

TITLE: IMAGE FILTERING USING AVERAGING FILTER

1.0 INTRODUCTION

Image processing is any form of information processing for which the input is an image, such as photographs or frames of video and the output is not necessarily an image, but can be for instance a set of features of the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. Image filtering is a process by which we can enhance (or otherwise modify, warp, and mutilate) images and the procedure of reducing or attenuating the noise components of a measured signal is commonly known as filtering. Image filtering also allows us to apply various effects on images. There are many different ways to design filters, but the most common ones have their roots in simple averaging.

In this task, we have to experiment an image filtering process by using an averaging filtering. An image that was added with noise will be filtered by using averaging filter. Averaging is used to reduce the effects of noise, then smoothing the image. Averaging filter also can make an image dimmer or brighter. Every output will show and original image that was added by noise and the filtered image using averaging filter. So we can compare an original images and filtered images. Graphic User Interface was build with all needed parameters to show an averaging filtering process

2.0 OBJECTIVES

The filtering method can be build by using Matlab. In order to realize it, several objectives are stated as below:

- 1) To understand image filtering method by using averaging filter.
- 2) To apply Matlab software in order to filter noisy images.
- 3) To recognize suitable coding to build an averaging filter program.
- 4) To build an interface between user and the program that called Graphic User Interface (GUI).

3.0 SCOPE

For this task, we are required to apply an Image filtering by using an averaging filter. Averaging is an operation that takes an image as input, and produces a new image as output. There are two common types of smoothing methods: filtering (averaging) and local regression. Each smoothing method requires a span. The span defines a window of neighboring points to include in the smoothing calculation for each data point. We have to select an image that was added with noise and we are going to use the image processing which is the application of averaging filter. It consists of several processes for example from storing images in computer memory and adds with noise. The noisy images will be filtered using an averaging filter to see the effect. With such a small filter matrix, this gives only a very soft blur and with a bigger filter you can blur it a bit more. Hence, more smooth that we want, the bigger the filter has to be.

4.0 LITERATURE REVIEW

4.1 MATLAB

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include Math and computation Algorithm development Data acquisition Modeling, simulation, and prototyping Data analysis, exploration, and visualization Scientific and engineering graphics Application development, including graphical user interface building MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or Fortran. The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects.

Image Processing In Matlab

When working with images in Matlab, there are many things to keep in mind such as loading an image, using the right format, saving the data as different data types, how to display an image, conversion between different image formats, etc. there are some of the commands designed for these operations. Most of these commands require user to have the *Image processing tool box* installed with Matlab. To find out if it is installed, type `ver` at the Matlab prompt. This gives you a list of what tool boxes that are installed on your system. For further reference on image handling in Matlab you are recommended to use Matlab's help browser. There is an extensive (and quite good) on-line manual for the Image processing tool box that you can access via Matlab's help browser.

Fundamentals

A digital image is composed of *pixels* which can be thought of as small dots on the screen. A digital image is an instruction of how to color each pixel. A typical size of an image is 512-by-512 pixels. In the general case we say that an image is of size m -by- n if it is composed of m pixels in the vertical direction and n pixels in the horizontal direction. Let us say that we have an image on the format 512-by-1024 pixels. This means that the data for the image must contain information about 524288 pixels, which requires a lot of memory. Hence, *compressing* images is essential for efficient image processing. We can see how Fourier analysis and Wavelet analysis can help us to compress an image significantly. There are also a few "computer scientific" tricks (for example entropy coding) to reduce the amount of data required to store an image.

Image formats supported by Matlab

The following image formats are supported by Matlab:

- BMP
- HDF
- JPEG
- PCX
- TIFF
- XWB

Most images you find on the Internet are JPEG-images which is the name for one of the most widely used compression standards for images. If you have stored an image you can usually see from the suffix what format it is stored in. For example, an image named myimage.jpg is stored in the JPEG format and we will see later on that we can load an image of this format into Matlab.

