UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTE OF COMPUTING

INTRODUCTION TO DIGITAL IMAGE PROCESSING
MC920 / MO443

HOMEWORK 3

STUDENT: Gian Franco Joel Condori Luna
RA: 234826

1.- Problem specification:

The objective of this work is to obtain some measurements of objects present in digital images. The main steps are described in the sections to follow.

1.1.- Color Transformation

Once you have uploaded the image, the first thing you have to do is convert to grayscale with OpenCV. When we work with color images, the computational cost grows exponentially.

For example, if you have three color components as in RGB space (R for Red, G for Green and B for Blue) it is as if you were working with 3 different images, one for each component.

In OpenCV there is a method that allows us to switch between color spaces.[1]

```
converted_image = cv2.cvtColor(img, conversion_type)
```

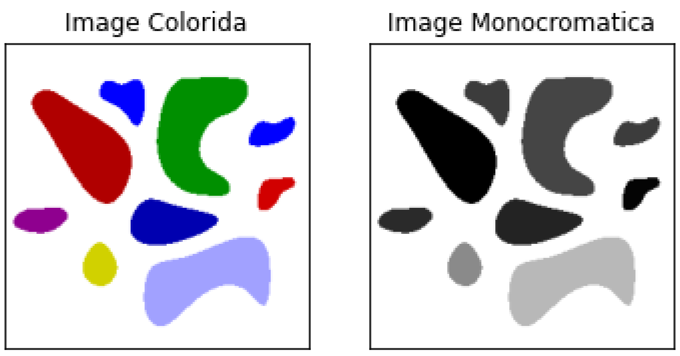
Where:

- converted_image: It is the resulting image in the new color space.
- img: It is the original image. We have to know what color space it uses.
- conversion_type: It is a constant that indicates from which space to which space we are going to convert.

The last parameter, conversion_type, allows us to convert between various color spaces. If what we want is to convert between RGB (in OpenCV it is named BGR for historical reasons) to grayscale, the conversion_type is cv2.COLOR_BGR2GRAY.

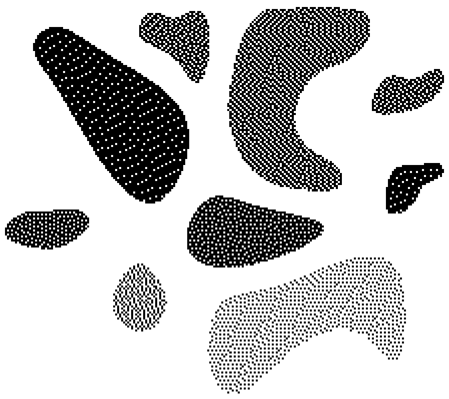
In the end our code is as follows:

```
img_monocromatica = cv2.cvtColor(img_colorida, cv2.COLOR_BGR2GRAY)
```



Additionally, it was also tried to convert the image through the PILLOW library[2] of python, with the following code:

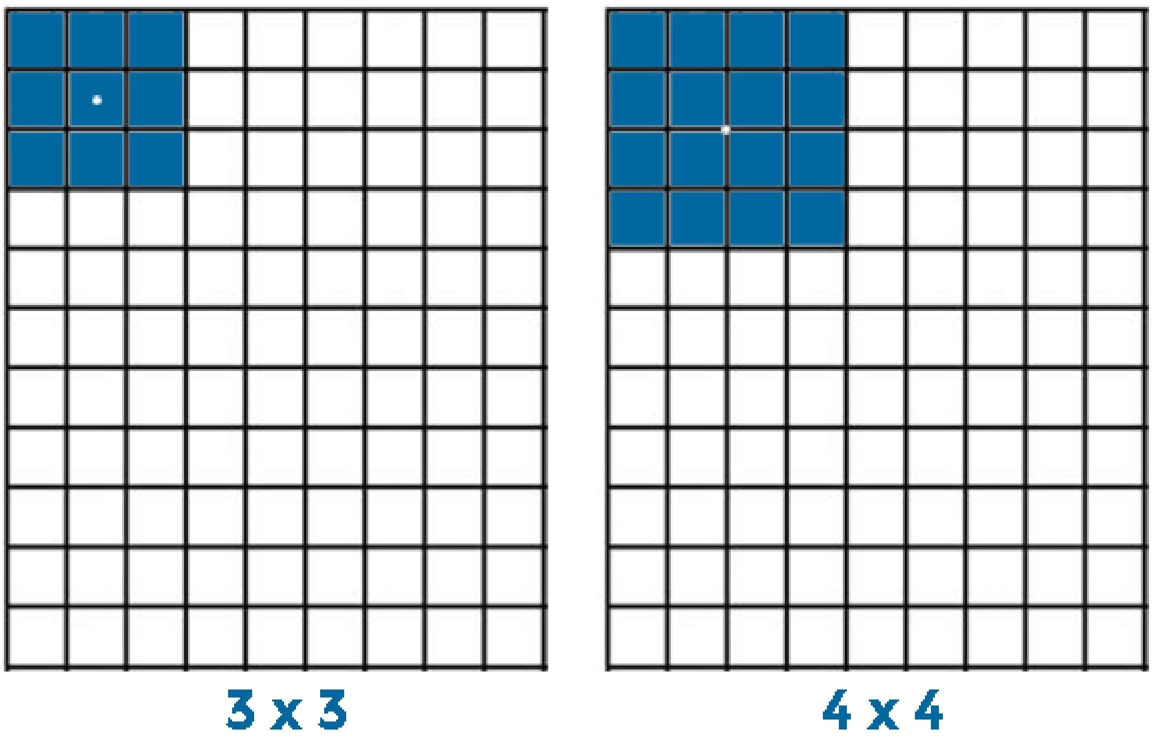
```
from PIL import Image
image_file = Image.open("/content/drive/My
Drive/imagenes/imagens_objetos_coloridos/objetos3.png") # open colour image
image_file= image_file.convert('L') # convert image to monochrome - this
works
image_file= image_file.convert('1') # convert image to black and white
image_file
```



1.2.- Outlines of Objects

Digital images are not perfect. The inherent noise of the devices themselves or counterproductive lighting effects alter reality.

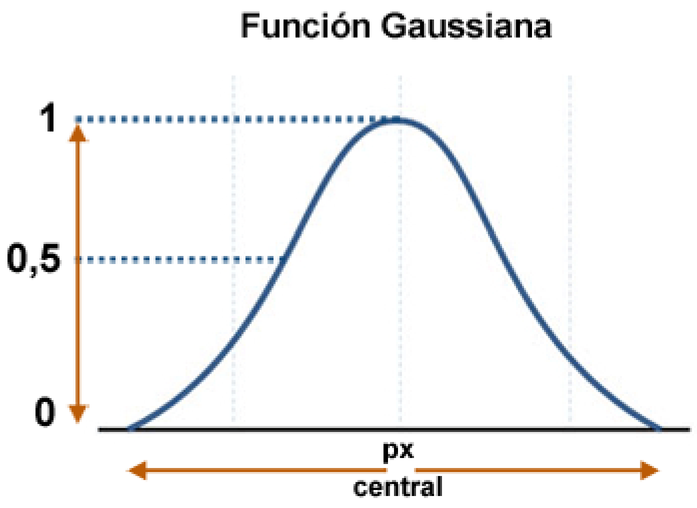
In digital image processing there are different methods to eliminate noise. They all use the same mathematical operation, the convolution. It consists of going through an image pixel by pixel with an N x N mask or kernel. This size determines the number of pixels with which we are going to work.



