université de BORDEAUX

Analysing and processing digital images

Serge Bouter

The libraries used for this module

- Opencv: (Open Source Computer Vision Library) is a library of functions imed at real-time computer vision and image processing. Interfaces with C++ (the first) and Python, Java and MATLAB/OCTAVE. It help to manage the Image file and the camera

```
C:\Users\sbouter>python -m pip install opencv-python
```

- Numpy is a library for the Python programming language intended for the computing on multi-dimensional arrays and matrices. (multiplying,sum…)

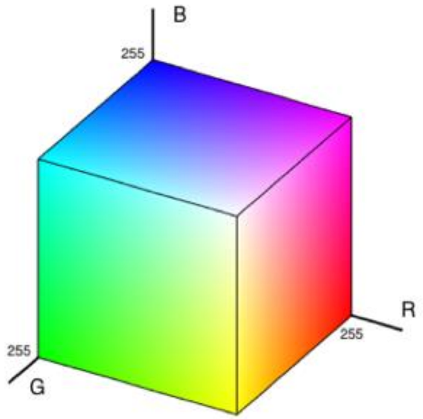
```
C:\Users\sbouter>python -m pip install numpy
```

- Matplotlib is a plotting library for the Python programming language and its extension NumPy

The main objectives

- The main functions of OpenCv library: reading a image or a video, saving a image or a video, accessing to a pixel of an image, drawing lines and other simple shapes…
- Pixel format : BRG or HSV
- Filters : Blur or low band, Contrast or high band, Gaussian, Median
- Edge detection: Sobel, Laplacian, Canny
- Inertia centre, Orientation of a shape in an image
- Hough Transform: detecting lines in an image

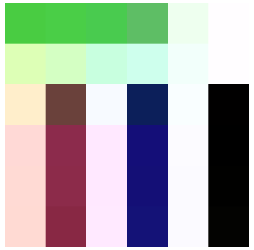
The elementary unit of an image, the pixel



	blue	red	green
dark	0	0	0
yellow	0	255	255
Silver	192	192	192
cyan	255	255	0
white	255	255	255

- Besides, the Blue, Green and Red components, it exists a fourth, the alpha compositing or alpha blending. This component combines one image with a background to create the appearance of partial or full transparency.

Structure of a digital image



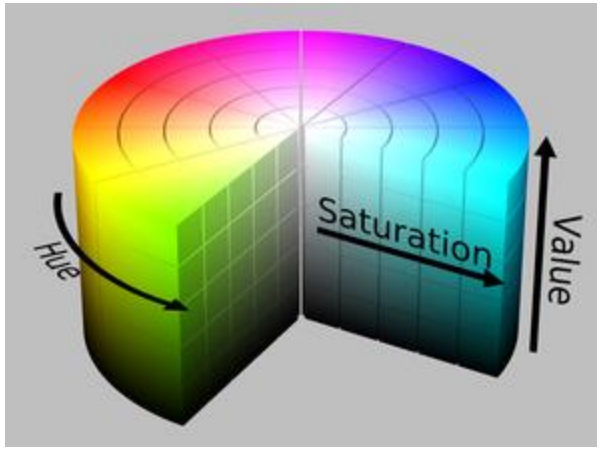
Blue,Green, Red

X or column

	0	1	2	3	4	5	6
0	(0,255,0)	(0,255,0)	(0,255,0)	(0,255,0)	(255,255,255)	(255,255,255)	(255,255,255)
1	(255,255,255)	(255,255,255)	(255,255,255)	(255,255,255)	(255,255,255)	(255,255,255)	(255,255,255)
2	(255,255,255)	(0,0,255)	(255,255,255)	(255,0,0)	(255,255,255)	(0,0,0)	(255,255,255)
3	(255,255,255)	(0,0,255)	(255,255,255)	(255,0,0)	(255,255,255)	(0,0,0)	(255,255,255)
4	(255,255,255)	(0,0,255)	(255,255,255)	(255,0,0)	(255,255,255)	(0,0,0)	(255,255,255)
5	(255,255,255)	(0,0,255)	(255,255,255)	(255,0,0)	(255,255,255)	(0,0,0)	(255,255,255)

y or row

Alternative way to code the colour of a pixel



	Hue(°)	Saturation(%)	Value(%)
dark	0	0	0
yellow	60	100	100
Silver	0	0	75
cyan	180	100	100
white	0	0	100

- The hue is coded according to the corresponding angle on the colour circle: - 0° or 360° red; - 60° yellow; - 120° green; - 180° cyan; - 240° blue; - 300° magenta.
- The Saturation gives the scale of a colour 
- The "value" or "brightness" gives an indication of the lightness or darkness of a colour. The values for brightness vary between 0% (black) and 100% (white).

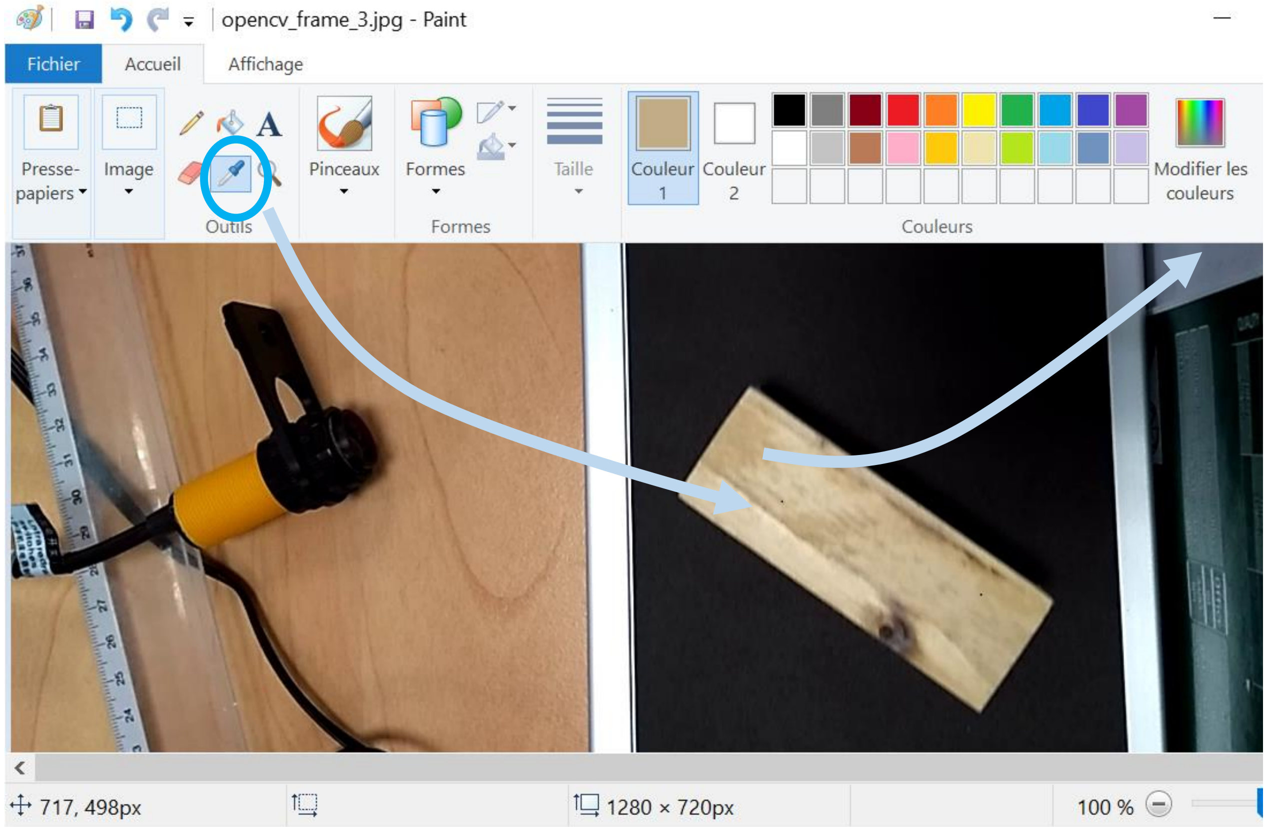
BRG to HVS conversion

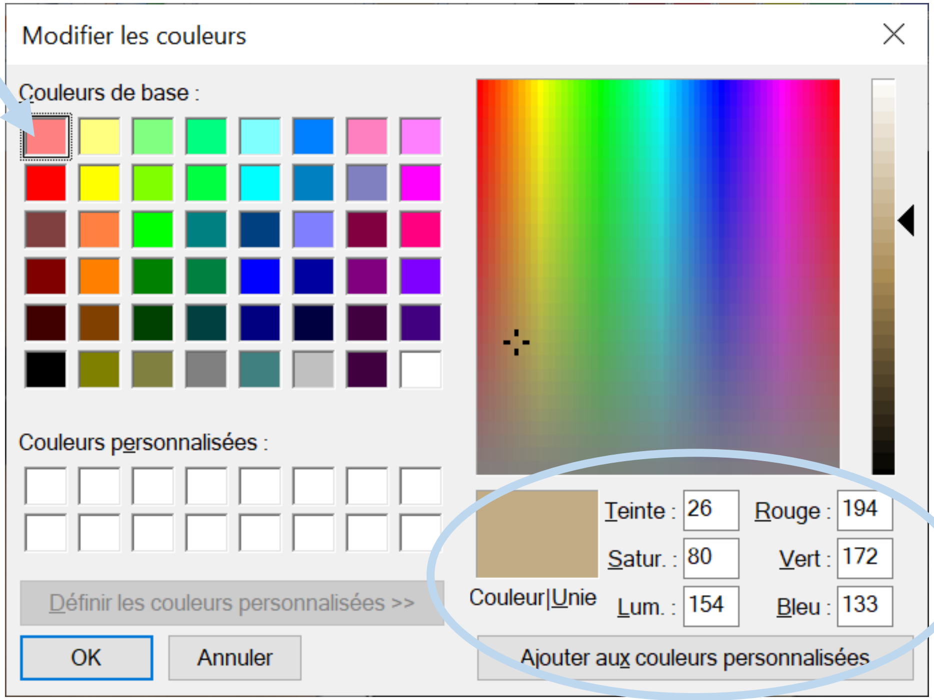
$$V = \max(R, G, B)$$

$$S = \frac{V - \min(R, G, B)}{V}$$

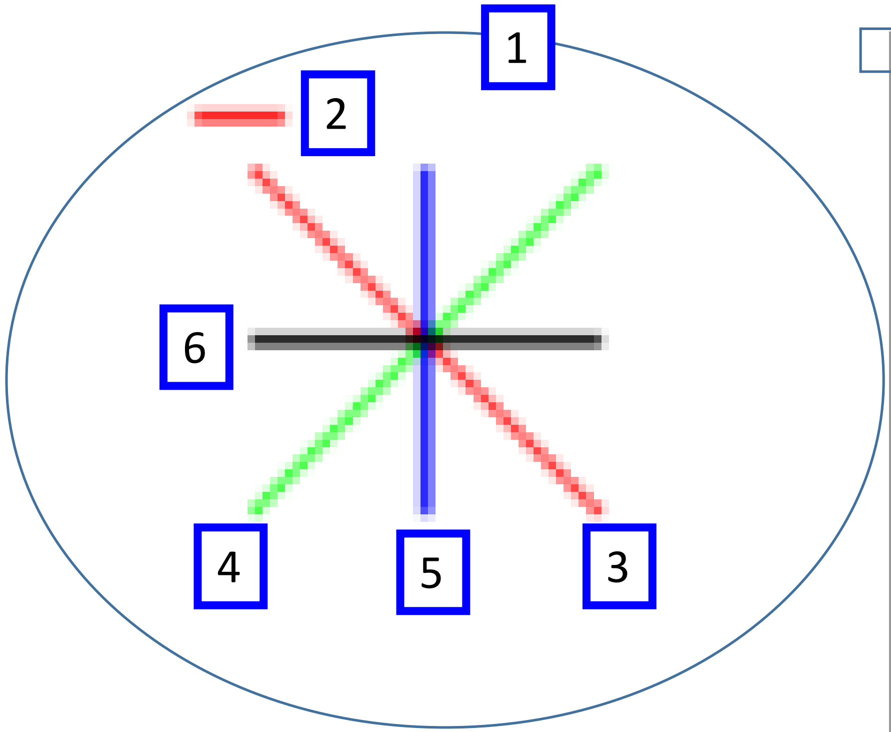
$$H = \frac{1}{6}\begin{cases} \dfrac{G - B}{V - \min(R, G, B)}, \text{ si } V = R \\ 2 + \dfrac{B - R}{V - \min(R, G, B)}, \text{ si } V = G \\ 4 + \dfrac{R - G}{V - \min(R, G, B)}, \text{ si } V = B \end{cases}$$

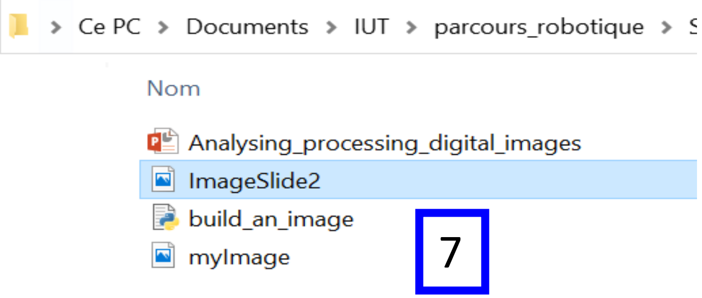
How to get BRG and HSV components with Paint





