Image Processing

Report on Laboratory Session #3

Giacomo Deodato
Alessandro Patti

October 16-23, 2017

Part #1

Pre-processing: de-noising

Step 1: load the image and artificially add some noise

We used the function *randn()* to create a noise matrix to add to the image.

```
noise = randn(size(img,1),size(img,2));
noise_strength = 64;
img_n = img+uint8(noise_strength*noise);
```

To easily change the noise intensity we multiplied the noise matrix with the variable *noise_strength*.

Step 2: de-noise the image

We calculated the filtered image with the 3 different methods and plot them to compare results. The size of the filter is set using the variable *mark_size*, for which, after some tests, we found an optimal value of 5.

```
mask_size=5;
img_fw=wiener2(img_n, [mask_size mask_size]);
img_fm=medfilt2(img_n, [mask_size mask_size]);
img_fa=imfilter(img_n,fspecial('average',mask_size));
```

The best result has been achieved with the Wiener filter because it better preserves the edges.
The Wiener filter tailors itself to the local image variance: when the variance is large, it performs little smoothing; when the variance is small, it performs more smoothing.
This approach often produces better results than linear filtering like averaging, and it is more selective, it preserves edges and other high-frequency parts of an image.
Moreover, it performs better than the median filter because it takes into consideration both the mean and the standard deviation of the neighbourhood of each pixel instead of just the median.

Part #2

Processing: low level feature detection

Step 3: highlight edges

Gradient filter

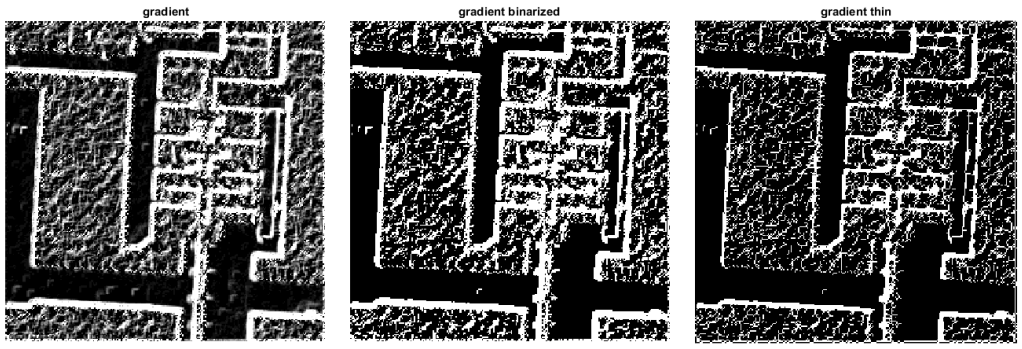
The first technique we used to highlight edges is the gradient filter.

First we created the gradient masks over the x and y axis and we applied them to the image using the *imfilter()* function. To obtain the final gradient the two components have been summed:

$$|\bigtriangledown f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Then we used *im2bw()* to binarize the image with threshold computed by the function *greythresh()*. Finally, we used *bwmorph()* to perform morphological operations to make the edges clearer.

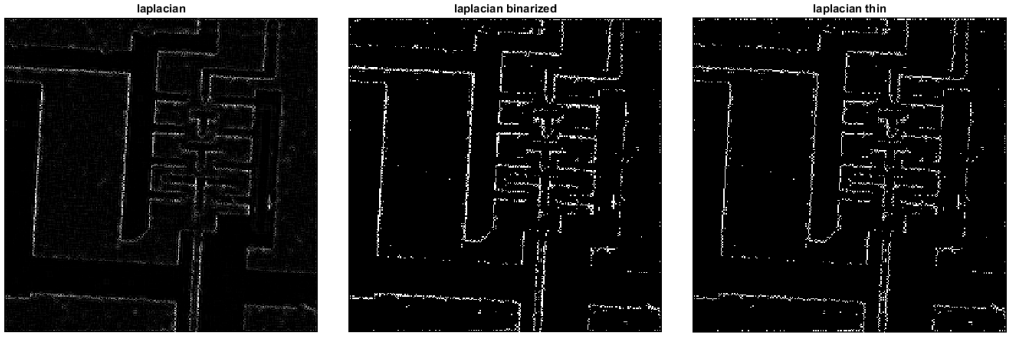
```
Gx=zeros(3,3);
Gx(2,1)=1; Gx(2,3)=-1; %create x-axis gradient mask
Gy=zeros(3,3);
Gy(1,2)=1; Gy(3,2)=-1; %create y-axis gradient mask

gradientx=imfilter(img_f,Gx); %gradient over x-axis direction
gradienty=imfilter(img_f,Gy); %gradient over y-axis direction

gradient=uint8(sqrt(double(gradientx.^2+gradienty.^2)));
gradient=im2bw(gradient, graythresh(gradient));
gradient=bwmorph(gradient,'thin');
```



