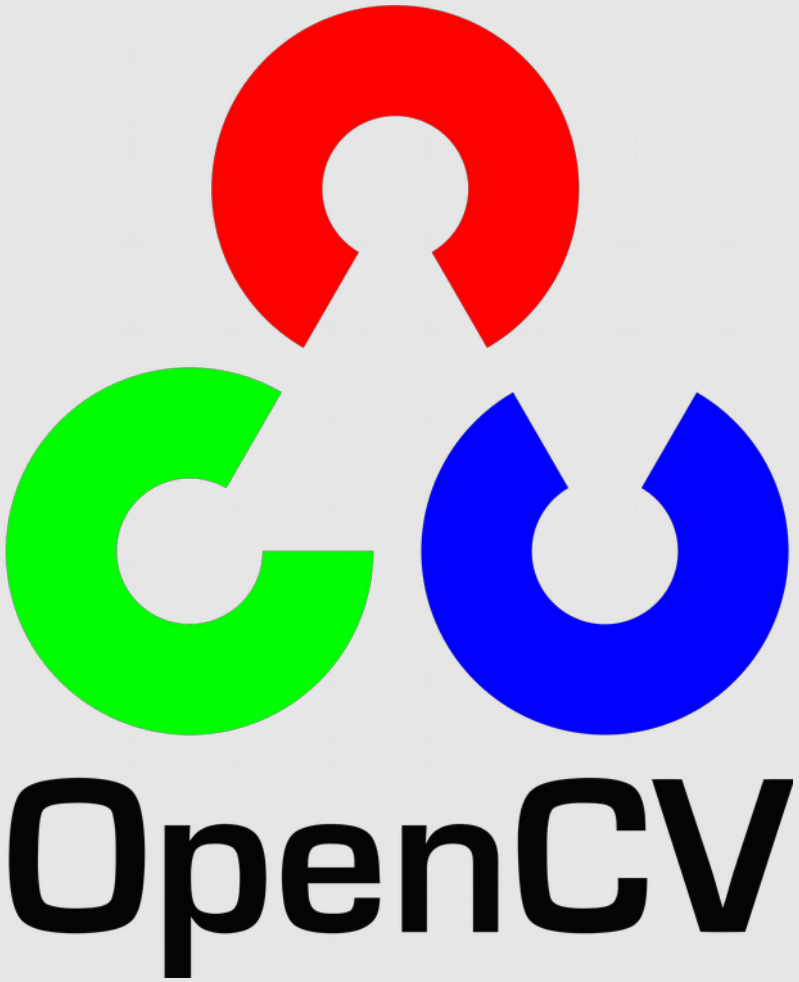
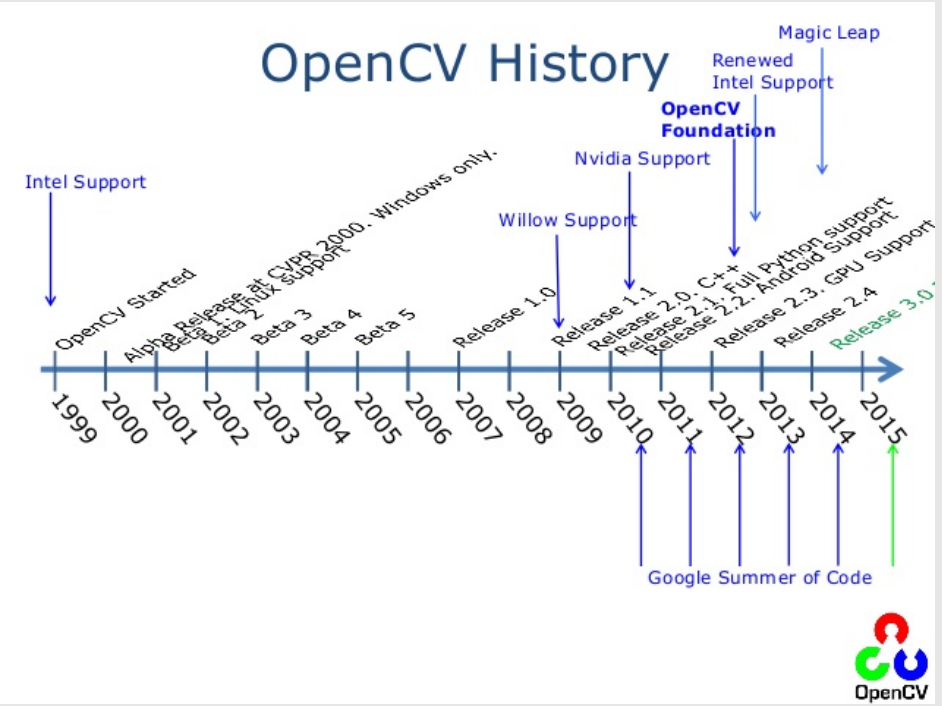
Introduction to OpenCV



