**test\_mat:**

//创建一个8\*8,位数为8的三通道Mat

cv::Mat mat2(8,8,CV\_8UC3);

//将Mat置0，使得矩阵内所有像素值为0

mat2.zeros(mat2.size(),mat2.type());

//输入一张图

cv::Mat mat3 = cv::imread("lena.jpg");

//修改这个图片Mat的使用尺寸

cv::resize(mat3,mat3,cv::Size(8,8));

//输入一张图

cv::Mat mat4 = cv::imread("lena.jpg");

//浅拷贝一个Mat，浅拷贝只数据地址是公用的，所以修改其中一个Mat，两个Mat的值都会被修改

cv::Mat mat4\_1 = mat4;

//垂直翻转图片

cv::flip(mat4\_1,mat4\_1,1);

//显示效果,两个Mat都被翻转了

cv::imshow("mat4",mat4);

cv::imshow("mat4\_1",mat4\_1);

cv::waitKey(0);// 将无限制地显示窗口直到任何按键

//输入一张图片

cv::Mat mat5 = cv::imread(".\\test5\\lena5.jpg");

//深拷贝一个Mat，数据完全复制，没有共用的部分，所以修改任意一个Mat，互不影响

cv::Mat mat5\_1;

mat5.copyTo(mat5\_1);

//上下翻转矩阵

cv::flip(mat5\_1,mat5\_1,0);

//输入一张图片

cv::Mat mat7 = cv::imread("lena.jpg");

//设置ROI位置

cv::Rect roi(mat7.cols/4, mat7.rows/4, mat7.cols/2, mat7.rows/2);

cv::Mat mat7\_roi = mat7(roi);

//垂直翻转ROI，这时候，对ROI的任意操作都是在修改原图的基础上操作的

cv::flip(mat7\_roi, mat7\_roi, 1);

//显示原图，ROI的部分被修改了

cv::imshow("mat7", mat7);

**test\_video:**

cv::VideoCapture capture(argv[1]);

//capture直接定义视频文件路径，如果路径设置为0，则读取电脑摄像头的数据

//cv::VideoCapture capture(0);

capture.isOpened()//为1 ，则video is open.

//获取当前视频帧率

double rate = capture.get(CV\_CAP\_PROP\_FPS); //feet per second.

//定义当前视频帧

cv::Mat frame;

//每一帧之间的延时，与视频的帧率相对应

int delay = 1000/rate;

capture.read(frame)//读取帧，并返回，有为1，无为0

cv::cvtColor(frame,frame,CV\_RGB2GRAY);//彩色变灰色

cv::imshow("video",frame);

// waitKey（25）将显示一个帧持续25毫秒之后显示将自动关闭。 （如果你把它放在循环中阅读视频，它将逐帧显示视频）

if(cv::waitKey(delay)>0) //没有帧了

//关闭视频，手动调用析构函数（非必须）

capture.release();

**test\_video\_set\_pos：**

//当前是第几帧

int frame\_num = capture.get(CV\_CAP\_PROP\_POS\_FRAMES);

//设置当前帧为第十帧

capture.set(CV\_CAP\_PROP\_POS\_FRAMES,10);

**test\_video\_save：**

//创建VideoWriter做保存视频

cv::VideoWriter writer;

//视频会被保存在VideoOutput.avi中

writer.open("VideoOutput.avi",CV\_FOURCC('M','J','P','G'),rate,cv::Size(frame.cols,frame.rows));

//写入保存视频

writer<<frame;

**test\_simple\_segment：**

//--2.转换灰度图

cv::Mat gray;

cv::cvtColor(image,gray,CV\_RGB2GRAY);

//--3.二值化灰度图

cv::Mat dst;

cv::threshold(gray,dst,60,255,cv::THRESH\_BINARY\_INV);

//平滑处理(平滑/高斯平滑两种)