Laplacian filter

The second filter we used is the Laplacian. Zero crossings of the Laplacian are more accurate than gradient at localizing edges as can be seen after applying again *im2bw()* and *bwmorph()*.

```
laplacian=imfilter(img_f,fspecial('laplacian'));
laplacian=im2bw(laplacian,graythresh(laplacian));
laplacian=bwmorph(laplacian,'thin');
```

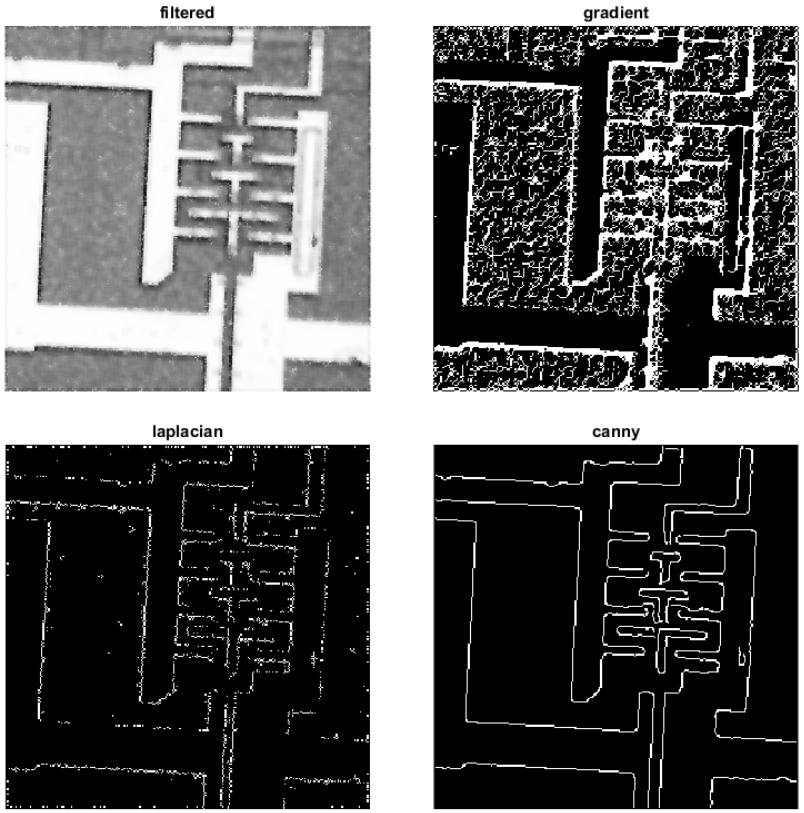


Canny edge detector

The last technique we used is the Canny edge detector. After some tests we found an optimal threshold value of 0.4. This method does not require neither binarization nor morphological operations.

```
canny=edge(img_f,'canny', 0.4);
```

Comparing the different results it can be immediately seen how the Canny's method gives a clearer result.



Step 4: compute the Radon transform

To compute the Radon transform we used the function *radon()*. The relation between the transform and the edge image has been analysed by means of the function *interactiveLine()* on step 5.

How does the Radon transform relate to the Hough trasform for lines?

