Computer Vision Laboratory

COMPUTER VISION AND IMAGE PROCESSING

LAB SESSION 1
INTRODUCTION TO OPENCV

DR. FEDERICO TOMBARI, DR. SAMUELE SALTI

The OpenCV library

- Open Computer Vision Library: a collection of open source algorithms for computer vision and image processing
- Originally developed by Intel, then funded and supported by Willow Garage. Currently a non-profit foundation (www.opencv.org)
- Code is released under the BSD licence – free for both academical and commercial use
- Main language: C/C++, with optimized routines (multi-thread, SIMD, ..)
- Current version: **2.4**
- Freely downloadable from:
  - http://sourceforge.net/projects/opencvlibrary
- Available for Windows, Linux, iOS. Currently supporting also Android
- During our lab sessions:
  - O.S.: Windows-based
  - C++ Compiler: Visual Studio 2010

In the meanwhile..

- Download the lab material from:
  - didattica.arces.unibo.it
  - Prof. Luigi Di Stefano
  - Computer Vision and Image Processing course
  - Material (from lefthand menu)
  - «Laboratory: Slides, Software and Images»
  - Software ElabImage, OpenCV and Documentation (zip file)
  - Unzip the archive and open up the VisualStudio (.sln) solution included in the sub-folder: “Elabimage”

OpenCV structure (before 2.2)

- **4 main libraries:**
  - **CV:** containing most of the IP/CV algorithms
    - Image processing
    - Motion analysis
    - Pattern recognition
    - 3D reconstruction
    - …
  - **CxCore:** support functionalities
    - Main data structures
    - Data structure access (initialization, value insert/modify, copy, delete, ..)
    - Matrix operations (arithmetical, logical, matrix inversion, permutations, ..)
    - Drawing (points, lines, ellipses, ..)
    - …
  - **HighGui:** simple I/O operations:
    - Window creation/destruction for showing images on screen
    - Image load/save
    - Video stream handler (from files and webcams)
    - …
  - **CvCam**

OpenCV structure (after 2.2)

- **Several modules, allows linking only to required features**
  - **core:** defines basic data types such as points, vectors, single/multi channel matrices (images), includes also functions for linear algebra, DFT, XML, YAML-based I/O
  - **imgproc:** algorithms for image filtering, morphology, resizing, color mapping, image histograms, etc..
  - **highgui:** window handler for displaying images, video stream handler,…
  - **calib3d:** camera calibration, stereo matching,…
  - **features2d** 2D feature detectors and descriptors (SIFT, SURF, FAST, etc., including the new feature detectors-descriptor-matcher framework)
  - **flann:** wrapper of the Fast Library for Approximate Nearest Neighbors (FLANN) for Nearest Neighbor Search over high dimensional spaces
  - **ml:** machine learning algorithms (SVM, Decision Trees, Boosting, Random Forests, etc.)
  - **objdetect:** object detection on images (Haar & LBP face detectors, HOG people detector etc.)
  - **video:** algorithms for computer vision on video streams (tracking, optical flow, background subtraction,…)
  - **gpu:** acceleration of some OpenCV functionalities using CUDA (stereo, HOG, linear algebra)
  - **contrib:** contributed code that is not mature enough (SpinImages, Chamfer distance, …)
  - **legacy:** obsolete code, preserved for backward compatibility

(Slightly) advanced operations..

- OpenCV traditionally offers limited capabilities for what concerns user interaction and GUIs
- Before version 2.2:
  - Trackbar creation for easier manipulation of parameters
  - Mouse click capture on images
  - Text print on images
- After version 2.2 (included): Qt backend for OpenCV
  - text rendering using TTF fonts,
  - separate "control panel" with sliders, push-buttons, checkboxes and radio buttons
  - interactive zooming, panning of the images displayed in highgui windows, “save as“, etc…
- As for the final project, and only for those interested (not mandatory!): Microsoft Foundation Classes (MFC), Java, Qt, …

How to get Help!

- Old versions (pre 2.2): in the OpenCV folder, after installing it (as well as included in the zip file of the course material) you'll find:
  - /doc/opencv.pdf

  (Mostly) detailed description of each algorithm of the library

- For more recent versions:
  - Online OpenCV documentation: ***docs.opencv.org***
  - OpenCV Cheatsheet (compact!)

