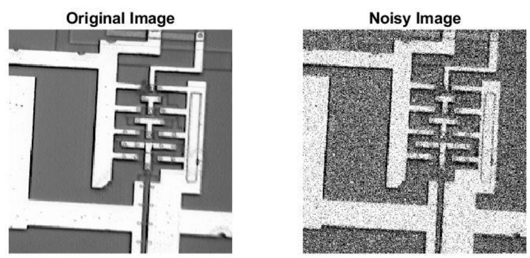
Fatemeh Changizian

15-11-2020

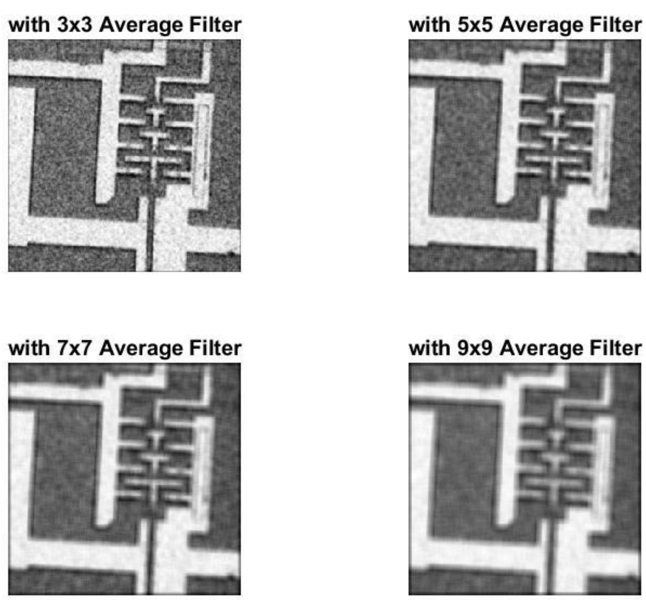
ImProc - Lab session 3: Edge detection

II. Pre-processing: de-noising

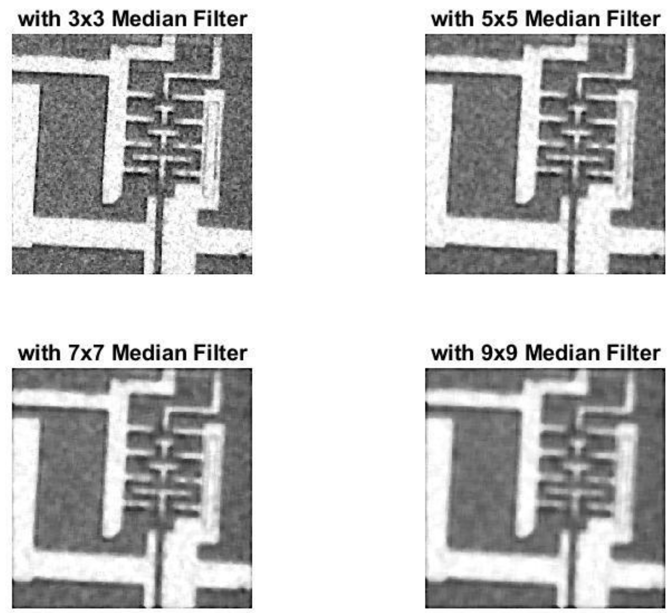
A. Load the image and artificially add some noise:
   Add **Gaussian** noise:



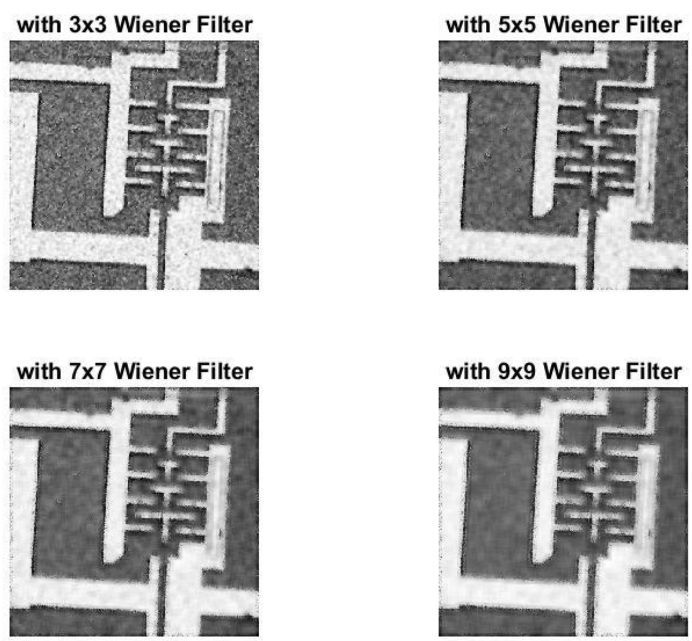
B. De-noising the image:
   Apply **average** filter with 4 different filter sizes:



Apply **median** filter with 4 different filter sizes:



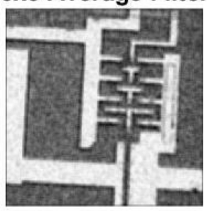
Apply **wiener** filter with 4 different filter sizes:



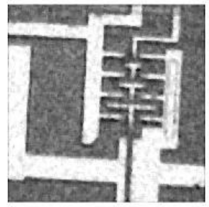
Comparison between the results of applying average, median and wiener filter:

** Personally found **5x5 filter size** more optimal than others. **

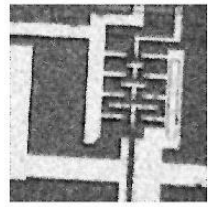
5x5 Average Filter



5x5 Median Filter



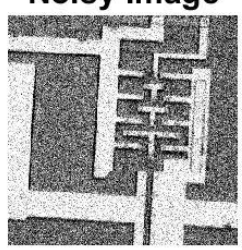
5x5 Wiener Filter



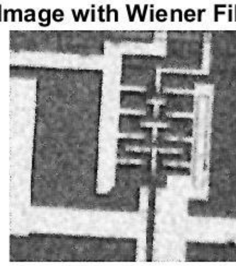
After comparing different filters, I think the best result for noise reduction has been achieved with the **Wiener Filter** because it works reasonably well in a way that we can still see the edges with acceptable noise reduction.

Therefore:

Noisy Image



Denoised Image with Wiener Filter size 5x5



III. Processing: low level feature detection

A. Highlight edges

Apply three different methods for **edge detection: gradient, zero crossing of Laplacian and Canny edge detector.**

Part1: Gradient edges detection

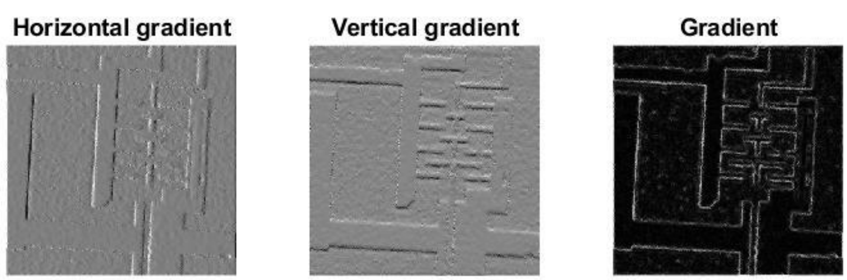
Applying horizontal and vertical gradients to detect vertical and horizontal edges:

```
gradient_h = filter2(horizontal_mask,denoised_image);

gradient_v =
filter2(vertical_mask,denoised_image);
```

and Compute the gradient magnitude:

```
gradient = sqrt(gradient_h.^2+gradient_v.^2);
```



