Image Processing and Computer Vision

Dhruv Shah
Om Kolhe
Shobhna Misra

Introduction to Image Processing

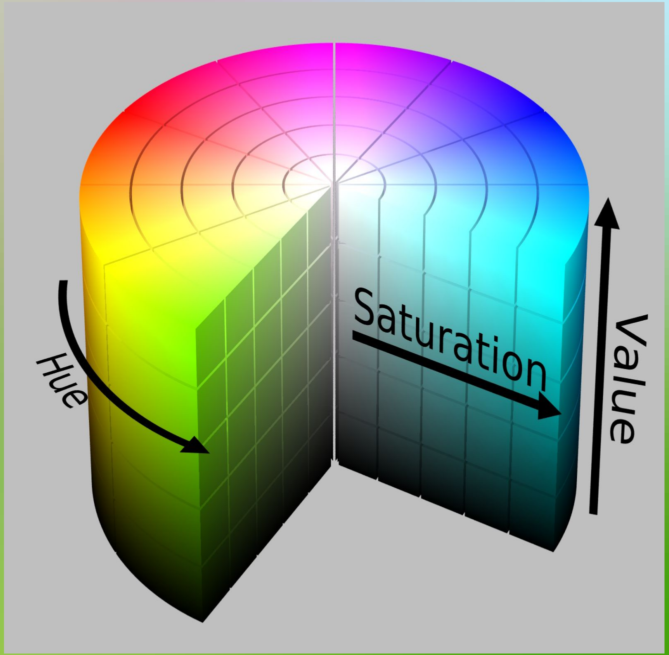
- Processing of images using mathematical operations
- Be it pattern recognition or image restoration, you can do endless stuff with Image Processing!

Representation of an image

- To store, sampling and quantisation (digitalisation)
- Sampling on regular grid of squares
- Each sample- pixel

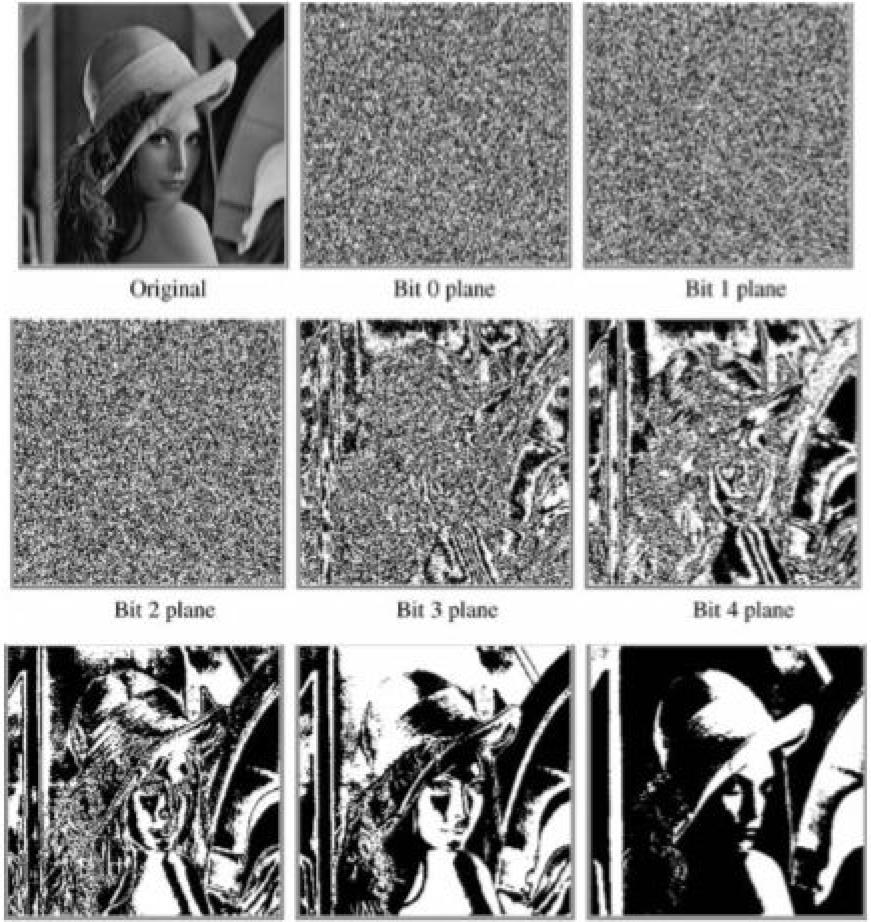
Color Schemes

- What is a color scheme?
- Types of color schemes.
- Types of multilayer color schemes
  - Red-Blue-Green
  - Hue-Saturation-Value
  - Cyan-Magenta-Yellow-Key
  - …
- Unique representation and converting.



Bit Planes

- Images as layers of matrices.
- Each pixel in the matrix represents a 8-bit number (uint8 or numpy.uint8 on Python).
- Imagine ‘slicing’ a grayscale layer into 8 bit planes.
- The layers represent decreasing significance and contribution to final image.



Original

Bit 0 plane

Bit 1 plane

Bit 2 plane

Bit 3 plane

Bit 4 plane

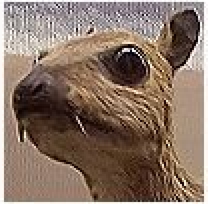
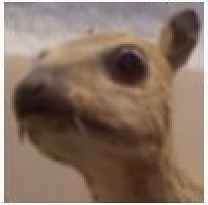
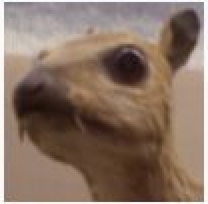
Bit 5 plane

Bit 6 plane

Bit 7 plane

Convolution / Kernel

- What is a convolution?
- What can the convolution do?

