Electronics and Robotics Club, IIT Bombay

CODE THE PIXELS

An introduction to IMAGE PROCESSING

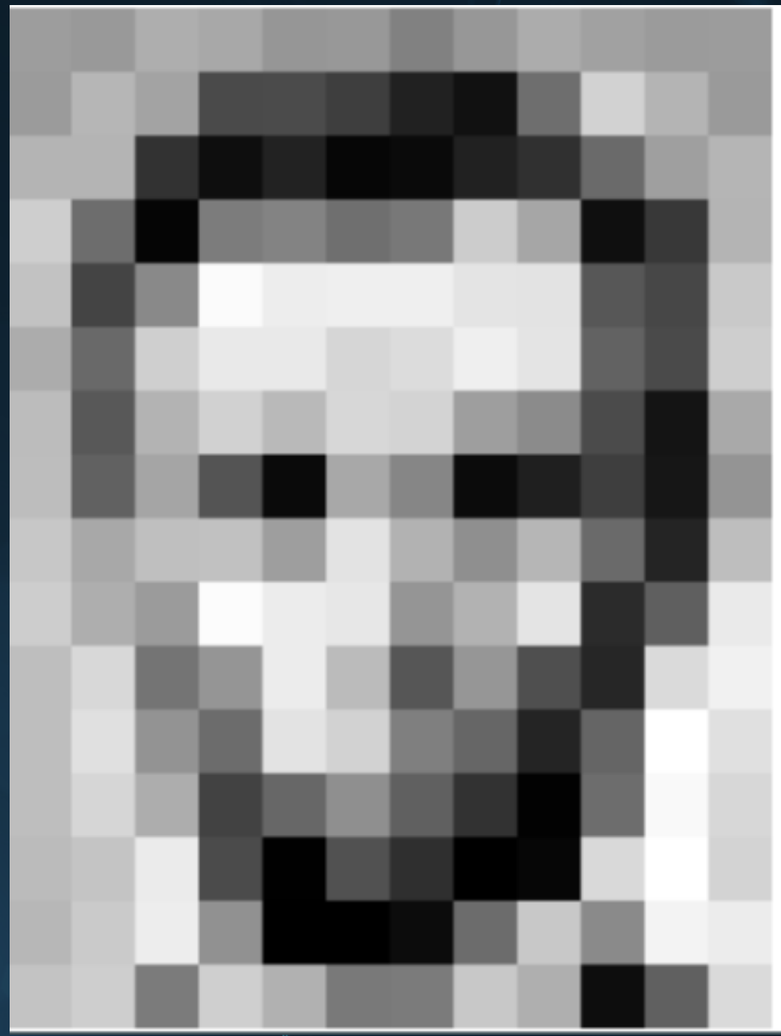
http://bit.ly/erc-ip

OVERVIEW

1. What are Images and What is Image Processing?
2. Reading images and videos in OpenCV
3. Thresholding
4. Colour/Object Detection using OpenCV
5. Blurring
6. Morphology
7. Edge Detection
8. Contours
9. Applications and Uses of Image Processing
10. Going “Deep"

Images as seen by Computers

- The digital images are not continuous
- They are divided into small blocks known as pixels(short for picture elements)
- Each pixel has a intensity value
  - Range: 0 to 255
  - 8-bit
- More the pixels, more the variations and so better the quality
- (Now you know why megapixel of camera matters)



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

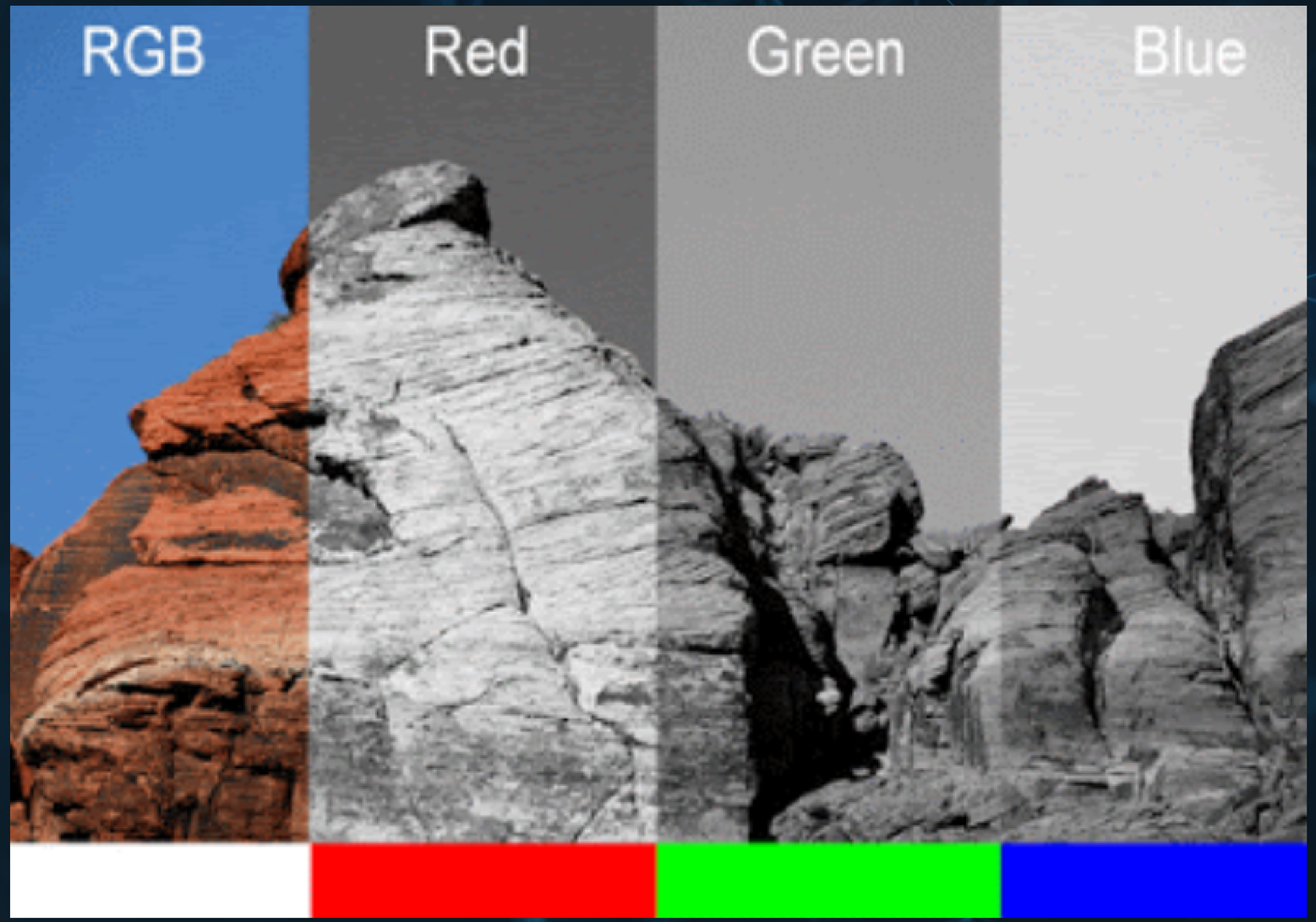
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

- More the intensity, whiter the pixel
- 255 = pure white
- 0 = pure black

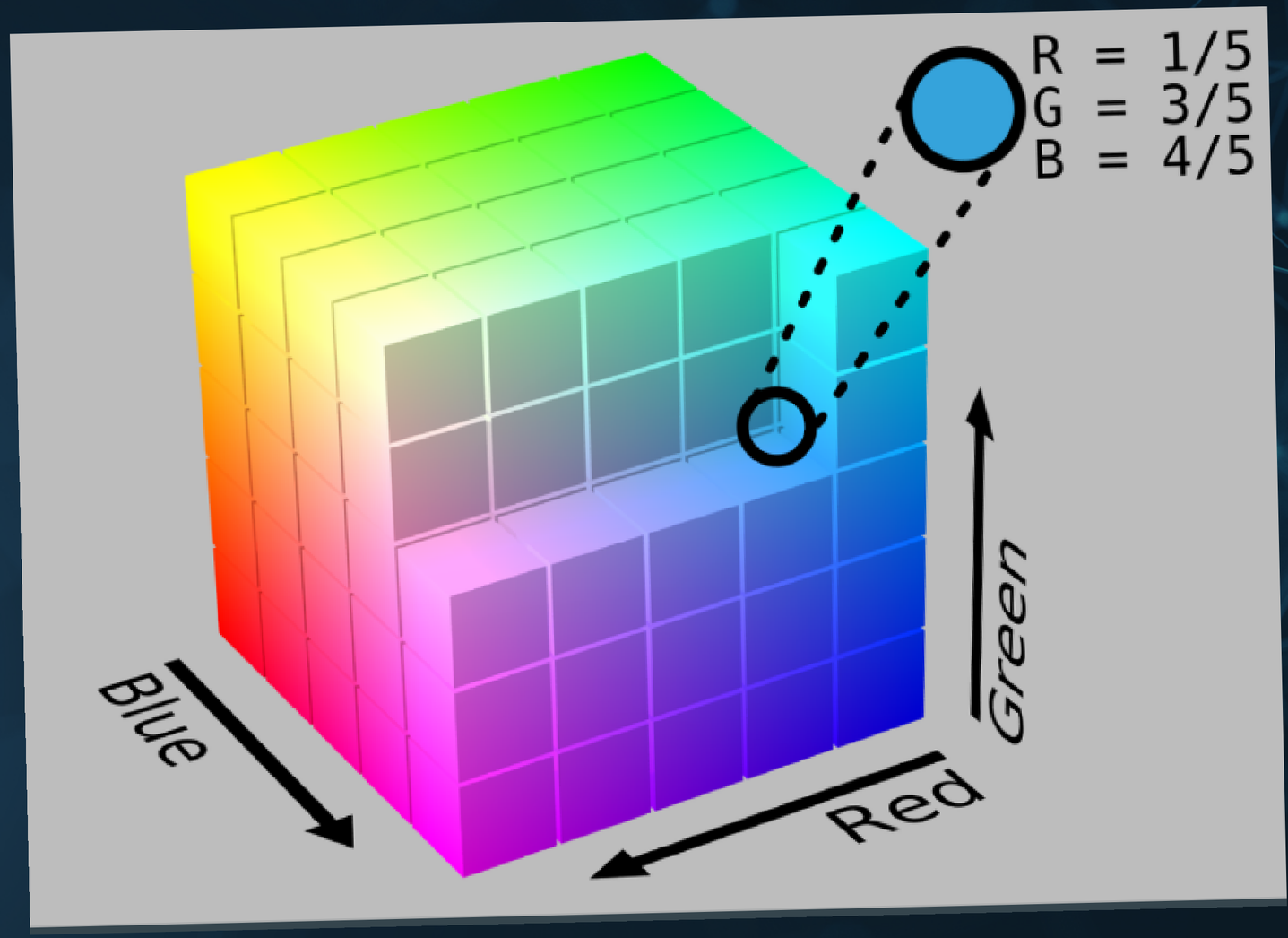

technology
label
search
engineering
reusable
information
options
presentation
modern
responsibility
laptop
choice
step
chart
inspiration
world
dramatic
color
internet
data
design
website
nature
people
business
process
innovation
digital
graphic
advertising
computer
media
art
image processing

Colors!!!

