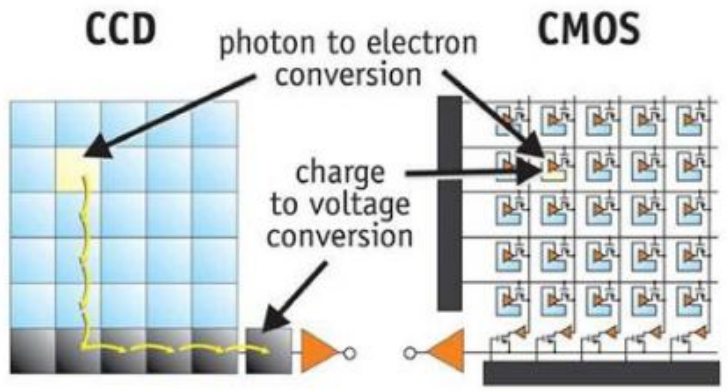
Coding lesson: traditional computer vision

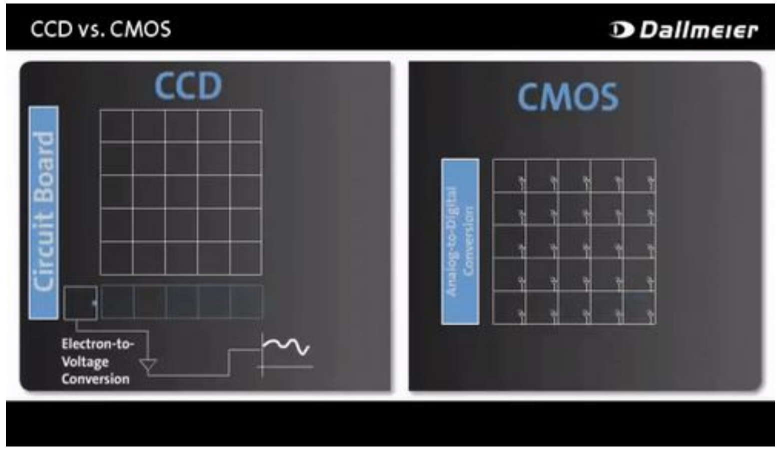
Dipartimento di Ingegneria Chimica, dei Materiali e della Produzione Industriale
Università degli Studi di Napoli Federico II

Speaker: Eng. Giulio Mattera

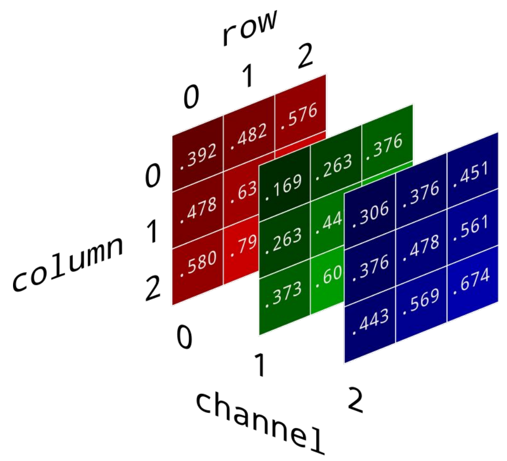
mail:giulio.mattera@unina.it

Recap : Sensors for Computer vision



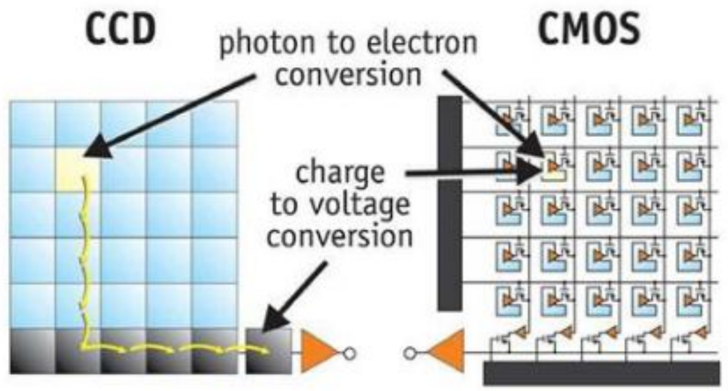


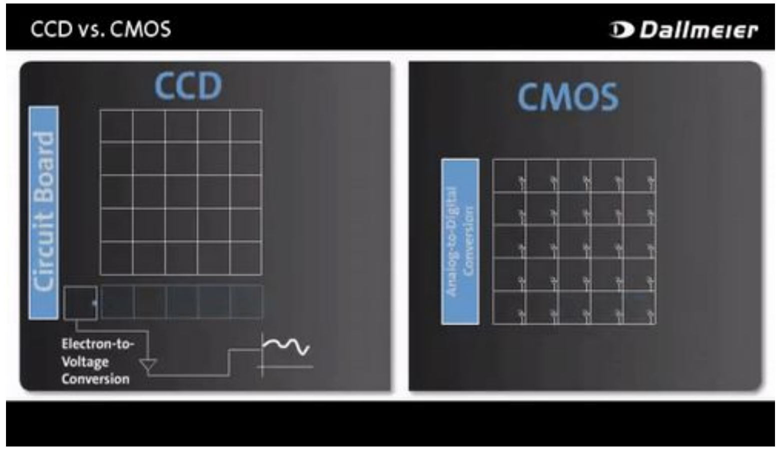
Digital Conversion



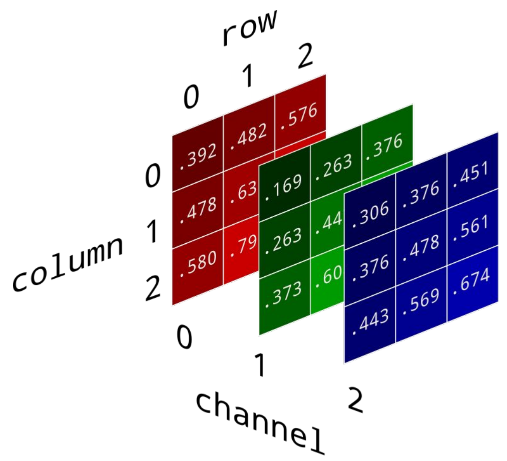
In range of 0-255 for each channel if uint(8) representation of the data.

Recap : Sensors for Computer vision



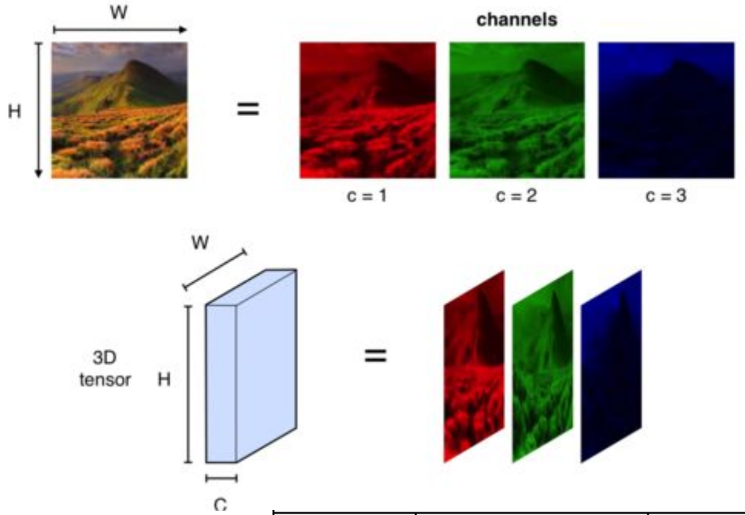


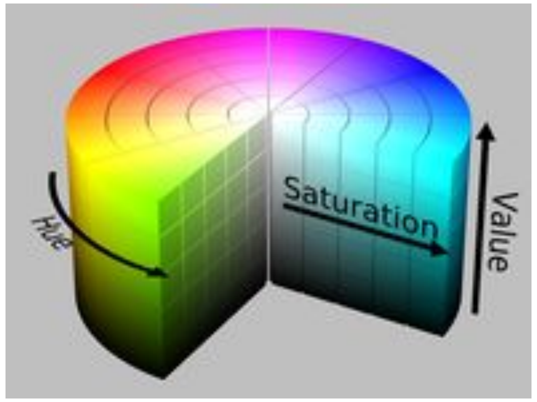
Digital Conversion

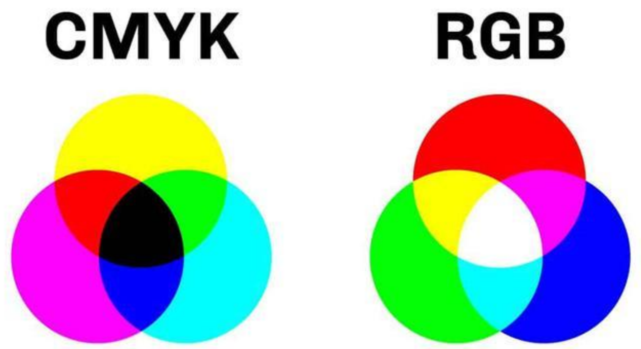


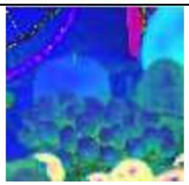
In range of 0-255 for each channel if uint(8) representation of the data.

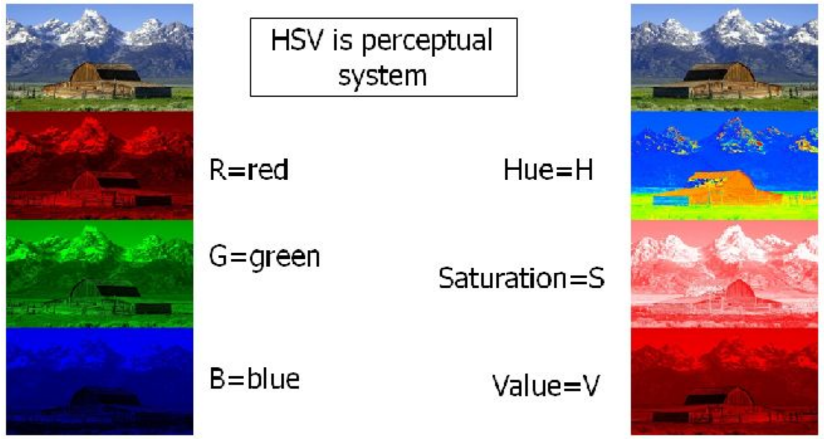
Recap: Color space







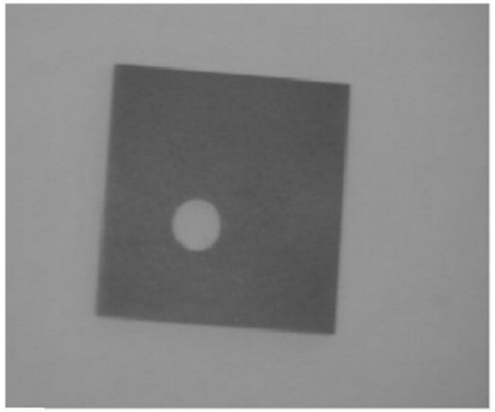
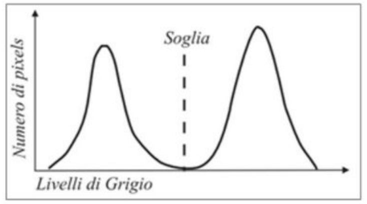
Input image	Original image	RGB → HSV
girl		
fruits		

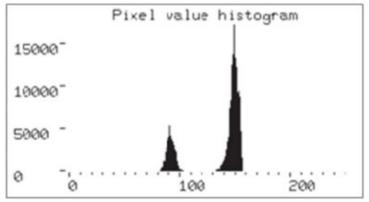


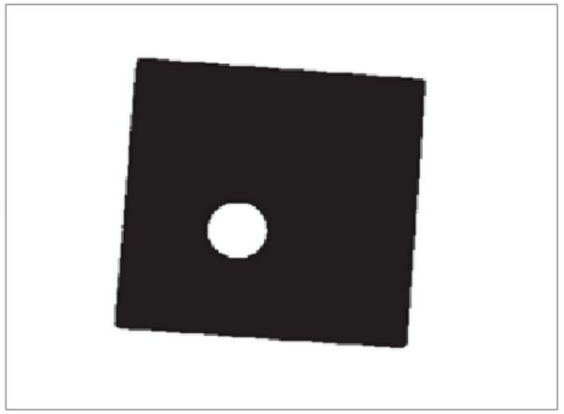
Recap: Histogram, grayscale and binarization

There are some cameras that does not have the capacity to capture color (**monocromatic).** The digital result is a matrix (2D tensor) that contain only information about **gray level.**

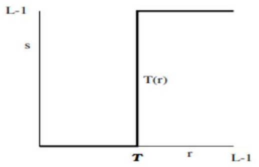
- From any image we can get the greyscale one with different methods.
- From a grayscale image we can get the binary one (b/w) observing the histogram of pixel's intensity.
- Founded a threshold in this way we can apply a low pass filter on image:



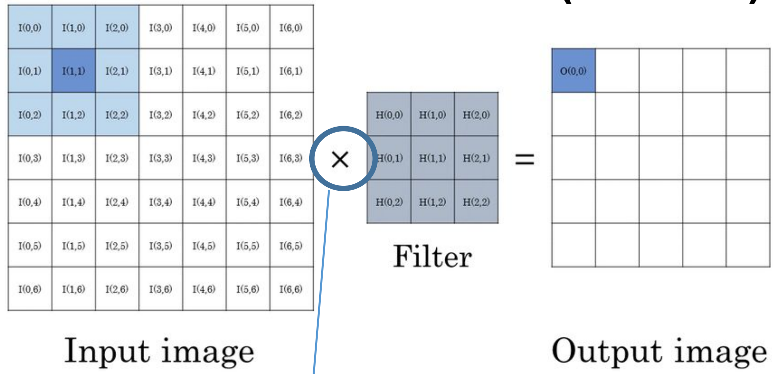


$$s = \{0\ L - 1\ \frac{if\ r \le T}{if\ r > T}$$

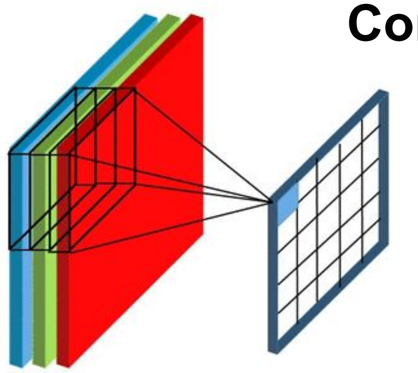


Recap: Convolution operation and filters (I)

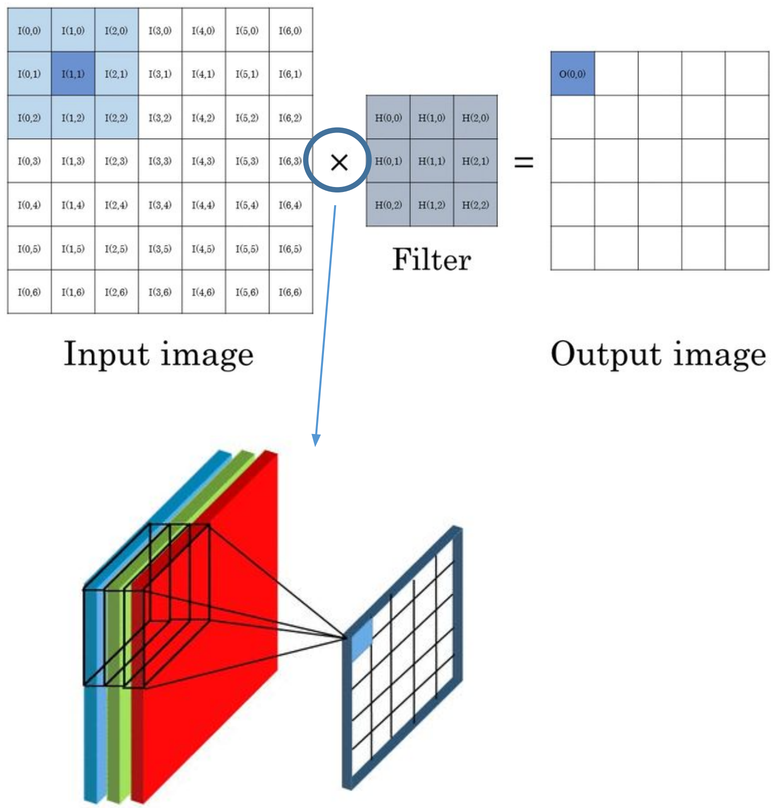
Kernel (or filter)


I(0,0) I(1,0) I(2,0) I(3,0) I(4,0) I(5,0) I(6,0)
I(0,1) I(1,1) I(2,1) I(3,1) I(4,1) I(5,1) I(6,1)
I(0,2) I(1,2) I(2,2) I(3,2) I(4,2) I(5,2) I(6,2)
I(0,3) I(1,3) I(2,3) I(3,3) I(4,3) I(5,3) I(6,3)
I(0,4) I(1,4) I(2,4) I(3,4) I(4,4) I(5,4) I(6,4)
I(0,5) I(1,5) I(2,5) I(3,5) I(4,5) I(5,5) I(6,5)
I(0,6) I(1,6) I(2,6) I(3,6) I(4,6) I(5,6) I(6,6)
Input image
×
H(0,0) H(1,0) H(2,0)
H(0,1) H(1,1) H(2,1)
H(0,2) H(1,2) H(2,2)
Filter
=
O(0,0)
Output image

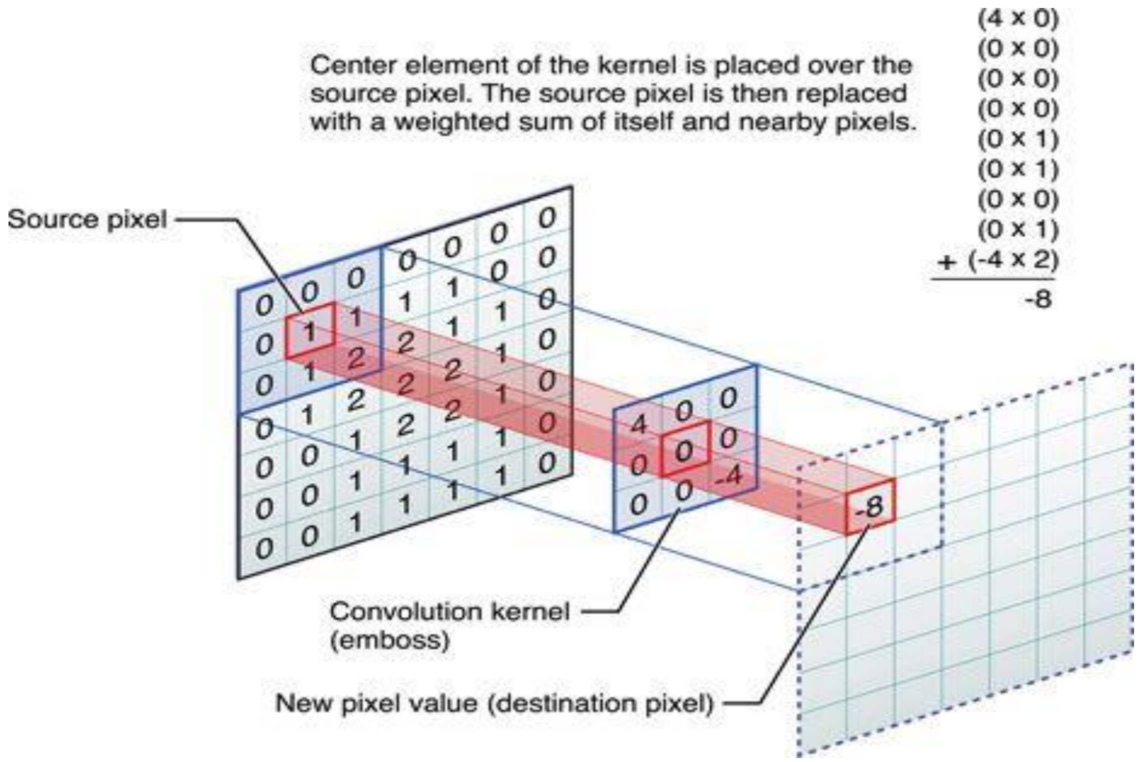
Convolution operation



Recap: Convolution operation and filters (II)

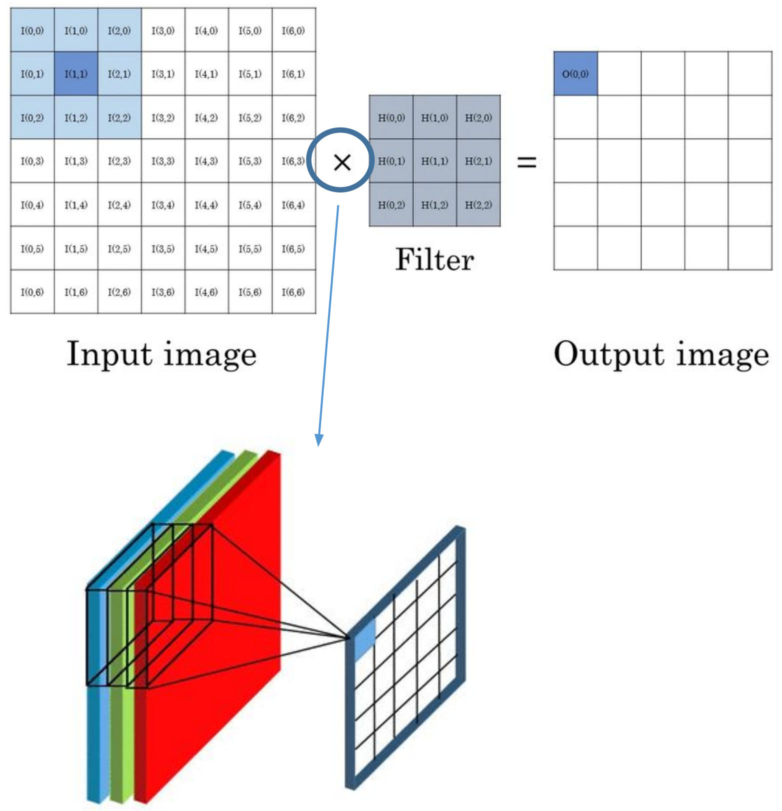


$$\boldsymbol{O}[u,v] = \sum_{i=-h}^{h}\sum_{j=-h}^{h} \boldsymbol{I}[u-i, v-j]\boldsymbol{K}[i, j] \ , \qquad \forall (u,v) \in \boldsymbol{I}$$



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Recap: Convolution operation and filters (III)

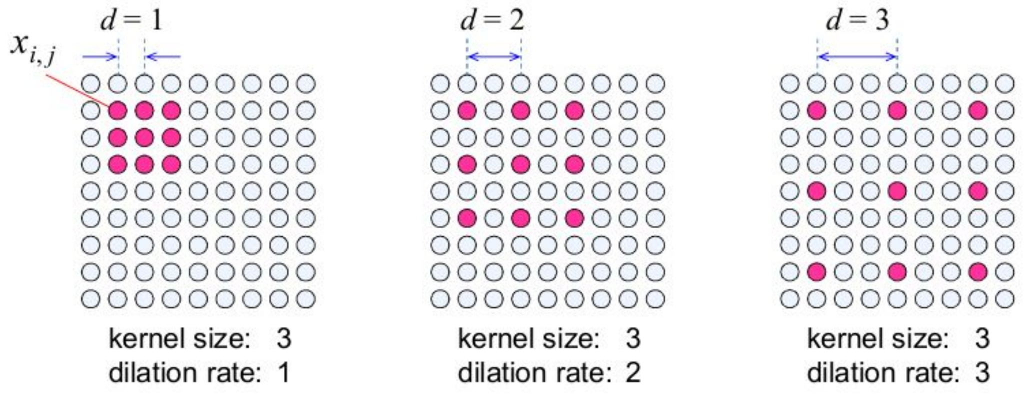

I(0,0) I(1,0) I(2,0) I(3,0) I(4,0) I(5,0) I(6,0)
I(0,1) I(1,1) I(2,1) I(3,1) I(4,1) I(5,1) I(6,1)
I(0,2) I(1,2) I(2,2) I(3,2) I(4,2) I(5,2) I(6,2)
I(0,3) I(1,3) I(2,3) I(3,3) I(4,3) I(5,3) I(6,3)
I(0,4) I(1,4) I(2,4) I(3,4) I(4,4) I(5,4) I(6,4)
I(0,5) I(1,5) I(2,5) I(3,5) I(4,5) I(5,5) I(6,5)
I(0,6) I(1,6) I(2,6) I(3,6) I(4,6) I(5,6) I(6,6)
H(0,0) H(1,0) H(2,0)
H(0,1) H(1,1) H(2,1)
H(0,2) H(1,2) H(2,2)
O(0,0)
×
=
Input image
Filter
Output image

The output image has a new size:

$$o = d - f + 1$$

But I can decide to use a dilatation rate or slide different from 1:

$$o = \frac{d - f}{s} - 1$$



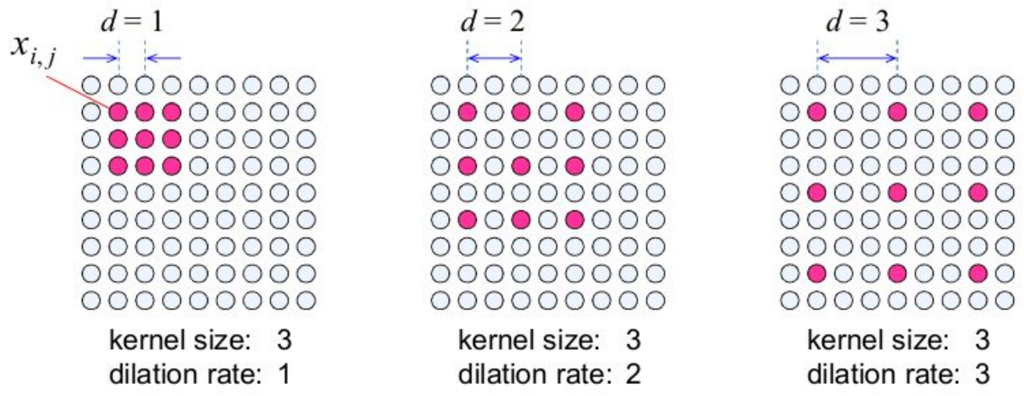
Recap: Convolution operation and filters (VI)

The output image has a new size:

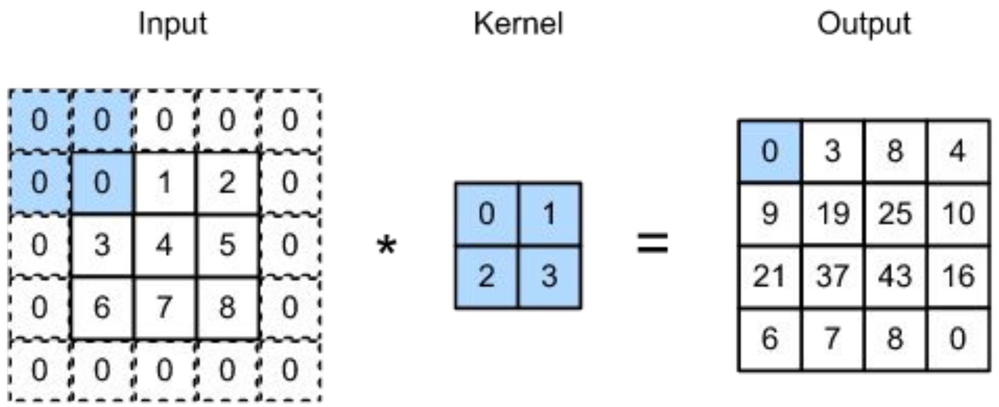
$$o = d - f + 1$$

But I can decide to use a dilatation rate or slide different from 1:

$$o = \frac{d - f}{s} + 1$$



In this way pixels on the border are less affected by the convolution, hence features on the border could not be detected. To solve this problme we can introduce padding: **adding an extra border** filled by zeros.



$$o = \frac{d - f + 2p}{s} + 1$$

Recap: Convolution operation and filters (V)

In this way pixels on the border are less affected by the convolution, hence features on the border could not be detected. To solve this problme we can introduce padding: **adding an extra border** filled by zeros.

$$o = \frac{d - f + 2p}{s} + 1$$

We can distinguish 2 different kind of padding:

- Same, to have the same size for output image $p = \frac{s * (o-1) - n + f}{2}$
- Valid – no padding

OpenCV library

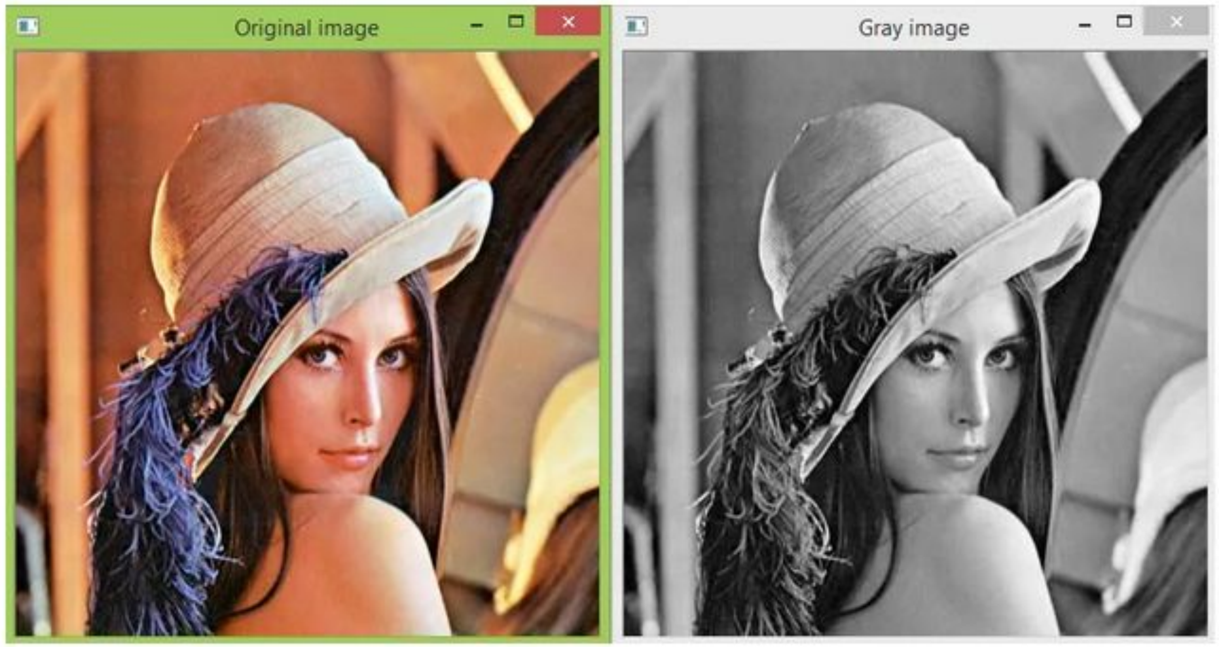
https://github.com/opencv/opencv/wiki https://opencv.org/opencv-free-course/

- OpenCV (**Open Source Computer Vision Library**) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products

- The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras…

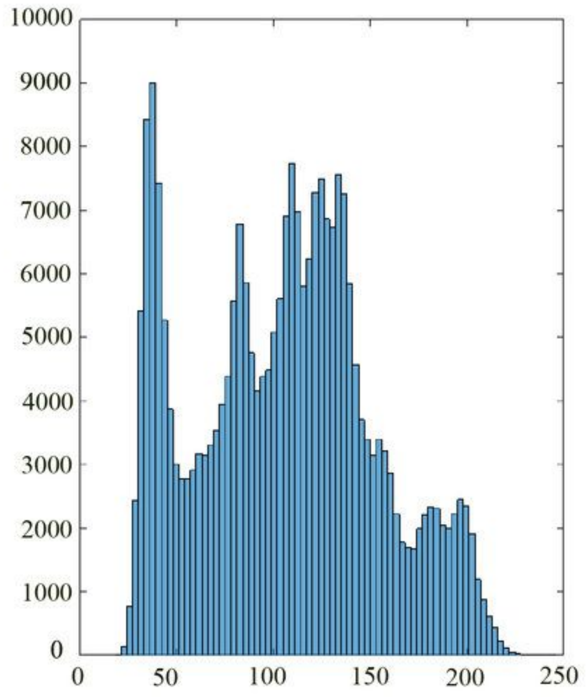
https://opencv.org/ https://opencv.org/platforms/cuda/

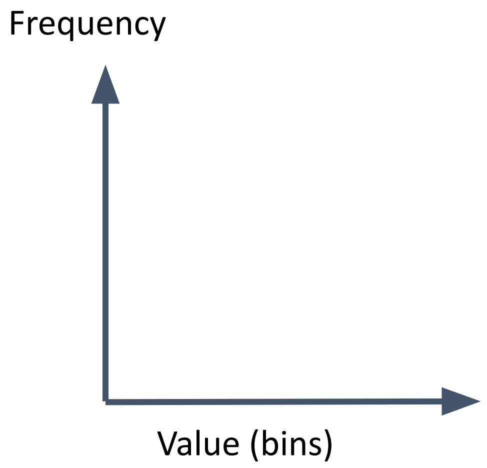
Coding: Gray scale image



```
import cv2

image = cv2.imread('C:/Users/N/Desktop/Test.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Original image',image)
cv2.imshow('Gray image', gray)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

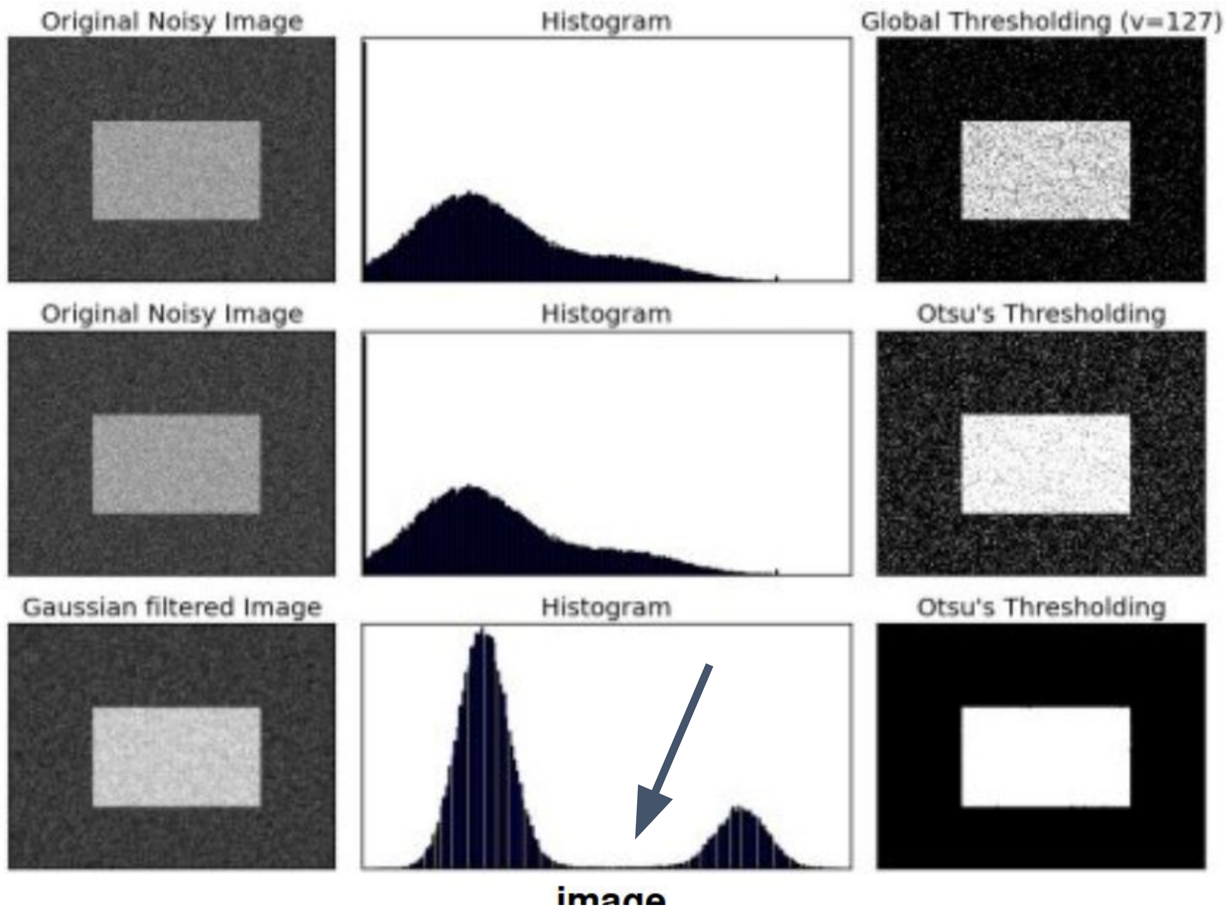
Coding: Histogram

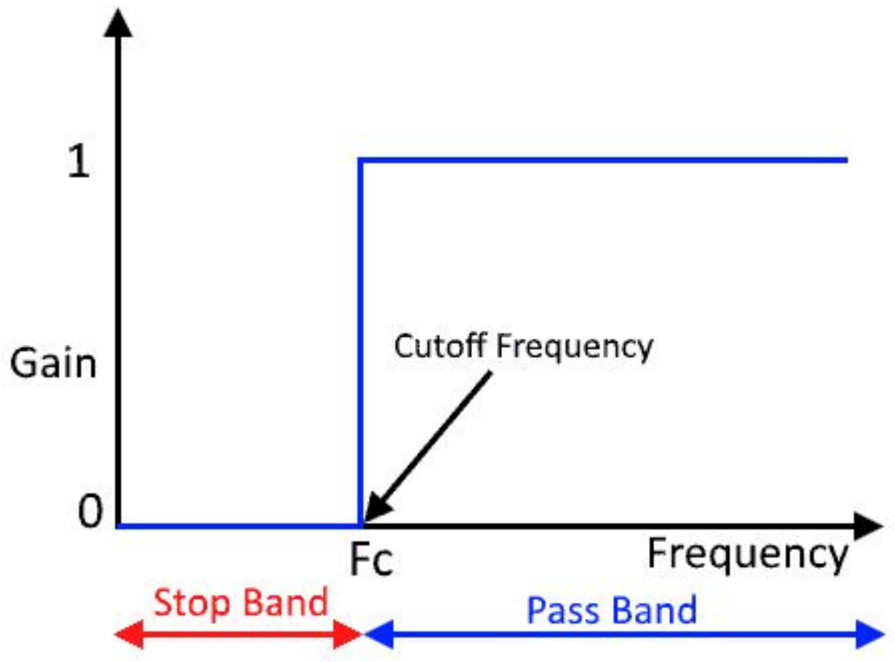




https://www.khanacademy.org/math/statistics-probability/displaying-describing-data/quantitative-data-graphs/a/histograms-review

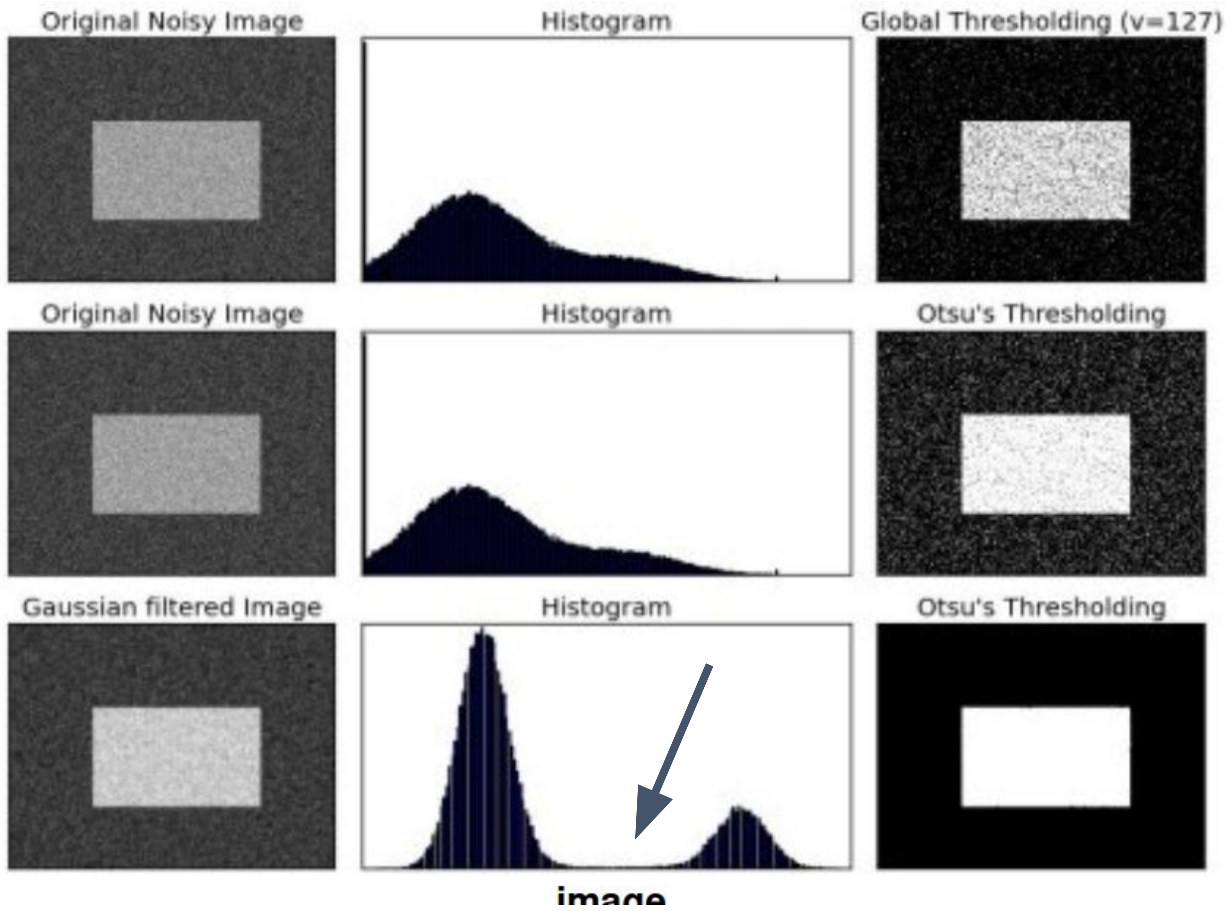
Coding: Binarization

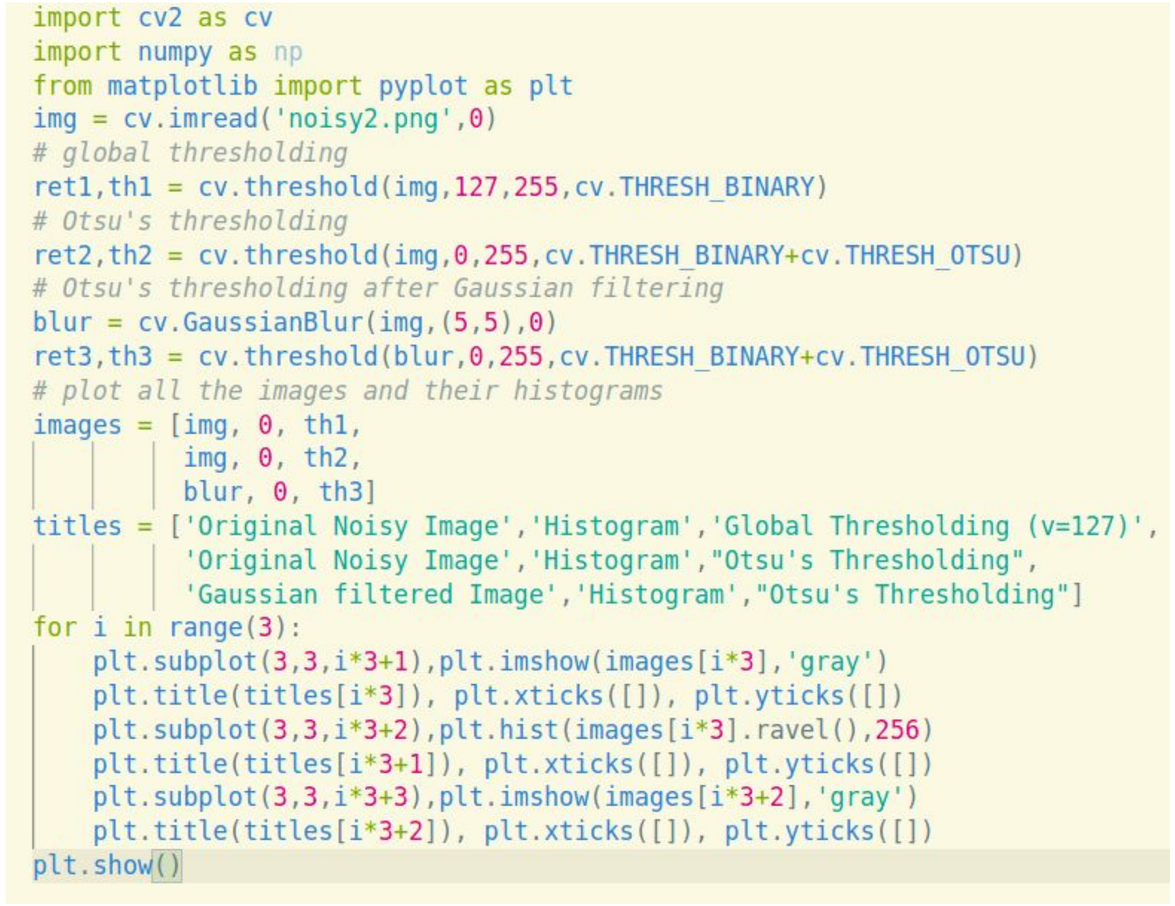




https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Coding: Binarization





