一 GUI指令

1.窗口管理

cvNamedWindow("win1",CV\_WINDOW\_AUTOSIZE) //创建一个新窗口

cvMoveWindow("win1",100,100) //定位一个新窗口，offset from the UL corner of the screen

IplImage\* img=0;

img=cvLoadImage(fileName,CV\_LOAD\_IMAGE\_COLOR); //载入图像

cvShowImage("win1",img); //显示图像

该函数可以在上面建立的窗口（win1）中显示彩色或灰度的字节型/浮点型图像。字节型图像像素值范围为[0-255]；浮点型图像像素值范围为[0-1]。彩色图像的三色元素按BGR（蓝-绿-红）顺序存储

cvDestroyWindow("win1"); //关闭窗口

cvResizeWindow("win1",100,100); //改变窗口大小 new width/heigh in pixels

2.输入处理

（1）鼠标处理程序

void mouseHandler(int event,int x,int y,int flags,void\*param)

{

switch(event)

{

case CV\_EVENT\_LBUTTONDOWN:

if(flags&CV\_EVENT\_FLAG\_CTRLKEY)

break;

case CV\_EVENT\_LBUTTONUP:

break;

}

}

注册该事件处理程序：

mouseParam=5;

cvSetMouseCallback(“windowName”,mouseHandler,&mouseParam); //第三个参数可以设置为NULL.

(2)键盘事件：

对键盘输入没有专门的事件处理程序。

Int key; key = cvWaitKey(10); //wait 10ms for input

中止程序等待键盘输入： int key;key = cvWaitKey(0); //wait indefinitely for input

键盘输入循环处理程序：

While(1)

{

Key = cvWaitKey(10);

If(key ==’q’)

Break;

Switch(key)

{

Case ‘h’:

....

Break;

Case ‘i’:

...

Break;

}

}

1. 处理滑动条事件：

定义一个滑动条处理程序：

Void trackbarHandler(int pos)

{

Printf(“trackbar position:%d\n”,pos);

}

注册该事件处理程序

Int trackbarVal = 25;

Int maxVal = 100;

cvCreateTrackbar(“barName”,”windowName”,&trackbarVal,maxVal,trackbarHandler);

获取当前滑动条位置：

Int pos = cvGetTrackbarPos(“barName”,”windowName”);

设置滑动条位置：

cvSetTrackbarPos(“barName”,”windowName”,nPos);

二 OpenCV的基本数据结构

1. 图像数据结构

IPL图像：

IplImage:  
 int nChannels; //颜色通道数目（1,2,3,4）

Int depth; //像素的位深 IPL\_DEPTH\_8U,IPL\_DEPTH\_8S,IPL\_DEPTH\_32F

Int width; //图像宽度 (像素为单位)

Int height; //图像高度

Char\* imageData; //图像数据指针 ;注意彩色图像按RGB顺序存储数据

Int dataOrder; //0--将像素点不同通道的值交错排在一起，形成单一像素平面；

//1--把所有像素同通道值排在一起，形成若干个通道平面，再把平面排列起来。

//cvCreateImage 智能创建像素交错排列式的图像

Int origin; //0--像素远点为左上角；1--像素远点为左下角。

Int widthStep; //相邻行的同列点之间的字节数 width\*nChannels\*depth/8

Int imageSize;//图像的大小（字节为单位）=height\*widthStep

Struct \_IplROI \*roi; //图像的感兴趣区域（ROI）。ROI为非空时对图像的，处理仅限于ROI区域。

Char \*imageDataOrigin; //图像数据未对齐时的数据原点指针//需要正确重新分配图像内存

Int align; //图像数据的行对齐 4 or 8 byte alignment //OpenCV中用widthStep代替

Char colorModel[4]; //颜色模型 OpenCV中忽略此项。

1. 矩阵与相量
2. 矩阵：

CvMat: //2D矩阵

Int type; //元素类型（uchar,short,int,float,double）与标志

Int step; //整行长度字节

Int rows,cols; //行，列数

Int height,width; //矩阵高度，宽度，与rows、cols对应

Union data;

uchar\* ptr; //data pointer for an unsigned char matrix

short\* s; // data pointer for a short matrix

int\* i; // data pointer for an integer matrix

float\* fl; // data pointer for a float matrix

double\* db; // data pointer for a double matrix

CvMatND //N维矩阵

Int type; //元素类型

Int dims; //矩阵维数

Union data;

uchar\* ptr; //data pointer for an unsigned char matrix

short\* s; // data pointer for a short matrix

int\* i; // data pointer for an integer matrix

float\* fl; // data pointer for a float matrix

double\* db; // data pointer for a double matrix

Struct dim[]; //各维信息

Size; //元素数目

Step; //元素间距（字节为单位）

CvSparseMat //N-维稀疏矩阵

1. 一般矩阵

CvArr\* //仅作为函数定义的参数使用

1. 标量

CvScalar

Double val[4]; //4D 向量

初始化函数：

CvScalar s = cvScalar(double val0,double val1=0,double val2=0,double val3=0);