- Pixel no longer has 1 value
- Multiple Channels to accommodate colors
  - Each channel is same as a grayscale image



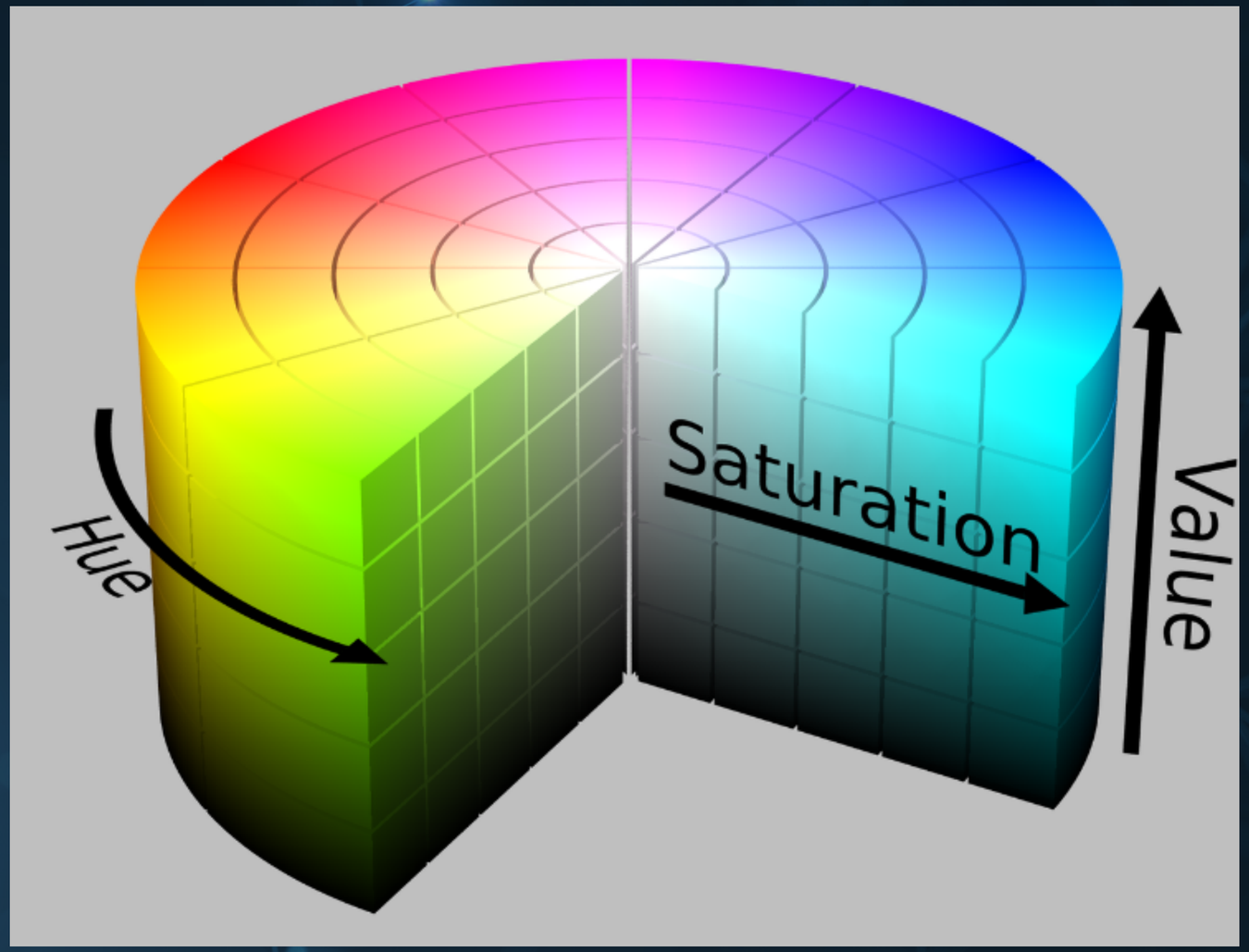
Color Formats



RGB format -

Red, Green, Blue format

OpenCV uses reverse format i.e. BGR.

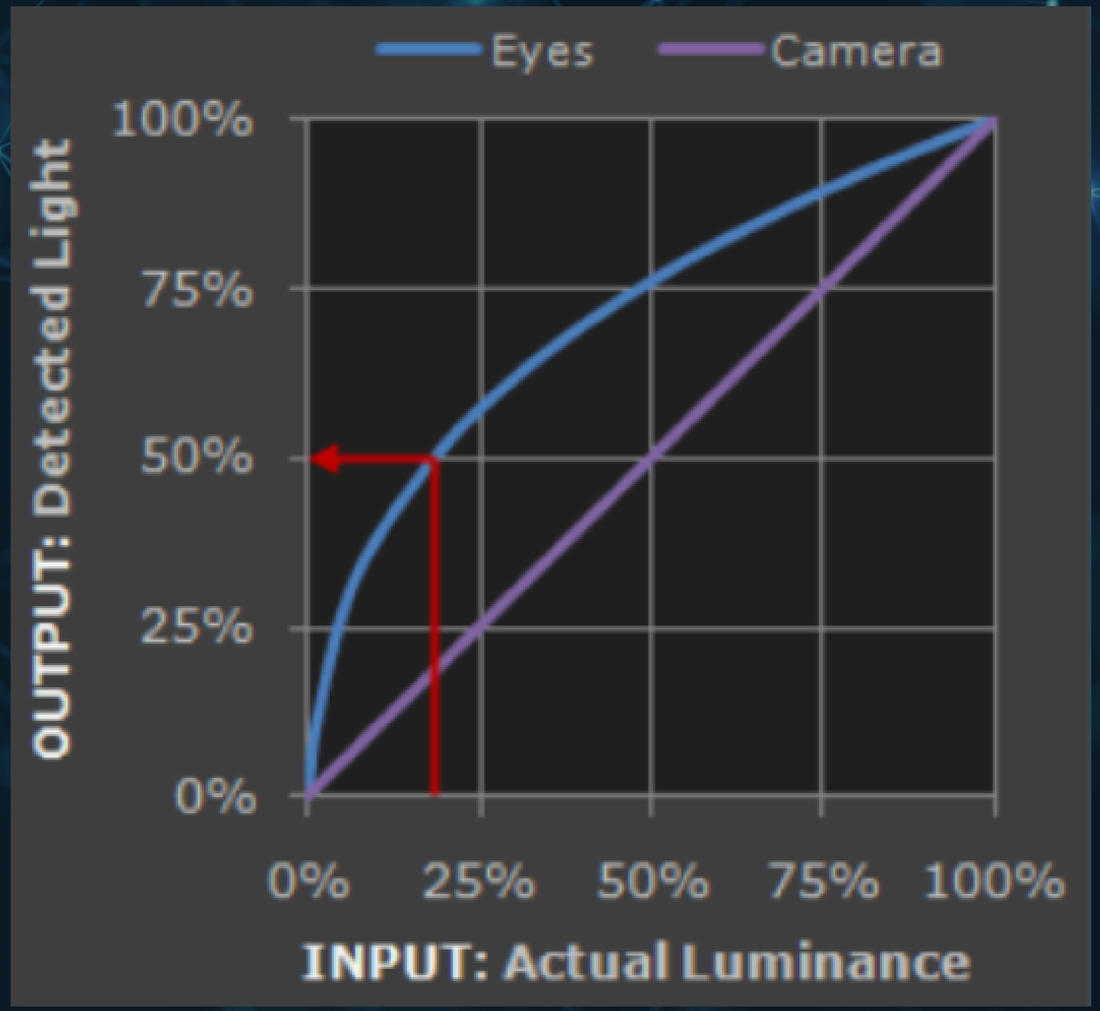


HSV format -

Hue, Saturation, Value format

Gamma Correction

- The human eye response is not linear with respect to intensity
- It responds more to lower intensity level changes
- Power Law:
  Detected = (Luminance)^ɣ
-
- Correction is done by taking the root




γ=2
γ=1/2
γ=1 (original)
γ=1/3

OpenCV Basics

Reading an Image in OpenCV

```
import cv2

#Reading an image
image = cv2.imread("joker_1.jpg", 1)

#Resizing the image because it is huge!
image = cv2.resize(image, (720, 720))

#Displaying the image
cv2.imshow('window', image)

k = cv2.waitKey(0) #imshow does not render without waitK
if k == 95:
    cv2.destroyAllWindows()
```

cv2.imread(file_name, flag)

FLAG:

0 - Grayscale

1 - Coloured(BGR) image

-1 - Coloured image with alpha channels

Showing an Image in OpenCV

```python
import cv2

#Reading an image
image = cv2.imread("joker_1.jpg", 1)

#Resizing the image because it is huge!
image = cv2.resize(image, (720, 720))

#Displaying the image
cv2.imshow('window', image)

k = cv2.waitKey(0) #imshow does not render without waitKey
if k == 95:
    cv2.destroyAllWindows()
```

cv2.imshow(Window_name, Image)

Window_name:
Name displayed on the display

Image:
The image to be showed

Showing an Image in OpenCV

```python
import cv2

#Reading an image
image = cv2.imread("joker_1.jpg", 1)

#Resinzing the image because it is huge!
image = cv2.resize(image, (720, 720))

#Displaying the image
cv2.imshow('window', image)

k = cv2.waitKey(0) #imshow does not render without waitKey
if k == 95:
    cv2.destroyAllWindows()
```

cv2.waitKey(delay)

delay:
How much time(msec.) to wait before closing the window

0: don’t close automatically

returns:
Returns the ASCII value of the key pressed.

