OpenCV

Seminario per Sistemi Multimediali A.A. 2018-2019

Ing. Giovanni Cozzolino (giovanni.cozzolino@unina.it)

What is OpenCV?

- OpenCV (**Open** Source **C**omputer **V**ision) is a library of programming functions containing all the standard algorithms for Computer Vision;
- Implemented for:
  - Windows
  - Linux
  - Mac systems
  - Mobile (iOS, Android)
- Implemented in:
  - C/C++
  - Python
  - Java
  - Matlab…
- Can be freely downloaded from: http://opencv.org
- Last Version is 3.2.0 (23/12/2016)

Brief history of OpenCV

- **1998** OpenCV started by Intel Research Labs (CPU-intensive applications)
- **2000** Presented at CVPR2000 (Computer Vision and Pattern Recognition)
- **2006** First release (v 1.0)
- **2008** First corporate support (Willow Garage – R.O.S.)
- **2009** Released OpenCV2 (C++ interfaces)
- **2012** Supported by non-profit foundation OpenCV.org
- **2014** Released OpenCV3 (IPP, GPU, mobile support)

Initial Goals

Computer Vision	Standard	Open
Advance vision research by providing not only open but also optimized code for basic vision infrastructure. <u>No more reinventing the wheel</u>.	Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.	Advance vision-based commercial applications by making portable, performance-optimized code available for free with a license that did not require to be open or free themselves.

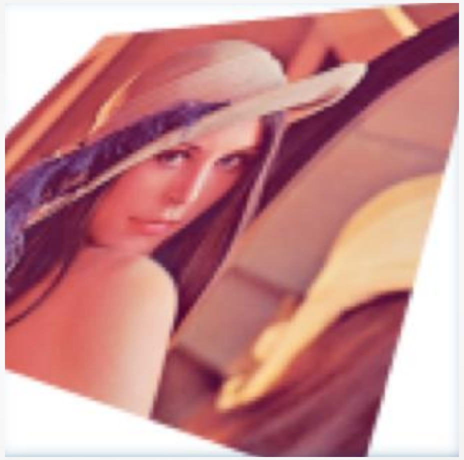
OpenCV main packages

- **core** Basic functionalities and data structures;
- **imgproc** Image processing functions (blurring, histograms, registration, tracking, detection);
- **highgui** High-level Graphical User Interface;
- **calib3d** Camera calibration and 3D Reconstruction;
- **features2d** Features detection and description;
- **objdetect** Object detection;
- **ml** Machine Learning and Pattern Recognition tools (e.g. k-means, SVM, knn)

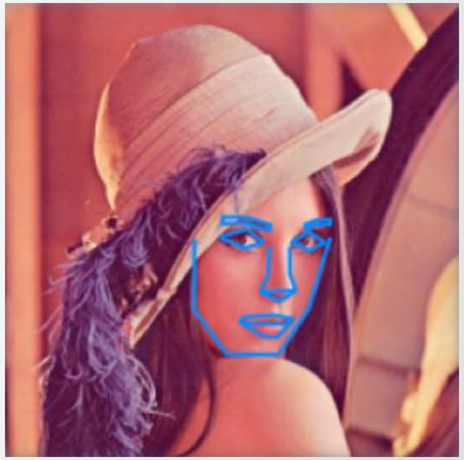
What OpenCV can do?