The Radon transform and the Hough transform are strictly related when doing line detection, we could say that the former is a simplified version of the latter.
The main difference between the two transformations is in the mathematical formulation, while the Hough transform is a discrete algorithm, the Radon transform is defined as an integral.
The Radon transform is also easier to understand from a conceptual point of view: it projects the whole image over an axis rotating from 0 to 179 degrees. The Hough transform instead takes one or more features (pixels) from the image and associates it to one or more parametric curves in the Hough space.

Why the sum over any column of the Radon transform is always the same?

The sum corresponds to the amount of information of the image, it is constant because each column contains the entire projection. For example, if we make the Radon transform of an horizontal line we'll notice that the vector corresponding to 0 degrees has uniform distributed values while the column corresponding to 90 degrees has a unique peak whose value is equal to the sum of all the values of the horizontal projection.

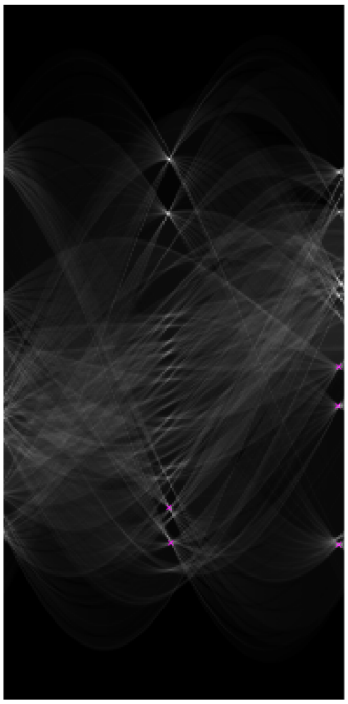
Part #3

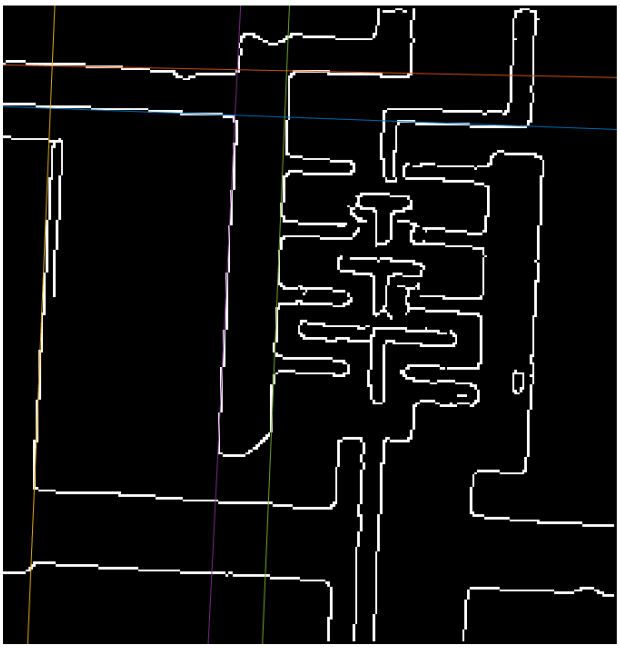
Post-processing: high level detection and interpretation

Step 5: choose points in Radon transform and observe associated lines

Using the function *interactiveLine()* we can understand the relation between the points in the Radon space and the lines in the image. It can be noticed that each point in the image corresponds to a sinusoid in the transform domain and the main lines of the image (the edges) correspond to the inter-

section points between beams of sinusoids, because these sinusoids are the transform of the points belonging to the same line.