OpenCV History

- OpenCV is an open library with programmation fonctions for implementing computer vision tasks:
  - Main interface in C++
  - Also interfaces in Java, Matlab and **Python:**
    https://opencv-python-tutroals.readthedocs.io

- History :
  - 1999 : Original version by Intel Research
  - 2006 : Version 1.0
  - 2008 : Support by Willow Garage
  - 2009 : Version 2.0 (**OpenCV 2**)
  - 2012 : Support by OpenCV.org
  - 2015 : Version 3.0 (OpenCV 3)
  - 2018 : Version 4.0 (OpenCV 4)



Source: http://fr.slideshare.net/embeddedvision/e04-open-cvbradsk

OpenCV-Python Installation

- Command for installation in Ubuntu/Raspbian:

  sudo apt-get install libopencv-dev python-opencv python-matplotlib

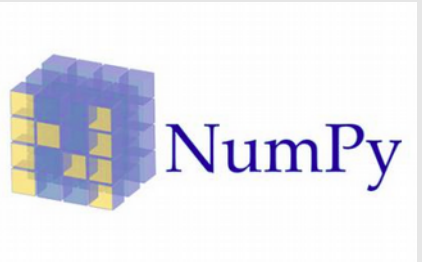
- For Windows, follow tutorial in :

  https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html

- For verifying, execute python in terminal :

  >>> import cv2

  >>> print cv2.__version__



- Why using Python ? :
  - Programming simplicity
  - Support of library **Numpy** :
    - Optimised for numerical operations
    - Similar syntax to Matlab

OpenCV Functions

HighGUI:
I/O, Interface

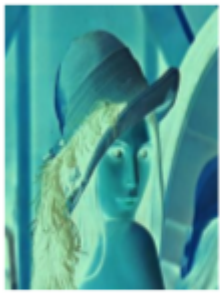
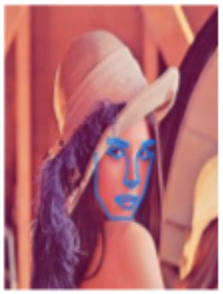


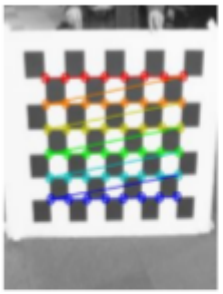
Image Processing

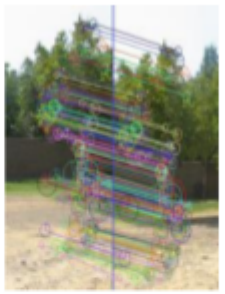

Transforms

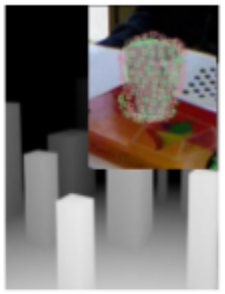

Fitting


Optical Flow
Tracking

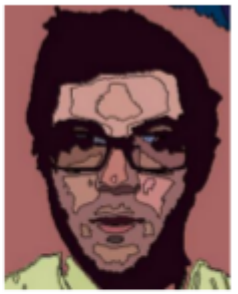

Segmentation


Calibration


Features
VSLAM


Depth, Pose
Normals, Planes,
3D Features


Object recognition
Machine learning


Computational
Photography

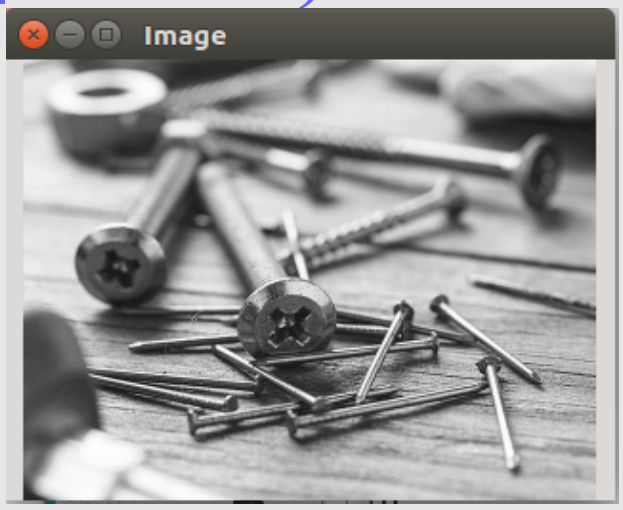
CORE:
Data structures, Matrix math, Exceptions etc

