Sequence 4 exercises

The aim of exercise 4.1 s to build a **binary** image *B* indicating if a pixel is an edge pixel or not. As a consequence, *B(i,j)* is defined as (see Fig.1 for illustration):

$$B(i,j) = \begin{cases} 1 & if\ (i,j) is\ an\ edge\ pixel \\ 0 & otherwhile \end{cases}$$

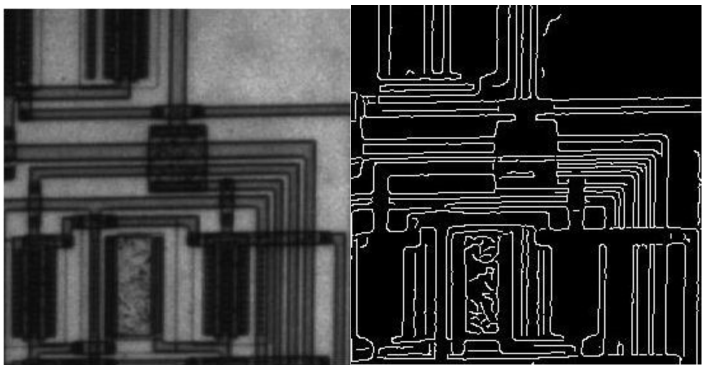


Fig.1: left, *circuit* image; right, binary image with edge pixels of the *circuit* image

4.1 Edges detection via luminance gradient analysis (Homework)

A contour is strongly related to the fast spatial variations of the luminance function. The first intuitive way to extract contour pixels is to deal with the gradient of the luminance function (first derivative of the luminance function) that exhibits high values at edge pixels. The idea is then to detect the highest values of the luminance gradient magnitude and the main issue is to be able to define quantitatively what a high gradient magnitude value is.

Download the file *Sequence4_part1_code.py*. Let's consider the images *toyobjects.png, alumgrns.tif* and *muscle.tif* presented in Fig.2.

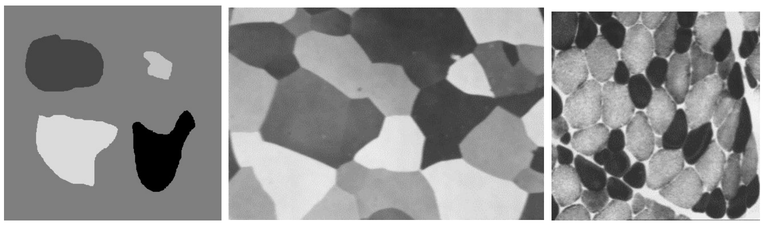


Fig.2: left, *toyobjects* image, middle, *alumgrns* image and right, *muscle* image

Toyobjects is a very simple image in which contours should not be too difficult to detect. On the contrary, *alumgnrs* and *muscle* are more challenging images for contours extraction.

4.1.1 Compute the horizontal *Gx* and vertical *Gy* luminance gradients of each image by considering the Sobel operator (cf. *sobel_h() and sobel_v()* functions). Display the matrices Gx and Gy. Comment the results.

4.1.2 Compute for each image the gradient magnitude *G* and orientation defined as:

$$G(x,y) = \sqrt{Gx(i,j)^2 + Gy(i,j)^2}$$

$$\theta(x,y) = arctang(\frac{G_y}{G_x})$$

Display for each image the gradient magnitude map and the orientation map.

4.1.3 Edge pixels detection: 1st method. Build the B image by selecting the pixels whose gradient magnitude is between the max gradient value and T% of the max gradient value. Try different values of the T threshold so that the obtained binary image looks good to you (all the edge pixels being detected). Please report the tests you did for each of the three images, display your best results and indicate the threshold value. What are the conclusions of your experiments?

4.1.4 Edge pixels detection, 2nd method: hysteresis thresholding. Two thresholds are going to be considered, a low threshold T_L and a high threshold T_H. Pixels with gradient magnitude lower than T_L are definitely non-edge pixels. On the contrary, pixels with gradient magnitude higher than T_H are definitely edge pixels. For pixels with gradient magnitude in between, there are going to be considered as edge pixels only if they are connected to other edge pixels. Otherwise they are labelled as non-edge pixels. Implement this second method and choose appropriate T_L and T_H values in order to improve contours detection in image *coffee*. Define the two thresholds by considering a % of the highest gradient magnitude value of the image.