- Forum on Yahoo Groups:
  - http://tech.groups.yahoo.com/group/OpenCV/

  Ask online, then wait for an answer..

OpenCV (pre 2.2) and VS projects

- **Once and for all: set DLL path to the Opencv folder:**
  - Append the OpenCV «bin» subfolder to the Windows «PATH» environment variable. In our solution, the subfolder is the «cvdll» within «Elabimage»
- **Each time we create a new project:**
  - Create a new project via «File->New->New Project», specifying «**Visual C++ Empty Project**»
  - Specify the folder containing the include (.h) files: in “Project -> Properties”, choose “Configuration Properties -> C/C++ -> General” and add in the field “Additional include directories” the (relative) path to the .h files; e.g. add a string such as:
    - “../include”
  - Specify the required .lib files: in “Project -> Properties”, choose “Configuration Properties -> Linker -> Input” and add in the field “Additional dependencies” a string such as:
    - “../cvlib/cv.lib ../cvlib/cvaux.lib ../cvlib/cxcore.lib ../cvlib/highgui.lib”
  - Add the appropriate “#include” commands for the OpenCV headers at the beginning of your code. E.g.:
    - #include “highgui.h “ #include “cv.h“ #include “cxcore.h“ #include “cvaux.h“

OpenCV (post 2.2) and VS projects

- For each new project:
  - Add the OpenCV «include» folder as an «additional include directory»: in “Project –> Properties”, select the “Configuration Properties –> C/C++ –> General” tab and add in the field “Additional Include Directories” your own include path:
    - *“ROOT_OPENCV\include”*
  - Add the OpenCV «libs» folder as an «additional library directory»: in “Project –> Properties”, select the “Configuration Properties –> C/C++ –> General” tab and add in the “Additional Include Directories” your own lib path:
    - *“ROOT_OPENCV\lib”*
  - Specify the required OpenCV libs for the current project. In “Project –> Properties”, select the “Configuration Properties –> Linker –> Input” tab and add in the field “Additional dependencies” the required lib files, eg.
    - *“opencv_core220.lib opencv_imgproc220.lib opencv_highgui220.lib”*
  - Add the “#include” command and the required .h files appropriately at the beginning of the header files of your project, e.g.
    - *#include “opencv2/opencv.hpp*
- Append the OpenCV «bin» subfolder to your Windows «PATH» environment variable
  - this should be already done by the autoinstaller - just say yes when prompted; otherwise you can do it manually.

IplImage

- □ Basic OpenCV data structure (pre version 2.2; successively substituted by `cv::Mat`) representing an image
- □ Derived from the IPL (Intel Image Processing Library – not used anymore)
- □ Defined in CxCore (pre 2.2) / opencv_core (2.2)
- □ All useful info specified on the CxCore/core doc

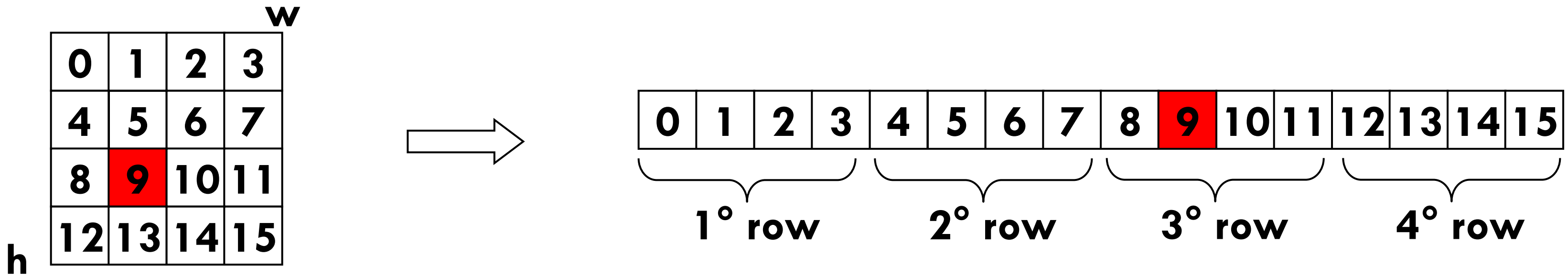
Accessing the image elements

- Images: bidimensional (2D), but stored in memory as monodimensional vectors:

```
(imagename -> imageData)[index]
```

