Intership report

FY 2020

Development of a scratch algorithmic process for end face testing

Developed by: Sonia Abbes

An engineering student at international institute of applied sciences and technologies

Supervisor: Ghazi Abbes

Product Development Engineer

Huber+Suhner

Fax : +216 73 830 903

Localisation :
Zone industrielle Sidi Abdelhamid
4061 Sousse

Web site :
https://www.hubersuhner.com/en
[Email]

Contents

Acknowledgement

First and foremost, a special big appreciation from my heart to the most wonderful women in my life, my mom **SOUAD OUN ABBES** who left me this days and I'm offering this work to her because she left behind her a strong ambitious successful lady who will work hard to make her proud of me. Love you from the deepest of my heart

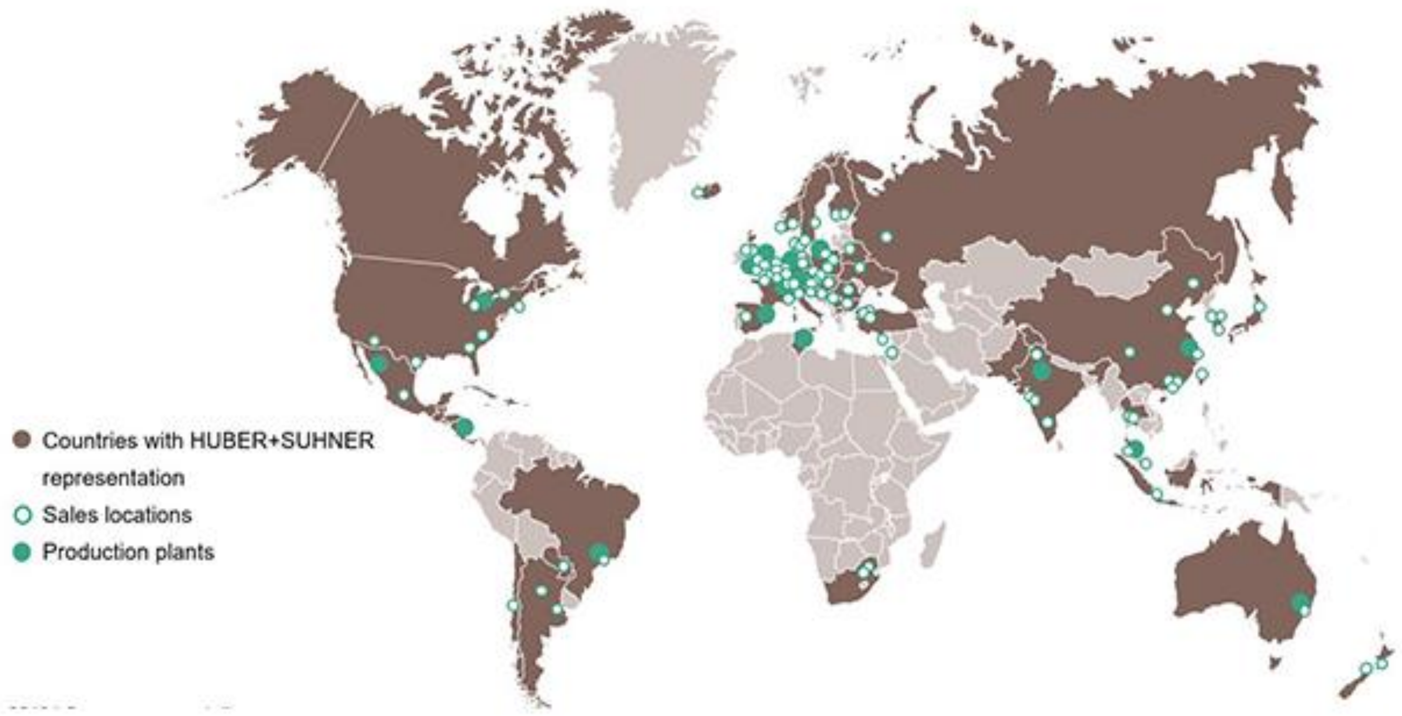
I pay my gratitude of course to the almighty Allah for giving me the ability to work hard successfully, then I would like to express my indebtedness appreciation to my supervisor **GHAZI ABBES.** His constant guidance and advice played a vital role in making the execution of the report.

Finally, I express my deep gratefulness to all Huber+Shuner members for giving me this opportunity to improve myself and my knowledge through this internship.

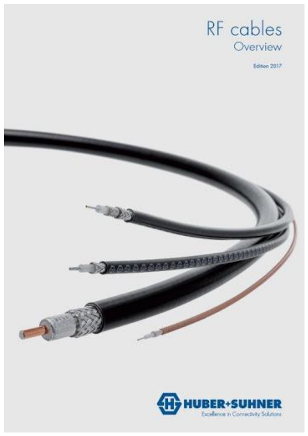
Introduction

HUBER+SUHNER offers to their customers around the globe outstanding products and services for their electrical and optical connectivity needs. It focus on combining products from the three technologies of Radio Frequency, Fiber Optics and Low Frequency to create solutions for Communication, Transportation and Industrial. And the balanced diversity of our 3x3 strategy gives us long-term financial stability.

HUBER+SUHNER products deliver high performance, quality, reliability and a long service life – even under the toughest of conditions. Our global production network, combined with group companies and agencies in over 80 countries, puts HUBER+SUHNER close to its customers

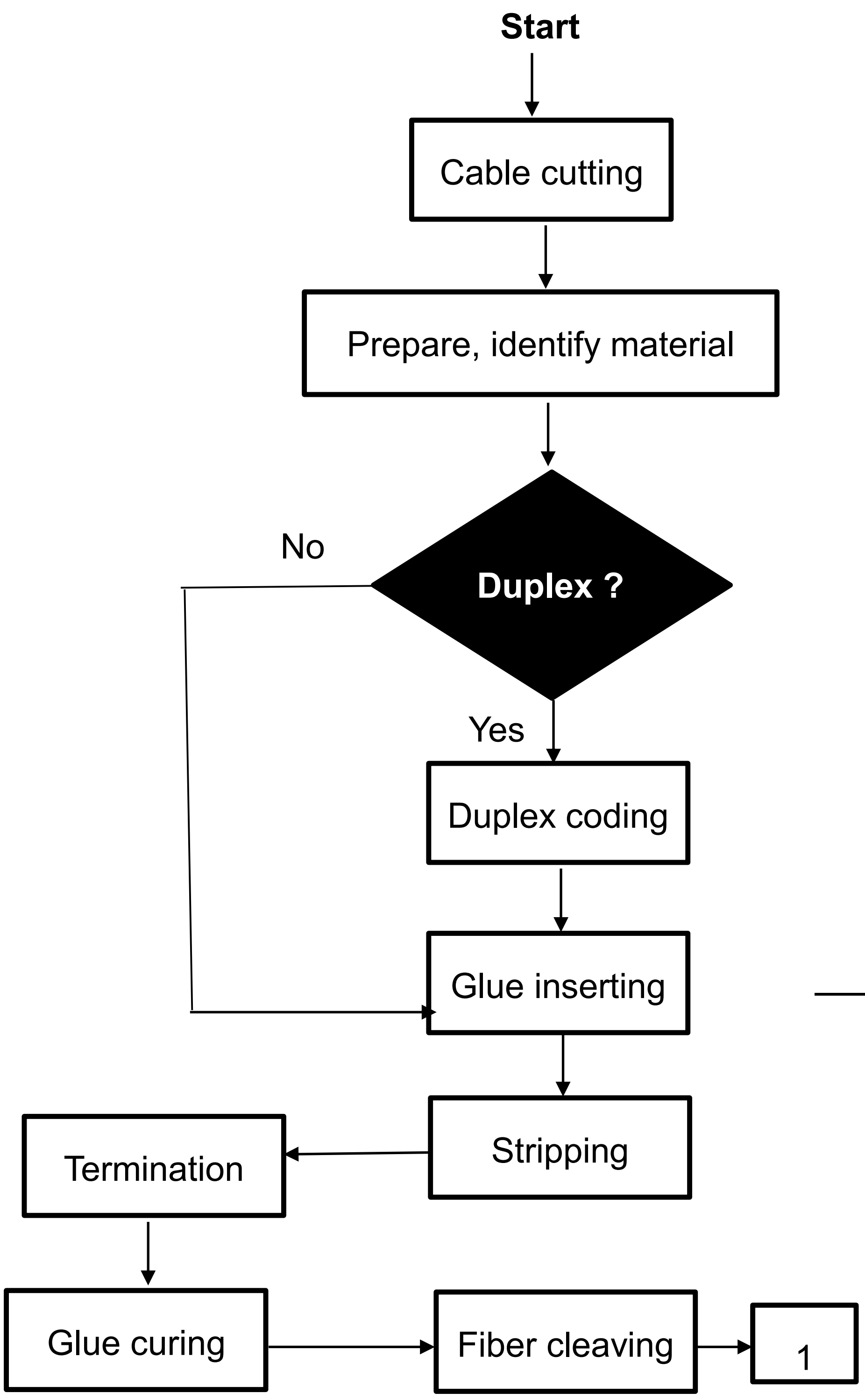


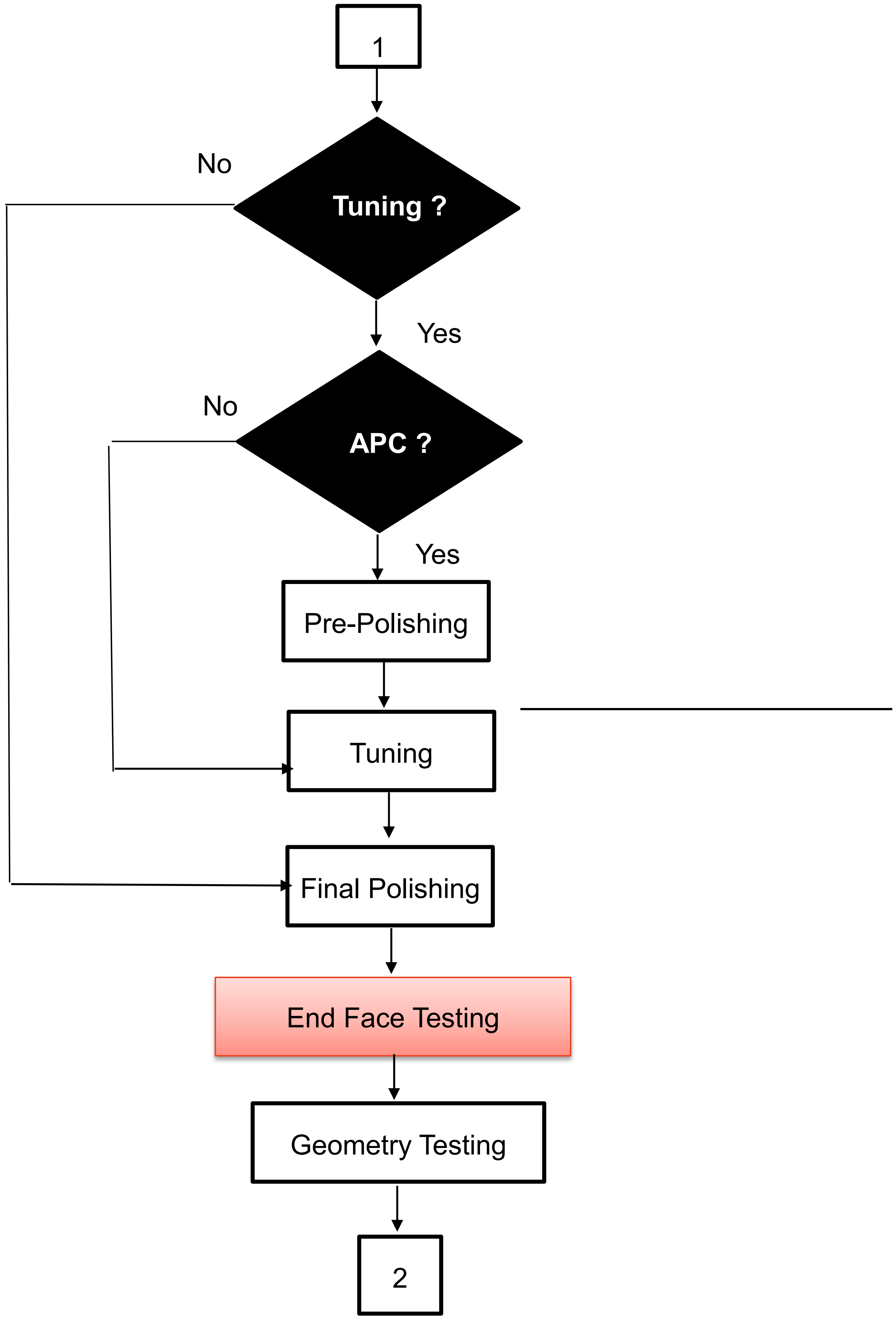
Products

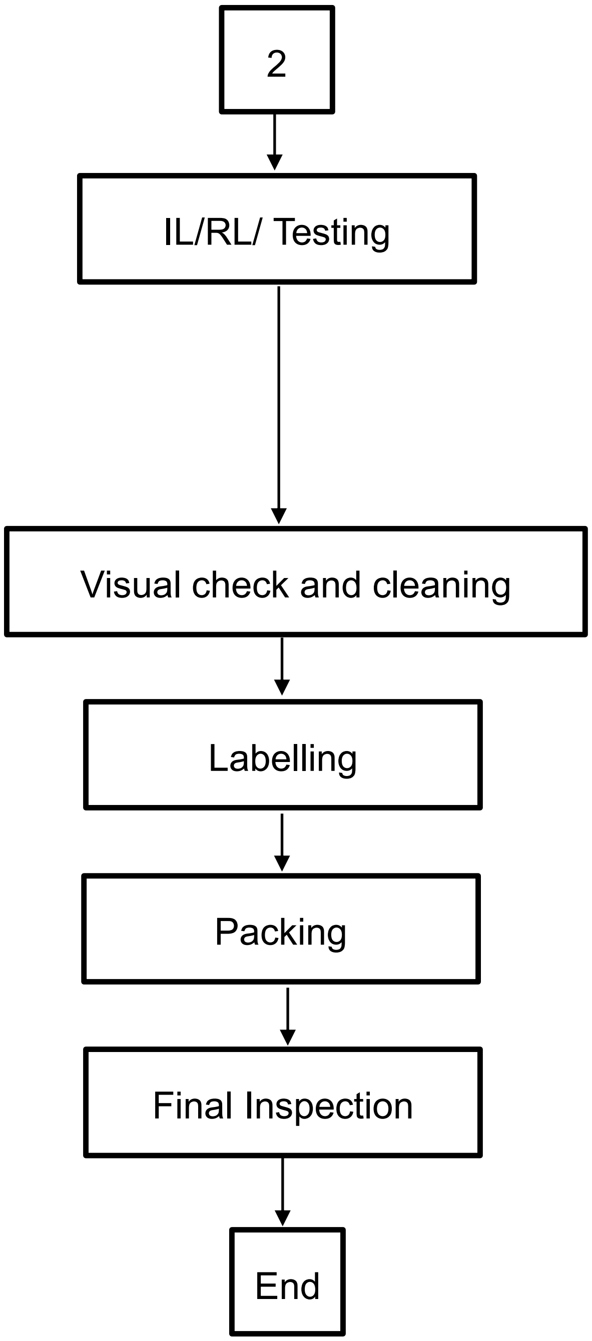


Radio Frenquency	FIBER OPTICS	LOW AND HIGH VOLTAGE
Antennas	Connectors/ optical components	Cable systems
Assemblies	Cables	Data transmission
Cables	Assemblies/ Cable systems	Signal-/ Power cables
Connectors/ adaptors	Fiber management systems	
Lightning and EMP protectors	Hybrid installation systems	
RF components	Optical switches	
RF-over-fiber-series	Network systems	
Ex certified	Manufacturing and cleaning tools	

Fiber Optics Instructions: Standard Assemblies Process Workflow




1
Tuning ?
No
Yes
APC ?
No
Yes
Pre-Polishing
Tuning
Final Polishing
End Face Testing
Geometry Testing
2


2
IL/RL/ Testing
Visual check and cleaning
Labelling
Packing
Final Inspection
End

Preface and Problematic

During my internship I'm interested of handling the phase of End Face Testing wish must be done with a 200x/400x microscope acc to IEC standards.

HUBER+SUHNER use 200x microscope. To achieve the best possible results for interpretation, they adjust the following parameters:

- Contrast ratio min 1:500 to 1:3000
- Standard screen with the appropriate interfaces.