```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('noisy2.png',0)
# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Coding: Image gradients (Sobel)

The Sobel Operator is a discrete differentiation operator. It computes an approximation of the gradient of an image intensity function.

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

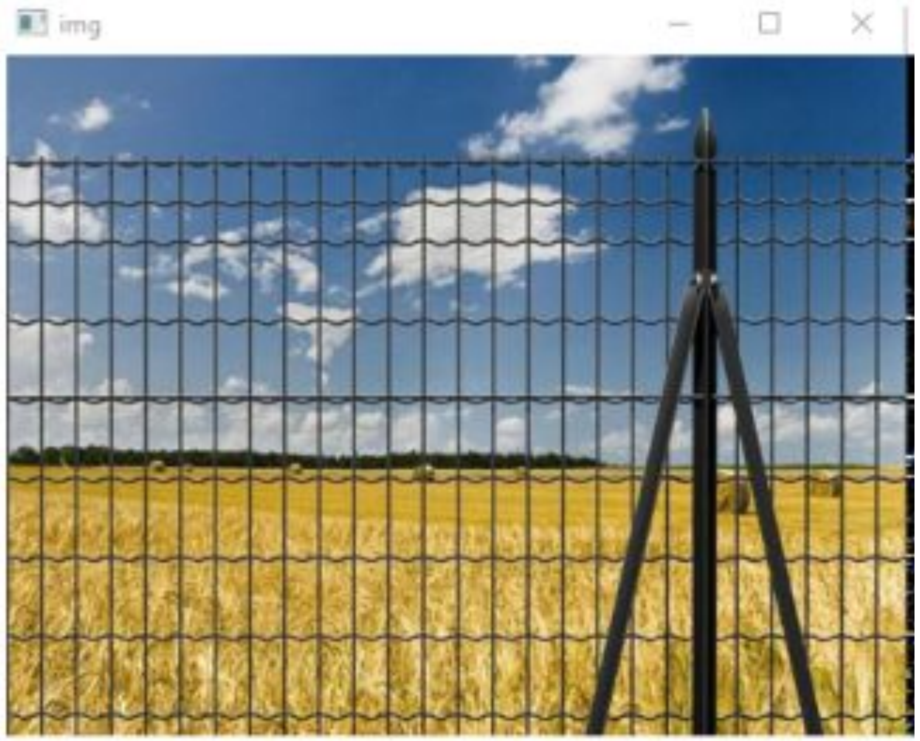
TOTAL GRADIENT (feature)

