Lab 1: Filtering

Daniel Sherman (0954083), *Student, University of Guelph*

***Abstract*—Average, Gaussian, Ideal Low Pass, and a custom filter were investigated to observe their effects on test images. Window size and standard deviation were varied when testing. The effects of noise were observed when filtering as well. It was seen that to see noise filtering effects easily, the window size and standard deviation of the filters needed to be so large that a lot of blurring was observed in the images, and that an inverse filter amplified noise in an image.**

***Index Terms*—Imaging, Filtering, Average Window, Gaussian, Ideal Low Pass, Noise, Inverse**

I. Introduction

DUE to the complicated nature of images compared to a 1D signal such as an audio clip, a differential equation representation (and therefore an analog representation) is impractical and near impossible to reliably create. As a result of this, images are mostly, if not always processed in the digital domain. Any operation done on an image, such as filtering, needs to be done in the digital domain.

Many operations in image processing are done based off frequencies. For 1D signal processing, a low-pass frequency cutoff filter is quite a common operation to use. Image processing is no different. In an audio signal, where a high frequency would relate to the pitch of a note, a frequency in an image relates to a difference between two pixels, a unit area in an image. A high frequency in an image relates to a sharp difference between adjacent pixels, and a drastic colour difference, as an example changing from all white to all black in a short amount of space. A low frequency in an image would gradually transition from white, to grey, to black.

Noise in images has the effect of muddling up the frequency content. Where in the actual image high frequency content would be, noise can distort the high frequencies, lowering the overall average frequency. The same effect can happen for low frequencies, just in reverse. Where there would be a group of grey pixels, an increase in black pixels or white pixels can increase the average frequency content in an area.

An average filter is a specifically applied low pass filter (1). It calculates the average intensity over a number of pixels in an image. To easily iterate through a whole image, the window that is selected for an average should be an odd number. Most commonly, FIR average filters are used to avoid the complex calculations that come from feedback loops involved with IIR filters (1). In imaging, average filters have the tendency to blur sharp features in images due to its low pass nature.

Gaussian filters can be effective in noise suppression, though it has a tendency to distort the original signal where there are abrupt changes in brightness (2). However, a large enough variance can be selected to be effective at eliminating high frequency noise, maintaining edges (2).

Lab report received January 30, 2020

A radially symmetric filter in Fourier Space can be thought of as an ideal low pass filter in 2D space. As the Fourier transform of an ideal low pass box is a sinc function, the radially symmetric filter is effective at smoothing an image. However, as the sinc function has periodic lobes to it, there is an amplification at multiples of the fundamental frequency. In the image, this is seen as ”ringing” in the features of the image.

In this lab, average, Gaussian, radially symmetric, and exponential filters were investigated to filter features in a checkerboard test image, and an MRI image. The effects of noise was investigated with the performance of each aforementioned filter. Finally, inverse filters were investigated in order to undo image filtering effects.

II. Methods

Test images were obtained and loaded into MATLAB, which can be seen in Figure 1 and Figure 2.

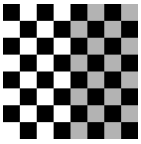

Fig. 1: Test image ’checker.png’ (128x128 Pixels)

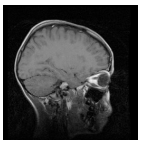

Fig. 2: Test image ’mri.jpg’ (256x256 Pixels)

The above test images were filtered using low pass box filters (window size: 3x3, 5x5, 9x9), Gaussian low pass filters with standard deviation of 1.7, 3, and 5 (window size: 11x11), and radially symmetric ideal frequency domain low pass filters (cutoff radii of 0.1, 0.03, and 0.05 times the image size).

The filtered images were corrupted with random noise ($\mu = 0, \sigma^2 = 100, 900$) and the above filters were applied to the noisy images.

$$h(x, y) = k(0.8)^{(|x|+|y|)} \tag{1}$$

A custom low pass filter was derived to have the point spread function seen in Equation 1, seen in section B. The test images were passed through this filter, and then passed through the inverse filter. Noise was added to the result of the custom low pass filter ($\mu = 0, \sigma^2 = 25, 100$) for each test

image and were passed through the inverse filter to observe the results.

All code used can be seen in section C.

III. Results and Discussion

When applying an average filter, as the size of the filter increased, more features would blur, as seen in Figure 3 and Figure 4. This is to be expected, because as more pixels are included in the average, more pixels will be made to be close to the same value (the local average), which is easily seen as pixels in Figure 3 that were originally white turning more grey in the filtered image.

For the Gaussian filter, a window size of 11x11 needed to be used to see meaningful results in the filtered images (Figure 5 and Figure 6). As the standard deviation of the filter increases, more edge blurring is seen, which is shown as thicker grey edges in Figure 5 when the standard deviation is 5, rather than 1.7. This is expected as when the standard deviation increases, there is a more gradual increase in the amplitude of the Gaussian curve. The large blurring effect is attributed to the large (11x11) window size of the filter. The window size needed to be selected to be large enough to visibly see a difference in the images before and after filtering. A consequence of this is that if the window is too large, the image will be over blurred, which seen in Figure 5.

When using the radially symmetric Ideal Low Pass filter, a ringing effect in the brightness of the image can be seen more easily for the MRI image than the checkered image. The ringing effect is best seen for the filters that are 0.1 and 0.05 times the size of the image (Figure 10 and Figure 12). The ringing effect is due to the fact that the Fourier Transform of the Ideal Low Pass filter is the sinc function. The lobes on either side of the sinc function cause the ringing effect, as the lobes negate and amplify multiples of the fundamental frequency of the filter. A border can be seen in Figure 7, Figure 8, and Figure 9, as the images were not padded when filtered in the frequency domain. A black border is not seen in any of the MRI images, as the edge of those images is already black. If the lab were to be redone, padding should have been done to eliminate this effect.

When corrupted with noise, the average filter with larger windows do a more effective job at filtering out noise, especially when the standard deviation for the noise is lower (Figure 14). For images with noise that have a larger standard deviation, after filtering noise remains (Figure 16). This can be because noise with a larger standard deviation in a particular pixel will be more different in amplitude than noise in a nearby pixel, making the average of the group of pixels unlike the average of the pixels without noise. Comparing the filtered images, the noise has no discernible effect on the amount of blurring there is for the checker or MRI image (Figure 3, Figure 4, Figure 14, Figure 16, Figure 15, Figure 21). It is possible that in the MRI image, the pixel value changes enough in a small area that adding a bit of noise does not corrupt the image as much as the extremely ordered checker image.