http://www.cs.cmu.edu/~chuck/lennapg/

Filters

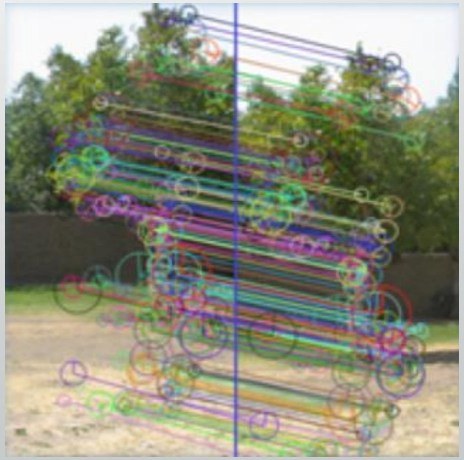


Transformations



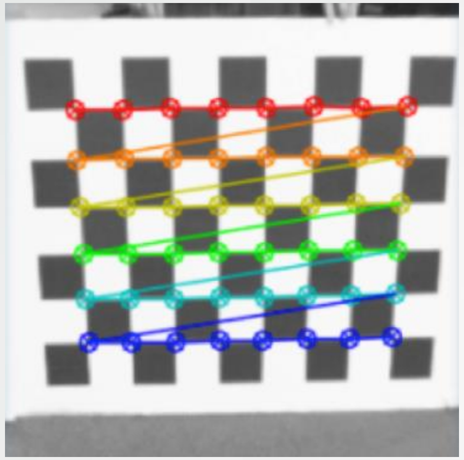
Edges & Contours

Segmentation

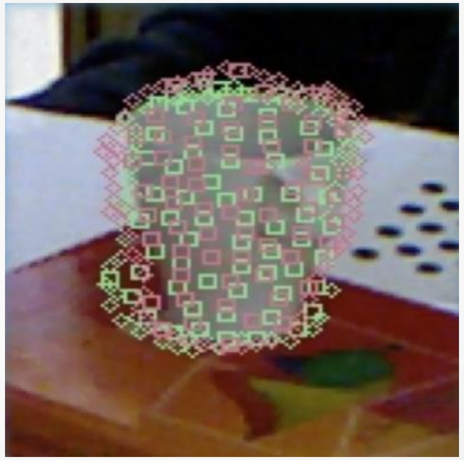


Features Extraction

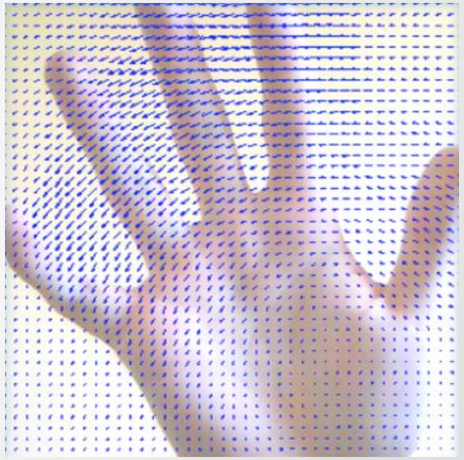
What OpenCV can do?