CvScalar s = cvScalar(20.0);

1. 其它结构类型
2. 点：

CvPoint p = cvPoint(int x,int y);

CvPoint2D32f p = cvPoint2D32f(float x, float y);

CvPoint3D32f p = cvPoint3D32f(float x,float y,float z);

1. 矩形框大小（以像素位精度）：

CvSize r = cvSize(int width,int height);

CvSize2D32f r = cvSize2D32f(float width,float height);

1. 矩形框的偏置和大小：

CvRect r = cvRect(int x,int y,int width,int height);

三 图像处理

1.图像的内存分配与释放

1. 分配内存给一幅新图像：

IplImage\* cvCreateImage(CvSize size,int depth,int channels);

size: cvSize(width,height);

depth: 像素深度: IPL\_DEPTH\_8U, IPL\_DEPTH\_8S, IPL\_DEPTH\_16U,IPL\_DEPTH\_16S, IPL\_DEPTH\_32S, IPL\_DEPTH\_32F, IPL\_DEPTH\_64F

channels: 像素通道数. Can be 1, 2, 3 or 4.各通道是交错排列的. 一幅彩色图像的数据排列格式如下：

b0 g0 r0 b1 g1 r1 ...

Eg:

IplImage\* img1 = cvCreageImage(cvSize(640,480),IPL\_DEPTH\_8U,1);// Allocate a 1-channel byte image

IplImage\* img2 = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_32F,3);// Allocate a 3-channel float image

1. 释放图像：

IplImage\* img = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

cvReleaseImage(&img);

1. 复制图像：

IplImage\* img1 = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

IplImage\* img2;

Img2 = cvCloneImage(img1); // 注意通过cvCloneImage得到的图像 也要用 cvReleaseImage 释放，否则容易产生内存泄漏

1. 设置/获取感兴趣区域ROI：

Void cvSetImageROI(IplImage\* image,CvRect rect);

Void cvResetImageROI(IplImage\* image);

CvRect cvGetImageROI(const IplImage\* image); 大多数OpenCV函数都支持 ROI, region of interests，设置了ROI之后，函数对图片的处理只会应用到ROI区域上.

1. 图像读写
2. 从文件中读入图像：

IplImage\* img=0;

img=cvLoadImage(fileName);

if(!img) printf("Could not load image file: %s\n",fileName);

支持的图像格式: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM,SR, RAS, TIFF, TIF

OpenCV默认将读入的图像强制转换为一幅三通道彩色图像. 不过可以按以下方法修改读入方式:

img=cvLoadImage(fileName,flag);

flag: >0 将读入的图像强制转换为一幅三通道彩色图像

=0 将读入的图像强制转换为一幅单通道灰度图像

<0 读入的图像通道数与所读入的文件相同.

（2） 保存图像:

if(!cvSaveImage(outFileName,img))

printf("Could not save: %s\n", outFileName);

保存的图像格式由 outFileName 中的扩展名确定.

3、访问图像像素

（1） 假设你要访问第k通道、第i行、第j列的像素。

（2） 间接访问: (通用，但效率低，可访问任意格式的图像)

对于单通道字节型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

CvScalar s;

s=cvGet2D(img,i,j); // get the (j,i) pixel value, 注意cvGet2D与cvSet2D中坐标参数的顺序

与其它opencv函数坐标参数顺序恰好相反.本函数中i代表y轴，即height;j代表x轴，即width.

printf("intensity=%f\n",s.val[0]);

s.val[0]=111;

cvSet2D(img,i,j,s); // set the (j,i) pixel value

对于多通道字节型/浮点型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_32F,3);

CvScalar s;

s=cvGet2D(img,i,j); // get the (j,i) pixel value

printf("B=%f, G=%f, R=%f\n",s.val[0],s.val[1],s.val[2]);

s.val[0]=111;

s.val[1]=111;

s.val[2]=111;

cvSet2D(img,i,j,s); // set the (j,i) pixel value

（3） 直接访问: (效率高，但容易出错)

