CSC 449 - Project 1

Phyo Thiha

Introduction

For project 1, we worked with image I/O and image filtering in Matlab. We learned image filtering in the space domain by implementing Gaussian Low Pass FIR (finite impulse response), High Pass and Median Pass filters. We also tried Sobel edge detector and experimented with adding Gaussian and impulse noises to observe if the edge detector is immune to different type of noises. Finally, we experimented with 2-D Fourier Transform (FT) using built-in Matlab functions to observe their ability to discriminate image structures.

For each part of the project, we will present our findings in the order of methods used, results obtained and discussion of the results. We provide the resources we used as reference for this project in the Appendix section.

PART 1

Relevant File(s)

run_one.m - script that runs part 1 in its entirety

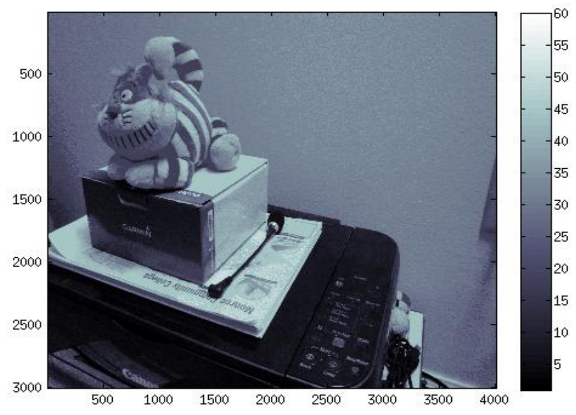
Methods

We used *imread*, *imshow* and *imagesc* functions in Matlab to read and display an image. We called *colormap('bone')* to set the image's colormap to one of Matlab's built-in colormaps. We then used *colorbar* to show the color scale of the image. Finally, we wrote the image using *imwrite*.

Results and Discussion

Figure below show the result from the above transformation.

a)



b)

Figure 1: a) The image is displayed by using *imread* and *imshow.* b) Using *imagesc* scales the image (to a smaller size) and shows axes representing pixel location. The colormap for the image is set to *bone* and we used *colorbar* function to show the color scale.

The *imshow* function only displays the image in a window; this function automatically eliminates the axes. The *imagesc* scales the original image data to fit in a tighter window, and display the image. It also shows the axes representing pixel coordinates by default. *Imagesc* accepts parameters for setting the axis properties, and can be used to scale, shift and zoom the image in the figure window.

We used *colormap* to transform the image into different color scale (in this case, we chose Matlab's default *bone,* which is a grayscale colormap with a higher value for the blue component).

PART 2

Relevant File(s)

run_two.m	- script to run part 2 in its entirety
two_a.m	- script to read and display an image
two_b.m	- script to run Gaussian filter and its 1D components on the image (see helper file below)
get_filter_components.m	- helper file to decompose nxn decomposable filter to its 1D components
two_c.m	- script to run 3x3 high pass filter on image (see helper file below)
pfilter.m	- our implementation of applying 3x3 filter to the image
two_d.m	- script to apply 3x3 median filter to the image
two_e.m	- script to run Sobel edge detector on the image
two_f.m	- script that adds Gaussian noise to the image and rerun part b-e above
two_g.m	- script that adds salt and pepper noise to the image and rerun part b-e above

Methods

We used *imread* and *imshow* to read and display the image (cheshire_normal_small.jpg) as described in previous part of the lab.

Next, we invoked *fspecial* command with parameter 'gaussian' to get a 3x3 Gaussian filter. We used *filter2* to filter the image with the aforementioned Gaussian filter, and called *freqz2* to obtain the frequency response of the filter.

We implemented *get_filter_components* function, which utilizes Singular Value Decomposition (SVD) to derive the 1D filter components of the original 3x3 Gaussian filter[1, 2]. We used Matlab function *svd* to obtain the outer product vectors of the original Gaussian filter. We then applied these 1D component vectors to the image using *filter2*.

We created a simple high pass filter (see Figure 2 a.), and wrote a Matlab function, named *pfilter*, to apply 3x3 filters to a given image. *pfilter* works by updating pixel values of the original image according to the weights in the filter as the center of it visits every pixel in the image (see Figure 2 a). In order to handle pixels at the boundaries, we padded the original image with zeros;

this allowed the filter to visit boundary pixels safely without incurring out-of-bound errors in Matlab (see Figure 2 b).

$$\begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \; * \; \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} \; \Rightarrow \; \begin{array}{lr} (-1)\,(2) + (-1)\,(3) + (-1)\,(3) & = -\;8 \\ (-1)\,(3) + (8)\,(5) + (-1)\,(5) & = \;32 \\ (-1)\,(4) + (-1)\,(4) + (-1)\,(6) & = -14 \\ & \text{-------} \\ & 10 \end{array}$$

a)

0	0	0	0	0
0	2	3	3	0
0	3	5	5	0
0	4	4	6	0
0	0	0	0	0

b)

Figure 2: a) Simple high pass filter on the left is applied to every pixel in the image, part of which is shown in the right table. The highlighted pixel value is updated from ‘5’ to ‘10’ after the visit.

b) We padded the image with zeros around the borders to avoid out-of-bound errors when applying the high pass filter to pixels on the edges.

As for filtering with a 3x3 median filter, we used Matlab built-in function *medfilt2.*

To use Sobel edge detector, we invoked *edge* function in Matlab with the second argument as ‘sobel’. To add image noise, we used *imnoise* with parameter ‘gaussian’ and ‘salt & pepper’ for Gaussian and impulse noise, respectively.

Results and Discussion

[Note: We were not sure from the description of the problem if we are supposed to carry the results from the steps prior . Therefore, from part 2 a-e, we decided not to carry the results from the previous steps to the next. That is, we reload the image from fresh in every step.]

The results from processing the image with Gaussian low pass filter and its components are shown in figure 3. We found that the original 3x3 Gaussian filter can be decomposed into 1D components by using a linear algebra technique called singular value decomposition (SVD). Mathematically, singular value decomposition is defined as follows[3].

Suppose *M* is an *m×n* matrix whose entries come from the field *K*, which is either the field of real or the field of complex numbers. Then there exists a factorization of the form

$$M = U \Sigma V^{*}$$

where *‘U’* is an *m×m* unitary matrix over *K*, the matrix Σ is an *m×n* diagonal matrix with non-negative real numbers on the diagonal, and *V**, an *n×n* unitary matrix over *K*, denotes the conjugate transpose of *V*.