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Thresholding

- What is thresholding?
- Types of thresholding:
  - Absolute Thresholding
  - Otsu’s Method
  - Adaptive Thresholding

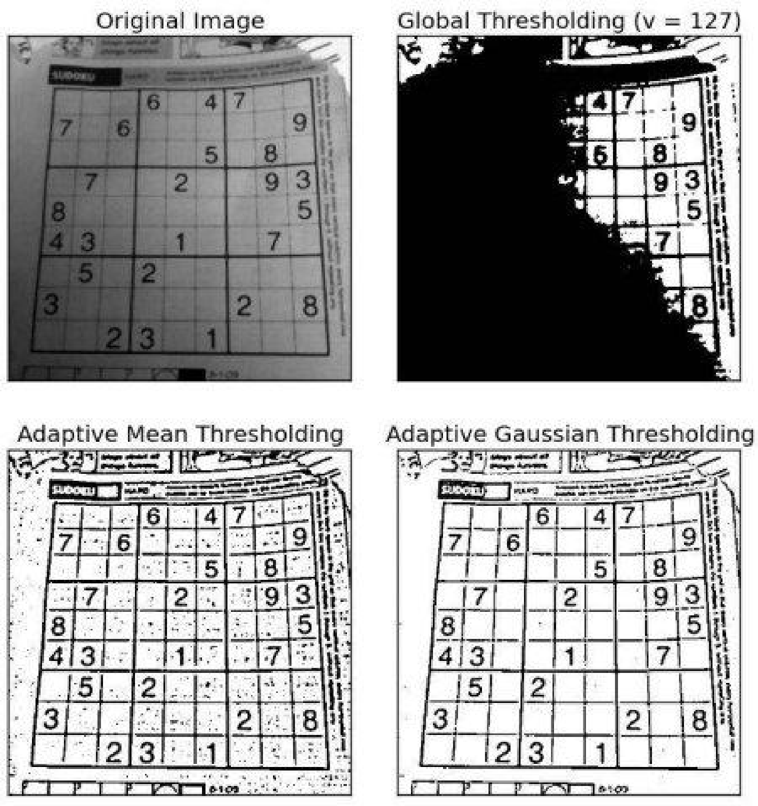

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Image Arithmetic

- Addition
- Subtraction
- Multiplication
- Division
- Logical Operators
- Bit Shift Operator

Arithmetic Operators

An image is represented in a matrix format.

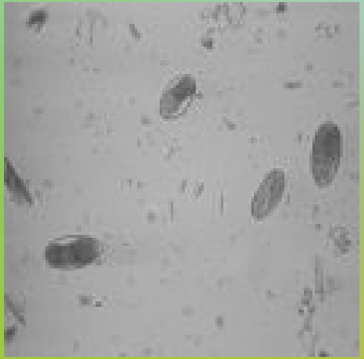
To perform image arithmetic the size of the two matrices should be same.

Image Addition

- Pixel-By-Pixel Addition
- RGB Image;
- C=A+B; minimum of A+B and 225 taken (saturation); wrapping

Application

- Add constant offset- change brightness of image



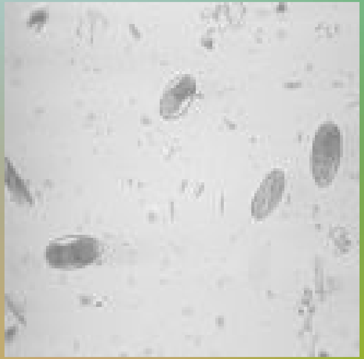


Image Subtraction

- Pixel-by-pixel subtraction
- Grayscale Image, RGB Image
- C=A-B; ie. Maximum value of A-B and zero; wrapping around

Applications

- Astrophotography: detection of asteroids/ Kuiper Belt objects
- Gesture Controlled bots
- Dark-frame subtraction to reduce image noise

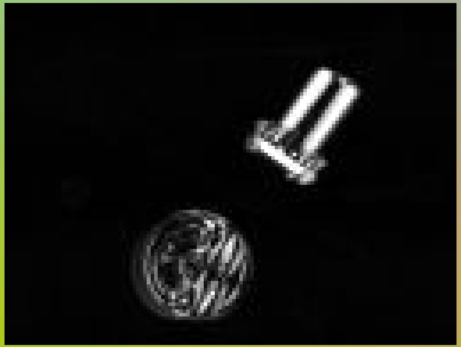


Image Multiplication

Image multiplication is used to increase the average gray level of the image by multiplying with a constant.

It is used for masking operations.

C=A.*B

Image Division

Image division can be considered as multiplication of one image and the reciprocal of other image.