4.1.5 Conclude about the pros and cons related to methods detecting edges by analyzing the luminance gradient

4.2 Edge detection via Canny filtering (Classwork)

In order to refine the edge detection of the three images, implement the whole Canny algorithm whose steps are:

- ✓ Noise reduction via Gaussian filtering
- ✓ Gradient magnitude and orientation computation
- ✓ Non-maximum suppression for contour "thinering"
- ✓ Hysteresis thresholding and connectivity testing

For the 3rd step, select the 2 neighbours in the gradient direction

```
If (0<=theta<22.5) or (157.5<theta(i,j)<=180)
        first_neigh = grad(i, j+1)
        second_neigh = grad(i, j-1)
…
```

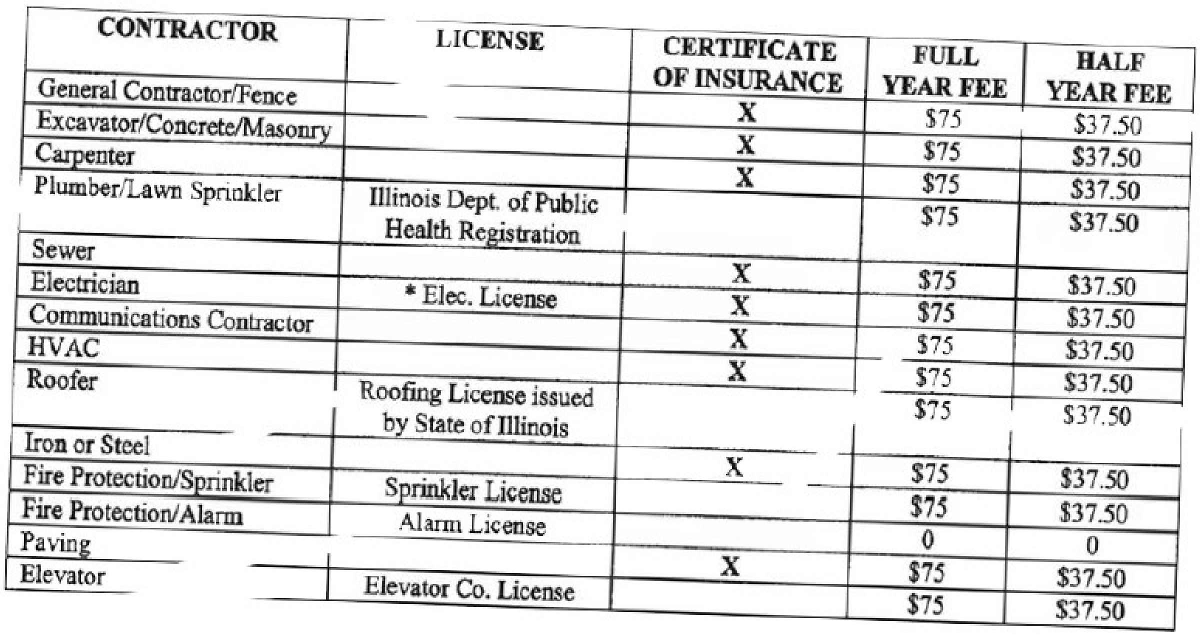
Then suppress the points with non-maximal grad value in the gradient direction

```
If (grad(i,j)>=first_neigh) and (grad(i,j)>=second_neigh)
        new_grad_img(i,j) = grad(i,j) (the gradient value is kept because locally max
Else
        new_grad_value(i,j) = 0
```

Study on the *muscle* image the influence of the three parameters (Gaussian filter variance, high and low thresholds) and give the extracted contours.

4.3 Document image restoration (Classwork)

Download the file *Sequence4_part3_code.py*.
Let's consider the following image *insurance_form.jpg.*



CONTRACTOR	LICENSE	CERTIFICATE OF INSURANCE	FULL YEAR FEE	HALF YEAR FEE
General Contractor/Fence		X	$75	$37.50
Excavator/Concrete/Masonry		X	$75	$37.50
Carpenter		X	$75	$37.50
Plumber/Lawn Sprinkler	Illinois Dept. of Public Health Registration		$75	$37.50
Sewer		X	$75	$37.50
Electrician	* Elec. License	X	$75	$37.50
Communications Contractor		X	$75	$37.50
HVAC		X	$75	$37.50
Roofer	Roofing License issued by State of Illinois		$75	$37.50
Iron or Steel		X	$75	$37.50
Fire Protection/Sprinkler	Sprinkler License		$75	$37.50
Fire Protection/Alarm	Alarm License		0	0
Paving		X	$75	$37.50
Elevator	Elevator Co. License		$75	$37.50

During faxing, the form has suffered some image distortions: it is slightly rotated and some portions of the table's horizontal and vertical lines have been erased.
The main goal here is to restore the image as much as possible.

- In a first step, apply the Hough transform in order to estimate the rotation angle of the image (cf. *hough_line()* and *hough_line_peaks()* functions). The *hough_line()* function requires as input a binary image with edge pixels. To obtain such an image, just binarize the original image with an appropriate threshold.
- Then applied the estimated reverse rotation in order to have an horizontally aligned image (look at the *rotate()* function of the scikit-image.transform module)
- Finally, restore the broken lines, some morphological operations are proposed in the file *Sequence4_part3_code.py*.
- Display the final restored image.