By calculating the SVD of the 3x3 Gaussian filter, we obtained U and V*. Extracting the horizontal and vertical vectors from U and V* gave us the 1D components of the original filter. When we applied them individually—by using *filter2* function in Matlab—to the image, we obtained similar results to the image as when we applied the 3x3 Gaussian filter directly.

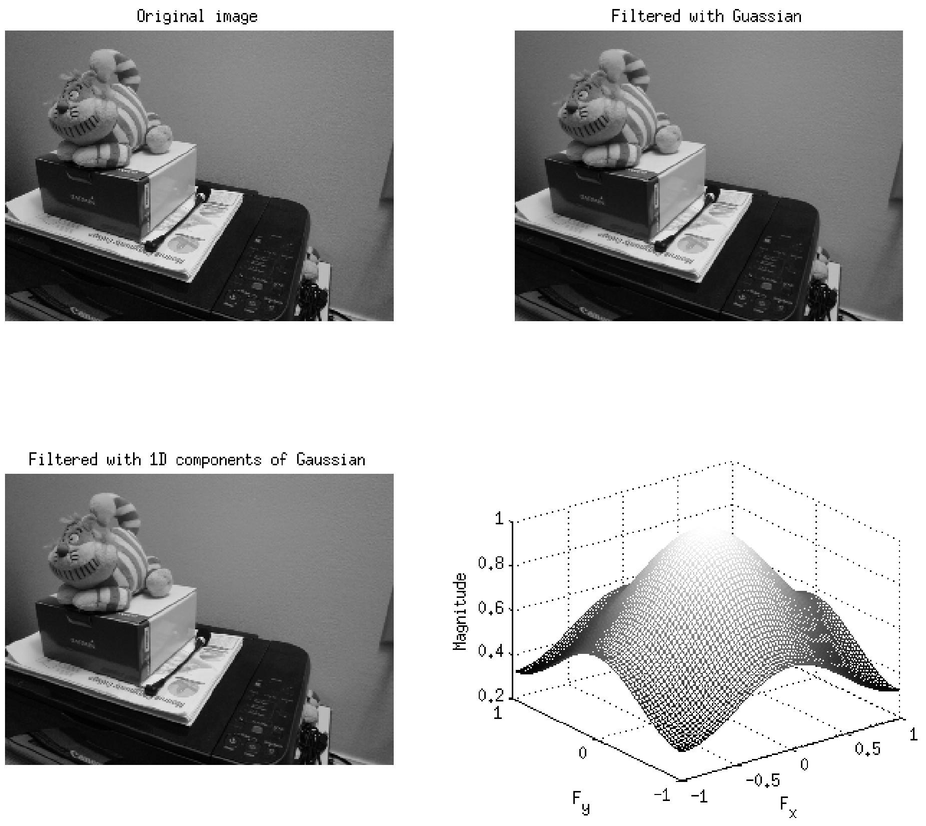


Figure 3: The original image is shown in the top left corner. The results from filtering with 3x3 Gaussian filter are displayed in the top right, while that of filtering with 1D components is on the bottom left. It can be seen (please zoom in) that the resulting images from these two filtering techniques are the same. The bottom right figure is the frequency response of the Gaussian filter.

We show sample pixels—in Matlab's *indexed* format—from these two filtering approaches below. Here, nine pixels in the top left corner of the original image, and those derived from 3x3 and 1D component filtering are displayed (the edges are highlighted with yellow and the rest in green). We observed that applying 3x3 filter and its 1D components individually achieve the same result.

```
original_image(1:3,1:3)

ans =

    0.3647    0.3569    0.3451
    0.3765    0.3804    0.3804
    0.3686    0.3686    0.3686
```

```
3by3_Gaussian_result(1:3,1:3)

ans =

  0.2917    0.3210    0.3130
  0.3347    0.3763    0.3748
  0.3308    0.3698    0.3679
```

```
1D_component_result(1:3,1:3)

ans =

  0.2917    0.3210    0.3130
  0.3347    0.3763    0.3748
  0.3308    0.3698    0.3679
```

In the frequency domain, the lowest frequencies usually contain most of the information, which is shown by the large peak in the center of the data. If the frequency response is shown as an image, the peak of low frequencies is composed of the brightest pixels[11]. Figure 4 shows a larger image of the frequency response than the one shown in figure 3.