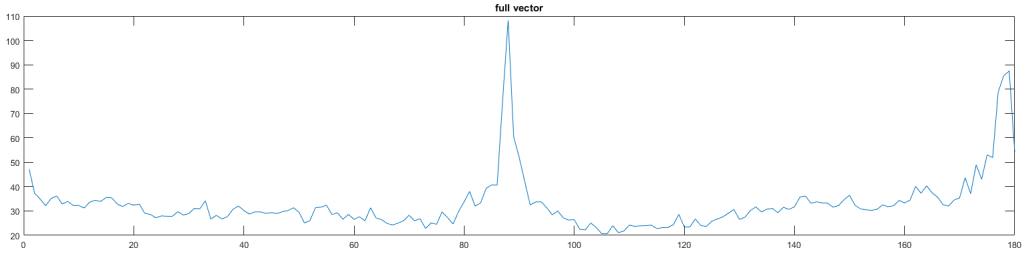
Step 6: find the image orientation and rotate it

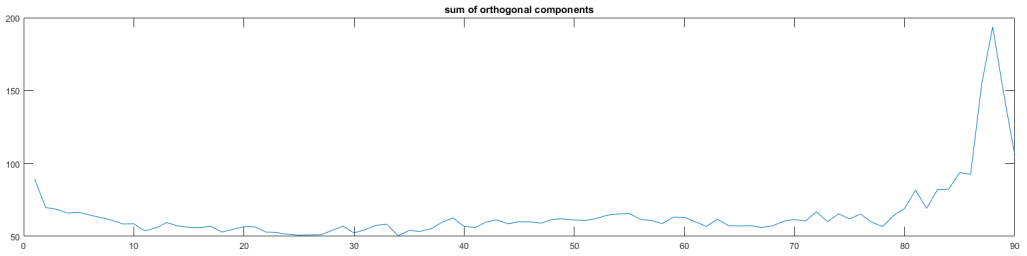
To find the image orientation we need to find the most significant columns in the Radon transform. In order to do this we compute the maximum value of each column of the matrix *rad* and we save it in the vector *high*.

```
rad=radon(edges);

high=zeros(1,180);
for i=1:180
  high(i)=max(rad(:,i));
end
```

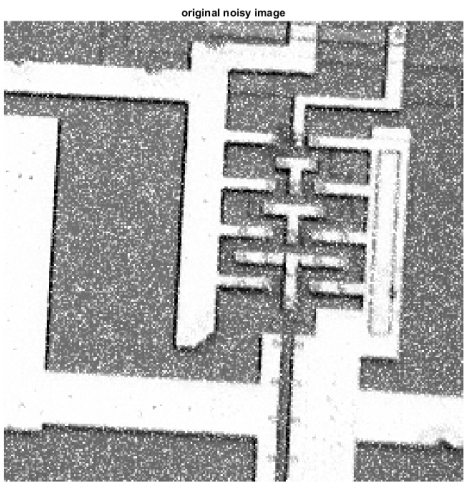
Since we want to find two main (orthogonal) directions we need to look for the maximum value of the sum of the two parts of the *high* vector, from 0 degrees to 89 and from 90 degrees to 179.

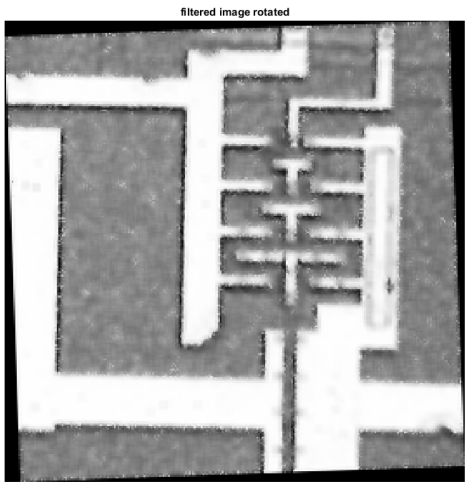
```
maxsum=max(high(1:90)+high(91:180));
indexsum=find(high(1:90)+high(91:180)==maxsum);

img_rot=imrotate(img_f,90-indexsum,'bicubic');
```





The orientation of the image corresponds to the index of the greatest value in the vector therefore to align the image we need to rotate it by *90* - *index* degrees.





Final remarks

When we increase the noise we lose the sharpness of the image also after preprocessing but the computation of the rotation angle is not influenced because the Canny method is able to detect edges anyway.

By increasing the size of the filter we can decrease the amount of noise but the image may become too blurred and it is possible to lose some segments of the edges.