Technical University of Lodz
Institute of Electronics

Image Processing in Practice: Intel OpenCV

Adam Kozłowski/Aleksandra Królak

OpenCV vs Matlab

- Matlab
  - Expensive packages
  - Relatively slow performance
  - Limited possibility of creating stand-alone applications
- OpenCV
  - You can have it for free
  - Works fast
  - Easy programming – almost like Matlab ;-)

Introduction

- OpenCV – open source library for computer vision, developed by Intel
- Very large number of functions available
  - Image processing, pattern recognition, video analysis, motion tracking, etc…
- Very fast operation
  - Eg. face detection works in real time for a live feed from a webcam

Introduction

- **Core**
  - Data Structures
  - Methematical & Statistical Functions
- **CV**
  - Image Processing
  - Computer Vision
- **ML**
  - Classifiers
  - Neural Networks

Core – basic functions

- Data structures:
  - Points, images, ROI, COI, matrices, sparse…
- Mathematical functions:
  - Adding, subtracting, multiplication, scaling, conversion, logical functions, means, LUT…
  - Operations on matrices
- Statistical functions:
  - Sum, mean, deviation, min, max, histogram…

CV – main functions

- Image processing:
  - Interpolation
  - Affine, perspective transformation, log-polar...
  - Edge, line, corner detection
    (Sobel, Canny, Harris, Hough)
  - Convolution, median…
  - Morphological operations
  - Color space conversion
  - FFT, distance transform
  - Multiscale analysis (image segmantation by pyramids)

CV – main functions continued

- Computer vision:
  - Image segmentation, connected components (CC), contour analysis…
  - Template matching
  - Object detection and tracking
  - Optical Flow
  - Camera calibration and 3D reconstruction

Other functions

- GUI:
  - Displaying, sliders, opening and saving images and image sequences…
- Drawing:
  - Points, lines, ellipses, contours, text…
- CVCAM:
  - Streams: cameras and AVI files

Installation

Please refer to:

http://opencv.willowgarage.com/wiki

First program

```
#include <cxcore.h>
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvLoadImage(imagename,option):
  - option 0: load image as grayscale (even if colour)
  - option 1: load image as colour (even if grayscale)
- cvNamedWindow(windowname,option):
  - option 0: window can be resized
  - option 1: window cannot be resized (recommended)

Explanation…

- cvReleaseImage – always remember!
  - You might end up running out of memory!
- Now:
  - cvSomething – does something (action)
  - CvSomething – is something (variable)

Let’s smooth it.

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvSmooth(img,img,CV_BLUR,15,15);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvSmooth(source,destination,type,x,y):
  - Source and destination may be the same
  - Type: see manual, but CV_BLUR and CV_GAUSSIAN are standard
  - X and Y – size of the kernel in pixels

Let's brighten up!

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvAddS(img,cvScalar(127),img);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvAddS – adds a value to the image
- cvAddS(source,value,destination):
  - Source and destination may be the same
  - Value – must be a CvScalar type, therefore:
    - cvScalar command gives a CvScalar output ☺
    - For colour images you use three values:
    - cvScalar(128,64,32) for each channel

Adding contrast

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvConvertScale(img,img,2,-127);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvConvertScale – this command can convert 8bit images to 32bit etc, with additional scaling and shifting of image values
- cvConvertScale(source,destination,scale,shift):
  - Source and destination may be the same
  - Scale and shift values should be of type ‘double’

Decreasing contrast

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvConvertScale(img,img,0.5,64);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Controlling brightness - again

```
…
cvAddS(img,cvScalar(127),img);
…
//has the same effect as:
…
cvConvertScale(img,img,1,127);
…
//but i don’t really know which one is faster,
   probably AddS…
```

Flipping the image

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvFlip(img,img,1);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvFlip(source,destination,type):
  - Source and destination might be the same
  - Type:
    - 1 – flip vertical
    - 0 – flip horizontal
    - -1 – flip both

Thresholding an image

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
cvThreshold(img,img,127,255,CV_THRESH_BINARY);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(„Image”);
…
```