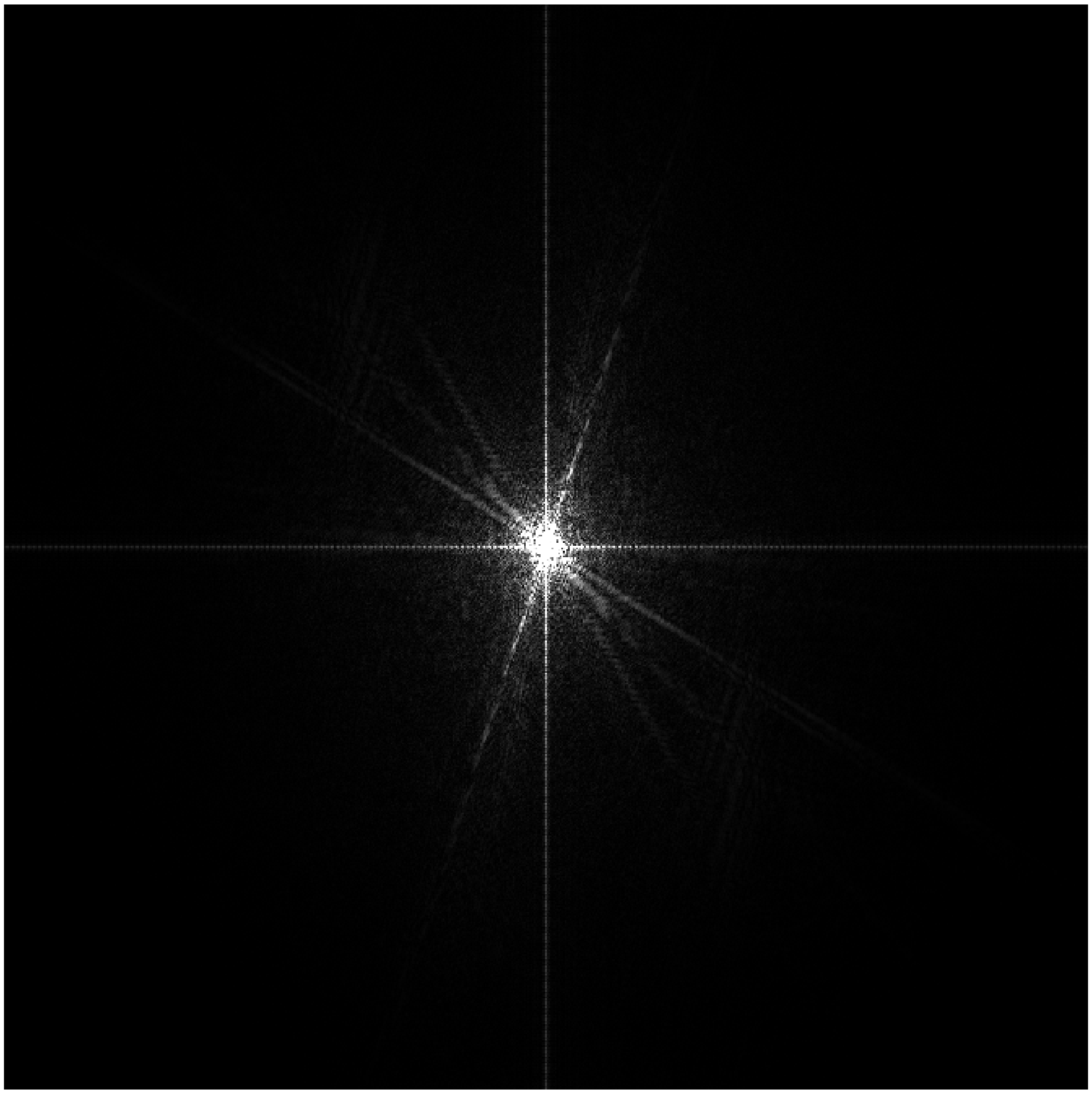


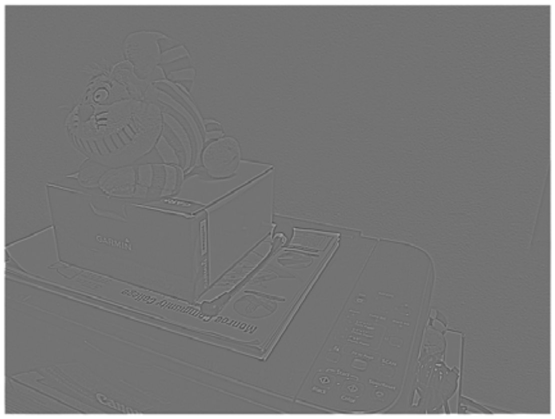
Figure 4: Magnitude of the frequency response on the image by the Gaussian filter

We observed that most of the energy in the image is represented by low frequencies, seen as the brightest area in the center of the image. This corresponds to many gradual changes in the image, which reflects the original Cheshire cat's image where the majority of the image is occupied by black-colored printer, the wall, white-colored box and mostly-pink cat. We also noticed some diagonal lines representing high frequencies; these lines represent the edges (boundaries) of sharp, geometric features in the original image—for example, the edge between the wall and the printer.

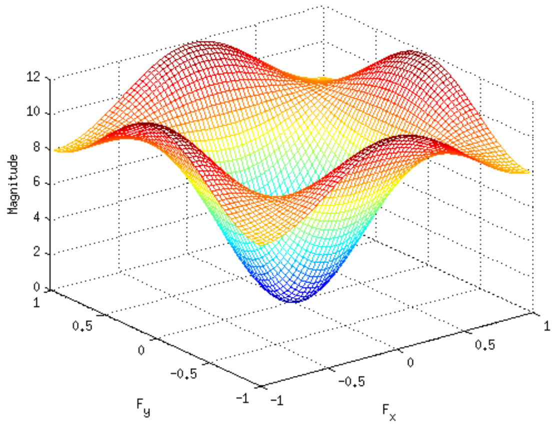
Figure 5 shows the results obtained from applying a 3x3 high pass filter to the image.

-1	-1	-1
-1	8	-1
-1	-1	-1

⇓



a)



b)

Figure 5: We applied a 3x3 high pass filters to the original image. a) Here, the high pass filter retained mainly the edges, where sharp differences in pixel values occur. b) The frequency response of the high pass filter shows that the higher frequencies are retained, but the lower frequencies in the middle are suppressed

Since a high pass filter tends to retain the high frequency information within an image while reducing the low frequency information, it is used in most image sharpening methods. The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels[4]. Consequently, we see in figure 5 a. that our high pass filter retains the edges where the sharp changes in neighboring colors occur (that is, between the boundaries of the printer, the wall and the cat).

The frequency response of the high pass filter looks different from that of low pass filter earlier in that bright area in the center of the image—representing the low level frequencies—is no longer dominant. Instead, we see many diagonal lines crossing the center; those diagonal lines represent the edges in the original image. If we look carefully, we can see that the there are diagonal lines, which represent Cheshire cat's body stripes, in the top-left and the bottom-right quadrants of the frequency response image.

When we processed the image using a 3x3 median filter, we obtained the result shown in figure 6 a). In image processing, median filtering is a nonlinear operation to reduce "salt and pepper" noise. A median filter is useful when the goal is to simultaneously reduce noise and preserve edges[5]. However, since our original image does not have much noise to begin with, the effect of median filtering is not noticeable—except that the resulting image is slightly blurry than the original one.

The frequency response of the median filter cannot be obtained just as we did in Gaussian case. However, we can approximate it by using a mathematical calculation as shown below.

$$I_{\text{processed with median filter}} = \text{Median Filter} * I_{\text{original image}}$$
$$\text{Median Filter} = I_{\text{processed with median filter}} \;./\; I_{\text{original image}}$$

When calculating approximation of median as shown above, we added a small value—in our case, it is '0.00001'—to each element of $I_{\text{original image}}$ to avoid the "division by zero" error. The frequency response of the resulting matrix is shown in figure 6.