C=A./B

Logical Operators

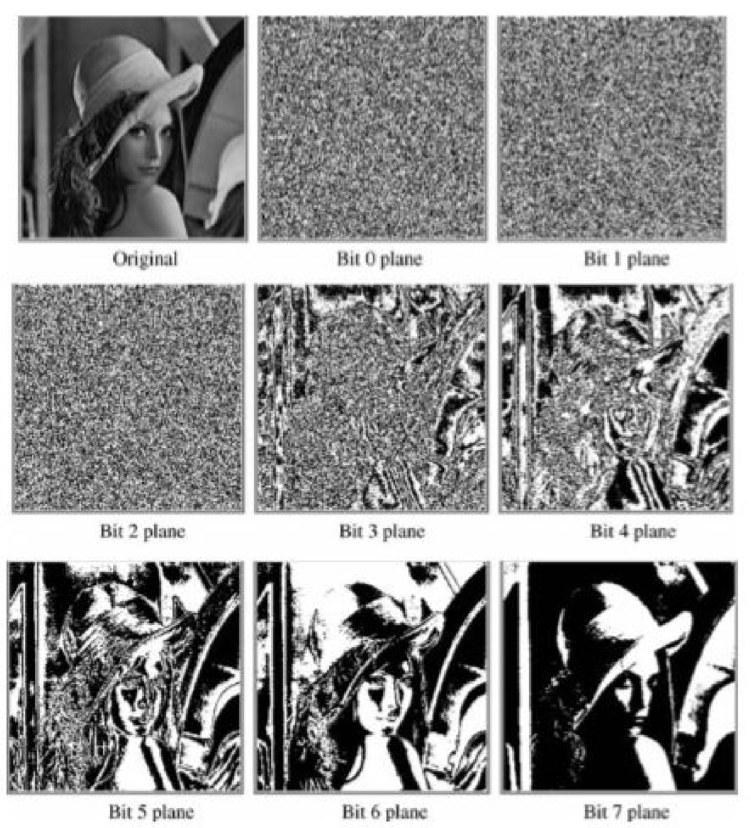
- AND
- NAND
- OR
- NOR
- XOR
- XNOR
- NOT

AND OPERATOR

- Intersection of two images- pointwise logical AND
- Used in masking and bit-slicing

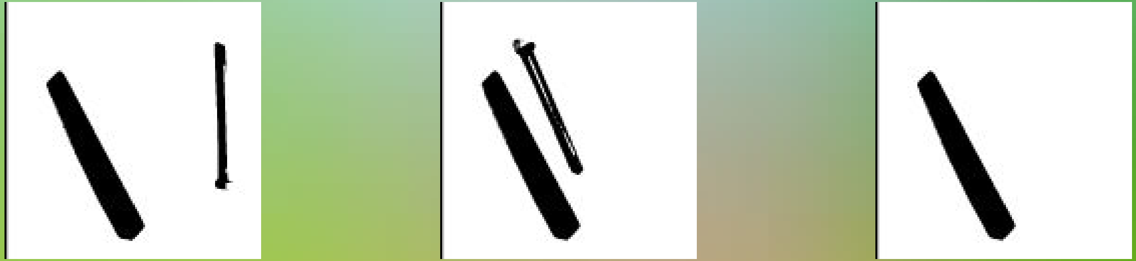
Masking

Bit Slicing


Original
Bit 0 plane
Bit 1 plane
Bit 2 plane
Bit 3 plane
Bit 4 plane
Bit 5 plane
Bit 6 plane
Bit 7 plane

OR operator

- Union of images- Point-wise logical OR
- Example-



Bit shift operator

- Divide/ Multiply an image by a power of 2
- Advantage over normal multiplication/division- computationally less expensive

Morphological Operators

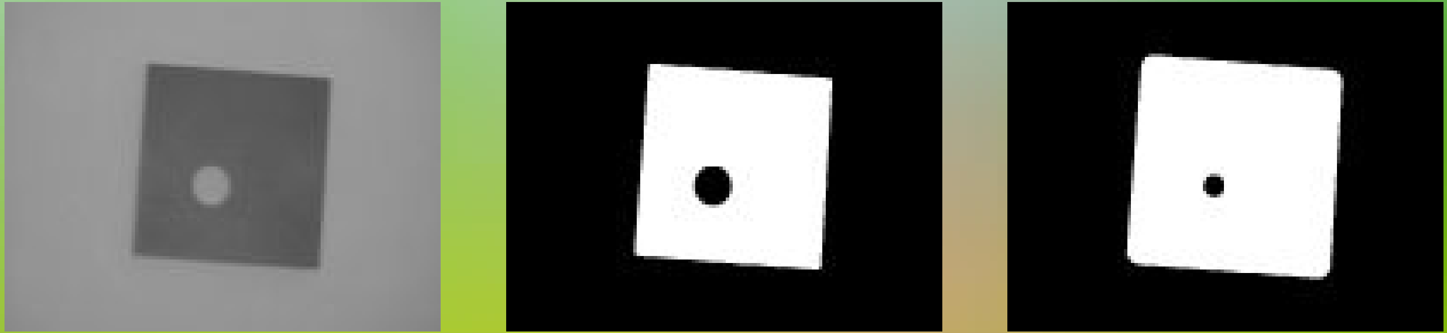
- Dilation
- Erosion
- Opening
- Closing
- Hit and Miss Transform
- Thinning
- Thickening
- Skeletonization/Medial Axis Transform