The Problematic:

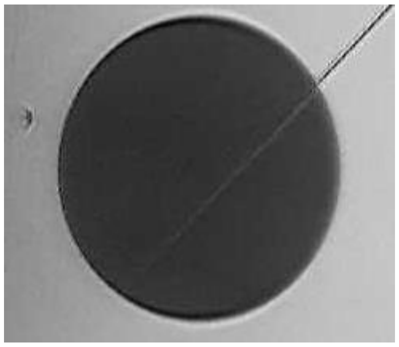
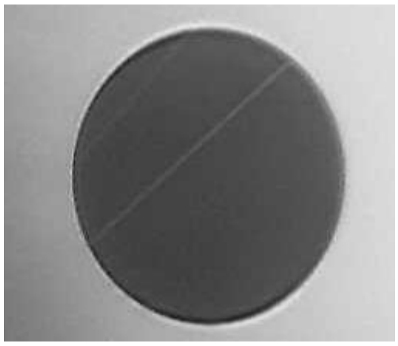
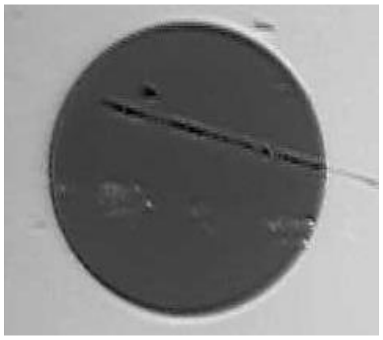
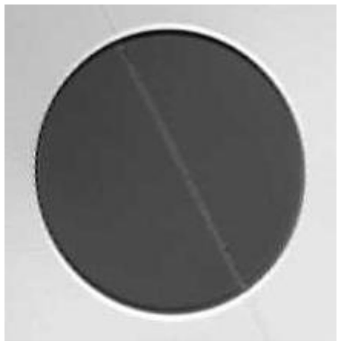
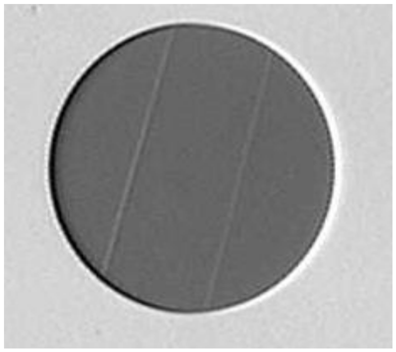
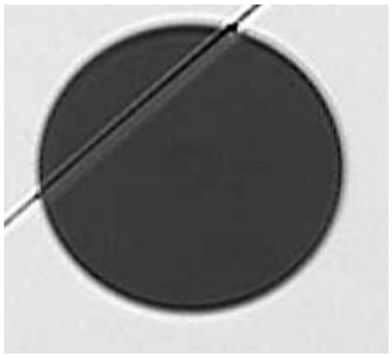
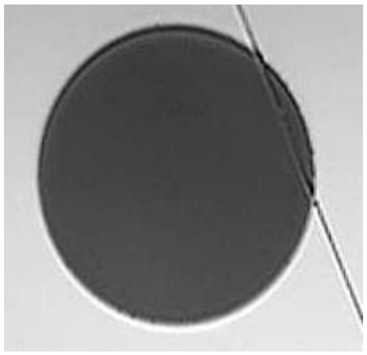
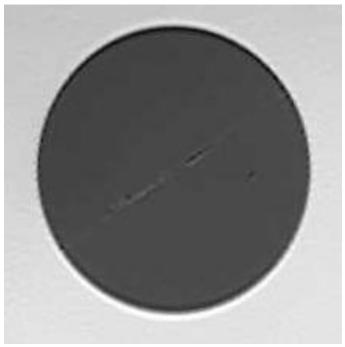
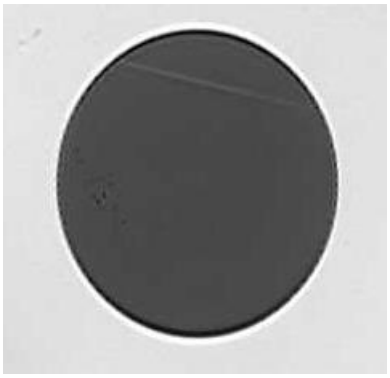
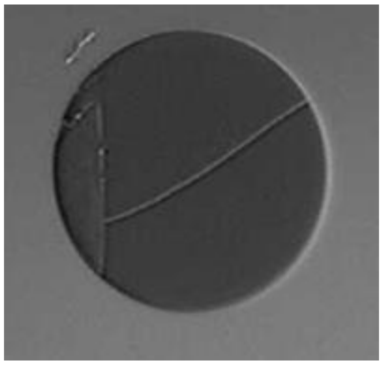
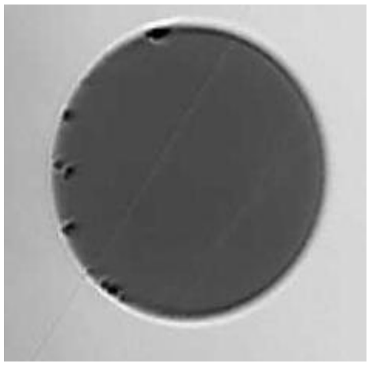
Manual inspection requires the presence of a person, an inspector who performs assessment of the entity under question and passes judgment on it according to some training or previous knowledge. No equipment are required except the naked eye of the trained inspector and a microscope. To reduce the effects of human subjectivity in the analysis process, the end face must be analyzed via an algorithmic process more accurate.

Here I will focus only on scratches because analyzing all the defects needs an intelligent system working with Artificial intelligence and neural network, a data set with high quality images and a long period of work, so as this is my first step in Data Sciences and computer vision I decided to work with image processing using Python. Thanks to libraries existing in Python I was be able to develop an algorithmic process that can analyses scratches on fiber's end face.

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

My first mission was to collect as possible as I can screen shots for fiber’s end face from microscopes trying to identify many cases of the emplacement of the scratches

The size of the images was impropriate because the cladding area was too big, so I snipped all the screen shots that I got and here some examples:

According to the requirements of the IEC standard for low or high resolution systems, for scratches the requirements refer to the width.

ZONE	SCRATCHES
A: Core Zone (dia = 0-25μm)	None (no scratches >= 1 μm)
B : Cladding Zone (dia = 25 -115 μm)	No limit < 3 μm None > 3 μm
C : Epoxy zone (dia = 115 -135 μm)	No limit
D : Contact Zone (dia 135-250μm)	No limit

Code and explanation

My code consists of many under functions:

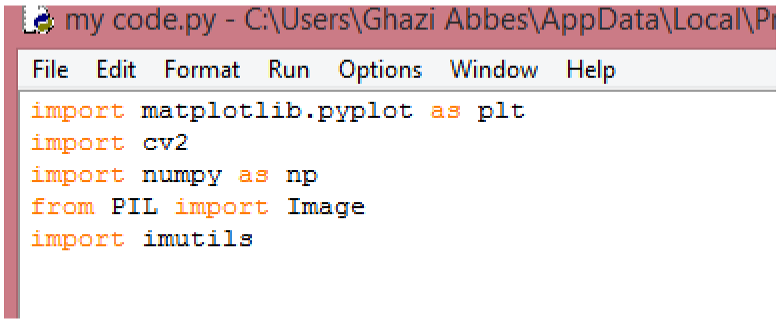
- Importing the necessary libraries.
- Eliminating the gladding area by cropping the image.
- Identifying the center of the fiber’s end face.
- Flirting.
- Detecting the scratches using Hough Line Transform.
- Identifying the Scratches.
- Drawing the 3 different zones.
- Identifying the emplacement of the scratches.
- Passing or Failing.
- Showing all the results in details.

PS: I want to mention that the quality of the images was too bad because they haven’t the same conditions like brightness, contrast and size. Also the main difficulty was the emplacement of the scratches. One fiber’s end face can have more than one scratch with different width, different length and the algorithmic process must work wet ever the case is.

Importing the necessary libraries

Import: a key word to call the main library.

As: rename a library as you want.


my code.py - C:\Users\Ghazi Abbes\AppData\Local\Pr
File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from PIL import Image
import imutils
```

Matplotlib: is a comprehensive library for creating static, animated, and interactive visualizations in Python.

OpenCv (CV2): is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel. The library is cross-platform and free for use under the open-source BSD license.

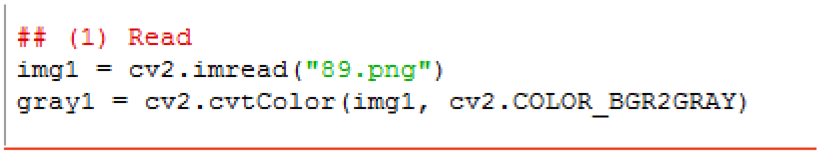
Numpy: is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

PIL : **Python Imaging Library** (abbreviated as **PIL**) (in newer versions known as **Pillow**) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

Imutils: A python library for simple image processing functions like translation, rotation, resizing and displaying matplotlib images, sorting contours and much more easier with OpenCv.

Eliminating the gladding area by cropping the image

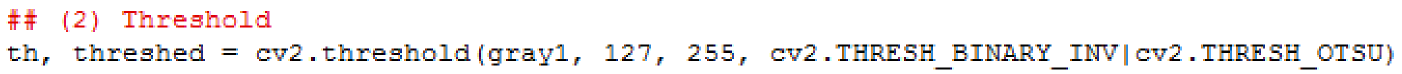
1/ reading the image and converting it to grayscale:



```
## (1) Read
img1 = cv2.imread("89.png")
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```

At first, our original image is a BGR image, it means that the image is composed from three grades B: blue, G: green, R: red. We need to convert our input image to grayscale level so that the computer can deal with them.

2/ thresholding:



```
## (2) Threshold
th, threshed = cv2.threshold(gray1, 127, 255, cv2.THRESH_BINARY_INV|cv2.THRESH_OTSU)
```

Thresholding: In digital image processing, **thresholding** is the simplest method of segmenting images. From a grayscale image, thresholding can be used to create binary images.

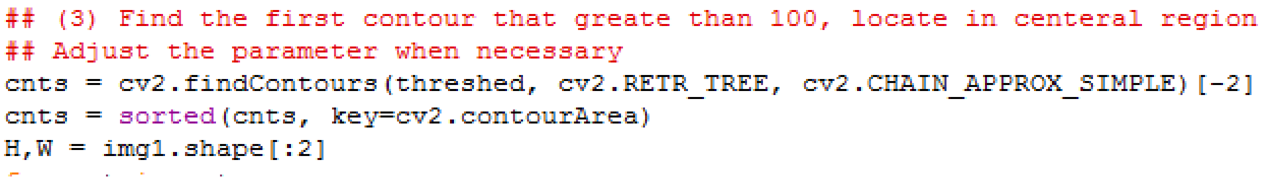
cv2.THRESH_BINARY_INV: the pixel value takes 1 if its value > thresh, else it takes maxval.

cv2.THRESH_OTSU: it automatically calculates a threshold value from image histogram for a bimodal image.

Parameters:

src – input array (single-channel, 8-bit or 32-bit floating point).
dst – output array of the same size and type as `src`.
thresh – threshold value.
maxval – maximum value to use with the `THRESH_BINARY` and `THRESH_BINARY_INV` thresholding types.
type – thresholding type (see the details below).

3/ Fiding contours:

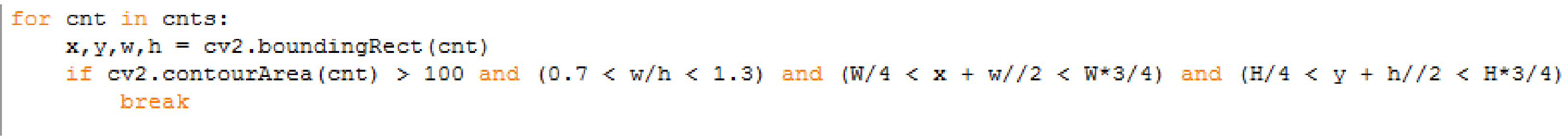


```
## (3) Find the first contour that greate than 100, locate in centeral region
## Adjust the parameter when necessary
cnts = cv2.findContours(threshed, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)[-2]
cnts = sorted(cnts, key=cv2.contourArea)
H,W = img1.shape[:2]
```

Eliminating the gladding area by cropping the image

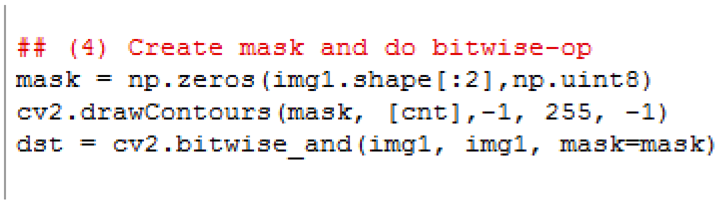
In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

4 / Looping over the contours:



```
for cnt in cnts:
    x,y,w,h = cv2.boundingRect(cnt)
    if cv2.contourArea(cnt) > 100 and (0.7 < w/h < 1.3) and (W/4 < x + w//2 < W*3/4) and (H/4 < y + h//2 < H*3/4)
        break
```

5/ Creating a mask and do bitwise-op:

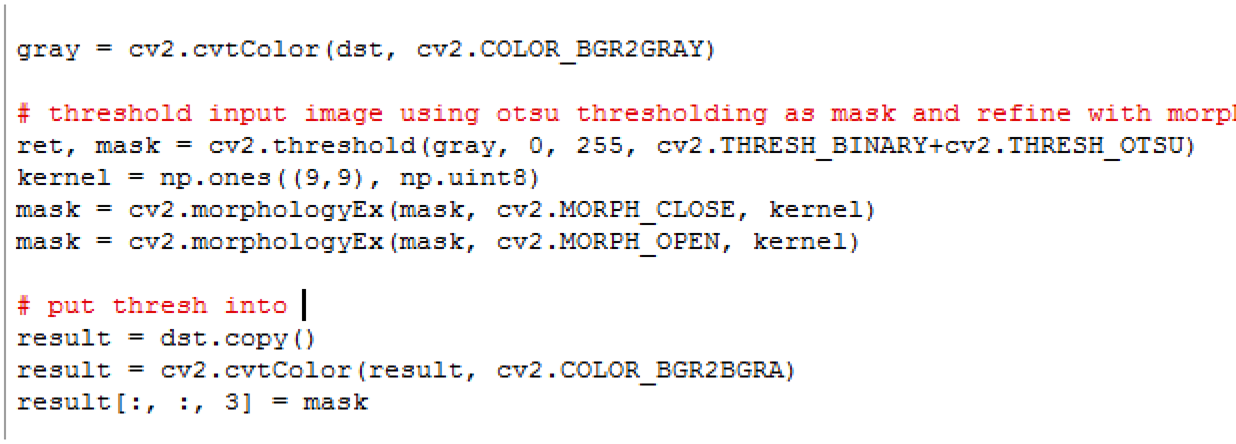


```
## (4) Create mask and do bitwise-op
mask = np.zeros(img1.shape[:2],np.uint8)
cv2.drawContours(mask, [cnt],-1, 255, -1)
dst = cv2.bitwise_and(img1, img1, mask=mask)
```

Bitwise-op:

In **Python**, **bitwise** opera -tors are used to perform **bitwise** calculati ons on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name **bitwise** operators.

6 */ thresholding the input image using Otsu thresholding as mask and refine With morphology:*



```
gray = cv2.cvtColor(dst, cv2.COLOR_BGR2GRAY)

# threshold input image using otsu thresholding as mask and refine with morp
ret, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
kernel = np.ones((9,9), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

# put thresh into
result = dst.copy()
result = cv2.cvtColor(result, cv2.COLOR_BGR2BGRA)
result[:, :, 3] = mask
```

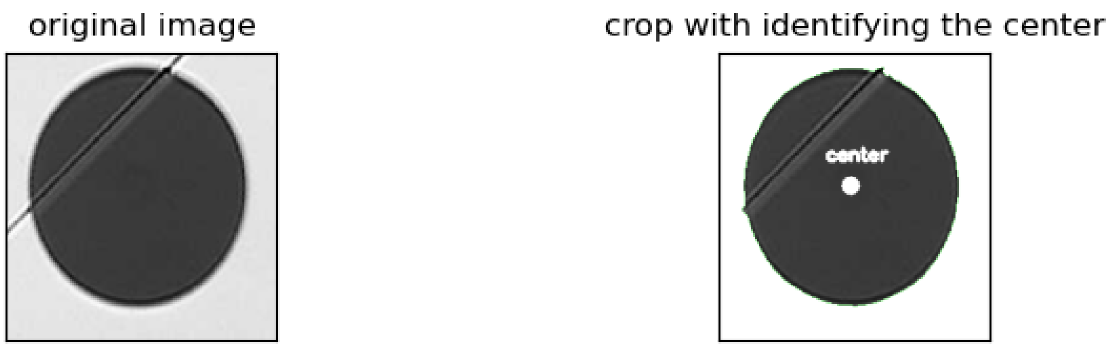
cv2.morphologyEx:

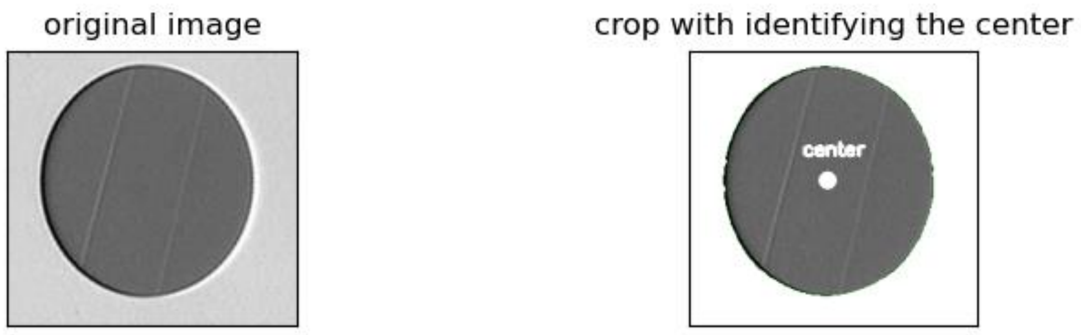
Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, and second one is called **structuring element** or **kernel** which decides the nature of operation

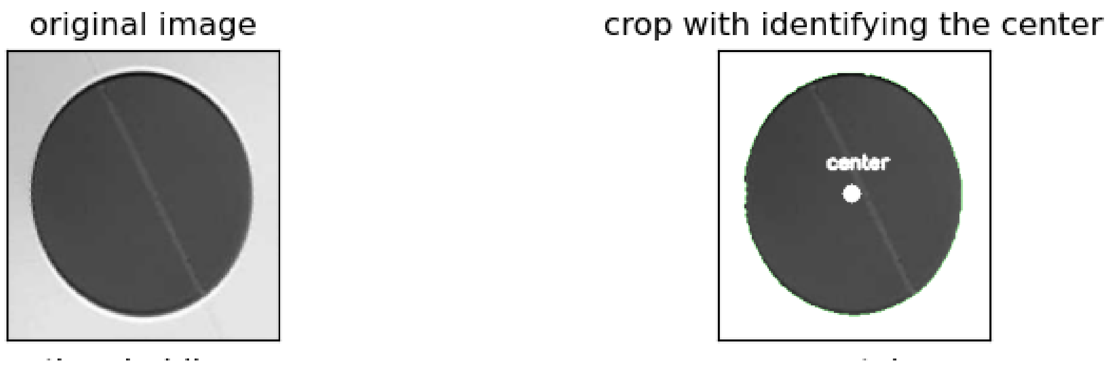
After those operations, I was able to remove the background wish is the gladding in order to concentrate only on the fiber's end face. The variable **result** is our output.

