In GOD WE TRUST

Signals and Systems Course

1397-1398 Second Semester

Matlab HW

Part 3

Omid Sharafi
96101838

Instructor:
Dr.Hamid Aghajan

May 25, 2019

Sharif
University
of
Technology

EE

SUT

Contents

1 Edge detection algorithms

1.1 Sobel operator

For this part, I implemented Sobel operator for each RGB and grayed picture so I got two different answers. In RGB images, each color shows edges of that specific color and for having better view, we could plot them in different pictures. For examining Sobel filter, I selected pictures containing obvious edges and especially horizontal or vertical lines and having some uniform parts that we know their answer should be completely black because there is no edge or difference in color and our filter answer for these pixels are zero.

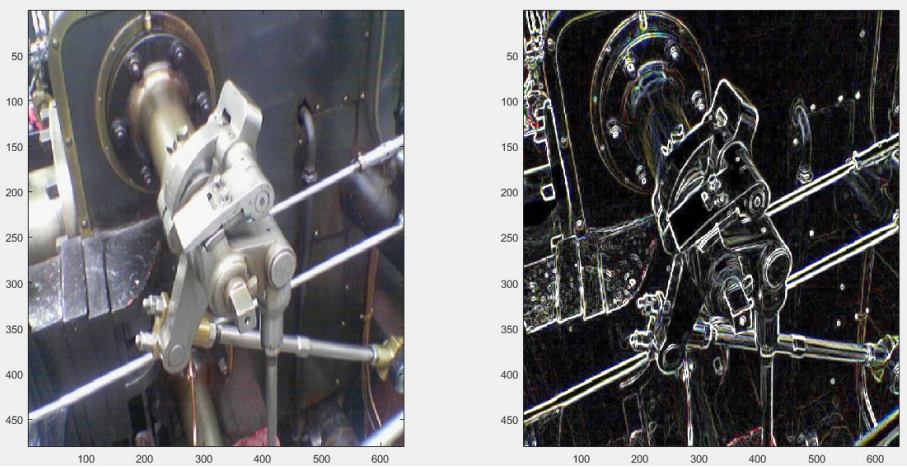

Figure 1: RGB edge detection result using Sobel operator

I could use $imfilter$ function but I think question wants to filter data by convolution but I didn't implement convolution using summation by myself (convolution implemented for part4.3) and I used $conv2$ function for getting faster result. (see appendix 1) This part is a routine, fundamental and important problem so that matlab even have Sobel operator in $fspecial$ function.

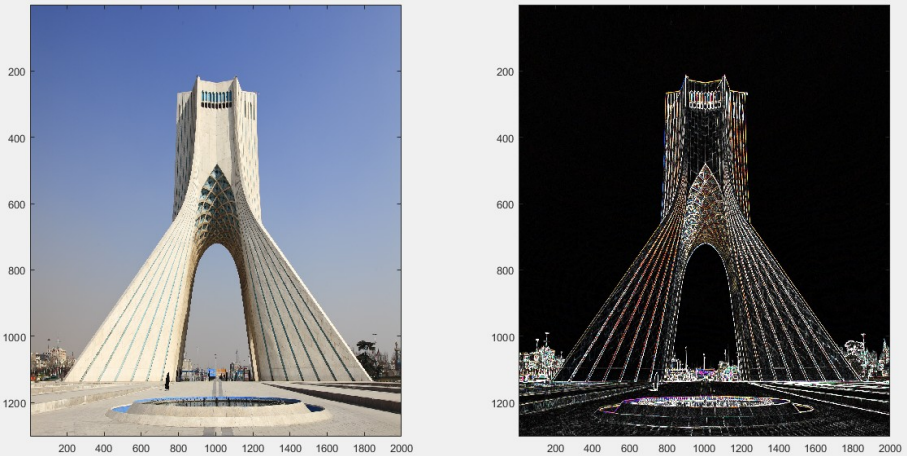

Figure 2: RGB edge detection result using Sobel operator

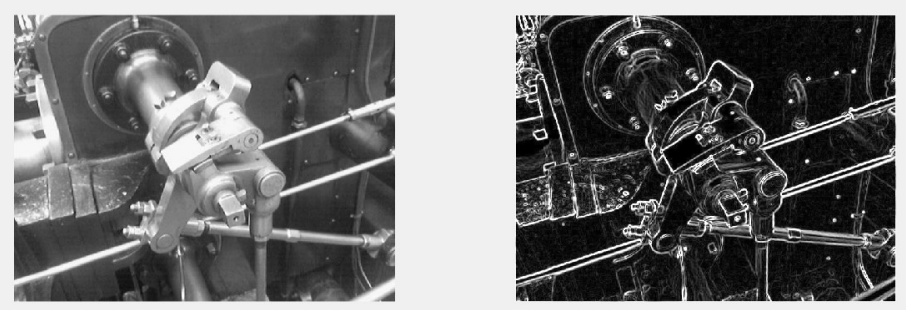

Figure 3: Grayed edge detection result using Sobel operator

Figure 4: Grayed edge detection result using Sobel operator

1.2 Kirsch Operator

Kirsch operator execution took 0.075446*seconds* while Sobel operator took 0.015679*seconds* (minus defining time of tmp matrix and plotting). As we guessed, Kirsch operator execution took more time because it has 4 time more kernels and it's more accurate but because we use for loop in Kirsch operator implementation, if we really want to compare exact execution time of each method and compare them, we should use exactly same method of implementation for both.