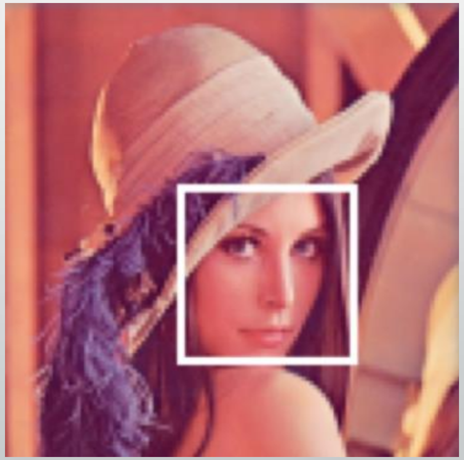
Camera Calibration



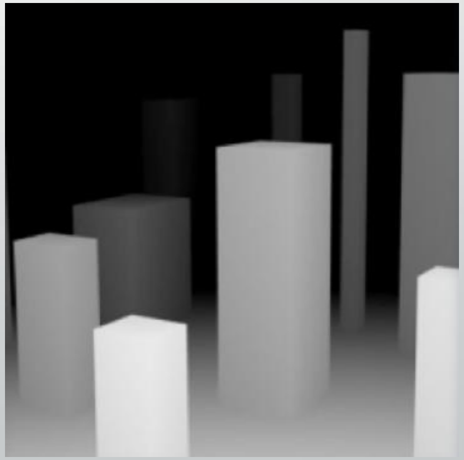
Pose Estimation



Optical Flow



Detection and Recognition

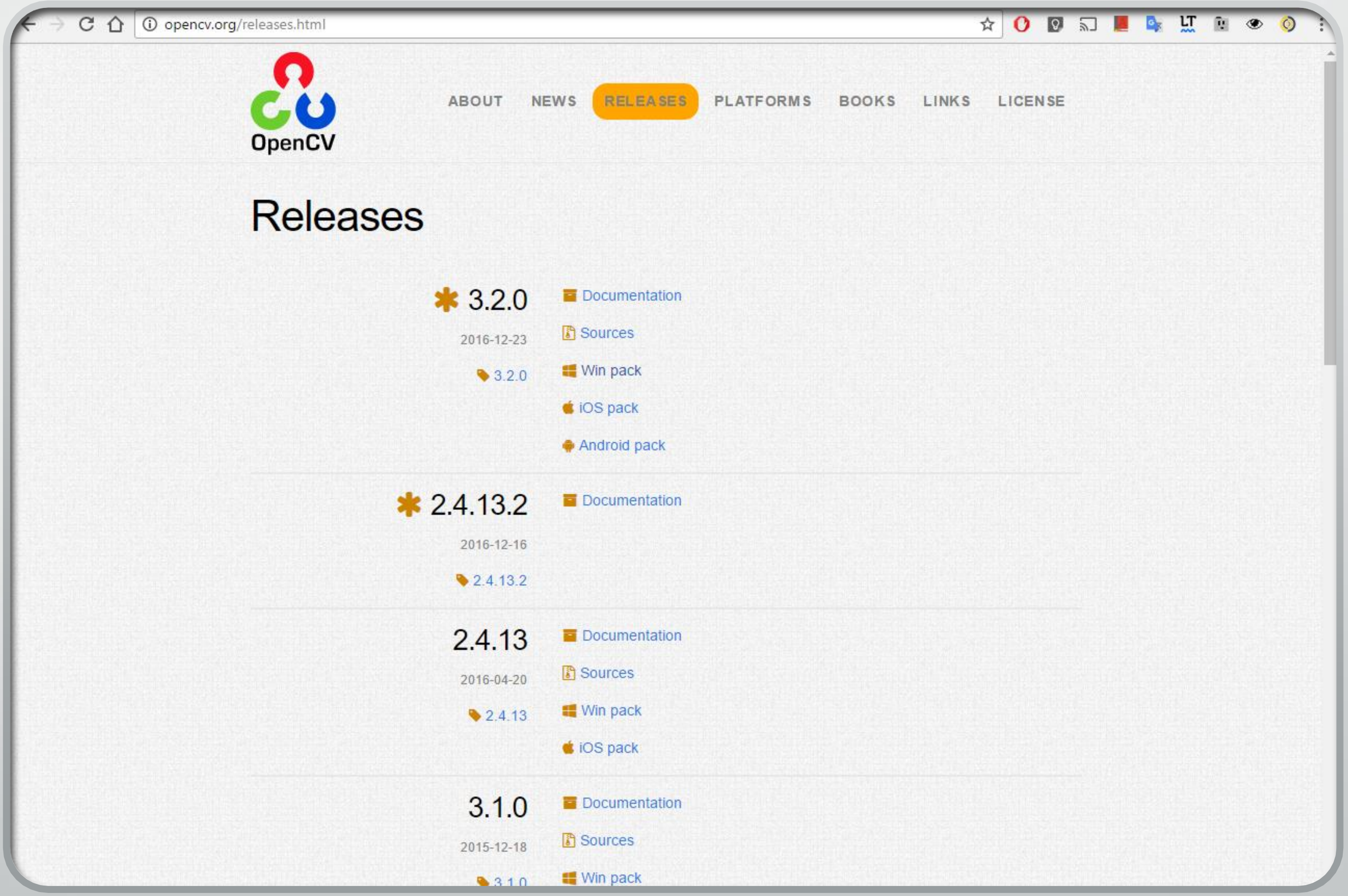


Depth Estimation

http://www.cs.cmu.edu/~chuck/lennapg/

Download OpenCV (windows)

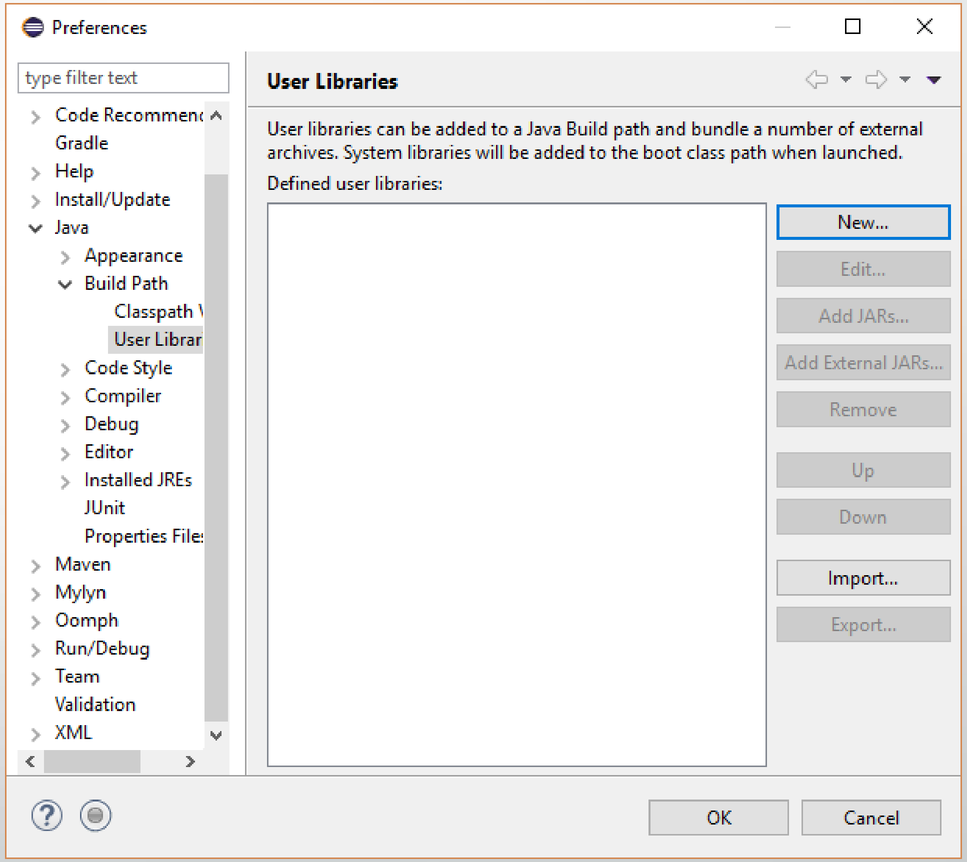
1. http://opencv.org
2. Releases
3. v. 3.2.0
4. Win pack
5. Execute the self-extracting file



Configure OpenCV (per Java & Eclipse)

- You only need the following files:
  - opencv\build\java\opencv-320.jar
  - opencv\build\java\x86\opencv_java320.dll [if you have a 32bit architecture]
  - opencv\build\ java\ x64\opencv_java320.dll [if you have a 64bit architecture]

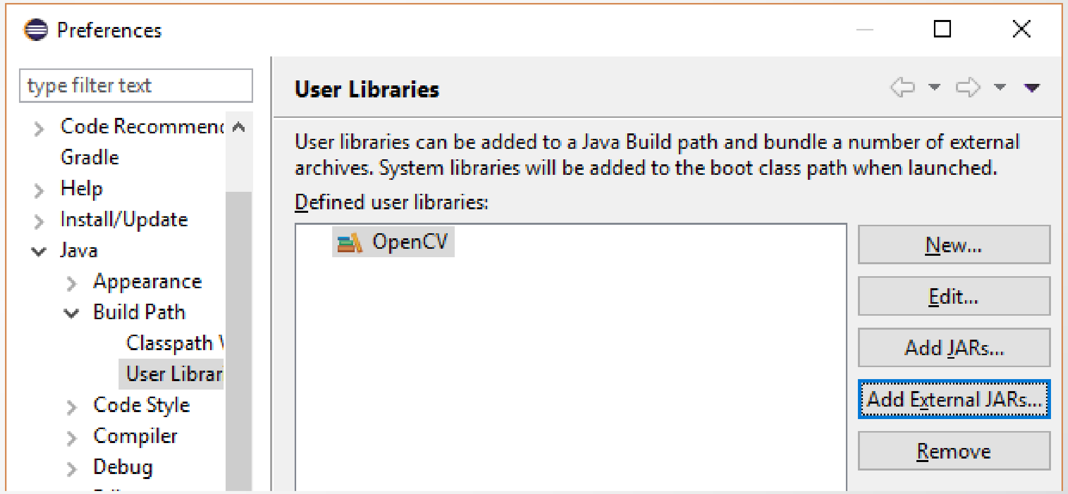
1. Create the User Library entry:
   1. Launch Eclipse
   2. Select Window -> Preferences from the menu