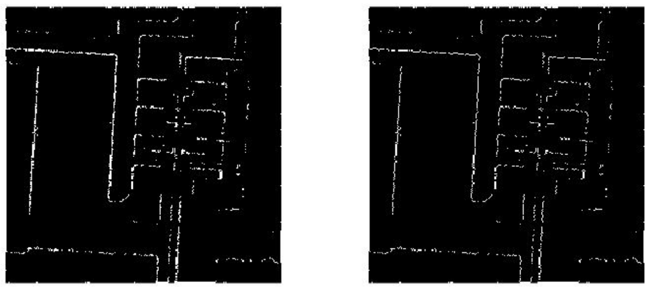
Additinal Step for part1:

Binarizing the image and performing morphological operations for making the edges more clearer:

```
BW = imbinarize(gradient,(graythresh(gradient)*255));
```

Creating more thin edges:

```
BW_morph = bwmorph(BW,'thin');
```



Part 2. Zero crossing of Laplacian

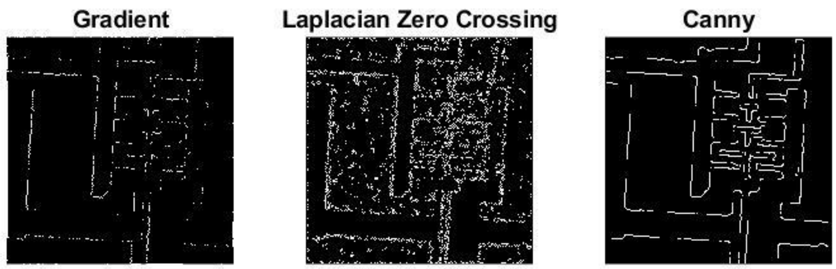
```
laplacian_mask = [0 1 0;1 -4 1;0 1 0];
lap_filtered =
edge (denoised_image,'zerocross',0.5, laplacian_mask);
```

Part 3. Canny edge detector

```
canny_filtered = edge(denoised_image,'canny',0.5);
```

Comparing the results of three different techniques to perform edge detection on denoised image:

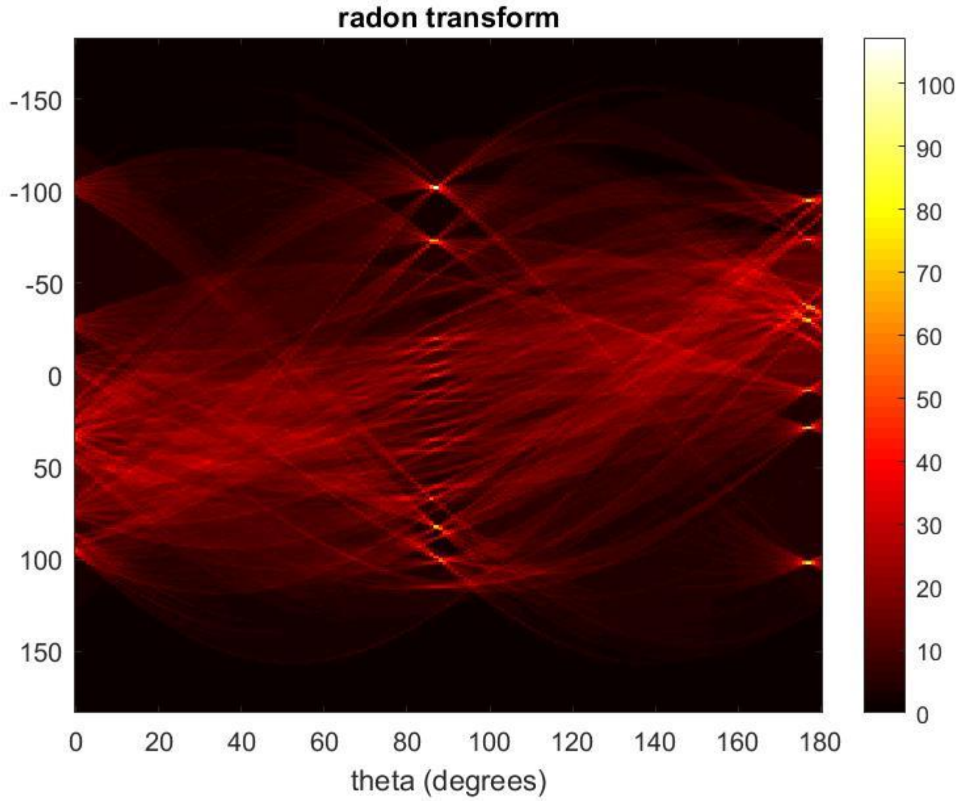
-As we can see in the results, the gradient and Laplacian zero crossing methods aren't good enough for edge detection but on the other hand, **"Canny edge detector"** seems to perform better and more efficient than others.