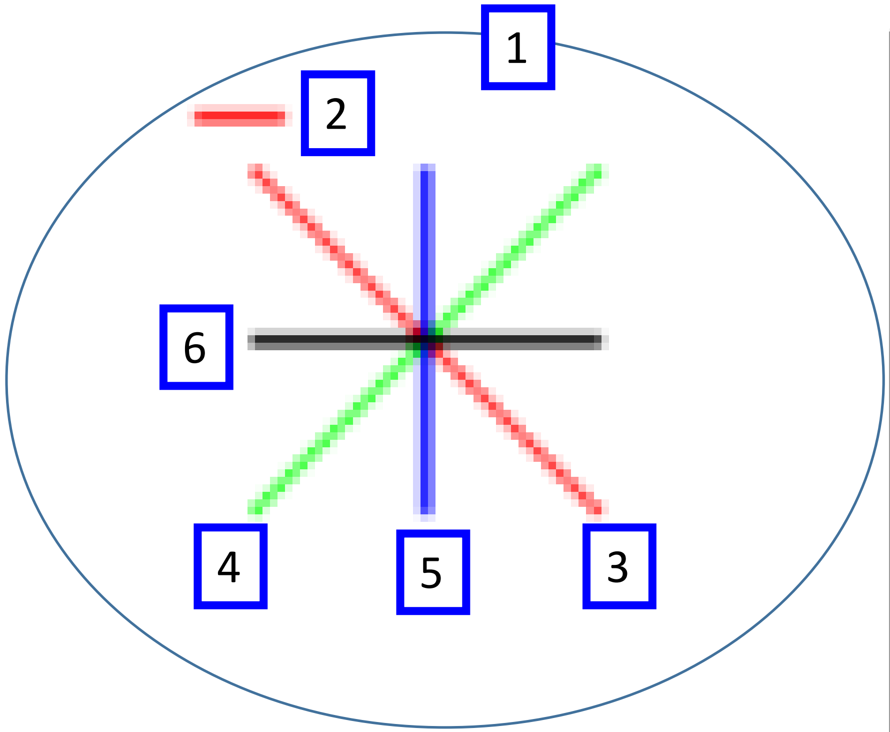
Building an image

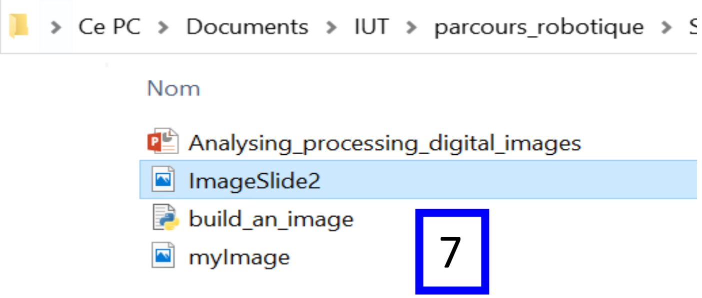





```python
import numpy as np
import cv2


# building a matrix thanks to the numpy library
# builing an image of which the background is white
newImg = 255 * np.ones((50,50, 3), np.uint8)
# builing an image of which the background is black
#newImg = 255 * np.zeros((50,50, 3), np.uint8)


newImg[5,5] = (0, 0, 255)
newImg[5,6] = (0, 0, 255)
newImg[5,7] = (0, 0, 255)
newImg[5,8] = (0, 0, 255)
newImg[5,9] = (0, 0, 255)
newImg[5,10] = (0, 0, 255)
newImg[5,11] = (0, 0, 255)
newImg[5,12] = (0, 0, 255)

cv2.line(newImg, (10, 10), (40, 40), (0, 0, 255))
cv2.line(newImg, (10, 40), (40, 10), (0, 255, 0))
cv2.line(newImg, (25, 10), (25, 40), (255, 0, 0))
cv2.line(newImg, (10, 25), (40, 25), (0, 0, 0))

cv2.imshow('image',newImg)
cv2.waitKey(0)
cv2.imwrite('myImage.jpg', newImg)
cv2.destroyAllWindows()
```

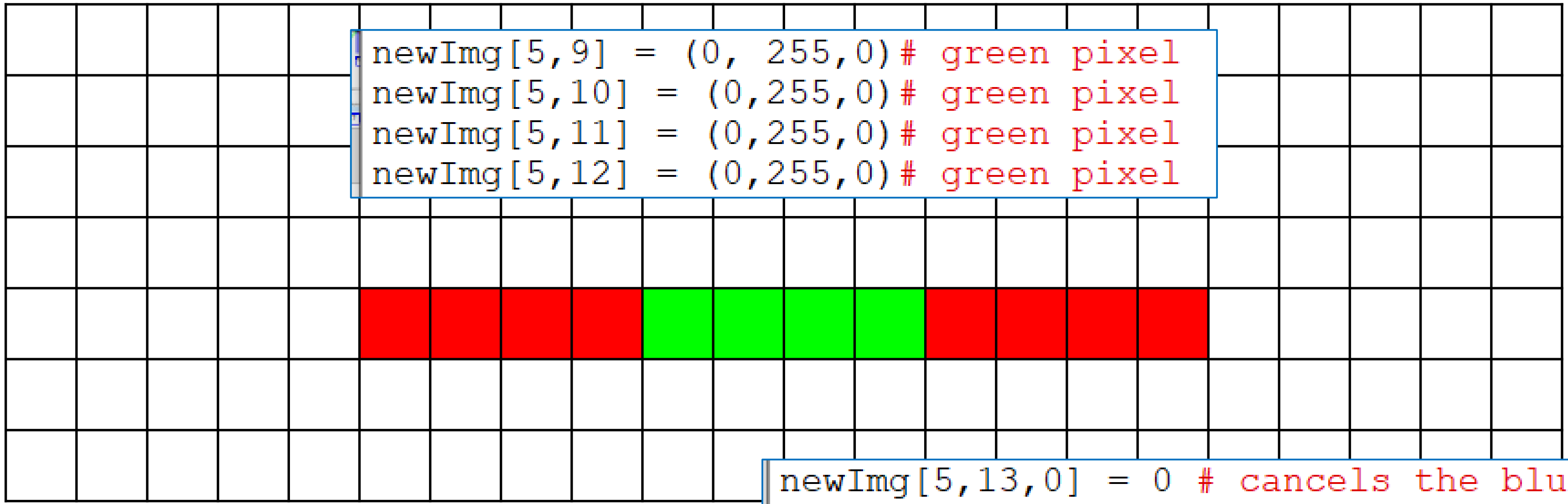
Drawing up the frame, displaying, saving






```python
import numpy as np
import cv2


# building a matrix thanks to the numpy library
# builing an image of which the background is white
newImg = 255 * np.ones((50,50, 3), np.uint8)
# builing an image of which the background is black
#newImg = 255 * np.zeros((50,50, 3), np.uint8)


newImg[5,5] = (0, 0, 255)
newImg[5,6] = (0, 0, 255)
newImg[5,7] = (0, 0, 255)
newImg[5,8] = (0, 0, 255)
newImg[5,9] = (0, 0, 255)
newImg[5,10] = (0, 0, 255)
newImg[5,11] = (0, 0, 255)
newImg[5,12] = (0, 0, 255)

cv2.line(newImg, (10, 10), (40, 40), (0, 0, 255))
cv2.line(newImg, (10, 40), (40, 10), (0, 255, 0))
cv2.line(newImg, (25, 10), (25, 40), (255, 0, 0))
cv2.line(newImg, (10, 25), (40, 25), (0, 0, 0))

cv2.imshow('image',newImg)
cv2.waitKey(0)
cv2.imwrite('myImage.jpg', newImg)
cv2.destroyAllWindows()
```

Locating and characterizing a pixel



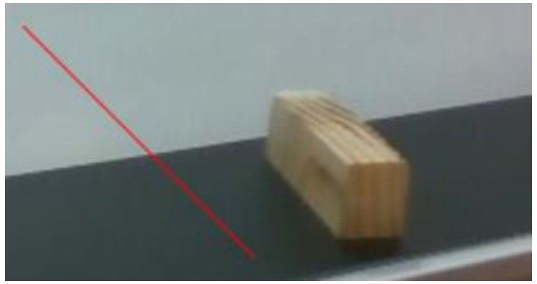
```
newImg[5,9] = (0, 255,0)# green pixel
newImg[5,10] = (0,255,0)# green pixel
newImg[5,11] = (0,255,0)# green pixel
newImg[5,12] = (0,255,0)# green pixel
```

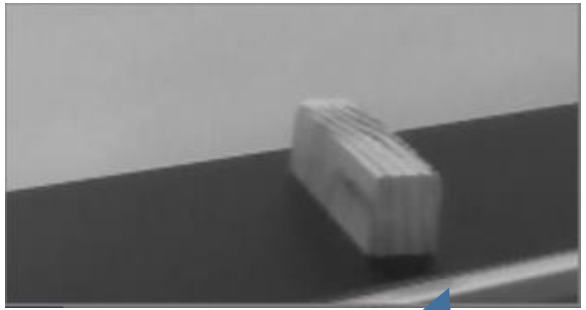
```
newImg[5,5] = (0, 0, 255)# red pixel
newImg[5,6] = (0, 0, 255)# red pixel
newImg[5,7] = (0, 0, 255)# red pixel
newImg[5,8] = (0, 0, 255)# red pixel
```

```
newImg[5,13,0] = 0 # cancels the blue component of the pixel
newImg[5,14,0] = 0 # cancels the blue component of the pixel
newImg[5,15,0] = 0 # cancels the blue component of the pixel
newImg[5,16,0] = 0 # cancels the blue component of the pixel
newImg[5,13,1] = 0 # cancels the green component of the pixel
newImg[5,14,1] = 0 # cancels the green component of the pixel
newImg[5,15,1] = 0 # cancels the green component of the pixel
newImg[5,16,1] = 0 # cancels the green component of the pixel
```

Loading an image from a file

```python
import numpy as np
import cv2

img = cv2.imread('kapla.jpg',cv2.IMREAD_COLOR)
print("rows colomns channels =", img.shape)
(height,width,channels) = img.shape
minFrame = min(width,height)
for i in range(10, minFrame-10):
    img[i,i] = (0,0,255)

cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected
cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode, one channel
cv2.IMREAD_UNCHANGED : Loads image as such, including alpha channel

Capturing images from a camera

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

```
# take a snapshot from a webcam and save it in file
cam = cv2.VideoCapture(0)
img_counter = 0

ret, frame = cam.read()
if not ret:
    print("failed to capture frame")
else:
    cv2.imshow("result", frame)
    img_name = "opencv_frame_{}.png".format(img_counter)
    cv2.imwrite(img_name, frame)
    print("{} stored!".format(img_name))

cam.release()
cv2.destroyAllWindows()
```

Simple processing: colour filtering

```
(height,width, channels) = img.shape

for i in range(height):
    for j in range(width):
        #img[i,j,0]= img[i,j,0] #remains such as
        img[i,j,1]= 0 # cancels the green component
        img[i,j,2]= 0 # cancels the red component

cv2.imshow('image',img)
```

Two channels are cleared

First exercises

You are asked to write programmes which

- Load a image file and modify the content by adding lines.
- The modify the image by adding geometrical elements(line, circle, rectangle…)
- Implement the camera and apply an colour filter to the shot image

université de BORDEAUX

Convolution: the purpose and the calculation

```
[[ 4  7 29 26  5]
 [21 21 26 23 26]
 [27  3 25 26  6]
 [21 13 13 23  1]
 [ 3 15 11 17 18]]
```

×

```
[[-1  0  1]
 [-1  0  1]
 [-1  0  1]]
```

=

```
[[ 0  0  0  0  0]
 [ 0  0  0  0  0]
 [ 0  0 35  0  0]
 [ 0  0  0  0  0]
 [ 0  0  0  0  0]]
```

21*(-1) + 3*(-1) + 13 *(-1) + 26*0 + 25*0 + 13*0 + 23*1 + 26*1 +23*1 = 35

```
[[ 4  7 29 26  5]
 [21 21 26 23 26]
 [27  3 25 26  6]
 [21 13 13 23  1]
 [ 3 15 11 17 18]]
```

×

```
[[-1  0  1]
 [-1  0  1]
 [-1  0  1]]
```

=

```
[[  0   0   0   0   0]
 [  0  28  44 -43   0]
 [  0  -5  35 -31   0]
 [  0  -2  35 -24   0]
 [  0   0   0   0   0]]
```

