# 透视变换

Mat srcImage, dstImage\_Aff, resImage;

//加¨®载?源¡ä图ª?像?并¡é作Á¡Â一°?些?初?始º?化¡¥

srcImage = imread("C:\\desktop\\xlib\\项?目?\\锦?润¨¨®汇?泽¨®相¨¤关?\\TuXiang\\Debug\\test.jpg", 1 );

if(!srcImage.data )

{

hlog("read img error");

return;

}

//指?定¡§结¨¢果?大ä¨®小?

resImage = Mat::zeros(500, 700, srcImage.type());

//投ª?影®¡ã变À?换?四?个?点Ì?的Ì?映®3射¦?关?系¦Ì

Point2f srcCorners[4];

Point2f resCorners[4];

//定¡§义°?一°?些?Mat变À?量¢?

Mat transMat( 3, 3, CV\_32FC1 ); //透ª?视º¨®变À?换?矩?阵¨®

//设¦¨¨置?源¡ä图ª?像?和¨ª目?标À¨º图ª?像?上¦?的Ì?四?组Á¨¦点Ì?以°?计?算?透ª?视º¨®变À?换?

srcCorners[0] = Point2f( 222,22 );

srcCorners[2] = Point2f( 869, 8 );

srcCorners[1] = Point2f( 5,271);

srcCorners[3]=Point2f(1082,259);

resCorners[0] = Point2f( 1,1 );

resCorners[2] = Point2f( resImage.cols, 1 );

resCorners[1] = Point2f(1, resImage.rows);

resCorners[3]=Point2f(resImage.cols,resImage.rows);

//求¨®得Ì?透ª?视º¨®变À?换?矩?阵¨®并¡é计?算?的Ì?透ª?视º¨®变À?换?

transMat = getPerspectiveTransform( srcCorners, resCorners );

//Output perspective transformation matrix

warpPerspective(srcImage, resImage, transMat,resImage.size());

imshow("origin",srcImage);

imshow("res", resImage );

将斜着的图倒过来

# 关于iplimage的origin参数

[http://hi.baidu.com/���Ϻ�/blog/item/e9f3be5ff63a6253fbf2c094.html](http://hi.baidu.com/%EF%BF%BD%EF%BF%BD%EF%BF%BD%CF%BA%EF%BF%BD/blog/item/e9f3be5ff63a6253fbf2c094.html)

|  |
| --- |
| 在使用opencv显示图像时会出现图像倒立的情况，IplImage的origin属性有关系。origin为0表示顶左结构，即图像的原点是左上角，如果为1为左下角。  一般从硬盘读入的图片或者通过cvCreateImage方法创建的IplImage图片默认的origin为0，即显示的时候都是正的。  而由摄像头或者视频文件获取的帧图像origin为1，此时显示的时候扫描顺序是从下到上，显示也是正的（opencv显示的时候是根据origin的值显示的，如果origin=1，则从下到上显示，否则反之）。  但是如果你自己创建了一个IplImage格式的图像img，且从帧图像中copy或者截取一部分区域进行显示的时候就会出现倒立情况。这是因为cvCreateImage方法得到的img的origin是0，而帧图像的origin为1，它会将帧图像的第i行赋值给img的第height-i行，因此就出现了倒立.解决办法是：在创建之后将origin调整为与帧图像的origin一致即可。           IplImage\* face=cvCreateImage(cvSize(width,height),copy\_Frame->depth,copy\_Frame->nChannels);          //因为IplImage的origin=0,所以要先将face->origin改为1         face->origin=copy\_Frame->origin;//1 |

# IplImage 与 Mat

**在OpenCV中， IplImage 与 Mat是可以相互转换的；**

 IplImage  转  Mat:

// extern  IplImage \* plpliamge; //假设 IplImage 已经创建；

 cv::Mat \* pmatImage = new cv:Mat( IplImage, 0  ): //第二个参数表示不进行像素数据copy；

Mat 转 IplImage:

//extern cv:Mat  matImage; //假设已经创建cv:Mat;

IplImage limage = IplImage ( matImage );//不进行数据copy；

通常情况对于图像的读取，IplImage 通过 cvLoadImage，   cv:Mat通过 cv::imread;

# 创建灰度图

Mat mt(200,200,CV\_8UC1);

Mat mt(200,200,CV\_8UC1,Scalar(112));初始化112

hlog(mt.depth());

hlog(mt.rows);

hlog(mt.cols);

imshow("test",mt);

# opencv2中鼠标截取图片

      鼠标操作属于用户接口设计，以前一直使用Qt来做，但是如果只需要简单的鼠标，键盘操作，直接调用opencv库的函数也未尝不可，鼠标操作之前已经接触很多了，在MFC，QT，OpenGL，等等中，理论主要就是两点，一是监控鼠标操作，鼠标点击，移动，松开，然后通过mouse\_event识别判断出那一种鼠标的操作，根据不同的操作然后进行处理，二是在主函数中加入鼠标的回调函数，将鼠标操作与程序的窗口绑定。