对于单通道字节型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

((uchar \*)(img->imageData + i\*img->widthStep))[j]=111;

对于多通道字节型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,3);

((uchar \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 0]=111; // B

((uchar \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 1]=112; // G

((uchar \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 2]=113; // R

对于多通道浮点型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_32F,3);

((float \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 0]=111; // B

((float \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 1]=112; // G

((float \*)(img->imageData + i\*img->widthStep))[j\*img->nChannels + 2]=113; // R

（4） 基于指针的直接访问: (简单高效)

对于单通道字节型图像:

IplImage\* img = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

int height = img->height;

int width = img->width;

int step = img->widthStep;

uchar\* data = (uchar \*)img->imageData;

data[i\*step+j] = 111;

对于多通道字节型图像:

IplImage\* img = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,3);

int height = img->height;

int width = img->width;

int step = img->widthStep;

int channels = img->nChannels;

uchar\* data = (uchar \*)img->imageData;

data[i\*step+j\*channels+k] = 111;

对于多通道浮点型图像（假设图像数据采用4字节（32位）行对齐方式）:

IplImage\* img = cvCreateImage(cvSize(640,480),IPL\_DEPTH\_32F,3);

int height = img->height;

int width = img->width;

int step = img->widthStep;

int channels = img->nChannels;

float \* data = (float \*)img->imageData;

data[i\*step+j\*channels+k] = 111;

（5） 基于 c++ wrapper 的直接访问: （更简单高效）

首先定义一个 c++ wrapper ‘Image’，然后基于Image定义不同类型的图像:

template<class T> class Image

{

private:

IplImage\* imgp;

public:

Image(IplImage\* img=0) {imgp=img;}

~Image(){imgp=0;}

void operator=(IplImage\* img) {imgp=img;}

inline T\* operator[](const int rowIndx) {

return ((T \*)(imgp->imageData + rowIndx\*imgp->widthStep));}

};

typedef struct{

unsigned char b,g,r;

} RgbPixel;

typedef struct{

float b,g,r;

} RgbPixelFloat;

typedef Image<RgbPixel> RgbImage;

typedef Image<RgbPixelFloat> RgbImageFloat;

typedef Image<unsigned char> BwImage;

typedef Image<float> BwImageFloat;

对于单通道字节型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,1);

BwImage imgA(img);

imgA[i][j] = 111;

对于多通道字节型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_8U,3);

RgbImage imgA(img);

imgA[i][j].b = 111;

imgA[i][j].g = 111;

imgA[i][j].r = 111;

对于多通道浮点型图像:

IplImage\* img=cvCreateImage(cvSize(640,480),IPL\_DEPTH\_32F,3);

RgbImageFloat imgA(img);

imgA[i][j].b = 111;

1. 图像转换
2. 字节型图像的灰度-彩色转换:

cvConvertImage(src,dst,flags=0);

Src = float/byte grayscale/color image

Dst = byte grayscale/color image

Flags = CV\_CVTIMG\_FLIP(垂直翻转图像)

CV\_CVTIMG\_SWAP\_RB(置换R和B通道)

1. 彩色图像->灰度图像：

cvCvtColor(cimg,gimg,CV\_BGR2GRAY);// cimg -> gimg

// Using a direct conversion:

for(i=0;i<cimg->height;i++) for(j=0;j<cimg->width;j++)

gimgA[i][j]= (uchar)(cimgA[i][j].b\*0.114 +

cimgA[i][j].g\*0.587 +

cimgA[i][j].r\*0.299);

（3） 不同彩色空间之间的转换:

cvCvtColor(src,dst,code); // src -> dst

code = CV\_<X>2<Y>

<X>/<Y> = RGB, BGR, GRAY, HSV, YCrCb, XYZ, Lab, Luv, HLS

e.g.: CV\_BGR2GRAY, CV\_BGR2HSV, CV\_BGR2Lab

5、绘图指令

（1） 绘制矩形:

// 在点 (100,100) 和 (200,200) 之间绘制一矩形，边线用红色、宽度为 1

cvRectangle(img, cvPoint(100,100), cvPoint(200,200), cvScalar(0,0,255), 1);

（2） 绘制圆形:

// 圆心为(100,100)、半径为20. 圆周绿色、宽度为1

cvCircle(img, cvPoint(100,100), 20, cvScalar(0,255,0), 1);

（3） 绘制线段:

// 在 (100,100) 和 (200,200) 之间、线宽为 1 的绿色线段

cvLine(img, cvPoint(100,100), cvPoint(200,200), cvScalar(0,255,0), 1);

（4） 绘制一组线段:

CvPoint curve1[]={10,10, 10,100, 100,100, 100,10};

CvPoint curve2[]={30,30, 30,130, 130,130, 130,30, 150,10};

CvPoint\* curveArr[2]={curve1, curve2};

int nCurvePts[2]={4,5};

int nCurves=2;

int isCurveClosed=1;

int lineWidth=1;

cvPolyLine(img,curveArr,nCurvePts,nCurves,isCurveClosed,cvScalar(0,255,255),lineWidth);

void cvPolyLine( CvArr\* img, CvPoint\*\* pts, int\* npts, int contours, int is\_closed,

CvScalar color, int thickness=1, int line\_type=8, int shift=0 );

img 图像。

pts 折线的顶点指针数组。

npts 折线的定点个数数组。也可以认为是pts指针数组的大小

contours 折线的线段数量。

is\_closed 指出多边形是否封闭。如果封闭，函数将起始点和结束点连线。

color 折线的颜色。

thickness 线条的粗细程度。

line\_type 线段的类型。参见cvLine。

shift 顶点的小数点位数

（5） 绘制一组填充颜色的多边形:

cvFillPoly(img,curveArr,nCurvePts,nCurves,cvScalar(0,255,255));

cvFillPoly用于一个单独被多边形轮廓所限定的区域内进行填充。函数可以填充复杂的区域,

例如，有漏洞的区域和有交叉点的区域等等。

void cvFillPoly( CvArr\* img, CvPoint\*\* pts, int\* npts, int contours,CvScalar color, int line\_type=8, int shift=0 );

img 图像。

pts 指向多边形的数组指针。

npts 多边形的顶点个数的数组。

contours 组成填充区域的线段的数量。

color 多边形的颜色。

line\_type 组成多边形的线条的类型。

shift 顶点坐标的小数点位数。

（6） 文本标注:

CvFont font;

double hScale=1.0;

double vScale=1.0;

int lineWidth=1;

cvInitFont(&font,CV\_FONT\_HERSHEY\_SIMPLEX|CV\_FONT\_ITALIC, hScale,vScale,0,lineWidth);

cvPutText (img,"My comment",cvPoint(200,400), &font, cvScalar(255,255,0));

其它可用的字体类型有: CV\_FONT\_HERSHEY\_SIMPLEX, CV\_FONT\_HERSHEY\_PLAIN, CV\_FONT\_HERSHEY\_DUPLEX,

CV\_FONT\_HERSHEY\_COMPLEX, CV\_FONT\_HERSHEY\_TRIPLEX, CV\_FONT\_HERSHEY\_COMPLEX\_SMALL,

CV\_FONT\_HERSHEY\_SCRIPT\_SIMPLEX, CV\_FONT\_HERSHEY\_SCRIPT\_COMPLEX,

四、矩阵处理

1、矩阵的内存分配与释放

（1） 总体上:

OpenCV 使用C语言来进行矩阵操作。不过实际上有很多C++语言的替代方案可以更高效地完成。

在OpenCV中向量被当做是有一个维数为1的N维矩阵.

矩阵按行-行方式存储，每行以4字节（32位）对齐.

（2） 为新矩阵分配内存:

CvMat\* cvCreateMat(int rows, int cols, int type);

type: 矩阵元素类型.

按CV\_<bit\_depth>(S|U|F)C<number\_of\_channels> 方式指定. 例如: CV\_8UC1 、CV\_32SC2.

示例:

CvMat\* M = cvCreateMat(4,4,CV\_32FC1);

（3） 释放矩阵内存:

CvMat\* M = cvCreateMat(4,4,CV\_32FC1);

cvReleaseMat(&M);

（4） 复制矩阵:

CvMat\* M1 = cvCreateMat(4,4,CV\_32FC1);

CvMat\* M2;

M2=cvCloneMat(M1);

（5） 初始化矩阵:

double a[] = { 1, 2, 3, 4,

5, 6, 7, 8,

9, 10, 11, 12 };

CvMat Ma=cvMat(3, 4, CV\_64FC1, a);

//等价于:

CvMat Ma;

cvInitMatHeader(&Ma, 3, 4, CV\_64FC1, a);

（6） 初始化矩阵为单位矩阵:

CvMat\* M = cvCreateMat(4,4,CV\_32FC1);

cvSetIdentity(M); // does not seem to be working properly

2、访问矩阵元素

（1） 假设需要访问一个2D浮点型矩阵的第（i, j）个单元.