//cv::blur(dst,dst,cv::Size(3,3));

cv::GaussianBlur(dst,dst,cv::Size(3,3),0,0); //高斯平滑

//形态学操作（此处效果不好，只用来展示加强理解）

cv::erode(dst,dst,cv::Mat(),cv::Point(-1,-1),1);//腐蚀

cv::dilate(dst,dst,cv::Mat(),cv::Point(-1,-1),1);//膨胀

//--4.创建一个新的图像

cv::Mat show\_image(cv::Size(2\*width,height),CV\_8UC3);

//将image拷贝到显示图片指定位置

image.copyTo(show\_image(cv::Rect(0,0,width,height)));

//将dst转换为3通道，使得show\_image和dst通道数一致，或者使用convertTo()函数做操作

cv::cvtColor(dst,dst,CV\_GRAY2RGB);

//将dst拷贝image指定位置

dst.copyTo(show\_image(cv::Rect(width,0,width,height)));

//保存图片

vector<int> compression\_params;

compression\_params.push\_back(CV\_IMWRITE\_JPEG\_QUALITY);

//在这个填入你要的图片质量，较高的值意味着更小的尺寸和更长的压缩时间

compression\_params.push\_back(100);

cv::imwrite("homework.jpg", mat1, compression\_params);

**test\_blurry：（几种滤波方法）**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 均值滤波 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//这种滤波方法就是取一个像素的邻域内各像素的平均值作为滤波结果

cv::Mat filter\_box;

//函数形式：blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType=BORDER\_DEFAULT )

//邻域大小为 5 \* 5

//cv::Point(-1, -1) 表明邻域的零位就是邻域的中心，这个是默认值，如果不改变的话可以不填。

//cv::BORDER\_DEFAULT 是对边界的处理办法，这个一般也不需要改变的。（变模糊）

cv::blur(image,filter\_box,cv::Size(5,5),cv::Point(-1,-1),cv::BORDER\_DEFAULT);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 高斯滤波 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//高斯滤波采取邻域内越靠近的值提供越大的权重的方式计算平均值。权重的选取采用高斯函数的形式。高斯函数有个非常好的特点，就是无论在时域还是频域都是钟形的。通过控制 σ 可以控制低通滤波的截止频率。

cv::Mat filter\_gaussian;

//函数形式:GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER\_DEFAULT )

//Size(3,3)定义了核的大小

//最后一个参数是高斯滤波的 σ 。从函数原型上可以看到有 sigmaX 和 sigmaY 两个参数。通常情况下 sigmaY 取与 sigmaX 相同的值，这时可以不写出来。也就是用它的默认值 0.

cv::GaussianBlur(image,filter\_gaussian,cv::Size(3,3),1.5);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 中值滤波 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//中值滤波是一种非线性滤波器。它是取邻域内各点的统计中值作为输出。这种滤波器可以有效的去除椒盐噪声。还能保持图像中各物体的边界不被模糊掉。是一种最常用的非线性滤波器。这种滤波器只能使用正方形的邻域。

cv::Mat filter\_median;

//函数形式:medianBlur(InputArray src, OutputArray dst, int ksize)

//ksize一般取奇数，例如3，5，7，滤波尺寸

cv::medianBlur(image,filter\_median,7);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 双边滤波 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//双边滤波的思想是抑制与中心像素值差别太大的像素，输出像素值依赖于邻域像素值的加权合

cv::Mat filter\_bila;

//函数形式：bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor, double sigmaSpace,int borderType=BORDER\_DEFAULT )；

//d 表示滤波时像素邻域直径，d为负时由 sigaColor计算得到；d>5时不能实时处理。

//sigmaColor、sigmaSpace非别表示颜色空间和坐标空间的滤波系数sigma。可以简单的赋值为相同的值。<10时几乎没有效果；>150时为油画的效果。

//borderType可以不指定。

cv::bilateralFilter(image,filter\_bila,1,10,10);