For the corrupted images that were Gaussian filtered, very minimal noise passes through the filtering, however much of the image is blurred (Figure 18, Figure 20, Figure 19, Figure 21). The filtered images with noise appear to be incredibly similar to the filtered images without noise (Figure 5, Figure 6). The large window size (necessary to see differences before and after filtering) contribute to over filtering, and as a result, the images are incredibly blurry.

The filtered checker images with noise appear the same as the checker images that did not have noise, when being filtered with a radially symmetric ideal low pass filter with radius of 0.1 times the image size (Figure 7, Figure 22, Figure 28). However for the same filter, noise remains in the MRI image after filtering (Figure 10, Figure 25, Figure 31). The same effect happens for the checker images filtered through an ideal low pass filter with size of 0.03 times the image size; the filtered images with and without noise appear the same (Figure 8, Figure 23, Figure 29). The filtered noisy MRI images look completely blurred, and are difficult to make out features (Figure 11, Figure 26, Figure 32). The noisy checker images filtered through an ideal low pass filter with size of 0.05 times the image size appear the same whether or not they had noise (Figure 9, Figure 24, Figure 30). For the noisy MRI images filtered through the ideal low pass filter of size 0.05 times the image size, the ringing effect appears less defined as the noise standard deviation increases (Figure 12, Figure 27, Figure 33).

In general, the higher the variance of the noise, the more noise remains after filtering. Furthermore, the larger the standard deviation of the noise, the more the image will be blurred after filtering. The rand() function was used instead of the imnoise() function to add noise because they call their noise values from very slightly, but different probability distributions. The rand() function calls numbers from the standard normal distribution (3). The imnoise() function calls numbers from a uniformly distributed probability function (4).

When applying the custom low pass filter (Equation 1) with a window size of 31x31 (to see the desired effects), the image becomes blurry, which in itself is nothing special or interesting. What is really interesting is that when the inverse filter is applied to the blurry images, the original images are returned (Figure 34 and Figure 35). This is interesting, but to be expected because the inverse operation is being applied to the image (derived in section B). This is because the inverse filter, $H^{-1}(z)$ is defined to be $\frac{1}{H(z)}$, where *H(z)* is the original filter. Applying the original filter and the inverse filter is equivalent to multiplying the frequency domain representation of the image by the original filter and its inverse in the frequency domain, where $\frac{1}{H(z)}$ and *H(z)* cancel out to be 1, which does not change the image at all.

Another interesting effect is seen when filtering noisy images through the low pass filter and its inverse: the noise was amplified (Figure 36, Figure 38, Figure 37, and Figure 39). A possible explanation of this is that the high frequency components of the noise are not divided out by the inverse filter, causing the large amplification of noise when the low frequency content is removed by the inverse filter.

IV. Conclusion

The average filter blurred more features as the window size increased, due to the low pass behaviour of the filter. The Gaussian filter exhibited the same behaviour, but for standard deviation rather than window size. A window of 11x11 needed to be used to see meaningful results. The ideal low pass filter left ringing effects in the images due to the periodic effect of the sinc function, repeating multiples of the fundamental frequency.

Adding noise to the images did not change the behaviour of the filters very much, except for adding a little more blurring. However, when the standard deviation of the noise increased, more of the noise remained in the filtered image.

When applying the custom low pass filter and its inverse, for the clean images, the original image was exactly returned. However, when the images were noisy, the original images were not exactly returned, as the noise pattern was different. This can be attributed to the noise being amplified in the filtered image, and due to the fact that MATLAB applies the low pass filter first, then applies the inverse.

Appendix A
Image Results

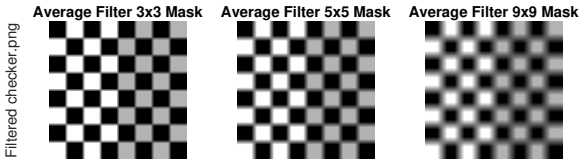

Fig. 3: 'checker.png' filtered through average filters of varying sizes

Fig. 4: 'mri.jpg' filtered through average filters of varying sizes

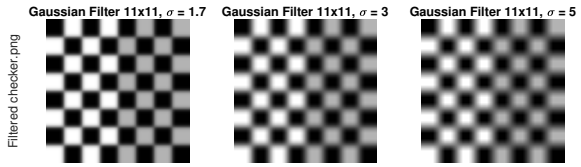

Fig. 5: 'cheker.png' filtered through an 11x11 Gaussian filter with varying Variances

Fig. 6: 'mri.jpg' filtered through an 11x11 Gaussian filter with varying Variances

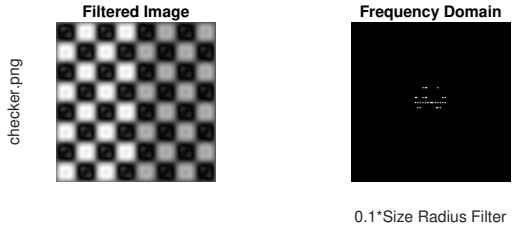

Fig. 7: Filtered Image (Left) and Frequency Domain Representation (Right) of 'checker.png' filtered through a radially symmetric filter of size 0.1 times the image size

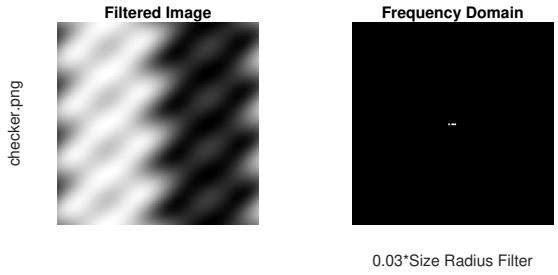

Fig. 8: Filtered Image (Left) and Frequency Domain Representation (Right) of 'checker.png' filtered through a radially symmetric filter of size 0.03 times the image size

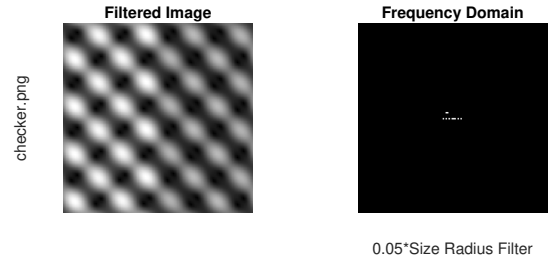

Fig. 9: Filtered Image (Left) and Frequency Domain Representation (Right) of 'checker.png' filtered through a radially symmetric filter of size 0.05 times the image size