（2） 间接访问:

cvmSet(M,i,j,2.0); // Set M(i,j)

t = cvmGet(M,i,j); // Get M(i,j)

（3） 直接访问（假设矩阵数据按4字节行对齐）:

CvMat\* M = cvCreateMat(4,4,CV\_32FC1);

int n = M->cols;

float \*data = M->data.fl;

data[i\*n+j] = 3.0;

（4） 直接访问（当数据的行对齐可能存在间隙时 possible alignment gaps）:

CvMat\* M = cvCreateMat(4,4,CV\_32FC1);

int step = M->step/sizeof(float);

float \*data = M->data.fl;

(data+i\*step)[j] = 3.0;

（5） 对于初始化后的矩阵进行直接访问:

double a[16];

CvMat Ma = cvMat(3, 4, CV\_64FC1, a);

a[i\*4+j] = 2.0; // Ma(i,j)=2.0;

3、矩阵/向量运算

（1） 矩阵之间的运算:

CvMat \*Ma, \*Mb, \*Mc;

cvAdd(Ma, Mb, Mc); // Ma+Mb -> Mc

cvSub(Ma, Mb, Mc); // Ma-Mb -> Mc

cvMatMul(Ma, Mb, Mc); // Ma\*Mb -> Mc

（2） 矩阵之间的元素级运算:

CvMat \*Ma, \*Mb, \*Mc;

cvMul(Ma, Mb, Mc); // Ma.\*Mb -> Mc

cvDiv(Ma, Mb, Mc); // Ma./Mb -> Mc

cvAddS(Ma, cvScalar(-10.0), Mc); // Ma.-10 -> Mc

（3） 向量乘积:

double va[] = {1, 2, 3};

double vb[] = {0, 0, 1};

double vc[3];

CvMat Va=cvMat(3, 1, CV\_64FC1, va);

CvMat Vb=cvMat(3, 1, CV\_64FC1, vb);

CvMat Vc=cvMat(3, 1, CV\_64FC1, vc);

double res=cvDotProduct(&Va,&Vb); // 向量点乘: Va . Vb -> res

cvCrossProduct(&Va, &Vb, &Vc); // 向量叉乘: Va x Vb -> Vc

注意在进行叉乘运算时，Va, Vb, Vc 必须是仅有3个元素的向量.

（4） 单一矩阵的运算:

CvMat \*Ma, \*Mb;

cvTranspose(Ma, Mb); // 转置：transpose(Ma) -> Mb (注意转置阵不能返回给Ma本身)

CvScalar t = cvTrace(Ma); // 迹：trace(Ma) -> t.val[0]

double d = cvDet(Ma); // 行列式：det(Ma) -> d

cvInvert(Ma, Mb); // 逆矩阵：inv(Ma) -> Mb

（5） 非齐次线性方程求解:

CvMat\* A = cvCreateMat(3,3,CV\_32FC1);

CvMat\* x = cvCreateMat(3,1,CV\_32FC1);

CvMat\* b = cvCreateMat(3,1,CV\_32FC1);

cvSolve(&A, &b, &x); // solve (Ax=b) for x

（6） 特征值与特征向量 (矩阵为方阵):

CvMat\* A = cvCreateMat(3,3,CV\_32FC1);

CvMat\* E = cvCreateMat(3,3,CV\_32FC1);

CvMat\* l = cvCreateMat(3,1,CV\_32FC1);

cvEigenVV(A, E, l); // l = A 的特征值(递减顺序)

// E = 对应的特征向量 (行向量)

（7） 奇异值分解（SVD）:====

CvMat\* A = cvCreateMat(3,3,CV\_32FC1);

CvMat\* U = cvCreateMat(3,3,CV\_32FC1);

CvMat\* D = cvCreateMat(3,3,CV\_32FC1);

CvMat\* V = cvCreateMat(3,3,CV\_32FC1);

cvSVD(A, D, U, V, CV\_SVD\_U\_T|CV\_SVD\_V\_T); // A = U D V^T

标志位使矩阵U或V按转置形式返回 (若不转置可能运算出错).