低通滤波器

作用：边缘平滑

常用低通滤波：高斯滤波、中值滤波、均值(方框)滤波、双边滤波

高通滤波器

作用：边缘提取和增强（因为边缘区域的灰度变换加大，也就是频率较高）

常用算子：Canny算子、Sobel算子、Laplace算子

线性滤波：输出像素是输入领域像素的算术关系。

非线性滤波例子：输出像素是输入领域像素的逻辑关系，例如中值滤波。

**test\_sharpening：**

//基于拉普拉斯算子的图像锐化（变清楚）

//拉普拉斯滤波核3\*3

// 0 -1 0

// -1 5 -1

// 0 -1 0

cv::Mat kernel = (cv::Mat\_<float>(3,3) << 0,-1,0,-1,5,-1,0,-1,0);

cv::Mat sharpen\_laplace;

cv::filter2D(image,sharpen\_laplace,image.depth(),kernel);

**test\_edge\_detect：**

//Sobel 算子是一个离散微分算子 (discrete differentiation operator)。

//它用来计算图像灰度函数的近似梯度。

cv::Sobel(src,dst\_sobel,src.depth(),1,1);

//Laplace算子

cv::Laplacian(src,dst\_laplace,src.depth());

//Canny算子

//Canny只处理灰度图

cv::cvtColor(src,dst\_canny,CV\_RGB2GRAY);

cv::Canny(dst\_canny,dst\_canny,50,150,3);

**test\_simple\_segment\_improve：**

//--4.寻找轮廓

std::vector<std::vector<cv::Point> > contours(10000);

//contours这里出现异常，可以手动分配内存空间大小contours(10000)

//因为findcontours函数会改变输入的图像，所以复制一个图像作为函数的输入

cv::findContours(binary\_copy,contours,CV\_RETR\_EXTERNAL/\*获取外轮廓\*/,

CV\_CHAIN\_APPROX\_NONE/\*获取每个轮廓的每个像素\*/, cv::Point());

//遍历每一个轮廓，把多余的轮廓去掉

std::vector<std::vector<cv::Point> >::const\_iterator it=contours.begin();

while(it!=contours.end()){

if(it->size()<500)

it = contours.erase(it);

else

++it;

}

//重新绘制轮廓

cv::Mat dst(image.size(),CV\_8U,cv::Scalar(0));

// Mat color; color.create(flow.rows, flow.cols, CV\_8UC3);

cv::drawContours(dst,contours,-1/\*绘制所有轮廓\*/,cv::Scalar(255)/\*绘制为白色\*/,CV\_FILLED/\*轮廓全部填充\*/);

**test\_get\_histogram ：**

//转化为灰度图

cv::cvtColor(image,image,CV\_RGB2GRAY);

//获取直方图

int histsize[] = {256}; //项的数量，也就是直方图的高

float range[] = {0, 256}; //像素值的范围

const float \*ranges[] = {range};

int channels[] = {0}; //通道

cv::Mat hist; //直方图结果

calcHist(&image, 1, channels, cv::Mat() /\*不使用图像作为掩码\*/, hist, 1, histsize, ranges);

//以图形方式显示直方图

//获取最大值和最小值

double maxVal = 0;

double minVal = 0;

cv::minMaxLoc(hist,&minVal,&maxVal,0,0);

//显示直方图的图像

cv::Mat hist\_img(histsize[0],histsize[0],CV\_8U,cv::Scalar(255));

//设置最高点为图高的90%

int high\_pt = static\_cast<int>(0.9\*histsize[0]);

//每个条目绘制一条垂直线

for(int h = 0; h<histsize[0]; h++){

float binVal = hist.at<float>(h);

//垂直线的高度

int intensity = static\_cast<int>(binVal\*high\_pt/maxVal);

//两点之间绘制一条线

cv::line(hist\_img,cv::Point(h,histsize[0]),cv::Point(h,histsize[0]-intensity),cv::Scalar::all(0));

}

cv::imshow("hist",hist\_img);

cv::waitKey(0);

**class4:**

cv::Mat R,G,B;

std::vector<cv::Mat>rgb;

//把图像分离成三个通道的数据

cv::split(image, rgb);

R = rgb[0];

G = rgb[1];

B = rgb[2];