```
for i in range(1,height-1):
    for j in range(1,width-1):
        result = 0
         for m in range(-1,2):
            for n in range(-1,2):
                result += img[i+m,j+n]*kernelX[m+1,n+1]
        im[i,j] = result
```

Image processing:

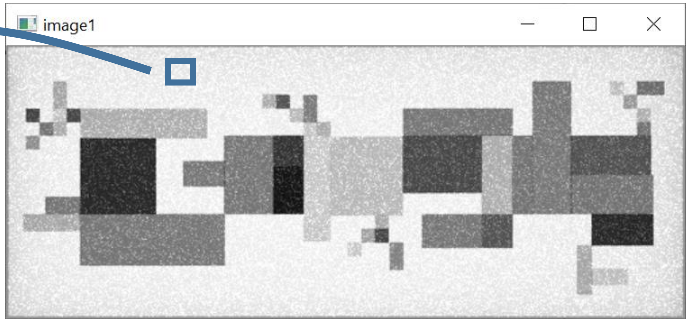
- Blur or low band filter
- Constrast improvement or high band filtre
- Edge detection
- …

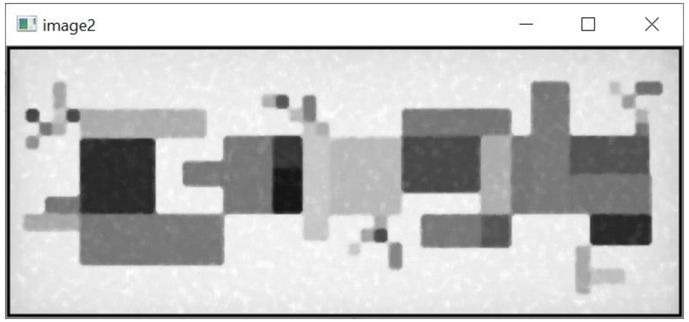
ImageIn ImageOut

CONVOLUTION

ImageOut having been assigned the results of the calculation

Median filter 5x5, a non-linear filter





70	45	41	45	38
84	63	39	44	43
44	101	77	77	42
109	73	43	54	42
50	42	82	46	71

Settling the matrix as a list:

```
neighbours = []
…
neighbours.append(img[100-k,100-l])
```

49	50	48	45	44
46	50	45	45	43
46	49	46	45	43
46	50	52	48	43
46	50	52	54	48

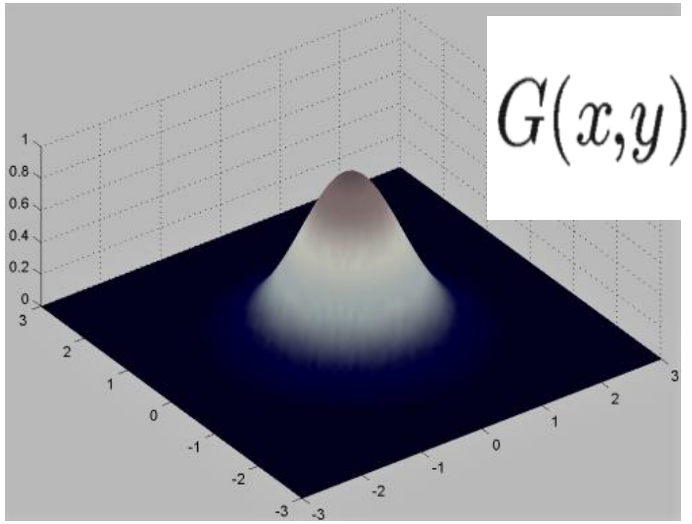
[70, 45, 41, 45, 38, 84, 63, 39, 44, 43, 44, 101, 77, 77, 42, 109, 73, 43, 54, 42, 50, 42, 82, 46, 71]

```
neighbours.sort()
```

[38, 39, 41, 42, 42, 42, 43, 43, 44, 44, 45, 45, 46 50, 54, 63, 70, 71, 73, 77, 77, 82, 84, 101, 109]

Gaussian Filter

```
Gaussian Filter
[[0.0582318  0.10133665 0.0582318 ]
 [0.10133665 0.17634897 0.10133665]
 [0.0582318  0.10133665 0.0582318 ]]
Normalized Gaussian Filter
[[0.07148314 0.12439703 0.07148314]
 [0.12439703 0.2164793  0.12439703]
 [0.07148314 0.12439703 0.07148314]]
sum of the terms of the Kernel
0.9999999999999998
```



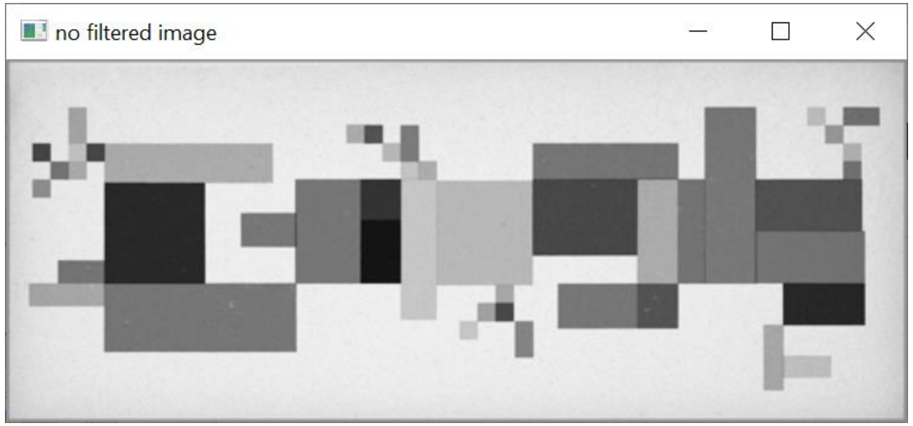
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

```
def Gaussian(x,y,sigma):
    num = ( (y**2)+(x**2))/(2*(sigma**2))
    exp = np.exp( -num )
    den = 2*np.pi*(sigma**2)
    return 1/den*exp
```

```
for y in range(-1,2):
    for x in range(-1,2):
        GaussianBlurKernel[x+1,y+1] = Gaussian(x,y,0.95)
        sumTerm += GaussianBlurKernel[x+1,y+1]
```

```
GaussianBlurKernel=GaussianBlurKernel/sumTerm
```

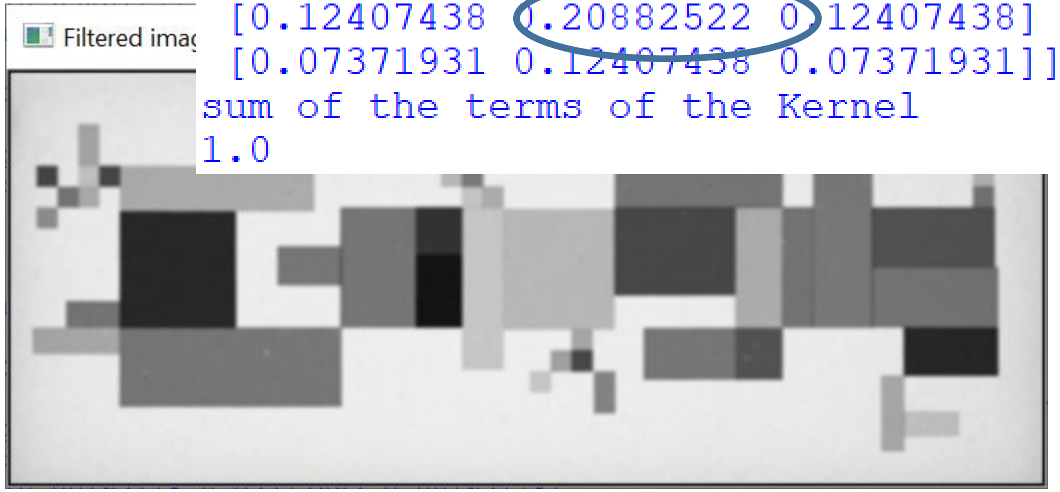
Gaussian Filter(2)



Sigma = 0.95

Normalised matrix

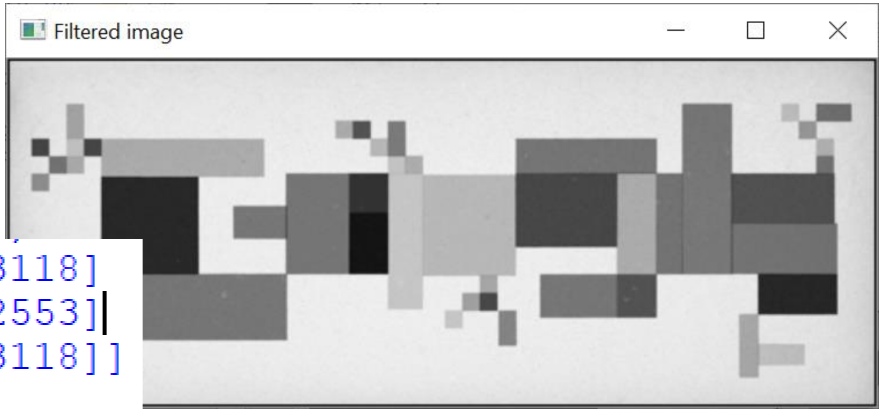
```
[[0.07371931 0.12407438 0.07371931]
 [0.12407438 0.20882522 0.12407438]
 [0.07371931 0.12407438 0.07371931]]
sum of the terms of the Kernel
1.0
```



Sigma = 0.4

Normalised matrix