Figure 5: Grayed edge detection result using Kirsch operator

Figure 6: Grayed edge detection result using Kirsch operator

Kirsch filter has 8 kernels for 8 direction of edges and it's more accurate especially for curves so for examining Kirsch filter, I selected pictures that have curves plus other futures that described for previous part.

2 Circle detection

I implemented this problem with two ways. First with noise reduction ,edge detection and using $imfindcircles$ function of matlab.

In second way, first I moved radius and in each step made a circular filter with that radius and convolved it with picture and find output, and after this, by finding maximum of each pixel and by setting a threshold, we can find center of circles. This method can be improved by using other function instead of finding maximum or using clustering. This algorithm is $O(n^4)$ that $A = n^2$ is our picture size but it's fast enough, because for matrix calculations I tried to use matlab built in functions as much as possible. My algorithm draw centers of circles but because circles' boarders' lines are not just one pixel, my algorithm gives several joint pixels for each center and for calculating number of circles we can use dsu or clustering algorithms or using filters.

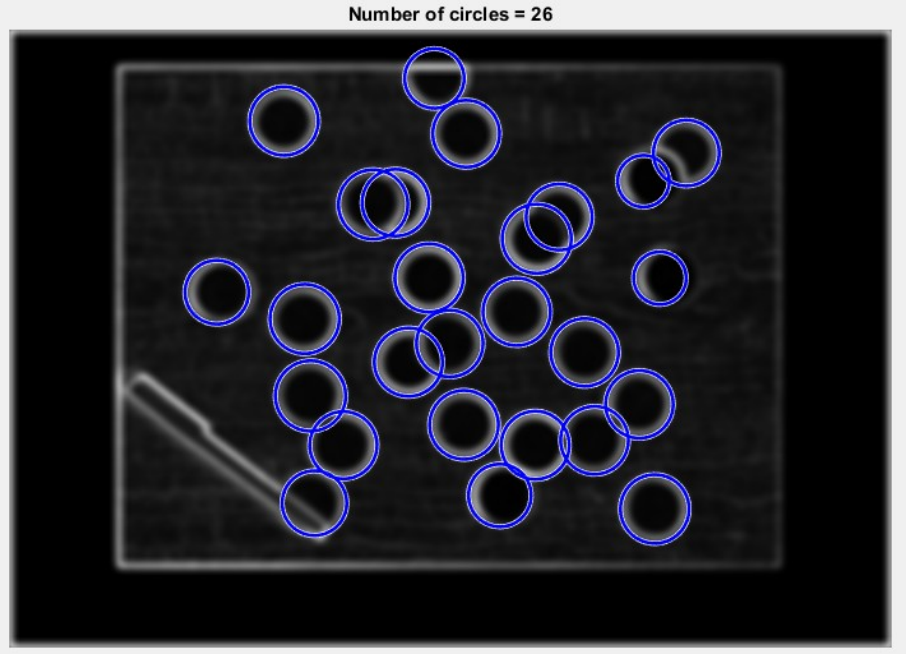

Figure 7: Circle detection using imfindcircles matlab function

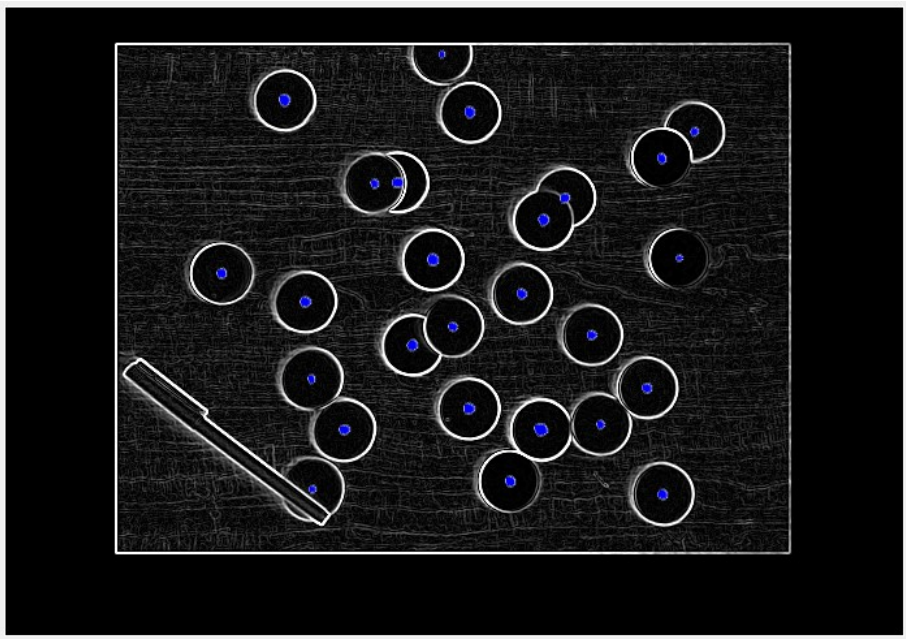

Figure 8: Center detection output by my algorithm

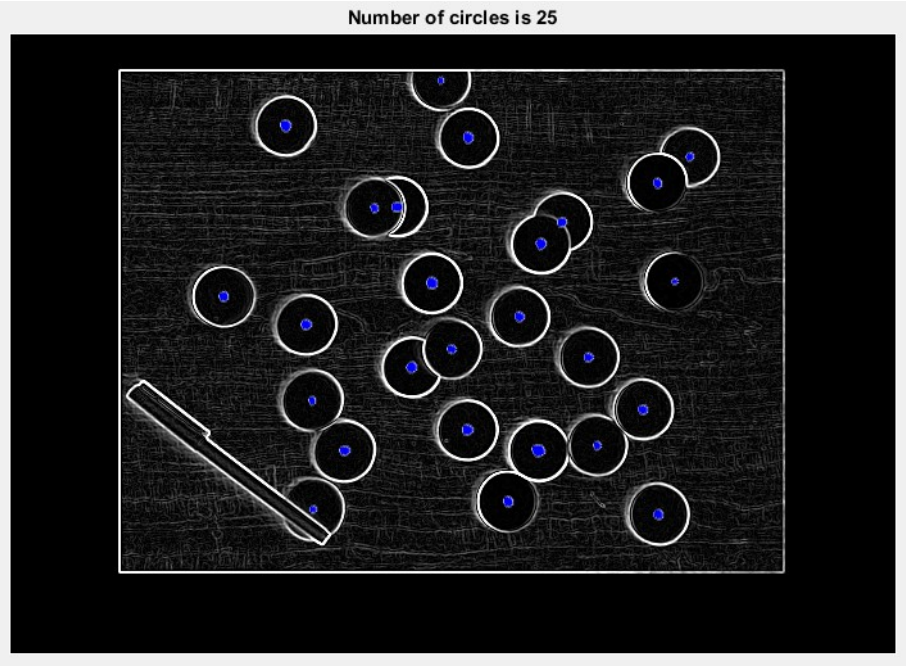

Figure 9: Number of circles by clustered data (1 circle missed because I implemented really easy clustering method :))

3 Phase and absolute value of Fourier transform

As shown below, phase of FFT is more important and effective than it's magnitude and we can see pic1 picture with some distortion. (For more information about reason of this phenomenon see appendix 2)