Dilation

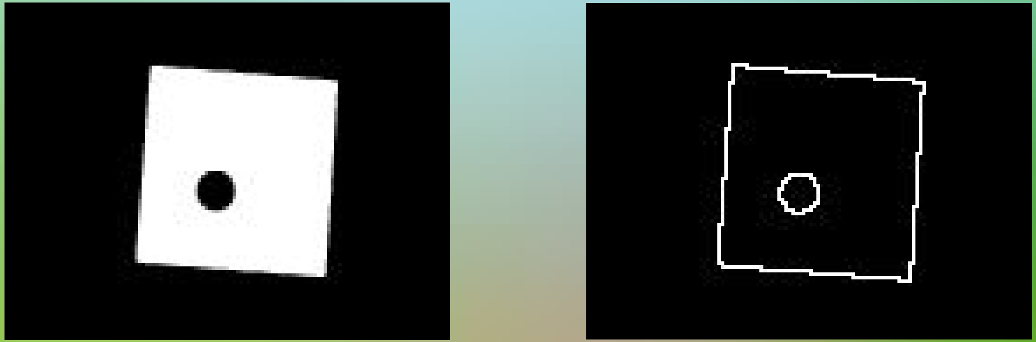
- Grow image regions
- Wide applications-can also be used for edge detection

Applications

Basic effect of dilation-



Edge detection- take dilation of an image, subtract the original image



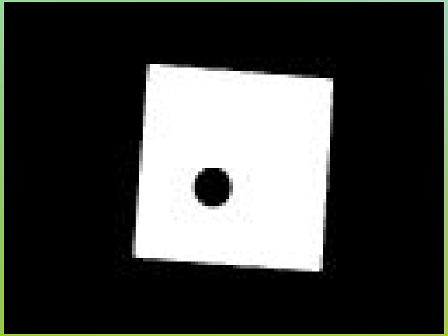
A specialist application of dilation- can be used to fill in spurious holes (pepper noise) in images.

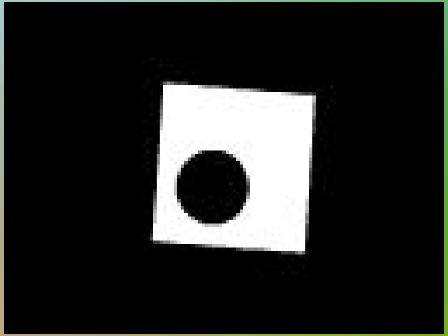
Erosion

- Applied mainly on binary images
- Erodes away boundaries of regions of foreground pixels
- Shrinking of binary images

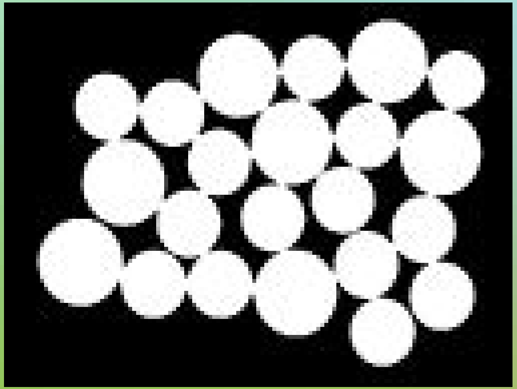
Applications

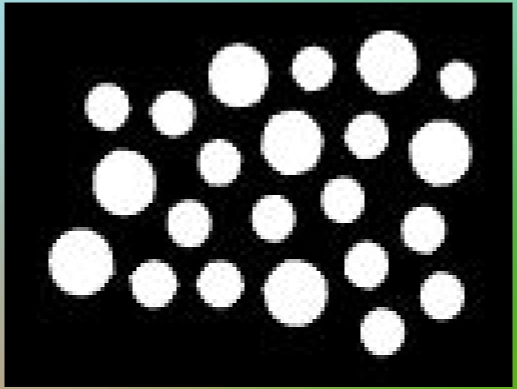
A simple example of erosion-





Difficult to count coins without erosion.





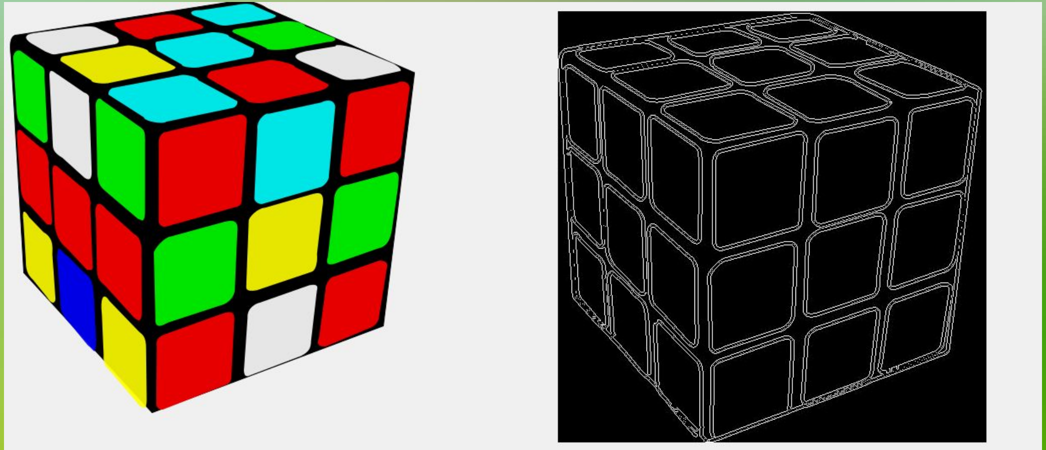
- Can also be used to remove small spurious bright spots (salt noise) in images.
- Can also be used for edge detection.

Edge Detection

- Edges are calculated by using difference between corresponding pixel intensities of an image.