Changing between Color Spaces

```
#Reading a image
frame = cv2.imread("joker.jpg", 1)
#Resizing the image because it is huge!
frame = cv2.resize(frame, (500, 500))
#Converting to HSV space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
cv2.imshow("Showing image", mask)
cv2.imshow("Showing hsv", hsv)
k = cv2.waitKey(0)
if k == 37:
    cv2.destroyAllWindows()
```

cv2.cvtColor(src, code)

src:
Source image

code:
Code to specify the change in color space
General Format:
COLOR_<current>2<wanted>

https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html

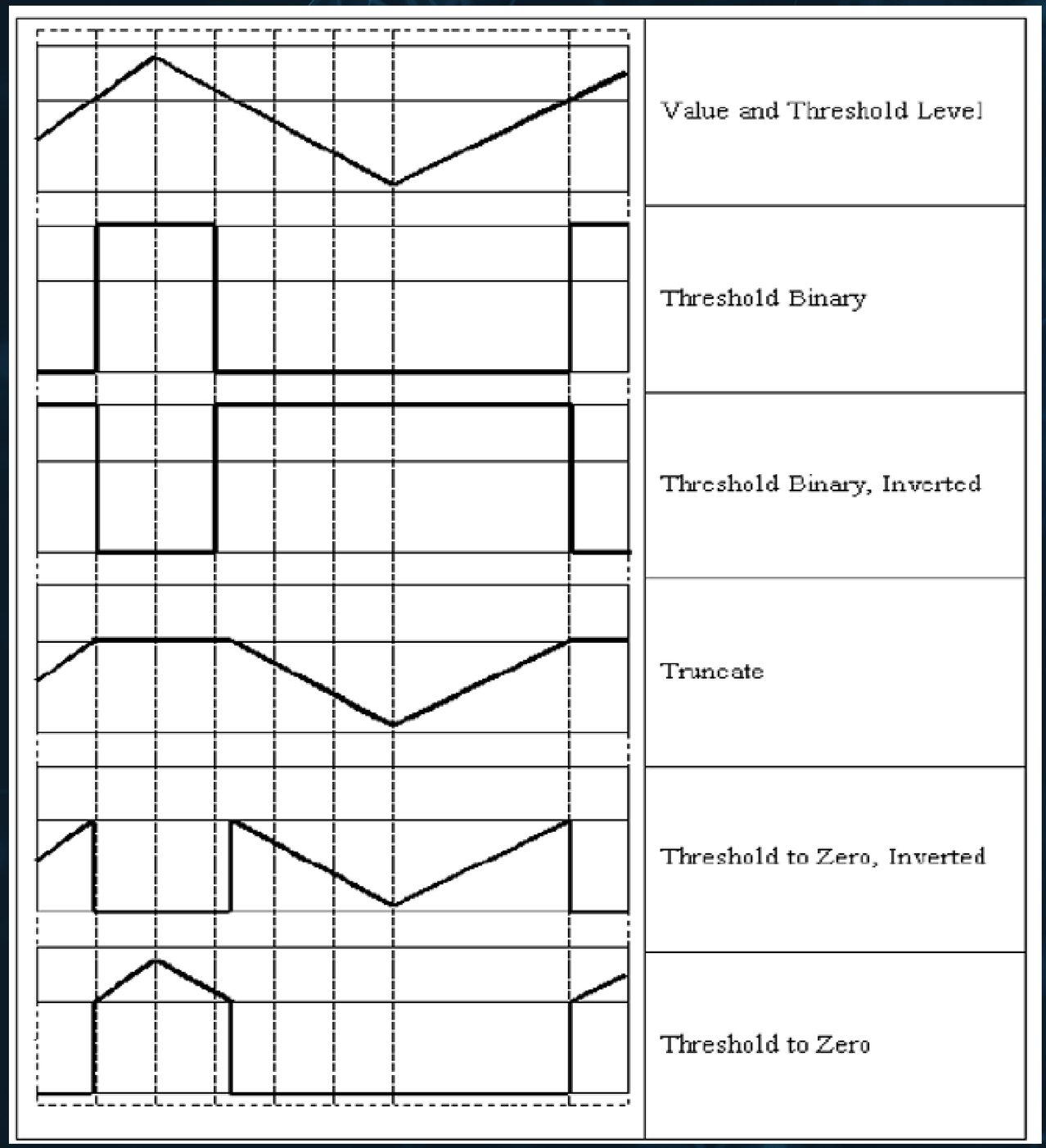
Real Processing Starts Now!

Thresholding

Pendulum Tracking using Thresholding

- Simple Thresholding
- Adaptive Thresholding

5 types of thresholding supported by openCV



Simple Thresholding

- Replace each pixel with some value depending on a given threshold.
- Converts the grayscale image to binary image.

THRESH_BINARY Python: cv.THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$

Simple Thresholding - OpenCV

cv2.threshold(src, thresh, maxval, type)

src:
The (grayscale) image to be thresholded

thresh:
The threshold to be used

maxval:
The maximum value to be used in THRESH_BINARY

type:
Type of thresholding to be used

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(gray, 127, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(gray, 127, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(gray, 127, 255, cv2.THRESH_TOZERO_INV)
```

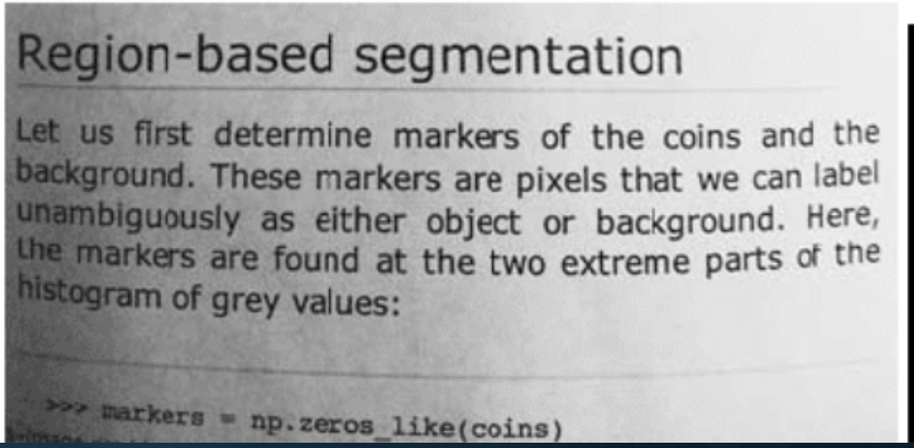
Adaptive Thresholding

- Adaptive thresholding chooses a threshold for a pixel based on its neighboring pixels
- There are two ways to calculate this threshold:
  - Using mean: Given a window size (let's say 5 × 5) we calculated the mean for all pixels in that window and then use that a threshold. Note that you will need to specify what type of thresholding you need(one of the 5 types discussed before)
  - Using weighted mean: This is similar as above only that weights are chosen from a normal distribution

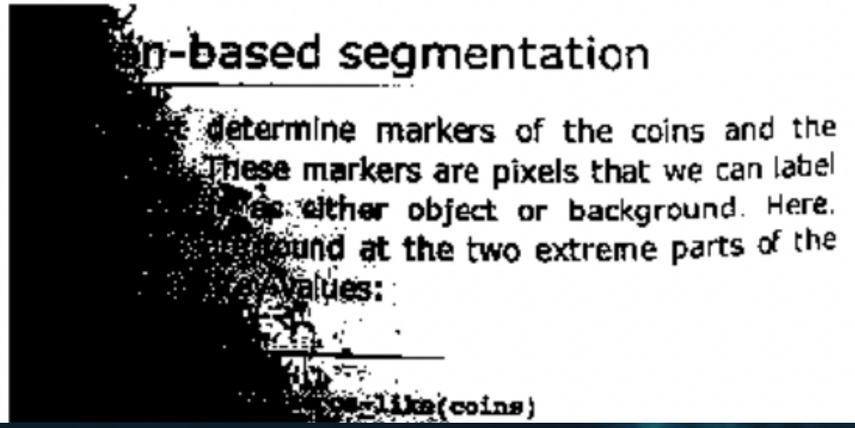
Src maxValue adaptiveMethod thresholdType(only binary ones allowed) blockSize C