HoughCircles：

利用 Hough 变换在灰度图像中找圆

void HoughCircles(

InputArray image, //输入 8-比特、单通道 (二值) 图像

OutputArray circles, //输出圆心坐标（x、y），和半径r

int method, //Hough 变换方式，目前只支持CV\_HOUGH\_GRADIENT

double dp, //累加器图像的分辨率。如果dp设置为1，则分辨率是相同的；如果设置为更大的值（比如2），累加器的分辨率受此影响会变小（此情况下为一半）。dp的值不能比1小。

double minDist, //让算法能明显区分的两个不同圆之间的最小距离。

double param1=100, //用于Canny的边缘阀值上限，下限被置为上限的一半。

double param2=100, //累加器的阀值。

int minRadius=0, //最小圆半径

int maxRadius=0 //最大圆半径，默认为最大值 max(image\_width, image\_height)

)

HoughLinesP：

通过统计概率的霍夫线变换找到线段

void HoughLinesP(

InputArray image, //输入 8-比特、单通道 (二值) 图像

OutputArray lines, //输出线段的两个端点，保存为(x\_1, y\_1, x\_2, y\_2)

double rho, //与象素相关单位的距离精度，((width + height) \* 2 + 1) / rho 表示r的空间范围,一般取1

double theta, //弧度测量的角度精度，（CV\_PI/theta）表示霍夫空间角度方向的大小

int threshold, //阈值参数。如果相应的累计值大于 threshold， 则函数返回的这个线段.

double minLineLength=0, //最小的线段长度

double maxLineGap=0 //这个参数表示在同一条直线上进行碎线段连接的最大间隔值(gap), 即当同一条直线上的两条碎线段之间的间隔小于maxLineGap时，将其合二为一

)

bgfg\_segm.cpp:背景前景分离

class6:

cornerHarris(Harris角点检测)

void cornerHarris(

InputArray src, //InputArray类型的src，输入图像，即源图像，填Mat类的对象即可，且需为单通道8位或者浮点型图像

OutputArray dst, //OutputArray类型的dst，函数调用后的运算结果存在这里，即这个参数用于存放Harris角点检测的输出结果，和源图片有一样的尺寸和类型

int blockSize, //int类型的blockSize，表示邻域的大小，更多的详细信息在cornerEigenValsAndVecs（）中有讲到。

int ksize, //int类型的ksize，表示Sobel()算子的孔径大小

double k, //double类型的k，Harris参数。

int borderType=BORDER\_DEFAULT //图像像素的边界模式，注意它有默认值BORDER\_DEFAULT。更详细的解释，参考borderInterpolate( )函数。

)

Shi-Tomasi强角点--Harris角点的改进

void goodFeaturesToTrack(

InputArray image, //输入图像，即源图像

OutputArray corners, //输出角点

int maxCorners, //最多检测到的角点数

double qualityLevel, //阈值系数

double minDistance, //角点间的最小距离

InputArray mask=noArray(),

int blockSize=3,

bool useHarrisDetector=false,   
double k=0.04

);

cornerEigenValsAndVecs--角点检测中计算扫描图像块的特征向量与特征值.

void cornerEigenValsAndVecs(

InputArray src, //输入单通道8-bit或者浮点类型图像

OutputArray dst, //用来存储结果的图像，大小与输入图像一致并且为CV\_32FC(6)类型

int blockSize, //邻域大小

int ksize, //Sobel算子当中的核大小，只能取1、3、5、7（具体可参考这里：Sobel()）

int borderType=BORDER\_DEFAULT //像素扩展的方法,因为在滤波处理的过程中会扩展图像边缘，每扩张一个边界像素，都需要计算出该像素点在原图中的位置

);

cornerMinEigenVal--在角点检测中计算梯度矩阵的最小特征值

cornerMinEigenVal(

InputArray src,

OutputArray dst, //用于存储最小特征值的图像

int blockSize,

int ksize=3,

int borderType=BORDER\_DEFAULT

);

//除了dst必须为CV\_32FC1类型以外，其它与cornerEigenValsAndVecs()函数的一致。

//创建一个轨迹栏并将其附加到指定的窗口。

createTrackbar( "Quality:", HarrisWindow, &HarrisQualityLevel, maxQualityLevel, HarrisFunction );