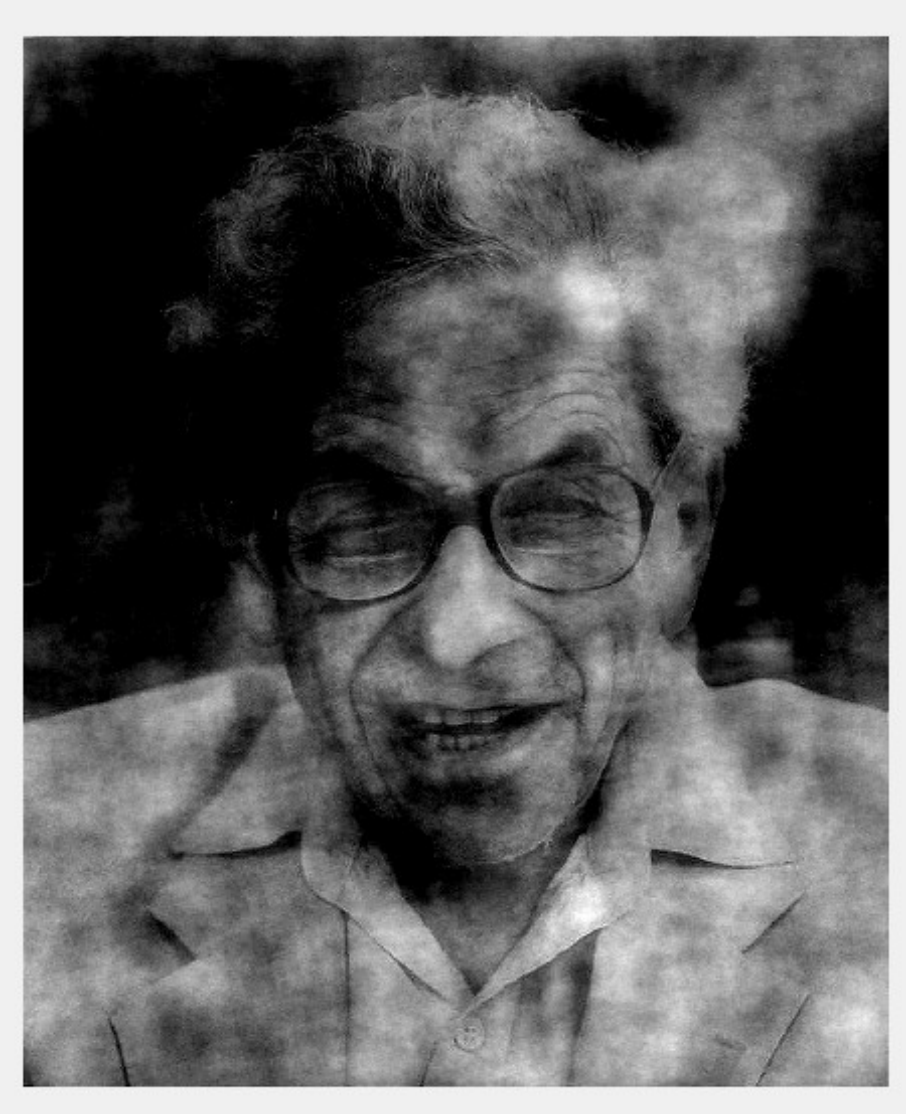

Figure 10: combination of phase and absolute value of two Fourier transform

4 Noise reduction

4.1 Types of noise

- **Salt and Pepper** : Salt and pepper noise is a form of noise sometimes seen on images. It is also known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels. An effective noise reduction method for this type of noise is a median filter or a morphological filter.[1]

- **Gaussian** : Gaussian noise is statistical noise having a probability density function (PDF) equal to that of the normal distribution, which is also known as the Gaussian distribution. Principal sources of Gaussian noise in digital images arise during acquisition e.g. sensor noise caused by poor illumination and/or high temperature, and/or transmission e.g. electronic circuit noise. Gaussian noise can be reduced using a spatial filter (such as median filter and Gaussian filter). [2]

- **Poisson** : Shot noise or Poisson noise is a type of noise which can be modeled by a Poisson process. In electronics shot noise originates from the discrete nature of electric charge. Shot noise may be dominant when the finite number of particles that carry energy (such as electrons in an electronic circuit or photons in an optical device) is sufficiently small so that uncertainties due to the Poisson distribution, which describes the occurrence of independent random events, are of significance.[3]