Different methods:

- Sobel Operators
- Prewitt Operators
- Canny edge detection



Some more filters

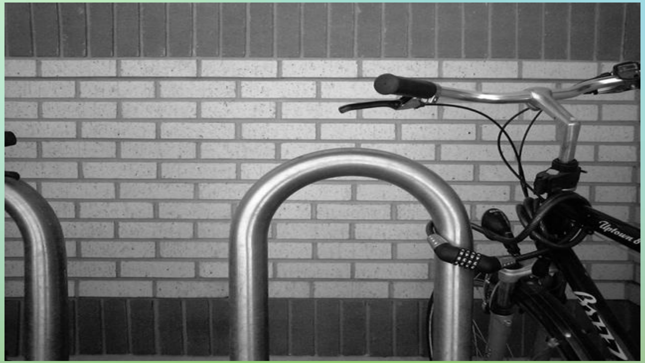
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

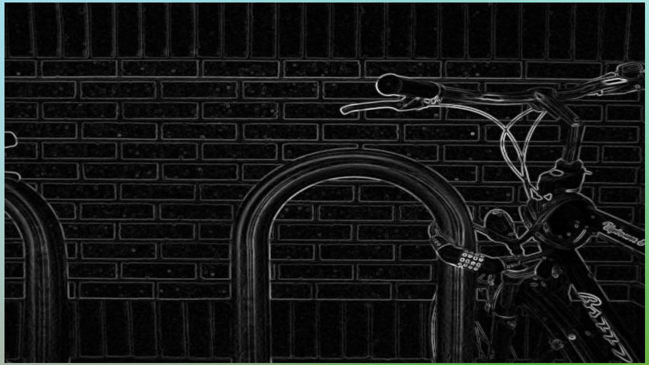
Sobel Operators

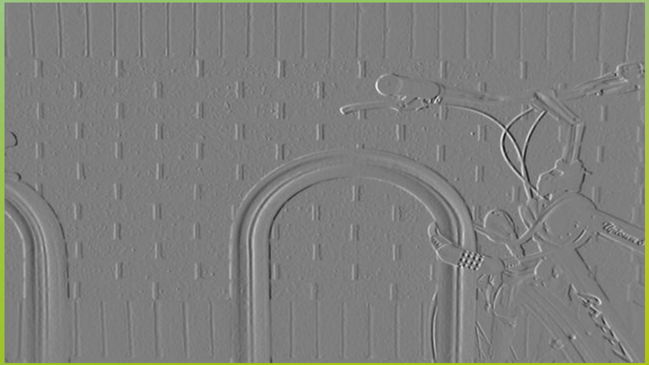
$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

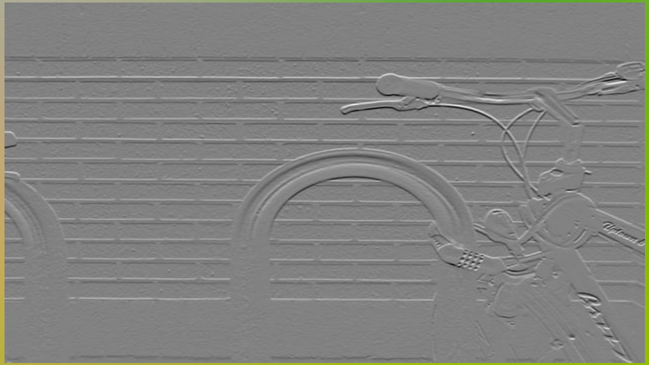
Prewitt Operators

Sobel Operator Output









Edge Detectors

- Sobel and Prewitt operators are the edge detectors.
- However instead of a single thin line for an edge, they give a thick line.
- Hence we use Canny edge detector.

This is not a good edge!

Canny Edge Detector

Canny edge detectors tries to achieve the following

- Low error rate: ie all the edges should be detected
- A given edge should be marked only once, ie we should get a thin line as output for an edge