Source: http://fr.slideshare.net/embeddedvision/e04-open-cvbradsk

Read/Write of Image Files

- Commands for capturing images with the camera :
  - *With PiCam: raspistill -o cam.jpg*
  - *With Ubuntu Webcam: cheese*
- Reading/Writing an image (*file.py*) :

```
import numpy as np
import cv2

img= cv2.imread('cam.jpg', cv2.IMREAD_GRAYSCALE)
cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
print img.shape # (rows,cols,channels)
cv2.imshow('Image', img)
k = cv2.waitKey(0) & 0xFF
if k == ord('s'): # 's' key
   cv2.imwrite('camGray.png', img)
cv2.destroyAllWindows()
```

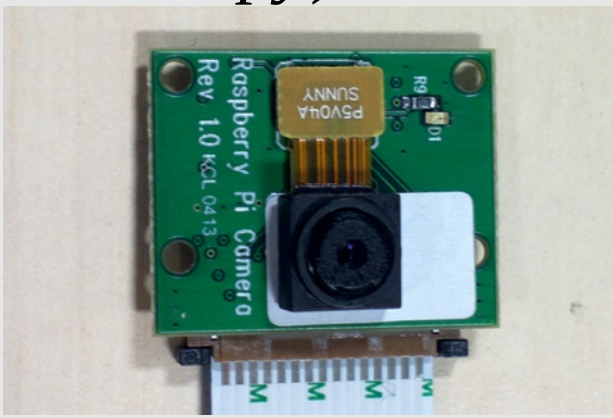


Capturing video from PiCamera

- Install PiCamera package:

sudo apt-get install python-picamera

- Recovering video frames from PiCamera (*cameraPi.py*) :




```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time, cv2

camera= PiCamera()
camera.resolution= (320,240)
rawCapture = PiRGBArray(camera, size=(320,240))
time.sleep(2) # Allow the camera to warmup

for frame in camera.capture_continuous(rawCapture, format='bgr',
    use_video_port=True):
  image= frame.array # Get raw NumPy array of the frame
  cv2.imshow('Frame',image) # Show frame
  rawCapture.truncate(0) # Clear stream for next frame
```

Capturing video from webcam

- Recovering video frames from webcam (*camera.py*):

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Modify this code for discovering the size of each frame and reduce its resolution to a half.

Use :
cap.get(propId)
cap.set(propId, value)

Modify the code for showing the frames in color

Binary Operations with Images

- How to overlap one image over another (*bitwise.py*):

```
# Load two images
img1 = cv2.imread('cam.jpg')
img2 = cv2.imread('opencv_logo.png')

# I want to put logo on top-left corner, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]

# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)

# Now black-out the area of logo in ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)

# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst
cv2.imshow('res',img1)
```

Filtering : Convolution-Smoothing

- Filtering an image with noise (‘*smoothing.py*’):

```
img = cv2.imread('bike.jpeg')
cv2.imshow('original',img)
```

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

```
# 2D convolution with 5x5 mean kernel
kernel = np.ones((5,5),np.float32)/25
filtered2D = cv2.filter2D(img,-1,kernel)
```

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

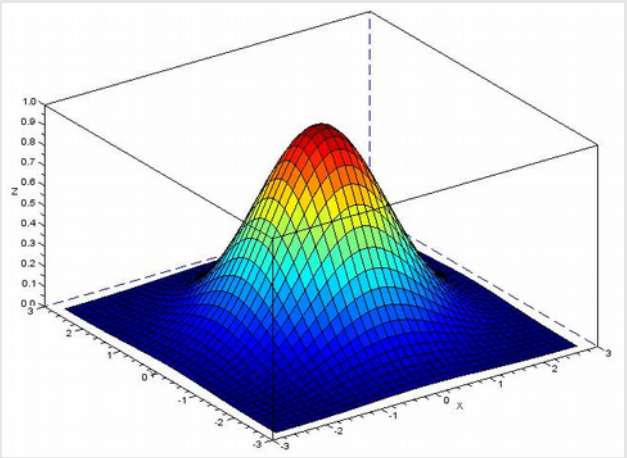
```
# Blur with 5x5 kernel (equal to 2D convolution)
blur = cv2.blur(img,(5,5))
```