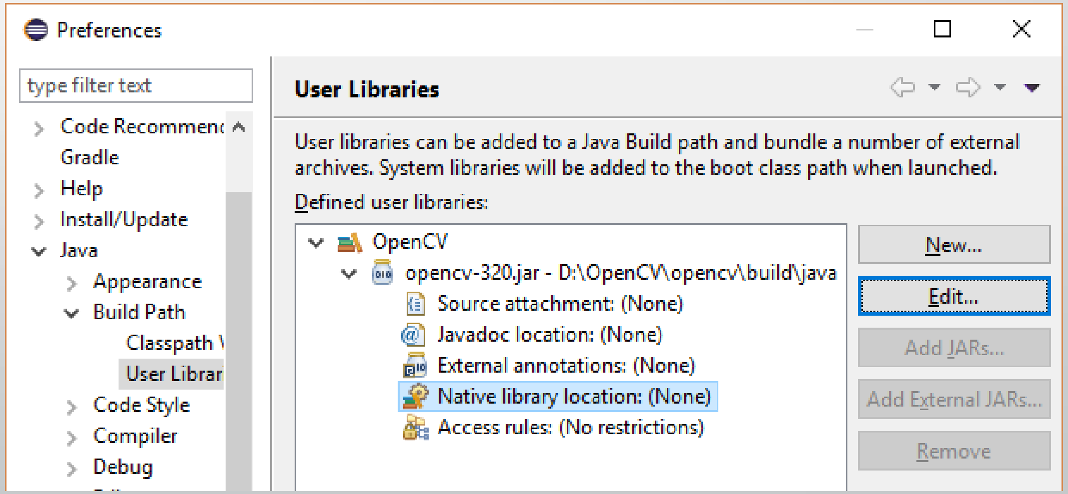


   3. Navigate under Java -> Build Path -> User Libraries and click New
   4. Enter the name for your library. For example, OpenCV.

Configure OpenCV (per Java & Eclipse)

2. Link the JARs:
    1. Select the new user library (OpenCV)
    2. Click on “Add External JARs…”
    3. Browse through opencv\build\java\ and select opencv-320.jar
3. Link the Library files:
    1. extend the opencv-246.jar
    2. select Native library location
    3. press “Edit…”
    4. Select “External Folder…”
    5. browse to select the folder opencv\build\java\x86 or opencv\build\java\x64