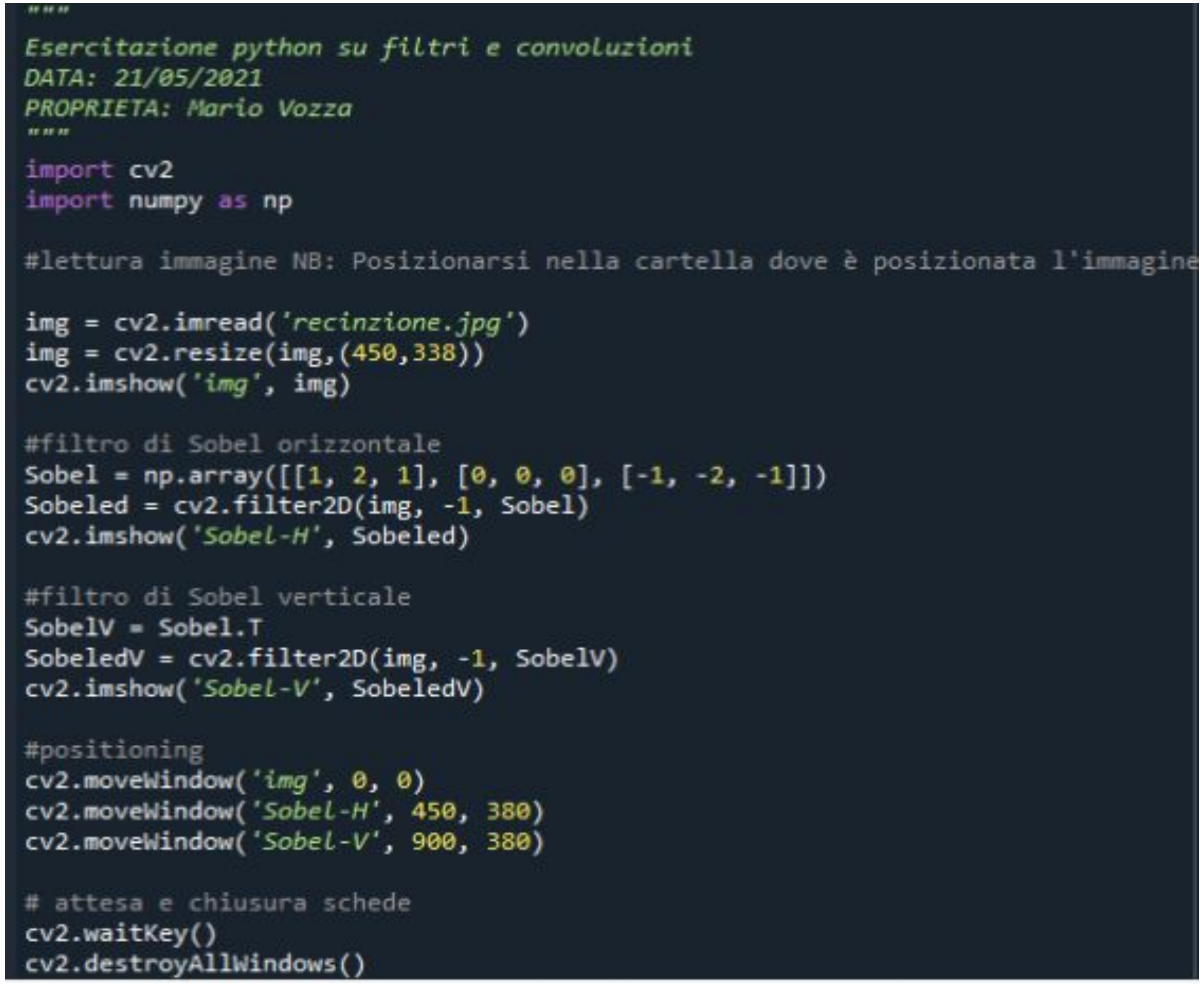
$$G = \sqrt{G_x^2 + G_y^2}$$

OR

$$G = |G_x| + |G_y|$$

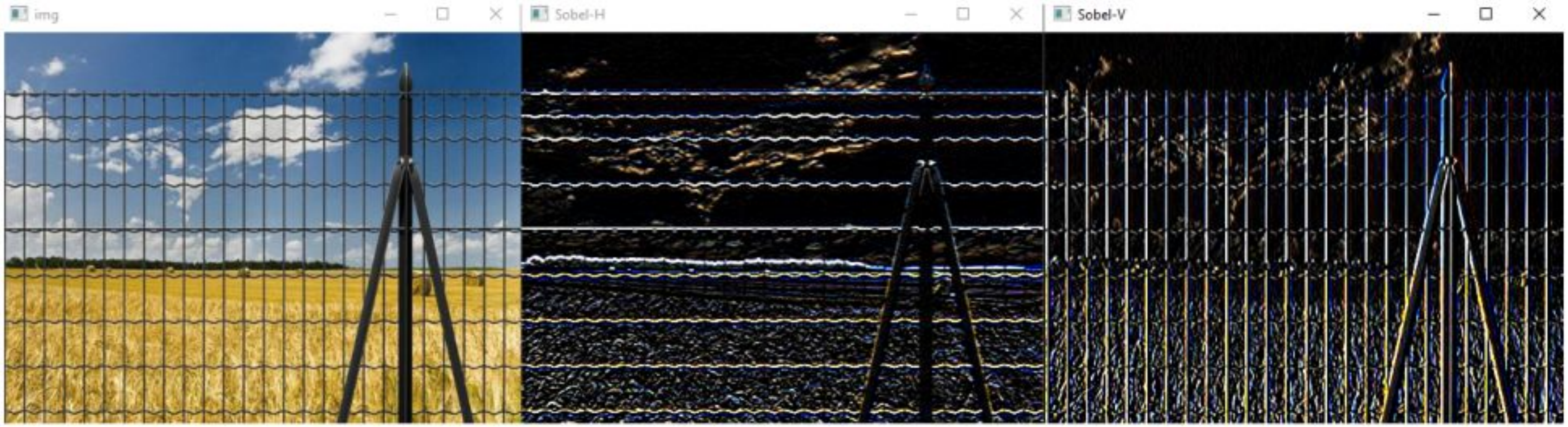
Coding: Image gradients (Sobel)





```
"""
Esercitazione python su filtri e convoluzioni
DATA: 21/05/2021
PROPRIETA: Mario Vozza
"""
import cv2
import numpy as np

#lettura immagine NB: Posizionarsi nella cartella dove è posizionata l'immagine

img = cv2.imread('recinzione.jpg')
img = cv2.resize(img,(450,338))
cv2.imshow('img', img)

#filtro di Sobel orizzontale
Sobel = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
Sobeled = cv2.filter2D(img, -1, Sobel)
cv2.imshow('Sobel-H', Sobeled)

#filtro di Sobel verticale
SobelV = Sobel.T
SobeledV = cv2.filter2D(img, -1, SobelV)
cv2.imshow('Sobel-V', SobeledV)

#positioning
cv2.moveWindow('img', 0, 0)
cv2.moveWindow('Sobel-H', 450, 380)
cv2.moveWindow('Sobel-V', 900, 380)

# attesa e chiusura schede
cv2.waitKey()
cv2.destroyAllWindows()
```

Coding: Image gradients (Sobel)



Coding : Average Blur

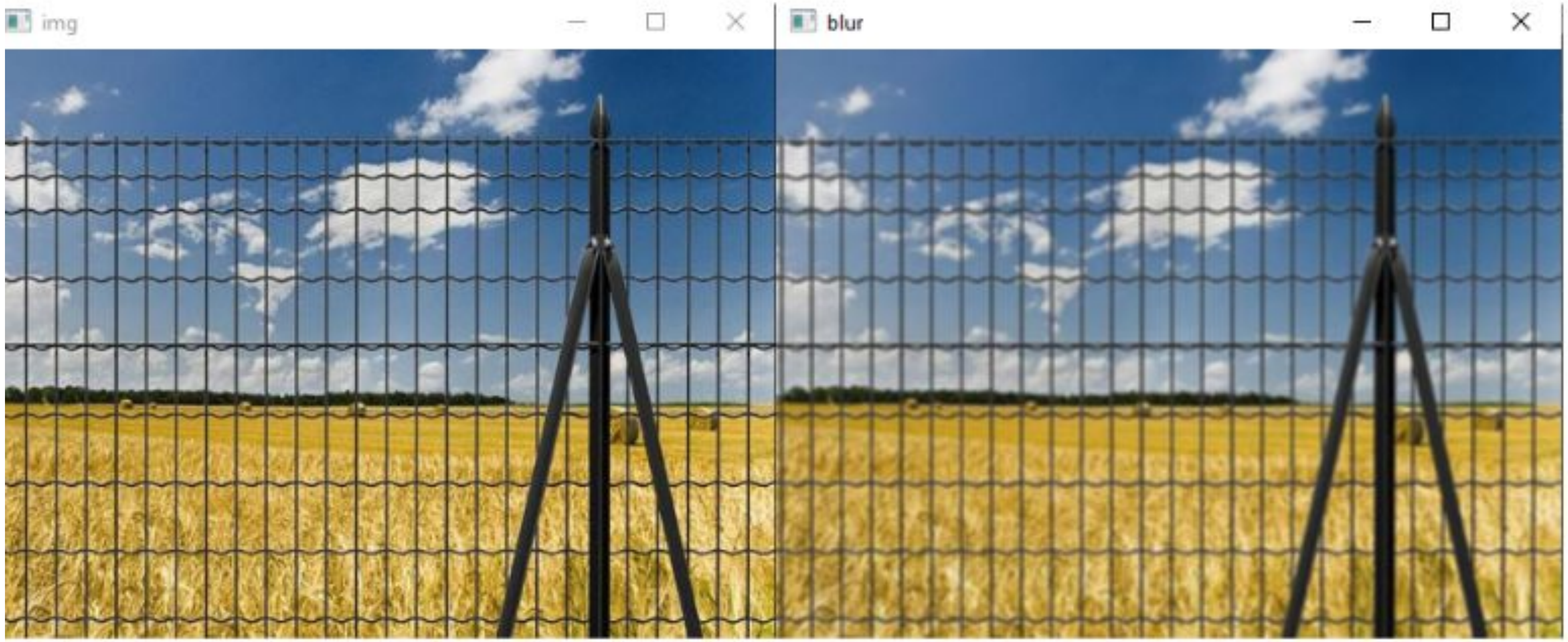
As in one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. LPF helps in removing noise, blurring images, etc. HPF filters help in finding edges in images. OpenCV provides a function **cv.filter2D()** to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel will look like the below:

```python
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('opencv_logo.png')

kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)
```

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

Keep this kernel above a pixel, add all the 25 pixels below this kernel, take the average, and replace the central pixel with the new average value.

Coding : Average Blur



Coding : Gaussian Blur

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

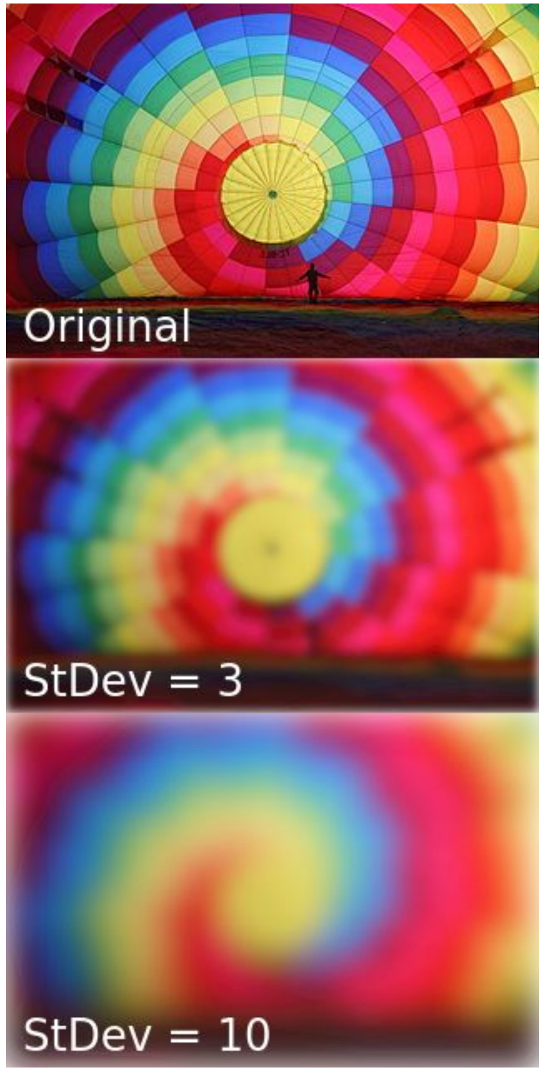
3 x 3 Gaussian Kernel

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

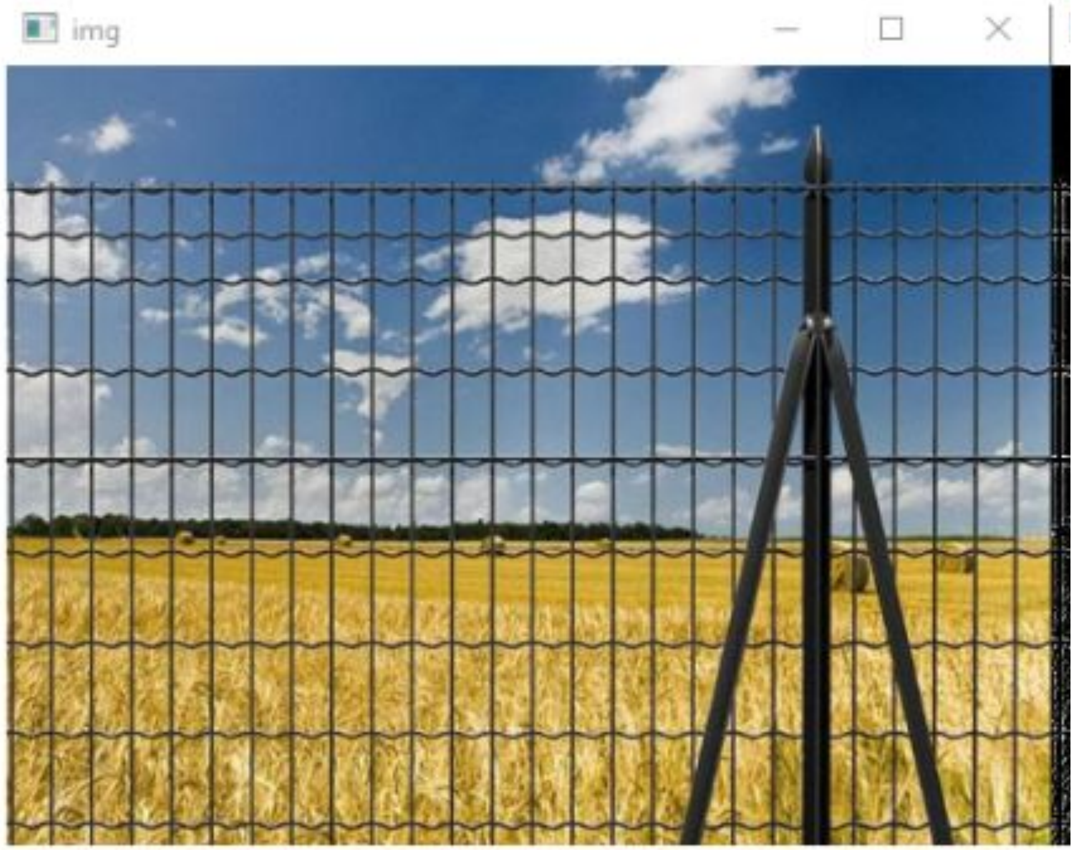
5 x 5 Gaussian Kernel

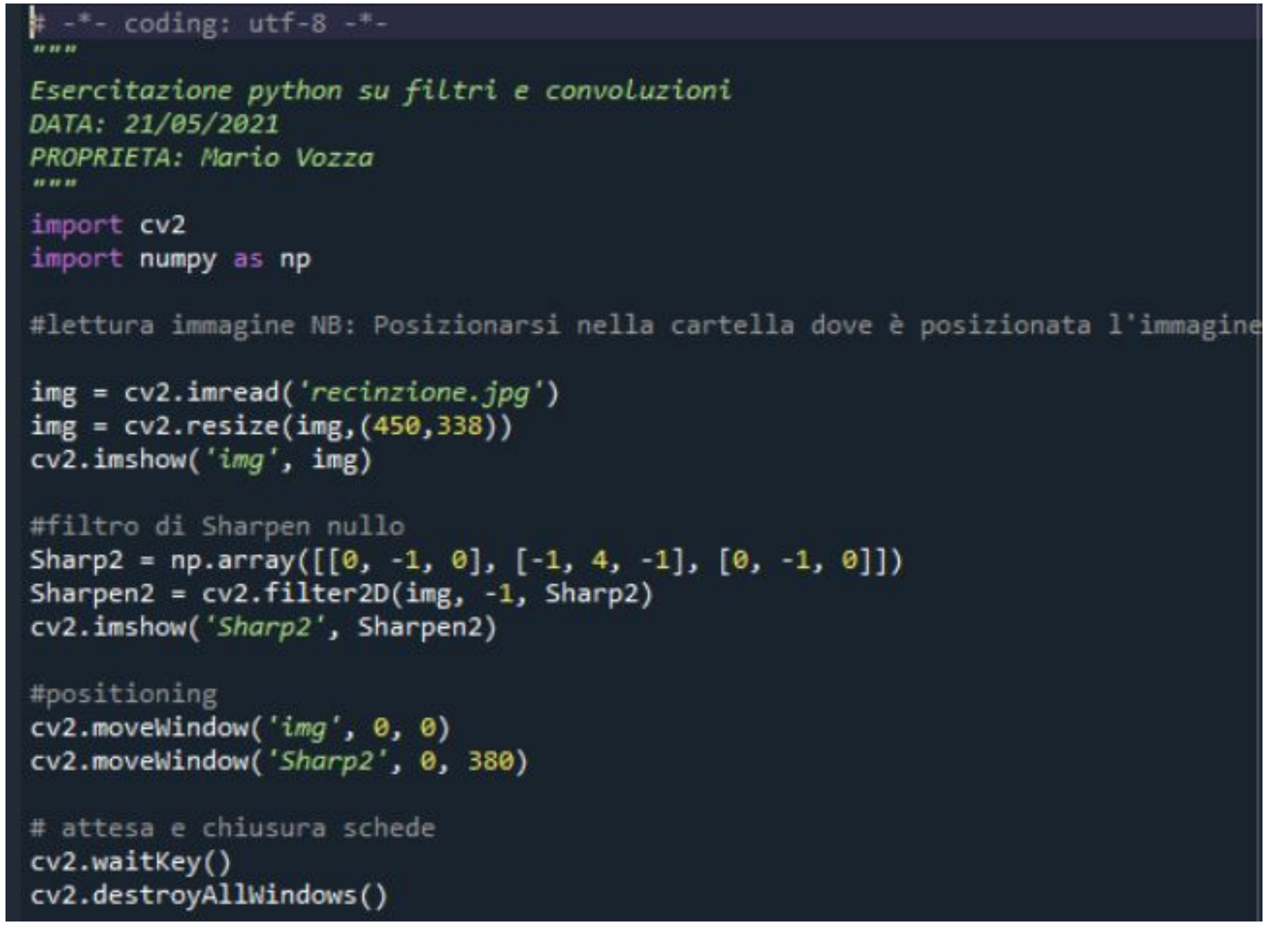
The standard deviation of the Gaussian distribution in X and Y direction should be chosen carefully considering the size of the kernel such that the edges of the kernel is close to zero.You have to choose a right size of the kernel to define the neighborhood of each pixel. If it is too large, small features of the image may be disappeared and the image will look blurred. If it is too small, you cannot eliminate noises in the image.

```
std = 3
blur = cv.GaussianBlur(img,(5,5), std)
```