```
#Adaptive threshold
adaptive_thresh1 = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
adaptive_thresh2 = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
```

Image



Global thresholding



Adaptive thresholding

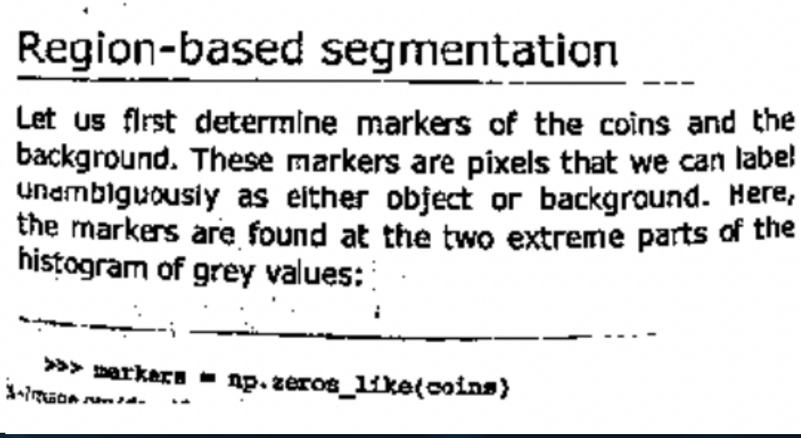


Figure 1: *Top:* Original input image. *Middle:* Applying global thresholding leads to a poor segmentation result. *Bottom:* Using adaptive thresholding creates a much cleaner segmentation (**image source**).

Color Detection

Masking

Masking

- Mask allows us to focus on the portions of the image that interests us.
- Use HSV format and apply filter on Hue channel
- Good Range: [h-10, h+10]
- Saturation and Value are generally kept for whole range. So, change according to need

Color Detection

inRange(src, lowerb, upperb)

src:
The image to be thresholded

lowerb:
Lower bounds on the pixel intensity values

upperb:
Upper bounds on the pixel intensity values

RETURNS:
Binary mask

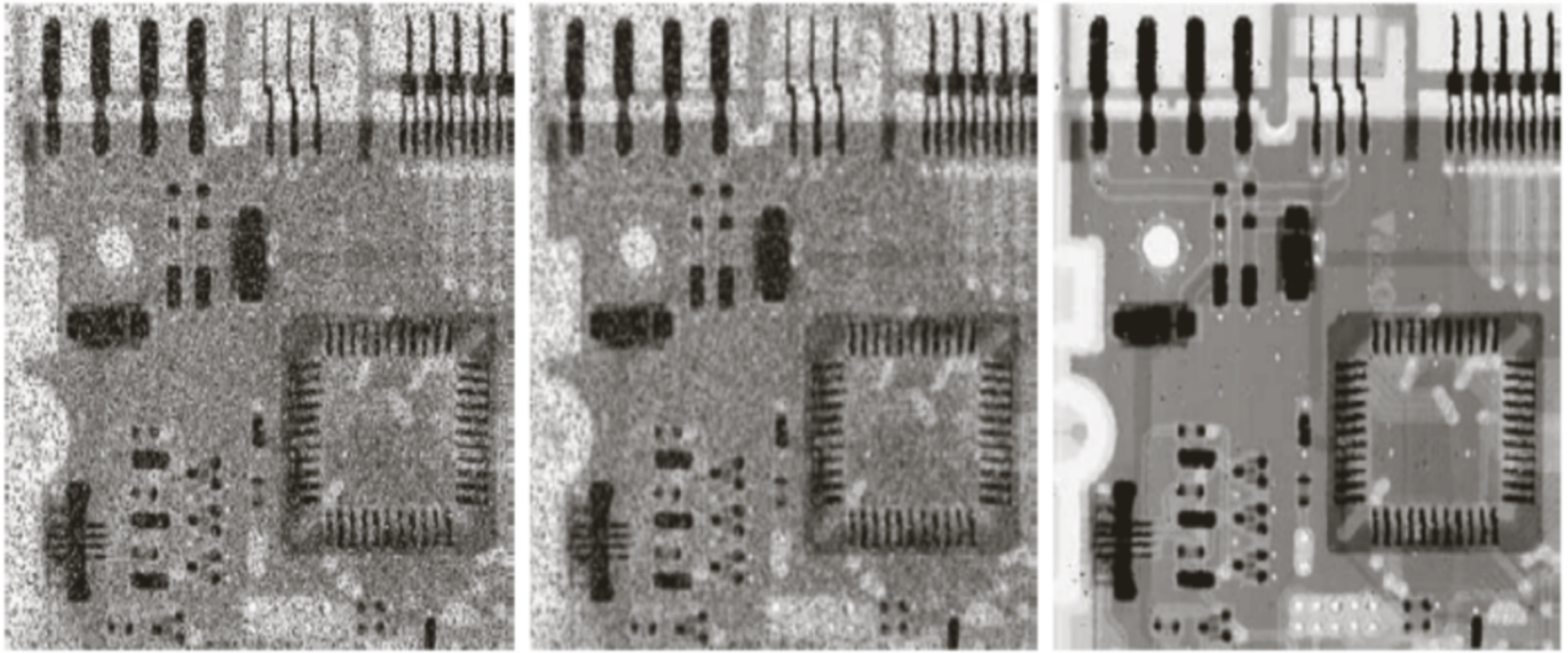
```
l_b = np.array([l_h, l_s, l_v])
u_b = np.array([u_h, u_s, u_v])

mask = cv2.inRange(hsv, l_b, u_b)

res = cv2.bitwise_and(frame, frame, mask=mask)
```

Note: In Opencv, Hue is between 0 and 180.

Blurring



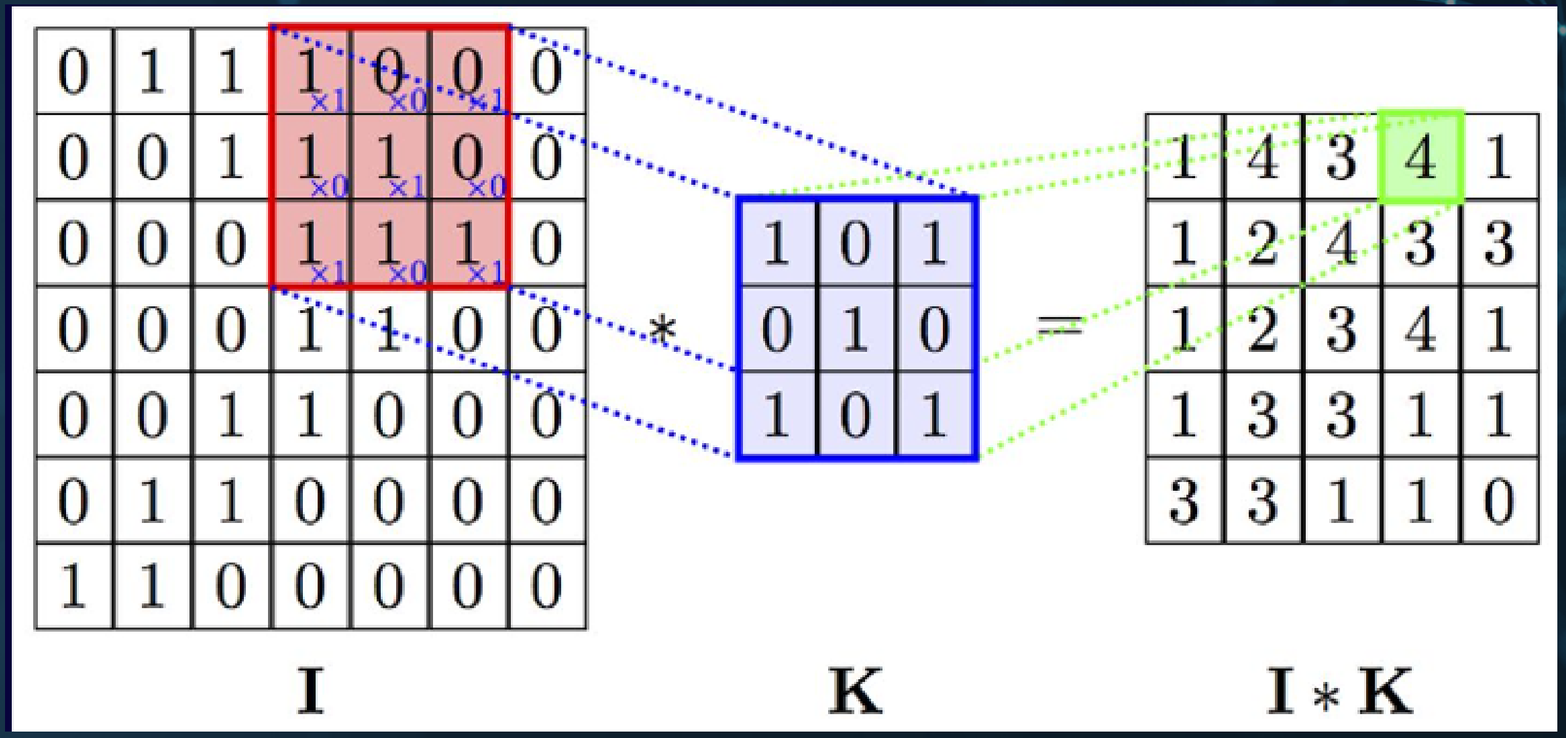
a b c

FIGURE 3.35 (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

- Averaging Blur(Box Filter)
- Gaussian Blur(Gaussian Filter)
- Median Blur(Median Filter)

Kernel and Convolution

- A kernel is nothing but a small matrix, also called convolution matrix.
- Convolution can be thought of as weighted averaging around a pixel
  - Weights = kernel



Kernel and Convolution - Padding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	25	100	75	49	130	0	0
0	0	50	80	0	70	100	0	0
0	0	5	10	20	30	0	0	0
0	0	60	50	12	24	32	0	0
0	0	37	53	55	21	90	0	0
0	0	140	17	0	23	222	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Averaging Blur(Box Filter)

- The center of a n✕n kernel is replaced by the average of the n^2 pixels in it.
- The kernel size is always an odd integer.

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

blur(image,size)

image: Image to be blurred

size: height and width of the kernel

RETURNS: blurred(convolved) image