Explanation…

- Thresholding is a process of comparing the image with a specific value (threshold), so that we arrive with a logical image (0/1, false/true).
- cvThreshold(source, destination, threshold, maxvalue, type):
  - Maxvalue – the value of logical „1”, for 8bit images this should be 255.
  - Type – many types, see manual, but CV_THRESH_BINARY is a standard

More advanced stuff… ☺

- Now let's move on to methods requiring a declaration of more than one image
- This includes:
  - Image resizing
  - Adding images
  - Joining images side by side
  - Splitting RGB images to separate channel images
  - Etc…

Resizing images

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
IplImage *big = cvCreateImage(cvSize(512,512),8,1);
cvResize(img,big,CV_INTER_CUBIC);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,big);
cvWaitKey(0);
cvReleaseImage(&img);
cvReleaseImage(&big);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvResize(source, destination, type):
  - Destination needs to be an image of the same type but different size
  - Type – see manual ☺ but CV_INTER_CUBIC gives best looking results
- cvCreateImage(size, bits per channel, channels):
  - Size needs to be CvSize, so we use cvSize(x,y)
  - Bits per channel – normally 8, sometimes we need 16 or 32
  - Channels – in case of grayscale images it's 1, for all others it should be 3

Adding images

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
IplImage *im2 = cvCloneImage(img);
cvFlip(im2,im2,1);
cvAdd(img,im2,im2);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,im2);
cvWaitKey(0);
cvReleaseImage(&img);
cvReleaseImage(&im2);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvCloneImage – clones an image ☺
- cvAdd(source1, source2, destination)
- But… the problem is, that the image is too bright!
- So let's try to make an average of two images…

Making an average of two images

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
IplImage *im2 = cvCloneImage(img);
cvFlip(im2,im2,1);
cvAddWeighted(img,0.5,im2,0.5,0,im2);
cvNamedWindow(„Image”,1);
cvShowImage(„Image”,im2);
cvWaitKey(0);
cvReleaseImage(&img);
cvReleaseImage(&im2);
cvDestroyWindow(„Image”);
…
```

Explanation…

- cvAddWeighted(s1, w1, s2, w2, scalar, destination):
  - S1 and S2 – sources
  - W1 and W2 – weights given to sources
  - Scalar – an optional value added to the sum
- But how about we want to add two parts of an image…

Adding two images with masking

```
…
IplImage *img = cvLoadImage(„lena.bmp”,0);
IplImage *msk = cvLoadImage(„half.bmp”,0);
IplImage *dst = cvCreateImage(cvSize(256,256),8,1);
cvZero(dst);
cvAdd(img,dst,dst,msk);
cvFlip(msk,msk,1);
cvFlip(img,img,1);
cvAdd(img,dst,dst,msk);
…
```

Explanation…

- You can use mask images to make selective adding/subtracting/multiplying
- Masks are standard 8bit/1ch images, which should be binary (say, 0 and 255 or 0 and 1)

Splitting an RGB image

```
IplImage* img = cvLoadImage("d:/lakeview.bmp",1);
IplImage* ch1 = cvCreateImage(cvSize(256,256),8,1);
IplImage* ch2 = cvCreateImage(cvSize(256,256),8,1);
IplImage* ch3 = cvCreateImage(cvSize(256,256),8,1);
cvSplit(img,ch1,ch2,ch3,0);
cvNamedWindow("Img",1);
cvNamedWindow("Ch1",1);
…
cvShowImage("Img",img);
cvShowImage("Ch1",ch1);
…
cvWaitKey(0);
cvReleaseImage(&img);
cvReleaseImage(&ch1);
…
cvDestroyWindow("Img");
cvDestroyWindow("Ch1");
```

Explanation…

- cvSplit(source, ch1, ch2, ch3, ch4):
  - As there can be 4 channels, you need to write „0” if you only have 3 channels in the source
- If you have a problem with overlapping windows, refer to the manual for the command *cvMoveWindow*

Splitting a YCC image