- **Speckle** : The vast majority of surfaces, synthetic or natural, are extremely rough on the scale of the wavelength. Images obtained from these surfaces by coherent imaging systems such as laser, SAR, and ultrasound suffer from a common phenomenon called speckle. Speckle, in both cases, is primarily due to the interference of the returning wave at the transducer aperture. Several different methods are used to eliminate speckle noise, based upon different mathematical models of the phenomenon. One method, for example, employs multiple-look processing, averaging out the

speckle noise by taking several "looks" at a target in a single radar sweep.[4]

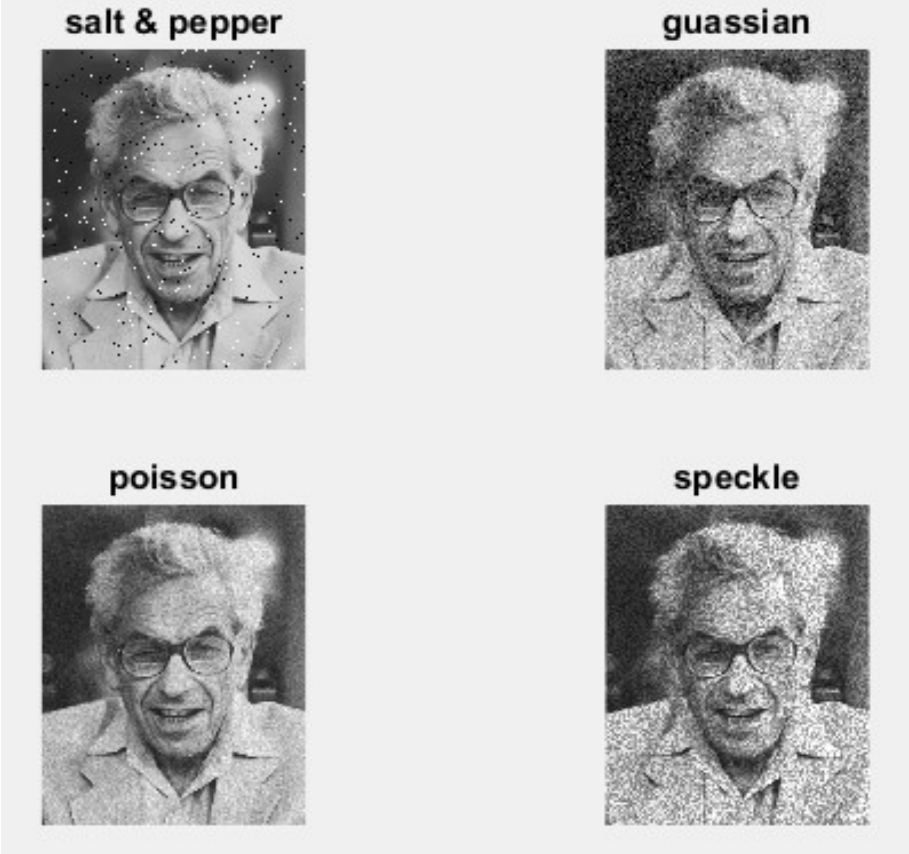

Figure 11: Noisy pictures (properties of noises are same as matlab defaults)

4.2 Median and Gaussian filter

- **Median filter** : The Median Filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image).[5] For implementing median filter, after expanding edges of picture, we should convolve filter that all cells of that is $\frac{1}{n^2}, n = kernelsize$ so that each pixel is mean of it's neighbours and we have sum of all cells of filter is one so energy doesn't change.