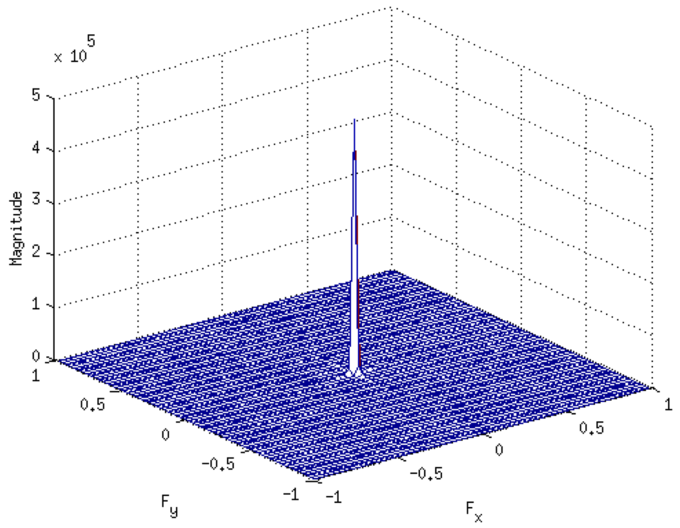


Figure 6: Approximation of the frequency response of median filter

As shown in figure 7, we found that the Sobel edge detector[6] performs reasonably well in detecting the edges in the original image.

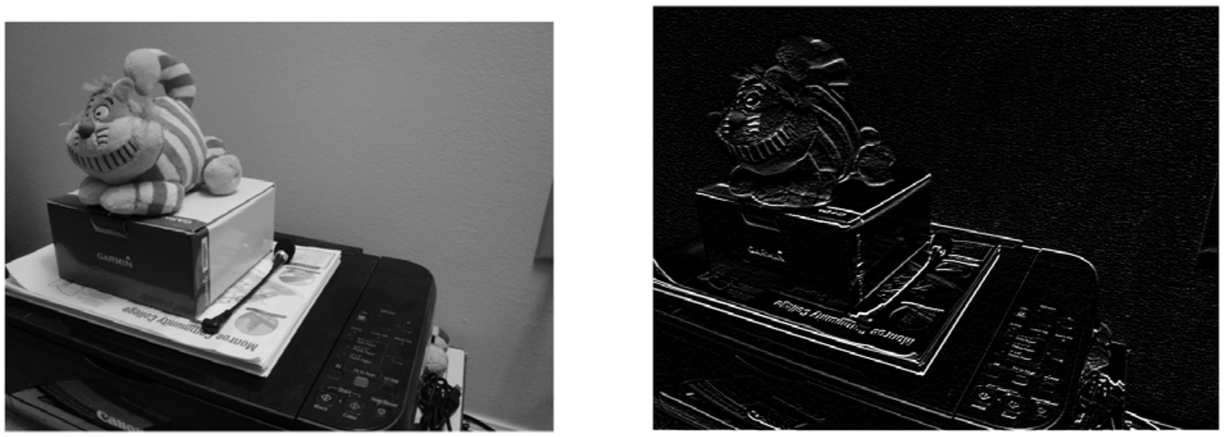


Figure 7: Sobel edge detector finds most of the edges in the image (right) in the absence of noise.

[Note: For 2 f-h, we load the image once and carry all the filtering results from one step to next. That is, we applied the filters to the same image without reloading and this effect will be noticeable in the results.]

Next, we added Gaussian noise to the images we took under different light settings, and applied the Gaussian low pass filter, the 3x3 high pass filter (mentioned in the previous section), and the median filter. We then applied the Sobel edge detector to the resulting images. The kernels for the Sobel detector and the results are shown in figure 8 and 9 respectively.