Eliminating the gladding area by cropping the image

These are the input images with their result:





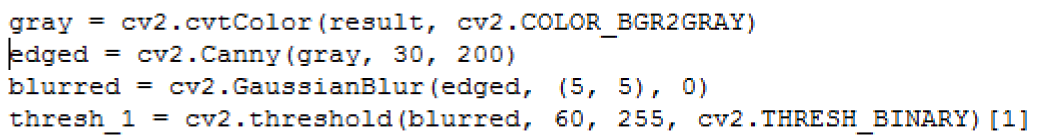


We can see that the gray background in every image is removed perfectly while the external contour of the fiber is preserved.

Identifying the center of the fiber’s end face

My output (result) is now my input in the following operations.

1 */ converting to Grayscale, filtering, blurring and thresholding:*



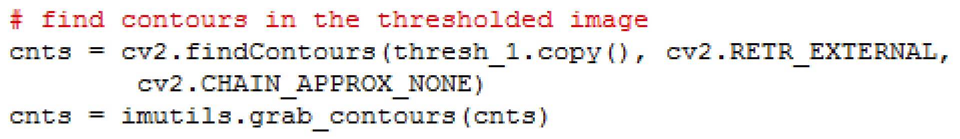
```
gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 200)
blurred = cv2.GaussianBlur(edged, (5, 5), 0)
thresh_1 = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]
```

Canny: The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a *computational theory of edge detection* explaining why the technique works.

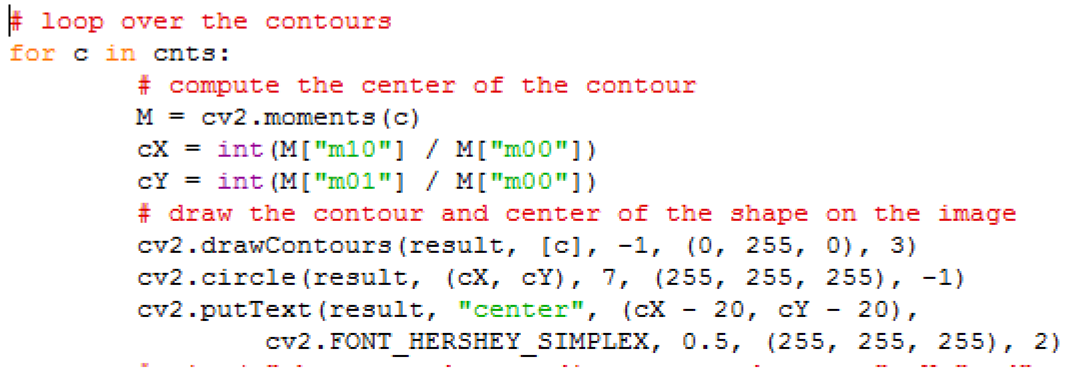
First of all, I converted the result to grayscale level like usual using the function

Cv2.cvtColor from the library OpenCV, secondly I used the filter named **canny edge detector** in order to find the edges of the contour, then I used the **Gaussian Blur filter** to make the image more smoothly and then threshold it with the **THRESH_BINARY.**

2 */ finding the contour in the thresholded image and looping over the contours in order to extract the center of the fiber:*

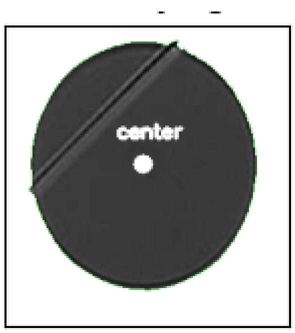


```
# find contours in the thresholded image
cnts = cv2.findContours(thresh_1.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_NONE)
cnts = imutils.grab_contours(cnts)
```



```
# loop over the contours
for c in cnts:
        # compute the center of the contour
        M = cv2.moments(c)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        # draw the contour and center of the shape on the image
        cv2.drawContours(result, [c], -1, (0, 255, 0), 3)
        cv2.circle(result, (cX, cY), 7, (255, 255, 255), -1)
        cv2.putText(result, "center", (cX - 20, cY - 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
```

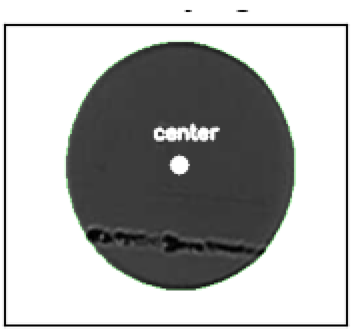
Identifying the center of the fiber's end face



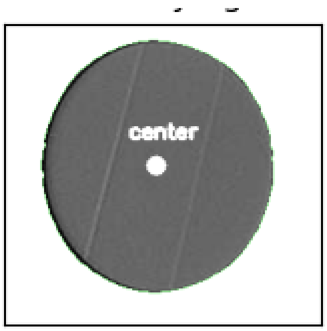
1/ After finding the fiber's contours using the function **cv2.findContours**, we grab the appropriate tuple value based on whether we are using OpenCV 2.4, 3, or 4.

2/ Start looping over each of the individual contours, followed by computing **image moments** for the contour region and the center.

"In computer vision and image processing, image moments are often used to characterize the shape of an object in an image. These moments capture basic statistical properties of the shape, including the *area* of the object, the centroid (i.e., the center (x, y)-coordinates of the object*)*, orientation, along with other desirable properties. Here I'm only interested in *the* center of the contour, which I compute".



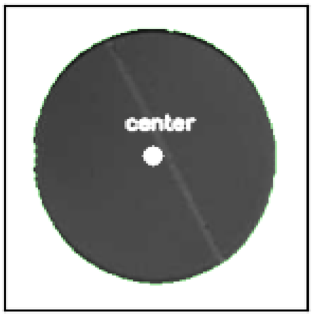
3/ Drawing the outline of the contour surrounding the current shape by making a call to cv2.drawContours



4/ Writing the text centre near the white circle.

PS:

- There is a function named **HOUGH CERCLE DETECTION** that can easily detect a circle and identify its radius and centre. Unfortunately, the circular contour of the fiber's end face isn't a perfect circle that is why I was unable to use this function.
- It is very important to find the centre because our evaluation will articulate on the core zone of the end face.



Filtering

In order to improve the quality of the images, filtering is required to make easier the extraction of valuable information from images such as edges, corners, and blobs and make them more visually appealing.

Sobel filter:
The **Sobel operator**, sometimes called the **Sobel–Feldman operator** or **Sobel filter**, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges.

1 / *The Sobel filter:*

```
sobelX= cv2.Sobel(gray, cv2.CV_64F, 1,0)
sobelY= cv2.Sobel(gray, cv2.CV_64F, 0,1)

sobelX=np.uint8(np.absolute(sobelX))
sobelY=np.uint8(np.absolute(sobelY))

sobelCombined=cv2.bitwise_or(sobelX,sobelY)
```

First of all, I used the sobelX and sobelY filters on the grayscale image and then combined them with bitwise_or to get a better result.

2/ The Gaussian filter and thresholding:

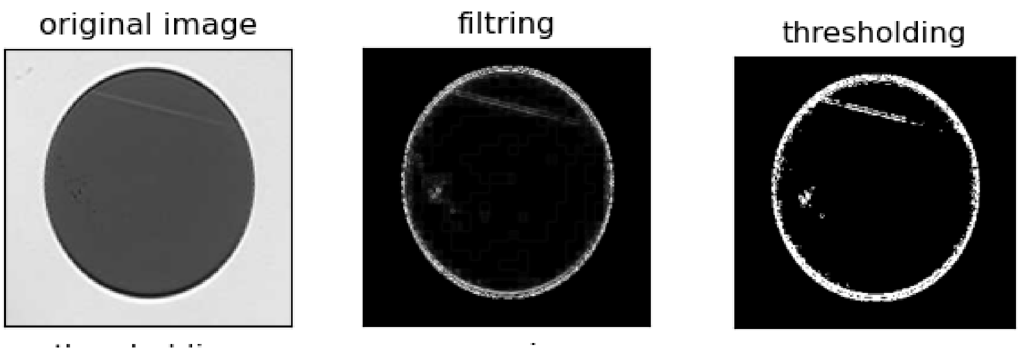
Gaussian smoothing is used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales.

Gaussian blur:

In image processing, a **Gaussian blur** (also known as **Gaussian smoothing**) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss).

```
gblur = cv2.GaussianBlur(sobelCombined, (1,1),0)

_, th1= cv2.threshold(sobelCombined,50,255, cv2.THRESH_BINARY)
cv2.imwrite('th1.png',th1)
```

For segmentation, I used the thresholding to binarize my image so that the scratch with the borders of the contour become explicit.



Detecting the scratches using Hough Line Transform

I saved my output in the last line named th1.

1/ pre-treatment:

- Reading the input th1
- Convert it to grayscale level.
- Filtering it using Gaussian Blur filter.
- Apply canny edge detector

```
img = cv2.imread('th1.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 50
high_threshold = 150
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```

These are the parameters of **the Hough Line Transform** where every parameter is explicated as below in red:

```
rho = 1  # distance resolution in pixels of the Hough grid
theta = np.pi / 180  # angular resolution in radians of the Hough grid
threshold = 15  # minimum number of votes (intersections in Hough grid cell)
min_line_length = 120 # minimum number of pixels making up a line
max_line_gap =20   # maximum gap in pixels between connectable line segments
line_image = np.copy(img) * 0  # creating a blank to draw lines on the image
```

PS: the scratches in the images haven't the same gap of the parameter **min_line_length**, so we can change this parameter according to the scratch that we have.

Kernel: in image processing, a **kernel**, **convolution matrix**, or **mask** is a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between a kernel and an image.

Hough transform:

The **Hough transform** is a feature extraction technique used in image analysis, computer vision, and digital image processing.[1]

The classical Hough transform was concerned with the identification of lines in the image

Detecting the scratches using Hough Line Transform

Applying now the Hough function on the edge detected image:

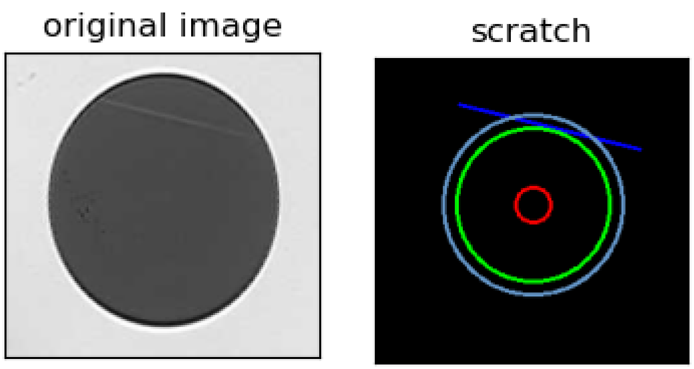
```
lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                    min_line_length, max_line_gap)

for line in lines:
    for x1,y1,x2,y2 in line:
          cv2.line(line_image,(x1,y1),(x2,y2),(0,0,255),2,4)
```

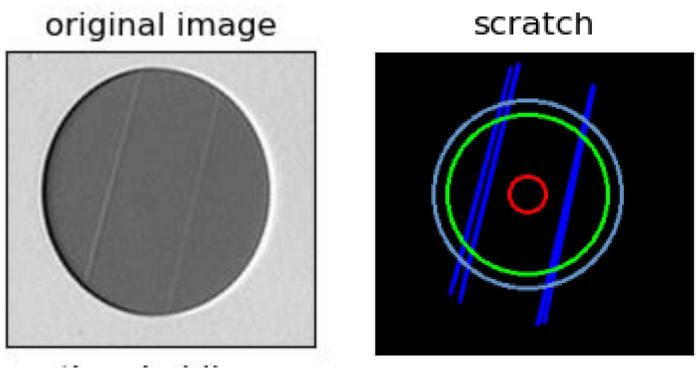
The output "**lines**" is an array containing endpoints of detected line segments.

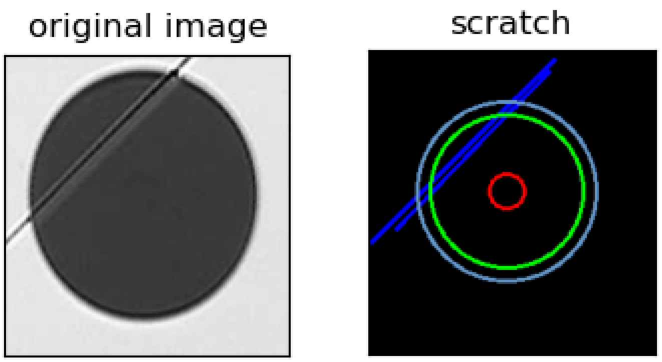
Start looping over lines and identify the line with two points a (x1, y1) and b (x1, y2) in blue color with thickness =2, and then draw the line on the image.

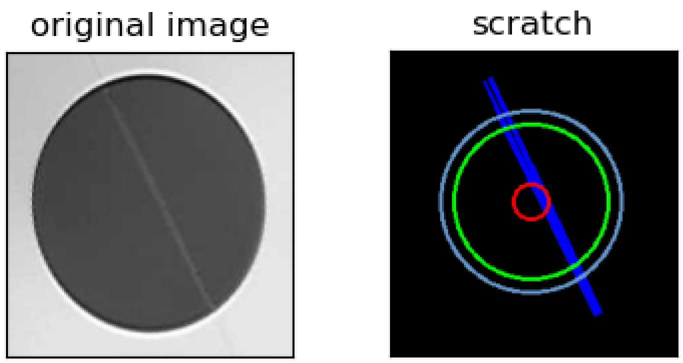
```
lines_edges = cv2.addWeighted(img, 0.8, line_image, 1, 0)
```

These are some examples:









Identifying the scratches

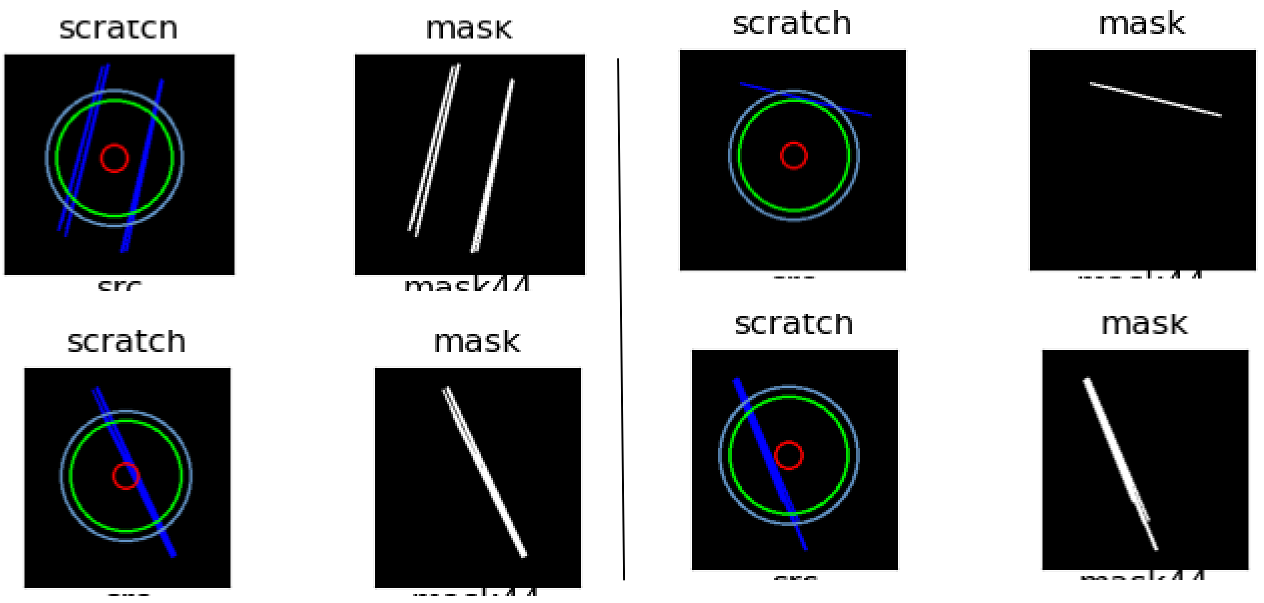
After detecting the scratches with Hough Transform, I'm trying now to identify the scratch only wish is drawled with blue color.

So this how it works with few lines of coding:

```
line_image2 = cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB)

hsv = cv2.cvtColor(line_image2, cv2.COLOR_BGR2HSV)
lower_range = np.array([110,50,50])
upper_range = np.array([130,255,255])
mask3 = cv2.inRange(hsv, lower_range, upper_range)
```

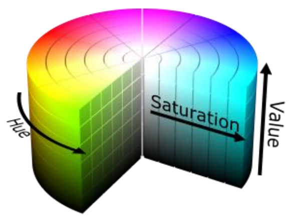
- Convert the line_image variable from BGR to RGB because the library OpenCV reads the images only in RBG mode.
- Convert the RGB image (line_image2) to HSV.
- Identify 'lower_range' and 'upper-range' denotes the lower and upper boundary of the blue color.
- Using the cv2.inRange function to segment out a particular blue region from the image.

The result:



HSV: (**hue, saturation, value**, also known as or **HSB** [**hue, saturation, and brightness**]) is an alternative representation of the RGB color model.

The **HSV** representation models the way paints of different colors mix together, with the *saturation* dimensio n resembling various tints of brightly colored paint, and the *value* dimension resembling the mixture of those paints with varying amounts of black or white paint



Drawing the three different zones

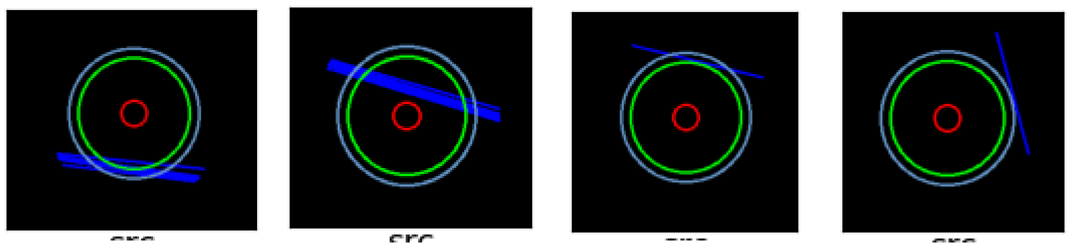
We have three different zones:

- Zone A in **red** named the core zone wish is the most important one.
- Zone B in **green** named the cladding zone.
- Zone C in **blue** named the Epoxy zone.

It's easy to identify these three zones thanks to the function cv2.circle.

Parameters are:
image: It is the image on which circle is to be drawn.
center_coordinates: It is the center coordinates of circle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value) wish are the coordinates of the center's fiber.
Radius: It is the radius of circle.
Color: It is the color of border line of circle to be drawn. For **BGR**, we pass a tuple. Eg: (255, 0, 0) for blue color.
Thickness: It is the thickness of the circle border line in **PX**. Thickness of **-1 PX** will fill the circle shape by the specified color.
Return Value: It returns an image.