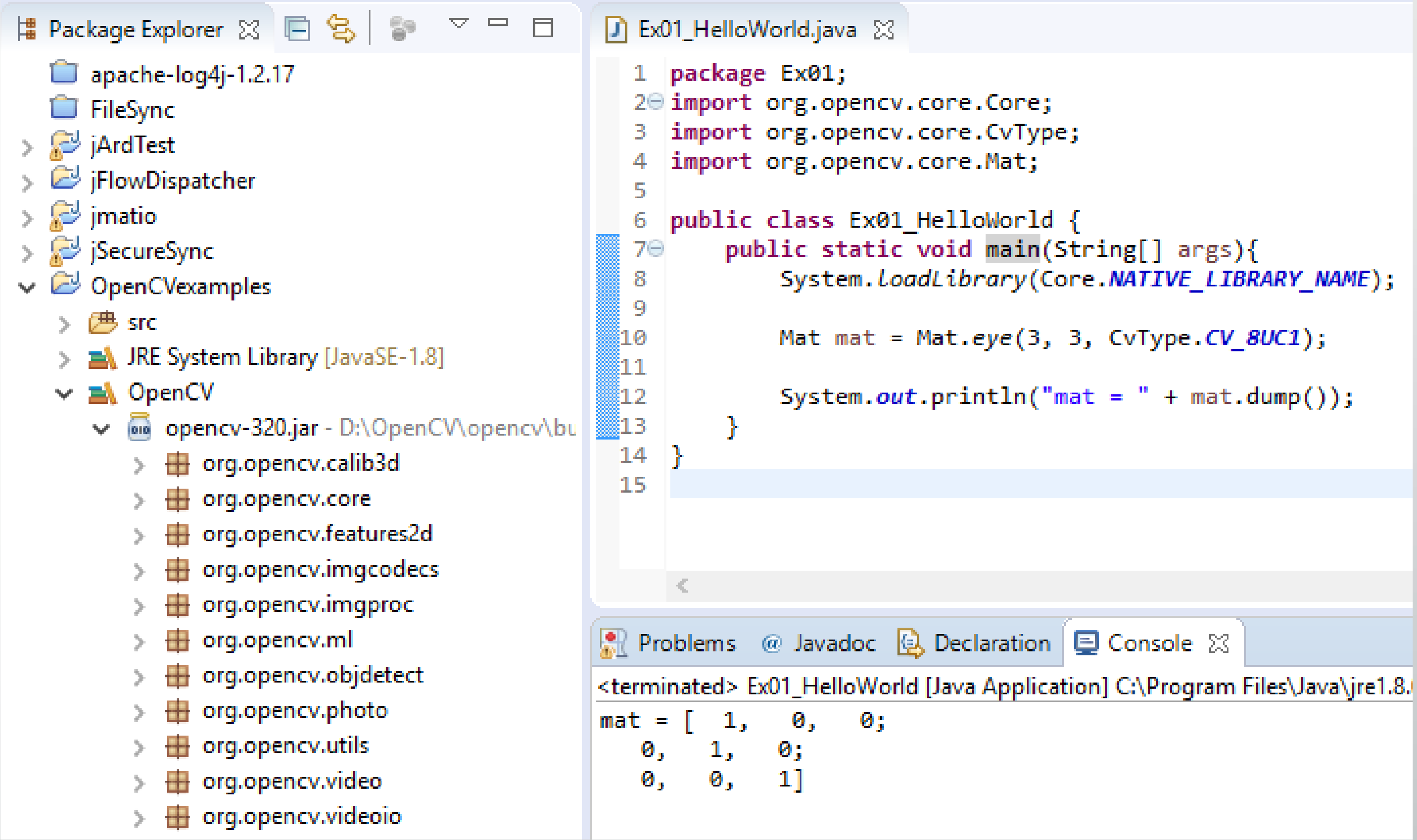




Hello World

Add the OpenCV to the a Java Project:

- Select Project -> Properties
- Java Build Path -> Libraries
- Press Button "Add Library…"
- Select "User Library"
- Select "OpenCV"



Class Mat

- The class **Mat** represents an n-dimensional dense numerical single-channel or multi-channel array.
- **Mat** can represent:
  - A matrix
  - A filter
  - An image
  - A set of vectors (e.g. descriptors)
  - …