```
…
cvNamedWindow("Img",1);
cvShowImage("Img",img);
cvCvtColor(img,img,CV_RGB2YCrCb);
cvSplit(img,ch1,ch2,ch3,0);
cvNamedWindow("Ch1",1);
…
cvMoveWindow("Img",10,10);
cvMoveWindow("Ch1",10,310);
cvMoveWindow("Ch2",280,310);
cvMoveWindow("Ch3",550,310);
cvShowImage("Ch1",ch1);
…
```

Explanation…

- cvCvtColor converts an image from one colour space to another:
  - RGB <-> Gray, YCrCb, Lab, HSV, HLS, XYZ, Luv…
- cvCvtColor(source, destination, type), where:
  - Source and destination may be the same (unless it's the RGB2Gray conversion)
  - Type is for example:
    - CV_RGB2YCrCb or CV_RGB2HSV or CV_Lab2RGB

Skin color segmentation

- Algorithm:
  - R > 95 and G > 40 and B > 20 and
  - max{R,G,B}–min{R,G,B} > 15 and
  - |R–G| > 15 and R > G and R > B

1. Get pixel value: getPixel
2. Find min and max
3. Set new pivel velue: setPixel

getPixel

```
…
pixR = getPixel(img,i,j,0);
…
uchar getPixel( IplImage*img, int lin,
  int col, int channel ) {
  return ((uchar*)(img->imageData +
  img->widthStep*lin))[col+channel];
}
```

setPixel

```
…
setPixel(img,i,j,val);
…
void setPixel(IplImage*img, int lin, int
  col, uchar val ) {
((uchar*)(img->imageData + img-
  >widthStep*lin))[col] = val;
}
```

Get/Set Pixel

- lin, col – number of row / column, later referred as i, j
- channel – number of color channel, for RGB i OpenCV:
  - 0 = B, 1 = G, 2 = R.
- val – integer velue for color in grayscale (0 for black, 255 for white)

Min/Max

```
…
int tab [3] = {pixR, pixG, pixB};
int min, max;
min = max = tab[0];
    for(int i = 0; i < 2; i++){
        if(tab[i] > max){max = tab[i];}
        if(tab[i] < min){ min = tab[i]; }
    }
…
```

Haar-like face detection

- The object detector proposed by Paul Viola and improved by Rainer Lienhart
- Classifier trained with a few hundreds of sample views of a particular object (i.e., a face or a car) and negative examples - arbitrary images
- See „boosted Haar classifier structures" and ***opencv/apps/haartraining*** for details

Haar-like face detection

```
…
static CvHaarClassifierCascade* cascade = 0;
const char* cascade_name =
        "haarcascade_frontalface_alt.xml";
…
IplImage *img = cvLoadImage("af.jpg",1);
static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;
CvPoint pt1, pt2;
…
```

Haar-like face detection

```
…
cascade =
   (CvHaarClassifierCascade*)cvLoad(cascade_name,0,0,0);
    if (!cascade) return;
        storage = cvCreateMemStorage(0);
        cvClearMemStorage(storage);
        //…cream of the cream… - on the next slide ☺
cvReleaseHaarClassifierCascade(&cascade);
cvReleaseMemStorage(&storage);
…
```

Haar-like face detection

```
…
if (cascade)
  {
  CvSeq* faces = cvHaarDetectObjects(img,cascade,storage,1.2,2, \
                        CV_HAAR_DO_CANNY_PRUNING,cvSize(25,25));
        for (int i=0;i<(faces ? faces->total:0);i++)
        {
                CvRect* r =(CvRect*)cvGetSeqElem(faces,i);
                pt1.x = r->x; pt2.x = r->x+r->width;
                 pt1.y = r->y; pt1.y = r->y + r->height;
                cvRectangle(img,pt1,pt2,CV_RGB(0,0,255),3,8,0);
        }
  }
…
```

Explanation…

- **CvHaarClassifierCascade** - structure used for representing a cascaded of boosted Haar classifiers
- **cvLoadHaarClassifierCascade** - loads a trained cascade classifier from file or the classifier database embedded in OpenCV. Ready *.xml files for face detection are here: **OpenCV\data\haarcascades**
- Don't forget to **cvReleaseHaarClassifierCascade** ☺

Explanation…

- CvSeq* **cvHaarDetectObjects**

  (const CvArr* image,

  CvHaarClassifierCascade* cascade,

  CvMemStorage* storage,

  double scale_factor=1.1,

  int min_neighbors=3,

  int flags=0,

  CvSize min_size=cvSize(0,0));

Explanation…

- **image** – image to detect objects in
- **cascade** – Haar classifier cascade
- **storage** – structure to store the resultant sequence of the object candidate rectangles
- **scale_factor** – the factor by which the search window is scaled between the subsequent scans, e.g. 1.1 means increasing window by 10%
- **min_neighbors** – min. number (minus 1) of neighbor rectangles that makes up an object
- **flags** – mode of operation. Currently only CV_HAAR_DO_CANNY_PRUNING may be specified
- **min_size** - minimum window size

Playing avi file

- Not all avi formats are supported by OpenCV
- CvCapture – video capturing structure
  - cvCaptureFromFile – inititalizes capturing video from file
  - cvCaptureFromAVI(filename)
  - cvCaptureFromCAM(index) – index of the camera to be used. If there is only one – put 0 or 1.
- cvQueryFrame – grabs and returns a frame from a camera or file
- Remember about **cvReleaseCapture** at the end!

Let’s play avi file!