```
img=cv2.circle(line_image,(cX,cY), 13, (255,0,0), 2)
img=cv2.circle(line_image,(cX,cY), 58, (0,255,0), 2)
img=cv2.circle(line_image,(cX,cY), 68, (100,150,200), 2)
```

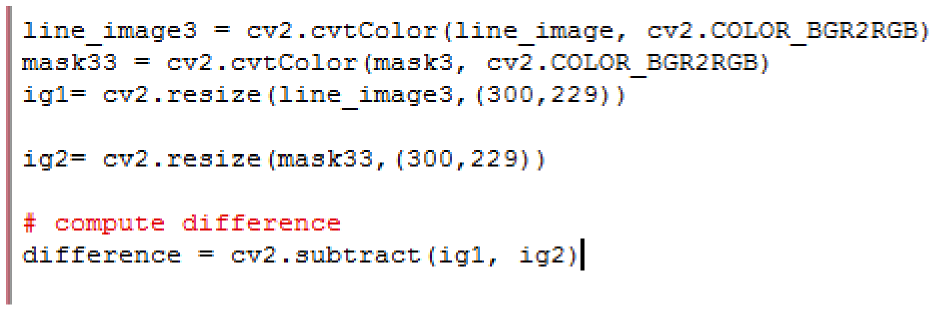


Identifying the emplacement of the scratch

In the first, my idea was finding the intersection between the scratch and the main zone circle. Unfortunately, that was very hard to achieve because finding the intersection needs many parameters wish are the equation of every scratch, the equation of every circle and there is a big part of calculations in that.

So why thinking mathematically! Let’s think easier than that and make things simpler. In fact, my solution was subtracting the scratch from the image by doing the difference between the image where the scratch exists only and the image with scratch and three zones.

So every time the scratch passes from a zone, we will find as result that the contour of this zone is open while the contours of the intact zones are close.

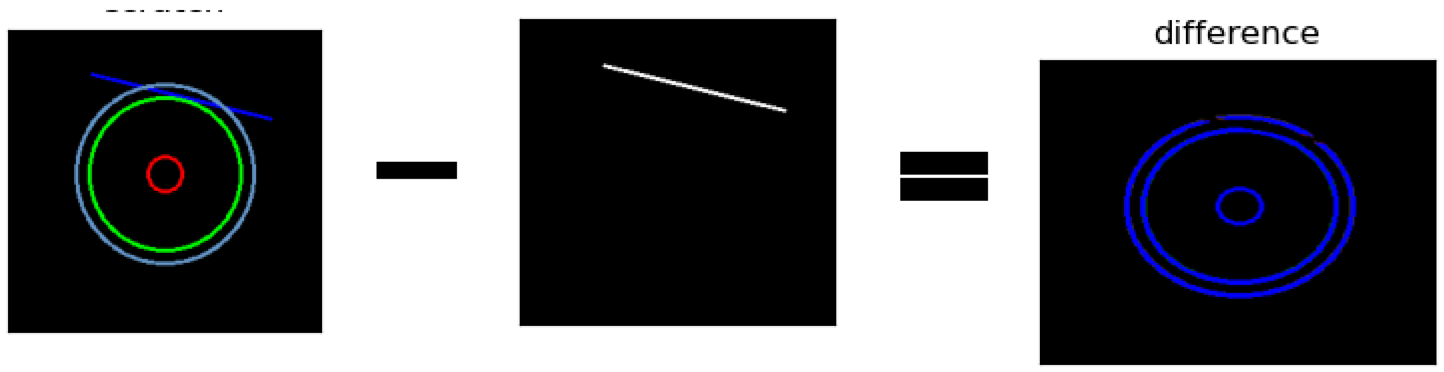


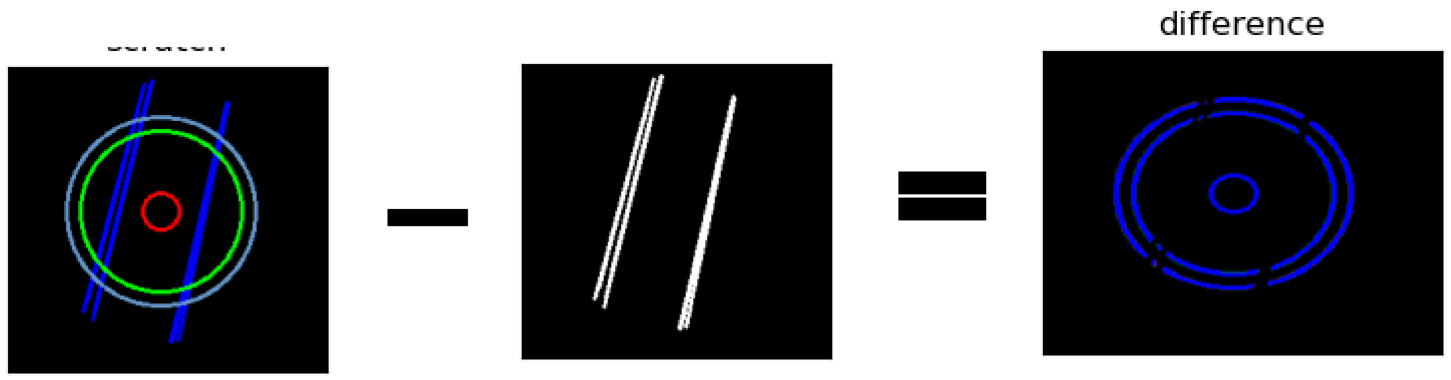
```
line_image3 = cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB)
mask33 = cv2.cvtColor(mask3, cv2.COLOR_BGR2RGB)
ig1= cv2.resize(line_image3,(300,229))

ig2= cv2.resize(mask33,(300,229))

# compute difference
difference = cv2.subtract(ig1, ig2)
```

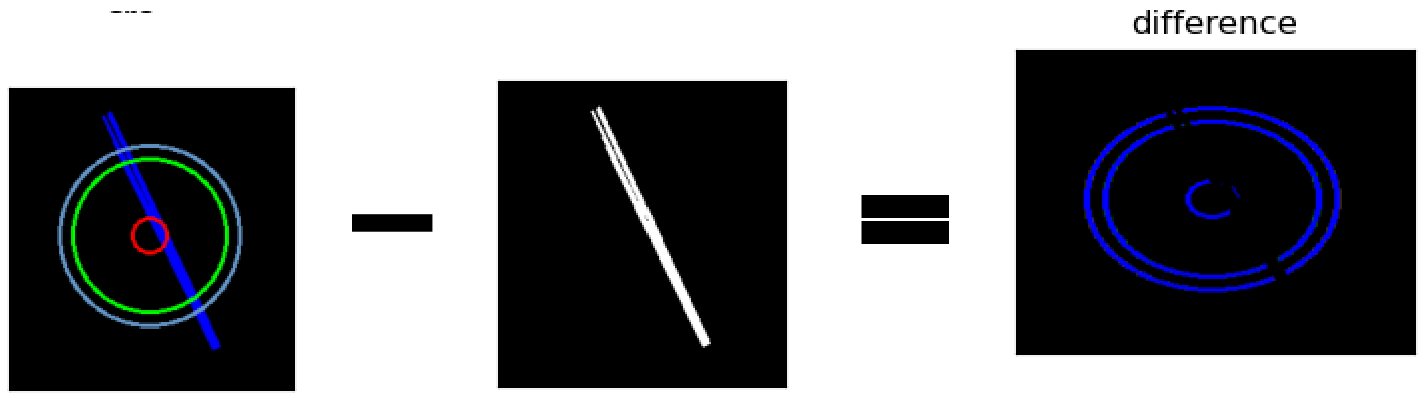
The variable “**difference**” is our output.

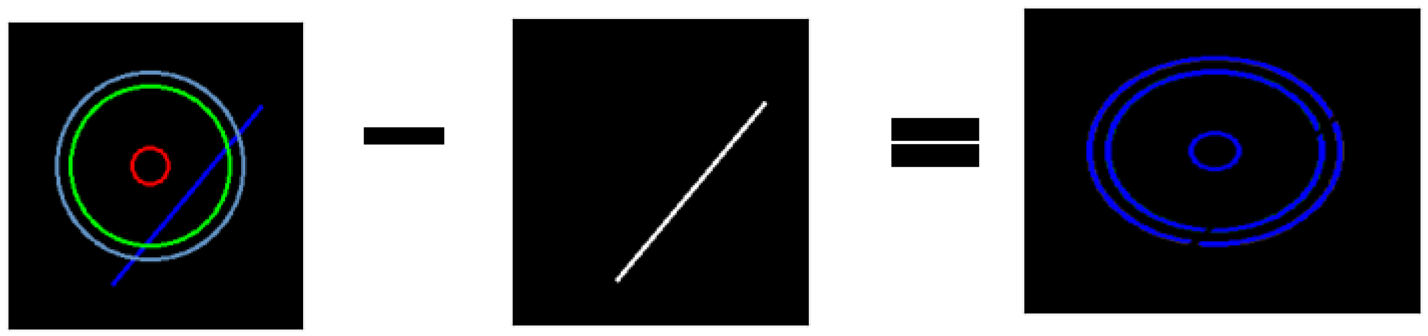
Identifying the emplacement of the scratch

The result:



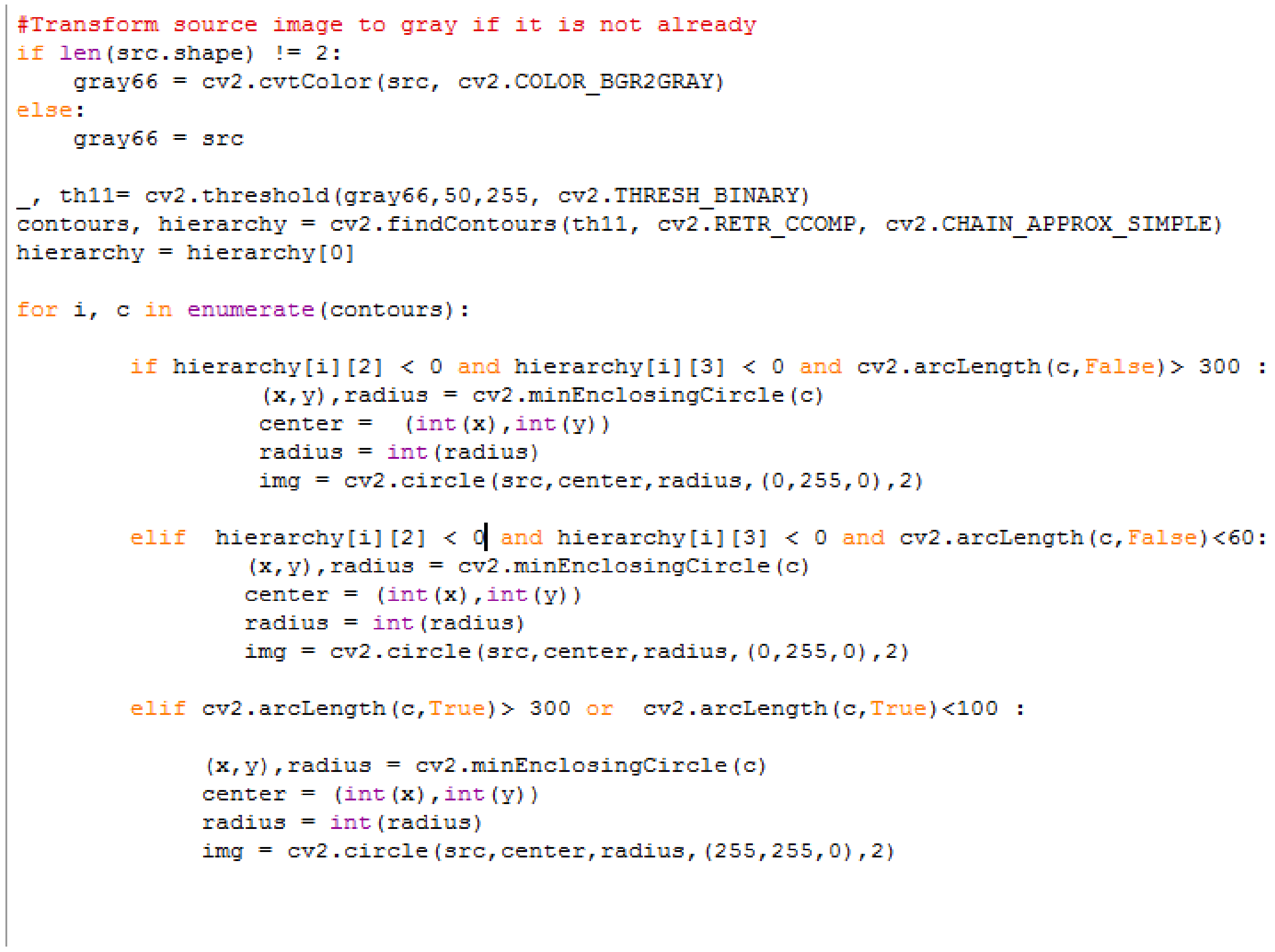






Identifying the emplacement of the scratch

Now we have to identify wish zone is open and wish zone is close to know from where the scratch passed.

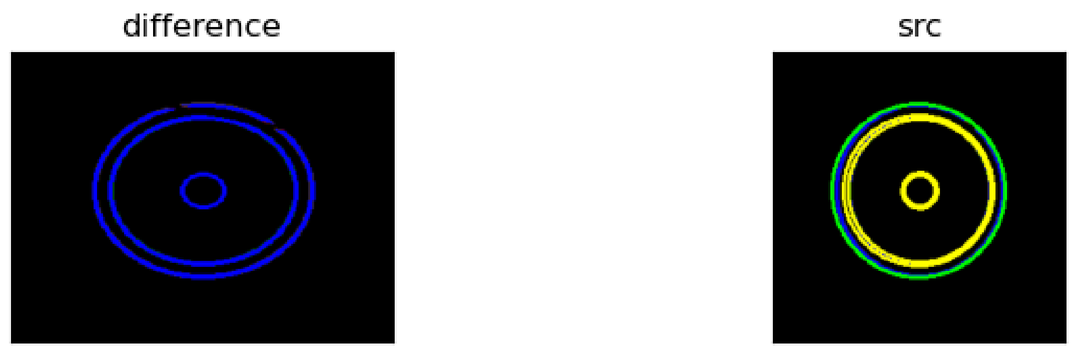


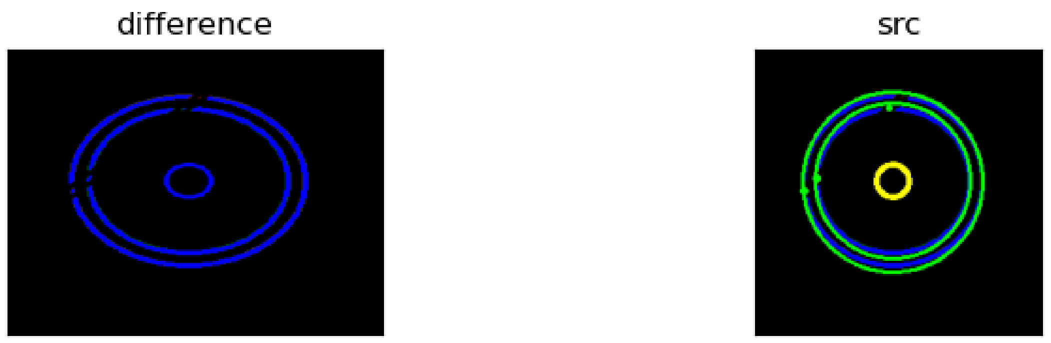
```
#Transform source image to gray if it is not already
if len(src.shape) != 2:
    gray66 = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
else:
    gray66 = src

_, th11= cv2.threshold(gray66,50,255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(th11, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
hierarchy = hierarchy[0]

for i, c in enumerate(contours):

        if hierarchy[i][2] < 0 and hierarchy[i][3] < 0 and cv2.arcLength(c,False)> 300 :
                 (x,y),radius = cv2.minEnclosingCircle(c)
                 center =  (int(x),int(y))
                 radius = int(radius)
                 img = cv2.circle(src,center,radius,(0,255,0),2)

        elif  hierarchy[i][2] < 0 and hierarchy[i][3] < 0 and cv2.arcLength(c,False)<60:
                (x,y),radius = cv2.minEnclosingCircle(c)
                center = (int(x),int(y))
                radius = int(radius)
                img = cv2.circle(src,center,radius,(0,255,0),2)

        elif cv2.arcLength(c,True)> 300 or  cv2.arcLength(c,True)<100 :

             (x,y),radius = cv2.minEnclosingCircle(c)
             center = (int(x),int(y))
             radius = int(radius)
             img = cv2.circle(src,center,radius,(255,255,0),2)
```

Just I used the function **cv2.<u>FindContours ()</u>** in my image, then deciding whether the contour is closed or not by examining the hierarchy passed to the **findContours ()** function. The hierarchy parameter which is optional output vector, contain information about the image topology. It has as many elements as the number of contours. If for the contour **i** there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. So by checking the value hierarchy[i][2] you can decide the contour belongs to closed or not, that is for a contour if the hierarchy[i][2] = -1 then no child and it belongs to opened.

In every “**if”** condition controlled the length of the detected contour. The opened contours are circled in green using the function **cv2.minEnclosingCircle ()** while the closed contours are circled in yellow circles.

The result:





- The yellow circles are the safe zones.
- The green circles are the contact zones.

Pass or Fail

Judging the passing or no of the fiber’s end face during the test is based on the scratch’s location.

So in this paragraph i will explain how I managed the different possible cases.

My main idea is detecting the safe untouched zones, it means the yellow circles, and from that deciding whatever the end face is ready to go or not.