## 第一节 函数介绍

       暂时只接触了两个关于opencv2鼠标响应操作的函数，下面分别介绍一下:

### 1.1 回调函数

      opencv2.4.5中，提供的鼠标回调函数是 setMouseCallback，函数声明如下：

1. CV\_EXPORTS **void** setMouseCallback(**const** string& winname, MouseCallback onMouse, **void**\* userdata = 0);

     函数参数介绍

            const string& winname，windows视窗名称，对名为winname的视窗进行鼠标监控。

            MouseCallback onMouse，鼠标响应处理函数，监听鼠标的点击，移动，松开，判断鼠标的操作类型，并进行响应的函数处理。

            void\* userdata = 0 鼠标响应处理函数的ID，与鼠标相应处理函数相匹配就行，暂时只用到默认为0的情况。

     函数使用实例：

1. namedWindow("img");
2. setMouseCallback("img",on\_mouse,0);

### 1.2 鼠标响应处理函数

       opencv2.4.5中，鼠标相应处理函数一般默认形参和返回参数，函数形式如下：

1. **void** on\_mouse(**int** event,**int** x,**int** y,**int** flags,**void** \*ustc)

      函数参数介绍:

      int event，鼠标操作时间的整数代号，在opencv2.4.5中，event鼠标事件总共有10中，从0-9依次代表如下:

Event:

1. #define CV\_EVENT\_MOUSEMOVE 0             滑动
2. #define CV\_EVENT\_LBUTTONDOWN 1           左键点击
3. #define CV\_EVENT\_RBUTTONDOWN 2           右键点击
4. #define CV\_EVENT\_MBUTTONDOWN 3           中间点击
5. #define CV\_EVENT\_LBUTTONUP 4             左键释放
6. #define CV\_EVENT\_RBUTTONUP 5             右键释放
7. #define CV\_EVENT\_MBUTTONUP 6             中间释放
8. #define CV\_EVENT\_LBUTTONDBLCLK 7         左键双击
9. #define CV\_EVENT\_RBUTTONDBLCLK 8         右键双击
10. #define CV\_EVENT\_MBUTTONDBLCLK 9         中间释放

     int x,int y，代表鼠标位于窗口的（x，y）坐标位置，窗口左上角默认为原点，向右为x轴，向下为y轴，

     int flags，代表鼠标的拖拽事件，以及键盘鼠标联合事件，总共有32种事件，依次如下：

flags:

1. #define CV\_EVENT\_FLAG\_LBUTTON 1           左键拖拽
2. #define CV\_EVENT\_FLAG\_RBUTTON 2           右键拖拽
3. #define CV\_EVENT\_FLAG\_MBUTTON 4           中间拖拽
4. #define CV\_EVENT\_FLAG\_CTRLKEY 8     (8~15)按Ctrl不放事件
5. #define CV\_EVENT\_FLAG\_SHIFTKEY 16   (16~31)按Shift不放事件
6. #define CV\_EVENT\_FLAG\_ALTKEY 32       (32~39)按Alt不放事件（后面8-39还有待研究）