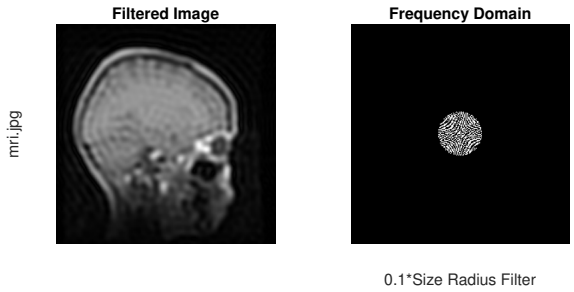

Fig. 10: Filtered Image (Left) and Frequency Domain Representation (Right) of 'mri.jpg' filtered through a radially symmetric filter of size 0.1 times the image size

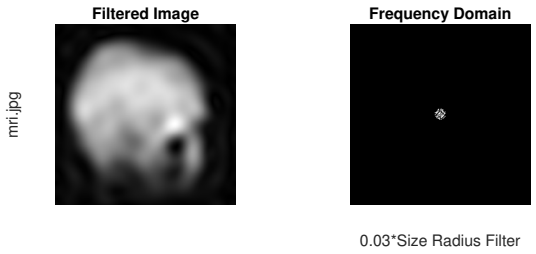

Fig. 11: Filtered Image (Left) and Frequency Domain Representation (Right) of 'mri.jpg' filtered through a radially symmetric filter of size 0.03 times the image size

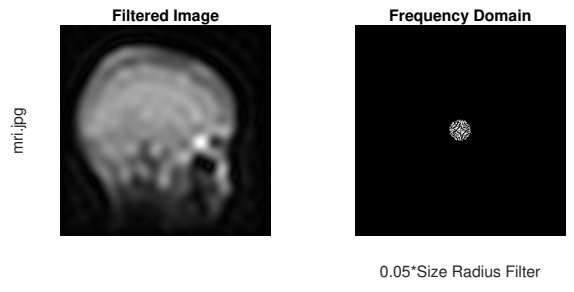

Fig. 12: Filtered Image (Left) and Frequency Domain Representation (Right) of 'mri.jpg' filtered through a radially symmetric filter of size 0.05 times the image size

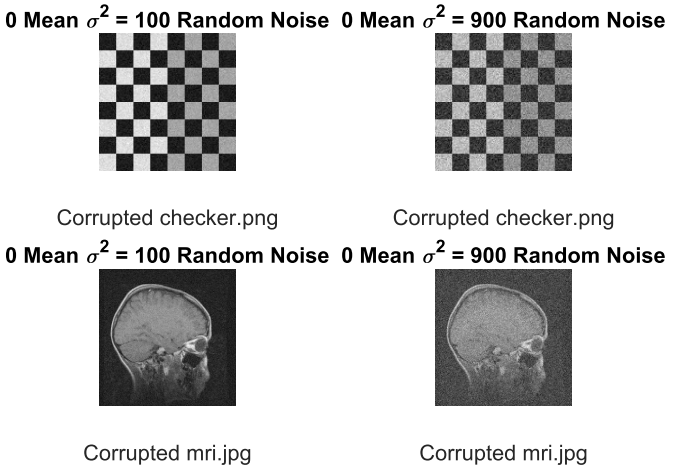

Fig. 13: Test images corrupted with noise of differing Variances

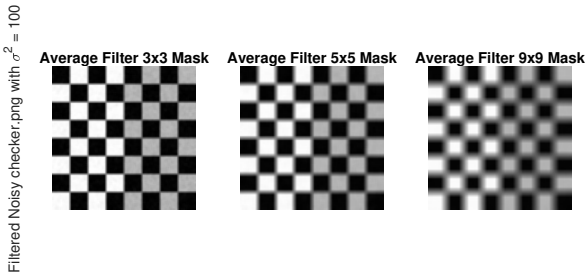

Fig. 14: Average filtered 'checker.png' corrupted with noise of $\sigma^2 = 100$

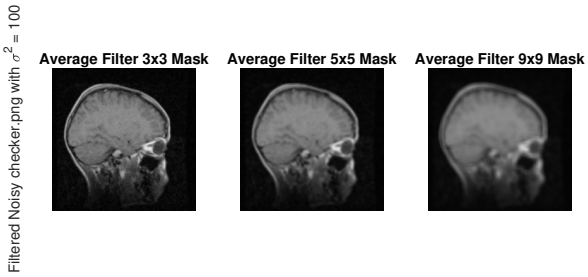

Fig. 15: Average filtered 'mri.jpg' corrupted with noise of $\sigma^2 = 100$

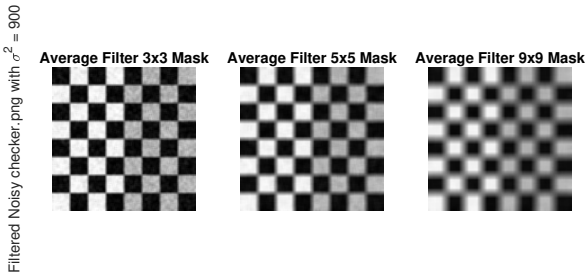

Fig. 16: Average filtered 'checker.png' corrupted with noise of $\sigma^2 = 900$

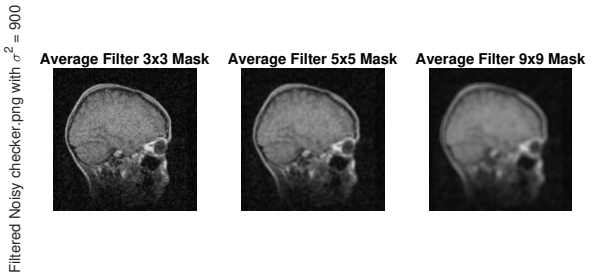

Fig. 17: Average filtered 'mri.jpg' corrupted with noise of $\sigma^2 = 900$

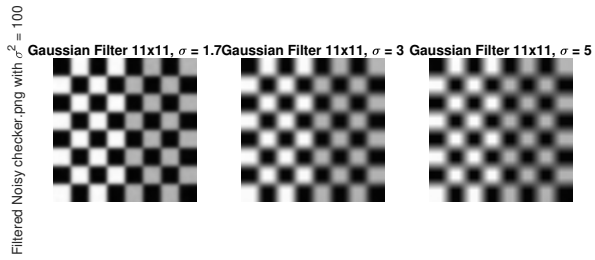

Fig. 18: Gaussian filtered 'checker.png' corrupted with noise of $\sigma^2 = 100$