- A cast is required, since within an `IplImage` the `imageData` field is defined as `char*`:

```
((datatype *) (imagename -> imageData)) [index]
```



0	1	2	3
4	5	6	7
8	🟥 9	10	11
12	13	14	15

0	1	2	3	4	5	6	7	8	🟥 9	10	11	12	13	14	15

Logical representation **Physical representation**

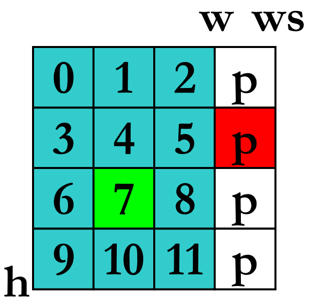
- Data access for reading/writing is done by means of only one index:

```
(row-1) * w + (column-1)
```

- E.g. : the element in 3rd row, 2nd column (n° 9) : (3-1) * w + (2-1) = 2 * 4 + 1 = 9

Accessing the image elements (2)

- In the IplImage case: the widthstep field does not always coincide with the width one. widthstep has always to be used when indexing the data array of an IplImage.



0	1	2	p
3	4	5	p
6	7	8	p
9	10	11	p

Eg. a 3x4 image = 12 elements; h= 4, w = 3, ws = 4.
A «padding» column with meaningless content is automatically added for a more efficient memory alignment of the image rows.

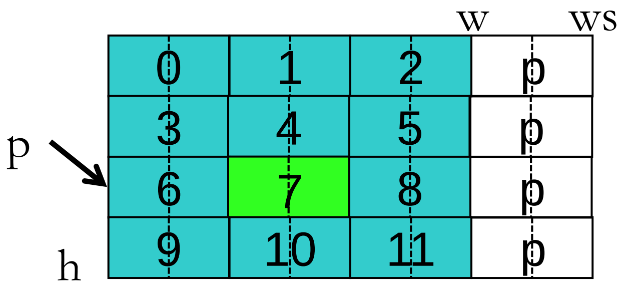
- To access the 7th image element (coordinates (r,c) = (3,2)):
  - by means of width: 2*w + 1 the red element is accessed (meaningless)
  - by means of widthstep: 2*ws + 1 the green element is accessed (correct)

Accessing the image elements (3)

- The previous examples are valid only if the image contains bytes (8 bit *depth*, either signed or unsigned)
  - This is referred in OpenCV as `IPL_DEPTH_8U`, `IPL_DEPTH_8S`
- A more general way to access image elements which is valid for all kind of image types is as follows

```
((datatype*)(imagename->imageData+(row-1)*ws))[col-1]
```

- Same example as before, with short data (2 bytes per element, 16 bit depth)



0	1	2	p
3	4	5	p
6	7	8	p
9	10	11	p

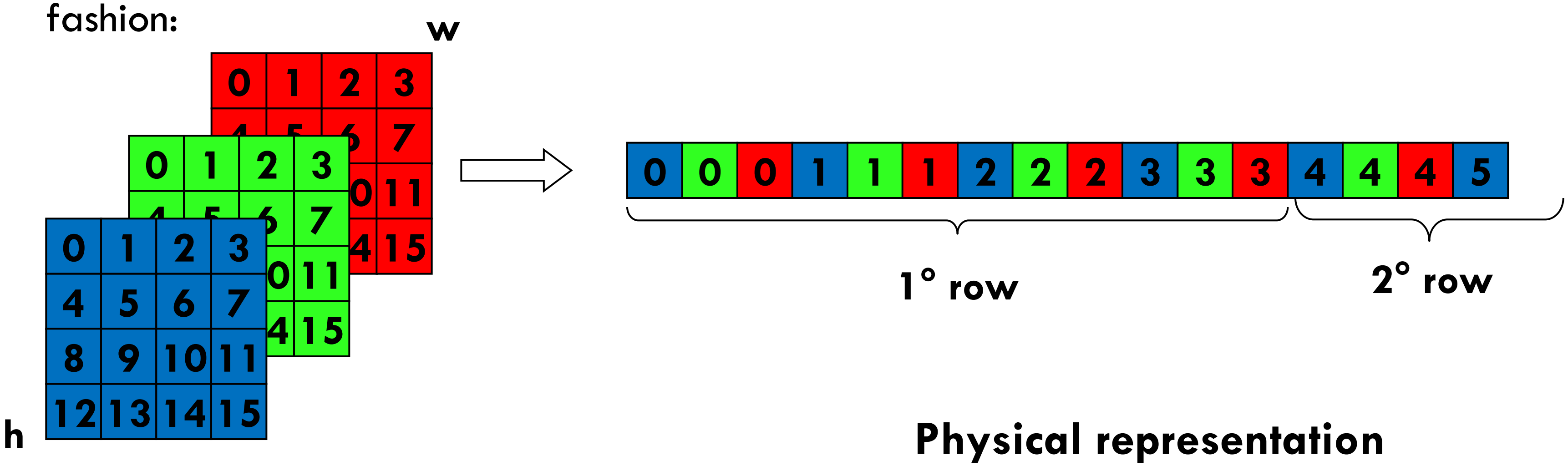
Eg. 3x4 image = 12 elements; h= 4, w = 3, ws = 8 (in byte!).

```
short* p = ((short*) (imagename -> imageData + 2*ws))
```