```
[[0.00163118 0.03712553 0.00163118]
 [0.03712553 0.84497315 0.03712553]
 [0.00163118 0.03712553 0.00163118]]
sum of the terms of the Kernel
0.9999999999999998
```



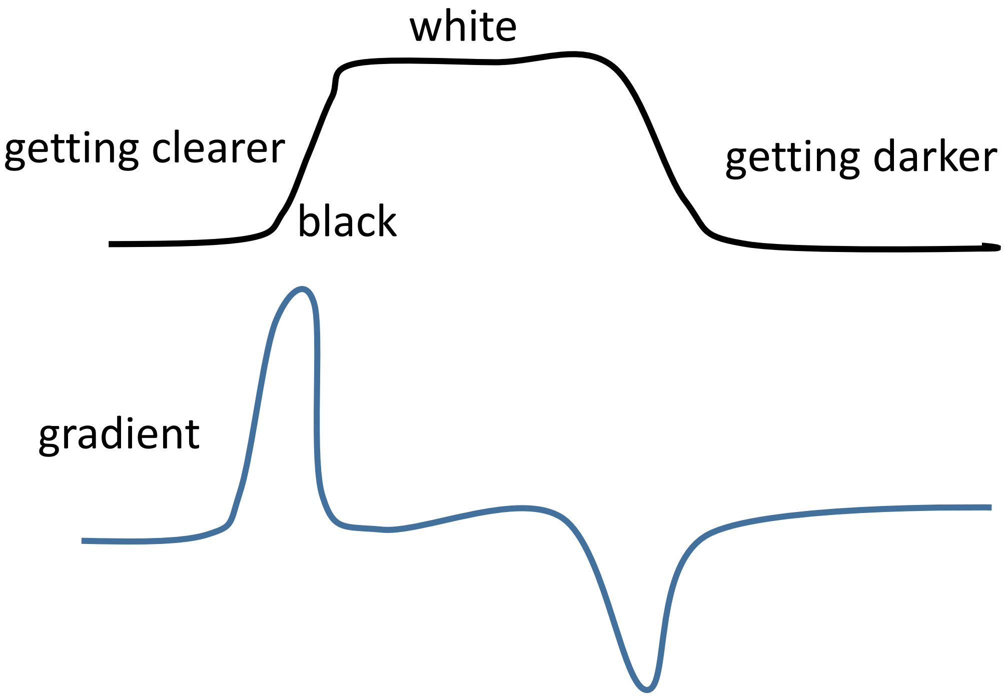
Edge: Definition of the gradient

The gradient is a vector which has magnitude and direction. This vector stands for a change of a variable, here the grayscale. It quantifies the magnitude of this change according to a direction.

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

magnitude of this change according to x axis

magnitude of this change according to y axis



magnitude of this change

$$magn(\nabla f) = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} = \sqrt{M_x{}^2 + M_y{}^2}$$

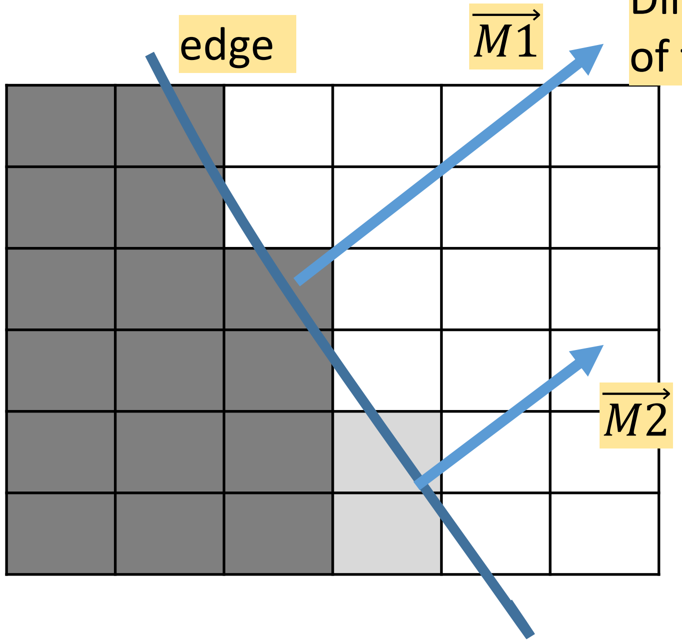
$$magn(\nabla f) \approx |M_x| + |M_y|$$

Approximation to save time of computations

direction of this change

$$dir(\nabla f) = \tan^{-1}(M_y/M_x)$$

Properties of the gradient



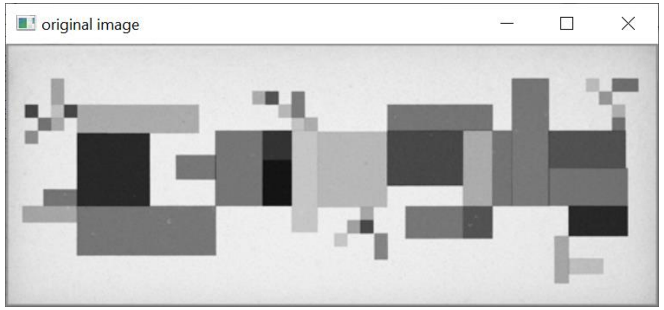
Direction and the magnitude of the gradient

$|\overrightarrow{M1}| > |\overrightarrow{M2}|$

- The magnitude of gradient provides information about the intensity of the change between areas separated by a “line” which define the edge.
- The direction of gradient is always perpendicular to the direction of the “line defining of the edge.

Edge Detection: Sobel according to the either X or Y axis

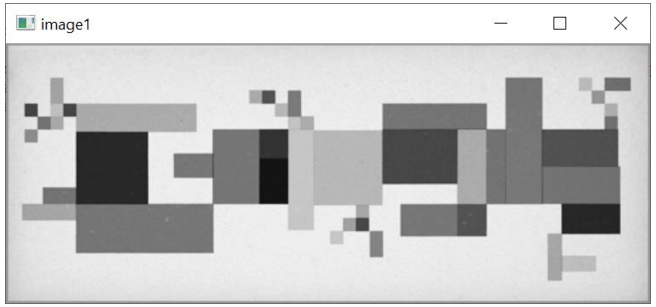
Horizontal changes: This is computed by convolving the image with a kernel Gx with odd size. An example with a kernel size of 3

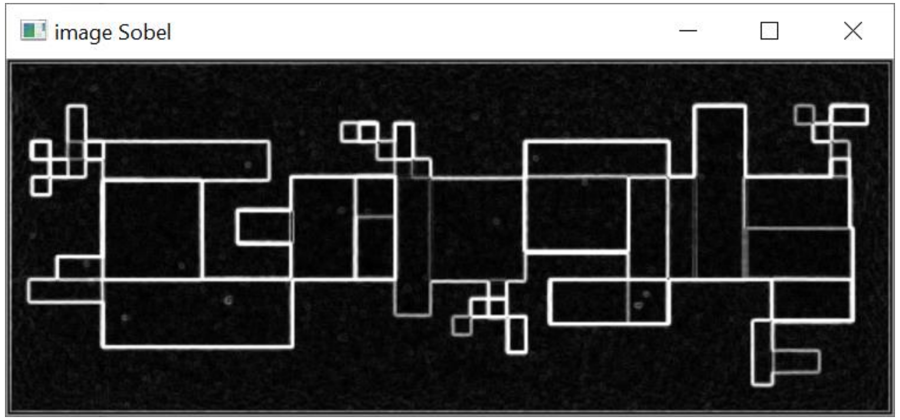


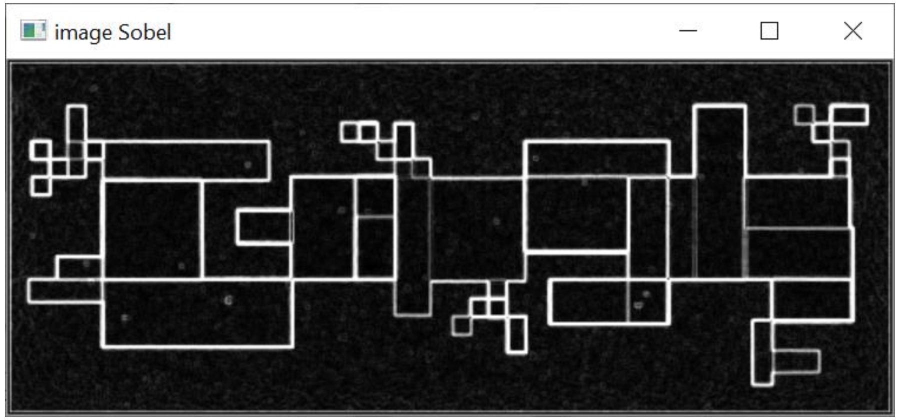
Vertical changes: This is computed by convolving the image with a kernel Gy with odd size. An example with a kernel size of 3

Edge Detection: Sobel

$$magn(\nabla f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} = \sqrt{M_x^{\,2} + M_y^{\,2}}$$







$$magn(\nabla f) \approx |M_x| + |M_y|$$

Approximation to save time of computations

A Detailled Sobel Algorithm

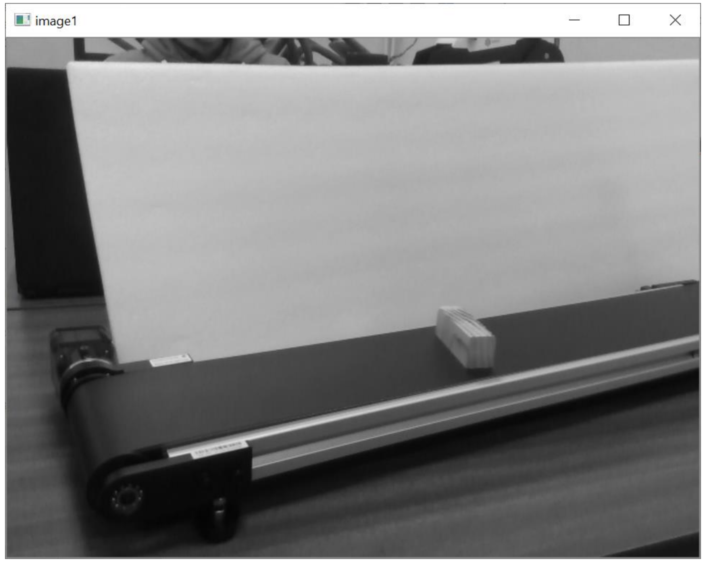
```
(height,width) = img.shape
im = np.zeros((height,width),np.uint8)

kernelY = np.array([-1,-2,-1,0,0,0,1,2,1],'int')
kernelY = kernelY.reshape(3,3)
kernelX = np.array([-1,0,1,-2,0,2,-1,0,1],'int')
kernelX = kernelX.reshape(3,3)
print("KernelX")
print(kernelX)
print("KernelY")
print(kernelY)
for i in range(1,height-1):
    for j in range(1,width-1):
        Gx = 0
        Gy = 0
        for k in range(-1,2):
            for l in range(-1,2):
                Gx += img[i+k,j+l]*kernelX[k+1,l+1]
                Gy += img[i+k,j+l]*kernelY[k+1,l+1]
        result = abs(Gx) + abs(Gy)
        if result > 255:
            result = 255
        elif result < 0:
            result = 0
        im[i,j] = result
```

- img: input image
- im: output image
- all the image are in grayscale

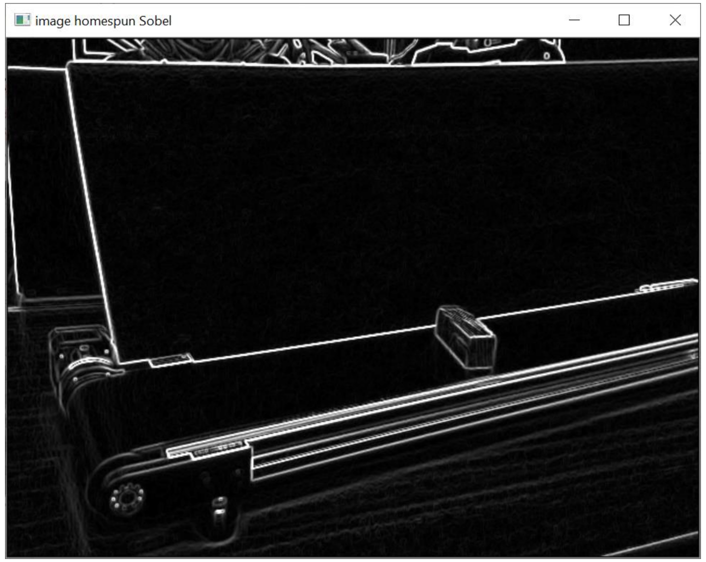
```
-1 -2 -1
0 0 0
1 2 1
```

```
-1 0 1
-2 0 2
-1 0 1
```

Example of Edge Detection with Sobel

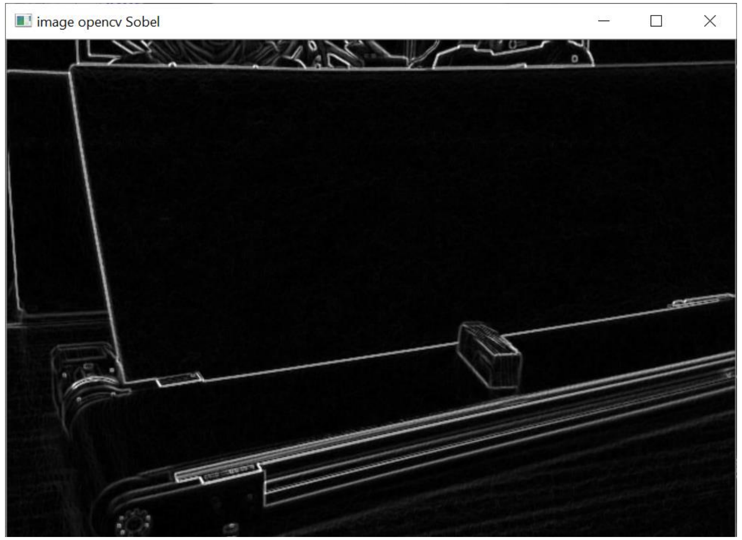


Grayscale Image

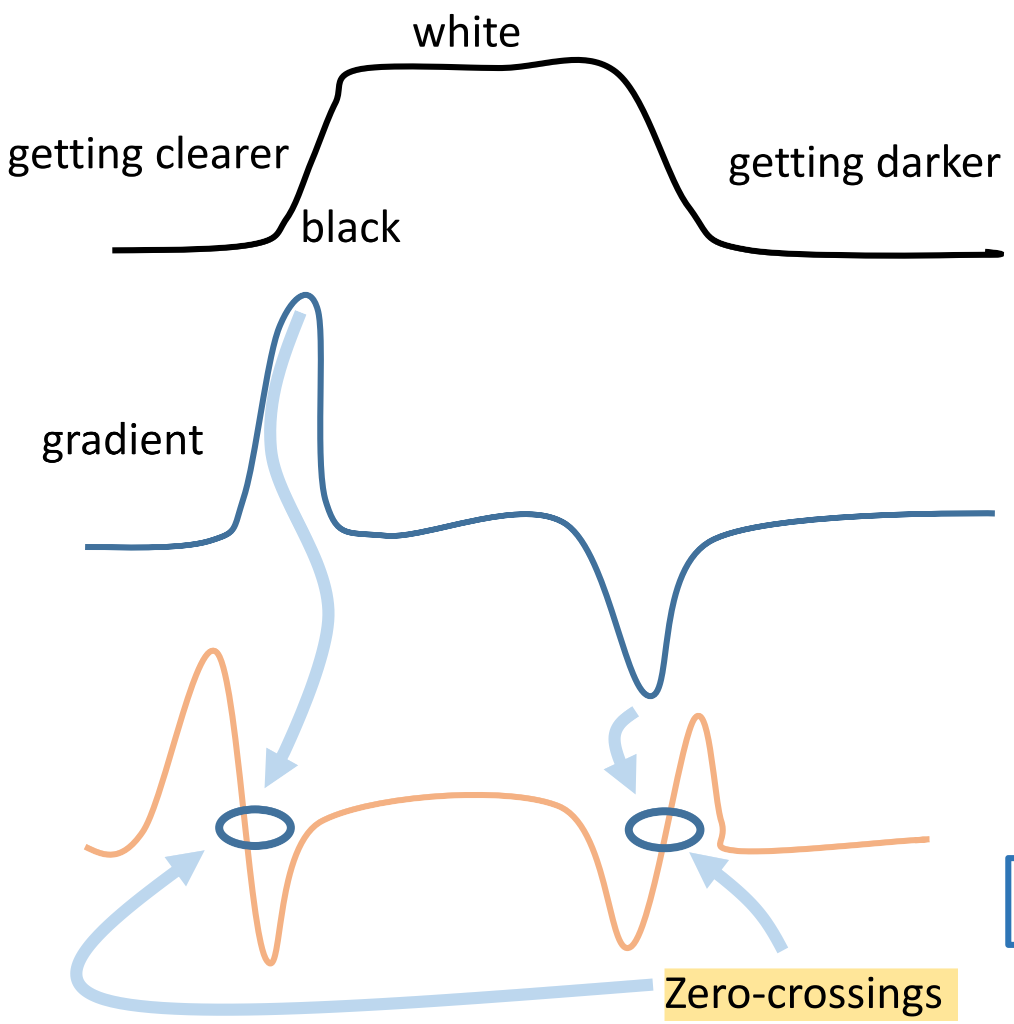


Sobel function developed from the algorithm

Sobel function from OpenCv library



Edge detection using the second derivative



There are two operators in 2D that correspond to the second derivative: Laplacian and Second directional derivative

The Laplacian operator $\nabla^2 f = \dfrac{\partial^2 f}{\partial x^2} + \dfrac{\partial^2 f}{\partial y^2}$

Approximating the Laplacian operator

$$\frac{\partial^2 f}{\partial x^2} = f(i, j+1) - 2f(i, j) + f(i, j-1)$$

$$\frac{\partial^2 f}{\partial y^2} = f(i+1, j) - 2f(i, j) + f(i-1, j)$$

$$\boxed{\nabla^2 f = -4f(i, j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j)}$$

Implementing the Laplacian operateur

$$\boxed{\nabla^2 f = -4f(i,j) + f(i,j+1) + f(i,j-1) + f(i+1,j) + f(i-1,j)}$$

Using The Laplacian operator matches the convolution with this Kernel below

0	1	0
1	-4	1
0	1	0

0	-1	0
-1	4	-1
0	-1	0

Other variants of Laplacian can be obtained by weighing the pixels in the diagonal directions also. The sum of all kernel elements must be zero so that the response in the homogeneous regions.

-1	-1	-1
-1	8	-1
-1	-1	-1

isotropic operator.
cheaper to implement (one mask only).
not information about edge direction.
more sensitive to noise (differentiates twice).

A filter can be needed(Gausian filter)

Laplacian: an example

Exercises: filters

- To begin, implement the Sobel and Laplacian filter to carry out the edge detection and the Median Filter 5x5
- We want to increase the contrast on a picture. For that, you are asked to implement the following filter

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

- implement the filter below and comment the result compared with the original

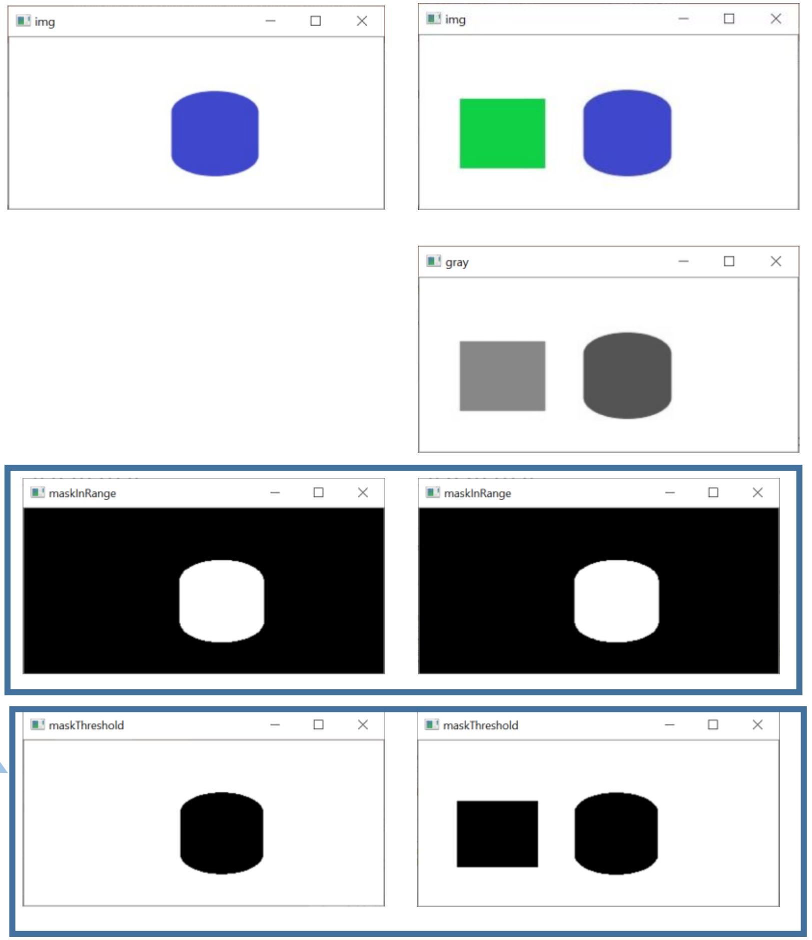
$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Thresholding