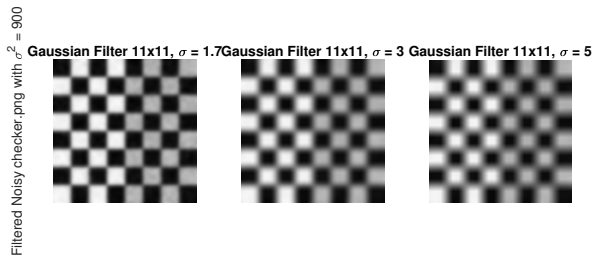

Fig. 19: Gaussian filtered 'checker.png' corrupted with noise of $\sigma^2 = 900$

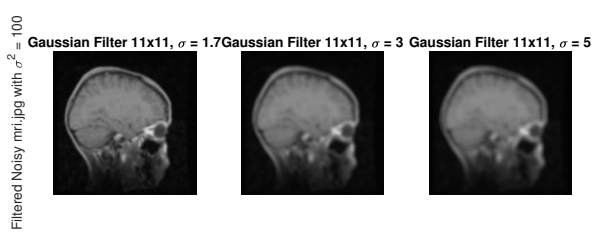

Fig. 20: Gaussian filtered 'mri.jpg' corrupted with noise of $\sigma^2 = 100$

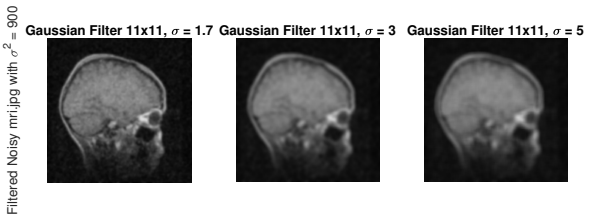

Fig. 21: Gaussian filtered 'mri.jpg' corrupted with noise of $\sigma^2 = 900$

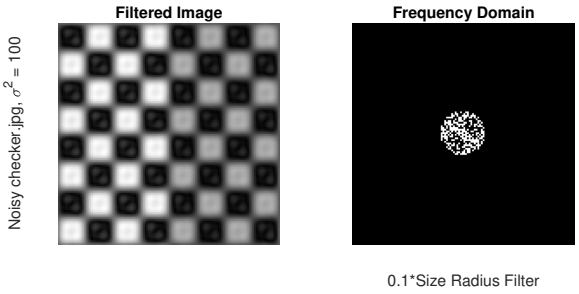

Fig. 22: Radially symmetric filtered (radius = 0.1 times image size) 'checker.png' corrupted with noise $\sigma^2 = 100$

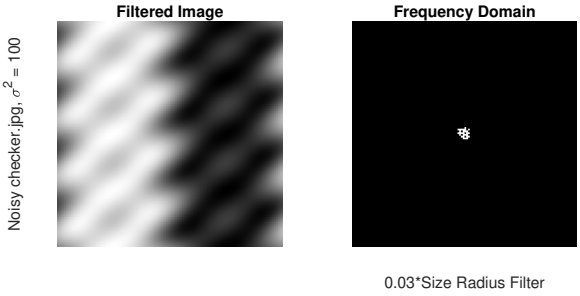

Fig. 23: Radially symmetric filtered (radius = 0.03 times image size) 'checker.png' corrupted with noise $\sigma^2 = 100$

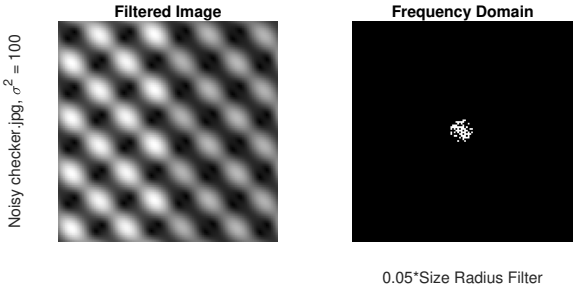

Fig. 24: Radially symmetric filtered (radius = 0.05 times image size) 'checker.png' corrupted with noise $\sigma^2 = 100$

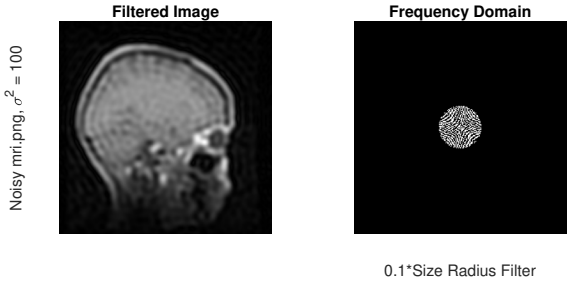

Fig. 25: Radially symmetric filtered (radius = 0.1 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 100$

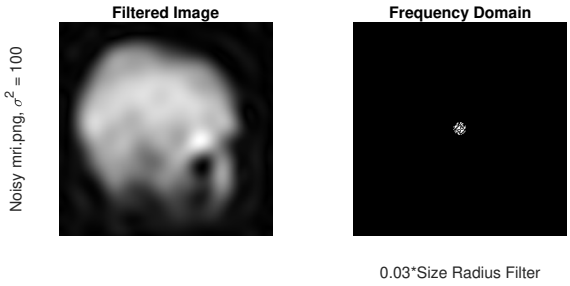

Fig. 26: Radially symmetric filtered (radius = 0.03 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 100$

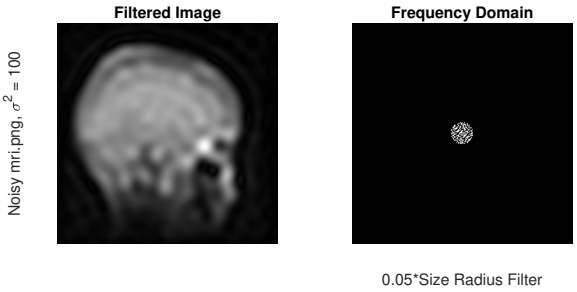

Fig. 27: Radially symmetric filtered (radius = 0.05 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 100$