`p[1]` is the required image element

Accessing the image elements (4)

- Color images have 3 channels: Blue, Green, Red (**BGR**), stored in an **interleaved** fashion:



Logical representation

Physical representation

- To access a specific channel (color) of a pixel, the row index has to be multiplied by 3 and summed to the appropriate offset:
  - B: `((datatype *) (imagename -> imageData + (row-1)*ws)) [3*(col-1)]`
  - G: `((datatype *) (imagename -> imageData + (row-1)*ws)) [3*(col-1)+1]`
  - R: `((datatype *) (imagename -> imageData + (row-1)*ws)) [3*(col-1)+2]`

CvMat

- Data structure that represents matrices and vectors.
- Similar to `IplImage` (merged into `cv::Mat` since OpenCV 2.2)
  - `rows, cols` instead of `height, width`
  - `step` instead of `widthStep`
  - `data.ptr` instead of `imageData`

    ```
    ((datatype*)(mat->data.ptr+(row-1)*mat->step))[col-1]
    ```

    - `data` is a `union` of pointers, you can directly use the pointer of the matrixtype, if the type implies **4 bytes alignment.** E.g., for a matrix of double, you can use

      ```
      mat->data.db[(row-1)* mat->cols+(col-1)]
      ```

    - Alternatively, you can use the macro `CV_MAT_ELEM(*mat, data_type, row-1,col-1)`
- To create a column vector of 3 `doubles`

  ```
  CvMat* point3D = cvCreateMat(3,1,CV_64F);
  ```

- You have to release it with `cvReleaseMat(&point3D);`
- OpenCV offers function for standard linear algebra (`cvGEMM`, `cvSolve`)

`cv::Mat`

- Basic data structure introduced by OpenCV 2.2 for representing an image
- It’s a C++ class included in the `cv` namespace
- It contains specific constructors and cast operators for converting to/from an `IplImage`.
- All useful info are included in the respective C++ cheatsheet
  - /doc/opencv_cheatsheet.pdf
- Memory organization is analogous to that of the IplImage
- `widthStep` replaced by `step`, `width` by `cols`, `height` by `rows`
- Image element access:

```
((datatype *) (imagename -> data + (col-1)*step)) [(row-1)]
```

alternatively:

```
imagename->at<datatype>(row-1, col-1);
```

- **Memory is automatically released by the destructor**

`cv::Mat` examples

- Linear algebra operations are carried out by re-definition of the operators; in such way, it is possible to use MATLAB-like syntax:

```
 cv::Mat A(3,3,CV_8UC1);
  cv::Mat B(3,3,CV_8UC1);
  cv::Mat C = A – B; // element-wise subtraction
```

```
  C = 255 – A; // «old» deallocates the memory space
previously occupied by C, allocates a new matrix and sets
each element of C to 255 minus the corresponding element
of A
```

cv::Mat examples (2)