五、视频处理

1、从视频流中捕捉一帧画面

（1） OpenCV 支持从摄像头或视频文件（AVI格式）中捕捉帧画面.

（2） 初始化一个摄像头捕捉器:

CvCapture\* capture = cvCaptureFromCAM(0); // capture from video device #0

（3） 初始化一个视频文件捕捉器:

CvCapture\* capture = cvCaptureFromAVI("infile.avi");

（4） 捕捉一帧画面:

IplImage\* img = 0;

if(!cvGrabFrame(capture)){ // capture a frame

printf("Could not grab a frame\n\7");

exit(0);

}

img=cvRetrieveFrame(capture); // retrieve the captured frame

若要从多个摄像头中同步捕捉画面，则须首先从每个摄像头中抓取一帧，紧接着要将被捕捉的帧画面

恢复到一个IplImage\*型图像中。（译注：这一过程其实可以用 cvQueryFrame() 函数一步完成）

（5） 释放视频流捕捉器:

cvReleaseCapture(&capture);

注意由视频流捕捉器得到的图像是由捕捉器分配和释放内存的，不需要单独对图像进行释放内存的操作。

2、获取/设置视频流信息

（1） 获取视频流设备信息:

cvQueryFrame(capture); // 在读取视频流信息前，要先执行此操作

int frameH = (int) cvGetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_HEIGHT);

int frameW = (int) cvGetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_WIDTH);

int fps = (int) cvGetCaptureProperty(capture, CV\_CAP\_PROP\_FPS);

int numFrames = (int) cvGetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_COUNT);

统计总帧数仅对视频文件有效，但似乎不太准确（译注：也许OpenCV2.0中此问题已解决）

（2） 获取帧图信息:

float posMsec = cvGetCaptureProperty(capture, CV\_CAP\_PROP\_POS\_MSEC);

int posFrames = (int) cvGetCaptureProperty(capture, CV\_CAP\_PROP\_POS\_FRAMES);

float posRatio = cvGetCaptureProperty(capture, CV\_CAP\_PROP\_POS\_AVI\_RATIO);

所抓取的帧的位置有三种表达方式：距离第一帧画面的时间间隔（毫秒为单位）, 或者距离第一帧画面

（序列号为0）的序列数. 第三种方式是按相对比率，第一帧的相对比率为0，最后一帧的相对比率为1.

此方式仅对读取视频文件时有效.

（3） 设置从视频文件抓取的第一帧画面的位置:

// start capturing from a relative position of 0.9 of a video file

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_POS\_AVI\_RATIO, (double)0.9);

注意此方法定位并不准确。

3、保存视频文件

（1） 初始化视频编写器:

CvVideoWriter \*writer = 0;

int isColor = 1;

int fps = 25; // or 30

int frameW = 640; // 744 for firewire cameras

int frameH = 480; // 480 for firewire cameras

writer=cvCreateVideoWriter("out.avi",CV\_FOURCC('P','I','M','1'),

fps,cvSize(frameW,frameH),isColor);

其它的编码器代号包括: CV\_FOURCC('P','I','M','1') = MPEG-1 codec CV\_FOURCC('M','J','P','G')

= motion-jpeg codec (does not work well) CV\_FOURCC('M', 'P', '4', '2') = MPEG-4.2 codec

CV\_FOURCC('D', 'I', 'V', '3') = MPEG-4.3 codec CV\_FOURCC('D', 'I', 'V', 'X') = MPEG-4 codec

CV\_FOURCC('U', '2', '6', '3') = H263 codec CV\_FOURCC('I', '2', '6', '3') = H263I codec

CV\_FOURCC('F', 'L', 'V', '1') = FLV1 codec 若编码器代号为 -1，则运行时会弹出一个编码器选择框.

（2） 保存视频文件:

IplImage\* img = 0;

int nFrames = 50;

for(i=0;i<nFrames;i++){

cvGrabFrame(capture); // capture a frame

img=cvRetrieveFrame(capture); // retrieve the captured frame

// img = cvQueryFrame(capture);

cvWriteFrame(writer,img); // add the frame to the file

}

要查看所抓取到的帧画面，可以在循环中加入以下语句:

cvShowImage("mainWin", img);

key=cvWaitKey(20); // wait 20 ms

注意 cvWaitKey 参数应该不小于 20 ms，否则画面的显示可能出错.

（3） 释放视频编写器:

cvReleaseVideoWriter(&writer);