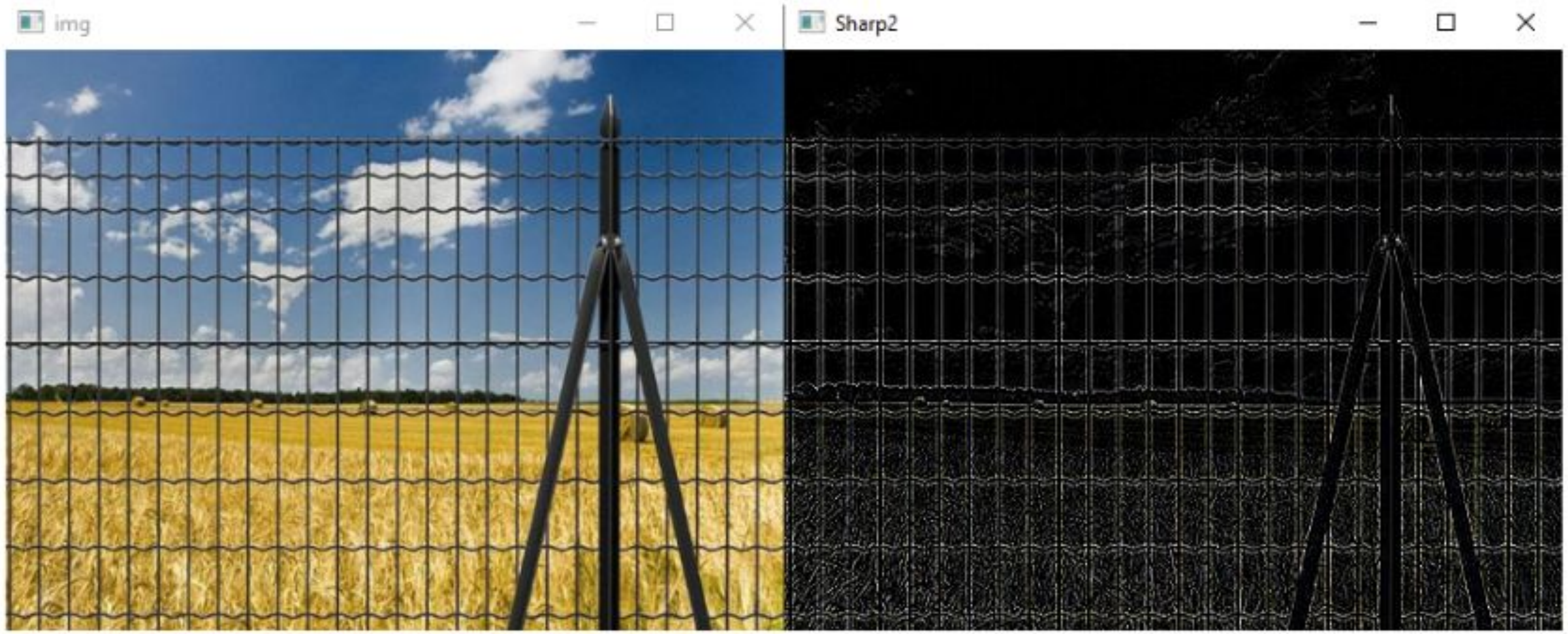
Coding: Sharpening filter 0 mean





```
# -*- coding: utf-8 -*-
"""
Esercitazione python su filtri e convoluzioni
DATA: 21/05/2021
PROPRIETA: Mario Vozza
"""
import cv2
import numpy as np

#lettura immagine NB: Posizionarsi nella cartella dove è posizionata l'immagine

img = cv2.imread('recinzione.jpg')
img = cv2.resize(img,(450,338))
cv2.imshow('img', img)

#filtro di Sharpen nullo
Sharp2 = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
Sharpen2 = cv2.filter2D(img, -1, Sharp2)
cv2.imshow('Sharp2', Sharpen2)

#positioning
cv2.moveWindow('img', 0, 0)
cv2.moveWindow('Sharp2', 0, 380)

# attesa e chiusura schede
cv2.waitKey()
cv2.destroyAllWindows()
```

Coding: Sharpening filter 0 mean

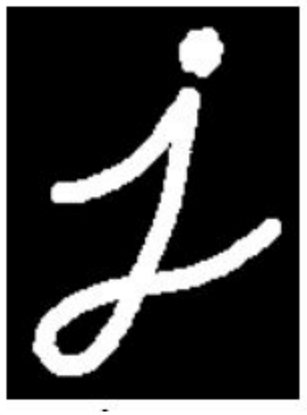


Coding: Morphological filter

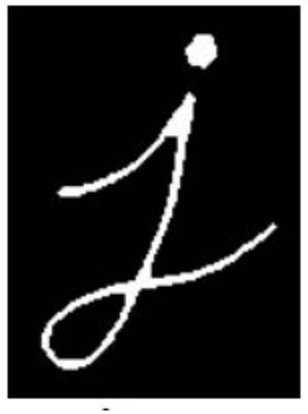
Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

Coding: Morphological filter

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

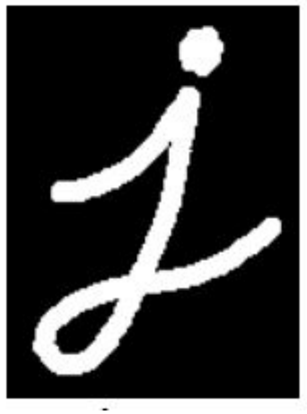


The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

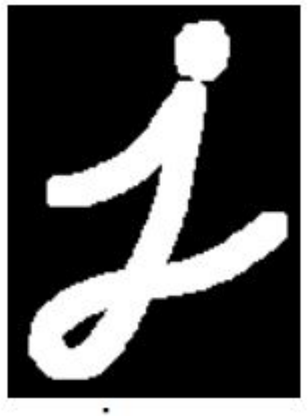


Coding: Morphological filter

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing.

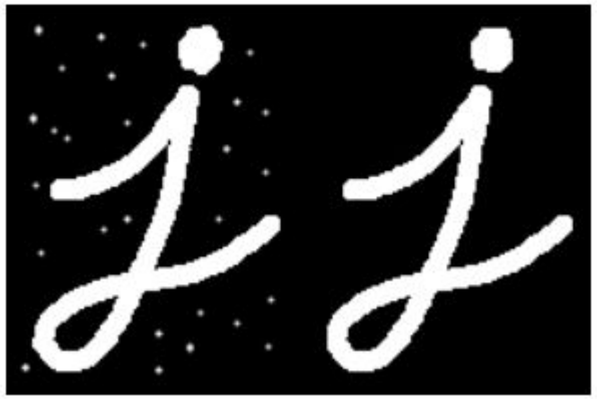


Erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

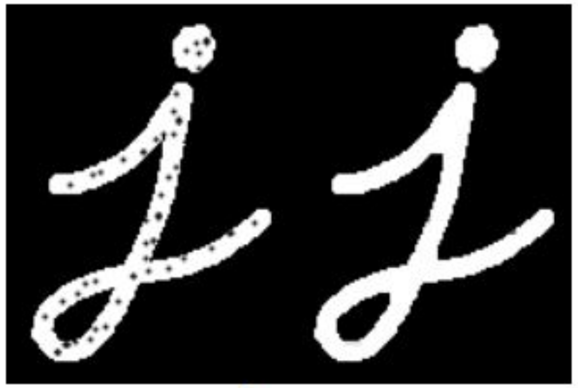


Coding: Morphological filter

Opening is just another name of **erosion followed by dilation**.
It is useful in removing noise

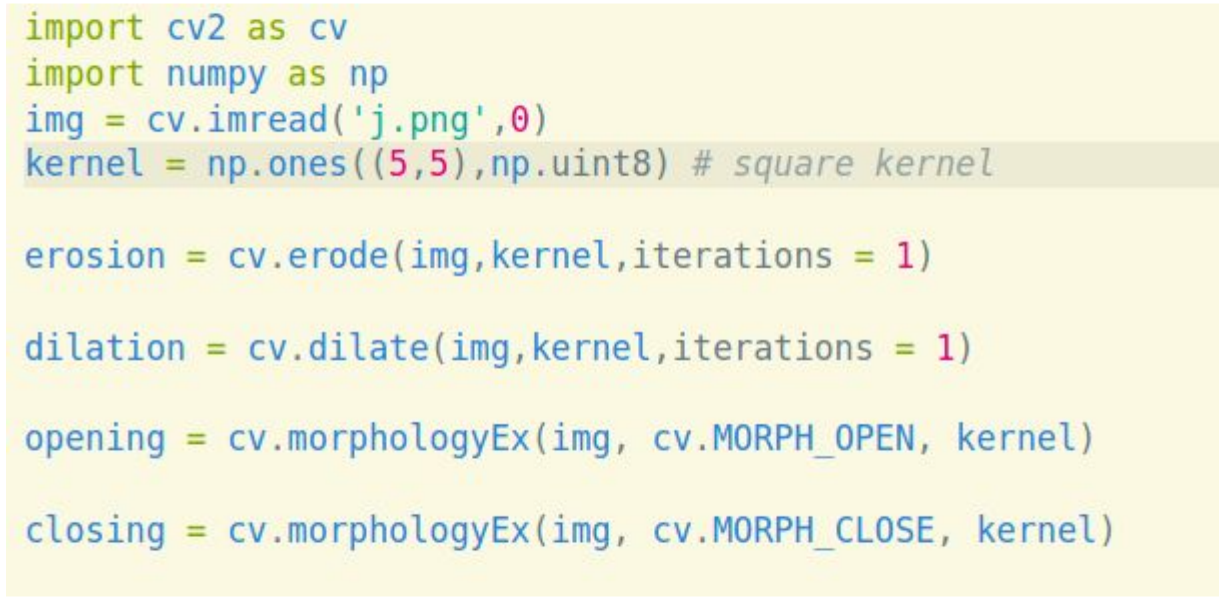


Closing is reverse of Opening, **Dilation followed by Erosion**.
It is useful in closing small holes inside the foreground objects, or small black points on the object.