```
…
CvCapture *cap = cvCaptureFromAVI(„clock.avi”);
if (cap) {
        cvNamedWindow("Video", 1);
        img = cvQueryFrame(cap);
        while (img && cvWaitKey(1)==-1) {
           if (img->origin) {
               cvFlip(img);}
           cvShowImage("Video", img);
           img = cvQueryFrame(cap);}
        cvDestroyWindow("Video");}
cvReleaseCapture(&cap);
…
```

Let's use a Webcam!

```
…
CvCapture *cap = cvCaptureFromCAM(0);
if (cap) {
        cvNamedWindow("Video", 1);
        img = cvQueryFrame(cap);
        while (img && cvWaitKey(1)==-1) {
           if (img->origin) {
               cvFlip(img);}
           cvShowImage("Video", img);
           img = cvQueryFrame(cap);}
        cvDestroyWindow("Video");}
cvReleaseCapture(&cap);
```

Another way of using webcam

- Use <cvcam.h>
- Needs additional function for frame grabbing and image processing – **callback**
- Allows for accessing camera properties:
  - cvcamSetProperty(int camera, const char* property, void* value);
  - cvcamGetProperty (int camera, const char* property, void* value);

cvcamSetProperty

- Sets the value of the specified **property** of the specified **camera** to **value**.
- **camera** – a number of the camera in 0-based index of cameras found in the system.
- **property** – a name of the property
- **value** – depends on the property's name.
- See cvcam Properties Interface for details

cvcamGetProperty

- If successful, **value** will contain the value of the specified **property** for the specified **camera**
- cvcamGetProperty(0,CVCAM_VIDEOFORMAT,NULL); displays camera settings of format, size, etc.
- cvcamGetProperty(0,CVCAM_CAMERAPROPS,NULL); displays control panel for contrast, brightness, etc.

You can do it in a different way…

```
#include <cvcam.h>
…
void callback(IplImage *image)
{
IplImage *res = cvCreateImage(cvSize(image->width,image-
   >height),IPL_DEPTH_8U,image->nChannels);
if (image->origin == IPL_ORIGIN_TL)
   cvCopy(image,res,0);
else
   cvFlip(image,res,0);
cvReleaseImage(&res);
}
```

Start/Stop

```
…
int ncams = cvcamGetCamerasCount();
cvcamSetProperty(0,CVCAM_PROP_CALLBACK,callback);
cvcamInit();
cvcamStart();
…
cvcamStop();
cvcamExit();
```

Access to camera properties

```
…
int ncams = cvcamGetCamerasCount();
cvcamSetProperty(0,CVCAM_PROP_ENABLE,CVCAMTRUE);
cvcamSetProperty(0,CVCAM_PROP_RENDER,CVCAMTRUE);
cvcamSetProperty(0,CVCAM_PROP_CALLBACK,callback);
cvcamInit();
cvcamStart();
…
cvcamGetProperty(0,CVCAM_VIDEOFORMAT,NULL);
cvcamGetProperty(0,CVCAM_CAMERAPROPS,NULL);
```

The end

- Thank You for today ☺