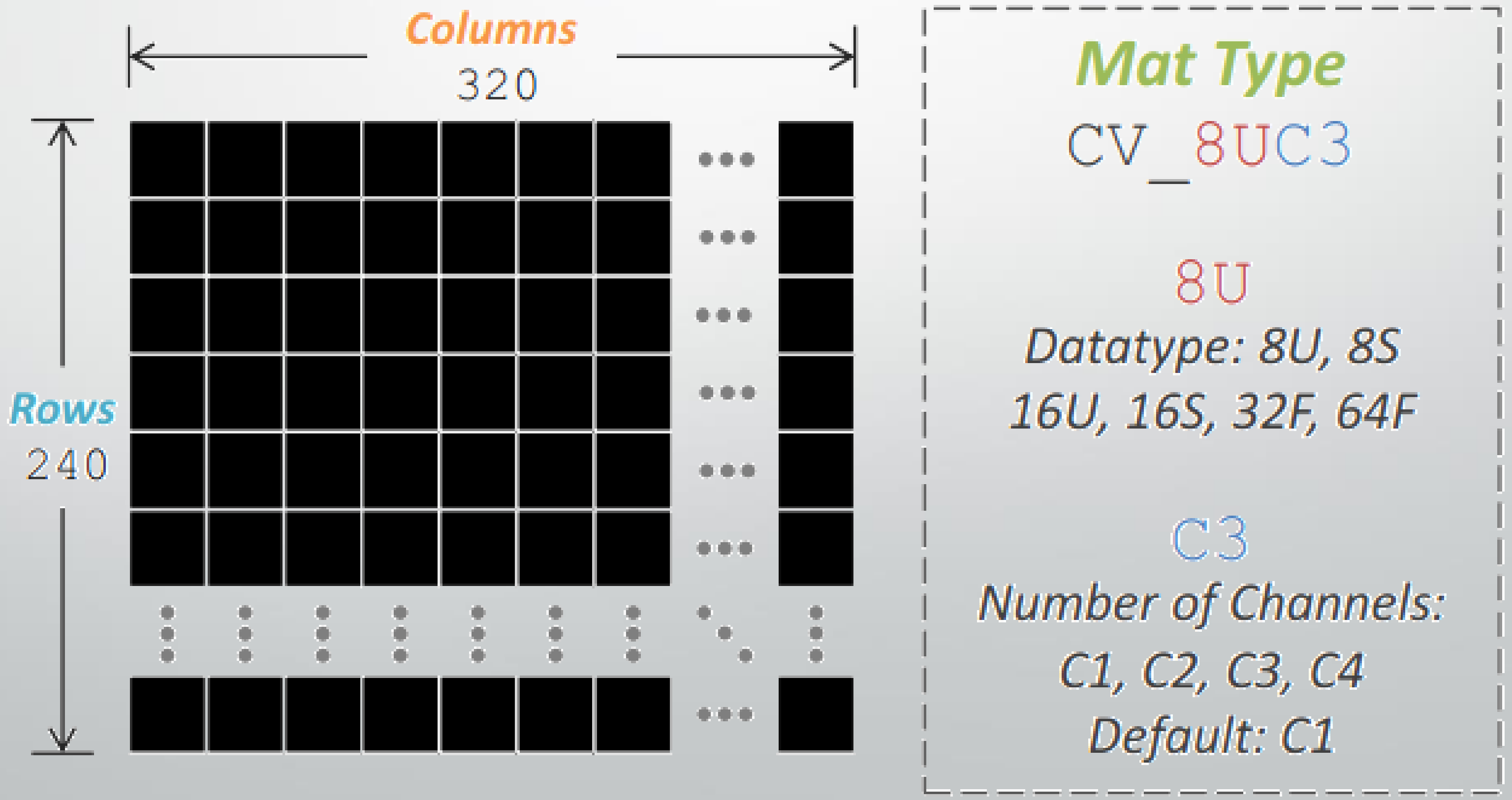
OpenCV matrix types

#	CvType	Description
0	CV_8U	8-bit unsigned integer (uchar)
1	CV_8S	8-bit signed integer (schar)
2	CV_16U	16-bit unsigned integer (ushort)
3	CV_16S	16-bit signed integer (short)
4	CV_32S	32-bit signed integer (int)
5	CV_32F	32-bit floating point number (float)
6	CV_64F	64-bit floating point number (double)
7	CV_USRTYPE1	User-defined type

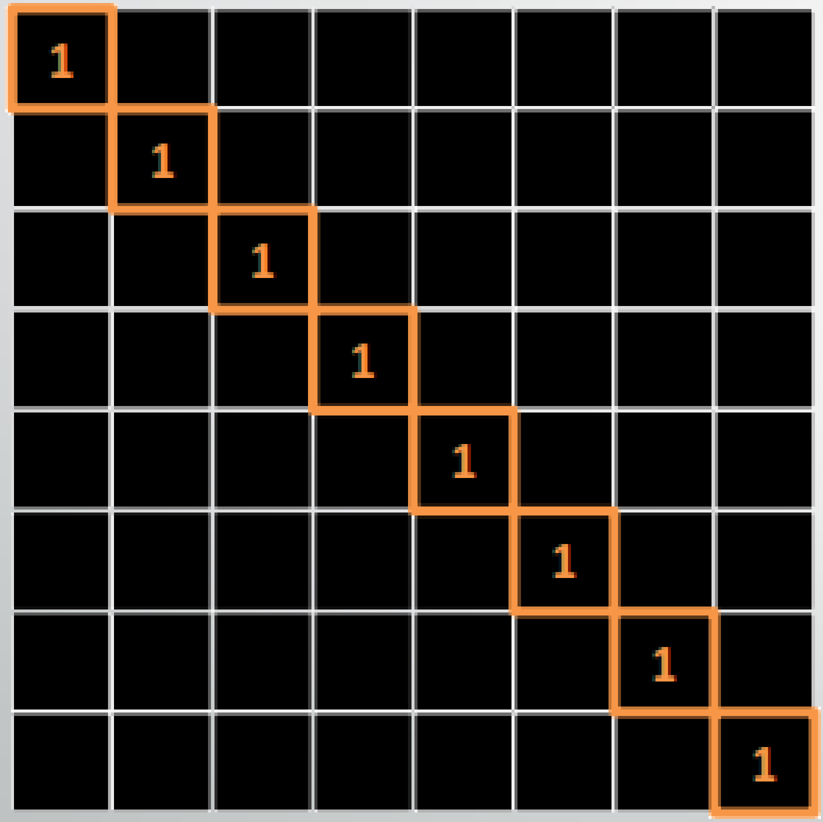
Images and matrices

Images and Matrixes are represented by the same type

Mat *image* = new Mat(240, 320, CvType.CV_8UC3);



Images and matrices



Initialise

- Matrix Initialised as Identity

  Mat *m* = Mat.eye(8, 8, CvType.CV_8UC1);

Images and matrices

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Initialise

- Matrix Initialised with zeroes

```
Mat m = Mat.zeros(8, 8, CvType.CV_8UC1);
```

Images and matrices

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Initialise

- Matrix Initialised with ones

```
Mat m = Mat.ones(8, 8, CvType.CV_8UC1);
```

Images and matrices

5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5

Initialise

- Matrix Initialised with a constant

```
Mat m = Mat.zeros(8, 8, CvType.CV_8UC1);
m.setTo(new Scalar(5));

Mat m = Mat.zeros(8, 8, CvType.CV_8UC1, new Scalar(5));

Mat m = Mat.ones (8, 8, CvType.CV_8UC1);
Core.multiply(m, new Scalar(5), m);

Mat m = Mat.zeros (8, 8, CvType.CV_8UC1);
Core.add(m, new Scalar(5), m);
```

Images and matrices

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Initialise

- Matrix Initialised with specific values

```
byte values[] = {0, 1, 2, 3, ..., 63};
Mat m = Mat.ones(8, 8, CvType.CV_8UC1);
m.put(0, 0, values);
```

Images and matrices

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Access to a pixel

- ***m*.get(3, 2);**
  Returns an array of Double (an element per each channel)
  <u>Note that OpenCV store pixels as BGR</u>
- ***m*.put(3, 2, 0);**

Images and matrices

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

- **Access to submatrixs**

  m.row(3);

  m.col(3);

  m.rowRange(new Range(2, 5));

  m.colRange(new Range(1, 6));

  m.submat(new Range(2, 5), new Range(1, 6));