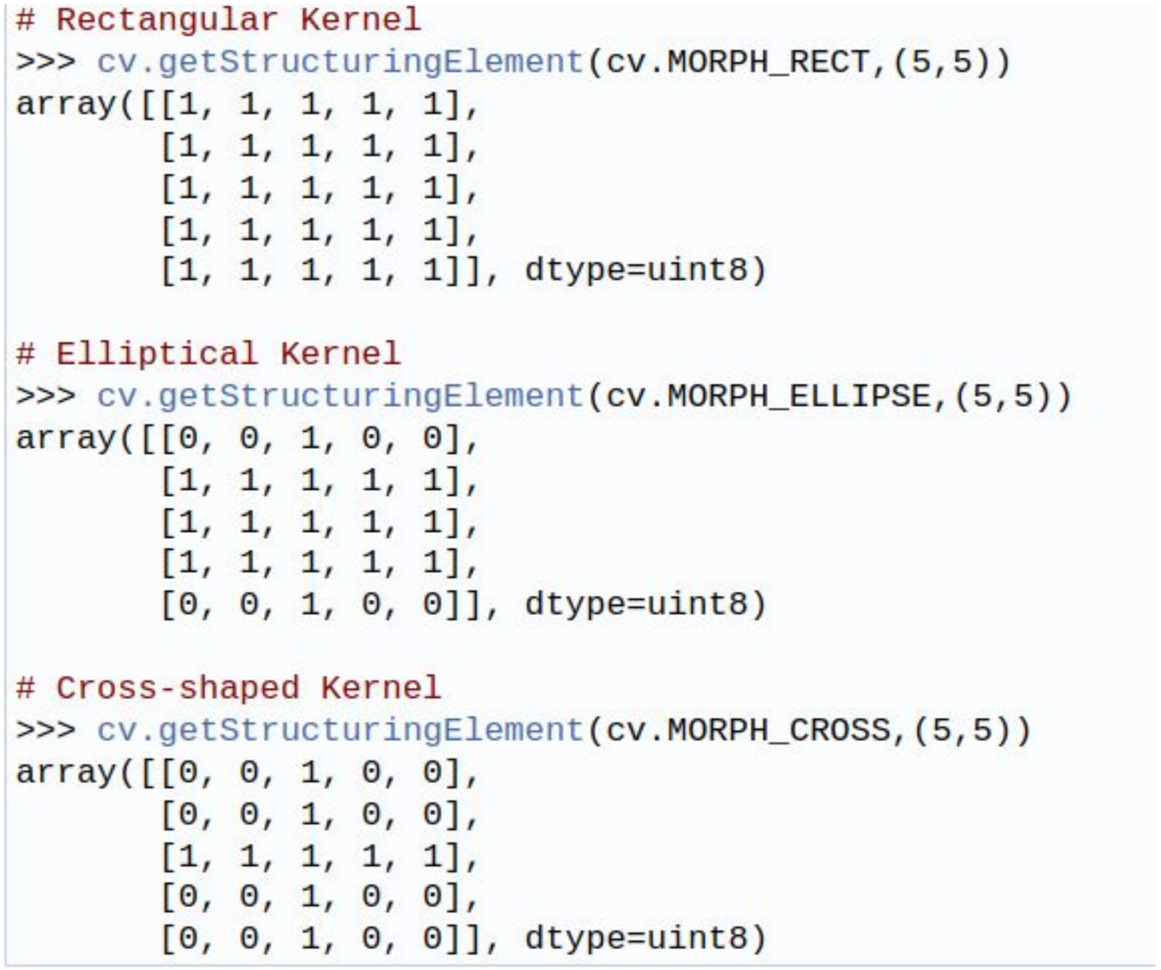


https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Coding: Morphological filter

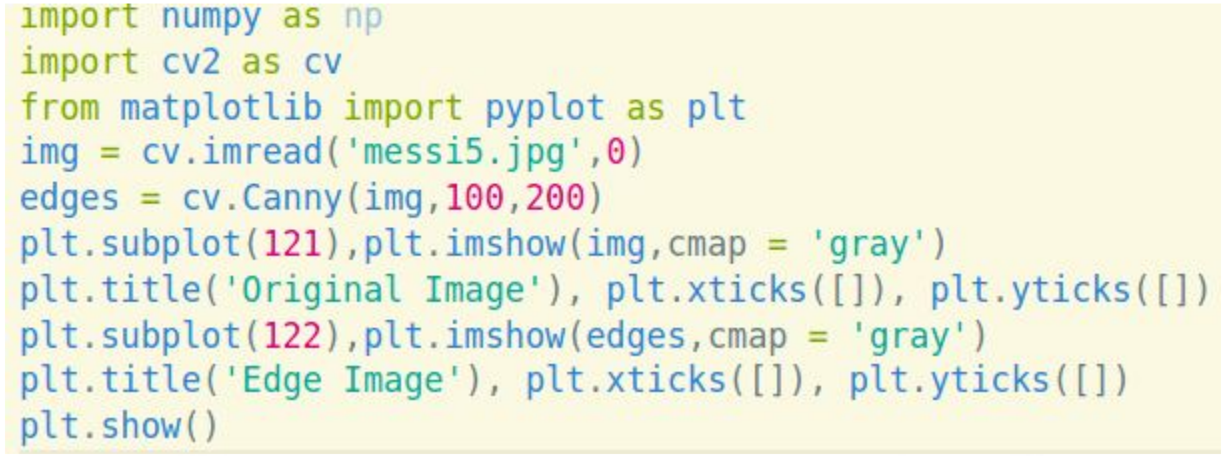


```python
import cv2 as cv
import numpy as np
img = cv.imread('j.png',0)
kernel = np.ones((5,5),np.uint8) # square kernel

erosion = cv.erode(img,kernel,iterations = 1)

dilation = cv.dilate(img,kernel,iterations = 1)

opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)

closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

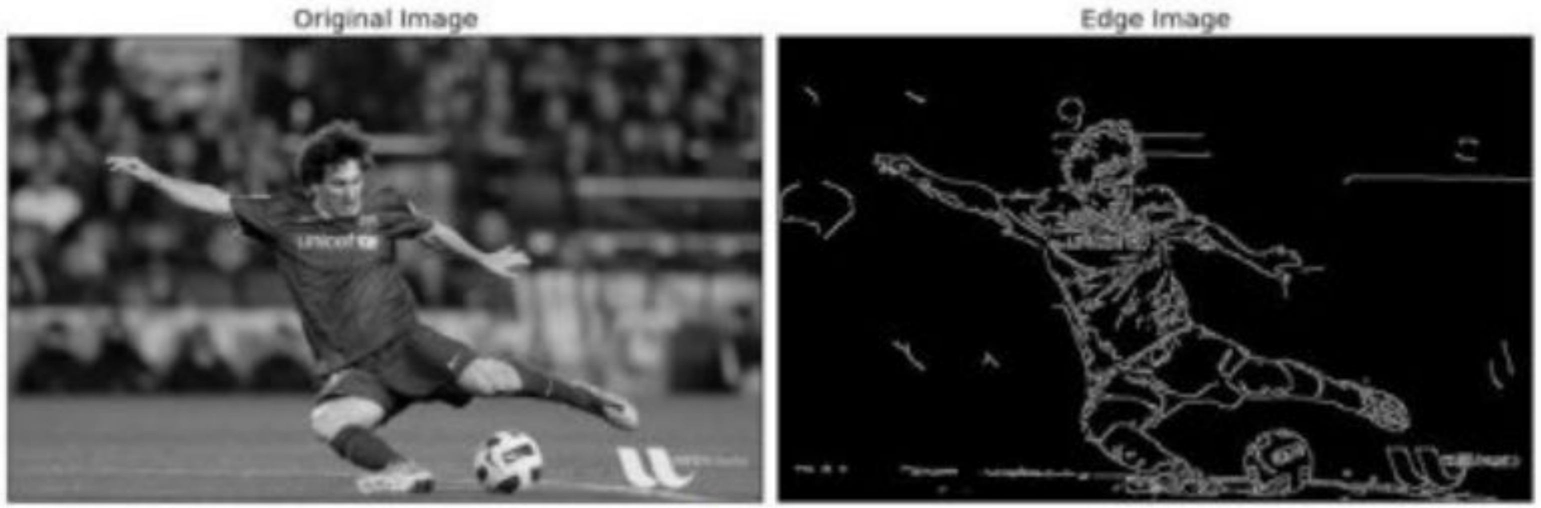


```
# Rectangular Kernel
>>> cv.getStructuringElement(cv.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Coding: Canny edge detector



```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



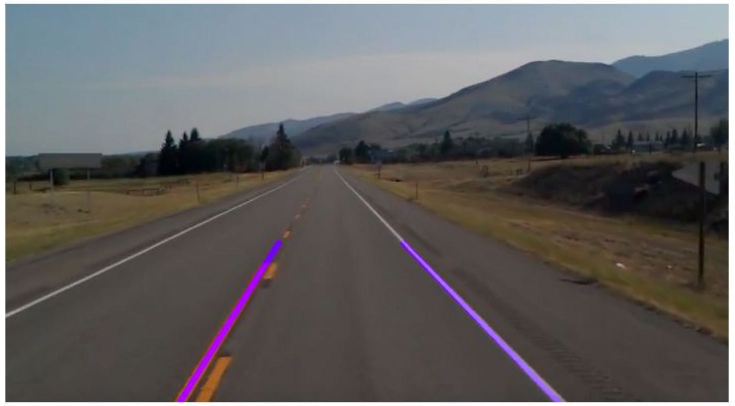
Coding: Hough transform

The Hough transform can be used to identify the parameter(s) of a curve which best fits a set of given edge points. This edge description is commonly obtained from a feature detecting operator such as:

- Laplacian
- Sobel
- Canny
- Roberts Cross
- …

Works also for noisy features, *i.e.* it may contain multiple edge fragments corresponding to a single whole feature. Furthermore, as the output of an edge detector defines only *where* features are in an image, the work of the Hough transform is to determine both *what* the features are and *how many* of them exist in the image.

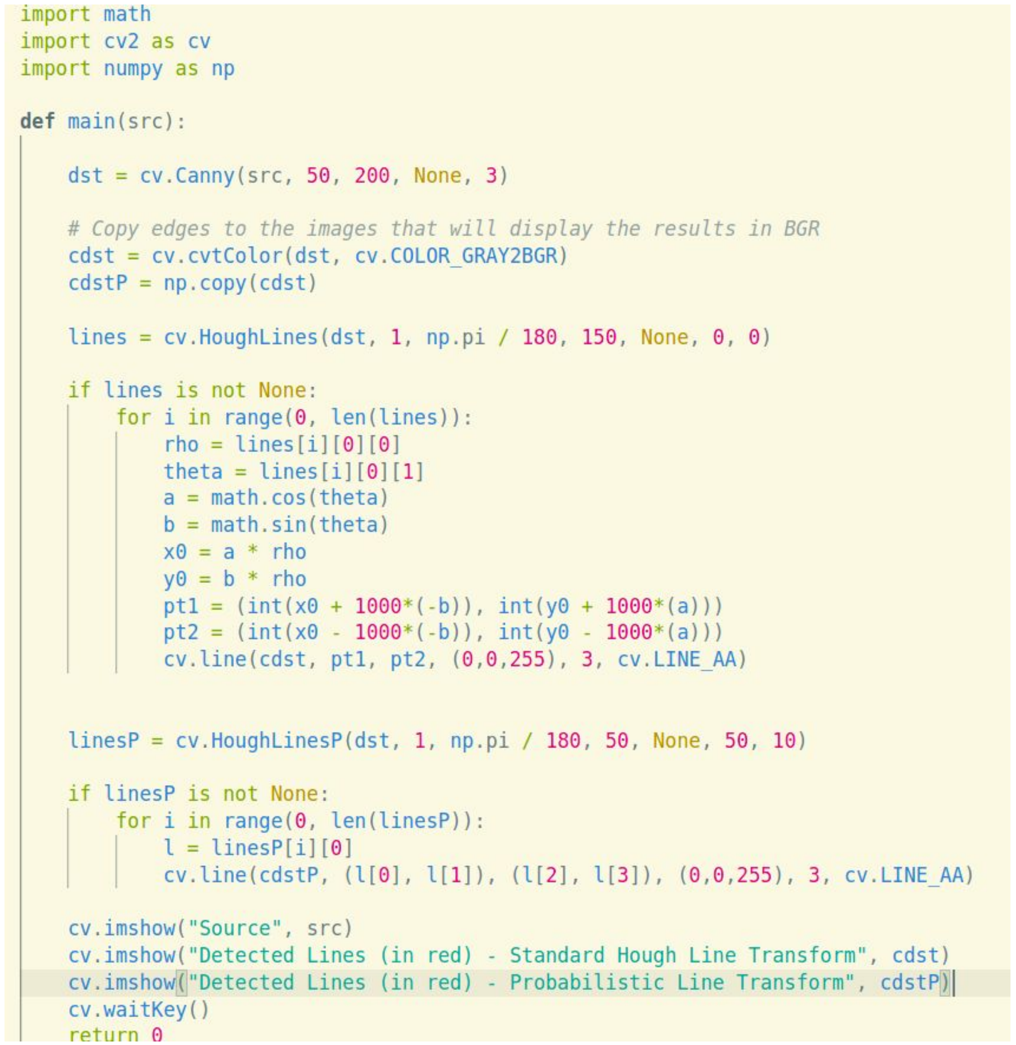
- Ex. used for line detection



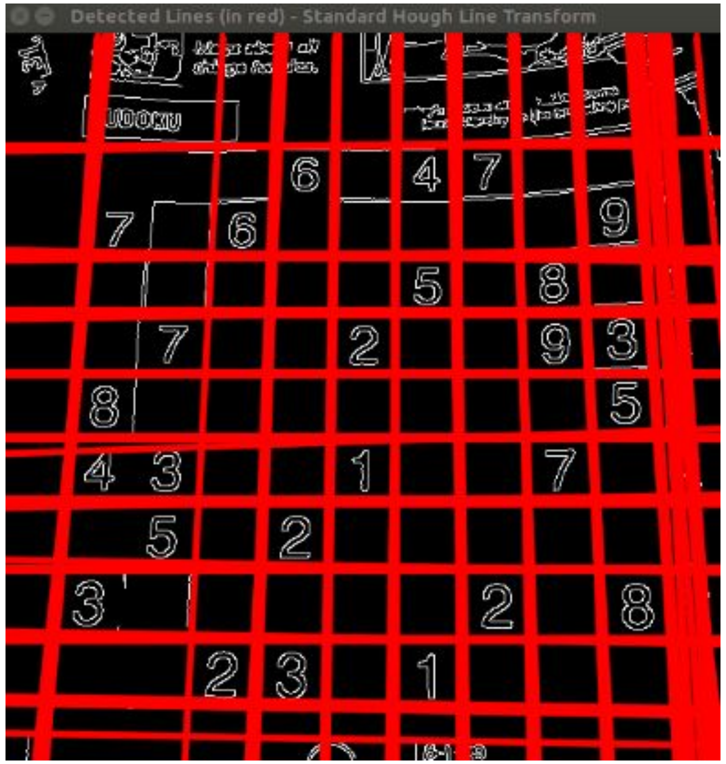
https://blog.paperspace.com/understanding-hough-transform-lane-detection/

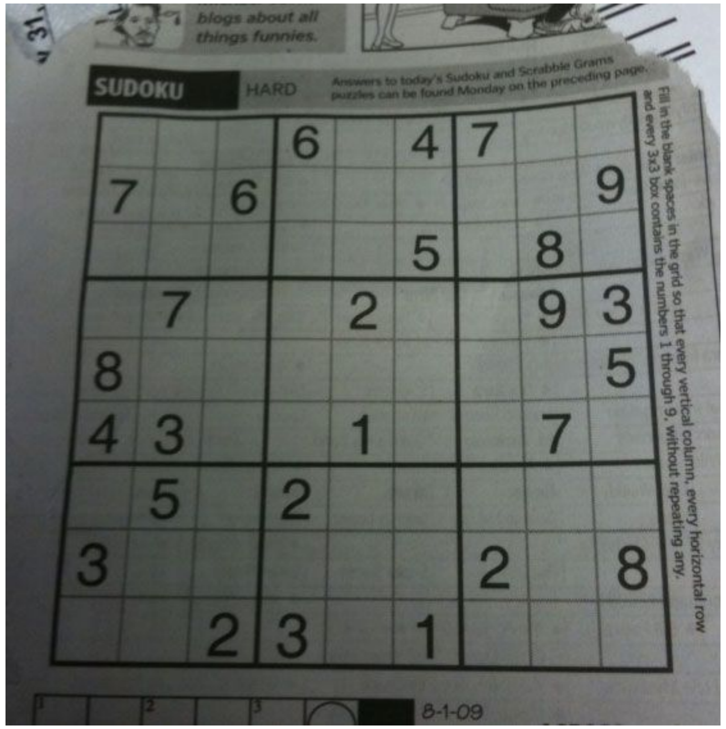
https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Coding: Hough transform



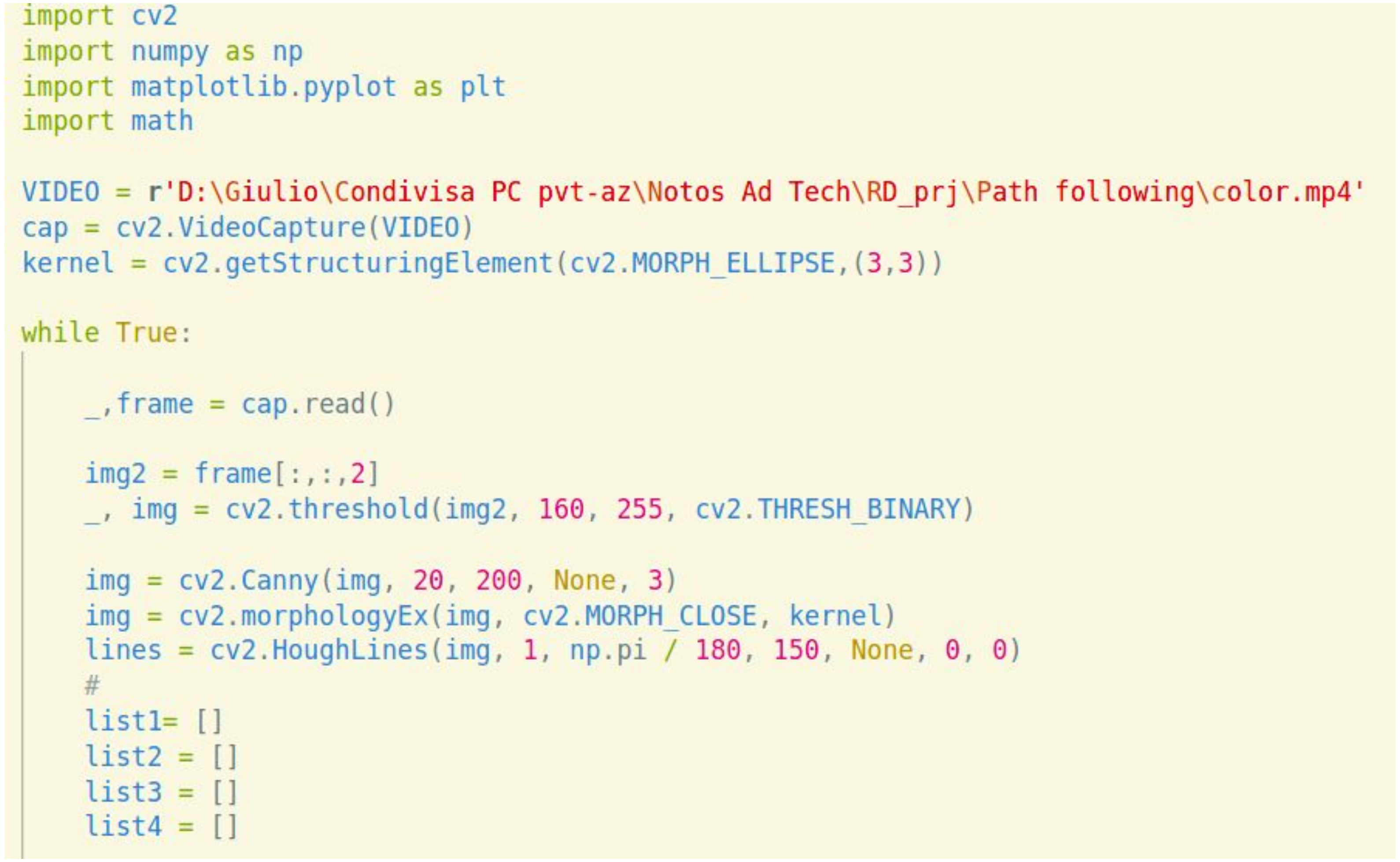
```python
import math
import cv2 as cv
import numpy as np