```
import cv2
import numpy as np


# loading a colour image
img = cv2.imread('blue_green.jpg',cv2.IMREAD_COLOR)

# changing colour spaces -> HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# define range of blue color in HSV ---
lower_blue = np.array([110,50,50])
upper_blue = np.array([130,255,255])

# Threshold the HSV image to get only blue colors
maskInRange = cv2.inRange(hsv, lower_blue, upper_blue)

# changing colour spaces -> GRAYSCALE
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Threshold the Gray image: discriminate
#the grayscales with a threshold = 150
ret,maskmaskThreshold = cv2.threshold(gray,150,255,cv2.THRESH_BINARY)

cv2.imshow('gray',gray)

cv2.imshow('img',img)
cv2.imshow('maskInRange',maskInRange)
cv2.imshow('maskThreshold',maskmaskThreshold)
```

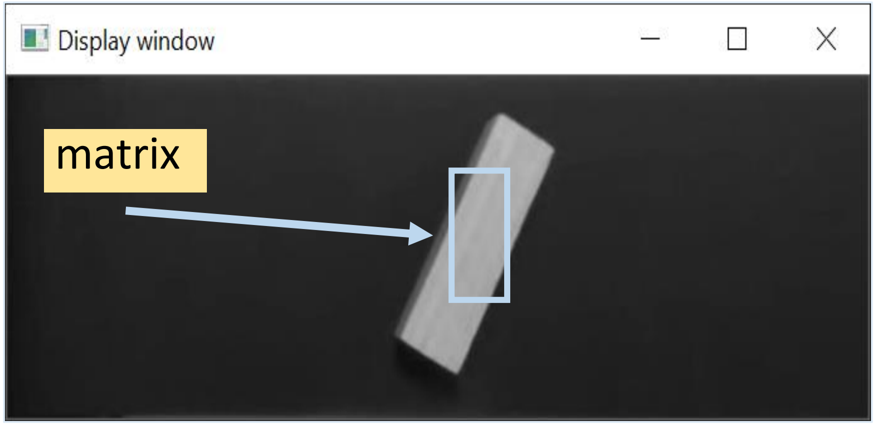
Defining the range of the blue inHSV

Defining the threshold



Detecting objects and computing the inertia centre

The change of the grayscale along the median line of the conveyor belt can be noticed; this change may be a good way to detect the wooden part. A matrix (nxm) must extracted from the image and the programme must calculate the average of the term of this matrix(it matches a convolution and the matrix is made up of term equal to 1). The centre of the matrix must be applied at the centre of the belt.



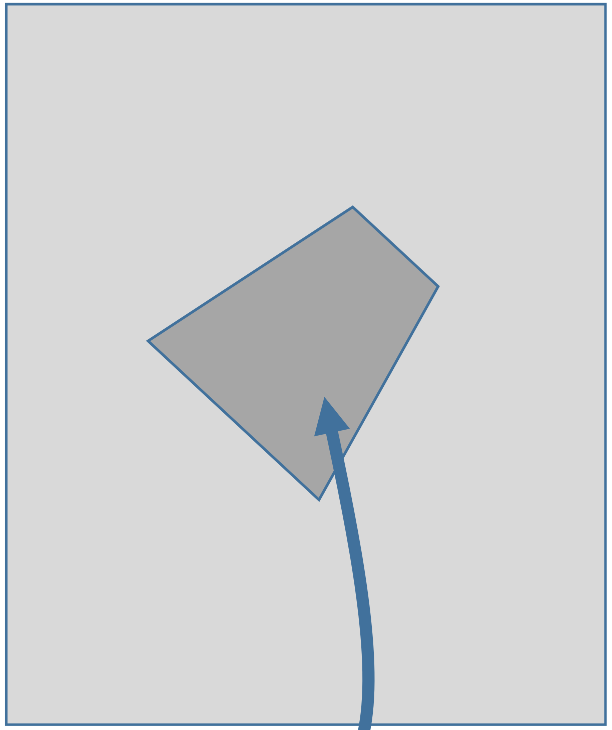
Method of moments (geometrical):surface area

Each pixel of an grayscale image is characterized by the function I(x,y). The function gives the graduation of gay of the pixel.
The p,q order moment of an image is given by:

$$M_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

Let an image with an uniform background including a shape. The background of the shape is different from that of the image. We assign 0 to the background colour of the image, 1 to that of the shape.
So the surface area is given by:

$$\text{Surface area} = M_{00} = \sum_x \sum_y x^p y^q I(x,y) = \sum_x \sum_y I(x,y)$$



Computing the surface area amounts to count the number of dark gray pixels

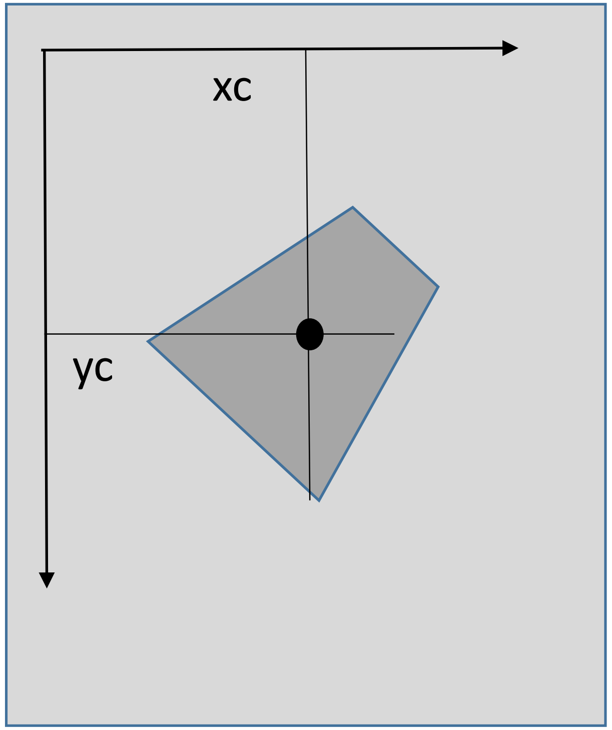
Method of moments: inertia centre

Let's take up the same image. We assign 0 to the background colour of the image, 1 to that of the shape.
The moment of the shape according to y axis is given by:

$$M_{10} = \sum_x \sum_y x^1 I(x,y)$$

The moment of the shape according to x axis is given by:

$$M_{01} = \sum_x \sum_y y^1 I(x,y)$$



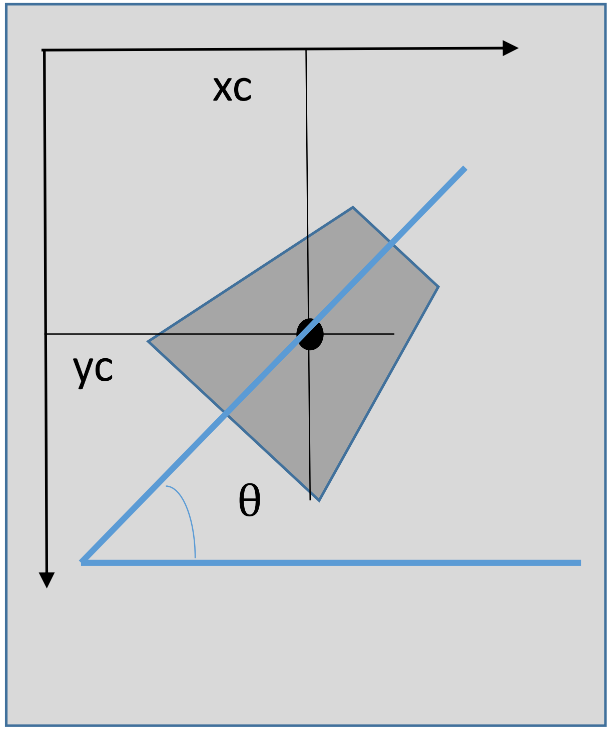
So coordinates of the inertia centre is given by:

$$x_c = \frac{M_{10}}{M_{00}} \qquad y_c = \frac{M_{01}}{M_{00}}$$

Method of moments: orientation

The main direction is given by:

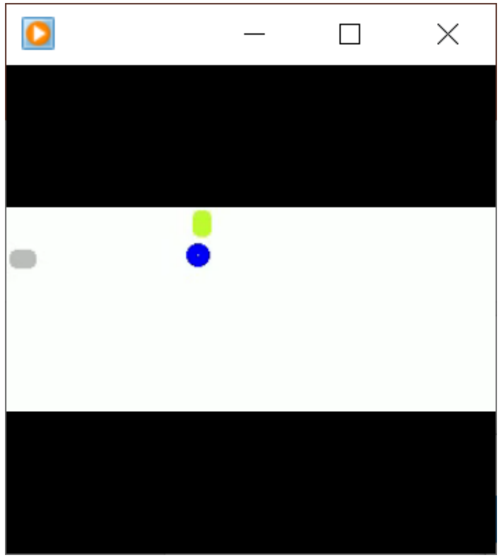
$$\theta = \frac{1}{2} artan(\frac{2M_{11}}{M_{10} - M_{01}})$$



Exercise: tracking object

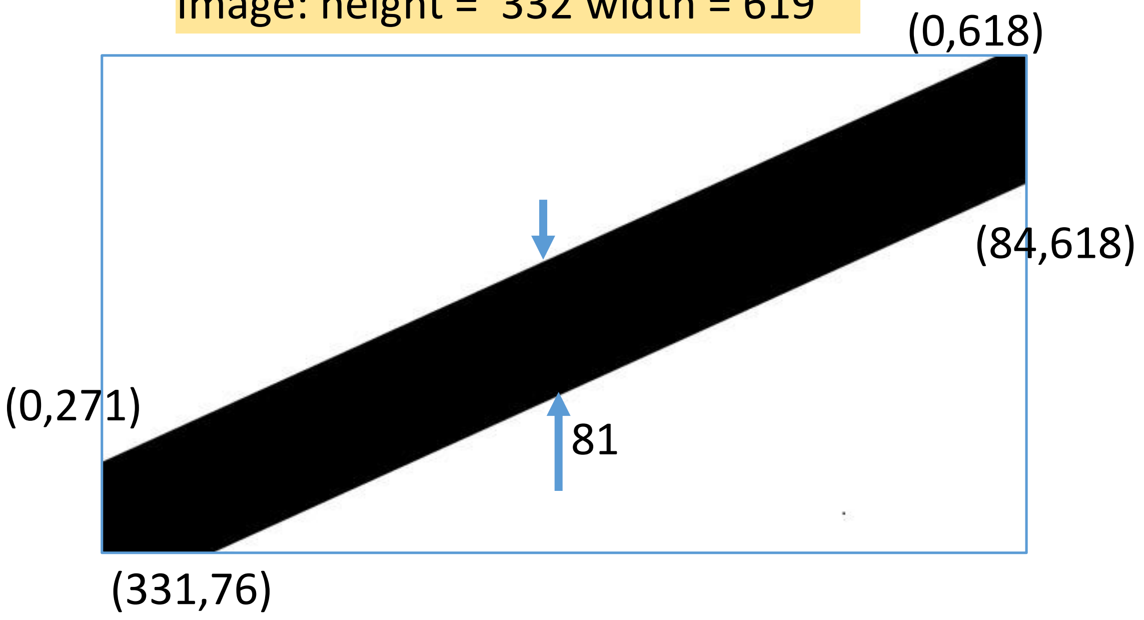
You are asked to:

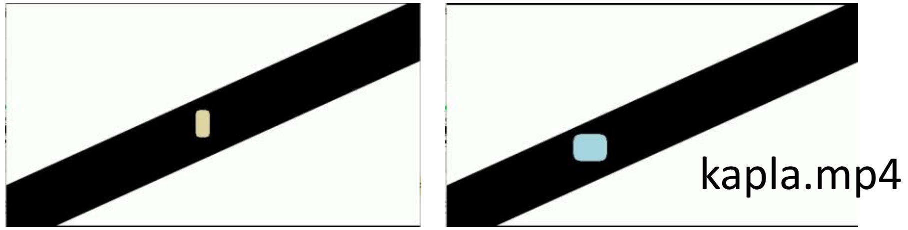
- Download the file « test.mp4 » on moodle.
- Write a programme which reads this file and displays the frame. Execute it.
- Modify the previous programme to apply a mask which allows us to pick out the blue object. Execute it.
- Modify the previous programme which calculates the inertia centre of this object. Execute it.
- Modify the previous programme which traces the trajectory of the object. Execute it.
- Take up the programme but instead of the direct calculation of the inertia centre, use the function «SimpleBlobDetector » which provides the inertia centre.



