## 1 读取文件名

### #if……#endif

#if 0

code

#endif

(1)code中定义的是一些调试版本的代码，此时code完全被编译器忽略。如果想让code生效，只需把#if 0改成#if 1

(2)#if 0还有一个重要的用途就是用来当成注释，如果你想要注释的程序很长，这个时候#if 0是最好的，保证不会犯错误。（但是林锐的书上说千万不要把#if 0 来当作块注释使用） #if 1可以让其间的变量成为局部变量。

(3)这个结构表示你先前写好的code，现在用不上了，又不想删除，就用这个方法，比注释方便。

### #ifdef \_WIN32……#endif

一般情况下，源程序中所有的行都参加编译。但是有时希望对其中一部分内容只在满足一定条件才进行编译，也就是对一部分内容指定编译的条件，这就是“条件编译”。有时，希望当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句。

条件编译命令最常见的形式为：

#ifdef 标识符

程序段1

#else

程序段2

#endif

它的作用是：当标识符已经被定义过(一般是用#define命令定义)，则对程序段1进行编译，否则编译程序段2。

其中#else部分也可以没有，即：

#ifdef

程序段1

#denif

### string的相关操作

### ifndef/#define/#endif

#ifndef A\_H意思是"if not define a.h"  如果不存在a.h，接着的语句应该#define A\_H  就引入a.h，最后一句应该写#endif   否则不需要引

## 图像处理部分



### 2.1 colorMatch

//颜色匹配

//! 根据一幅图像与颜色模板获取对应的二值图

//! 输入RGB图像, 颜色模板（蓝色、黄色）

//! 输出灰度图（只有0和255两个值，255代表匹配，0代表不匹配）

### sobelyuchuli

均值滤波→sobel边缘检测→二值化→闭运算

### findContours

### vector<vector<Point> >::iterator

### RotatedRect

class CV\_EXPORTS RotatedRect

{

public:

*//构造函数*

RotatedRect();

RotatedRect(const Point2f& center, const Size2f& size, float angle);

RotatedRect(const CvBox2D& box);

*//!返回矩形的4个顶点*

void points(Point2f pts[]) const;

*//返回包含旋转矩形的最小矩形*

Rect boundingRect() const;

*//!转换到旧式的cvbox2d结构*

operator CvBox2D() const;

Point2f center; *//矩形的质心*

Size2f size; *//矩形的边长*

float angle; *//旋转角度，当角度为0、90、180、270等时，矩形就成了一个直立的矩形*

}

### minAreaRect

RotatedRect minAreaRect(InputArray **points**)

### *CV\_EXPORTS\_W* *Rect* *boundingRect*( *InputArray* points );

### template<typename \_Tp> class Rect\_

template<typename \_Tp> class *Rect\_*

{

public:

typedef \_Tp *value\_type*;

//! various constructors

*Rect\_*();

*Rect\_*(\_Tp \_x, \_Tp \_y, \_Tp \_width, \_Tp \_height);

*Rect\_*(const *Rect\_*& r);

*Rect\_*(const *CvRect*& r);

*Rect\_*(const *Point\_*<\_Tp>& org, const *Size\_*<\_Tp>& sz);

*Rect\_*(const *Point\_*<\_Tp>& pt1, const *Point\_*<\_Tp>& pt2);

*Rect\_*& operator = ( const *Rect\_*& r );

//! the top-left corner

*Point\_*<\_Tp> *tl*() const;

//! the bottom-right corner

*Point\_*<\_Tp> *br*() const;

//! size (width, height) of the rectangle

*Size\_*<\_Tp> size() const;

//! area (width\*height) of the rectangle

\_Tp area() const;

//! conversion to another data type

template<typename \_Tp2> operator *Rect\_*<\_Tp2>() const;

//! conversion to the old-style CvRect

operator *CvRect*() const;

//! checks whether the rectangle contains the point

bool *contains*(const *Point\_*<\_Tp>& pt) const;

\_Tp x, y, width, height; //< the top-left corner, as well as width and height of the rectangle

};

### RotatedRect

class *CV\_EXPORTS* *RotatedRect*

{

public:

//! various constructors

*RotatedRect*();

*RotatedRect*(const *Point2f*& center, const *Size2f*& size, float angle);

*RotatedRect*(const *CvBox2D*& box);

//! returns 4 vertices of the rectangle

void points(*Point2f* pts[]) const;

//! returns the minimal up-right rectangle containing the rotated rectangle

*Rect* *boundingRect*() const;

//! conversion to the old-style CvBox2D structure

operator *CvBox2D*() const;

*Point2f* center; //< the rectangle mass center

*Size2f* size; //< width and height of the rectangle

float angle; //< the rotation angle. When the angle is 0, 90, 180, 270 etc., the rectangle becomes an up-right rectangle.