Images and matrices

Scalar Operator

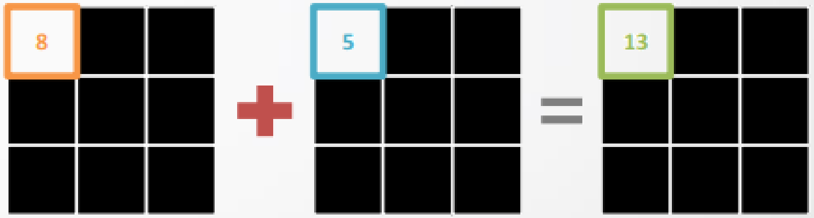
Core. add(*m1*, new Scalar(5), *res*);

Core. subtract(*m1*, new Scalar(5), *res*);

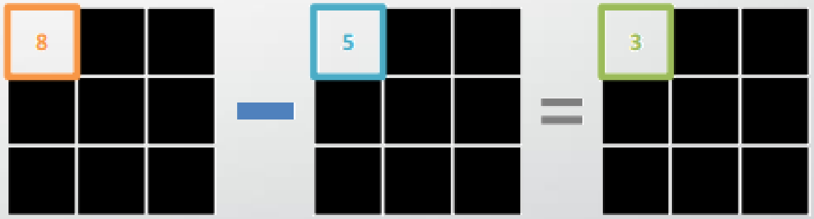
Core.multiply(*m1*, new Scalar(5), *res*);

Matrix Operator

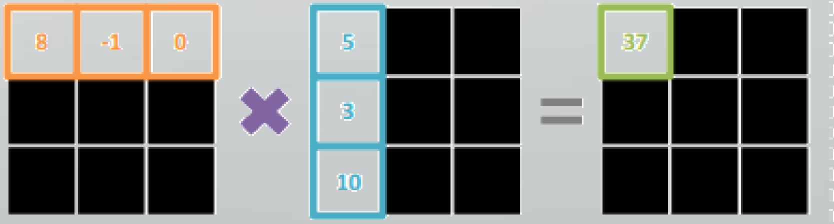
Core. add(*m1*, *m2*, *res*);



Core. subtract(*m1*, *m2*, *res*);



Core. multiply(*m1*, *m2*, *res*);



Images and matrices

Matrix Transpose

`mat.t()`

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^{T} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Matrix Inverse

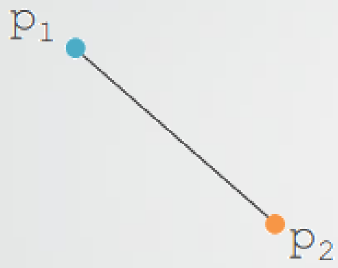
`mat.inv()`

$$\begin{pmatrix} 1 & -2 & 3 \\ 2 & -5 & 10 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 5 & 2 & 5 \\ 2 & -1 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

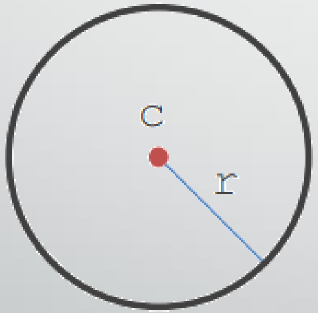
Geometric Primitives

- **Scalar** Template class for a 4-element vector
- **Range** Template class specifying a continuous subsequence (slice) of a sequence
- **Ptr** Template class for smart reference-counting pointers…
  not in Java! ;)
- **Point** Template class that represents a 2-column vector containing the coordinates of a point in a plane
- **Point3** Template class that represents a 3-column vector containing the coordinates of a point in the space
- **Rect** Template class that represents a rectangle, defined by the upper-left corner coordinates, width and height

Geometric Primitives




```
line(im,p1,p2,color,thickness);
   cv::Point(x,y)
```




```
circle(im,c,r,color,thickness);
             CV_RGB(r,g,b)
```

Graphic User Interface

- Read images:
  - C/C++: **imread(filename);**
  - Java: OpenCV 2.x (JavaDoc)

    Mat *image* = Highgui.imread("path/to/img");
  - Java: OpenCV 3.x (JavaDoc)

    Mat img = Imgcodecs.imread("path/to/img");
  - In both versions you can pass a second parameter specifying how to load the image:
    - CV_LOAD_IMAGE_ANYDEPTH: returns 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit
    - CV_LOAD_IMAGE_COLOR: always convert image to a colour one
    - CV_LOAD_IMAGE_GRAYSCALE: always convert image to a grayscale one

Graphic User Interface

- Show Images:
  - C/C++ **imageshow(windowname, image);**
  - Java

```
public static void imshow(String windowname, Mat m){
    int type = BufferedImage.TYPE_BYTE_GRAY;
    if ( m.channels() > 1 )
        type = BufferedImage.TYPE_3BYTE_BGR;
    int bufferSize = m.channels()*m.cols()*m.rows();
    byte [] b = new byte[bufferSize];
    m.get(0,0,b); // get all the pixels
    BufferedImage image = new BufferedImage(m.cols(),m.rows(), type);
    final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
    System.arraycopy(b, 0, targetPixels, 0, b.length);
    ImageIcon icon=new ImageIcon(image);
    JFrame frame=new JFrame(windowname);
    JLabel lbl=new JLabel(icon);
    frame.add(lbl);
    frame.pack();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
```

Graphic User Interface

- Save Images:
  - C/C++: **imwrite(filename, image, params);**
  - Java: **Mat *image* = Highgui. imwrite("path/to/img", image);**
  - **params (optional):** Format-specific save parameters encoded as pairs paramId_1, paramValue_1, paramId_2, paramValue_2,.... The following parameters are currently supported:
    - For **JPEG**, it can be a quality (CV_IMWRITE_JPEG_QUALITY) from 0 to 100 (the higher is the better). Default value is 95.
    - For **PNG**, it can be the compression level (CV_IMWRITE_PNG_COMPRESSION) from 0 to 9. A higher value means a smaller size and longer compression time. Default value is 3.
    - For **PPM**, **PGM**, or **PBM**, it can be a binary format flag (CV_IMWRITE_PXM_BINARY), 0 or 1. Default value is 1.