```
blurred_image=cv2.blur(image, (3,3))
```

Gaussian Blur(Gaussian Filter)

- The center of a n×n kernel is replaced by the average of the n^2 pixels in it.
- The kernel size is always an odd integer.

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

GaussianBlur(image,size,sigmaX,sigmaY=None,borderType=None)

image: Image to be blurred

size: height and width of the kernel

sigmaX: standard deviation of the kernel in X direction(if 0, calculated from size)

Median Blur(Median Filter)

- The center of a n✕n kernel is replaced by the median of the n^2 pixels in it.
- Highly useful in reducing noise from images.
  - In the above filters, a pixel is set to a calculated value that might not belong to any surrounding pixels. In this method, we set a pixel to value which is present in its neighbourhood. This reduces noise very effectively.

medianBlur(image,size)
image: Image to be blurred

size: size of the window in which median is to be taken

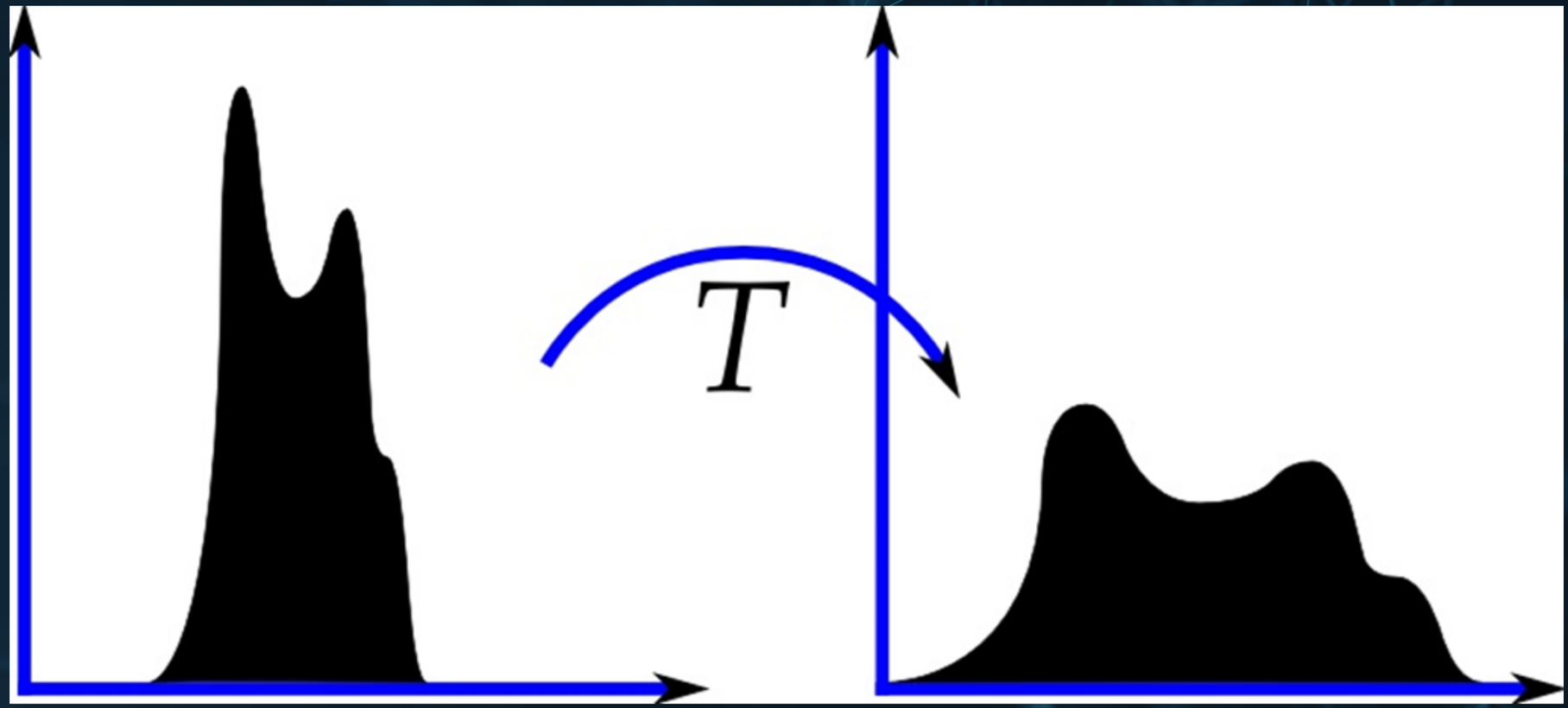
RETURNS: blurred(convolved) image

Noise Reduction using blur

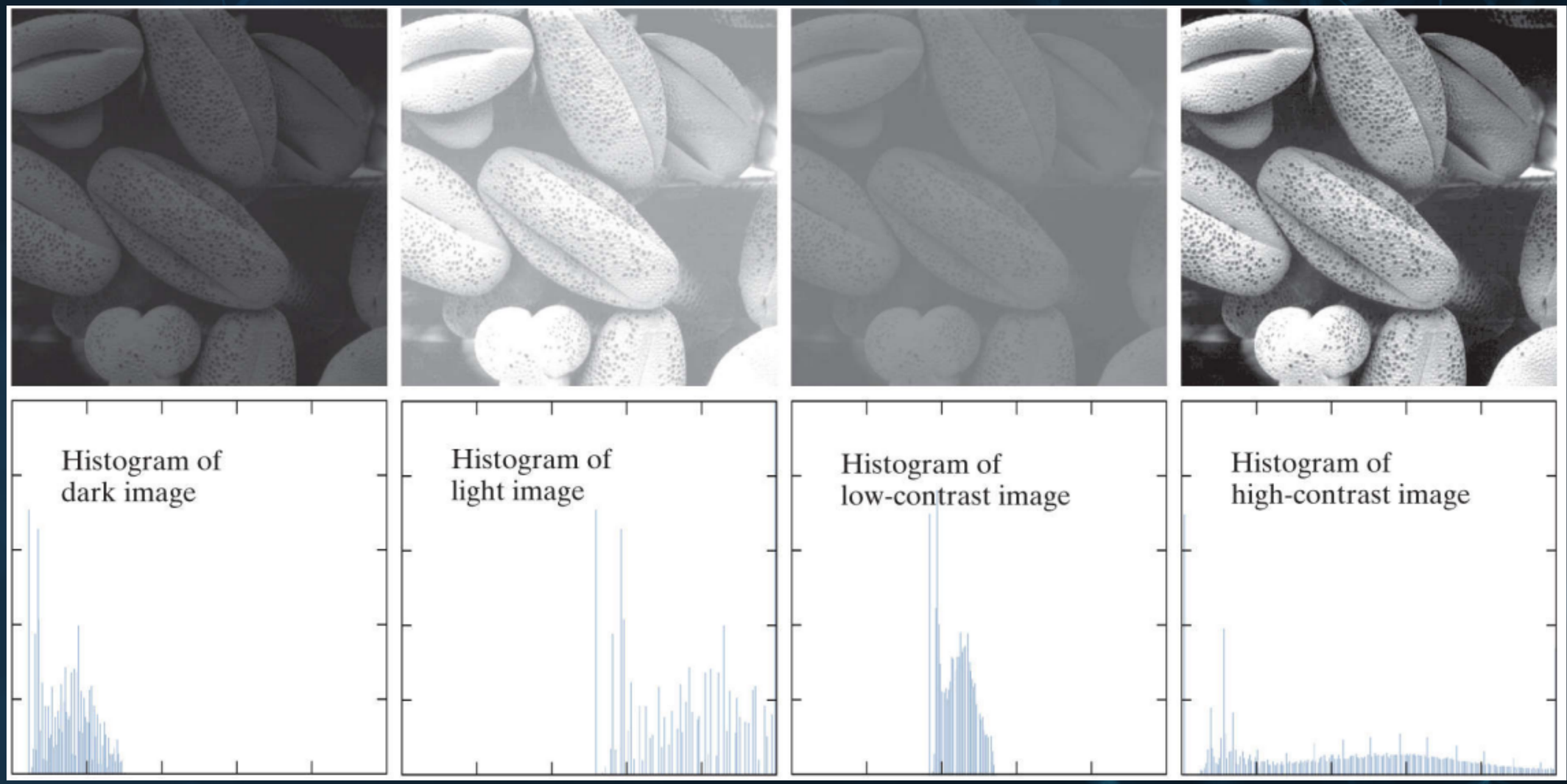
Some other Processing

Histogram

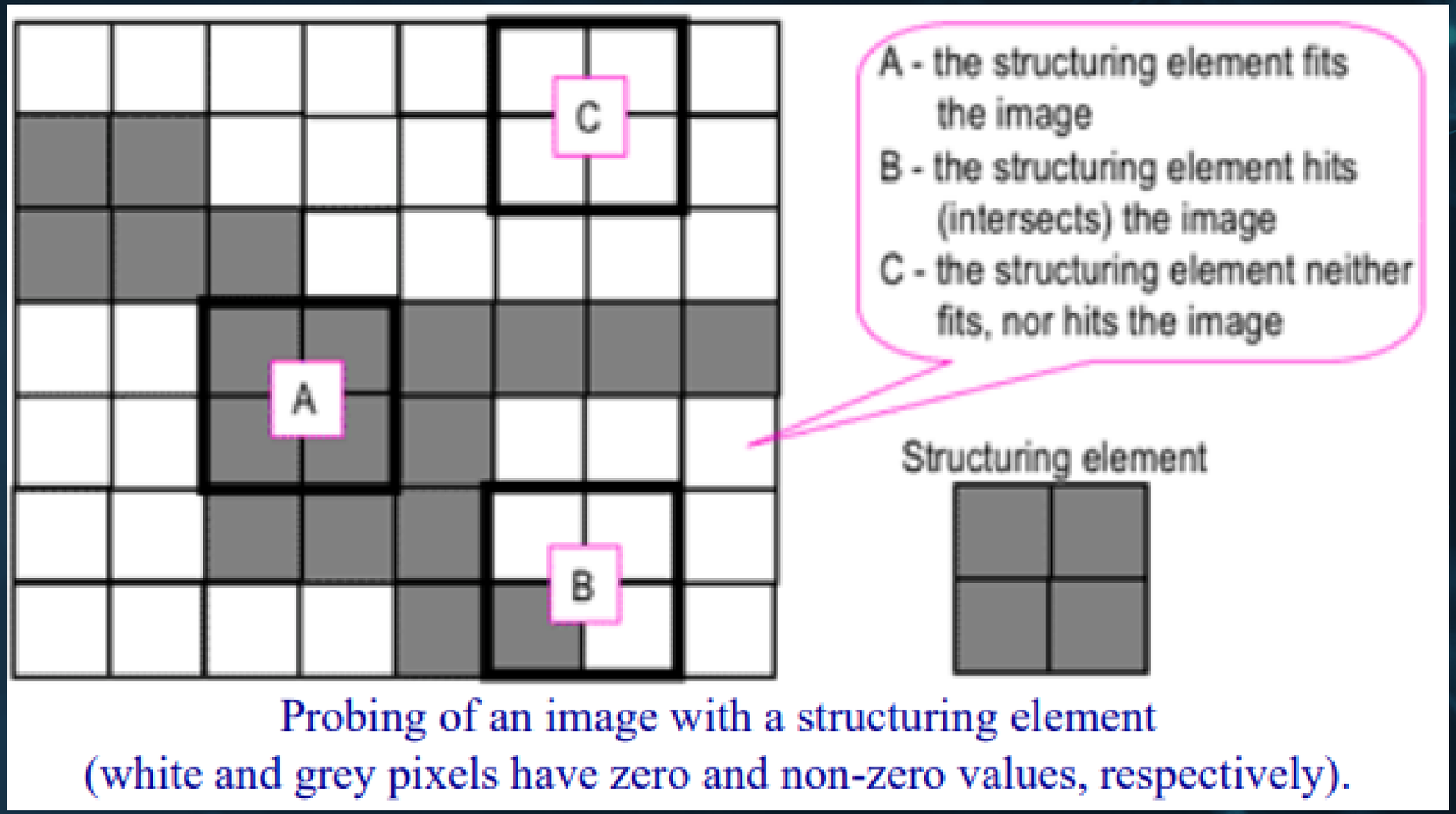
- Number of Pixels vs. Pixel Values
- More variance implies better "contrast"



Histogram Equalization



Morphology



Probing of an image with a structuring element
(white and grey pixels have zero and non-zero values, respectively).

Morphology

Erode -->

<-- Dilate

Morphology

Open - Erode and then Dilate

Essentially removes white dots

image

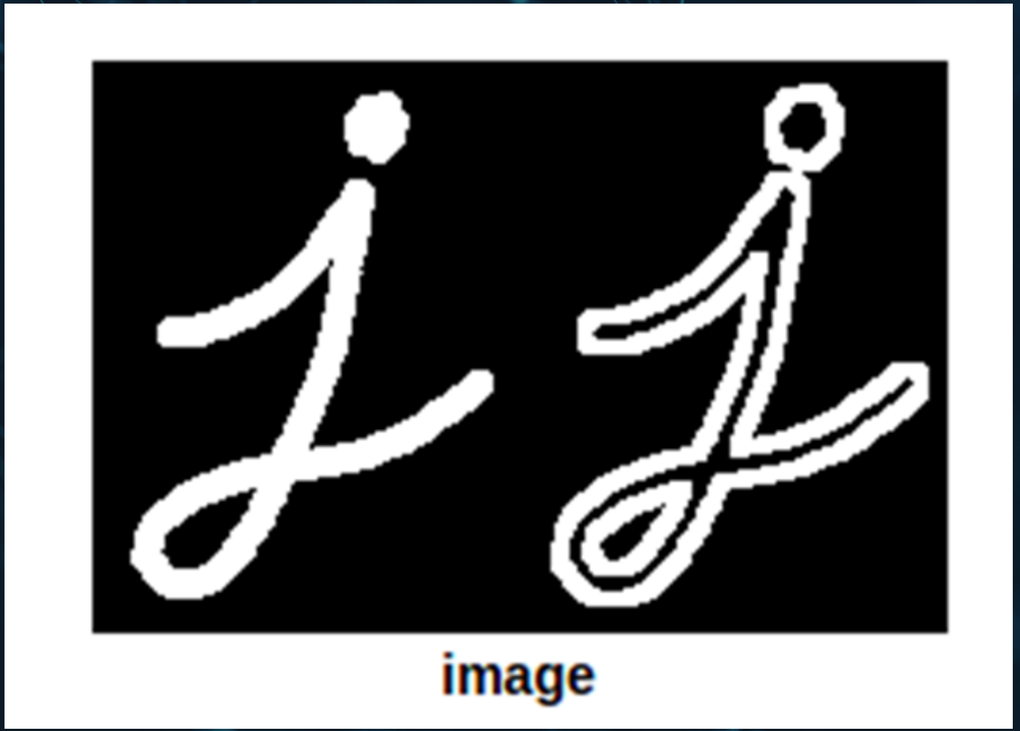
Close - Dilate and then Erode

Essentially fills up black dots

image

Morphology

Gradient - Difference in erosion and dilation



Capturing from webcam in OpenCV

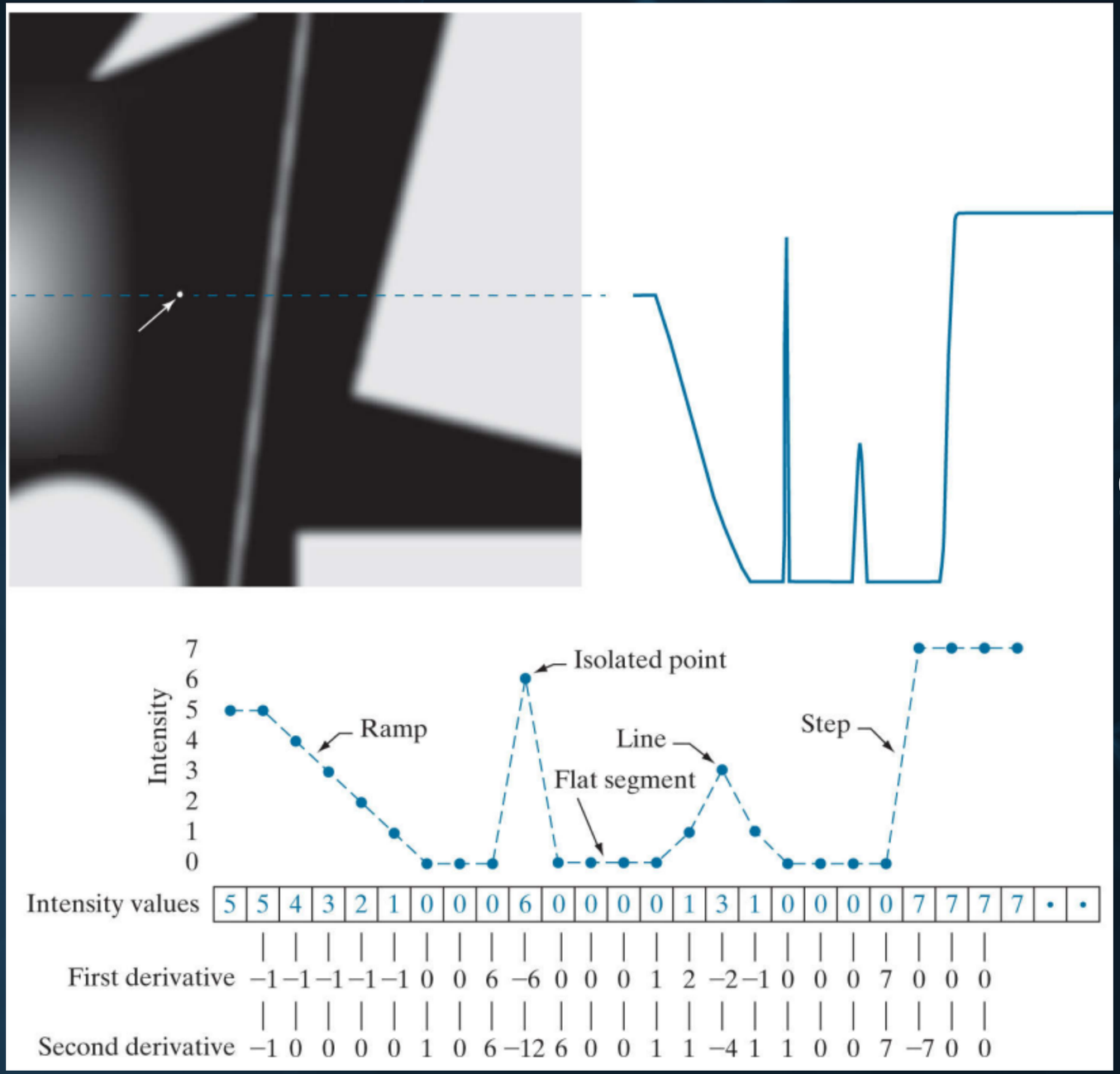
```python
import cv2