def main(src):

    dst = cv.Canny(src, 50, 200, None, 3)

    # Copy edges to the images that will display the results in BGR
    cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
    cdstP = np.copy(cdst)

    lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

    if lines is not None:
        for i in range(0, len(lines)):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            a = math.cos(theta)
            b = math.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
            pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
            cv.line(cdst, pt1, pt2, (0,0,255), 3, cv.LINE_AA)


    linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

    if linesP is not None:
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE_AA)

    cv.imshow("Source", src)
    cv.imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst)
    cv.imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP)
    cv.waitKey()
    return 0
```





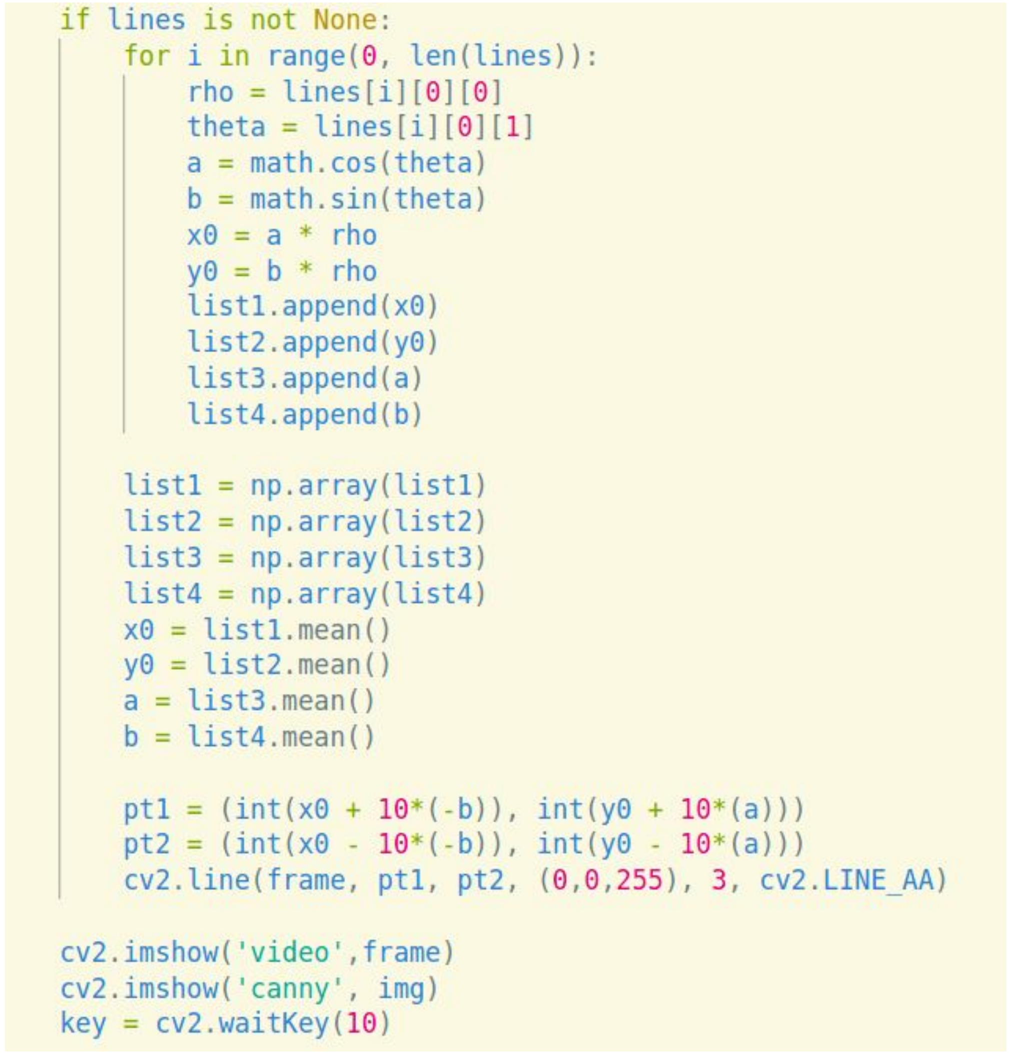
https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Coding: Seam tracker offline

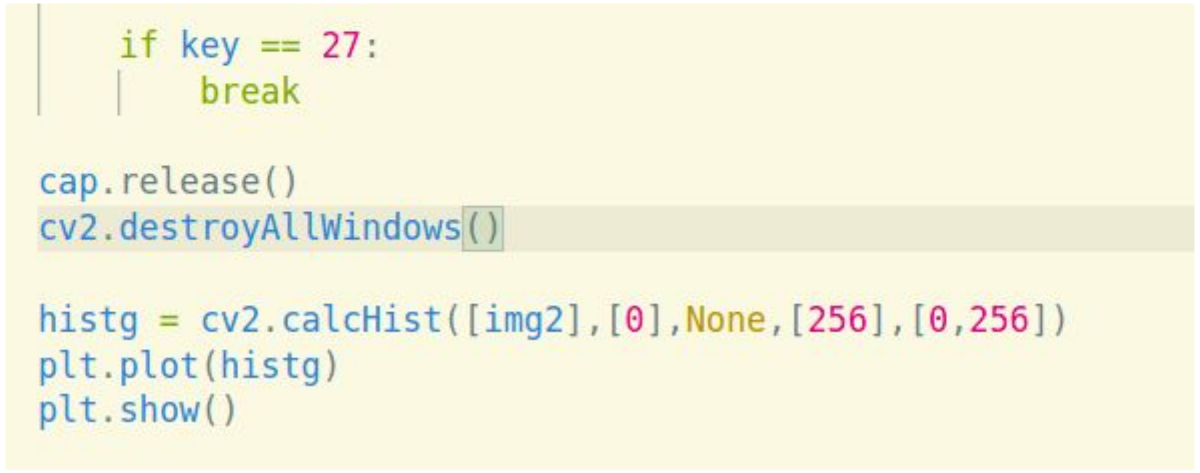


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

VIDEO = r'D:\Giulio\Condivisa PC pvt-az\Notos Ad Tech\RD_prj\Path following\color.mp4'
cap = cv2.VideoCapture(VIDEO)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))

while True:

    _,frame = cap.read()

    img2 = frame[:,:,2]
    _, img = cv2.threshold(img2, 160, 255, cv2.THRESH_BINARY)

    img = cv2.Canny(img, 20, 200, None, 3)
    img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    lines = cv2.HoughLines(img, 1, np.pi / 180, 150, None, 0, 0)
    #
    list1= []
    list2 = []
    list3 = []
    list4 = []
```

Coding: Seam tracker offline



```python
if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a * rho
        y0 = b * rho
        list1.append(x0)
        list2.append(y0)
        list3.append(a)
        list4.append(b)

    list1 = np.array(list1)
    list2 = np.array(list2)
    list3 = np.array(list3)
    list4 = np.array(list4)
    x0 = list1.mean()
    y0 = list2.mean()
    a = list3.mean()
    b = list4.mean()

    pt1 = (int(x0 + 10*(-b)), int(y0 + 10*(a)))
    pt2 = (int(x0 - 10*(-b)), int(y0 - 10*(a)))
    cv2.line(frame, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)

cv2.imshow('video',frame)
cv2.imshow('canny', img)
key = cv2.waitKey(10)
```



```python
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()

histg = cv2.calcHist([img2],[0],None,[256],[0,256])
plt.plot(histg)
plt.show()
```