Intensity image (gray scale image)

This is the equivalent to a "gray scale image" and this is the image we will mostly work with in this course. It represents an image as a matrix where every element has a value corresponding to how bright/dark the pixel at the corresponding position should be colored. There are two ways to represent the number that represents the brightness of the pixel: The double class (or data type). This assigns a floating number ("a number with decimals") between 0 and 1 to each pixel. The value 0 corresponds to black and the value 1 corresponds to white. The other class is called uint8 which assigns an integer between 0 and 255 to represent the brightness of a pixel. The value 0 corresponds to black and 255 to white. The class uint8 only requires roughly 1/8 of the storage compared to the class double. On the other hand, many mathematical functions can only be applied to the double class. We will see later how to convert between double and uint8.

Indexed image

This is a practical way of representing color images. (In this course we will mostly work with gray scale images but once you have learned how to work with a gray scale image you will also know the principle how to work with color images.) An indexed image stores an image as two matrices. The first matrix has the same size as the image and one number for each pixel. The second matrix is called the *color map* and its size may be different from the image. The numbers in the first matrix is an instruction of what number to use in the color map matrix.

RGB image

This is another format for color images. It represents an image with three matrices of sizes matching the image format. Each matrix corresponds to one of the colors red, green or blue and gives an instruction of how much of each of these colors a certain pixel should use.

IMAGE FILTERING

Image filtering is a process by which we can enhance (or otherwise modify, warp, and mutilate) images. I've seen enough posts asking about this that I've made this info file to answer most people's questions. This file has information (so far) only about "compare" filters, which compare a pixel somehow with the pixels around it to filter the image. Another term that may be important is that we're doing "weighted" filtering. (as opposed to "equal" filtering) That means that different pixels have different importance in calculating the image.

Using Averaging Filter

The Mean Filter or averaging filter can be used to remove noise from an image. It is a filter that takes the average of the current pixel and its neighbors.

This is an ordinary blur filter. We can test it on the following image with "Salt and Pepper" Noise:



(1a)



(1b)

Figure:(1a) Image with "Salt and Pepper" Noise (1b)When applied, it gives a blurry result

Using Linear Filtering

You can use linear filtering to remove certain types of noise. Certain filters, such as averaging or Gaussian filters, are appropriate for this purpose. For example, an averaging filter is useful for removing grain noise from a photograph. Because each pixel gets set to the average of the pixels in its neighborhood, local variations caused by grain are reduced.

Using Median Filtering

Median filtering is similar to using an averaging filter, in that each output pixel is set to an average of the pixel values in the neighborhood of the corresponding input pixel. However, with median filtering, the value of an output pixel is determined by the median of the neighborhood pixels, rather than the mean. The median is much less sensitive than the mean to extreme values (called outliers). Median filtering is therefore better able to remove these outliers without reducing the sharpness of the image. The `medfilt2` function implements median filtering.

The `medfilt2` function implements median filtering. Note Median filtering is a specific case of order-statistic filtering, also known as rank filtering. For information about order-statistic filtering, see the reference page for the `ordfilt2` function. The following example compares using an averaging filter and `medfilt2` to remove salt and pepper noise. This type of noise consists of random pixels' being set to black or white (the extremes of the data range). In both cases the size of the neighborhood used for filtering is 3-by-3. Read in the image and display it. `I = imread('eight.tif');`

Using Adaptive Filtering

The `wiener2` function applies a Wiener filter (a type of linear filter) to an image adaptively, tailoring itself to the local image variance. Where the variance is large, `wiener2` performs little smoothing. Where the variance is small, `wiener2` performs more smoothing. This approach often produces better results than linear filtering. The adaptive filter is more selective than a comparable linear filter, preserving edges and other high-frequency parts of an image. In addition, there are no design tasks; th

handles all preliminary computations and implements the filter for an input image. `wiener2`, however, does require more computation time than linear filtering. `wiener2` works best when the noise is constant-power ("white") additive noise, such as Gaussian noise. The example below applies `wiener2` to an image of Saturn that has had Gaussian noise added. For an interactive demonstration of filtering to remove noise, try running `nrfiltdemo`.