cap = cv2.VideoCapture(0)
while True:
    return_val, frame = cap.read()
    cv2.imshow("Showing video", frame)
    if(cv2.waitKey(1) & 0xFF == ord('q')):
        break
```

0 - Open Default Camera

return_val is True/False depending on success/failure of reading the video

Image Segmentation



First Order Derivative

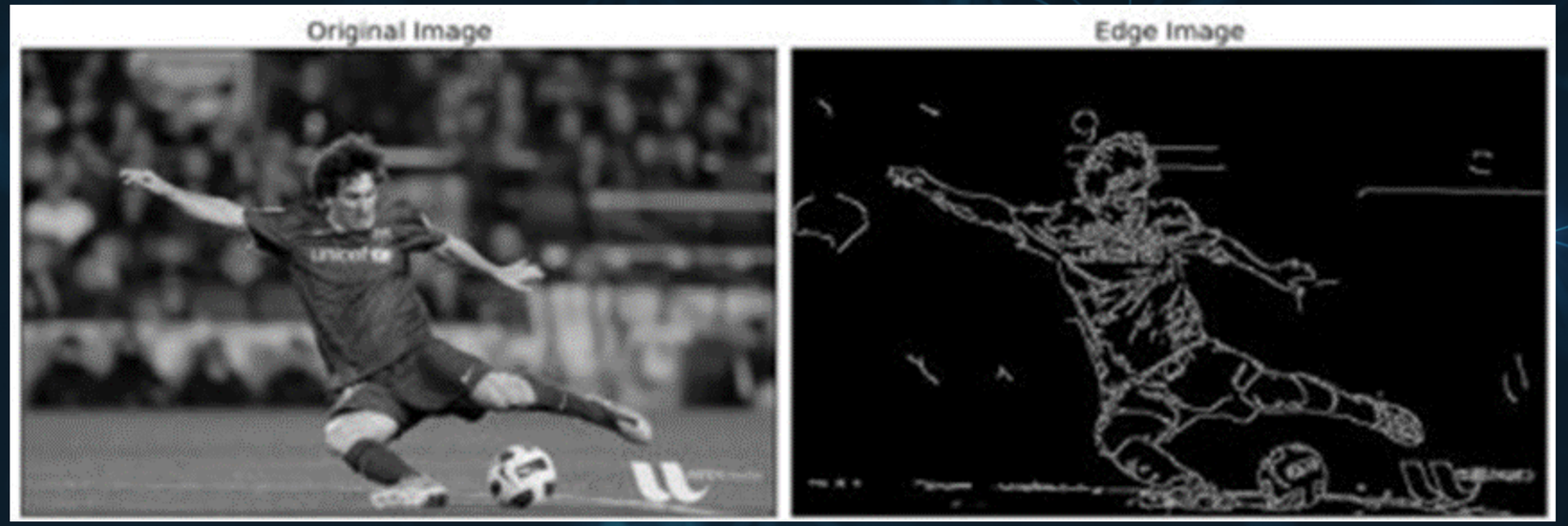
- thicker edges

Second-Order Derivative

- fine details - isolated points, line
- double edge response at ramp and step
- to determine transition from light to dark or dark to light

Edge Detection

- Canny Edge Detection
- Works on gray scale images