- **Gaussian filter** : In electronics and signal processing, a Gaussian filter is a filter whose impulse response is a Gaussian function. The idea of Gaussian smoothing is to use this 2-D distribution as a 'point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point. But based on what our problem needs for next part, I first fill each cell of Gaussian filter using given σ and then divide it by sum of cells so that filter doesn't change energy of picture. This is not really good Gaussian filter function and problem should give us σ or $kernelsize$ but not dictates both!

4.3 Noise reduction using filter

4.3.1 Main part

I implemented both filters and if you run code, filer parameters will sweep and you can see filtered pictures. Because problem said not using matlab functions, I even implemented convolution by my self with for loop so code is too slow. There are some outputs of related part code below.

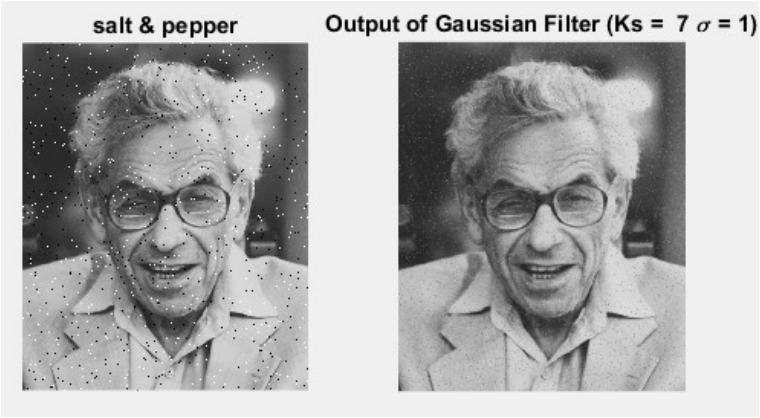

Figure 12: Output of Gaussian filter on salt and pepper noisy picture

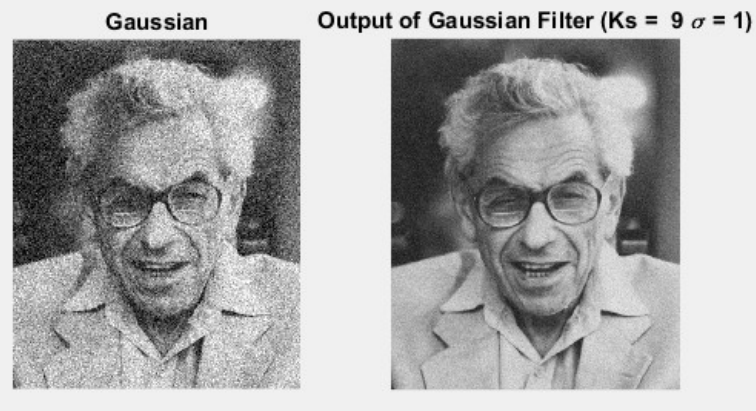

Figure 13: Output of Gaussian filter on Gaussian noisy picture

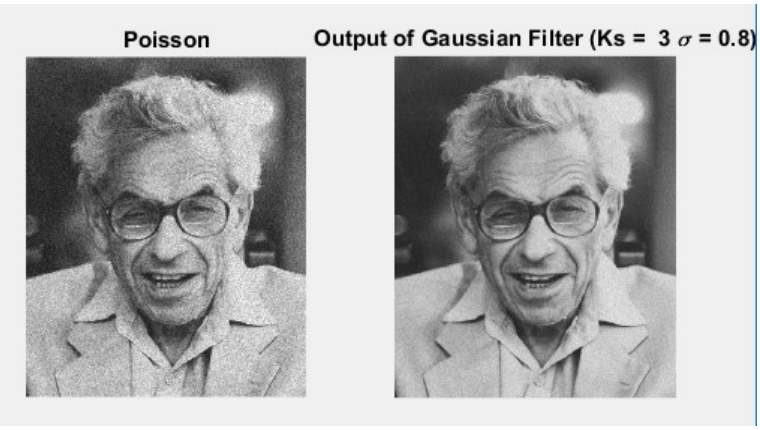

Figure 14: Output of Gaussian filter on Poisson noisy picture

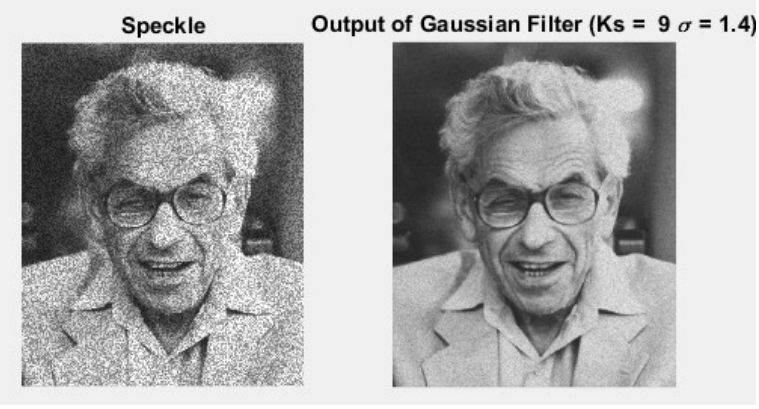

Figure 15: Output of Gaussian filter on Speckle noisy picture

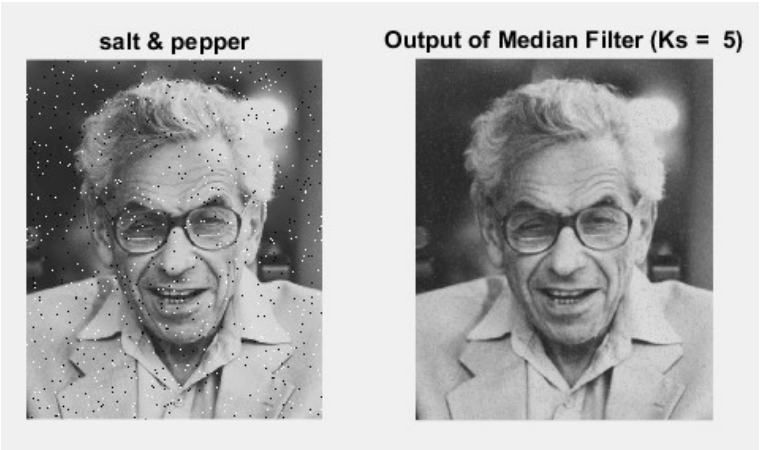

Figure 16: Output of median filter on salt and pepper noisy picture

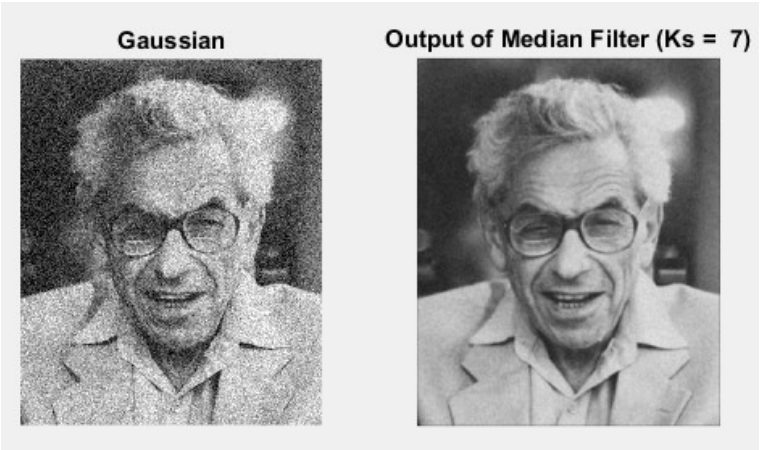

Figure 17: Output of median filter on Gaussian noisy picture

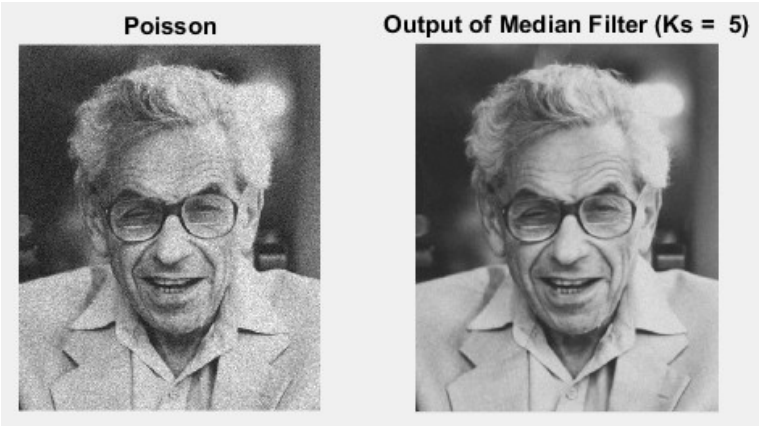