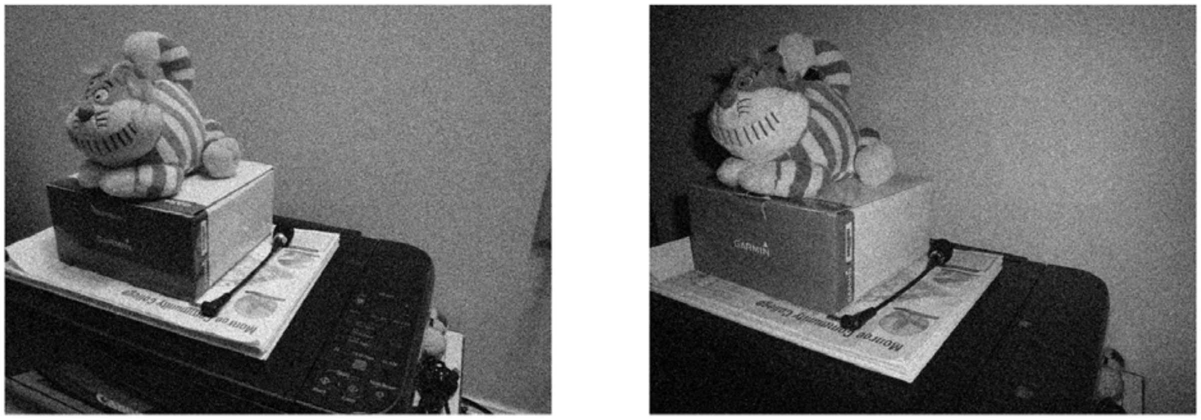
-1	0	1
-2	0	2
-1	0	1

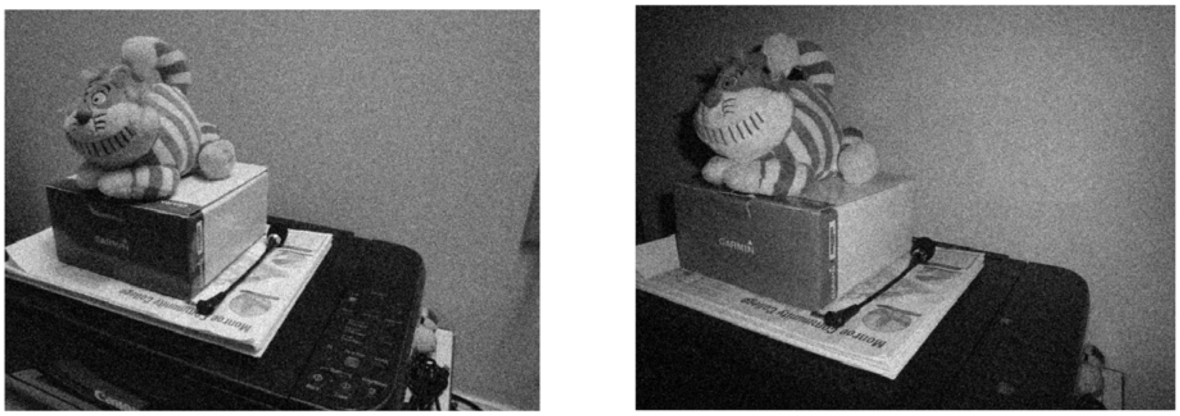
1	2	1
0	0	0
-1	-2	-1

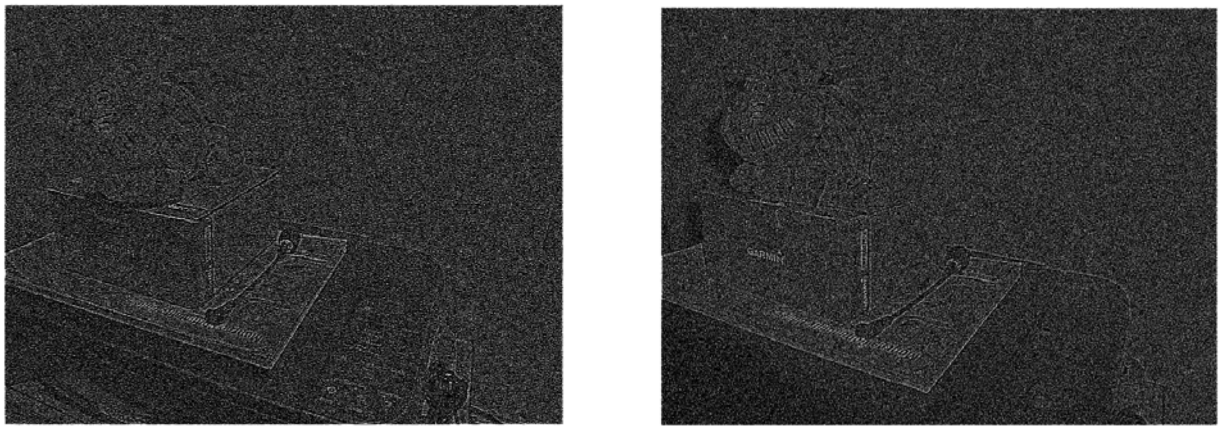
Figure 8: Kernels for Sobel edge detector. These kernels are applied to the image using Matlab's *filter2* command to achieve the edge detection.

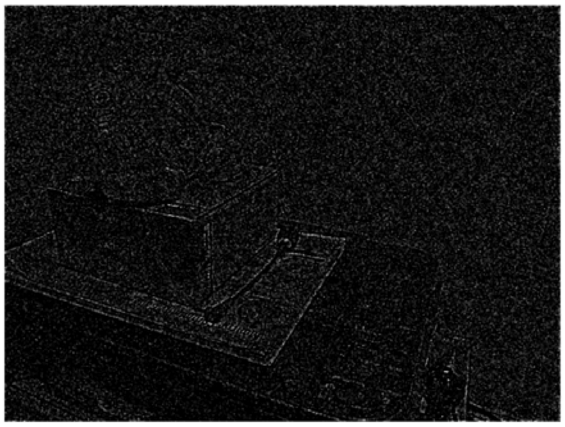
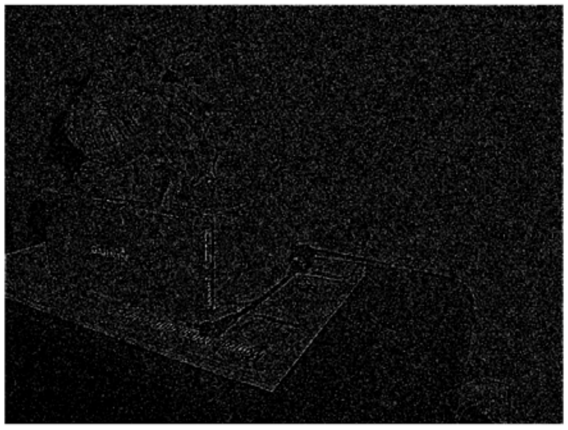
We observed that as we applied the Gaussian low pass filter, the image becomes smoother—that is, the edges and sharp features becomes less distinct—and more blurry than the step before. This reduced the Gaussian noise and consequently, helped the Sobel edge detector to detect edges to an acceptable degree.

Step 1. Original images with normal room light (left) and with camera flash (right)

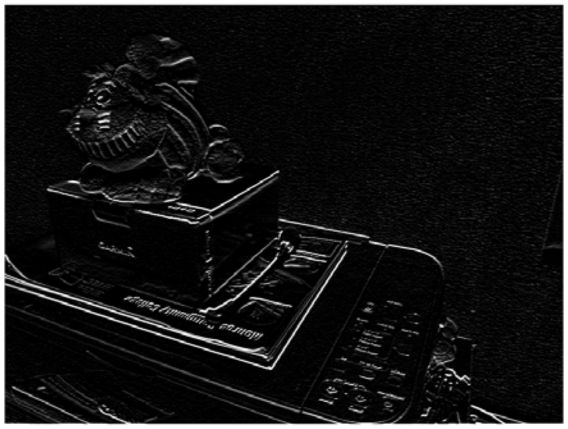
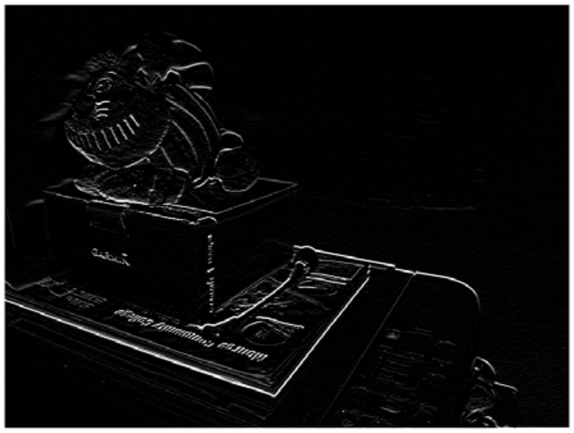

