up previous

Introduction to programming with OpenCV

Gady Agam

Department of Computer Science

January 27, 2006

Illinois Institute of Technology

Abstract:

The purpose of this document is to get you started quickly with OpenCV without having to go through lengthy reference manuals. Once you understand these basics you will be able to consult the OpenCV manuals on a need basis.

Contents

- Introduction
  - Description of OpenCV
  - Resources
  - OpenCV naming conventions
  - Compilation instructions
  - Example C Program
- GUI commands
  - Window management
  - Input handling
- Basic OpenCV data structures
  - Image data structure
  - Matrices and vectors
  - Other data structures
- Working with images
  - Allocating and releasing images
  - Reading and writing images
  - Accessing image elements
  - Image conversion
  - Drawing commands

- Working with matrices
  - Allocating and releasing matrices
  - Accessing matrix elements
  - Matrix/vector operations

- Working with video sequences
  - Capturing a frame from a video sequence
  - Getting/setting frame information
  - Saving a video file

Introduction

Description of OpenCV

- General description
  - Open source computer vision library in C/C++.
  - Optimized and intended for real-time applications.
  - OS/hardware/window-manager independent.
  - Generic image/video loading, saving, and acquisition.
  - Both low and high level API.
  - Provides interface to Intel's Integrated Performance Primitives (IPP) with processor specific optimization (Intel processors).

- Features:
  - Image data manipulation (allocation, release, copying, setting, conversion).
  - Image and video I/O (file and camera based input, image/video file output).
  - Matrix and vector manipulation and linear algebra routines (products, solvers, eigenvalues, SVD).
  - Various dynamic data structures (lists, queues, sets, trees, graphs).
  - Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).
  - Structural analysis (connected components, contour processing, distance transform, various moments, template matching, Hough transform, polygonal approximation, line fitting, ellipse fitting, Delaunay triangulation).
  - Camera calibration (finding and tracking calibration patterns, calibration, fundamental matrix estimation, homography estimation, stereo correspondence).
  - Motion analysis (optical flow, motion segmentation, tracking).
  - Object recognition (eigen-methods, HMM).
  - Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).
  - Image labeling (line, conic, polygon, text drawing)

- OpenCV modules:
  - *cv* - Main OpenCV functions.
  - *cvaux* - Auxiliary (experimental) OpenCV functions.
  - *cxcore* - Data structures and linear algebra support.
  - *highgui* - GUI functions.

Resources

- Reference manuals:
  - `<opencv-root>/docs/index.htm`
- Web resources:
  - Official webpage: `http://www.intel.com/technology/computing/opencv/`
  - Software download: `http://sourceforge.net/projects/opencvlibrary/`
- Books:
  - Open Source Computer Vision Library by Gary R. Bradski, Vadim Pisarevsky, and Jean-Yves Bouguet, Springer, 1st ed. (June, 2006).
- Sample programs for video processing (in `<opencv-root>/samples/c/`):
  - color tracking: `camshiftdemo`
  - point tracking: `lkdemo`
  - motion segmentation: `motempl`
  - edge detection: `laplace`
- Sample programs for image processing (in `<opencv-root>/samples/c/`):
  - edge detection: `edge`
  - segmentation: `pyramid_segmentation`
  - morphology: `morphology`
  - histogram: `demhist`
  - distance transform: `distrans`
  - ellipse fitting: `fitellipse`

OpenCV naming conventions

- Function naming conventions:

```
cvActionTargetMod(...)

Action = the core functionality (e.g. set, create)
Target = the target image area (e.g. contour, polygon)
Mod    = optional modifiers (e.g. argument type)
```

- Matrix data types:

```
CV_<bit_depth>(S|U|F)C<number_of_channels>

S = Signed integer
U = Unsigned integer
F = Float

E.g.: CV_8UC1 means an 8-bit unsigned single-channel matrix,
      CV_32FC2 means a 32-bit float matrix with two channels.
```