};

### getRotationMatrix2D函数

主要用于获得图像绕着 某一点的旋转矩阵

函数调用形式：

Mat **getRotationMatrix2D**(Point2f center, double angle, double scale)

参数详解：

Point2f center：表示旋转的中心点

double angle：表示旋转的角度

double scale：图像缩放因子

### warpAffine

**warpAffine函数**

void cv::warpAffine ( InputArray src,

OutputArray dst,

InputArray M,

Size dsize,

int flags = INTER\_LINEAR,

int borderMode = BORDER\_CONSTANT,

const Scalar & borderValue = Scalar()

)

**参数解释**   
. src: 输入图像   
. dst: 输出图像，尺寸由dsize指定，图像类型与原图像一致   
. M: 2X3的变换矩阵   
. dsize: 指定图像输出尺寸   
. flags: 插值算法标识符，有默认值INTER\_LINEAR，如果插值算法为WARP\_INVERSE\_MAP, warpAffine函数使用如下矩阵进行图像转换

. borderMode: 边界像素模式，有默认值BORDER\_CONSTANT

. borderValue: 边界取值，有默认值Scalar()即0

### getRectSubPix函数

函数作用：

从原图像中提取提取一个感兴趣的矩形区域图像

函数调用形式：

C++: void getRectSubPix(InputArray image, Size patchSize, Point2f center, OutputArray patch, int patchType=-1 )

参数理解：

InputArray image：输入图像

Size patchSize：获取矩形的大小

Point2f center：获取的矩形在原图像中的位置

OutputArray patch：表示输出的图像

int patchType=-1 ：表示输出图像的深度

### assert函数

assert宏的原型定义在<assert.h>中，其作用是如果它的条件返回错误，则终止程序执行，原型定义：

#include <assert.h>

void assert( int expression );

assert的作用是现计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息，

然后通过调用 abort 来终止程序运行。

### GetAffineTransform：由三对点计算仿射变换

CvMat\* cvGetAffineTransform( const CvPoint2D32f\* src,const CvPoint2D32f\* dst, CvMat\* map\_matrix );

src：输入图像的三角形顶点坐标。

dst：输出图像的相应的三角形顶点坐标。

map\_matrix：指向2×3输出矩阵的指针。

函数cvGetAffineTransform计算满足以下关系的仿射变换矩阵：

这里,dst(i)= (x'i,y'i),src(i)= (xi,yi),i = 0..2.

### WarpAffine对图像做仿射变换

void cvWarpAffine( const CvArr\* src, CvArr\* dst, constCvMat\* map\_matrix,

                  int flags=CV\_INTER\_LINEAR+CV\_WARP\_FILL\_OUTLIERS,

                  CvScalar fillval=cvScalarAll(0) );

src：输入图像.

dst：输出图像.

map\_matrix：2×3 变换矩阵

flags：插值方法和以下开关选项的组合：

·       CV\_WARP\_FILL\_OUTLIERS - 填充所有输出图像的象素。如果部分象素落在输入图像的边界外，那么它们的值设定为 fillval.