Step 2. After Gaussian noise was added


Step 3. After 3x3 Gaussian filter was applied


Step 4. After the high pass filter was applied

Step 5. After the median pass filter was applied

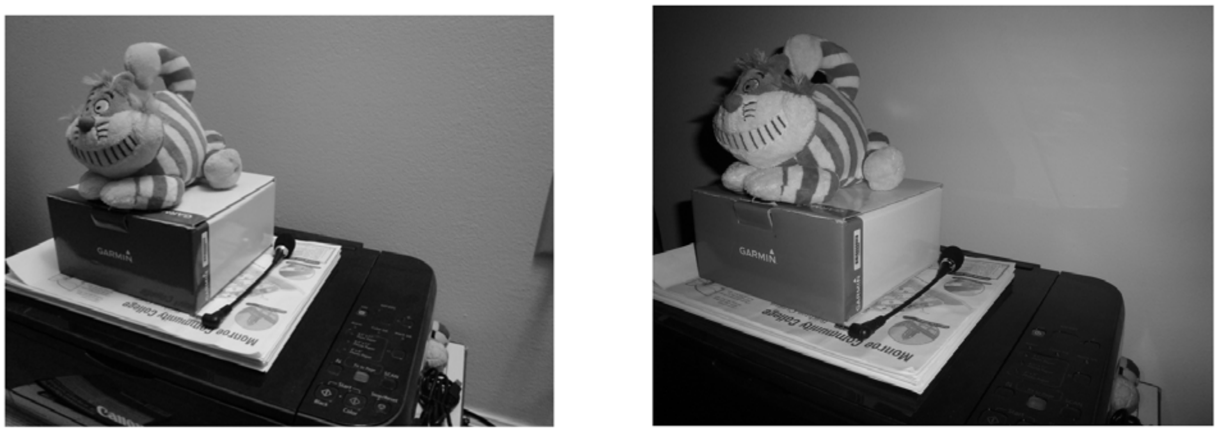
Step 6. After applying Sobel edge detector to the filtered images

Figure 9: Step by step comparison of applying impulse noise and different filters to the image taken under different light conditions—normal room light and camera flash light.

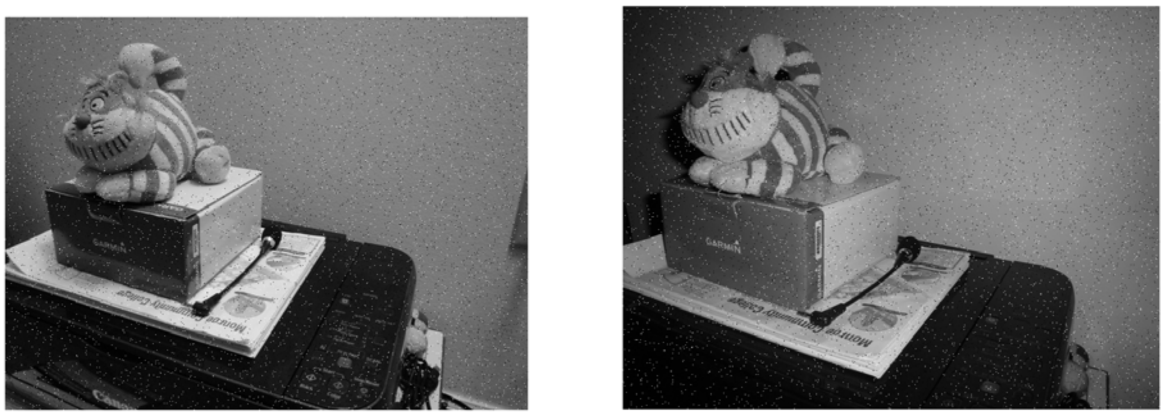
Although Sobel edge detector is not immune to noise, it performed reasonably well in detecting the edges thanks partly to the filtering performed steps prior. When we applied Gaussian low pass filter, it removed some of the Gaussian noise that were added to the original image. Applying high pass filter enhanced the remaining high frequency information—that is, edges and sharp features in the images. Yet, this might have also have enhanced the existing noise. The median filter suppressed them while retaining some high frequency content. This allowed the Sobel edge detector to detect all the major edges in the image.

We also observed that the Sobel operator produces considerably higher output values for edges. As a result, natural edges in images often lead to lines in the output image that are several pixels wide due to the smoothing effect of the Sobel operator. Some thinning may be desirable to counter this.

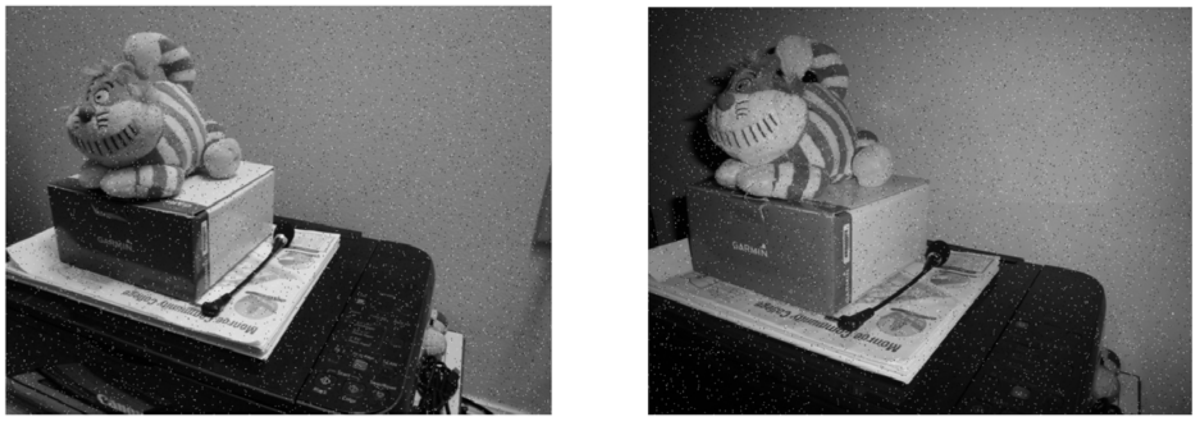
Similarly, we added salt and pepper noise—a form of impulse noise[7]—to the image and repeatedly applied different filters to it. We provide the comparison of edge detection performances along the stages and our assessment of it in figure 10 below.