- Image data types:

```
IPL_DEPTH_<bit_depth>(S|U|F)

E.g.: IPL_DEPTH_8U means an  8-bit unsigned image.
      IPL_DEPTH_32F means a 32-bit float image.
```

- Header files:

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
#include <cxcore.h>   // unnecessary - included in cv.h
```

Compilation instructions

- Linux:

```
g++ hello-world.cpp -o hello-world \
    -I /usr/local/include/opencv -L /usr/local/lib  \
    -lm -lcv -lhighgui -lcvaux
```

- Windows:

```
In the project preferences set the path to the OpenCV header files and
the path to the OpenCV library files.
```

Example C Program

```
//////////////////////////////////////////////////////////////////////
//
// hello-world.cpp
//
// This is a simple, introductory OpenCV program. The program reads an
// image from a file, inverts it, and displays the result.
//
//////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>


int main(int argc, char *argv[])
{
  IplImage* img = 0;
  int height,width,step,channels;
  uchar *data;
  int i,j,k;

  if(argc<2){
    printf("Usage: main <image-file-name>\n\7");
    exit(0);
  }

  // load an image
```

```
  img=cvLoadImage(argv[1]);
  if(!img){
    printf("Could not load image file: %s\n",argv[1]);
    exit(0);
  }

  // get the image data
  height    = img->height;
  width     = img->width;
  step      = img->widthStep;
  channels  = img->nChannels;
  data      = (uchar *)img->imageData;
  printf("Processing a %dx%d image with %d channels\n",height,width,channels);

  // create a window
  cvNamedWindow("mainWin", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("mainWin", 100, 100);

  // invert the image
  for(i=0;i<height;i++) for(j=0;j<width;j++) for(k=0;k<channels;k++)
    data[i*step+j*channels+k]=255-data[i*step+j*channels+k];

  // show the image
  cvShowImage("mainWin", img );

  // wait for a key
  cvWaitKey(0);

  // release the image
  cvReleaseImage(&img );
  return 0;
}
```

GUI commands

Window management

- Create and position a window:

  ```
  cvNamedWindow("win1", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("win1", 100, 100); // offset from the UL corner of the screen
  ```

- Load an image:

  ```
  IplImage* img=0;
  img=cvLoadImage(fileName);
  if(!img) printf("Could not load image file: %s\n",fileName);
  ```

- Display an image:

  ```
  cvShowImage("win1",img);
  ```

  Can display a color or grayscale byte/float-image. A byte image is assumed to have values in the

range $[0..255]$. A float image is assumed to have values in the range $[0..1]$. A color image is assumed to have data in BGR order.

- Close a window:

```
cvDestroyWindow("win1");
```

- Resize a window:

```
cvResizeWindow("win1",100,100); // new width/heigh in pixels
```

Input handling

- Handle mouse events:
  - Define a mouse handler:

```
void mouseHandler(int event, int x, int y, int flags, void* param)
{
  switch(event){
    case CV_EVENT_LBUTTONDOWN:
      if(flags & CV_EVENT_FLAG_CTRLKEY)
        printf("Left button down with CTRL pressed\n");
      break;

    case CV_EVENT_LBUTTONUP:
      printf("Left button up\n");
      break;
  }
}

x,y:   pixel coordinates with respect to the UL corner

event: CV_EVENT_LBUTTONDOWN,   CV_EVENT_RBUTTONDOWN,   CV_EVENT_MBUTTONDOWN,
       CV_EVENT_LBUTTONUP,     CV_EVENT_RBUTTONUP,     CV_EVENT_MBUTTONUP,
       CV_EVENT_LBUTTONDBLCLK, CV_EVENT_RBUTTONDBLCLK, CV_EVENT_MBUTTONDBLCLK,
       CV_EVENT_MOUSEMOVE:

flags: CV_EVENT_FLAG_CTRLKEY, CV_EVENT_FLAG_SHIFTKEY, CV_EVENT_FLAG_ALTKEY,
       CV_EVENT_FLAG_LBUTTON, CV_EVENT_FLAG_RBUTTON,  CV_EVENT_FLAG_MBUTTON
```

  - Register the handler:

```
mouseParam=5;
cvSetMouseCallback("win1",mouseHandler,&mouseParam);
```

- Handle keyboard events:
  - The keyboard does not have an event handler.
  - Get keyboard input without blocking:

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

  - Get keyboard input with blocking:

```
int key;
key=cvWaitKey(0); // wait indefinitely for input
```

- The main keyboard event loop:

```
while(1){
  key=cvWaitKey(10);
  if(key==27) break;

  switch(key){
    case 'h':
      ...
      break;
    case 'i':
      ...
      break;
  }
}
```

- Handle trackbar events:
  - Define a trackbar handler:

```
void trackbarHandler(int pos)
{
  printf("Trackbar position: %d\n",pos);
}
```

  - Register the handler:

```
int trackbarVal=25;
int maxVal=100;
cvCreateTrackbar("bar1", "win1", &trackbarVal ,maxVal , trackbarHandler);
```

  - Get the current trackbar position:

```
int pos = cvGetTrackbarPos("bar1","win1");
```

  - Set the trackbar position:

```
cvSetTrackbarPos("bar1", "win1", 25);
```

Basic OpenCV data structures

Image data structure

- IPL image:

```
IplImage
  |-- int  nChannels;     // Number of color channels (1,2,3,4)
  |-- int  depth;         // Pixel depth in bits:
  |                       //   IPL_DEPTH_8U, IPL_DEPTH_8S,
  |                       //   IPL_DEPTH_16U,IPL_DEPTH_16S,
  |                       //   IPL_DEPTH_32S,IPL_DEPTH_32F,
  |                       //   IPL_DEPTH_64F
```

```
  |-- int  width;         // image width in pixels
  |-- int  height;        // image height in pixels
  |-- char* imageData;    // pointer to aligned image data
  |                       // Note that color images are stored in BGR order
  |-- int  dataOrder;     // 0 - interleaved color channels,
  |                       // 1 - separate color channels
  |                       // cvCreateImage can only create interleaved images
  |-- int  origin;        // 0 - top-left origin,
  |                       // 1 - bottom-left origin (Windows bitmaps style)
  |-- int  widthStep;     // size of aligned image row in bytes
  |-- int  imageSize;     // image data size in bytes = height*widthStep
  |-- struct _IplROI *roi;// image ROI. when not NULL specifies image
  |                       // region  to be processed.
  |-- char *imageDataOrigin; // pointer to the unaligned origin of image data
  |                          // (needed for correct image deallocation)
  |
  |-- int  align;         // Alignment of image rows: 4 or 8 byte alignment
  |                       // OpenCV ignores this and uses widthStep instead
  |-- char colorModel[4]; // Color model - ignored by OpenCV
```

Matrices and vectors

- Matrices:

```
CvMat                     // 2D array
  |-- int   type;         // elements type (uchar,short,int,float,double) and flags
  |-- int   step;         // full row length in bytes
  |-- int   rows, cols;   // dimensions
  |-- int   height, width; // alternative dimensions reference
  |-- 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-dimensional array
  |-- int   type;         // elements type (uchar,short,int,float,double) and flags
  |-- int   dims;         // number of array dimensions
  |-- 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[];       // information for each dimension
      |-- size;           // number of elements in a given dimension
      |-- step;           // distance between elements in a given dimension


CvSparseMat // SPARSE N-dimensional array
```

- Generic arrays:

```
CvArr*     // Used only as a function parameter to specify that the
```

```
            // function accepts arrays of more than a single type, such
            // as: IplImage*, CvMat* or even CvSeq*. The particular array
            // type is determined at runtime by analyzing the first 4
            // bytes of the header of the actual array.
```

- Scalars:

  ```
  CvScalar
    |-- double val[4]; //4D vector
  ```

  Initializer function:

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

  Example:

  ```
  CvScalar s = cvScalar(20.0);
  s.val[0]=10.0;
  ```

  Note that the initializer function has the same name as the data structure only starting with a lower case character. It is not a C++ constructor.

Other data structures

- Points:

  ```
  CvPoint      p = cvPoint(int x, int y);
  CvPoint2D32f p = cvPoint2D32f(float x, float y);
  CvPoint3D32f p = cvPoint3D32f(float x, float y, float z);

  E.g.:
  p.x=5.0;
  p.y=5.0;
  ```

- Rectangular dimensions:

  ```
  CvSize       r = cvSize(int width, int height);
  CvSize2D32f  r = cvSize2D32f(float width, float height);
  ```

- Rectangular dimensions with offset:

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

Working with images

Allocating and releasing images

- Allocate an image:

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

    size:  cvSize(width,height);
  ```

```
depth: pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U,
  IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F

channels: Number of channels per pixel. Can be 1, 2, 3 or 4. The channels
  are interleaved. The usual data layout of a color image is
  b0 g0 r0 b1 g1 r1 ...
```

Examples:

```
// Allocate a 1-channel byte image
IplImage* img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);

// Allocate a 3-channel float image
IplImage* img2=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
```

- Release an image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
cvReleaseImage(&img);
```

- Clone an image:

```
IplImage* img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
IplImage* img2;
img2=cvCloneImage(img1);
```

- Set/get the region of interest:

```
void  cvSetImageROI(IplImage* image, CvRect rect);
void  cvResetImageROI(IplImage* image);
vRect cvGetImageROI(const IplImage* image);

The majority of OpenCV functions support ROI.
```

- Set/get the channel of interest:

```
void cvSetImageCOI(IplImage* image, int coi); // 0=all
int cvGetImageCOI(const IplImage* image);

The majority of OpenCV functions do NOT support COI.
```

Reading and writing images

- Reading an image from a file:

```
IplImage* img=0;
img=cvLoadImage(fileName);
if(!img) printf("Could not load image file: %s\n",fileName);

Supported image formats: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM,
                         SR, RAS, TIFF, TIF
```

By default, the loaded image is forced to be a 3-channel color image. This default can be modified by using:

```
img=cvLoadImage(fileName,flag);

flag: >0 the loaded image is forced to be a 3-channel color image
      =0 the loaded image is forced to be a 1 channel grayscale image
      <0 the loaded image is loaded as is (with number of channels in the file).
```

- Writing an image to a file:

```
if(!cvSaveImage(outFileName,img)) printf("Could not save: %s\n",outFileName);
```

The output file format is determined based on the file name extension.

Accessing image elements

- Assume that you need to access the k-th channel of the pixel at the i-row and j-th column. The row index i is in the range $[0, \text{height} - 1]$. The column index j is in the range $[0, \text{width} - 1]$. The channel index k is in the range $[0, \text{nchannels} - 1]$.

- *Indirect access:* (General, but inefficient, access to any type image)

  - For a single-channel byte image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
CvScalar s;
s=cvGet2D(img,i,j); // get the (i,j) pixel value
printf("intensity=%f\n",s.val[0]);
s.val[0]=111;
cvSet2D(img,i,j,s); // set the (i,j) pixel value
```

  - For a multi-channel float (or byte) image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
CvScalar s;
s=cvGet2D(img,i,j); // get the (i,j) 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 (i,j) pixel value
```

- *Direct access:* (Efficient access, but error prone)

  - For a single-channel byte image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
((uchar *)(img->imageData + i*img->widthStep))[j]=111;
```

  - For a multi-channel byte image:

```
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
```

- For a multi-channel float image:

```
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
```

- *Direct access using a pointer:* (Simplified and efficient access under limiting assumptions)
  - For a single-channel byte image:

```
IplImage* img  = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
int height     = img->height;
int width      = img->width;
int step       = img->widthStep/sizeof(uchar);
uchar* data    = (uchar *)img->imageData;
data[i*step+j] = 111;
```