Figure 18: Output of median filter on Poisson noisy picture

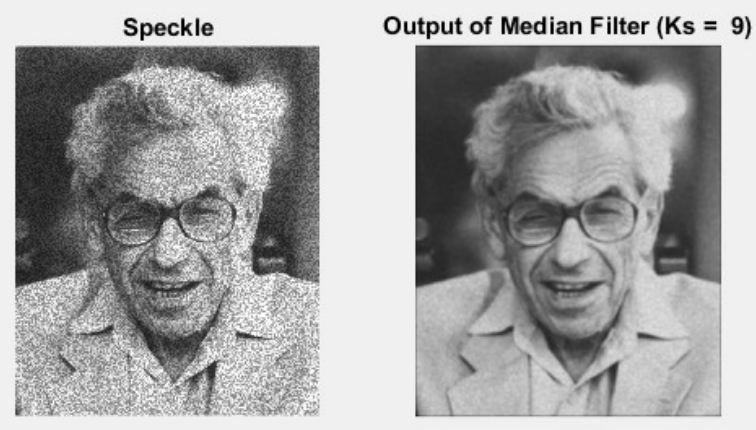

Figure 19: Output of median filter on Speckle noisy picture

4.3.2 Extra part

- **Dark frame subtraction**: A dark frame is an image captured with the sensor in the dark, i.e., with a closed shutter or the lens and viewfinder capped. Such a dark frame is essentially an image of noise produced by the sensor. A dark frame, or an average of several dark frames, can then be subtracted from subsequent images to correct for fixed-pattern noise such as that caused by dark current, but it also works for removing what is called amp glow, portions of the sensor lighting up due to internal heat sources.[5]

- **Unsharp masking** The software applies a Gaussian blur to a copy of the original image and then compares it to the original. If the difference is greater than a user-specified threshold setting, the images are (in effect) subtracted. The threshold control constrains sharpening to image elements that differ from each other above a certain size threshold, so that sharpening of small image details, such as photographic grain, can be suppressed.[6] Implementing this filter is really easy and just needs a if in Guassian filter function.

- **Combining Filters** For combining filters, we can use different methods, one method is to combine them in a convolutional neural networks that we implemented in neuroscience course. By these networks we can make very complex image processing and filters