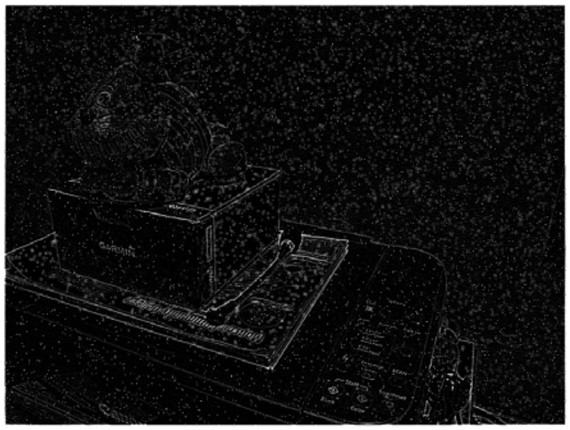
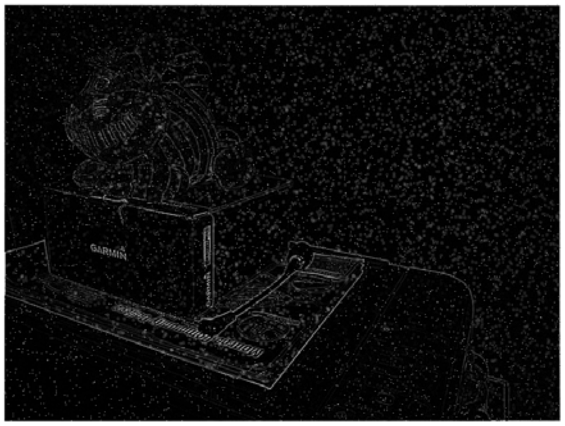
Step 1. Original images



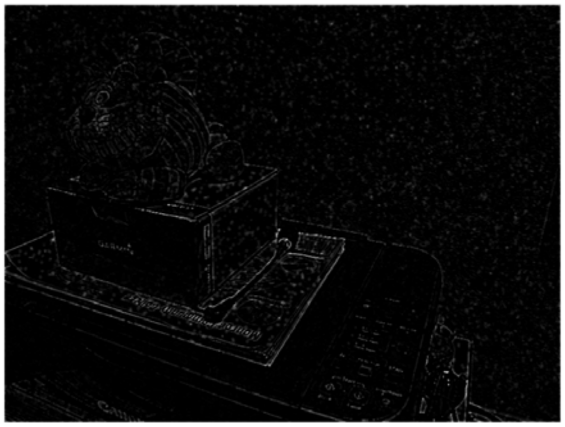
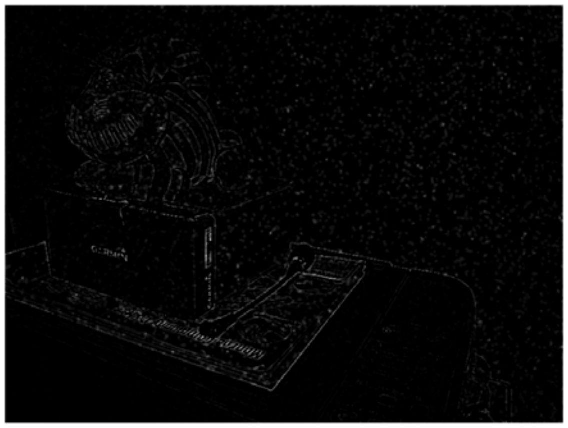
Step 2. After impulse noise was added



Step 3. After 3x3 Gaussian filter was applied

Step 4. After applying the high pass filter

Step 5. After applying the median filter

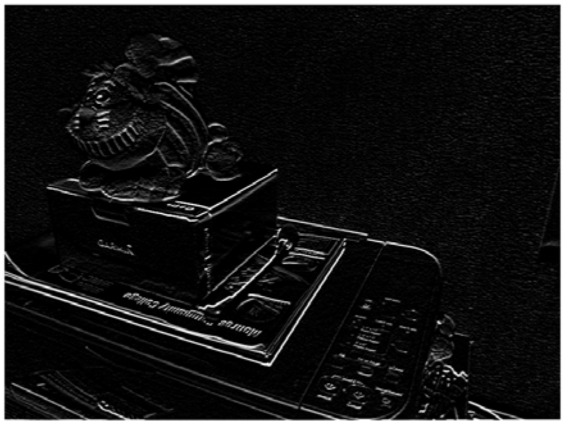
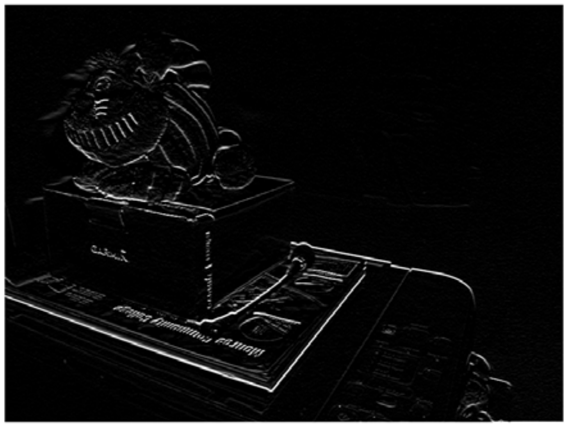



Figure 10: Step by step comparison of applying Gaussian noise and different filters to the image taken under different light conditions—normal room light and camera flash light.

Adding impulse noise affected the performance of the Sobel edge detector much more than it did in applying Gaussian noise. After applying Gaussian filter twice—3x3 and its 1D components—the noise level was reduced to a substantial level; yet, significant degree of noise remains in the image. When the high pass filter was introduced, it seemed to have introduced more noise However, we were surprised to find that most of the added noise in the previous steps was removed once the median filter was applied, thereby resulting in a superior edge detection performance from the Sobel edge detector than we had in the Gaussian noise case.

From our experience of applying different filters to the image, we are convinced that the Sobel edge detector is not immune to noise, and that certain filtering technique should be applied before running Sobel edge detector to have a reliable performance. In addition, we observed that all smoothing techniques are effective at removing noise in smooth patches or smooth regions of a signal, but adversely affect edges. However, we notice that median filter, under certain conditions, preserves edges while removing noise[8,9]. Therefore, we propose that median filter should be applied to images when high pass filtering or detecting edges.

We would also like to propose that the original image should be convolved with a low pass filter to remove all Gaussian-like noises. Nevertheless, we should be careful in doing so because smoothing filters tend to blur an image—that is, pixel intensity values that are significantly higher or lower than the surrounding neighborhood would "smear" across the area.

The lesson from this image filtering experience is that one should always observe and consider the nature of noise embedded in the image before proceeding with edge detection.