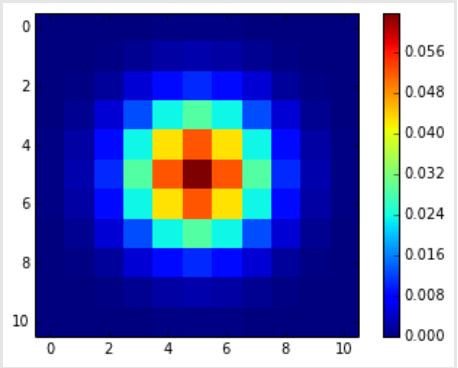
```
# Gaussian blur (Gaussian kernel)
GaussianBlur = cv2.GaussianBlur(img,(5,5),0)
```

Original

Gaussian Noise

Salt & Pepper Noise






```
# Median blur
median = cv2.medianBlur(img,5)
```

Test different sizes of kernel for filtering Gaussian noise in ‘bike.jpg’

Test différent sizes of kernel for filtering ‘salt & pepper’ noise in ‘cameraman.jpg’

Edge Detection : Sobel, Laplacian

- Detecting the edges in an image (‘*borders.py*’) :

```
ddepth = cv2.CV_64F
img = cv2.imread('sudoku.jpg')
img = cv2.GaussianBlur(img,(3,3),0)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Laplacian
laplacian = cv2.Laplacian(gray,ddepth)

# Sobel Gradient-X
grad_x = cv2.Sobel(gray,ddepth,1,0,ksize = 3)
# Sobel Gradient-Y
grad_y = cv2.Sobel(gray,ddepth,0,1,ksize = 3)

# Converting back from 64-bit floating-point (CV_64F)
# into original 8-bit unsigned integers (CV_8U)
abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)
abs_laplacian = cv2.convertScaleAbs(laplacian)
# Combining both gradients into one Sobel image
dst = cv2.addWeighted(abs_grad_x,0.5,abs_grad_y,0.5,0)
```

Laplacian = 2nd order derivatives in x and y

$$\mathtt{dst} = \Delta \mathtt{src} = \frac{\partial^2 \mathtt{src}}{\partial x^2} + \frac{\partial^2 \mathtt{src}}{\partial y^2}$$

Discrete approximation of Laplacian by the kernel :

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Sobel = Discrete differentiation of image intensity

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

Horizontal changes — Vertical changes

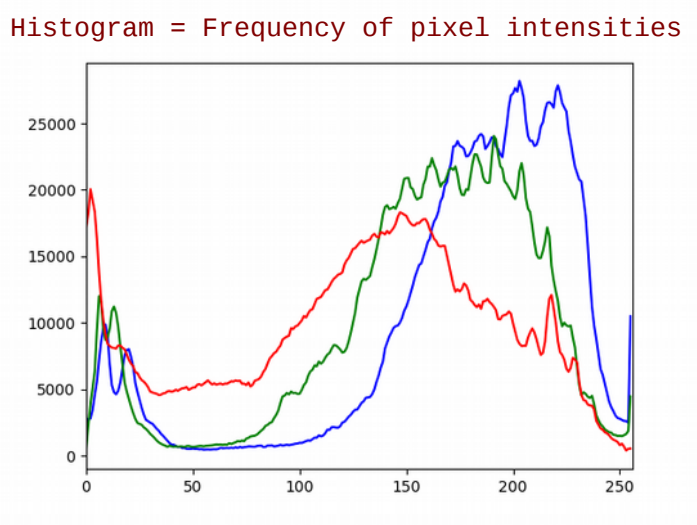
Total gradient approximation

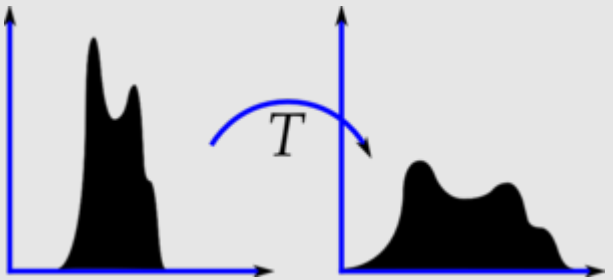
$$G = \sqrt{G_x^2 + G_y^2} \quad \text{or} \quad G = |G_x| + |G_y|$$