  - For a multi-channel byte image:

```
IplImage* img  = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
int height     = img->height;
int width      = img->width;
int step       = img->widthStep/sizeof(uchar);
int channels   = img->nChannels;
uchar* data    = (uchar *)img->imageData;
data[i*step+j*channels+k] = 111;
```

  - For a multi-channel float image (assuming a 4-byte alignment):

```
IplImage* img  = cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
int height     = img->height;
int width      = img->width;
int step       = img->widthStep/sizeof(float);
int channels   = img->nChannels;
float * data    = (float *)img->imageData;
data[i*step+j*channels+k] = 111;
```

- *Direct access using a c++ wrapper:* (Simple and efficient access)
  - Define a c++ wrapper for single-channel byte images, multi-channel byte images, and multi-channel float images:

```
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;
```

- For a single-channel byte image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
BwImage imgA(img);
imgA[i][j] = 111;
```

- For a multi-channel byte image:

```
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;
```

- For a multi-channel float image:

```
IplImage* img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
RgbImageFloat imgA(img);
imgA[i][j].b = 111;
imgA[i][j].g = 111;
imgA[i][j].r = 111;
```

Image conversion

- Convert to a grayscale or color byte-image:

```
cvConvertImage(src, dst, flags=0);

  src = float/byte grayscale/color image
  dst = byte grayscale/color image
  flags = CV_CVTIMG_FLIP     (flip vertically)
          CV_CVTIMG_SWAP_RB  (swap the R and B channels)
```

- Convert a color image to grayscale:

  Using the OpenCV conversion:

```
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);
```

- Convert between color spaces:

```
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
```

Drawing commands

- Draw a box:

```
// draw a box with red lines of width 1 between (100,100) and (200,200)
cvRectangle(img, cvPoint(100,100), cvPoint(200,200), cvScalar(255,0,0), 1);
```

- Draw a circle:

```
// draw a circle at (100,100) with a radius of 20. Use green lines of width 1
cvCircle(img, cvPoint(100,100), 20, cvScalar(0,255,0), 1);
```

- Draw a line segment:

```
// draw a green line of width 1 between (100,100) and (200,200)
cvLine(img, cvPoint(100,100), cvPoint(200,200), cvScalar(0,255,0), 1);
```

- Draw a set of polylines:

```
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);
```

- Draw a set of filled polygons:

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

- Add text:

```
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));
```

Other possible fonts:

```
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,
```

Working with matrices

Allocating and releasing matrices

- General:
  - OpenCV has a C interface to matrix operations. There are many alternatives that have a C++ interface (which is more convenient) and are as efficient as OpenCV.
  - Vectors are obtained in OpenCV as matrices having one of their dimensions as 1.
  - Matrices are stored row by row where each row has a 4 byte alignment.

- Allocate a matrix:

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

  type: Type of the matrix elements. Specified in form
  CV_<bit_depth>(S|U|F)C<number_of_channels>.  E.g.: CV_8UC1 means an
  8-bit unsigned single-channel matrix, CV_32SC2 means a 32-bit signed
  matrix with two channels.

  Example:
  CvMat* M = cvCreateMat(4,4,CV_32FC1);
```

- Release a matrix:

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
cvReleaseMat(&M);
```

- Clone a matrix:

```
CvMat* M1 = cvCreateMat(4,4,CV_32FC1);
CvMat* M2;
M2=cvCloneMat(M1);
```

- Initialize a matrix:

```
double a[] = { 1,  2,  3,  4,
               5,  6,  7,  8,
               9, 10, 11, 12 };

CvMat Ma=cvMat(3, 4, CV_64FC1, a);
```

Alternatively:

```
CvMat Ma;
cvInitMatHeader(&Ma, 3, 4, CV_64FC1, a);
```

- Initialize a matrix to identity:

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
cvSetIdentity(M); // does not seem to be working properly
```

Accessing matrix elements

- Assume that you need to access the (i,j) cell of a 2D float matrix.