Exercise: detecting objects(1)

Image: height = 332 width = 619



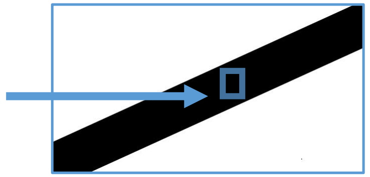


Conveyor belt

You are asked to:

- Download the file « kapla.mp4 » from moodle
- Write a programme which reads this file and displays the frame. Execute it.
- Modify the previous programme so that it detects an object passing along the conveyor belt. The change of the grayscale could be an indicator. Execute it
- Modify the previous programme so that it uses a matrix. A matrix (10x10) must extracted from the image and the programme must calculate the average of the term of this matrix(it matches a convolution and the matrix is made up of term equal to 1). Execute it.

matrix

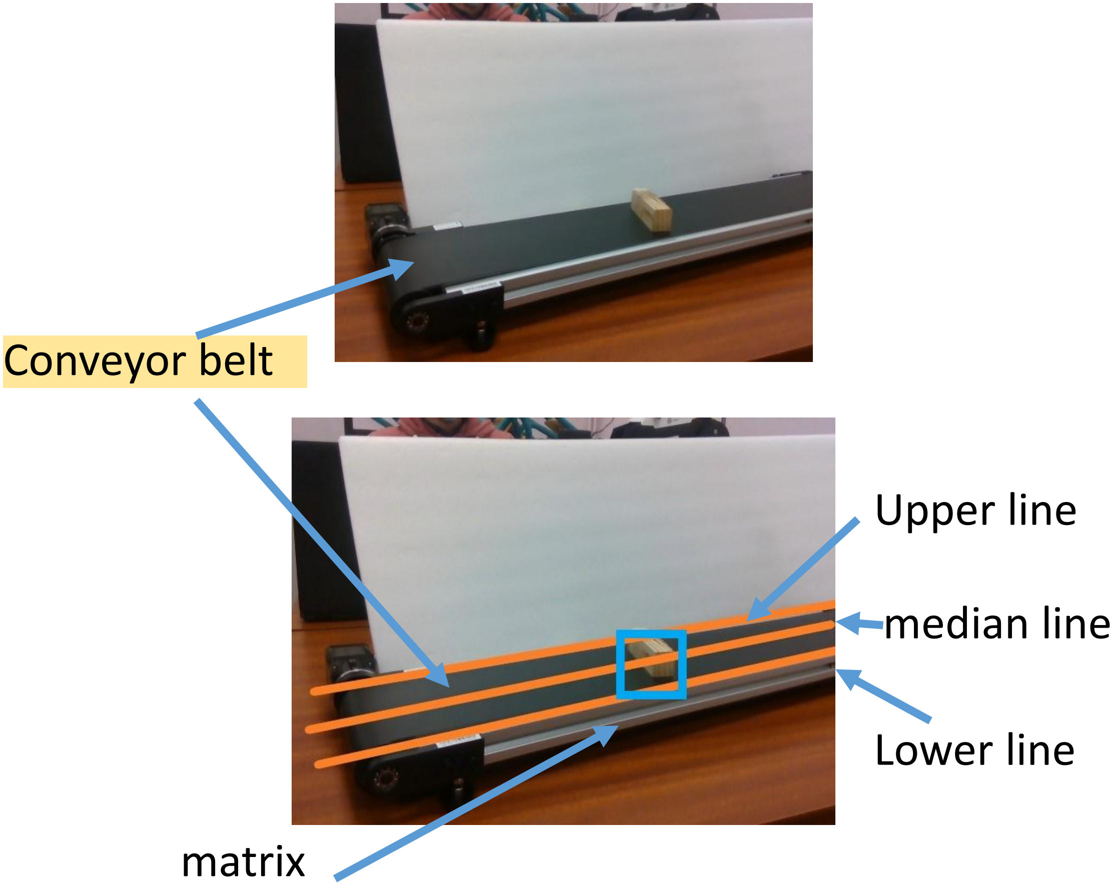


$$\frac{1}{100} \times$$

1	1	..	..	1
1		..	..	1
..	..	..	..	..
..	..	..	..	..
1		..	..	1

- Modify the previous programme so that it counts the objects. Execute it.

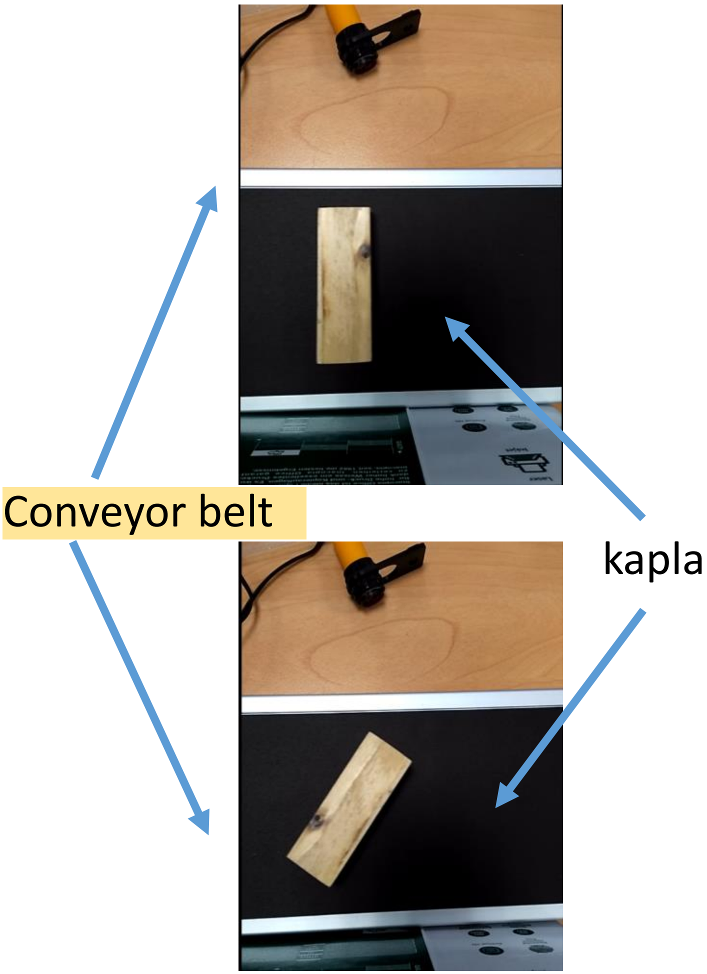
Exercise: detecting objects(2)



You are asked to:

- Download the file « kapla.jpg » from moodle
- Write a programme which reads this file and displays the frame. Execute it.
- Modify the previous programme so that it applies a matrix with a relevant dimension along the median line(convolution). The result will be stored in an array and displayed at the end of the programme. Execute it
- Determine a value which allows an efficient detection of the object. For that you must analyse the features of the results obtained previously.
- Implement this value to detect the object in the programme and execute it.

Exercise: detecting and tracking objects



You are asked to:

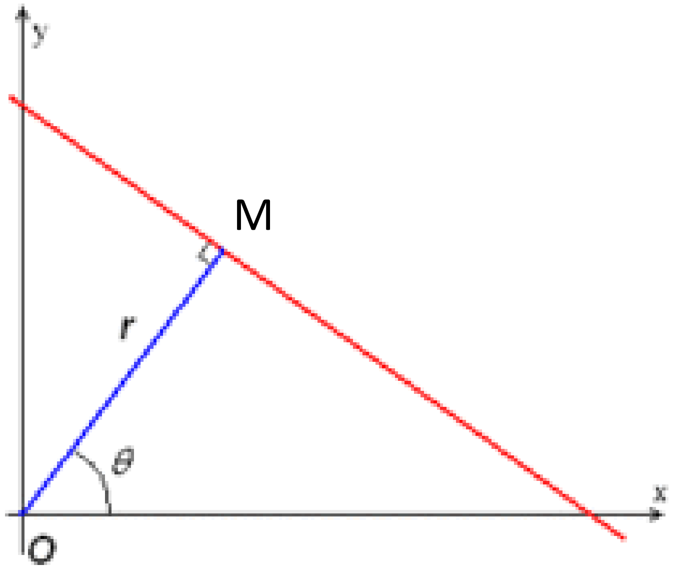
- Download the files “conveyorKapla1.mp4”, ”opencv_frame_1.jpg” from moodle. The last file is intended to make cropping measurements.
- Write a programme which reads this file and displays the frame. Execute it.
- Modify the previous programme so that it extracts the part of the image regarding the conveyor belt (cropping) and it applies a matrix with a relevant dimension in the middle of the conveyor. The result will be stored in an array and displayed at the end of the programme. Execute it

Now the programme must detect the object along the conveyor belt from the features of the results obtained previously.

- Implement this value to detect the object in the programme and locate its inertia centre.

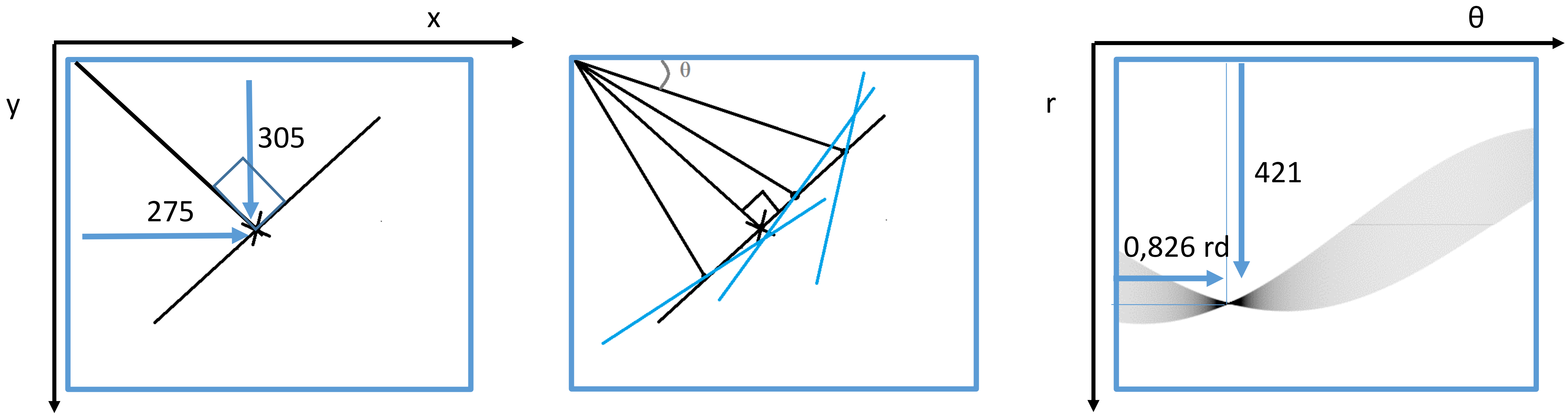
You can carry out this experiment again with “conveyorKapla2.mp4”, ”opencv_frame_2.jpg”, “conveyorKapla3.mp4”, ”opencv_frame_3.jpg”, “conveyorKapla4.mp4”, ”opencv_frame_4.jpg”

Detecting straight lines: Hough transform



Where M is the closest point on the straight line from the origin and θ is the angle between the x axis and the line connecting the origin with that closest point. The length r $= \overrightarrow{OM}$ which the Euclidean norm is the minimum distance between the origin and the straight line.

Hough Transform: the principle



The Hough algorithm makes a point on the transformed image darker if a lot of points on the original image lie on the corresponding line

a line can be *detected* by finding the number of intersections between curves. The more curves intersecting means that the line represented by that intersection have more points

x = r .cos(θ) and y = r .sin(θ)

- x = 416 .cos(0.826) = 282 and y 421.sin(0.826) = 305

Hough transform applied to straight line: algorithm