Contrast Adjustment : Histogram Equalization

- How to compute histograms and equalize them (‘*histogram.py*’) :

```
from matplotlib import pyplot as plt
img = cv2.imread('dark.png')
color = ('b','g','r')

for i,col in enumerate(color):
  histr = cv2.calcHist([img],[i],None,[256],[0,256])
  plt.plot(histr,color = col)
  plt.xlim([0,256])

gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
histr_gray= cv2.calcHist([gray],[0], None,[256],[0,256])
plt.plot(histr_gray,color='k') # Black plot

# histogram equalization
equ = cv2.equalizeHist(gray)
histr_equ= cv2.calcHist([equ],[0], None,[256],[0,256])
plt.plot(histr_equ,color='m') # Magenta plot
plt.show()
```

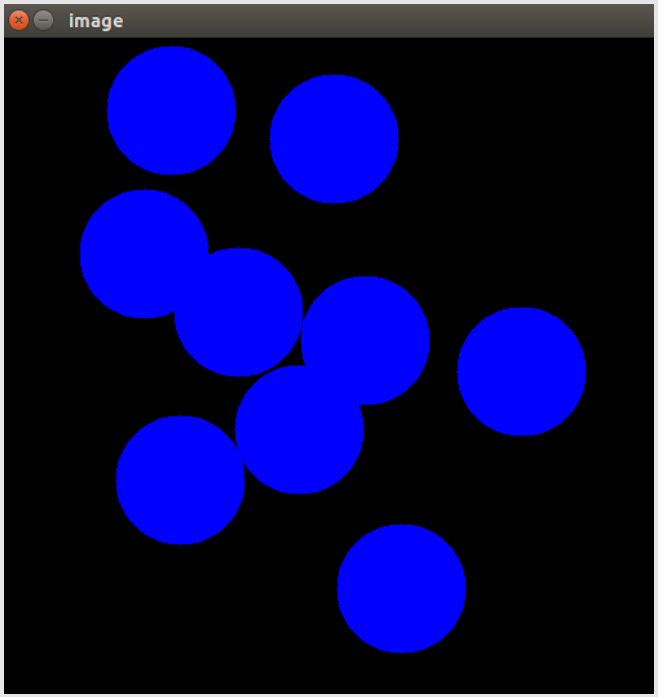




Mouse Events Handling

- Drawing circles by double-clicking on the window (‘*mouse.py*’):

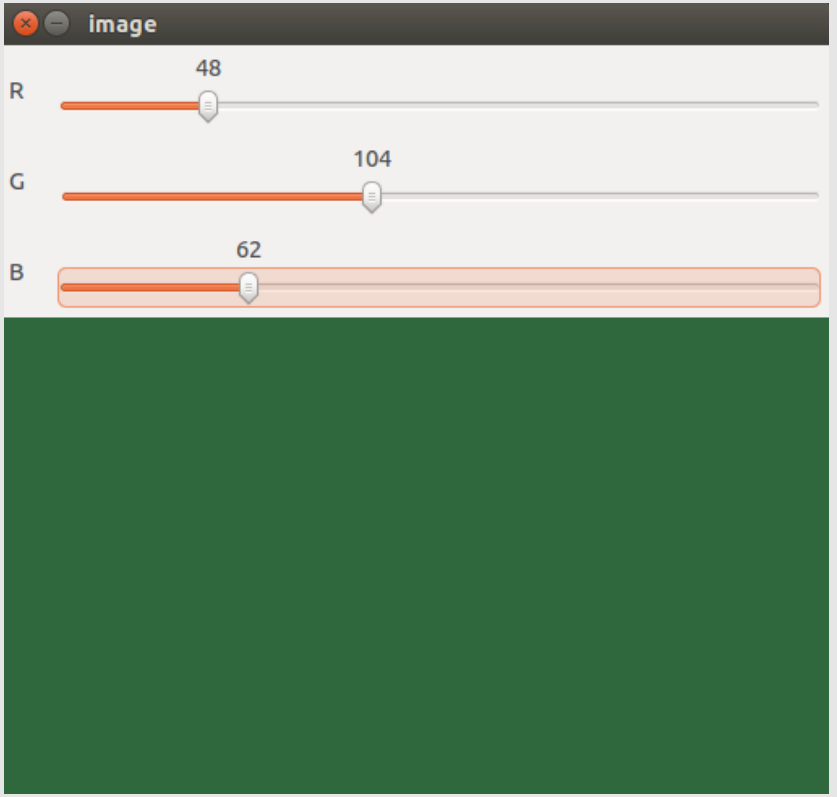
```
import cv2
import numpy as np

def draw_circle(event, x, y, flags, param): # mouse callback
    if event == cv2.EVENT_LBUTTONDBLCLK:
        cv2.circle(img, (x,y), 50, (255, 0, 0), -1)

img = np.zeros((512,512,3), np.uint8) # black image
cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)

while(True):
    cv2.imshow('image',img)
    if cv2.waitKey(1) & 0xFF == 27: # ESC key
        break
cv2.destroyAllWindows()
```