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces (GUIs) on your MATLAB applications. The MATLAB Application Program Interface (API). This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

4.2 GRAPHICAL USER INTERFACE (GUIs)

A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks. To perform these tasks, the user of the GUI does not have to create a script or type commands at the command line. Often, the user does not have to know the details of the task at hand. The GUI components can be menus, toolbars, push buttons, radio buttons, list boxes, and sliders. In MATLAB, a GUI can also display data in tabular form or as plots, and can group related components. GUI is also a type of user interface which allows people to interact with a computer and computer-controlled devices which employ graphical icons, visual indicators or special graphical elements called "widgets", along with text, labels or text navigation to represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements. Use of this acronym led to creation of the neologism *guituitive* (an interface which is intuitive).

GUIDE, the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These tools simplify the process of laying out and programming GUIs.

GUI Layout

Using the GUIDE Layout Editor, you can populate a GUI by clicking and dragging GUI components such as axes, panels, buttons, text fields, sliders, and so on into the layout area. You can also create menus and context menus for the GUI. From the Layout Editor, you can size the GUI, modify component look and feel, align components, set tab order, view a hierarchical list of the component objects, and set GUI options.

GUI Programming

GUIDE automatically generates an M-file that controls how the GUI operates. This M-file provides code to initialize the GUI and contains a framework for the GUI callbacks the routines that execute when a user interacts with a GUI component. Using the M-file editor, you can add code to the callbacks to perform the functions you want.

To start GUIDE, enter `guide` at the MATLAB prompt. This displays the GUIDE Quick Start dialog, as shown in the following figure.

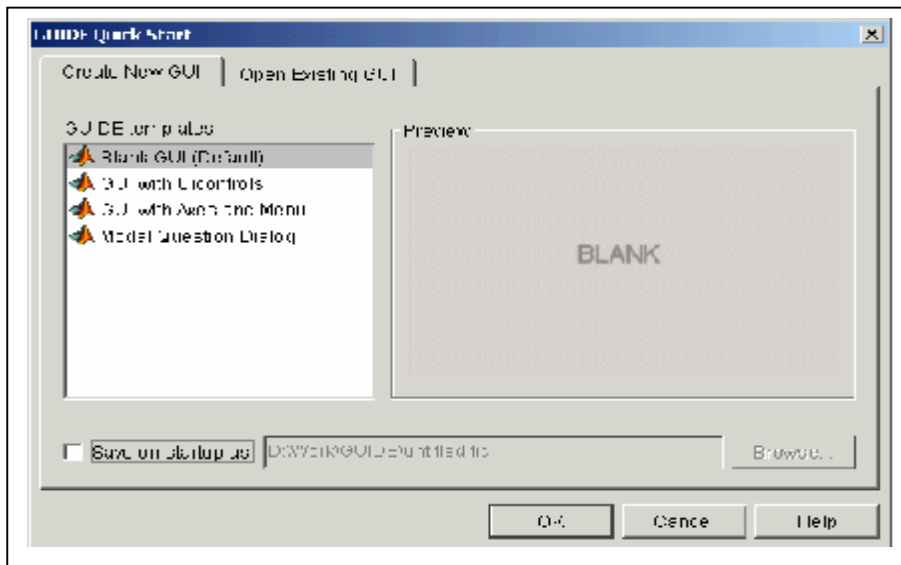


Figure 2: Start dialog box

When you open a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools with a blank GUI template. User can lay out the GUI by dragging components, such as push buttons, pop-up menus, or axes, from the component palette, at the left side of the Layout Editor, into the layout area as we need. It is appears as in the following figure. The GUIDE Quick Start dialog provides templates for several basic types of GUIs. The advantage of using templates is that often you can modify a template more quickly and easily than by starting from a blank GUI. When user select a template in the Templates pane, a preview of it appears in the right-hand pane. To display the names of the GUI components in the component palette, select the preferences from the File menu and check the box next to show names in component palette.