such as object detection. On the other hand, non-linear filters have many applications, especially in the removal of certain types of noise that are not additive. For example, the median filter is widely used to remove spike noise — that affects only a small percentage of the samples, possibly by very large amounts. In digital image processing, for example, one may wish to preserve the sharpness of silhouette edges of objects in photographs, or the connectivity of lines in scanned drawings. A linear noise-removal filter will usually blur those features; a non-linear filter may give more satisfactory results (even if the blurry image may be more "correct" in the information-theoretic sense).[7]

5 FMRI pictures

Our goal in this part is to maximize corr2 of two FMRI pictures by rotating and moving one of them. I implemented this code in two ways. First I implemented it using matlab *imregtform* function and although I set optimizer's futures tightly but it really fast and I think the reason is not only because it's built in function of matlab, but also because it doesn't check all combination of rotations and movements and for example by gradient descent method or something similar it converges to maximum point of correlation.

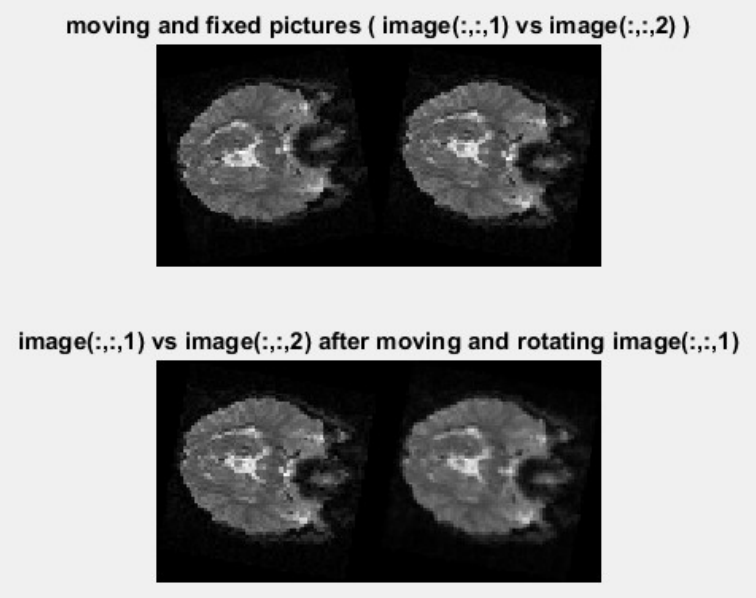

Figure 20: Output of rotated and moved FMRI picture

```
X = 1.536674e+01
Y = -1.068557e+01
Theta = 2.008641e+01
Max Corolation = 9.778391e-01
```

Figure 21: Parameters of rotation and movement of FMRI picture

On the other hand, I implemented this problem using three for loops and making transfer matrix for each a, b and theta, then calculating correlation and finally maximize correlation. But this method is really slow and although I know band of answer, but it takes a minute for 0.1 precision!