```
return_val, frame = cap.read()
image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(image, 100, 200)
```

Maximum length

Minimum Length

Contour Detection

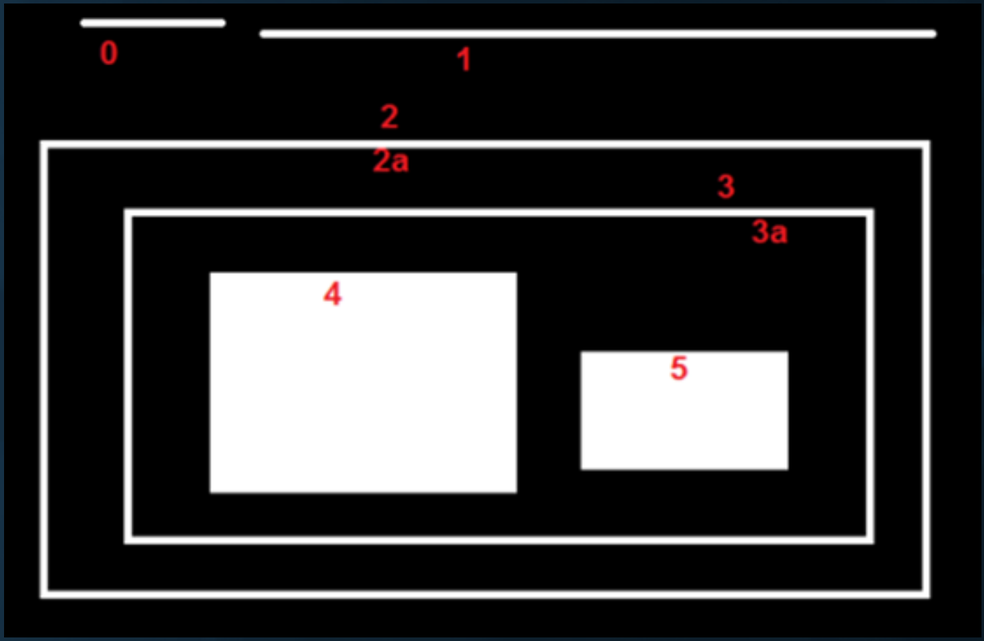
A Contour is a closed curve joining all continuous points having same color or intensity. This is a important tool in shape analysis and object detection.

Difference ?

Edge Detection and Contour Detection

Contour Detection

Better accuracy is obtained when we use binary images. So before finding contours we can apply thresholding or canny edge detection.



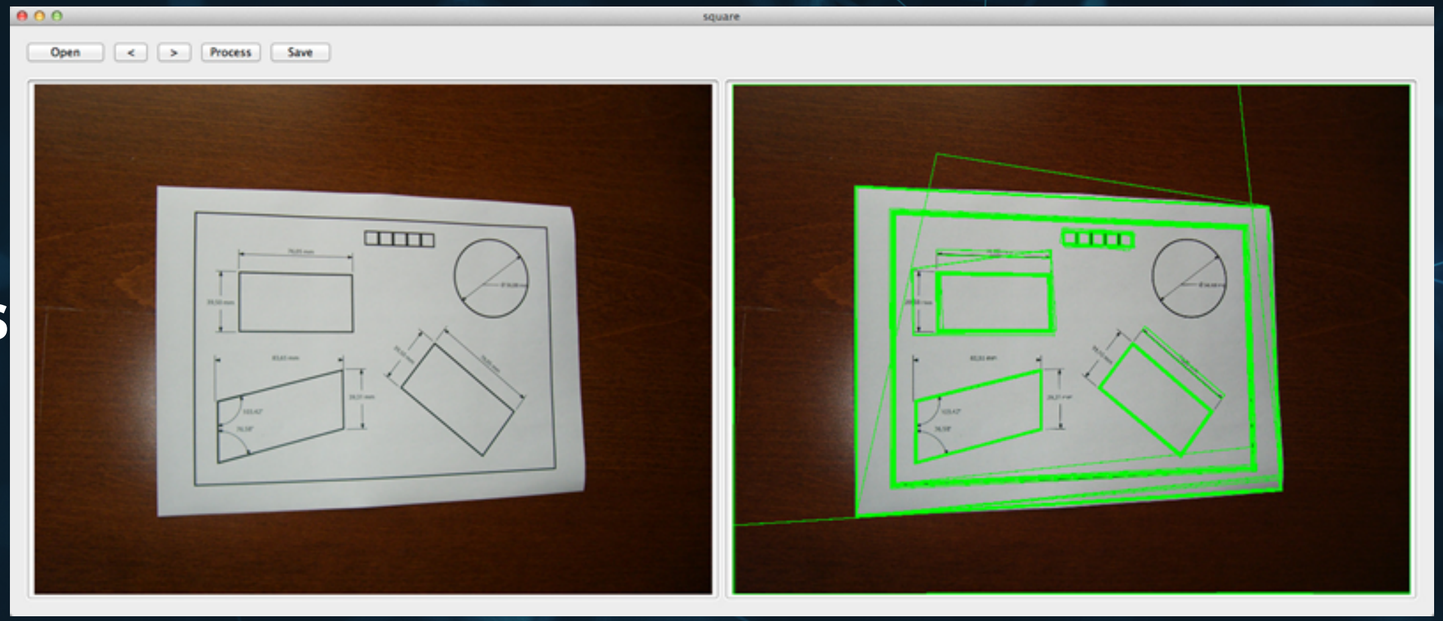
findContours(img, mode, method)
img: The image to be thresholded