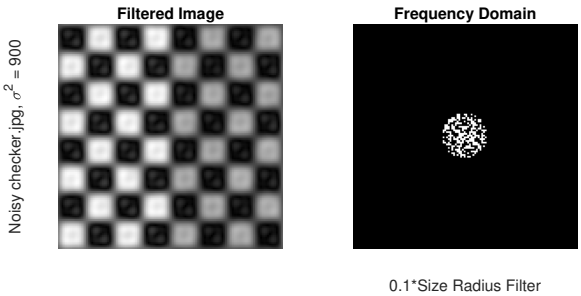

Fig. 28: Radially symmetric filtered (radius = 0.1 times image size) 'checker.png' corrupted with noise $\sigma^2 = 900$

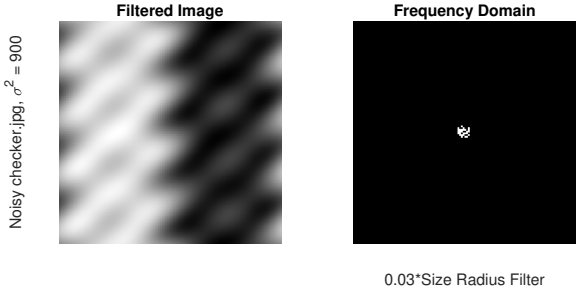

Fig. 29: Radially symmetric filtered (radius = 0.03 times image size) 'checker.png' corrupted with noise $\sigma^2 = 900$

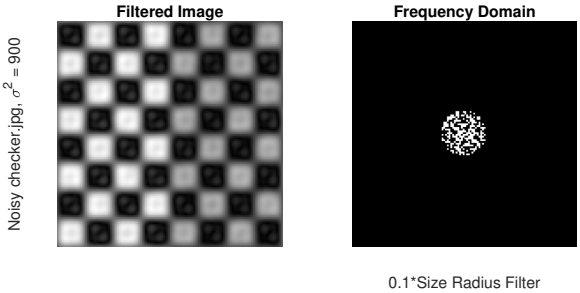

Fig. 30: Radially symmetric filtered (radius = 0.05 times image size) 'checker.png' corrupted with noise $\sigma^2 = 900$

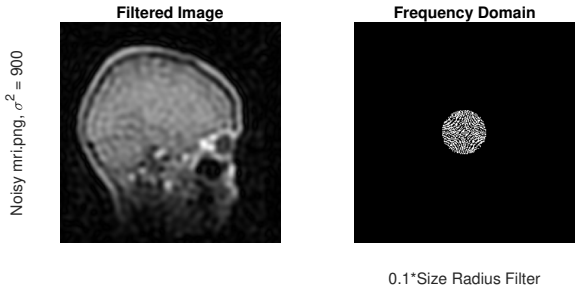

Fig. 31: Radially symmetric filtered (radius = 0.1 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 900$

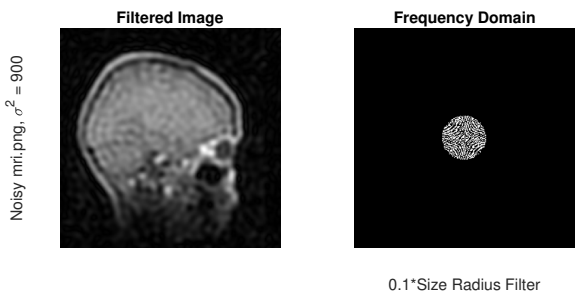

Fig. 32: Radially symmetric filtered (radius = 0.03 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 900$

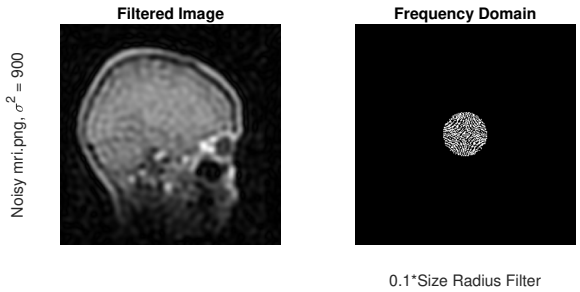

Fig. 33: Radially symmetric filtered (radius = 0.05 times image size) 'mri.jpg' corrupted with noise $\sigma^2 = 900$

Fig. 34: 'checker.png' passed through Low-Pass filter from 1, then passed through its inverse

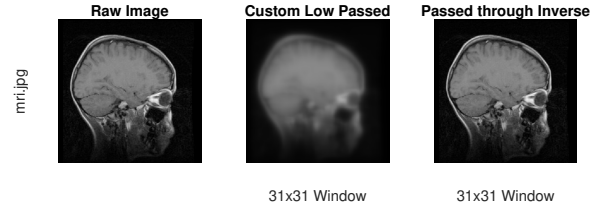

Fig. 35: 'mri.jpg' passed through Low-Pass filter from 1, then passed through its inverse

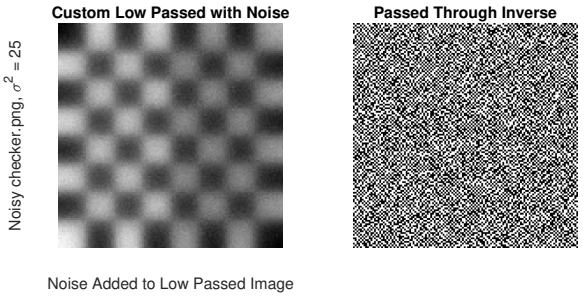

Fig. 36: Noisy 'checker.png' ($\sigma^2 = 25$) passed through Low-Pass filter from 1, then passed through its inverse

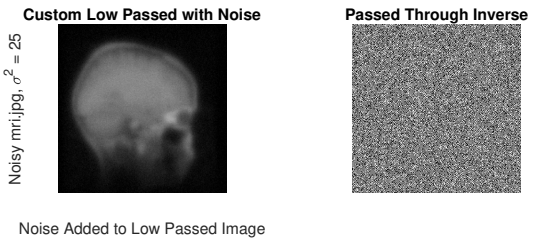

Fig. 37: Noisy 'mri.jpg' ($\sigma^2 = 25$) passed through Low-Pass filter from 1, then passed through its inverse