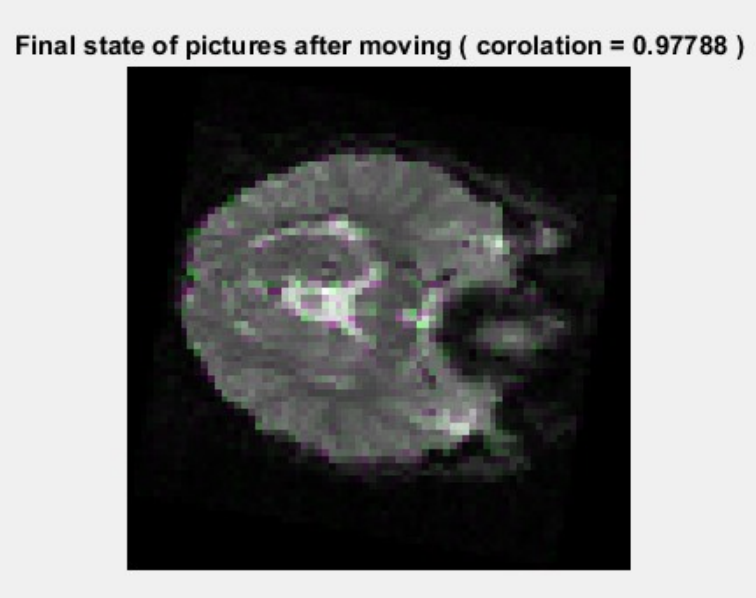

Figure 22: Output of rotated and moved FMRI picture using imshowpair with 'falsecolor' method (self implemented answer)

```
X = 1.550000e+01
Y = -1.070000e+01
Theta = -2.010000e+01
Max Corolation = 9.778755e-01
```

Figure 23: Parameters of rotation and movement of FMRI picture (self implemented answer with 0.1 precision)

A Appendix

A.1 Why matlab matrix built in functions are so fast

- Credit : stackoverflow

Generally the builtins will be faster owing to that they're compiled code as opposed to m-files which are interpreted. Although the JIT compiler does a good job on loops and other expressions, it still isn't the same as writing code for an optimizing compiler and using it instead.

Some built-ins have enough error checking and other features incorporated that the full benefit isn't realized as much as one might think; this is particularly true for smaller array sizes where the calling overhead and error checking may be a significant fraction of the total run time. Larger arrays will tend to show the benefit more and timing tests to try to quantify such results would need to include such effects.

Over the years (notably between the BLAS level 1 and level 2 releases: early 80s), hardware changed, with the advent of vector operations and cache hierarchies. These evolutions made it possible to increase the performance of the BLAS subroutines substantially. Different vendors then came along with their implementation of BLAS routines which were more and more efficient. Two of the most notable ones came out in the early 2000s: the Intel MKL and GotoBLAS. Your Matlab uses the Intel MKL, which is a very good, optimized BLAS, and that explains the great performance we see.

A.2 Why magnitude of FFT is more recognizable for our eyes

Our eyes are like a optical glass so this fact is not really related just to our eyes structure. Think that there is a ball. If we move it (that is shift in time) we have just change in phase so that magnitude is not really as important as it's phase and as we can see in result, when we changed it's magnitude (and keep phase), some part of original image become lighter or darker but we can recognize it's original image and phase show components of it's Fourier transform that is more important.

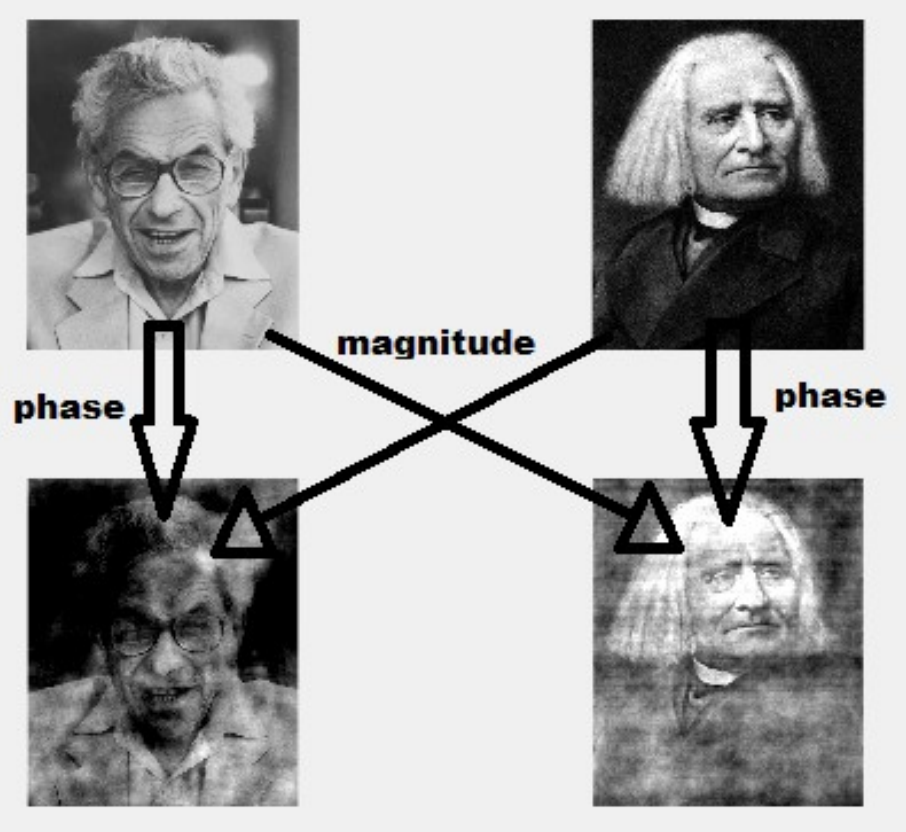

Figure 24: combination of phase and absolute value of two Fourier transform