Trackbar Handling

- Choosing a color with 3 RGB trackbars (‘*trackbar.py*’):

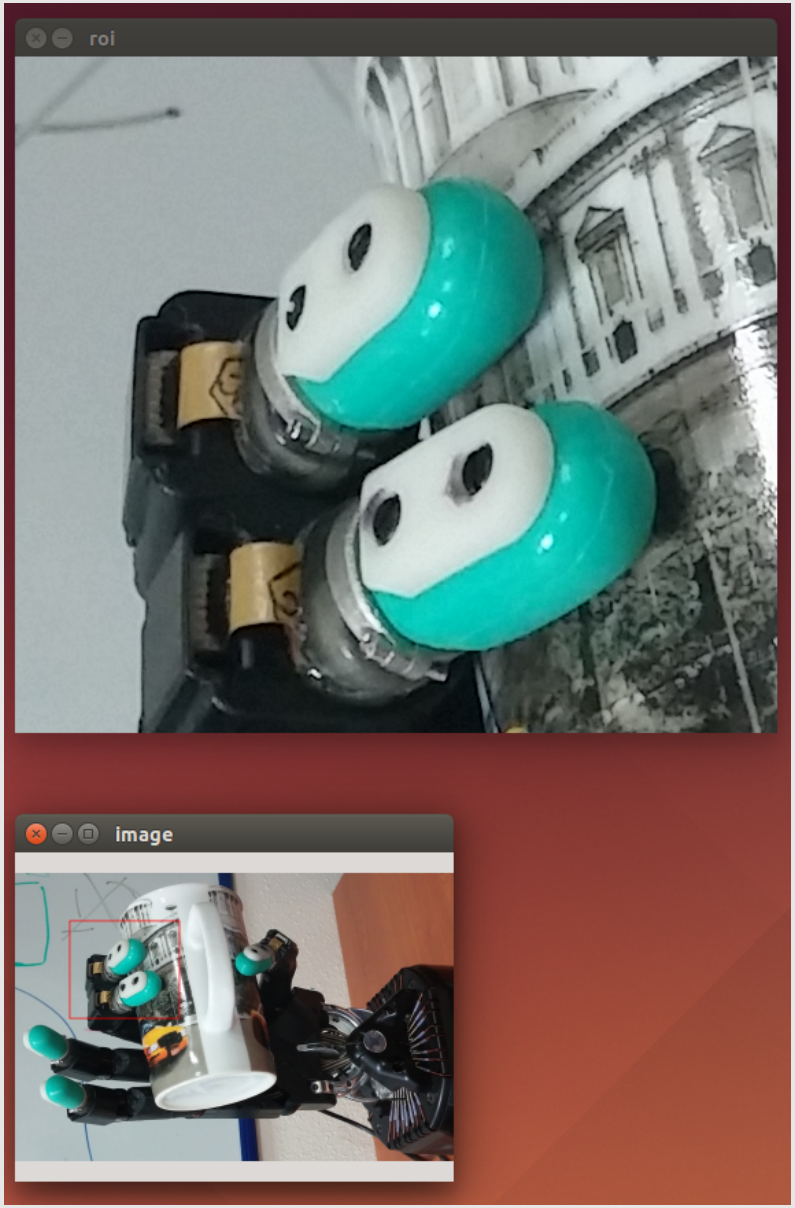
```
import cv2
import numpy as np

def nothing(x):
    pass
img = np.zeros((300,512,3), np.uint8)
cv2.namedWindow('image')
cv2.createTrackbar('R','image',0,255,nothing)
cv2.createTrackbar('G','image',0,255,nothing)
cv2.createTrackbar('B','image',0,255,nothing)

while(True):
    r = cv2.getTrackbarPos('R','image')
    g = cv2.getTrackbarPos('G','image')
    b = cv2.getTrackbarPos('B','image')
    img[:] = [b,g,r]
    cv2.imshow('image',img)
    if cv2.waitKey(1) & 0xFF == 27: # ESC key
        break
cv2.destroyAllWindows()
```



ROI selection with the mouse

- Choosing a ROI with the mouse and copy it (‘*roi.py*’):