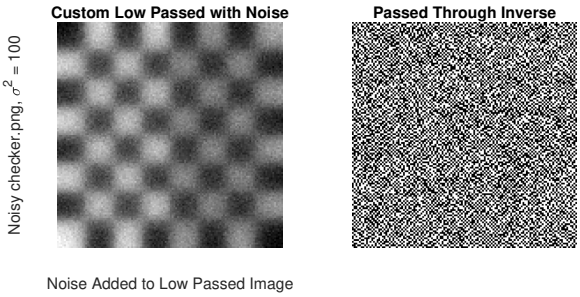

Fig. 38: Noisy 'checker.png' ($\sigma^2 = 100$) passed through Low-Pass filter from 1, then passed through its inverse

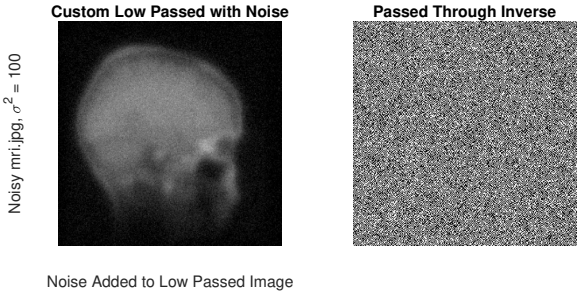

Fig. 39: Noisy 'mri.jpg' ($\sigma^2 = 100$) passed through Low-Pass filter from 1, then passed through its inverse

APPENDIX B
INVERSE FILTER DERIVATION

The inverse filter to Equation 1 is defined below.

$h(x) = k(0.8)^{|n|}$

Including the causal and anti-causal portion of the Z-Transform,

$H(z) = k\sum_{n=-\infty}^{-1}(0.8)^{|n|}z^{-n} + k\sum_{n=0}^{\infty}(0.8)^{|n|}z^{-n}$

$H(z) = k(\sum_{n=-\infty}^{-1}(0.8z)^{-n} + \sum_{n=0}^{\infty}0.8^n(\frac{1}{n})^n)$