B. Compute the Radon transform:

Compute the **Radon transform** of the final edge image obtained in the previous step from **Canny**:

```
theta = 0: 180;
[R, xp] = radon (canny_filtered, theta);
imagesc (theta, xp, R);
title ('radon transform');
xlabel ('theta (degrees)');
set (gca, 'XTick', 0: 20: 180);
colormap (hot);
colorbar
```



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

We know the Hough transform can associate each line of the image with a pair (theta, ?) point in Hough space. This point is the intersection of many curves and each curve represents a point of the image. We can clearly see several intersects of the curves for theta=90 on the y-axis which represent horizontal edges and for theta around 0 and 180 which represent vertical edges of the image on the x-axis.

[Theta: Projection angle (in degrees)]

-The Hough transform and the Radon transform are indeed very similar to each other and their relation can be loosely defined as the former being a **discretized** form of the latter. We can use the radon function to implement a form of the Hough transform used **to detect straight lines**.

-Specific features (geometries) in the original image produce **peaks** or collections of points. Masks can be easily applied to the image within the Radon domain to determine if and where these specific features occur.

IV. Post-processing: high level detection & interpretation

A. Choose points in Radon transform and observe associated lines:

```
FUNCTION interactiveLine(imgEdge,imgRadon,N);
  Parameters:
    - imgEdge: Edge image
    - imgRadon: Radon transform of imgEdge
    - N: Number of lines to be drawn
```

so I use:

```
interactiveLine(canny_filtered, R,3)
```

For points near Theta = 90, the associated lines are horizontal. For points near Theta = 180, the associated lines are vertical.

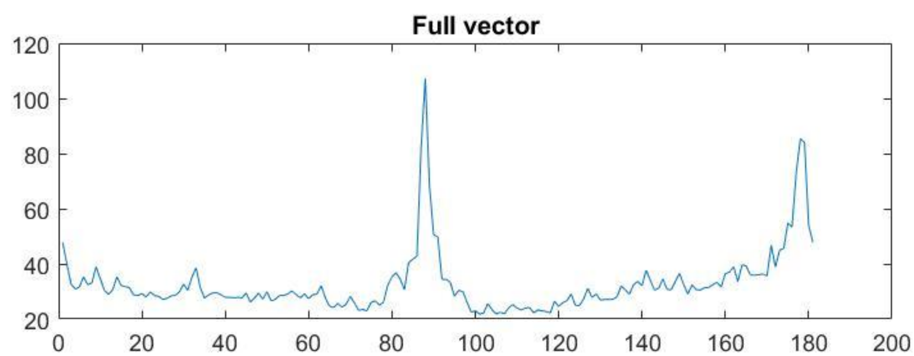
B. Find the image orientation and rotate it:

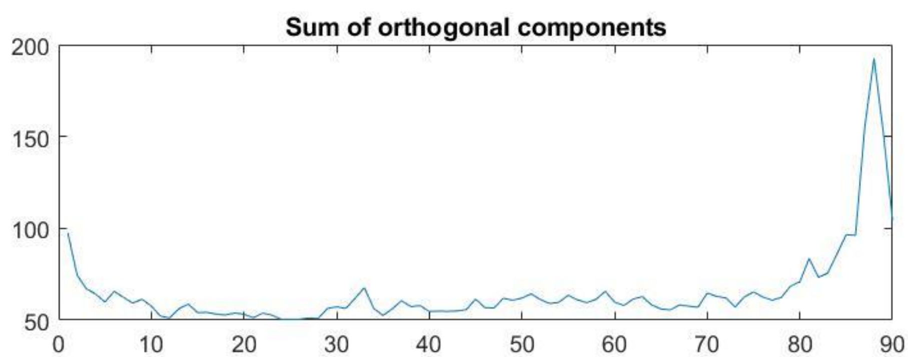
```
V = max(R);
```

Summing the vertical and horizontal lines for a better estimate of the true rotation angle.

```
V_orth = V(1:90) + V(91:180);
indexsum = find(V_orth == max(V_orth));
```

Since we look for two main directions (orthogonal), we need to look for the highest value of the sum of the V vector, from 0-89 and from 90-179 to find out which two orthogonal directions have more edges in the image.

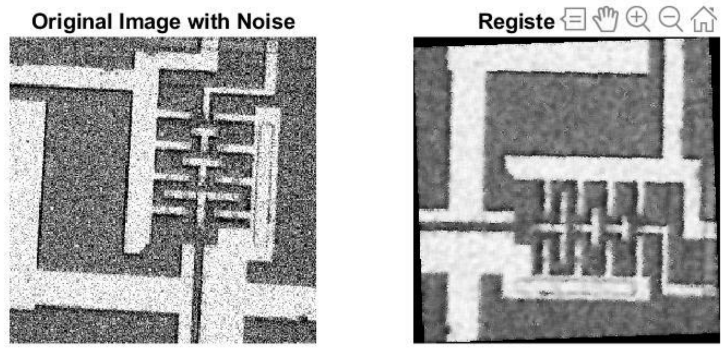




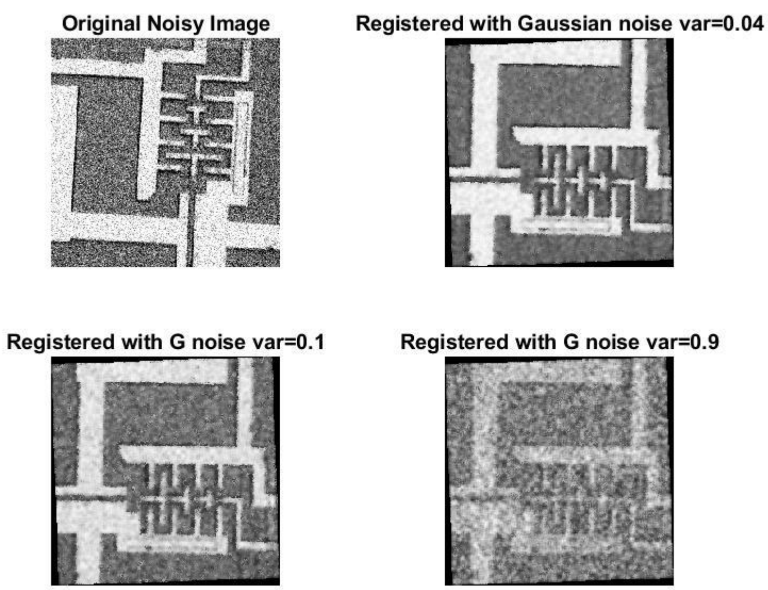
We know the locations of peaks in the radon transform correspond to the **locations of straight lines** in the original image.

-In V plot, as we can see there are 2 main peaks value for **angle(theta)=88** and for **angle=180** which means that most of the image edges are in these two directions.

-In the V_orth plot, we see only one peak value at **angle=88** which means the two orthogonal directions with most edges are **88 and 88+90**. So we rotate the original image with the angle = -88 so that the image edges, appear horizontal and vertical.



For Advance question in Step 6:
Increasing the **strength of the noise** in Step 1 and observe when the rotation angle fails to be estimated correctly:



Although by increase the noise strength, we have **blurred image** even after preprocessing, the orientation process and the computation of the rotation angle is still works well since **the Canny edge detector** is capable of detecting edges anyway.