·       CV\_WARP\_INVERSE\_MAP - 指定 map\_matrix是输出图像到输入图像的反变换，因此可以直接用来做象素插值。否则, 函数从 map\_matrix 得到反变换。

fillval：用来填充边界外面的值

函数 cvWarpAffine 利用下面指定的矩阵变换输入图像：http://hi.csdn.net/attachment/201110/19/0_1319025376b9L7.gif

* 如果没有指定 CV\_WARP\_INVERSE\_MAP ，http://hi.csdn.net/attachment/201110/19/0_1319025416j6qg.gif
* 否则， http://hi.csdn.net/attachment/201110/19/0_1319025445aaUB.gif

函数与 cvGetQuadrangleSubPix 类似，但是不完全相同。cvWarpAffine 要求输入和输出图像具有同样的数据类型，有更大的资源开销（因此对小图像不太合适）而且输出图像的部分可以保留不变。而 cvGetQuadrangleSubPix 可以精确地从8位图像中提取四边形到浮点数缓存区中，具有比较小的系统开销，而且总是全部改变输出图像的内容。要变换稀疏矩阵，使用 cxcore 中的函数 cvTransform 。

### resize函数

void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER\_LINEAR );

src：输入，原图像，即待改变大小的图像；

dst：输出，改变大小之后的图像，这个图像和原图像具有相同的内容，只是大小和原图像不一样而已；

dsize：输出图像的大小。如果这个参数不为0，那么就代表将原图像缩放到这个Size(width，height)指定的大小；如果这个参数为0，那么原图像缩放之后的大小就要通过下面的公式来计算：

dsize = Size(round(fx\*src.cols), round(fy\*src.rows))

其中，fx和fy就是下面要说的两个参数，是图像width方向和height方向的缩放比例。

fx：width方向的缩放比例，如果它是0，那么它就会按照(double)dsize.width/src.cols来计算；

fy：height方向的缩放比例，如果它是0，那么它就会按照(double)dsize.height/src.rows来计算；

interpolation：这个是指定插值的方式，图像缩放之后，肯定像素要进行重新计算的，就靠这个参数来指定重新计算像素的方式，有以下几种：

* **INTER\_NEAREST** - 最邻近插值
* **INTER\_LINEAR** - 双线性插值，如果最后一个参数你不指定，默认使用这种方法
* **INTER\_AREA** - resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire’-free results. But when the image is zoomed, it is similar to the INTER\_NEAREST method.
* **INTER\_CUBIC** - 4x4像素邻域内的双立方插值
* **INTER\_LANCZOS4** - 8x8像素邻域内的Lanczos插值

### Rect函数

Rect(int \_x,int \_y,int \_width,int \_height);   
参数意思为：左上角x坐标   
左上角y坐标   
矩形的宽   
矩形的高   
一般的用法为Rect g\_rectangle;   
g\_rectangle=Rect(a,b,c,d);

### sort函数

### assert函数

### boundingRect函数

boundingRect（）函数的使用方法

Calculates the up-right bounding rectangle of a point set.

C++: Rect boundingRect(InputArray points)

Parameters: points – Input 2D point set, stored in std::vector or Mat.

注意：（boundingRect读入的参数必须是vector或者Mat点集）

使用例程：

for( size\_t k = 0; k < contours.size(); k++ )

    {

    Rect ret1=boundingRect(Mat(contours[k]));//计算右上点集的边界矩形

    avgX = (ret1.x + ret1.x + ret1.width)/2; //运动物体的矩形的中点X位置

    avgY = (ret1.y + ret1.y + ret1.height)/2;//运动物体的矩形的中点Y位置

    cout<<"x:"<<avgX<<"y:"<<avgY<<endl;

     }

由上可以看出，boundingRect（）的用法

### addWeighted函数

void addWeighted(InputArray src1, double alpha, InputArray src2, double beta, double gamma, OutputArray dst, int dtype=-1)

第一个参数，InputArray类型的src1，表示需要加权的第一个数组，常常填一个Mat。

第二个参数，alpha，表示第一个数组的权重

第三个参数，src2，表示第二个数组，它需要和第一个数组拥有相同的尺寸和通道数。

第四个参数，beta，表示第二个数组的权重值。

第五个参数，dst，输出的数组，它和输入的两个数组拥有相同的尺寸和通道数。

第六个参数，gamma，一个加到权重总和上的标量值。看下面的式子自然会理解。

第七个参数，dtype，输出阵列的可选深度，有默认值-1。;当两个输入数组具有相同的深度时，这个参数设置为-1（默认值），即等同于src1.depth（）

addWeighted函数的作用可以被表示为为如下的矩阵表达式为：

dst = src1[I]\*alpha+ src2[I]\*beta + gamma;

### 函数

### 函数

### 函数

### 函数