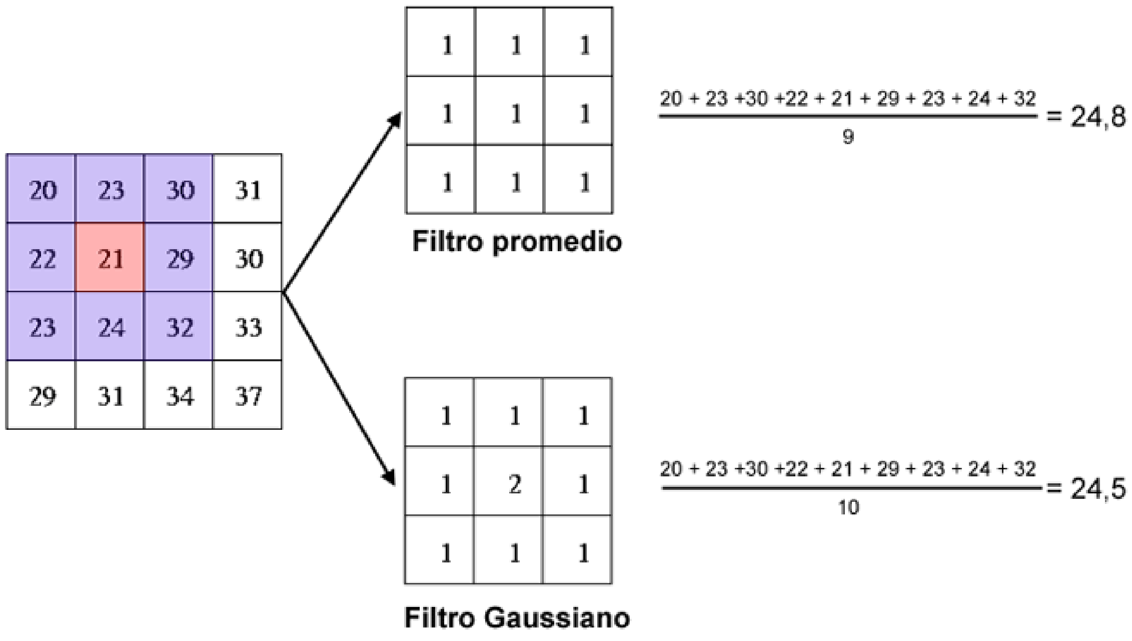
It is very important that the size is odd to always have a central pixel that will be the pixel in question that we are dealing with.

a) **Guassiano filter for the elimination of noise in images:**

Gaussian filtering is the same as averaging but weighted. This weighting is done following the Gaussian Bell. With this, it is possible to give more importance to the pixels that are closer to the center than those that are further away.



In order to apply it to an image we must do it in two dimensions. We do this through a convolution mask or kernel.



As you can see in the previous image, the central pixel is given more importance than the pixels around it.

In OpenCV we have a method that helps us to perform Gaussian filtering.[3]

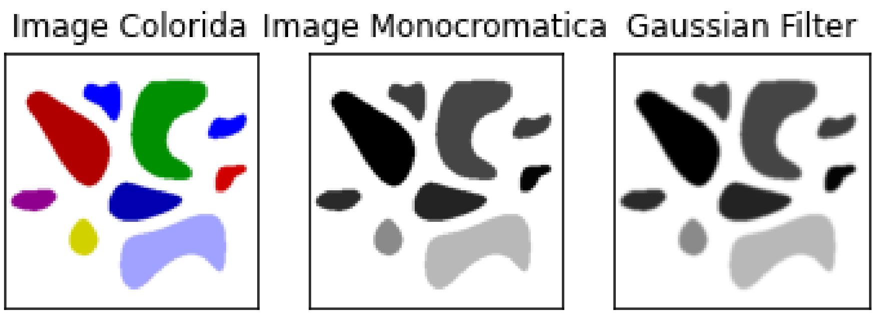
```
gaussian = cv2.GaussianBlur(image, (n, n), σ)
```

Where:

- gaussian: Is the resulting blurred image.
- image: It is the original image, the one we want to soften or blur.
- n x n: is the size of the kernel or convolution mask. Remember that it must be odd.
- σ: sigma (σ) represents the standard deviation on the X axis that is, the width of the Gaussian bell. If we put a 0, OpenCV will automatically take care of calculating that value for the kernel or mask that we have chosen. This is the advisable option.

If we apply the Gaussian filter with a 5 × 5 kernel the code would be as follows.

```
gaussian = cv2.GaussianBlur(img_monocromatica, (5,5), 0)
```



b) Canny edge detector

The process to detect edges with Canny is divided into 3 steps.