Steps in canny edge detector

Step 1 Noise is filtered out – usually a Gaussian filter is used

Step 2 Finding the edge strength

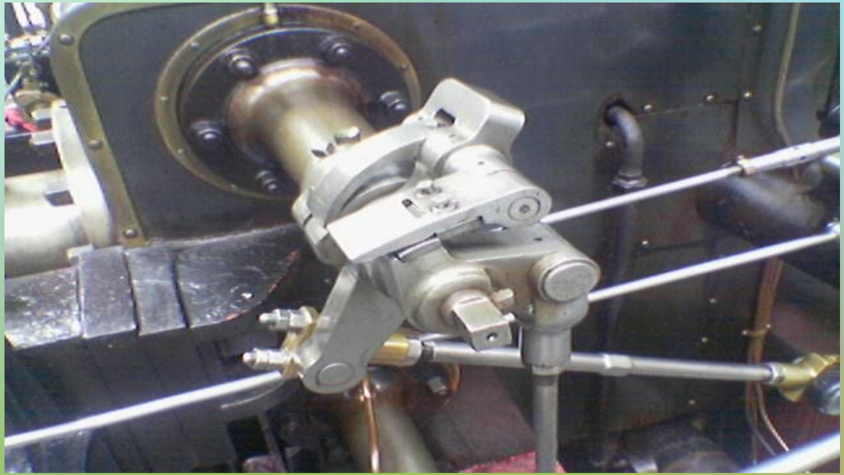
Step 3 Find the edge direction

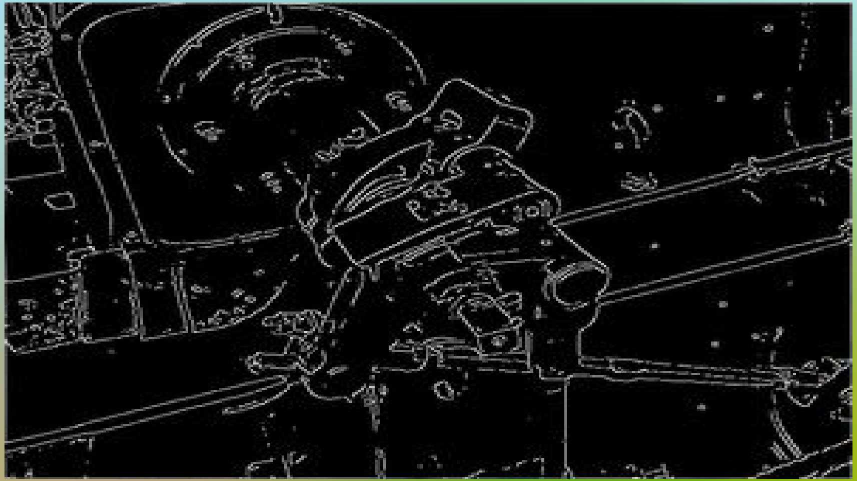
Step 4 Tracing the edge as per direction

Step 5 Non-maximum suppression

Step 6 Use hysteresis thresholding to eliminate streaking

Canny Results



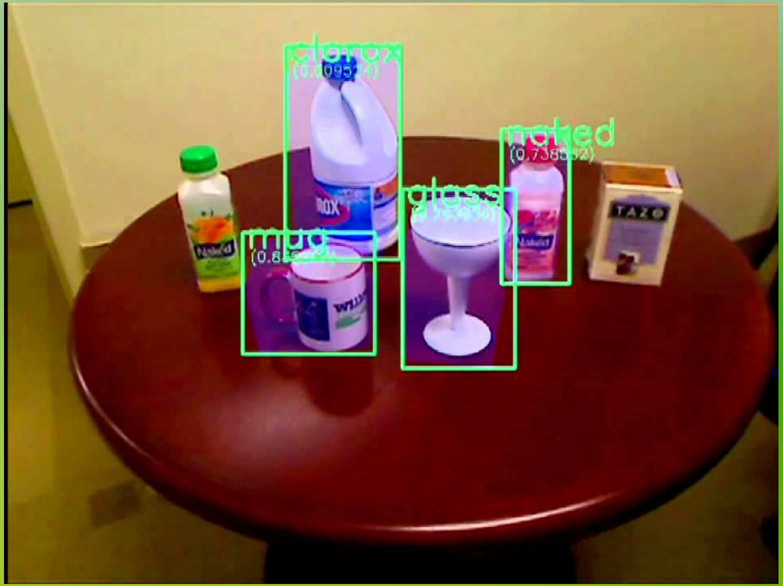


What is OpenCV ?

- **O**pen source **C**omputer **V**ision library.
- Originally developed by Intel.
- It has more than 2500 optimised algorithms.
- Latest Version - 3.1.0
- Supports a lot of different languages
  - C, C++, Python and Java.
  - Cross Platform also available for Android and iOS.

VARIOUS APPLICATIONS OF OPENCV

- Human-Computer Interaction
- Object identification
- Object recognition
- Face recognition
- Gesture recognition
- Motion tracking
- Image processing
- Real time tracking

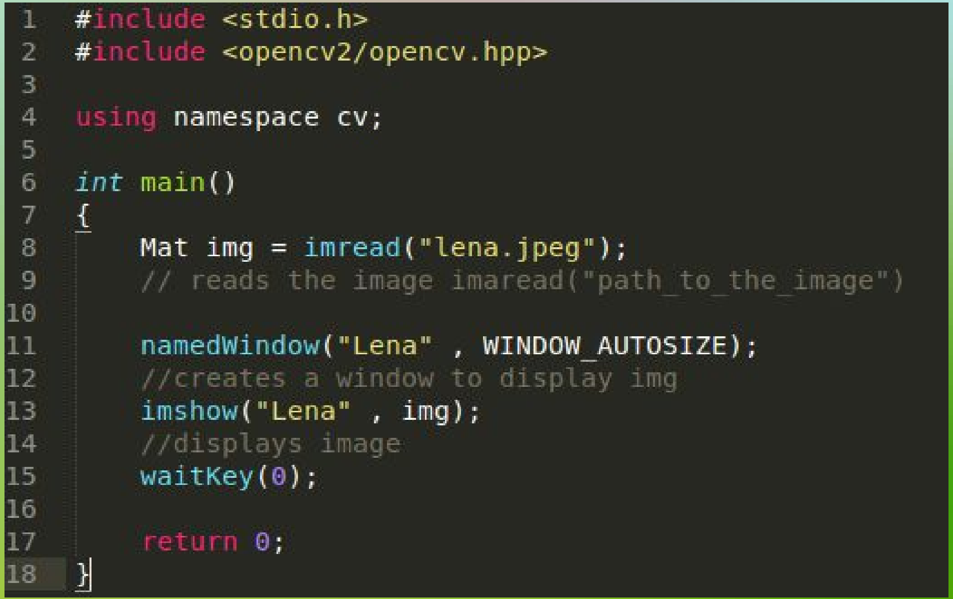


Why OpenCV ?!