Graphic User Interface

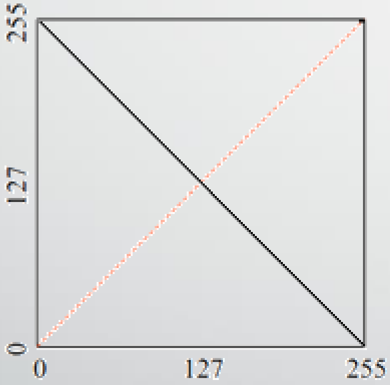
- Read from USB WebCam

```
VideoCapture camera = new VideoCapture(0);
Mat frame = new Mat();
camera.read(frame);
```

Transform Mapping

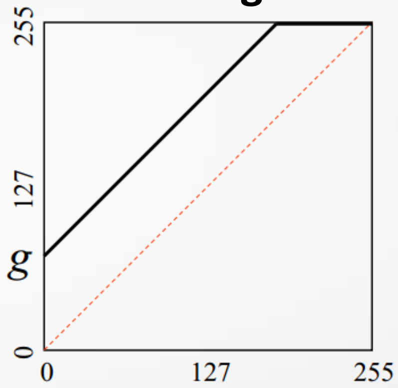
src.convertTo(dst, rtype, alpha, beta);

- *dsy(x,y) = saturate(alpha*src(x,y) + beta)*

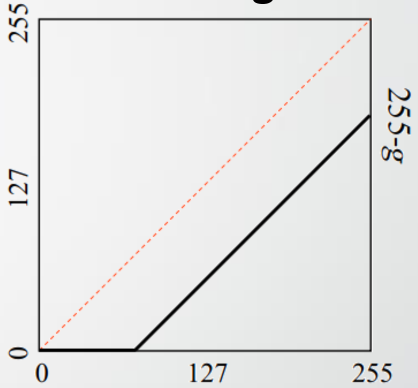


	Negative
	alpha = -1; beta =0

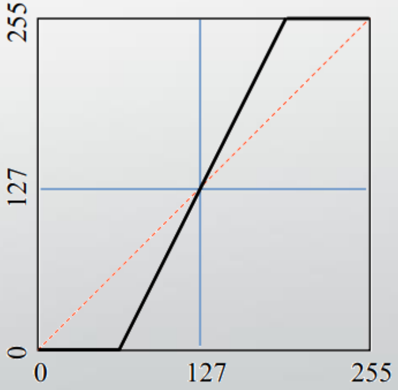
Increase Brightness



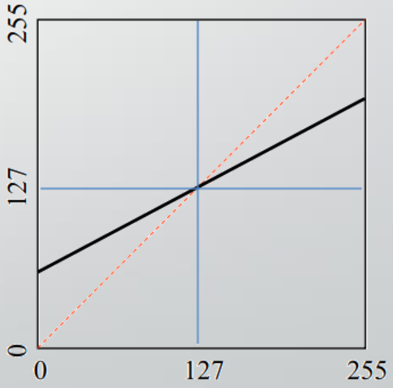
Decrease Brightness



Increase Contrast



Decrease Contrast



	Brightness	Contrast
+	*alpha* = 1; beta > 0	*alpha* > 1; beta = 127-(127*alpha)
-	*alpha* = 1; beta < 0	*alpha* < 1; beta = 127-(127*alpha)

Filtering

- General approach to filtering (convolution):

filter2D(Mat *src*, Mat dst, int ddepth, Mat kernel, Point anchor, Double delta, int borderType);

- Parameters:
  - **src** input image.
  - **dst** output image of the same size and the same number of channels as src.
  - **ddepth** desired depth of the destination image; if it is negative, it will be the same as src.depth();
  - **kernel** convolution kernel (or rather a correlation kernel), a single-channel floating point matrix; if you want to apply different kernels to different channels, split the image into separate color planes using "split" and process them individually.
  - **anchor (opt)** anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor is at the kernel center.
  - **delta (opt)** value added to the filtered pixels before storing them in dst.
  - **borderType (opt)** pixel extrapolation method (see "borderInterpolate" for details).

Filtering

- Built-in filters
  - **GaussianBlur** Blurs an image using a Gaussian filter
  - **medianBlur** Blurs an image using the median filter
  - **bilateralBlur** Applies the bilateral filter to an image
  - **Sobel** Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator
  - **Canny** Finds edges in an image using the Canny algorithm.
  - **HoughLines** Finds lines in a binary image using the standard Hough transform.)
  - **HoughLinesP** Finds line segments in a binary image using the probabilistic Houg transform.
  - **HoughCircles** Finds circles in a grayscale image using the Hough transform.