```python
def hough(im, ntx=460, mry=360):
    "Calculate Hough transform."
    nimy, nimx = im.shape # the length and the width of the image
    print("shape = ",im.shape)
    mry = int(mry/2)*2          #Make sure that this is even
    him = 255 * np.ones((mry,ntx )) # building an image, result of the hough transform
    rmax = hypot(nimx, nimy)
    print("rmax = ", rmax) # The math.hypot() method returns the Euclidean norm.
    # The Euclidian norm is the distance from the origin to the coordinates given.
    dr = rmax / (mry/2)
    dth = pi / ntx
    print("dr and dth =",dr, dth)
    for jx in range(nimx):
        for iy in range(nimy):
            col = im[iy, jx]
            if col == 255: continue # the brightest part of the image is not taken into account
            # the treatement is done on the black point the original
            for jtx in range(ntx):
                th = dth * jtx
                r = jx*cos(th) + iy*sin(th)
                iry = int(mry/2 + int(r/dr+0.5))
                him[iry, jtx] -= 1 # darker and darker drawing near zero
    return him
```

Using the OpenCv Hough transform functions

- **The Standard Hough Transform**

The standard Hough transform function consists in the algorithm given in the previous slides. It gives you as result a vector of couples (θ,r) we have explained in the previous slides. it is implemented with the function HoughLines()

- **The Probabilistic Hough Line Transform**

A more efficient implementation of the Hough Line Transform. Indeed it doesn’t take all the points into account, only a random subset of points(is enough for line detection) decreasing the computing . It gives as output the extremes of the detected lines (x0,y0,x1,y1). It is implemented with the function HoughLinesP()

The standard hough transform function

```
# loading a colour image
img = cv2.imread('opencv_frame_3.jpg',cv2.IMREAD_GRAYSCALE)
height,width = img.shape
gray = img[100:(height-1),560:940]
ret,BinaryImage = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
edges = cv2.Canny(BinaryImage,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,80)

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)))
        cv2.line(gray, pt1, pt2,255, 2)
```

Getting a binary image by setting a threshold value

Edge Detection

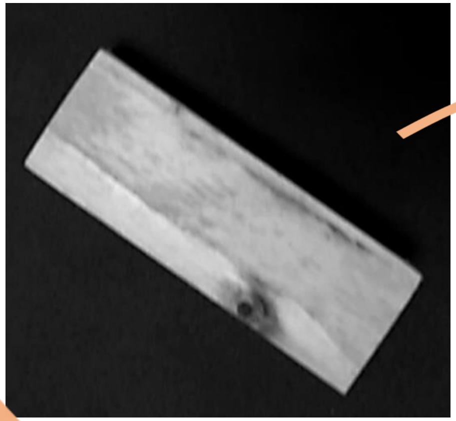
The resolution of r in pixels

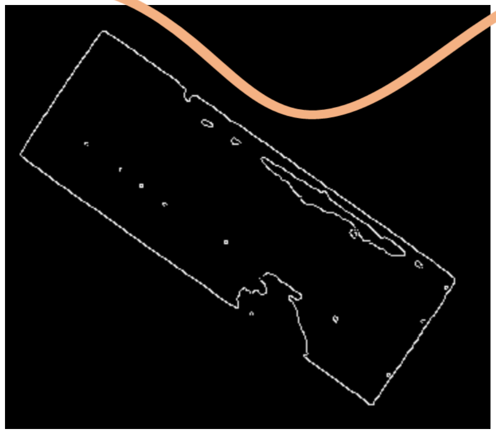
The resolution of θ in pixels (1° here)

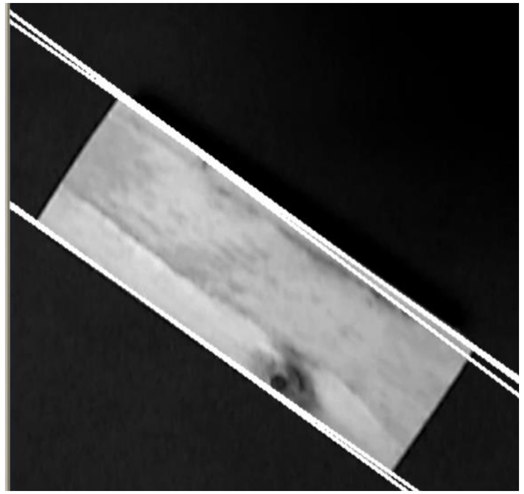
The minimum number of intersections to detect a line

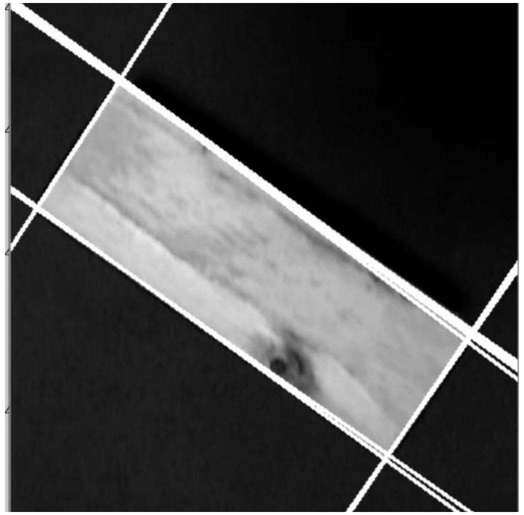
Hough Transform: the result of the previous code

```
lines = cv2.HoughLines(edges,1,np.pi/180,80)
```









with the minimum number of intersections to detect a line going down, the number of candidate lines increases

```
lines = cv2.HoughLines(edges,1,np.pi/180,70)
```

The Probabilistic Hough transform function

```
img = cv2.imread('opencv_frame_3.jpg',cv2.IMREAD_GRAYSCALE)
ret,BinaryImage = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
edges = cv2.Canny(BinaryImage,50,150,apertureSize = 3)

lines = cv2.HoughLinesP(edges,1, np.pi/180, 70, minLineLength=10, maxLineGap=250)

# Draw lines on the original image

for line in lines:
    x1, y1, x2, y2 = line[0]

    cv2.line(gray,(x1,y1),(x2,y2),255,2)
    cv2.circle(gray, (x1,y1), 4,255, 5)
    cv2.circle(gray, (x2,y2), 4,255, 5)
```

The resolution of θ in pixels (1° here)

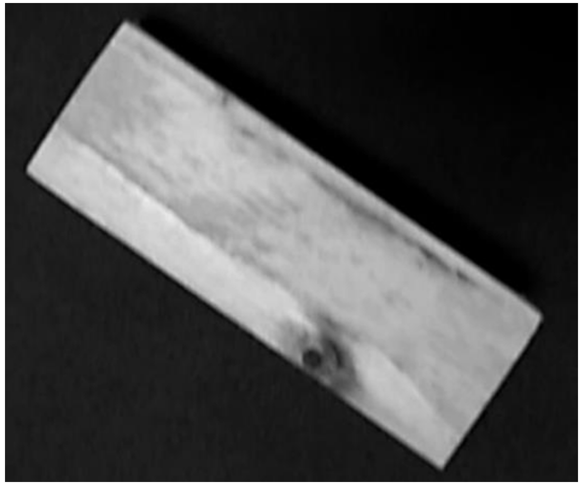
The resolution of r in pixels

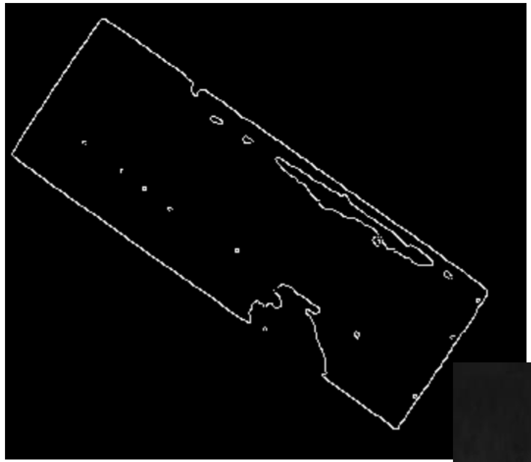
The minimum number of points that can form a line

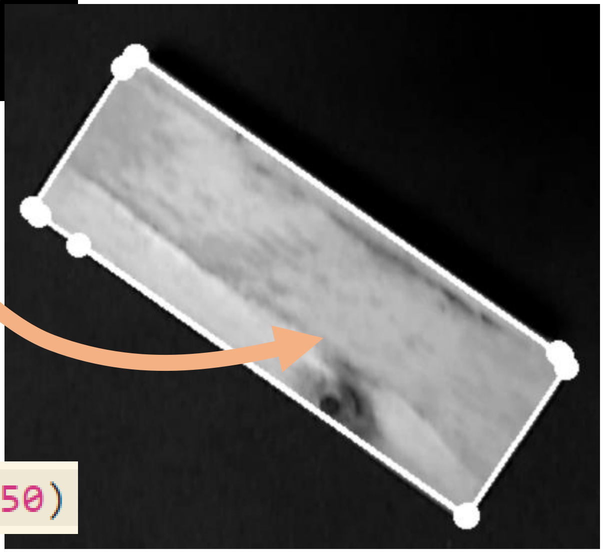
The minimum number of intersections to detect a line

The maximum distance between two points to be considered in the same line

Hough Transform: the result of the previous code