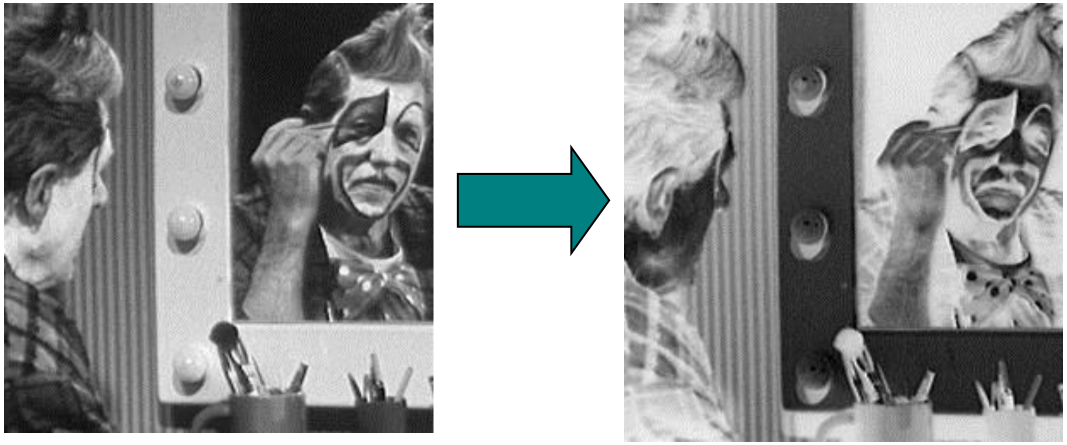
```
  C = cv::abs(2*A - cv::min(B)); // deallocates the memory
space previously occupied by C, allocates a new matrix and
sets each element of C to the absolute value of 2 times
the corresponding element of A minus the minimum element
in B; all data saturations are handled automatically
```

```
   Mat x = (A.t()*A).inv()*(A.t()*b); //Solves a linear
system with the pseudo-inverse matrix
```

Exercise 1 – “invert grey”

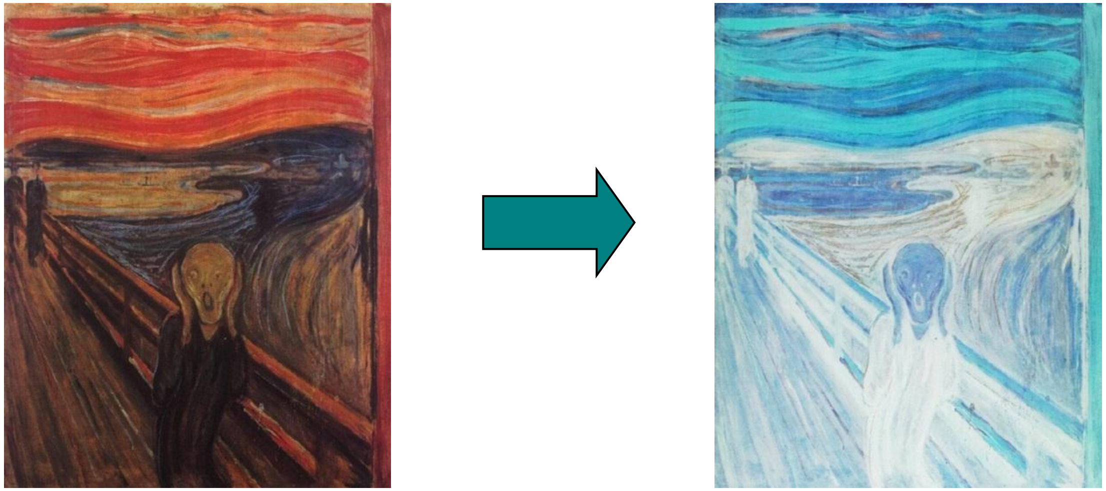
□ Compute the “negative” of a grayscale image

□ Given a grayscale image (range of each pixel between [0 255]), substitute each pixel having intensity I with the value: 255-I



Exercise 2 – “invert RGB”

- Same as before, but in this case we want to compute the negative of a color image.
- The image has 3 channels, representing the 3 RGB values
- The intensity of each channel ranges between [0 255]
- For each image pixel, we need to substitute the (B,G,R) triplet with its «inverse» (255-B, 255-G, 255-R)



Exercise 3 – Image difference

- Build a new VS project which performs the following:
  - loads 2 images (Image 1, I1 and Image 2, I2)
  - computes the pixel-wise difference between the two images:
  - computes an output image where each pixel of coordinates (x,y) contains the absolute difference of the corresponding pixels on I1 and I2:

$$Out(x,y) = abs(I1(x,y) - I2(x,y))$$

  - Displays on a window the output image