PART 3

Relevant File(s)

three.m - script to run part 3 in its entirety

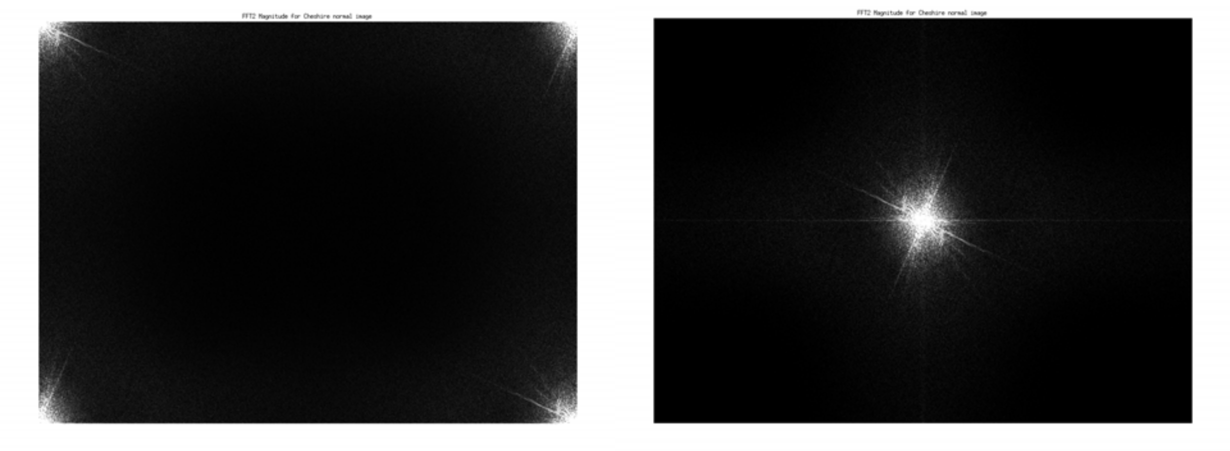
Methods

We used Matlab's command *fft2* to compute the 2-D fast Fourier transform (2-D FFT). We invoked *abs* function to retrieve the magnitude of the 2-D FFT and *ffshift* command to automatically relocate the origin of the magnitude image to the center. We then calculated the logarithm of the magnitude using Matlab's *log* function and displayed the resulting plots.

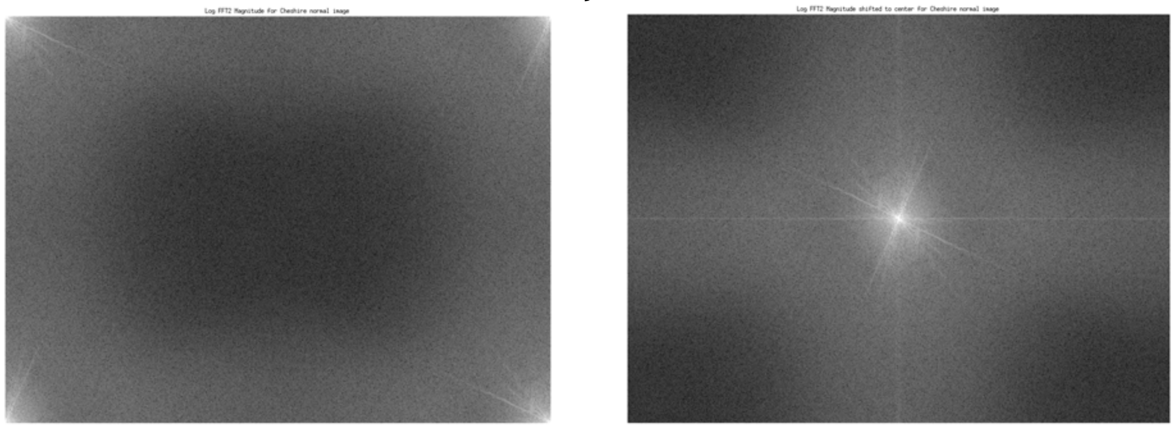
Results and Discussion

The function *fftshift* was used shift the quadrants of the FFT around to see the lowest frequencies in the center of the plot. *fftshift* only had to be called once, as it automatically shifts the quadrants around for 2D matrices[10].

We present the magnitude images obtained from 2-D FFT and its log counter part in figure 12 below.



a)



b)

Figure 12: a) The magnitude of 2-D FFT with (left) and without (right) using *ffshift* function. b) The logarithm of the magnitude shown with (left) and without (right) using *ffshift*.

As can be seen from figure 12, the logarithm plot brings out the subtle shades in the corners and along the axes, which are not visible in its counterpart (non-logarithmic) plot. We observed that the magnitude of the 2-D FFT hovers around the range of 10^6; as a result, when plotting the non-log magnitudes, we had to experiment with different axes limits to fit all the data points in the plot. In the case of log values, not only were we able to get every data point fit in a smaller frame of axis, but the scaled down magnitudes were within the right range for Matlab to display them to a more detailed level as shown in figure 12 b. Therefore, we would prefer the display of 2-D FFT with its magnitude set in logarithmic scale.

References
[1] Eddins, Steve. “Separable convolution”. Matlab® Central. 4 October 2006. 24 February 2012. URL: http://blogs.mathworks.com/steve/2006/10/04/separable-convolution/

[2] Eddins, Steve. “Separable convolution: Part 2”. Matlab® Central. 28 November 2006. 24 February 2012. URL: http://blogs.mathworks.com/steve/2006/11/28/separable-convolution-part-2/

[3] “Singular value decomposition”. Wikipedia. 24 February 2012. 24 February 2012. URL: http://en.wikipedia.org/wiki/Singular_value_decomposition

[4] “Filtering an Image”. NASA. The IDL Astronomy User's Library. 6 March 2007. 24 February 2012. URL: http://idlastro.gsfc.nasa.gov/idl_html_help/Filtering_an_Imagea.html

[5] “2-D median filtering”. MATLAB®. 2012. 24 February 2012. URL: http://www.mathworks.com/help/toolbox/images/ref/medfilt2.html

[6] R. Fishter, S. Perkins, et al. “Feature Detectors - Sobel Edge Detector”. 2003. 24 February 2012. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[7] “Salt and pepper noise”. Image noise. Wikipedia. 11 February 2012. 24 February 2012. URL: http://en.wikipedia.org/wiki/Image_noise

[8] E. Arias-Castro and D.L. Donoho, "Does median filtering truly preserve edges better than linear filtering?", Annals of Statistics, vol. 37, no. 3, pp. 1172–2009.

[9] G.R. Arce, "Nonlinear Signal Processing: A Statistical Approach", Wiley:New Jersey, USA, 2005.

[10] “Shift zero-frequency component to center of spectrum”. MATLAB®. 2012. 24 February 2012. URL: http://www.mathworks.com/help/techdoc/ref/fftshift.html

[11] E. Verner. “How to Do a 2-D Fourier Transform in Matlab”. Matlab Geeks. 20 June 2011. 24 February 2012. URL: http://matlabgeeks.com/tips-tutorials/how-to-do-a-2-d-fourier-transform-in-matlab/