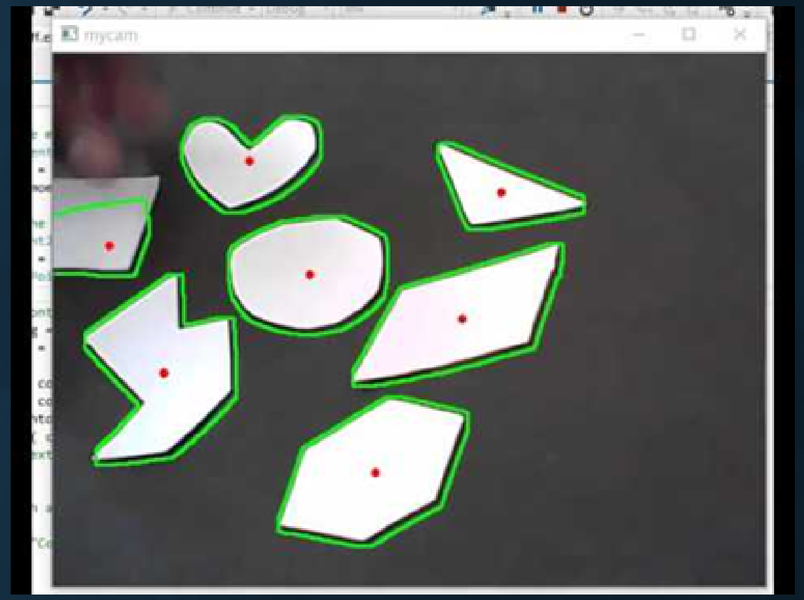
mode: retrieval method (hierarchy)

method: approximation method (corners)

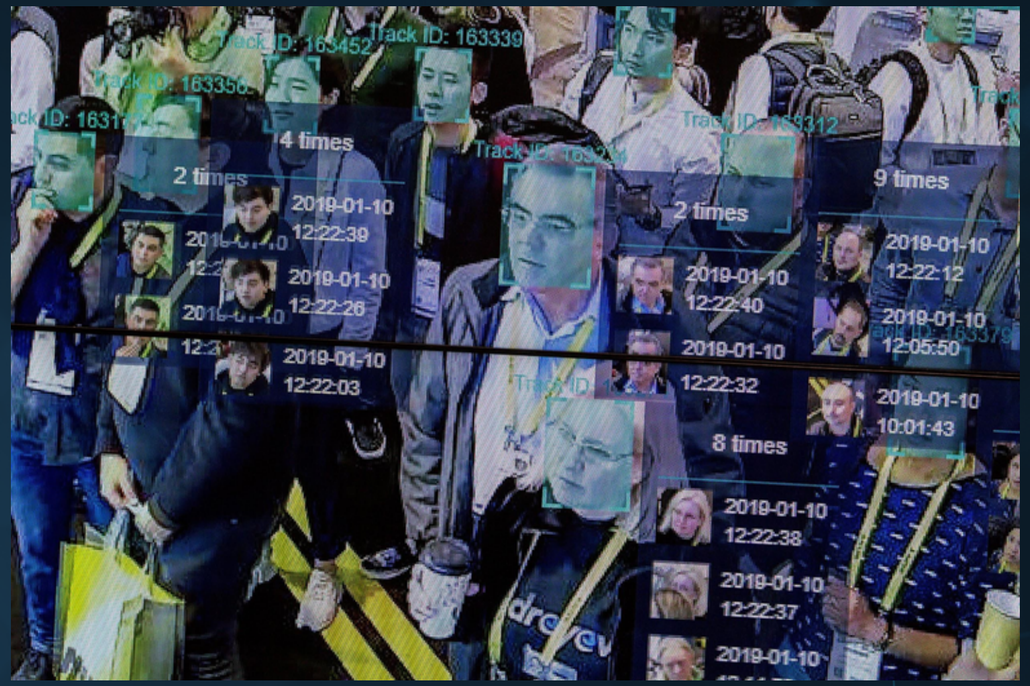
Contour Detection - Things to explore

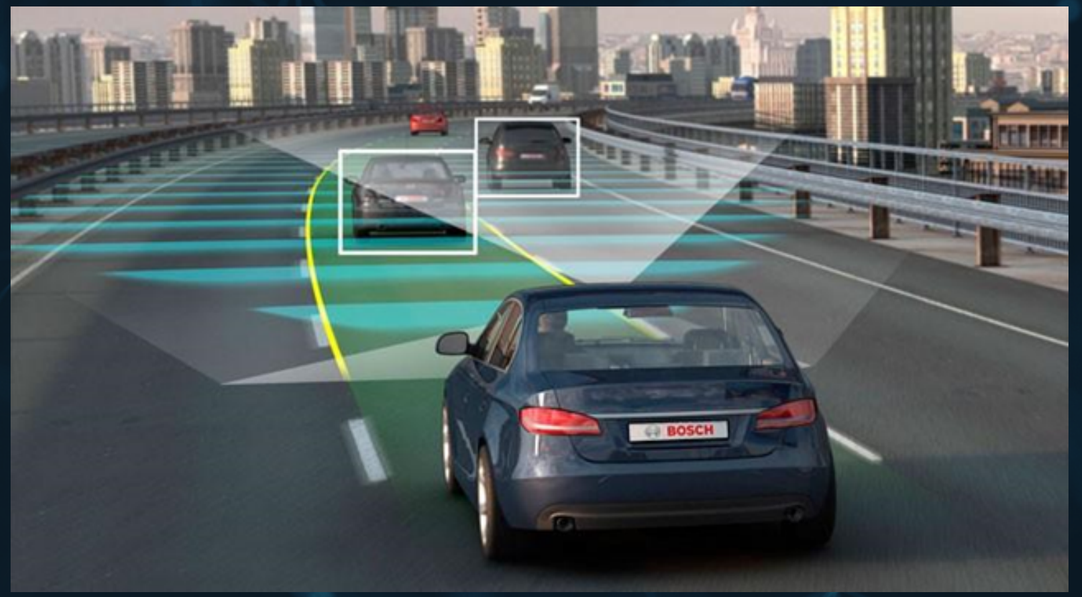
- Perimeter
- Area
- Moments (to find centre)
- Approximation using polygons





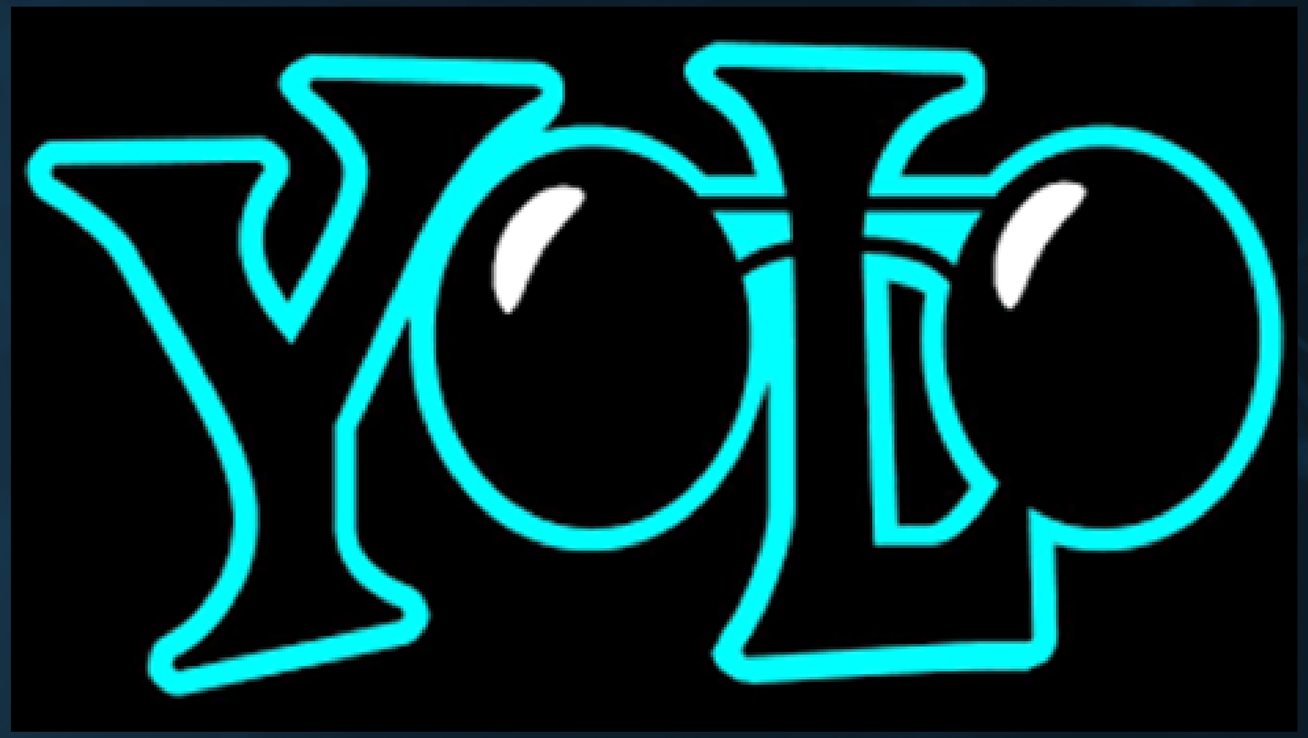
Applications of Image Processing/Computer Vision

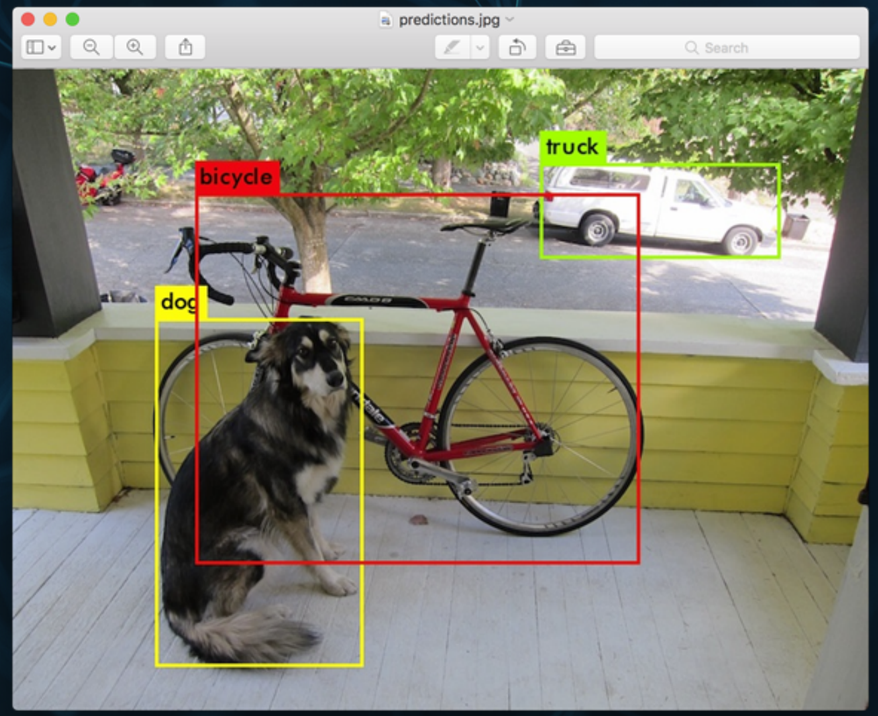




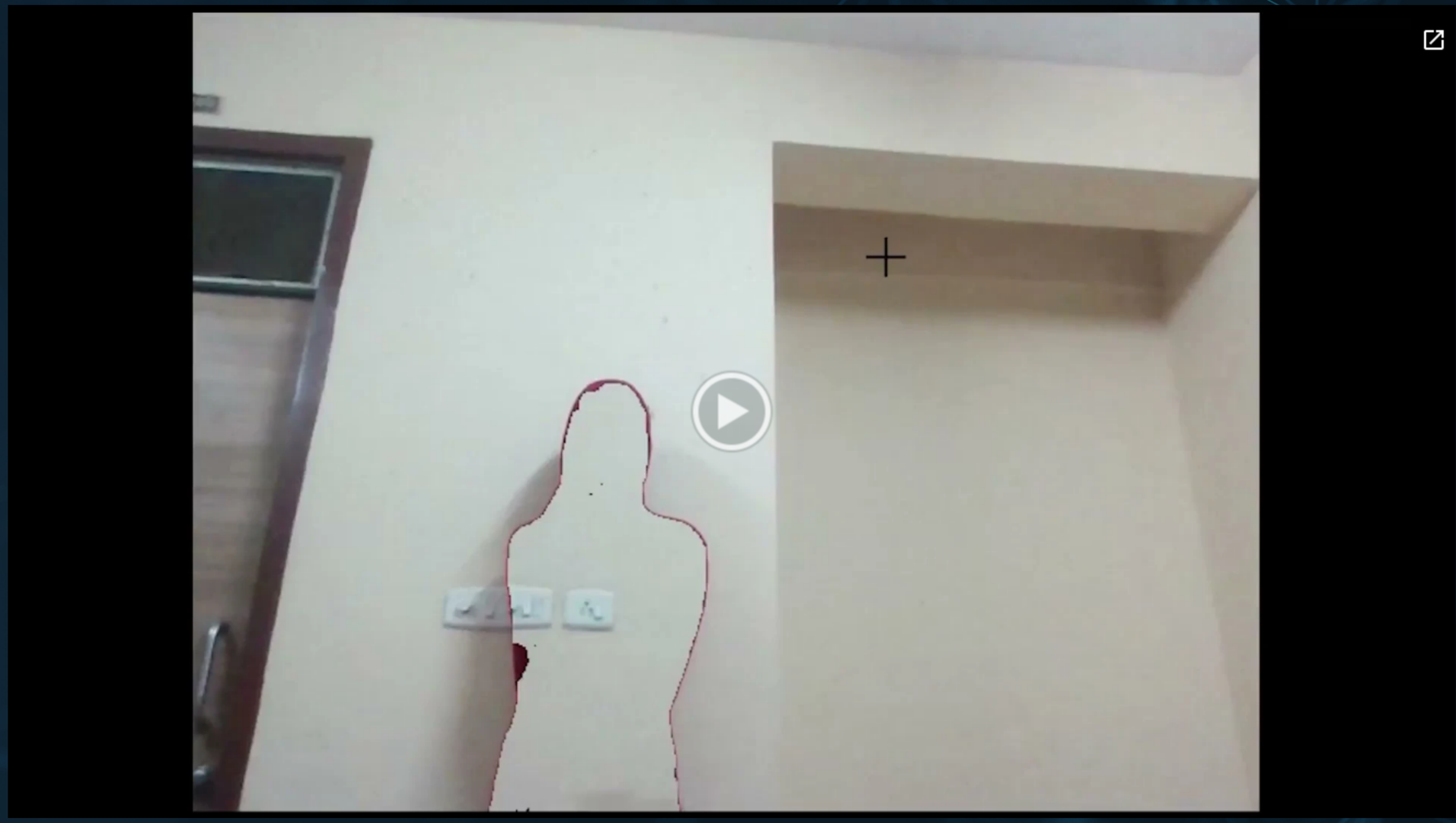
Machine Learning and other advances

https://pjreddie.com/darknet/yolo/
https://github.com/puzzledqs/BBox-Label-Tool
https://github.com/pytorch/vision/





Cool things to try out



Thank You !