- Integrated Development Environment
- Speed
  - Around 30 frames processed per seconds in real time image processing (OpenCV)
  - Around 4-5 frames processed per seconds in real time image processing (Matlab)

Reading an image

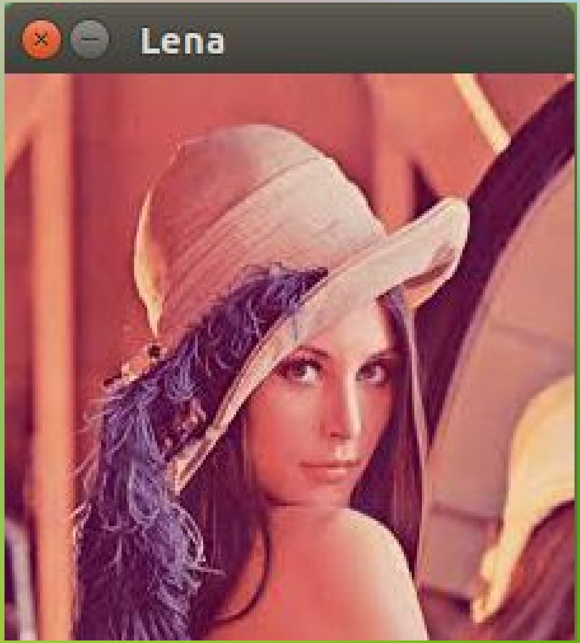
- The image “lena.jpeg” is stored in the variable img.
- Mat is the primary image structure in OpenCV.
- Two data parts:
  - Matrix header
  - pointer



```
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;

int main()
{
    Mat img = imread("lena.jpeg");
    // reads the image imaread("path_to_the_image")

    namedWindow("Lena" , WINDOW_AUTOSIZE);
    //creates a window to display img
    imshow("Lena" , img);
    //displays image
    waitKey(0);

    return 0;
}
```

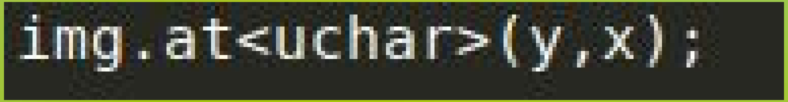
Displaying the image

```
Imread(“path_to_image”)

    - reads the image

namedWindow(title , size)

    - creates a window

imshow(window title , image)

    - displays the image

imwrite(name of the file , image)

    - creates a new image
```

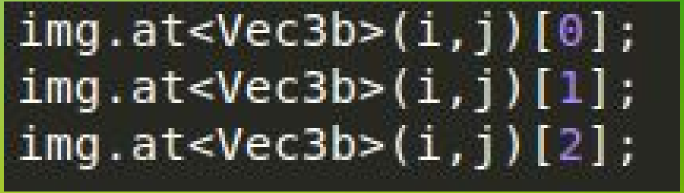


PIxels

- Mat stores the images in a form of matrix of the pixel values of the image.
- Each value is a 8 bit number. So the range of the pixel values is 0-255.
- In grayscale- 0-->black and 255--> white
- OpenCV stores the color image as a matrix of Scalars.
- A colored image is stored in the form of **BGR** (Not RGB)
- Each pixel is can be accessed using

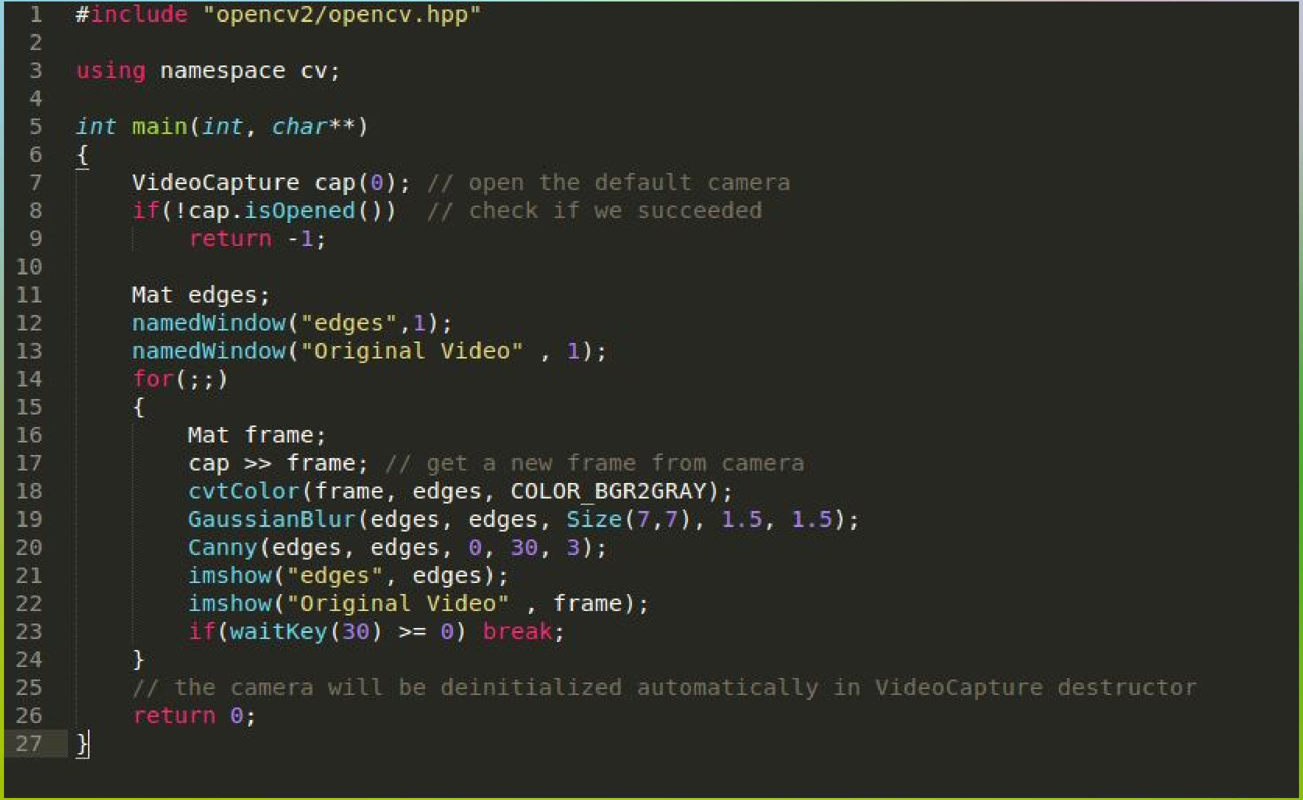


```
img.at<uchar>(y,x);
```



```
img.at<Vec3b>(i,j)[0];
img.at<Vec3b>(i,j)[1];
img.at<Vec3b>(i,j)[2];
```