```
src2 = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)

hsv2 = cv2.cvtColor(src2, cv2.COLOR_BGR2HSV)
lower_range = np.array([22, 93, 0])
upper_range = np.array([45, 255, 255])
mask44 = cv2.inRange(hsv2, lower_range, upper_range)

kernal = np.ones((3,3), np.uint8)
closing = cv2.morphologyEx(mask44, cv2.MORPH_CLOSE, kernal)
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernal)
contours, hierarchy = cv2.findContours(opening, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
for i, c in enumerate(contours):

        perimeter = cv2.arcLength(c,True)


        if  70 <perimeter <200 or  360 <perimeter <400 :
             print('zone A with or without B IS SAFE  ')
             print('\n')
             print ('***********PASS***************')
             break
        elif 70 <perimeter <200 or  410 <perimeter <490 :
             print ('zone A and zone C  are safe ')
             print('\n')
             print ('************PASS**************')
             break
        elif 360 <perimeter <490 or  410 <perimeter <490 :
             print ('zone A is not safe')
             print('\n')
             print ('************FAIL**************')
             break
        else :
             print ('************FAIL**************')
             break
```

Pass or Fail

Contour Perimeter:

It is also called arc length. It can be found out using **cv2.arcLength ()** function. Second argument specify whether shape is a closed contour (if passed `True`), or just a curve.

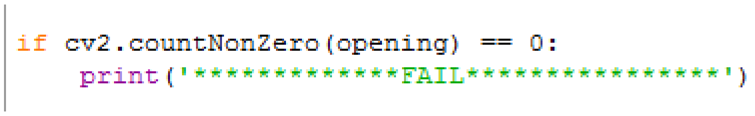
In the five first lines, the objective was to detect the circles in yellow color. So like I did before:

- Converting the image from BGR to RGB then from RGB to HSV.
- Identifying the lower and the upper range of the yellow color.
- Apply the function cv2.inRange on the mask.
- Apply morphological transformations to ameliorate the detected contours.

Now, the question is how can I identify the different zones?

The answer is by calculating the perimeter of the detected circles. We have a function in OpenCv library named **cv2.arcLength ()** wish returns the length of the detected contour (perimeter). So, as we have the radius of each zone we can calculate their length and compare the result with function output. Finally, we can judge the passing or failing of the end face.

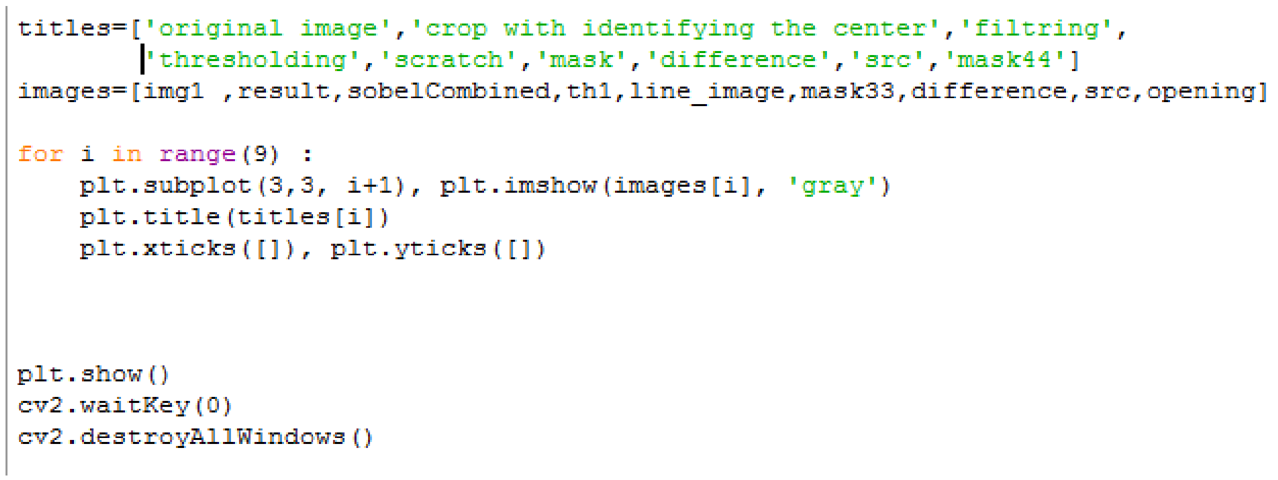
PS: if the scratch passes from the center it means that all the contours are opened and on the mask we are not going to detect any zone circle. The mask will be totally black and from that we can tell that the end face **fails** to go.



```
if cv2.countNonZero(opening) == 0:
    print('*************FAIL****************')
```

Showing the results in details

In this paragraph I used the package **Pyplot** (plt) from the library **Matplotlib** to print the different images.

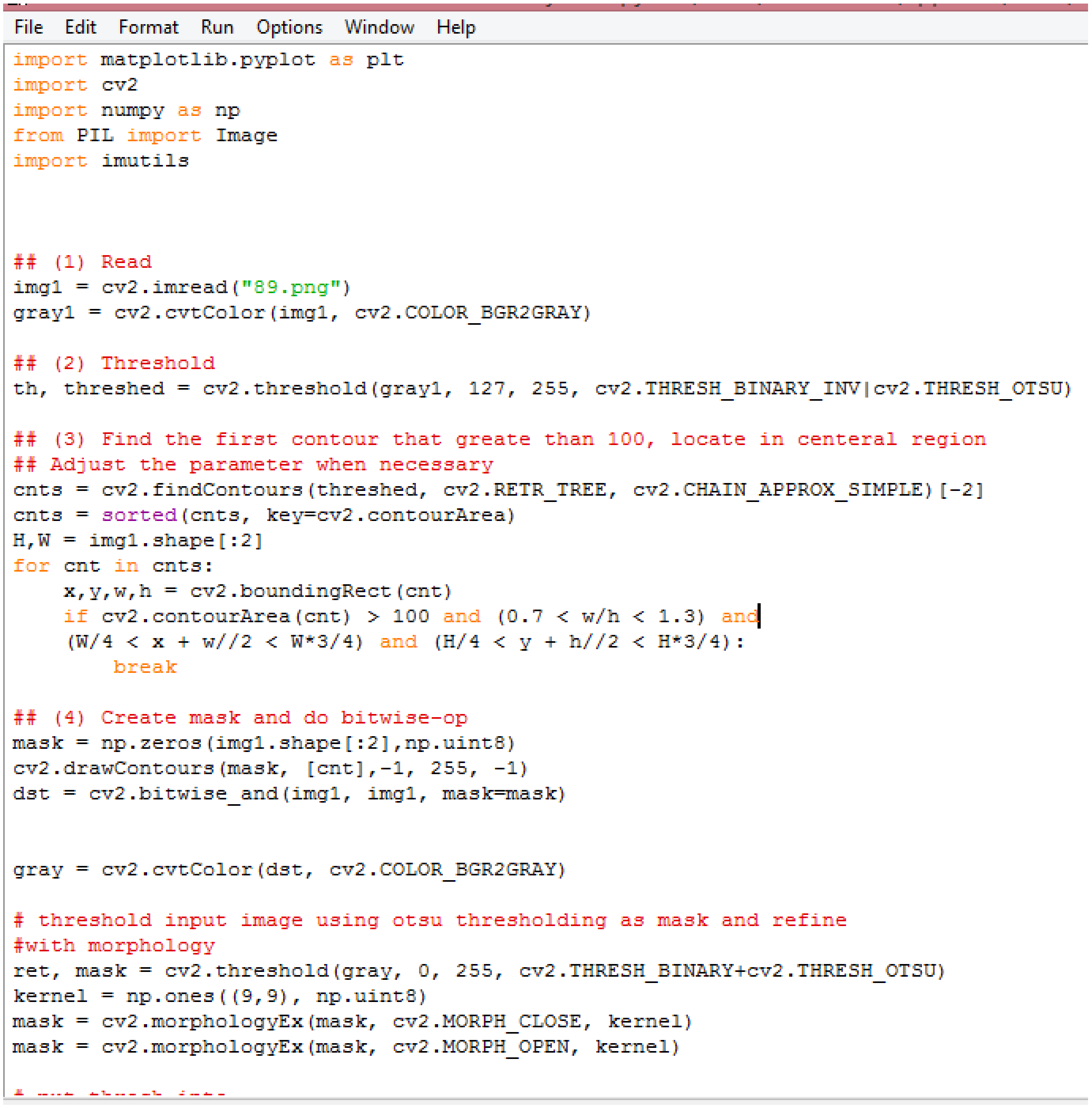


```
titles=['original image','crop with identifying the center','filtring',
        'thresholding','scratch','mask','difference','src','mask44']
images=[img1 ,result,sobelCombined,th1,line_image,mask33,difference,src,opening]

for i in range(9) :
    plt.subplot(3,3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])



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

Steps:

- Saving the titles of all images in a list named **titles**.
- Identify all the output images by their names in a list named **images**.
- Looping over the second list with identifying the number of columns and rows by using the function **plt.subplot ().**
- Showing the images with the function **plt.imshow ().**
- Integrating the titles of each image on the figure.

Recapitulation of the code



```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from PIL import Image
import imutils



## (1) Read
img1 = cv2.imread("89.png")
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

## (2) Threshold
th, threshed = cv2.threshold(gray1, 127, 255, cv2.THRESH_BINARY_INV|cv2.THRESH_OTSU)

## (3) Find the first contour that greate than 100, locate in centeral region
## Adjust the parameter when necessary
cnts = cv2.findContours(threshed, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)[-2]
cnts = sorted(cnts, key=cv2.contourArea)
H,W = img1.shape[:2]
for cnt in cnts:
    x,y,w,h = cv2.boundingRect(cnt)
    if cv2.contourArea(cnt) > 100 and (0.7 < w/h < 1.3) and
    (W/4 < x + w//2 < W*3/4) and (H/4 < y + h//2 < H*3/4):
        break

## (4) Create mask and do bitwise-op
mask = np.zeros(img1.shape[:2],np.uint8)
cv2.drawContours(mask, [cnt],-1, 255, -1)
dst = cv2.bitwise_and(img1, img1, mask=mask)


gray = cv2.cvtColor(dst, cv2.COLOR_BGR2GRAY)

# threshold input image using otsu thresholding as mask and refine
#with morphology
ret, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
kernel = np.ones((9,9), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
```

```
# put thresh into
result = dst.copy()
result = cv2.cvtColor(result, cv2.COLOR_BGR2BGRA)
result[:, :, 3] = mask


gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 200)
blurred = cv2.GaussianBlur(edged, (5, 5), 0)
thresh_1 = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]


# find contours in the thresholded image
cnts = cv2.findContours(thresh_1.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_NONE)
cnts = imutils.grab_contours(cnts)


# loop over the contours
for c in cnts:
        # compute the center of the contour
        M = cv2.moments(c)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        # draw the contour and center of the shape on the image
        cv2.drawContours(result, [c], -1, (0, 255, 0), 3)
        cv2.circle(result, (cX, cY), 7, (255, 255, 255), -1)
        cv2.putText(result, "center", (cX - 20, cY - 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
        #print( "the center's coordinates x and y are ",cX,"and", cY)


sobelX= cv2.Sobel(gray, cv2.CV_64F, 1,0)
sobelY= cv2.Sobel(gray, cv2.CV_64F, 0,1)

sobelX=np.uint8(np.absolute(sobelX))
sobelY=np.uint8(np.absolute(sobelY))

sobelCombined=cv2.bitwise_or(sobelX,sobelY)
```

```
gblur = cv2.GaussianBlur(sobelCombined, (1,1),0)

_, th1= cv2.threshold(sobelCombined,50,255, cv2.THRESH_BINARY)
cv2.imwrite('th1.png',th1)


img = cv2.imread('th1.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 50
high_threshold = 150
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)



rho = 1  # distance resolution in pixels of the Hough grid
theta = np.pi / 180  # angular resolution in radians of the Hough grid
threshold = 15  # minimum number of votes (intersections in Hough grid cell)
min_line_length = 120 # minimum number of pixels making up a line
max_line_gap =20   # maximum gap in pixels between connectable line segments
line_image = np.copy(img) * 0  # creating a blank to draw lines on the image

# Run Hough on edge detected image
# Output "lines" is an array containing endpoints of detected line segments

lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                    min_line_length, max_line_gap)

for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(0,0,255),2,4)


# Draw the lines on the  image
lines_edges = cv2.addWeighted(img, 0.8, line_image, 1, 0)
```

```
line_image2 = cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB)

hsv = cv2.cvtColor(line_image2, cv2.COLOR_BGR2HSV)
lower_range = np.array([110,50,50])
upper_range = np.array([130,255,255])
mask3 = cv2.inRange(hsv, lower_range, upper_range)
```

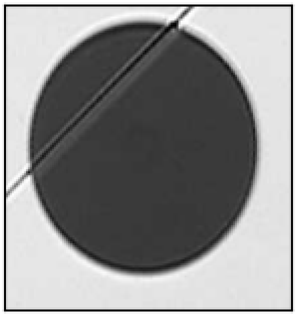
Recapitulation of the code

```
img=cv2.circle(line_image,(cX,cY), 13, (255,0,0), 2)
img=cv2.circle(line_image,(cX,cY), 58, (0,255,0), 2)
img=cv2.circle(line_image,(cX,cY), 68, (100,150,200), 2)


line_image3 = cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB)
mask33 = cv2.cvtColor(mask3, cv2.COLOR_BGR2RGB)
ig1= cv2.resize(line_image3,(300,229))

ig2= cv2.resize(mask33,(300,229))

# compute difference
difference = cv2.subtract(ig1, ig2)

# color the mask red
Conv_hsv_Gray = cv2.cvtColor(difference, cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(Conv_hsv_Gray, 0, 255,cv2.THRESH_BINARY_INV |cv2.THRESH_OTSU)
difference[mask != 255] = [0, 0, 255]
i= cv2.resize(difference,(229,229))
cv2.imwrite('difference.png',i)


src = cv2.imread('difference.png', cv2.IMREAD_COLOR)

#Transform source image to gray if it is not already
if len(src.shape) != 2:
    gray66 = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
else:
    gray66 = src

_, th11= cv2.threshold(gray66,50,255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(th11, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
hierarchy = hierarchy[0]

for i, c in enumerate(contours):
        if hierarchy[i][2] < 0 and hierarchy[i][3] < 0 and cv2.arcLength(c,False)> 300 :
                 (x,y),radius = cv2.minEnclosingCircle(c)
                 center =  (int(x),int(y))
                 radius = int(radius)
                 img = cv2.circle(src,center,radius,(0,255,0),2)

        elif  hierarchy[i][2] < 0 and hierarchy[i][3] < 0 and cv2.arcLength(c,False)<60:
                (x,y),radius = cv2.minEnclosingCircle(c)
                center = (int(x),int(y))
                radius = int(radius)
                img = cv2.circle(src,center,radius,(0,255,0),2)

        elif cv2.arcLength(c,True)> 300 or  cv2.arcLength(c,True)<100 :

             (x,y),radius = cv2.minEnclosingCircle(c)
             center = (int(x),int(y))
             radius = int(radius)
             img = cv2.circle(src,center,radius,(255,255,0),2)
```

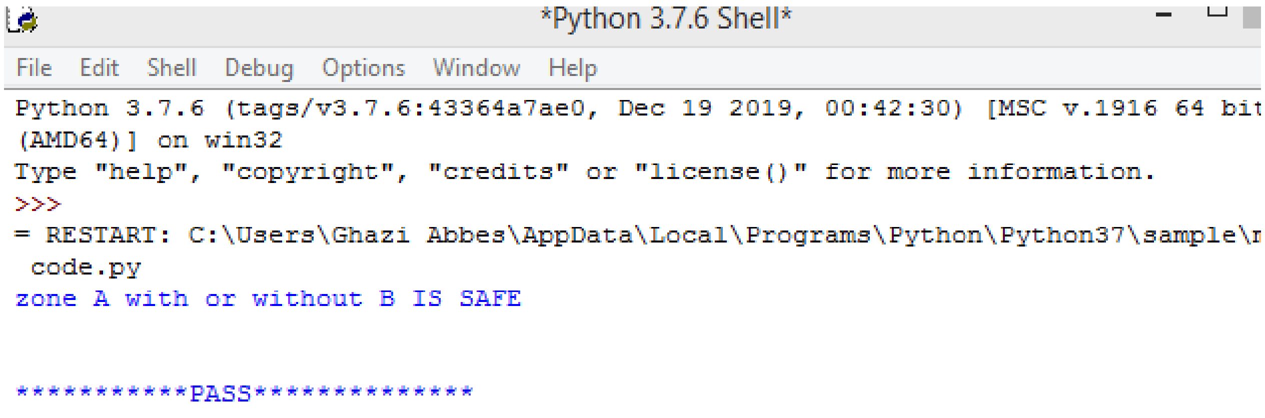
Recapitulation of the code

```
src2 = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)
hsv2 = cv2.cvtColor(src2, cv2.COLOR_BGR2HSV)
lower_range = np.array([22, 93, 0])
upper_range = np.array([45, 255, 255])
mask44 = cv2.inRange(hsv2, lower_range, upper_range)
kernal = np.ones((3,3), np.uint8)
closing = cv2.morphologyEx(mask44, cv2.MORPH_CLOSE, kernal)
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernal)
contours, hierarchy = cv2.findContours(opening, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
for i, c in enumerate(contours):

        perimeter = cv2.arcLength(c,True)

        if  70 <perimeter <200 or  360 <perimeter <400 :
             print('zone A with or without B IS SAFE  ')
             print('\n')
             print ('************PASS**************')
             break
        elif 70 <perimeter <200 or  410 <perimeter <490 :
             print ('zone A and zone C  are safe ')
             print('\n')
             print ('*************PASS*************')
             break
        elif 360 <perimeter <490 or  410 <perimeter <490 :
             print ('zone A is not safe')
             print('\n')
             print ('*************FAIL**************')
             break
        else :
             print ('*************FAIL**************')
             break
```

