;
    float T=1000;//矩形面积
    for(int i=0;i<contours.size();i++)
    {
        vector<Point> Points=contours[i];//如何获取一个包围框的点集
        Rect rect=boundingRect(Points);//如何获取该点集的外围框
        printf("i=%d, size=%d\n",i+1,rect.width*rect.height);
        if(rect.width*rect.height>T&&rect.width*rect.height<5000)
        {
            //存储
            contours_tmp.push_back(Points);

            //显示
            //显示指定的轮廓
            //方法1：使用离散点的方式画出目标的轮廓
            int i;
            for(i=0;i<Points.size()-1;i++)
            {
                line(ShowImg,Points[i],Points[i+1],Scalar(255),1,8,0);
            }
            line(ShowImg,Points[i],Points[Points.size()-1],Scalar(255),1,8,0);

            //方法2：可以按照以下格式，逐个填充轮廓
            vector<vector<Point>>>M;
            M.push_back(Points);
            fillPoly(ShowImg,M,Scalar(255));//画出填充结果

            //方法3：
            Rect b=boundingRect(Points);//如何获取该点集的外围框
            rectangle(ShowImg,b,Scalar(255,255,255),2);//画出框框
        }
    }
    imshow("disp",ShowImg);
    cvWaitKey(0);
}
```


上机练习内容：

1. OpenCV环境配置；
2. 实现对图像的读取；
3. 把一副彩色图像的三个通道变成3个单通道图像存储到硬盘上并显示；
4. 计算一幅单通道图像的直方图；
5. 编程实现对一幅单通道图像的边缘检测。
6. 对一幅灰度图像进行直方图均衡化
7. 对图像进行二值化操作；
8. 图像形态学操作；

上机练习内容（作业）：

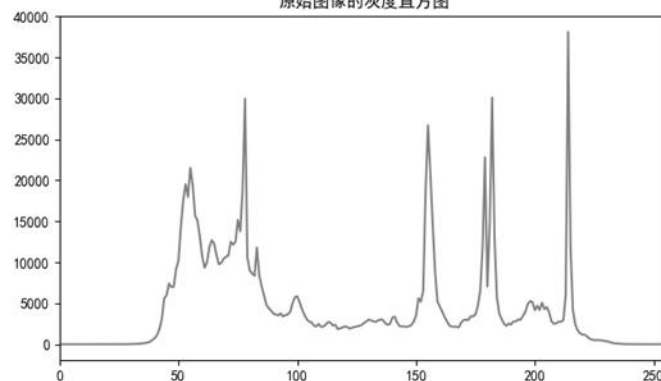
- 1、对一幅灰度图像进行直方图均衡化；
- 2、对比直方图均衡化前后图像和对应灰度直方图的变化；
- 3、思考题

对于对比度本来就较高的图像，直方图均衡化的效果会如何？

原始灰度图像



原始图像的灰度直方图



直方图均衡化后的图像



均衡化后图像的灰度直方图