Taking a video from the webcam



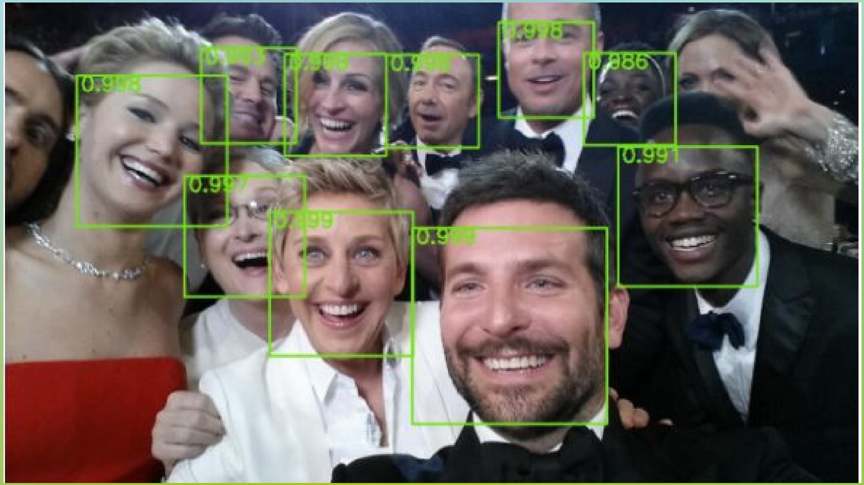
```cpp
#include "opencv2/opencv.hpp"

using namespace cv;

int main(int, char**)
{
    VideoCapture cap(0); // open the default camera
    if(!cap.isOpened())  // check if we succeeded
        return -1;

    Mat edges;
    namedWindow("edges",1);
    namedWindow("Original Video" , 1);
    for(;;)
    {
        Mat frame;
        cap >> frame; // get a new frame from camera
        cvtColor(frame, edges, COLOR_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
        Canny(edges, edges, 0, 30, 3);
        imshow("edges", edges);
        imshow("Original Video" , frame);
        if(waitKey(30) >= 0) break;
    }
    // the camera will be deinitialized automatically in VideoCapture destructor
    return 0;
}
```

Matlab

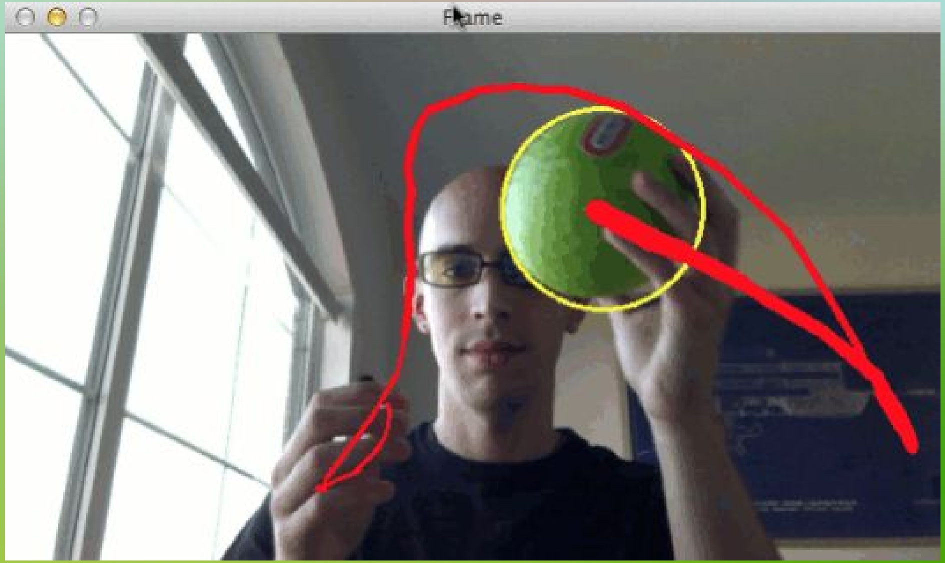
A fast-forward of everything we did on OpenCV.. Now in MATLAB! B|

Object Detection

Face detection



Real time tracking of a object



References (OpenCV)

- Installation - http://docs.opencv.org/3.1.0/df/d65/tutorial_table_of_content_introduction.html#gsc.tab=0
- Documentation - http://docs.opencv.org/3.1.0/index.html#gsc.tab=0
- Face Detection http://docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0
- Real time tracking http://docs.opencv.org/3.0-beta/modules/tracking/doc/tracker_algorithms.html