- Edge detection with Sobel
- Off-edge pixel suppression
- Apply hysteresis threshold

This whole process can be extremely difficult to implement yourself. Thanks to OpenCV all this is relatively simple and practical.[4]

The method for calculating the edges with the Canny method is as follows.

```
canny = cv2.Canny(image, threshold_min, threshold_max)
```

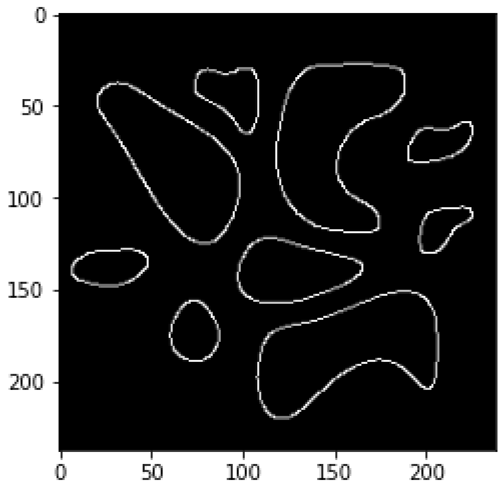
Where:

- canny: Is the resulting image. The detected edges will appear after the process.
- image: It is the original image.
- threshold_min: is the minimum threshold in hysteresis thresholding
- threshold_max: is the maximum threshold in hysteresis thresholding

As I have already mentioned, the minimum and maximum threshold will depend on each situation. You must experiment with the scene you are analyzing and choose the most suitable ones.

In this case, after several tests we decided to use 50 as the minimum threshold and 150 as the maximum threshold.

```
canny = cv2.Canny(gaussian, 50, 150)
```



1.3.- Object Properties Extraction

a) Find the outlines of an image

We have to know the difference between an edge and an outline. Edges, as we have seen previously, are pronounced intensity changes. However, a contour is a point curve without gaps or jumps, that is, it has a beginning and the end of the curve ends at that beginning.

The objective of this phase is to analyze all the detected edges and check if they are contours or not. In OpenCV we can do it with the following method.[5]

```
(contours,_) = cv2.findContours(binarized_image, countour_mode,
approximation_method)
```

Where:

- Result: We get 3 values as results, the interesting one for us is contours.
  - contours - This is a Python list of all the contours it has found. Then we will see how to draw these outlines on an image.
- binarized_image: It is the image where we have detected the edges or thresholding. Be careful, this method modifies this image so it is convenient that it be a copy. As a general rule, the binarized image must have a black background and the objects to be searched must be white.
- countour_mode: It is an internal parameter of the algorithm that indicates the type of contour we want. It can take different values RETR_EXTERNAL, RETR_LIST, RETR_COMP and RETR_TREE. We will focus on the first one, RETR_EXTERNAL which gets the external outline of an object.
- approximation_method: this parameter indicates how we want to approximate the contour. Basically we tell it if we want to store all the contour points. Imagine that you have a straight line, why do you want to store all the points? two would do. Maybe now you don't make sense of it, but when many objects are analyzed in large images, you will have to be very careful with the processing time and with the memory. It can take two values

CHAIN_APPROX_NONE which takes all points and
CHAIN_APPROX_SIMPLE which eliminates all redundant points.

b) Draw the contours in an image with OpenCV

Finally we have to draw the contours that we have found in the image. We will do this through another OpenCV method.

```
cv2.drawContours (image, list_contours, num_contours, color_RGB,
thickness)
```

Where:

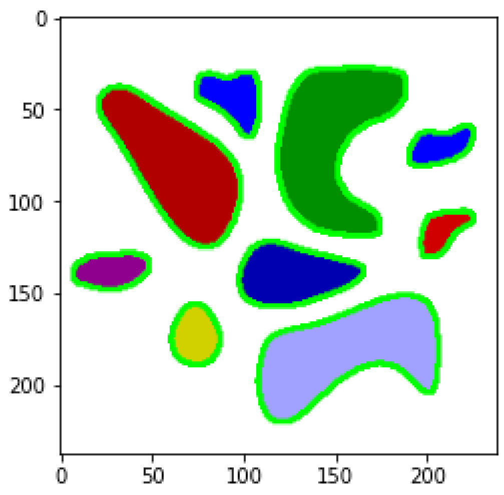
- image: is the image where we are going to draw the contours.
- list_contours: is the python list with the outlines.
- num_contours,: the number of contours we want to draw if they are all pass -1.
- color_RGB: is an array or tuple with the RGB color of the outline.
- thickness: thickness of the line to draw.