$H(z) = k(\frac{0.8z}{1-0.8z} + \frac{1}{1-0.8z^{-1}})$

$H(z) = k(\frac{0.8z(1-0.8z^{-1}+1-0.8z)}{(1-0.8z)(1-0.8z^{-1})})$

$H(z) = k(\frac{0.8z-0.8^2+1-0.8z}{1-0.8z^{-1}-0.8z+0.8^2})$

$H(z) = \frac{k(1-0.8^2)}{-0.8z^{-1}+(1-0.8^2)-0.8z}$

$H^{-1}(z) = \frac{-0.8z+(1-0.8^2-0.8z^{-1})}{k(1-0.8^2)}$

APPENDIX C
MATLAB CODE

Listing 1: Main Code used for filtering images

```
%% ENGG 4660: MEDICAL IMAGE PROCESSING
% LAB 1: FILTERING
% DANIEL SHERMAN
% 0954083
% JANUARY 15, 2020

%% CLEAN UP

close all
clear all
clc

%% START OF CODE

checker = imread('checker.png'); %load 'checker.png'
[r_checker c_checker] = size(checker); %grab size of checker phantom

mri = imread('mri.jpg'); %load 'mri.jpg'
[r_mri c_mri] = size(mri); %grab size of mri image

figure() %display checker.png
imshow(checker)
title('checker.png')

figure() %display mri.jpg
imshow(mri)
title('mri.jpg')

%% FILTER CHECKER AND MRI IMAGE THROUGH AVERAGE FILTERS OF SIZE 3X3, 5X5, AND 9X9

av_filt(checker, 'checker.png')
av_filt(mri, 'mri.jpg')

%% FILTER CHECKER AND MRI IMAGE THROUGH GAUSSIAN FILTERS WITH STANDARD DEVIATIONS 1.7, 3, AND 5

gau_filt(checker, 'checker.png')
gau_filt(mri, 'mri.jpg')

%% DEFINE RADIALLY SYMMETRIC IDEAL FREQUENCY–DOMAIN LOW PASS FILTERS

bit = 8; %define bit size of images used

%create radially symmetric filters of appropriate sizes to use with
%checker.png
rad_c01 = radfilt(checker, 0.1*c_checker, bit);
rad_c003 = radfilt(checker, 0.03*c_checker, bit);
rad_c005 = radfilt(checker, 0.05*c_checker, bit);

%display radially symmetric filters for use with checker.png
figure()
subplot(1,3,1)
imshow(rad_c01, [])
ylabel('Radially Symmetric Checker Filters')
title('Radius = 12.8')
subplot(1,3,2)
imshow(rad_c003, [])
title('Radius = 3.84')
subplot(1,3,3)
imshow(rad_c005, [])
title('Radius = 6.4')

%define radially symmetric filters to be used with mri.jpg
rad_m01 = radfilt(mri, 0.1*c_mri, bit);
rad_m003 = radfilt(mri, 0.03*c_mri, bit);
rad_m005 = radfilt(mri, 0.05*c_mri, bit);

%display radially symmetric filters to be used with mri.jpg
figure()
subplot(1,3,1)
imshow(rad_m01, [])
ylabel('Radially Symmetric MRI Filters')
title('Radius = 12.8')
subplot(1,3,2)
imshow(rad_m003, [])
title('Radius = 3.84')
subplot(1,3,3)
imshow(rad_m005, [])
title('Radius = 6.4')

%% FILTER RADIALLY SYMMETRIC – CHECKER

%filter checker.png with radially symmetric filter radius 0.1*size
rad_filtering(checker, rad_c01, '0.1', 'checker.png')

%filter checker.png with radially symmetric filter radius 0.05*size
rad_filtering(checker, rad_c005, '0.05', 'checker.png')

%filter checker.png with radially symmetric filter radius 0.03*size
rad_filtering(checker, rad_c003, '0.03', 'checker.png')

%% FILTER RADIALLY SYMMETRIC – MRI
```

```
92
93 %filter mri.jpg with radially symmetric
       filter radius 0.1*size
94 rad_filtering(mri, rad_m01, '0.1', 'mri.
       jpg')
95
96 %filter mri.jpg with radially symmetric
       filter radius 0.05*size
97 rad_filtering(mri, rad_m005, '0.05', 'mri
       .jpg')
98
99 %filter mri.jpg with radially symmetric
       filter radius 0.03*size
100 rad_filtering(mri, rad_m003, '0.03', 'mri
       .jpg')
101
102 %% CORRUPT IMAGES WITH GAUSSIAN NOISE
103
104 checker_100 = double(checker) + sqrt(100)
       .*randn(size(checker)); %corrupt
       checker.png with 100 variance noise
105 checker_900 = double(checker) + sqrt(900)
       .*randn(size(checker)); %corrupt
       checker.png with 900 variance noise
106
107 mri_100 = double(mri) + sqrt(100).*randn(
       size(mri)); %corrupt mri.jpg with 100
       variance noise
108 mri_900 = double(mri) + sqrt(900).*randn(
       size(mri)); %corrupt mri.jpg with 900
       variance noise
109
110 %display noisy test images
111 figure()
112 subplot(4,2,1)
113 imshow(checker_100, [])
114 title('0 Mean \sigma^2 = 100 Random Noise
       ')
115 xlabel('Corrupted checker.png')
116 subplot(4,2,2)
117 imshow(checker_900, [])
118 title('0 Mean \sigma^2 = 900 Random Noise
       ')
119 xlabel('Corrupted checker.png')
120 subplot(4,2,3)
121 imshow(mri_100,[])
122 title('0 Mean \sigma^2 = 100 Random Noise
       ')
123 xlabel('Corrupted mri.jpg')
124 subplot(4,2,4)
125 imshow(mri_900, [])
126 title('0 Mean \sigma^2 = 900 Random Noise
       ')
127 xlabel('Corrupted mri.jpg')
128
129 %% FILTER NOISY IMAGES
130
131 %average filter noisy trest images with
       variance 100
132 av_filt(uint8(checker_100), 'Noisy
       checker.png with \sigma^2 = 100')
133 av_filt(uint8(mri_100), 'Noisy checker.
       png with \sigma^2 = 100')
134
135 %average filter noisy trest images with
       variance 900
136 av_filt(uint8(checker_900), 'Noisy
       checker.png with \sigma^2 = 900')
137 av_filt(uint8(mri_900), 'Noisy checker.
       png with \sigma^2 = 900')
138
139 %gaussian filter noisy trest images with
       variance 100
140 gau_filt(uint8(checker_100), 'Noisy
       checker.png with \sigma^2 = 100')
141 gau_filt(uint8(checker_900), 'Noisy
       checker.png with \sigma^2 = 900')
142
143 %gaussian filter noisy trest images with
       variance 900
144 gau_filt(uint8(mri_100), 'Noisy mri.jpg
       with \sigma^2 = 100')
145 gau_filt(uint8(mri_900), 'Noisy mri.jpg
       with \sigma^2 = 900')
146
147 %radially symmetric filter noisy checker.
       png with variance 100
148 rad_filtering(checker_100, rad_c01, '0.1'
       , 'Noisy checker.jpg, \sigma^2 = 100')
149 rad_filtering(checker_100, rad_c005, '
       0.05', 'Noisy checker.jpg, \sigma^2 =
       100')
150 rad_filtering(checker_100, rad_c003, '
       0.03', 'Noisy checker.jpg, \sigma^2 =
       100')
151
152 %radially symmetric filter noisy mri.jpg
       with variance 100
153 rad_filtering(mri_100, rad_m01, '0.1', '
       Noisy mri.png, \sigma^2 = 100')
154 rad_filtering(mri_100, rad_m005, '0.05',
       'Noisy mri.png, \sigma^2 = 100')
155 rad_filtering(mri_100, rad_m003, '0.03',
       'Noisy mri.png, \sigma^2 = 100')
156
157 %radially symmetric filter noisy checker.
       png with variance 900
158 rad_filtering(checker_900, rad_c01, '0.1'
       , 'Noisy checker.jpg, \sigma^2 = 900')
159 rad_filtering(checker_900, rad_c005, '
       0.05', 'Noisy checker.jpg, \sigma^2 =
       900')
160 rad_filtering(checker_900, rad_c003, '
       0.03', 'Noisy checker.jpg, \sigma^2 =
       900')
```

```

%radially symmetric filter noisy mri.jpg with variance 100
rad_filtering(mri_900, rad_m01, '0.1', 'Noisy mri.png, \sigma^2 = 900')
rad_filtering(mri_900, rad_m005, '0.05', 'Noisy mri.png, \sigma^2 = 900')
rad_filtering(mri_900, rad_m003, '0.03', 'Noisy mri.png, \sigma^2 = 900')

%% CREATE CUSTOM LOW PASS FILTER

%apply the custom low pass filter and inverse to test images
inverse_filter(checker, 'checker.png')
inverse_filter(mri, 'mri.jpg')
```

Listing 2: Function that defines an average filter and applies it

```
function av_filt(img, name)
%% DOCUMENTATION

% FUNCTION TAKES AN IMAGE AND A FILE NAME.
% FILTERS IMAGE THROUGH AVERAGE FILTERS OF SIZE 3X3, 5X5, AND 9X9 AND PLOTS

% MADE BY: DANIEL SHERMAN
% JANUARY 20, 2020

%% START OF CODE

%define and apply average filters of size 3x3, 5x5, and 9x9
img_3 = imfilter(double(img), fspecial('average', [3 3]), 'replicate');
img_5 = imfilter(double(img), fspecial('average', [5 5]), 'replicate');
img_9 = imfilter(double(img), fspecial('average', [9 9]), 'replicate');

%plot filtered images
figure()
subplot(1,3,1)
imshow(uint8(img_3))
title('Average Filter 3x3 Mask')
ylabel(strcat(['Filtered' ' ' name]))
subplot(1,3,2)
imshow(uint8(img_5))
title('Average Filter 5x5 Mask')
subplot(1,3,3)
imshow(uint8(img_9))
title('Average Filter 9x9 Mask')
```