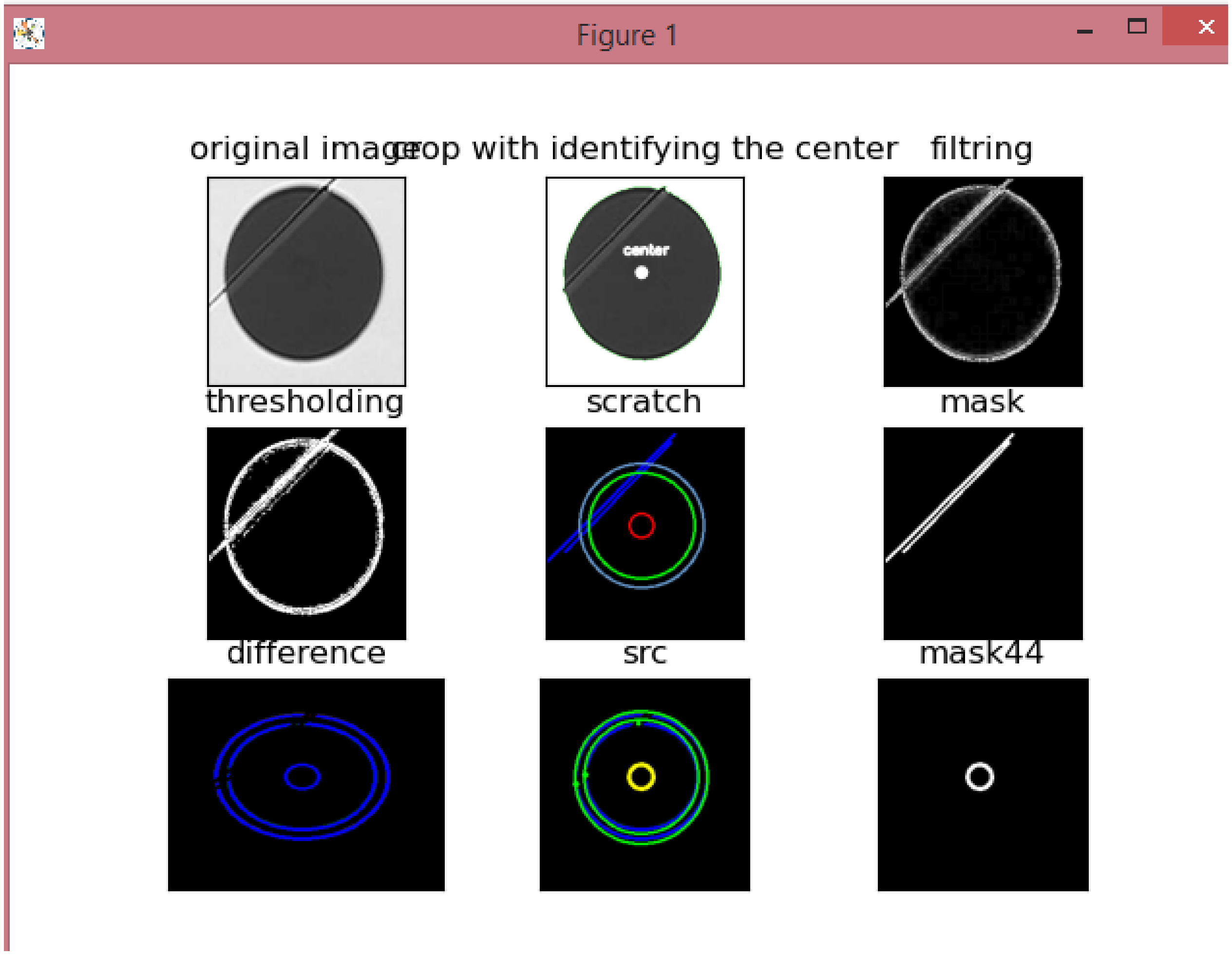
```
if cv2.countNonZero(opening) == 0:
    print('*************FAIL****************')

titles=['original image','crop with identifying the center','filtring',
        'thresholding','scratch','mask','difference','src','mask44']
images=[img1 ,result,sobelCombined,th1,line_image,mask33,difference,src,opening]

for i in range(9) :
    plt.subplot(3,3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])



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

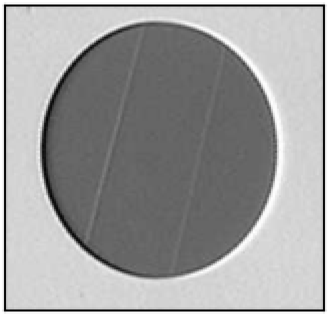

Input image

Examples

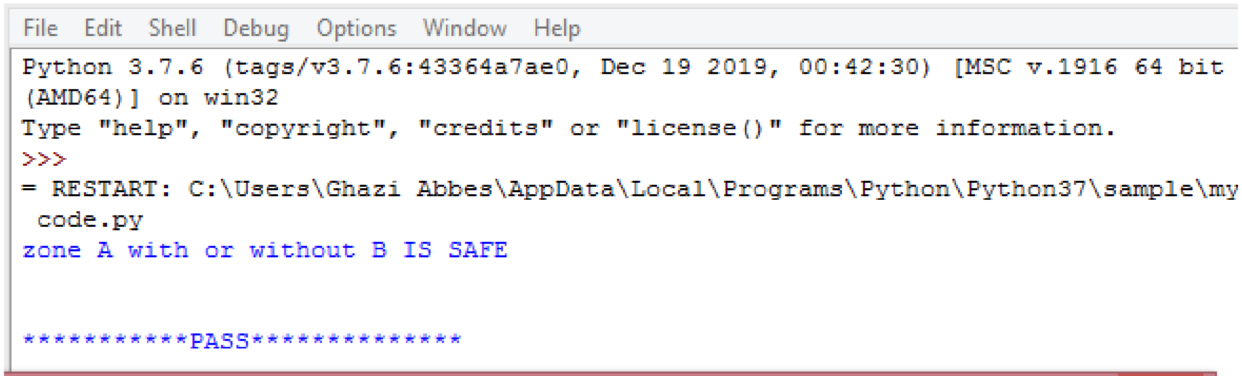


```
*Python 3.7.6 Shell*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample\r
 code.py
zone A with or without B IS SAFE


***********PASS**************
```

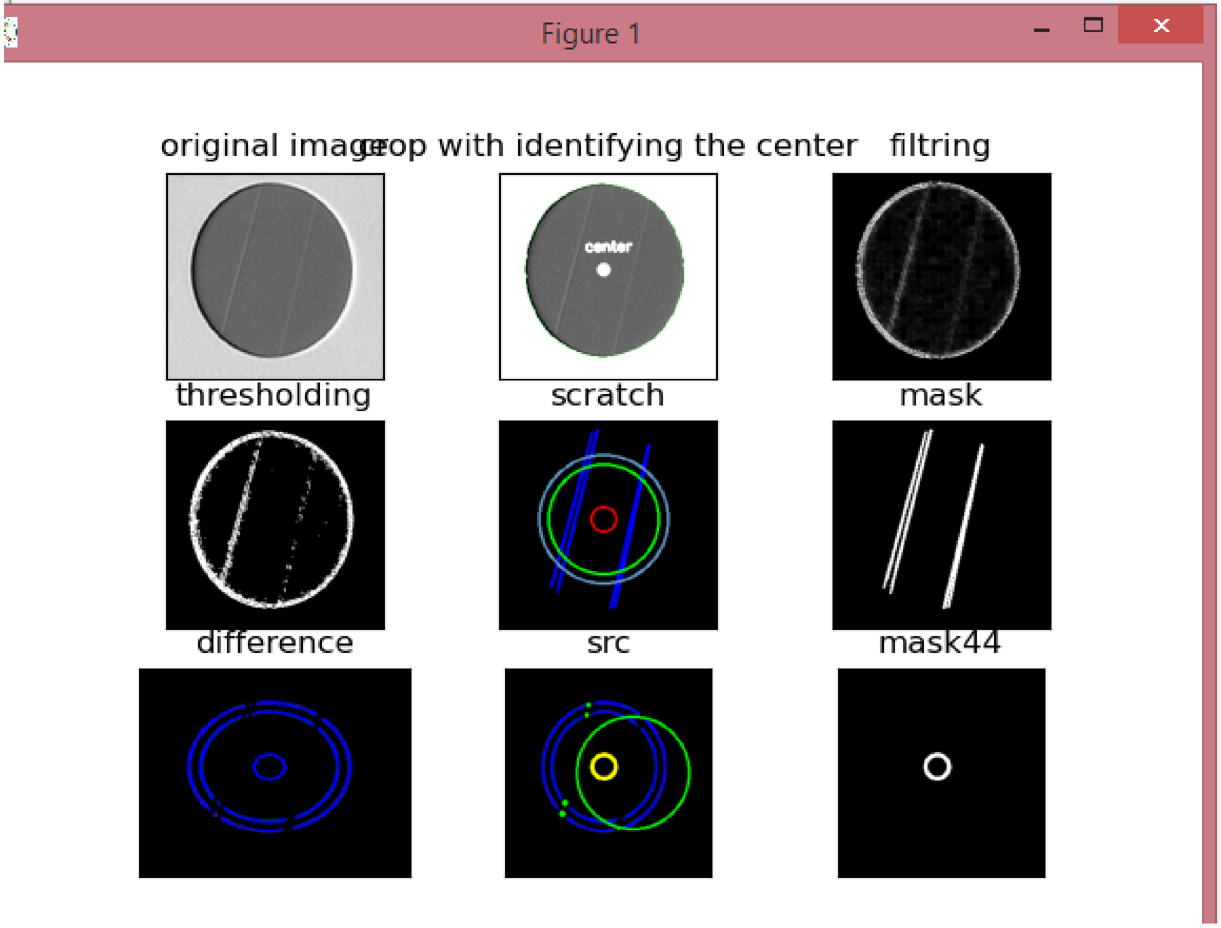




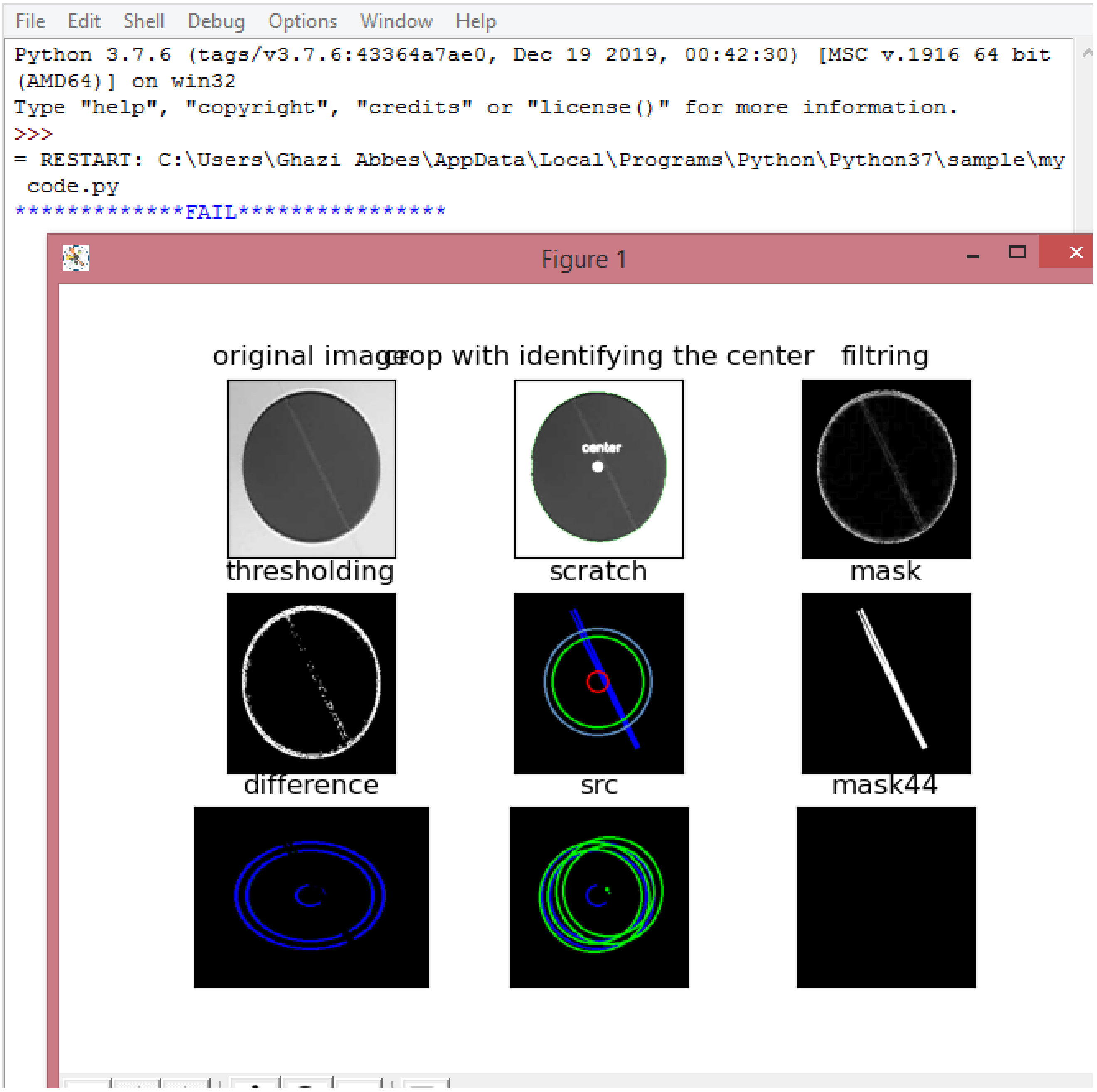
Input image



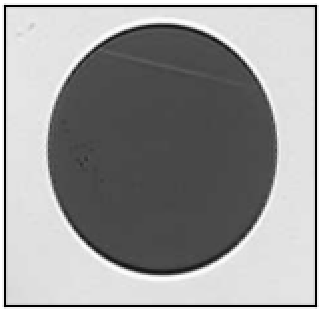
```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample\my
 code.py
zone A with or without B IS SAFE