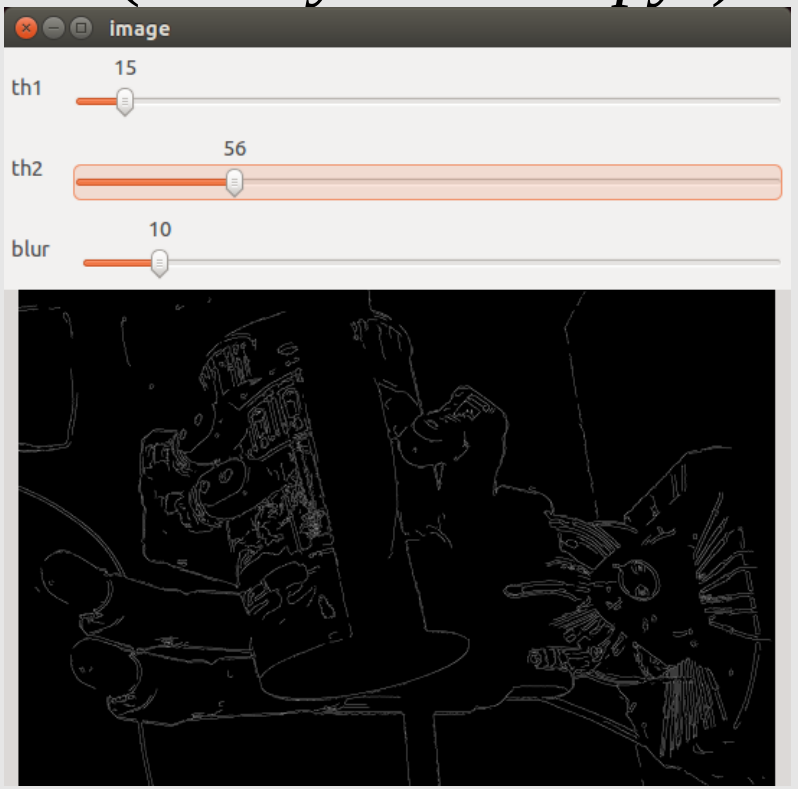
```
import cv2
img= cv2.imread('cam.jpg')
drag=False, point1=None, point2=None, selection=False

def choose_roi(event, x, y, flags, param): # mouse callback
    global drag, point1, point2, selection
    img2=img.copy()
    if event == cv2.EVENT_LBUTTONDOWN:
        point1=(x,y)
        drag=True
    if event == cv2.EVENT_MOUSEMOVE and drag:
        point2=(x,y)
        cv2.rectangle(img2,point1,point2,(0,0,255),4)
    if event == cv2.EVENT_LBUTTONUP and drag:
        point2=(x,y)
        cv2.rectangle(img2,point1,point2,(0,0,255),4)
        drag=False
        selection=True
    cv2.imshow('image',img2)

cv2.setMouseCallback('image', choose_roi)
while(True):
    if selection:
        cv2.imshow('roi',img[point1(0):point2(0),point1(1):point2(1)])
        selection= False
```



Canny Edge Detection

- Fixing thresholds for Canny edge detection (‘*cannyDetector.py*’) :

```
import cv2
import numpy as np

img= cv2.imread('cam.jpg')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) # Change to gray
def nothing(x):
    pass

cv2.namedWindow('image',cv2.WINDOW_NORMAL)
cv2.namedWindow('blurw',cv2.WINDOW_NORMAL)
cv2.createTrackbar('th1','image',0,255,nothing)
cv2.createTrackbar('th2','image',0,255,nothing)
cv2.createTrackbar('blur','image',1,10,nothing)

while(True):
    th1=cv2.getTrackbarPos('th1','image')
    th2=cv2.getTrackbarPos('th2','image')
    size=cv2.getTrackbar('blur','image')
    kernel= np.ones((size,size),np.float32)/(size*size)
    filteredGray= cv2.filter2D(gray, -1, kernel) # Convolution with kernel
    edges=cv2.Canny(filteredGray, th1, th2) # Edge detection with Canny
    cv2.imshow('image',edges)
    cv2.imshow('blurw',filteredGray)
    if cv2.waitKey(1) & 0xFF==27: # ESC key
        break
cv2.destroyAllWindows()
```