Now we are going to put all the pieces together and see what the complete code would look like.

```
(contours,_) = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

cv2.drawContours(img_colorida,contours,-1,(0,255,0), 2)

plt.imshow(img_colorida)
```



Now we have to put a number to each object and also find the area, perimeter, eccentricity and solidity [6]

a) Put a number to each object

```
#Finding central coordinates of each element
M = cv2.moments(c)
if(M["m00"]==0):M["m00"]=1
x = int(M["m10"]/M["m00"])
y = int(M["m01"]/M["m00"])
#Message
message = str(i+1)
cv2.putText(img, message, (x-5,y+5), font, 0.4, (0,0,0), 2, cv2.LINE_AA)
cv2.drawContours(img,[c],-1,(0,255,0), 2)
```

b) Area

```
#Area
area = cv2.contourArea(c)
```

c) Perimeter

```
#Perimeter
perimeter = cv2.arcLength(c,True)
```

d) Eccentricity

```
#Eccentricity
ellipse = cv2.fitEllipse(c)
#center, axis_length and orientation of ellipse
(center, axes, orientation) = ellipse
# length of MAJOR and minor axis
majoraxis_length = max(axes)
minoraxis_length = min(axes)
# eccentricity = sqrt( 1 - (ma/MA)^2) --- ma= minor axis --- MA= major axis
eccentricity = np.sqrt(1-(minoraxis_length/majoraxis_length)**2)
```

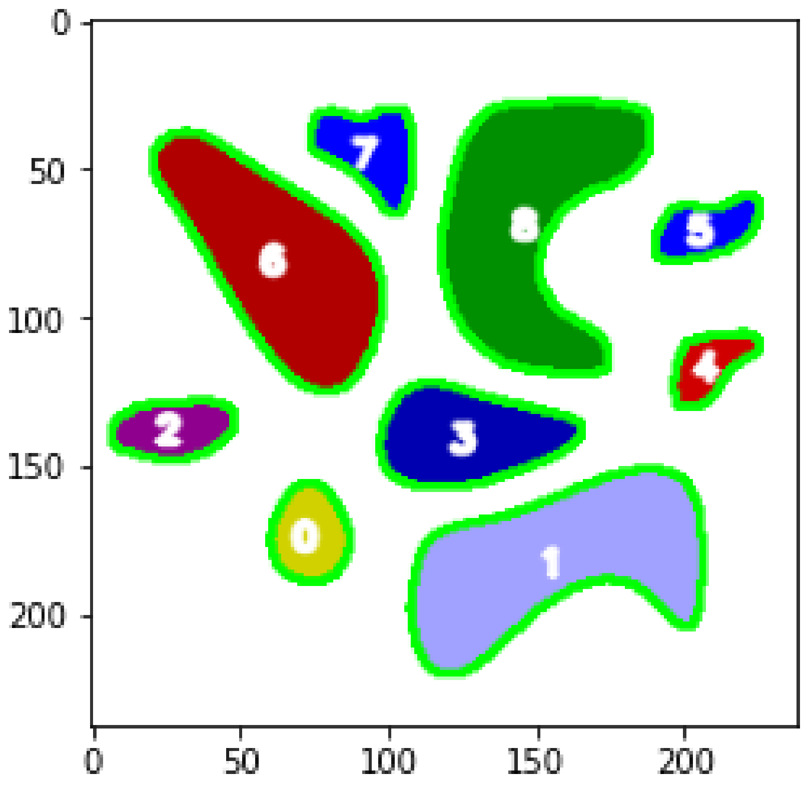
e) Solidity

```
# convex hull
convex_hull = cv2.convexHull(c)
# convex hull area
convex_area = cv2.contourArea(convex_hull)
# solidity = contour area / convex hull area
```

```
solidity = area/float(convex_area)
```

```
Número de regiones: 9
Region 0: area:674.5  perimeter:99.49747359752655
eccentricity:0.6312034041818577  solidity:0.9761215629522432
Region 1: area:3949.5  perimeter:308.8355675935745
eccentricity:0.8905062610081823  solidity:0.7833977982743231
Region 2: area:645.0  perimeter:105.5979790687561
eccentricity:0.8882099241852289  solidity:0.967741935483871
Region 3: area:1678.5  perimeter:177.2964632511139
eccentricity:0.8831362891669164  solidity:0.9705117085862967
Region 4: area:440.0  perimeter:91.59797894954681
eccentricity:0.8470928476756375  solidity:0.918580375782881
Region 5: area:539.0  perimeter:100.76955199241638
eccentricity:0.8820129333615935  solidity:0.922945205479452
Region 6: area:3611.5  perimeter:262.634556889534
eccentricity:0.9041831756591622  solidity:0.9770052752603814
Region 7: area:783.5  perimeter:122.32590091228485
eccentricity:0.7514617808196435  solidity:0.8969662278191185
Region 8: area:3993.0  perimeter:316.10764515399933
eccentricity:0.7413146192426724  solidity:0.7528280542986425

<matplotlib.image.AxesImage at 0x7f879a95c5d0>
```