***********PASS**************
```

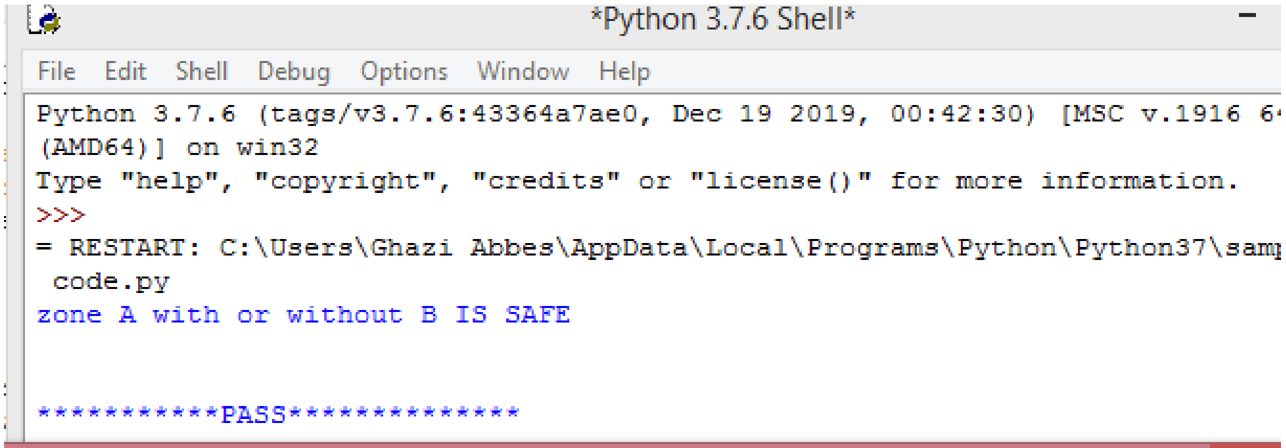


Input image

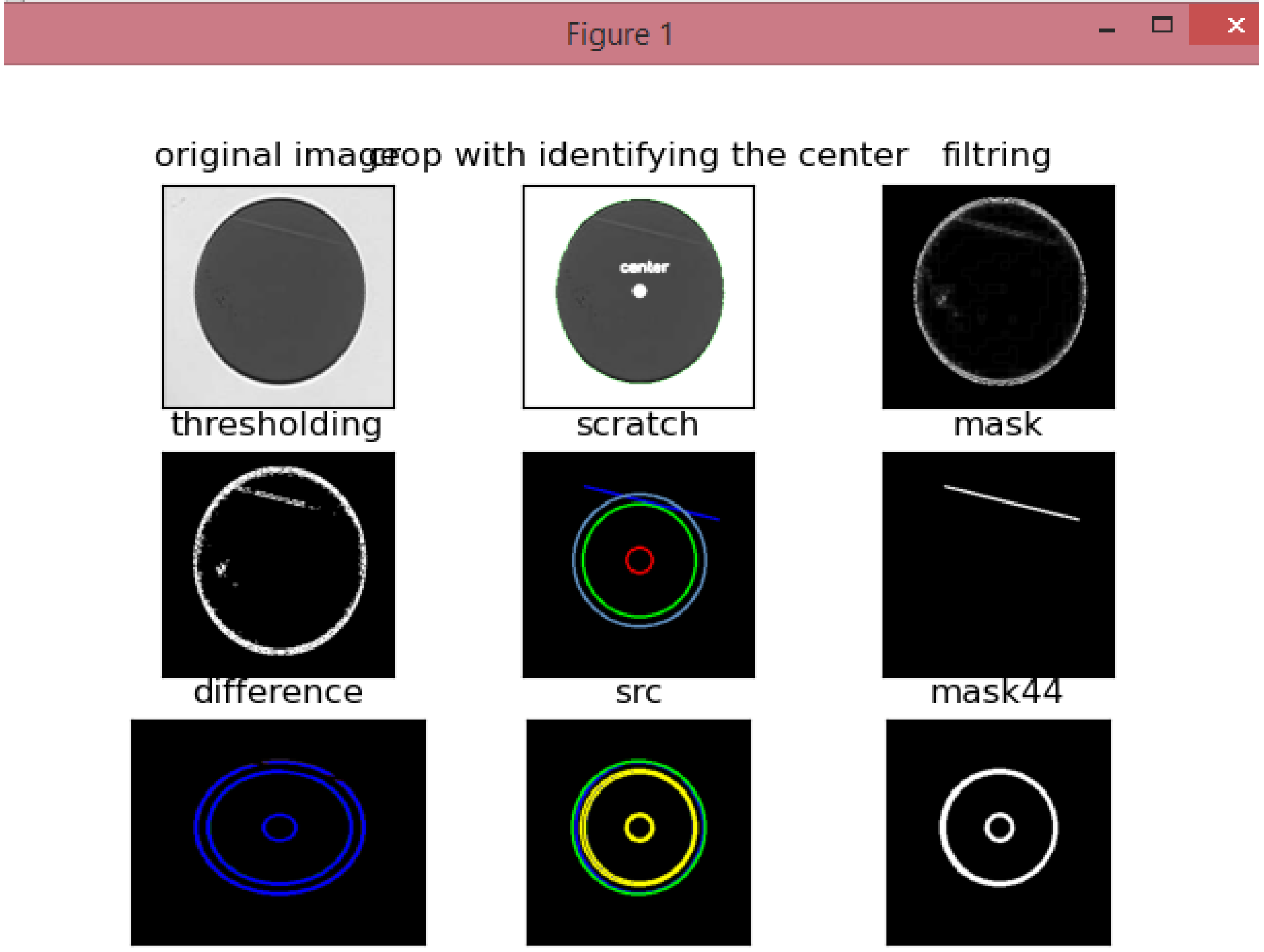


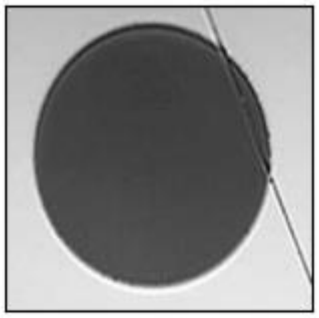
```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample\my
 code.py
*************FAIL****************
```


Input image

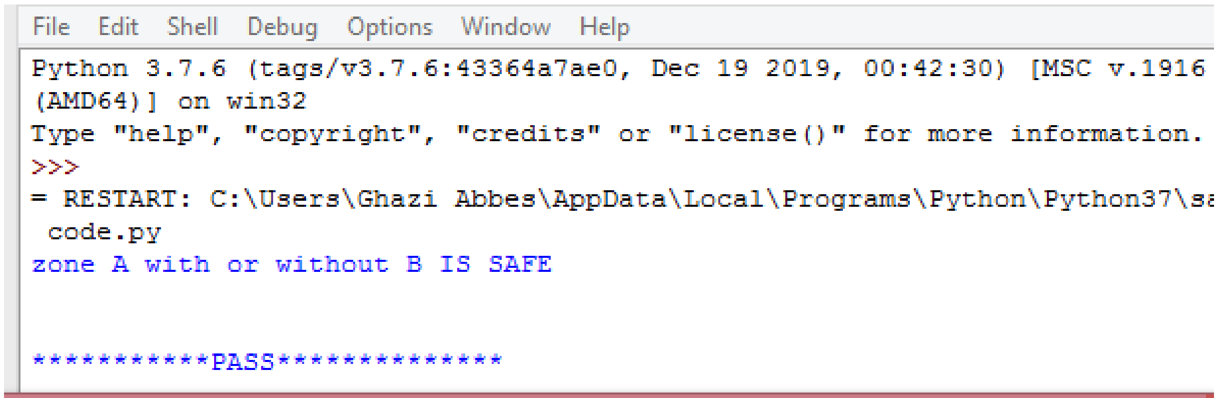


```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 6
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sam
 code.py
zone A with or without B IS SAFE


***********PASS**************
```

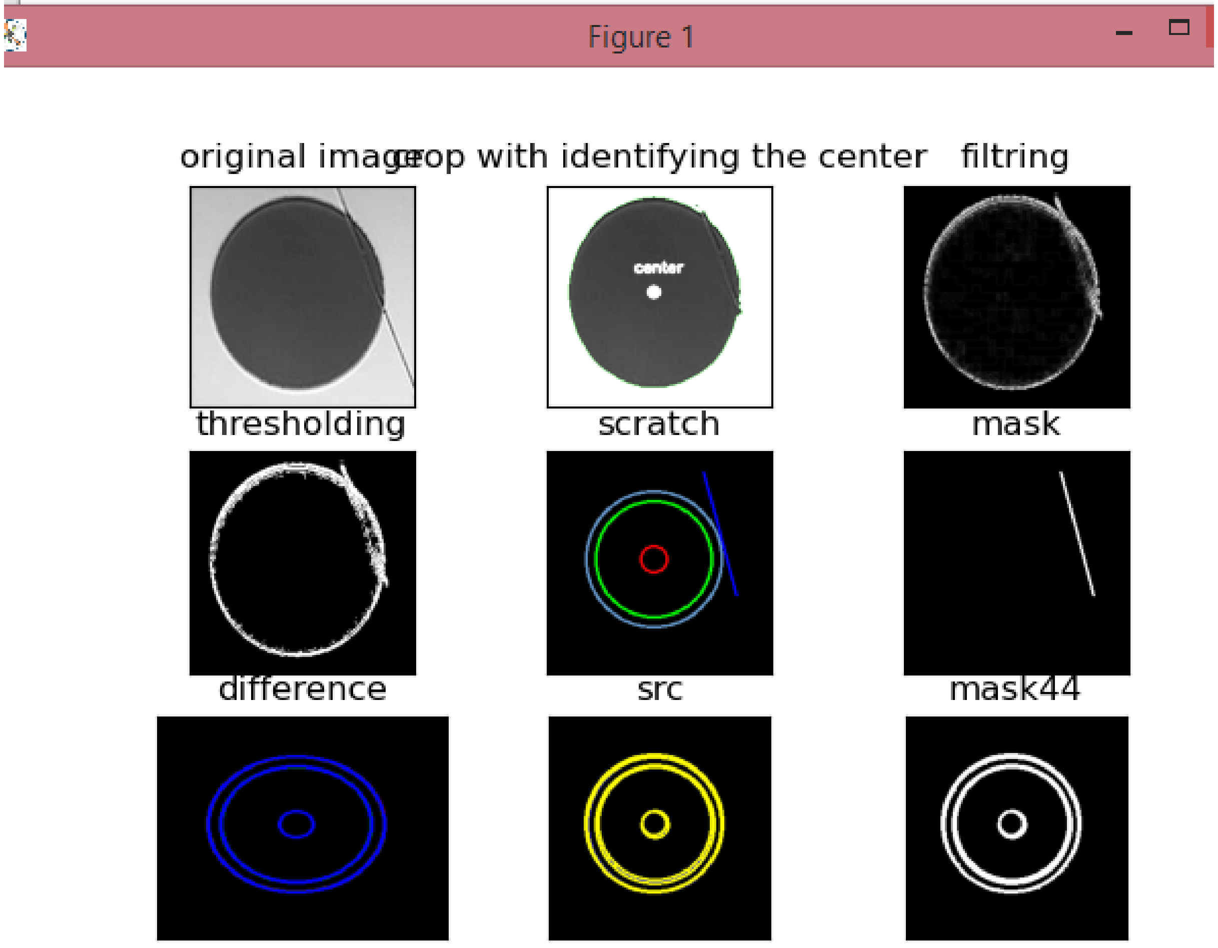


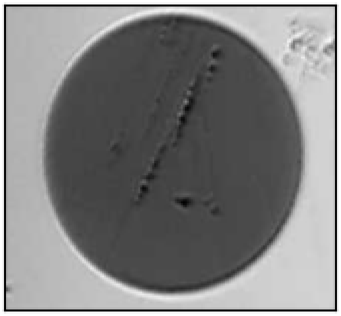


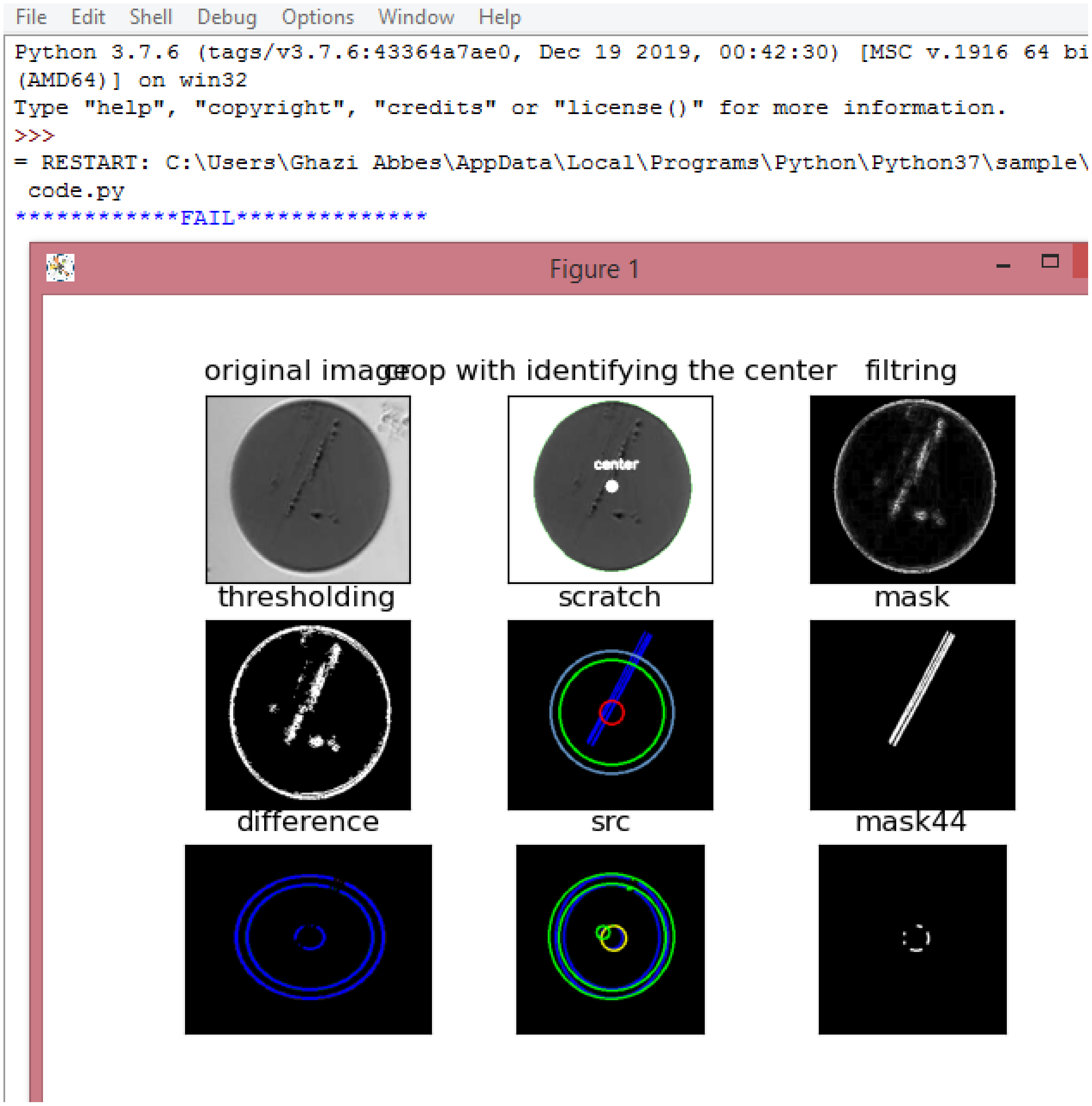
Input image



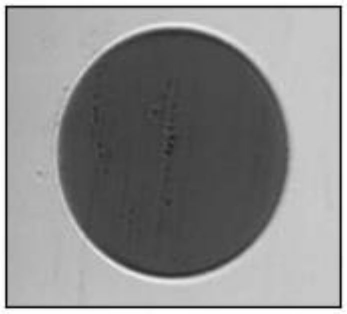
```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sa
 code.py
zone A with or without B IS SAFE


************PASS***************
```



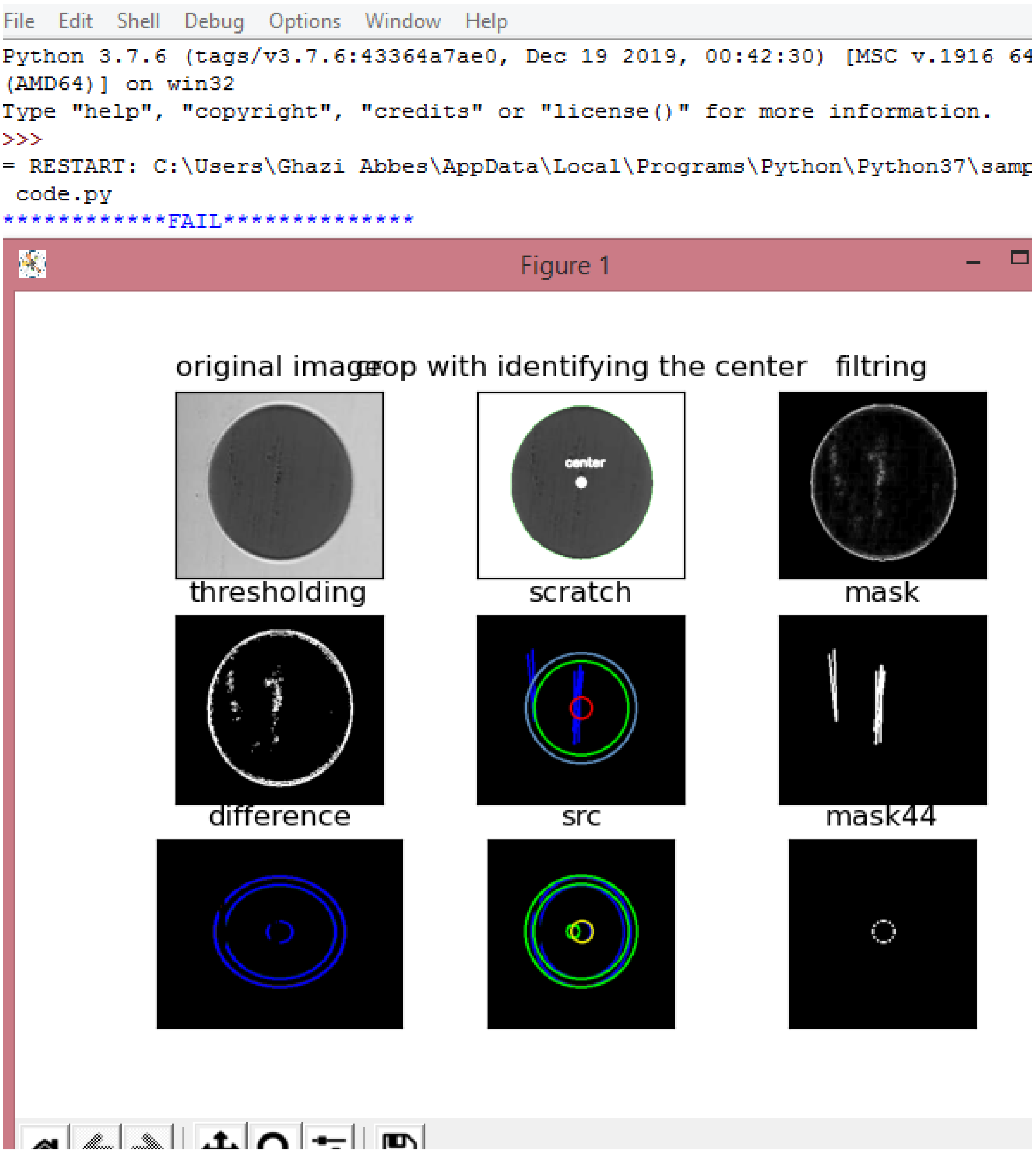

Input image



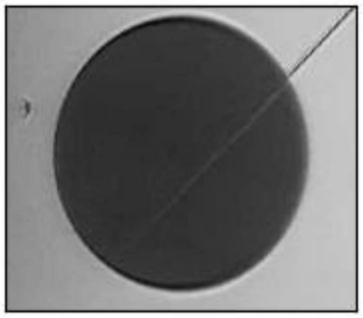
```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bi
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample\
 code.py
************FAIL**************
```

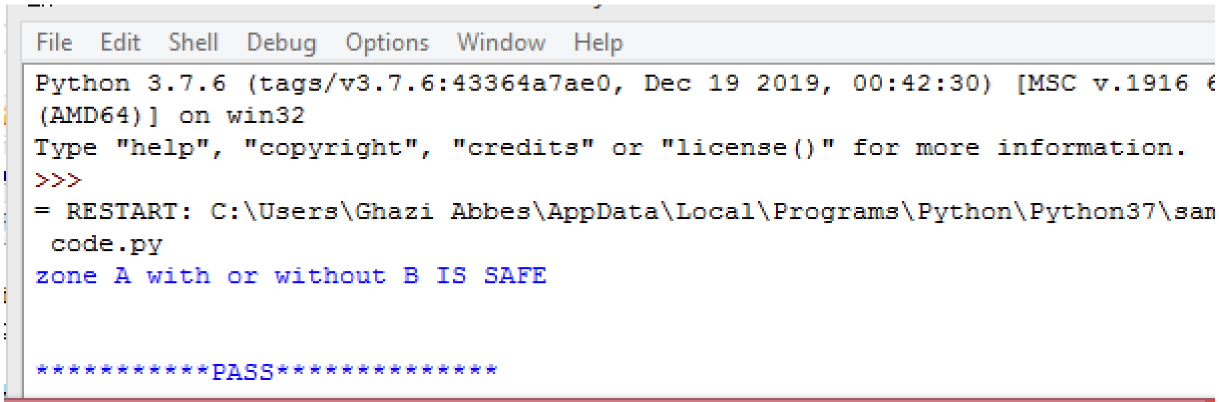


Input image

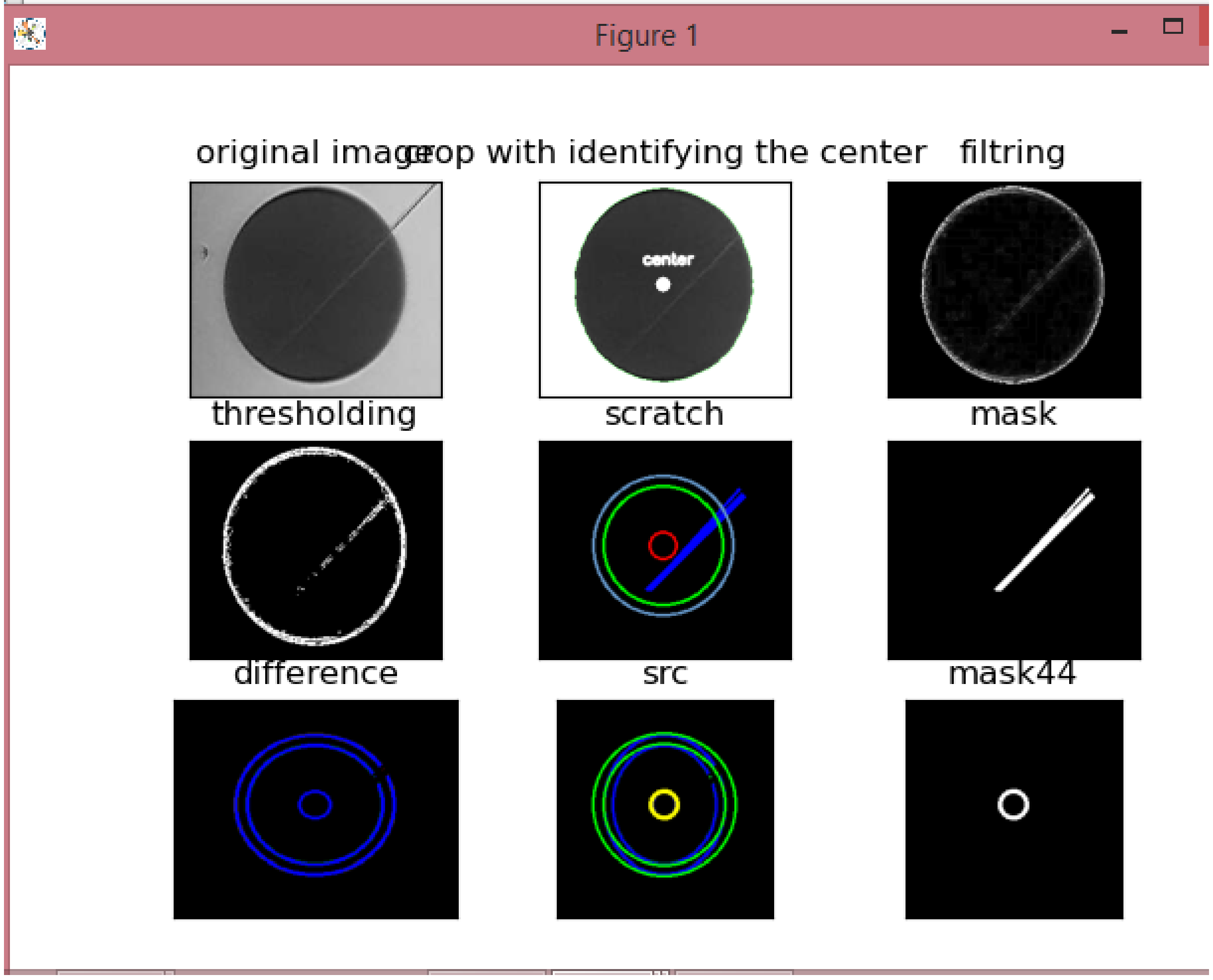