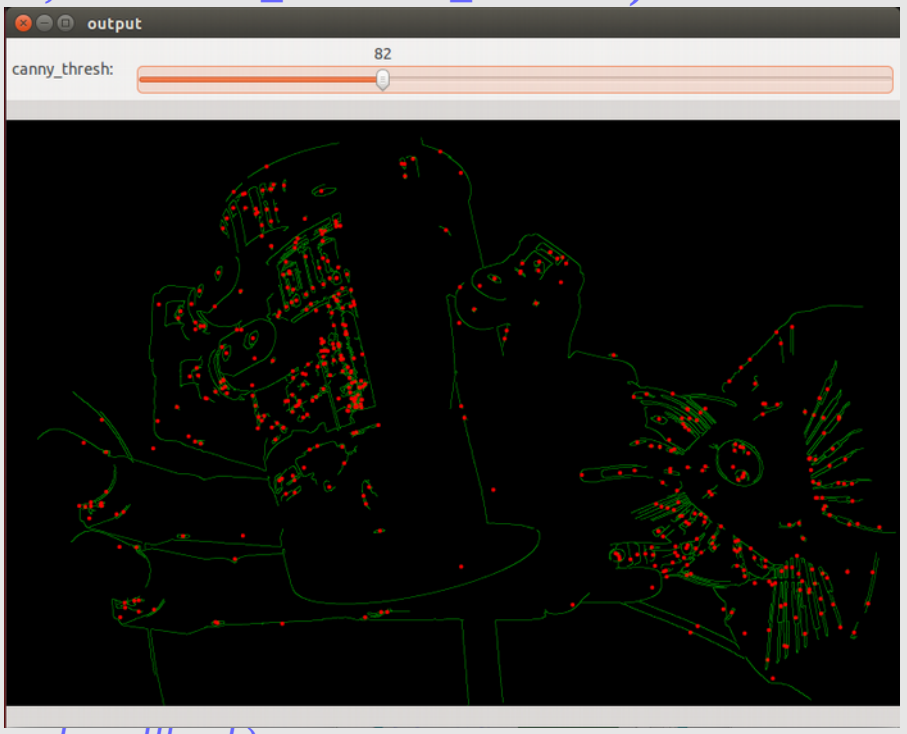


Contour segmentation + moments

- Segmenting contours et getting their moments (‘*moments.py*’):

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

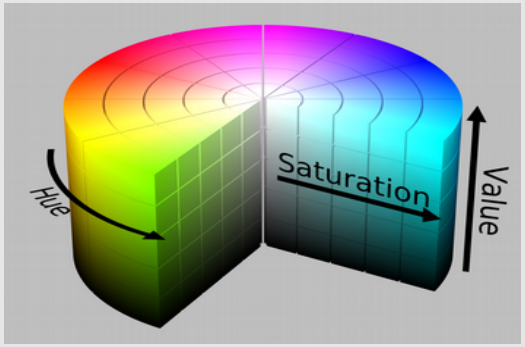
```
import cv2
import numpy as np

def thresh_callback(thresh):
    edges = cv2.Canny(blur,thresh,thresh*3)
    drawing = np.zeros(img.shape,np.uint8)
    contours,hierarchy = cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours: # For each contour
        moments = cv2.moments(cnt)
        if moments['m00']>0: # area of contour
            cx = int(moments['m10']/moments['m00']) #Centroid x
            cy = int(moments['m01']/moments['m00']) #Centroid y
            cv2.drawContours(drawing,[cnt],0,(0,255,0),1)
            cv2.circle(drawing,(cx,cy),5,(0,0,255),-1)
    cv2.imshow('output',drawing)

img = cv2.imread('cam.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray,(5,5),0)
cv2.namedWindow('output',cv2.WINDOW_NORMAL)
thresh = 100
max_thresh = 255
cv2.createTrackbar('canny_thresh:','output',thresh,max_thresh,thresh_callback)
thresh_callback(200)
if cv2.waitKey(0) == 27: #Wait until ESC
    cv2.destroyAllWindows()
```



Exercise :
Tracking an object with color

Exercise (Use file ‘camera.py’ or ‘cameraPi.py’ as base) :

- Goal : Segment an object according to its color
- Inputs : Video frames from camera + desired color
- Output : Centroid of the biggest object



- Steps :
  1. Transform image from BGR to HSV
  2. Binarize image for detecting color : cv2.inRange
  3. Find contours of binarized image (cv2.findContours)
  4. Choose biggest contour (cv2.contourArea), compute centroid (moments.py) and draw it.
- Bonus :
  - Choose binarization thresholds with trackbars
  - Choose color (H) with the mouse by clicking on it
  - Choose thresholds automatically from a ROI
  - Draw a bounding box around the object