    void \*ustc，函数参数的编号（暂时用不到）

## 第二节 鼠标操作实例

### 2.1 示例程序代码

        程序如下，已经附上说明：

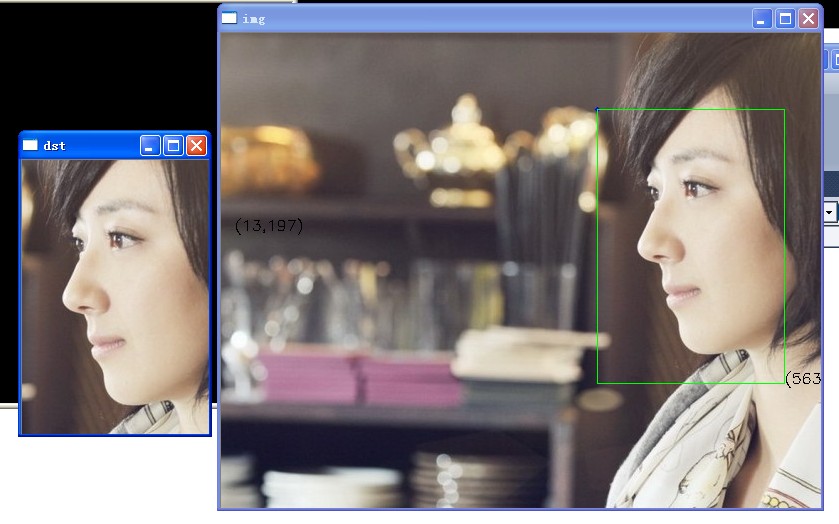
1. #include <opencv2/core/core.hpp>
2. #include <opencv2/highgui/highgui.hpp>
3. #include <stdio.h>
5. **using** **namespace** cv;
7. cv::Mat org,dst,img,tmp;
8. **void** on\_mouse(**int** event,**int** x,**int** y,**int** flags,**void** \*ustc)//event鼠标事件代号，x,y鼠标坐标，flags拖拽和键盘操作的代号
9. {
10. **static** Point pre\_pt = (-1,-1);//初始坐标
11. **static** Point cur\_pt = (-1,-1);//实时坐标
12. **char** temp[16];
13. **if** (event == CV\_EVENT\_LBUTTONDOWN)//左键按下，读取初始坐标，并在图像上该点处划圆
14. {
15. org.copyTo(img);//将原始图片复制到img中
16. sprintf(temp,"(%d,%d)",x,y);
17. pre\_pt = Point(x,y);
18. putText(img,temp,pre\_pt,FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,0,255),1,8);//在窗口上显示坐标
19. circle(img,pre\_pt,2,Scalar(255,0,0,0),CV\_FILLED,CV\_AA,0);//划圆
20. imshow("img",img);
21. }
22. **else** **if** (event == CV\_EVENT\_MOUSEMOVE && !(flags & CV\_EVENT\_FLAG\_LBUTTON))//左键没有按下的情况下鼠标移动的处理函数
23. {
24. img.copyTo(tmp);//将img复制到临时图像tmp上，用于显示实时坐标
25. sprintf(temp,"(%d,%d)",x,y);
26. cur\_pt = Point(x,y);
27. putText(tmp,temp,cur\_pt,FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,0,255));//只是实时显示鼠标移动的坐标
28. imshow("img",tmp);
29. }
30. **else** **if** (event == CV\_EVENT\_MOUSEMOVE && (flags & CV\_EVENT\_FLAG\_LBUTTON))//左键按下时，鼠标移动，则在图像上划矩形
31. {
32. img.copyTo(tmp);
33. sprintf(temp,"(%d,%d)",x,y);
34. cur\_pt = Point(x,y);
35. putText(tmp,temp,cur\_pt,FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,0,255));
36. rectangle(tmp,pre\_pt,cur\_pt,Scalar(0,255,0,0),1,8,0);//在临时图像上实时显示鼠标拖动时形成的矩形
37. imshow("img",tmp);
38. }
39. **else** **if** (event == CV\_EVENT\_LBUTTONUP)//左键松开，将在图像上划矩形
40. {
41. org.copyTo(img);
42. sprintf(temp,"(%d,%d)",x,y);
43. cur\_pt = Point(x,y);
44. putText(img,temp,cur\_pt,FONT\_HERSHEY\_SIMPLEX,0.5,Scalar(0,0,0,255));
45. circle(img,pre\_pt,2,Scalar(255,0,0,0),CV\_FILLED,CV\_AA,0);
46. rectangle(img,pre\_pt,cur\_pt,Scalar(0,255,0,0),1,8,0);//根据初始点和结束点，将矩形画到img上
47. imshow("img",img);
48. img.copyTo(tmp);
49. //截取矩形包围的图像，并保存到dst中
50. **int** width = abs(pre\_pt.x - cur\_pt.x);
51. **int** height = abs(pre\_pt.y - cur\_pt.y);
52. **if** (width == 0 || height == 0)
53. {
54. printf("width == 0 || height == 0");
55. **return**;
56. }
57. dst = org(Rect(min(cur\_pt.x,pre\_pt.x),min(cur\_pt.y,pre\_pt.y),width,height));
58. namedWindow("dst");
59. imshow("dst",dst);
60. waitKey(0);
61. }
62. }
63. **void** main()
64. {
65. org = imread("1.jpg");
66. org.copyTo(img);
67. org.copyTo(tmp);
68. namedWindow("img");//定义一个img窗口
69. setMouseCallback("img",on\_mouse,0);//调用回调函数
70. imshow("img",img);
71. cv::waitKey(0);
72. }

### 2.2 程序运行结果分析

     程序源图形：



   程序运行结果：



## 第三节，参考资料

         既然参考别人的博文，当然要把源资料介绍给大家，让大家参考一下：

       1. [在OpenCV中利用鼠标绘制矩形和截取图像的矩形区域](http://blog.csdn.net/quarryman/article/details/6435527)

                                        http://blog.csdn.net/quarryman/article/details/6435527

      2.[OpenCV响应鼠标函数cvSetMouseCallback（）和其副程式onMouse（）的使用（OpenCV2.4.5）](http://blog.csdn.net/glb562000520/article/details/8938582)

<http://blog.csdn.net/glb562000520/article/details/8938582>

## 全局变量保存位置

若在mainwindow中使用全局变量保存位置，全局变量必须在cpp中，不然就出重定义错误