```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\samp
 code.py
************FAIL**************
```

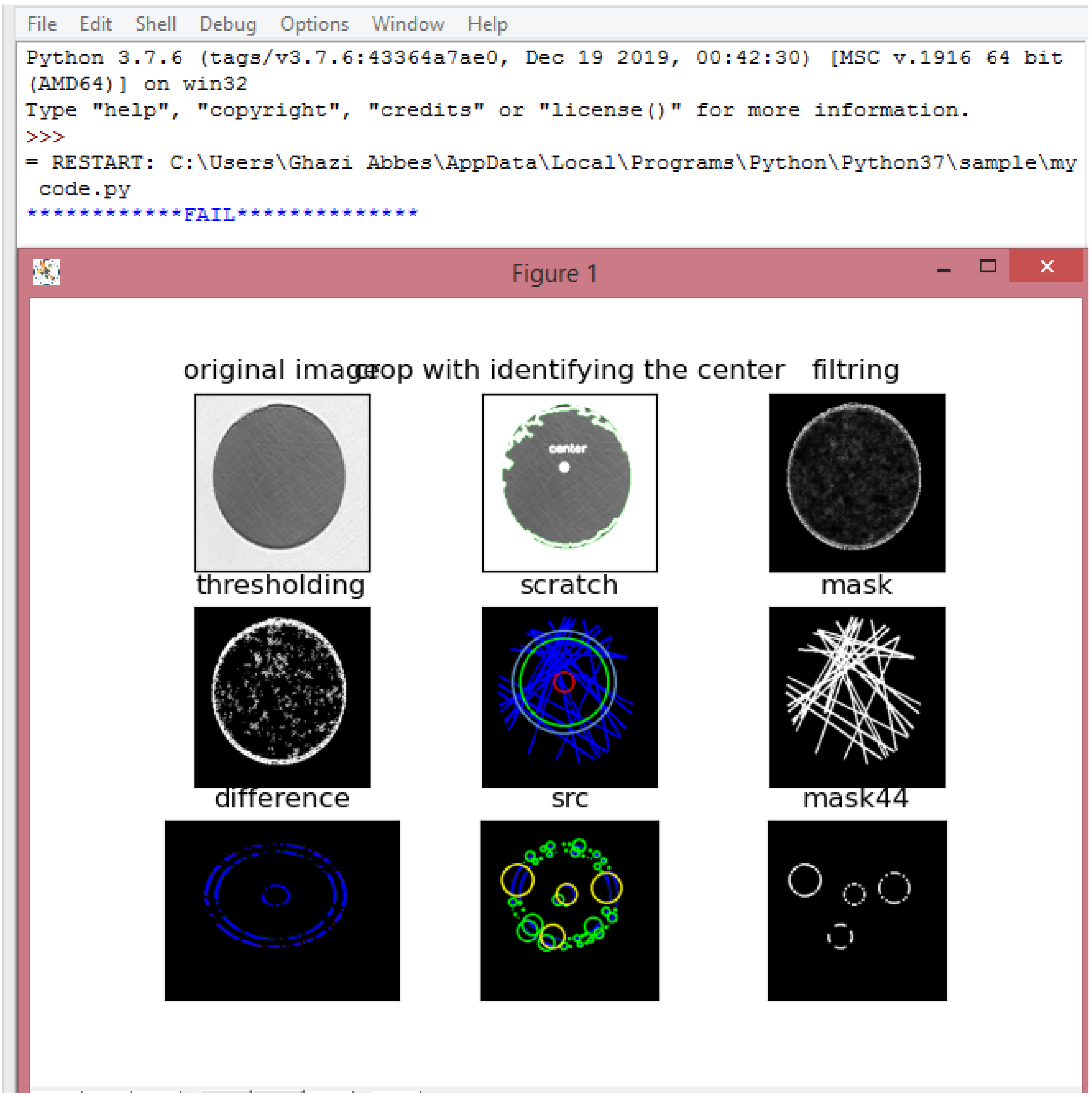

Input image



```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 6
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sam
 code.py
zone A with or without B IS SAFE


***********PASS***************
```

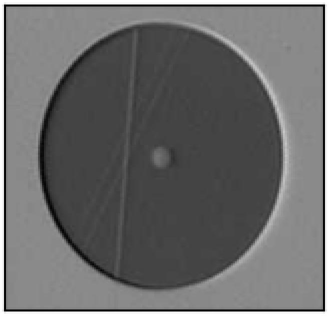


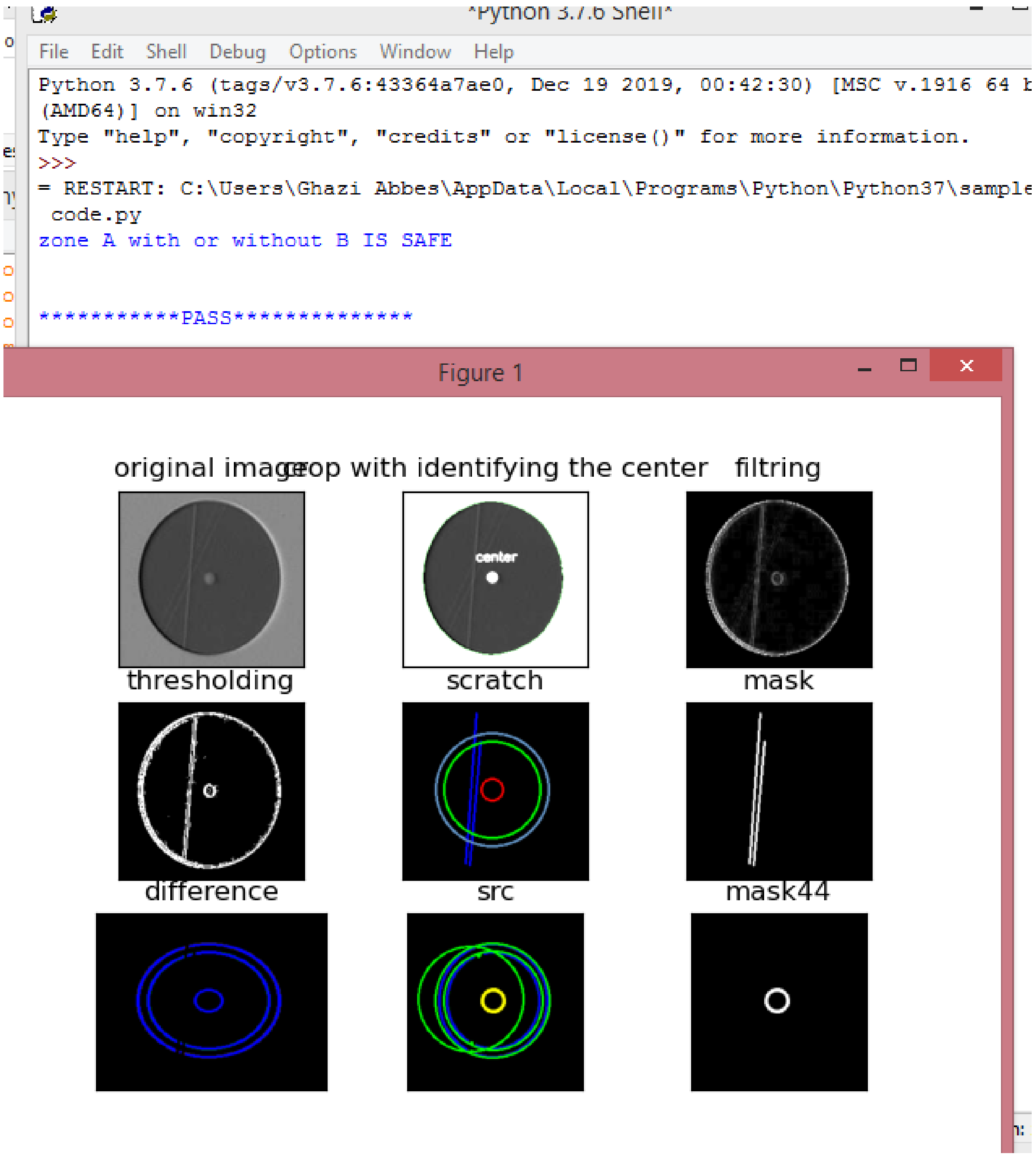
Input image


File Edit Shell Debug Options Window Help

```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample\my
 code.py
************FAIL**************
```

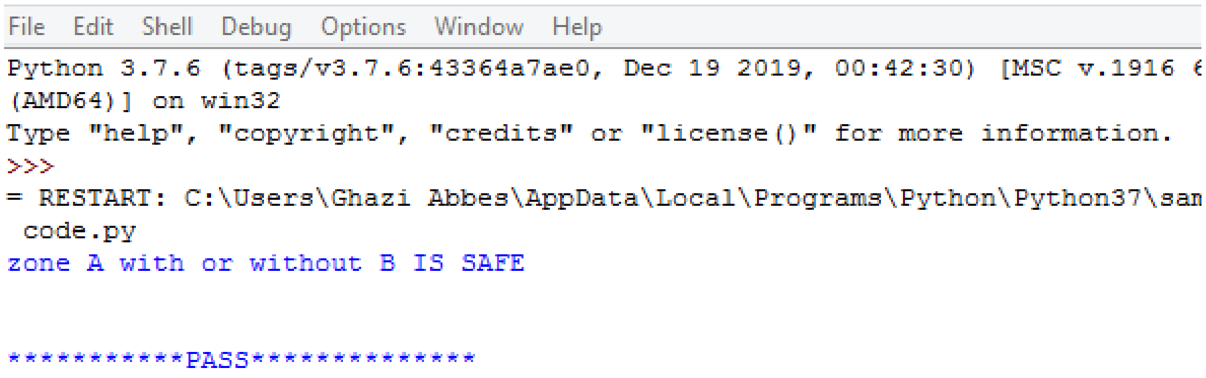
Examples


Input image

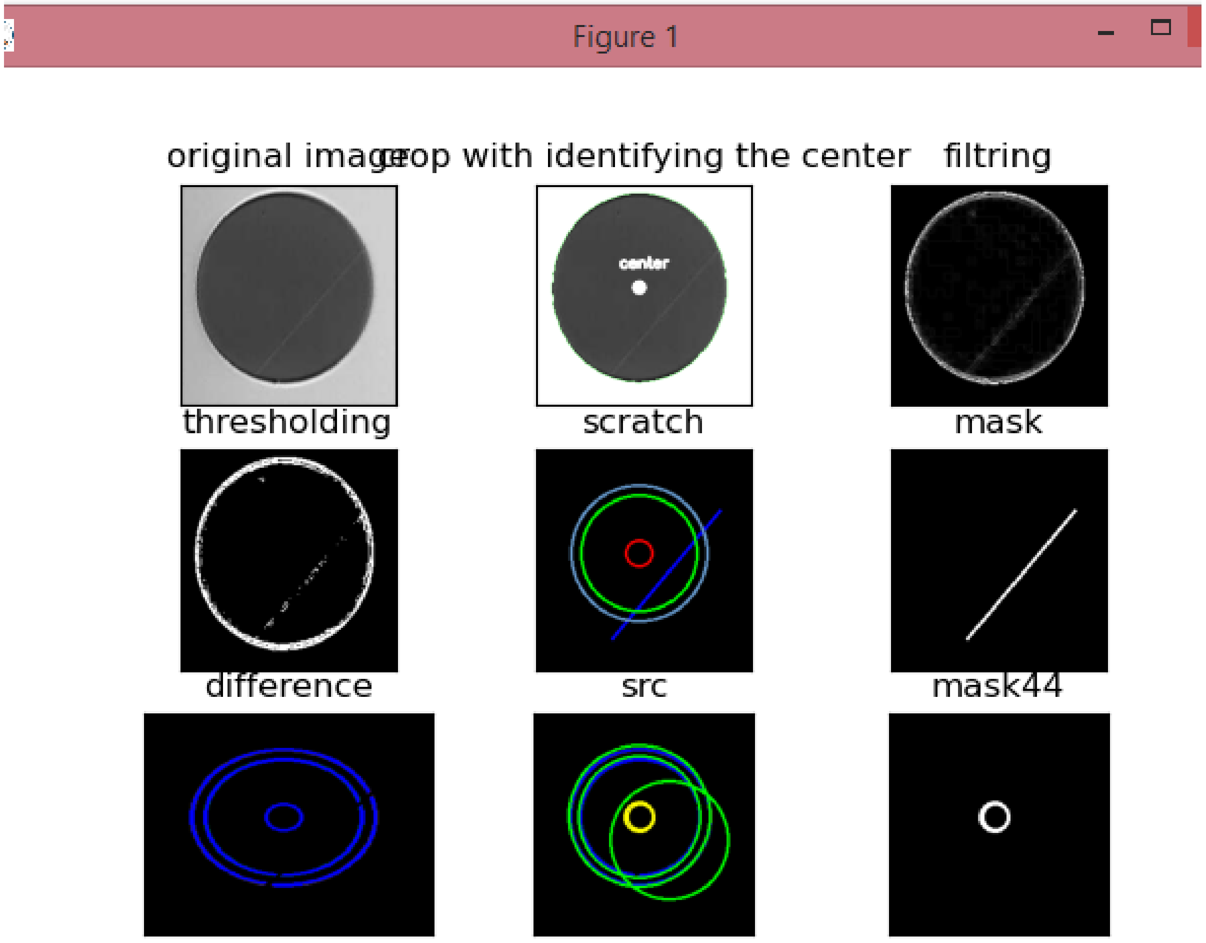


```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 b
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sample
 code.py
zone A with or without B IS SAFE


***********PASS***************
```

Input image



```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 6
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ghazi Abbes\AppData\Local\Programs\Python\Python37\sam
 code.py
zone A with or without B IS SAFE


***********PASS***************
```



Conclusion

On the whole, this internship was a useful experience. I have gained new knowledge, I achieved several of my learning goals. I got insight into professional practice. I learned the different facets of working with computer vision.

I was be able during my internship to:

- Discover the world of computer vision.
- Apply many functions on my image.
- Detect easily the scratch and identify in wish zone it is located.
- Make the end face testing more efficient, accurate and reliable.
- Learn a new computing technology for automatic visual inspection that takes place along the production line.

The internship was also good to find out what my strengths and weaknesses are. This helped me to define what skills and knowledge I have to improve in the coming time.