Listing 3: Function that defines a Gaussian filter and applies it

```
function gau_filt(img, name)
%% DOCUMENTATION

% FUNCTION TAKES AN IMAGE AND A FILE NAME.
% FILTERS IMAGE THROUGH GAUSSIAN FILTERS WITH STANDARD DEVIATIONS 1.7, 3,
% AND 5 AND PLOTS

% MADE BY: DANIEL SHERMAN
% JANUARY 20, 2020

%% START OF CODE

%define and apply gaussian filters of standard deviation 1.7, 3, and 5
img_G17 = imfilter(double(img), fspecial('gaussian', [11 11], 1.7), 'replicate');
img_G3 = imfilter(double(img), fspecial('gaussian', [11 11], 3), 'replicate');
img_G5 = imfilter(double(img), fspecial('gaussian', [11 11], 5), 'replicate');

%plot filtered images
figure()
subplot(1,3,1)
imshow(uint8(img_G17))
title('Gaussian Filter 11x11, \sigma = 1.7')
ylabel(strcat(['Filtered' ' ' name]))
subplot(1,3,2)
imshow(uint8(img_G3))
title('Gaussian Filter 11x11, \sigma = 3')
subplot(1,3,3)
imshow(uint8(img_G5))
title('Gaussian Filter 11x11, \sigma = 5')
```

Listing 4: Function developed with Z. Szentimrey that defines a radially symmetric Frequency-Domain Ideal Low Pass filter

```
function B = radfilt(I, radius, bit)
%% DOCUMENTATION

% DANIEL SHERMAN
% JANUARY 13, 2020
%
% FUNCTION ACCEPTS IMAGE I, DOUBLE radius, AND INTEGER bit AND CREATES AN
% IDEAL LOW PASS CIRCULAR FILTER

%% BEGIN CODE

[row, column] = size(I); %find out size of image

B = uint8(zeros(row, column)); %define
```

```
    initial matrix of zeros to populate
15
16 for i = 1 : row
17     for j = 1 : column
18         d = sqrt((i - round(row/2))^2 + (
               j - round(column/2))^2); %
               define circle which will be
               the filter
19         if (d < radius) %check if a pixel
               is within the defined circle
20             B(i,j) = 2^bit - 1; %if it is
                   , set to the highest bit
                   value we can have
21         else
22             B(i,j) = 0; %otherwise, leave
                   as 0
23         end
24     end
25 end
```

Listing 5: Function that applies the radially symmetric Frequency-Domain Ideal Low Pass filter

```
1  function rad_filtering(img, filter, size,
       name)
2  %% DOCUMENTATION
3
4  % FUNCTION TAKES AN IMAGE, SIZE OF
       ORIGINAL IMAGE AND THE RADIALLY
       SYMMETRIC FILTER TO BE USED.
5  % FILTERS IMAGE THROUGH RADIALLY
       SYMMETRIC FILTER AND PLOTS
6
7  % MADE BY: DANIEL SHERMAN
8  % JANUARY 20, 2020
9
10 %% START OF CODE
11
12 img_fft = double(fftshift(fft2(img))); %
       take and shift fft appropriately
13
14 img_filter_fft = img_fft.*double(filter);
       %filter in frequency domain with the
       appropriately sized filter
15 img_filter_spat = ifft2(fftshift(
       img_filter_fft)); %go back to spatial
       domain
16
17 %plot filtered images
18 figure()
19 subplot(1,2,1)
20 imshow(abs(img_filter_spat), [])
21 ylabel(name)
22 title('Filtered Image')
23 subplot(1,2,2)
24 imshow(real(img_filter_fft))
25 title('Frequency Domain')
26 xlabel(strcat(size,'*Size Radius Filter')
   )
```

Listing 6: Function that defines and applies filter and its inverse from Equation 1 and plots

```
1  function inverse_filter(img, name)
2
3  %% DOCUMENTATION
4
5  % FUNCTION ACCEPTS AN IMAGE AND A TITLE
       STRING
6  % CREATES A CUSTOM LOW PASS FILTER,
       FILTERS THE IMAGE, AND AFTER DERIVING
       THE INVERSE FILTER, TAKES THE INVERSE
       FILTER
7  % FUNCTION RETURNS AN IMAGE OF ALL THREE
       WITH TITLE
8
9  %% START OF CODE
10
11 h0 = 0.8.^(abs(-30:30));
12 k = 1/sum(h0);
13
14 h = k.*h0;
15
16 img_low = imfilter(imfilter(double(img),
       h, 'replicate'), h.', 'replicate');
17
18 H_inv = [-0.8, 1+(0.8)^2, -0.8]./(k - k
       *(0.8).^2);
19
20 img_low_25 = double(img_low) + sqrt(25).*
       randn(size(img_low));
21 img_low_100 = double(img_low) + sqrt(100)
       .*randn(size(img_low));
22
23 img_inv = imfilter(imfilter(double(
       img_low),H_inv, 'replicate'), H_inv.',
        'replicate');
24 img_inv_25 = imfilter(imfilter(double(
       img_low_25), H_inv, 'replicate'),
       H_inv.', 'replicate');
25 img_inv_100 = imfilter(imfilter(double(
       img_low_100), H_inv, 'replicate'),
       H_inv.', 'replicate');
26
27 figure()
28 subplot(1,3,1)
29 imshow(uint8(img))
30 title('Raw Image');
31 ylabel(name)
32 subplot(1,3,2)
33 imshow(uint8(img_low))
34 title('Custom Low Passed')
35 xlabel('31x31 Window')
36 subplot(1,3,3)
37 imshow(uint8(img_inv))
38 title('Passed through Inverse')
```

```
xlabel('31x31 Window')

figure()
subplot(1,2,1)
imshow(uint8(img_low_25))
title('Custom Low Passed with Noise')
ylabel(['Noisy ', name, ', \sigma^2 = 25'])
xlabel('Noise Added to Low Passed Image')
subplot(1,2,2)
imshow(uint8(img_inv_25))
title('Passed Through Inverse')

figure()
subplot(1,2,1)
imshow(uint8(img_low_100))
title('Custom Low Passed with Noise')
ylabel(['Noisy ', name, ', \sigma^2 = 100'])
xlabel('Noise Added to Low Passed Image')
subplot(1,2,2)
imshow(uint8(img_inv_100))
title('Passed Through Inverse')
```

REFERENCES

[1] J. Gonzales-Barajas and D. Montenegro, *Average Filtering: Theory, Design, and Implementation*, Universidad Santo Tomás, Bogotá, Columbia, 2016.

[2] G. Deng and L.W. Cahill, *An Adaptive Gaussian Filter for Noise Reduction and Edge Detection*, Department of Electronic Engineering, La Trobe University, Australia, 1994.

[3] MATLAB r2019b, *rand*, MATLAB Mathworks, 2020.

[4] MATLAB r2019b, *imnoise*, MATLAB Mathworks, 2020.