- Indirect matrix element access:

```
cvmSet(M,i,j,2.0); // Set M(i,j)
t = cvmGet(M,i,j); // Get M(i,j)
```

- Direct matrix element access assuming a 4-byte alignment:

```
CvMat* M    = cvCreateMat(4,4,CV_32FC1);
int n       = M->cols;
float *data = M->data.fl;

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

- Direct matrix element access assuming 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;
```

- Direct matrix element access of an initialized matrix:

```
double a[16];
CvMat Ma = cvMat(3, 4, CV_64FC1, a);
a[i*4+j] = 2.0; // Ma(i,j)=2.0;
```

Matrix/vector operations

- Matrix-matrix operations:

```
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
```

- Elementwise matrix operations:

```
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
```

- Vector products:

```
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); // dot product:   Va . Vb -> res
cvCrossProduct(&Va, &Vb, &Vc);    // cross product: Va x Vb -> Vc
end{verbatim}
```

Note that Va, Vb, Vc, must be 3 element vectors in a cross product.

- Single matrix operations:

```
CvMat *Ma, *Mb;
cvTranspose(Ma, Mb);      // transpose(Ma) -> Mb (cannot transpose onto self)
CvScalar t = cvTrace(Ma); // trace(Ma) -> t.val[0]
double d = cvDet(Ma);     // det(Ma) -> d
cvInvert(Ma, Mb);         // inv(Ma) -> Mb
```

- Inhomogeneous linear system solver:

```
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
```

- Eigen analysis (of a symmetric matrix):

```
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 = eigenvalues of A (descending order)
                        // E = corresponding eigenvectors (rows)
```

- Singular value decomposition:

```
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
```

The flags cause U and V to be returned transposed (does not work well without the transpose flags).

Working with video sequences

Capturing a frame from a video sequence

- OpenCV supports capturing images from a camera or a video file (AVI).
- Initializing capture from a camera:

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

- Initializing capture from a file:

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

- Capturing a frame:

```
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
```

  To obtain images from several cameras simultaneously, first grab an image from each camera. Retrieve the captured images after the grabbing is complete.

- Releasing the capture source:

```
cvReleaseCapture(&capture);
```

  Note that the image captured by the device is allocated/released by the capture function. There is no need to release it explicitly.

Getting/setting frame information

- Get capture device properties:

```
cvQueryFrame(capture); // this call is necessary to get correct
                       // capture properties
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);
```

  The total frame count is relevant for video files only. It does not seem to be working properly.

- Get frame information:

```
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);
```

  Get the position of the captured frame in [msec] with respect to the first frame, or get its index where the first frame starts with an index of 0. The relative position (ratio) is 0 in the first frame and 1 in the last frame. This ratio is valid only for capturing images from a file.

- Set the index of the first frame to capture:

```
// start capturing from a relative position of 0.9 of a video file
cvSetCaptureProperty(capture, CV_CAP_PROP_POS_AVI_RATIO, (double)0.9);
```

This only applies for capturing from a file. It does not seem to be working properly.

Saving a video file

- Initializing a video writer:

```
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);
```

Other possible codec codes:

```
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
```

A codec code of -1 will open a codec selection window (in windows).

- Writing the video file:

```
IplImage* img = 0;
int nFrames = 50;
for(i=0;i<nFrames;i++){
  cvGrabFrame(capture);          // capture a frame
  img=cvRetrieveFrame(capture);  // retrieve the captured frame
  cvWriteFrame(writer,img);      // add the frame to the file
}
```

To view the captured frames during capture, add the following in the loop:

```
cvShowImage("mainWin", img);
key=cvWaitKey(20);           // wait 20 ms
```

Note that without the 20[msec] delay the captured sequence is not displayed properly.

- Releasing the video writer:

```
cvReleaseVideoWriter(&writer);
```

---

up previous

Gady Agam 2006-03-31