1.4.- Area Histogram of Objects

a) Classify objects according to their area

```
areas = []
cant_pequeno = 0
cant_medio = 0
cant_grande = 0
for c in contours:
 #Area
 area = cv2.contourArea(c)
 areas.append(area)#accumulator

 if (area < 1500):
   cant_pequeno = cant_pequeno+1
 elif (area >= 1500 and area < 3000):
   cant_medio = cant_medio+1
 else :
   cant_grande = cant_grande+1

print("Numero de regiones pequenas:"+str(cant_pequeno))
print("Numero de regiones medianas:"+str(cant_medio))
print("Numero de regiones grandess:"+str(cant_grande))
```

Result:

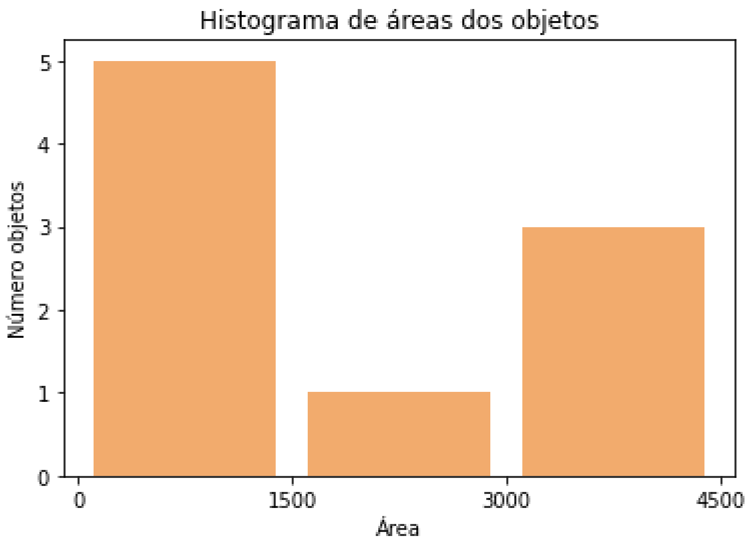
```
Numero de regiones pequenas:5
Numero de regiones medianas:1
Numero de regiones grandes:3
```

b) Histogram[7]

```
#Histogram
intervalos = [0, 1500, 3000, 4500] #calculamos los extremos de los
intervalos

plt.hist(x=areas, bins=intervalos, color='#F2AB6D', rwidth=0.85)
plt.title('Histograma de áreas dos objetos')
plt.xlabel('Área')
plt.ylabel('Número objetos')
plt.xticks(intervalos)

plt.show() #dibujamos el histograma
```


Histograma de áreas dos objetos
Número objetos
Área
0
1
2
3
4
5
0
1500
3000
4500

REFERENCES
1.- https://www.geeksforgeeks.org/python-opencv-cv2-cvtcolor-method/
2.- https://pillow.readthedocs.io/en/stable/
3.- https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/
4.- https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
5.- https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html
6.- http://opencvpython.blogspot.com/2012/04/contour-features.html
7.- https://www.codigopiton.com/como-hacer-un-histograma-en-python/