```
lines = cv2.HoughLinesP(edges,1, np.pi/180, 70, minLineLength=10, maxLineGap=250)
```

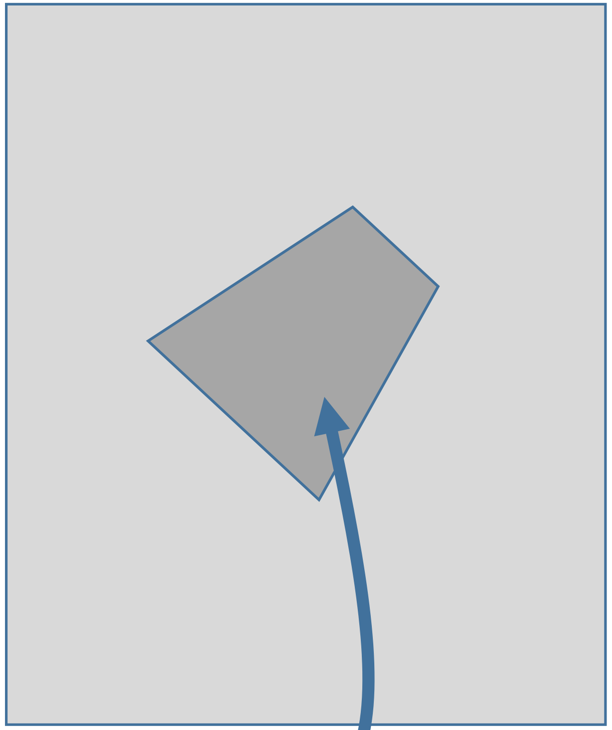
Method of moments (geometrical):surface area

Each pixel of an grayscale image is characterized by the function I(x,y). The function gives the graduation of gay of the pixel.
The p,q order moment of an image is given by:

$$M_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

Let an image with an uniform background including a shape. The background of the shape is different from that of the image. We assign 0 to the background colour of the image, 1 to that of the shape.
So the surface area is given by:

$$\text{Surface area} = M_{00} = \sum_x \sum_y x^p y^q I(x,y) = \sum_x \sum_y I(x,y)$$



Computing the surface area amounts to count the number of dark gray pixels

Method of moments: inertia centre

Let's take up the same image. We assign 0 to the background colour of the image, 1 to that of the shape.
The moment of the shape according to y axis is given by:

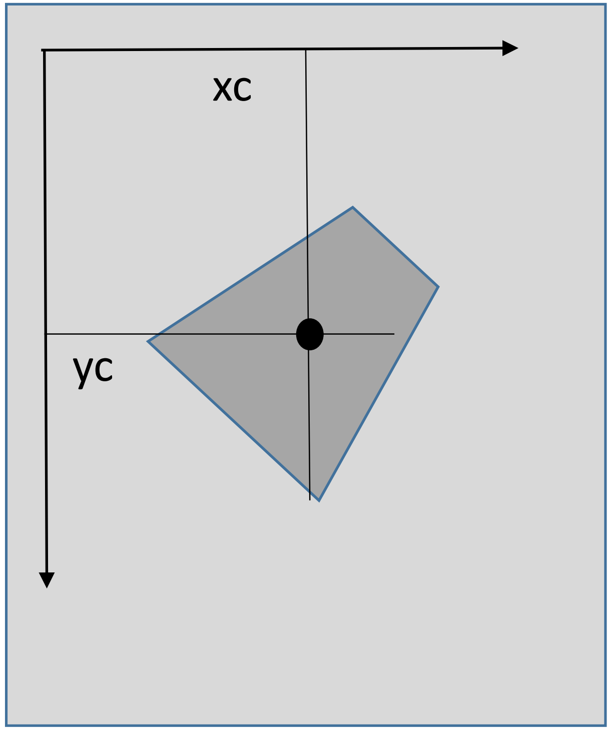
$$M_{10} = \sum_x \sum_y x^p I(x,y)$$

The moment of the shape according to x axis is given by:

$$M_{01} = \sum_x \sum_y y^q I(x,y)$$

So coordinates of the inertia centre is given by:

$$x_c = \frac{M_{10}}{M_{00}} \qquad y_c = \frac{M_{01}}{M_{00}}$$

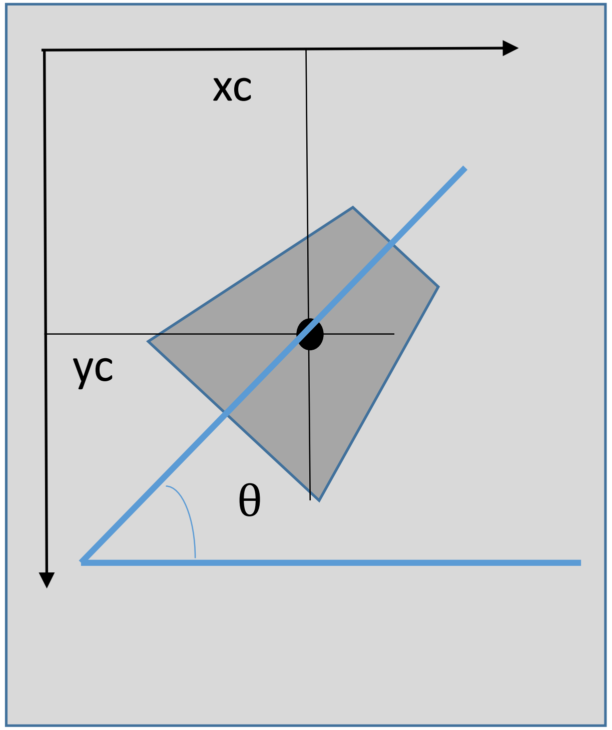


Method of moments: orientation

The main direction is given by:

```
mup20 = M20 / M00 - Xc * Xc
mup02 = M02 / M00 - Yc * Yc
mup11 = M11 / M00 - Xc * Yc
```

$$\theta = \frac{1}{2} artan\left(\frac{2mup_{11}}{mup_{20} - mup_{02}}\right)$$



Calculating Moment with OpenCv function

```python
# convert the colour image to the grayscale image
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# convert the grayscale image to binary image
ret,thresh = cv2.threshold(gray,127,255,0)
# calculate moments of binary image
M = cv2.moments(thresh)
```

dictionary

keys

values

{'m00': 5865.0,'m10': 2629050.0, 'm01': 677790.0, 'm20': 1178504430.0, 'm11': 303822810.0, 'm02': 78372210.0, 'm30': 528281174400.0, 'm21': 136190606040.0, 'm12': 35130300420.0, 'm03': 9067058460.0, 'mu20': 4190.869565263765, 'mu11': -3924.782608685601, 'mu02': 43261.3043478286, 'mu30': -3383.93195747421, 'mu21': 4043.364834677445, 'mu12': 12552.36294786982, 'mu03': -42030.170132483516, 'nu20': 0.00012183389012823704, 'nu11': -0.00011409840503937634, 'nu02': 0.001257660950465572, 'nu30': -1.284551677851666e-06, 'nu21': 1.5348745624389288e-06, 'nu12': 4.764918174573595e-06, 'nu03': -1.5954790534373253e-05}

Miscellaneous

- Matrix and image : operations/cropping
- Detecting simple shapes

Operations on the matrix: cropping

```python
import numpy as np

a = np.array(range(49)).reshape(7,7)
print(a)
b = a[1:5,2:4]
print(b)
print(b.size)
print(np.sum(b))
```

```
[[ 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]]
```

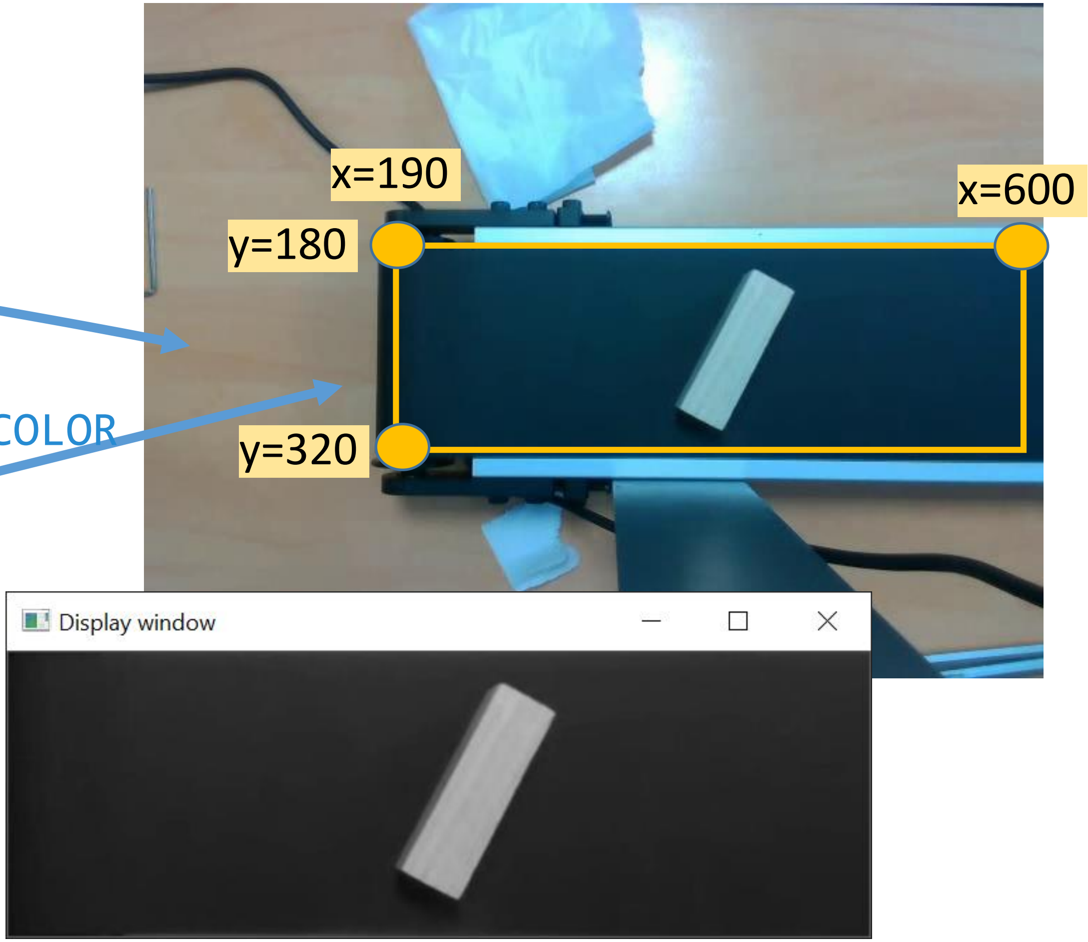
2:4

1:5

```
[[ 9 10]
 [16 17]
 [23 24]
 [30 31]]
8
160
```

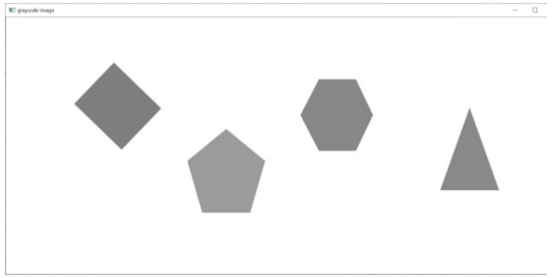
Image matrix: collecting a part of the matrix (cropping)

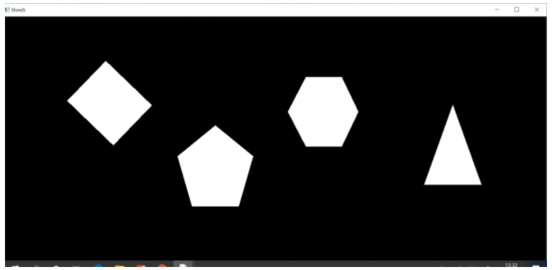
```
import numpy as np
import cv2

Img  =  cv2.imread("opencv_frame_3.jpg",cv2.IMREAD_COLOR
)

NvlleImage = Img[180:320,190:600]

cv2.imshow("cropped image", NvlleImage)

cv2.waitKey(0)
```

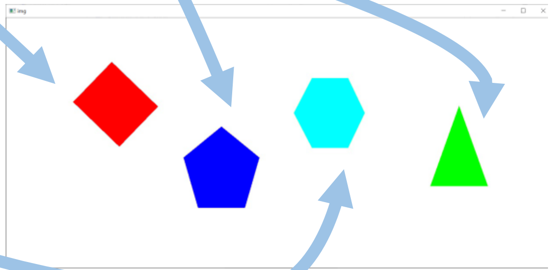


Detecting simple shapes(1)

```
import numpy as np
import cv2

img = cv2.imread('simpleShapes.jpg',cv2.IMREAD_COLOR)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow('grayscale image',gray)
ret,thresh = cv2.threshold(img,200,255,cv2.THRESH_BINARY_INV)
contours,h = cv2.findContours(thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
    if len(approx)==5:
        print ("pentagon")
        cv2.drawContours(img,[cnt],0,(255,0,0),-1)
    elif len(approx)==3:
        print ("triangle")
        cv2.drawContours(img,[cnt],0,(0,255,0),-1)
    elif len(approx)==4:
        print ("square")
        cv2.drawContours(img,[cnt],0,(0,0,255),-1)
    elif len(approx) == 6:
        print ("hexagone")
        cv2.drawContours(img,[cnt],0,(255,255,0),-1)
    elif len(approx) > 15:
        print ("circle")
        cv2.drawContours(img,[cnt],0,(0,255,255),-1)
cv2.imshow('img',img)
cv2.imshow('thresh',thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```







```
= RESTART: C:\Users\
ogrammes_and_images\
pentagon
triangle
hexagone
square
```

Detecting simple shapes(2)

```
import numpy as np
import cv2

img = cv2.imread('simpleShapes.jpg',cv2.IMREAD_COLOR)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow('grayscale image',gray)
ret,thresh = cv2.threshold(img,200,255,cv2.THRESH_BINARY_INV)
contours,h = cv2.findContours(thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
    if len(approx)==5:
        print ("pentagon")
        cv2.drawContours(img,[cnt],0,(255,0,0),-1)
    elif len(approx)==3:
        print ("triangle")
        cv2.drawContours(img,[cnt],0,(0,255,0),-1)
    elif len(approx)==4:
        print ("square")
        cv2.drawContours(img,[cnt],0,(0,0,255),-1)
    elif len(approx) == 6:
        print ("hexagone")
        cv2.drawContours(img,[cnt],0,(255,255,0),-1)
    elif len(approx) > 15:
        print ("circle")
        cv2.drawContours(img,[cnt],0,(0,255,255),-1)
cv2.imshow('img',img)
cv2.imshow('thresh',thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

With CHAIN_APPROX_NONE, all the boundary points are stored. The contours are made up of straight lines. CHAIN_APPROX_SIMPLE removes all redundant points and leaves the extreme points.

CV_RETR_LIST gives all the contours without the hierarchy. The fact that one shape is nested inside another does not matter. CV_RETR_TREE calculates the full hierarchy of the contours.

A screenshot from a film with VLC