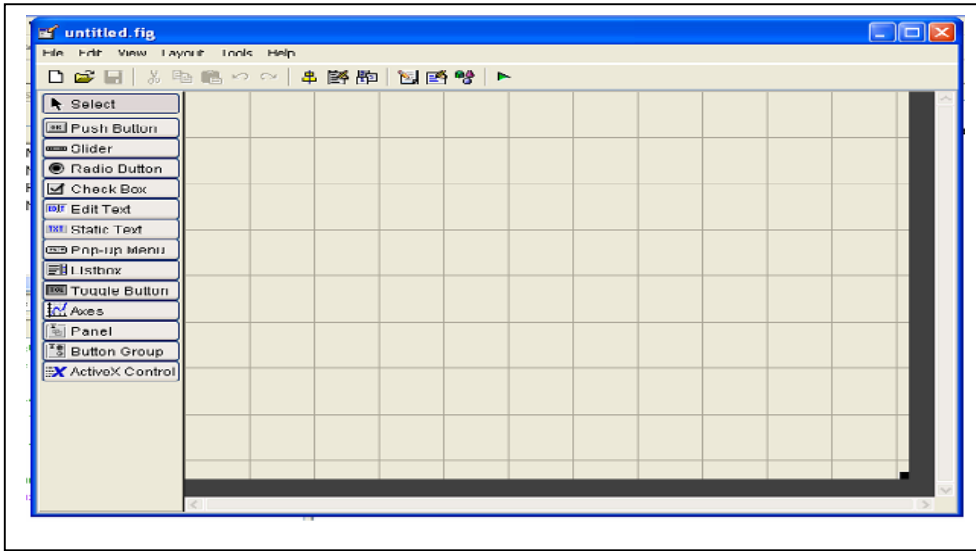


Figure 3:Blank GUI

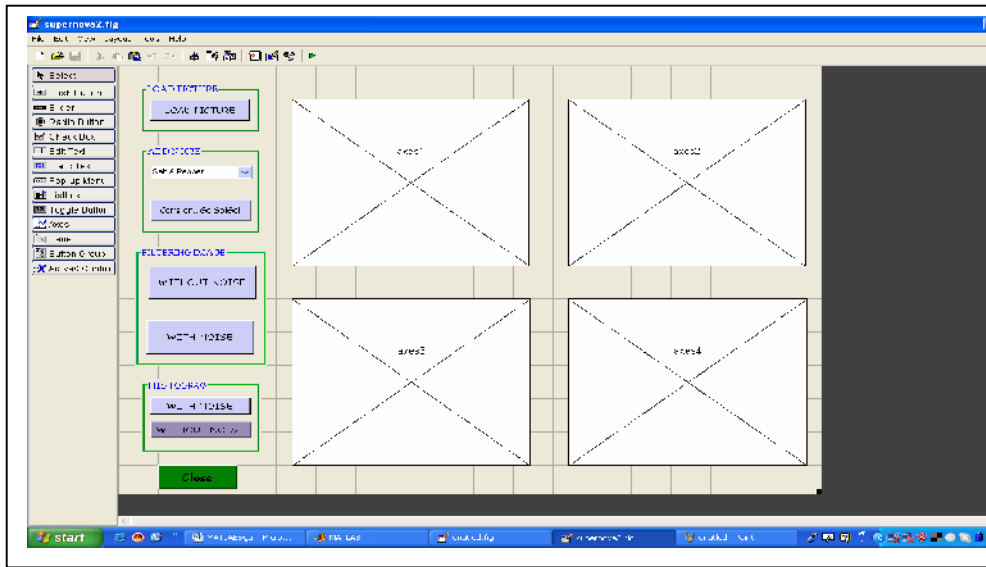


Figure 4: GUI with preference template.

For this task, the design is about to filter a picture which add by noise or without noise. There are seven pushbutton represent different functions which are for loading picture, adding noise, filtering image and histogram.

Loading picture

User can loading picture from any pathname or location by clicking at LOAD PICTURE and it will displayed in GUI .

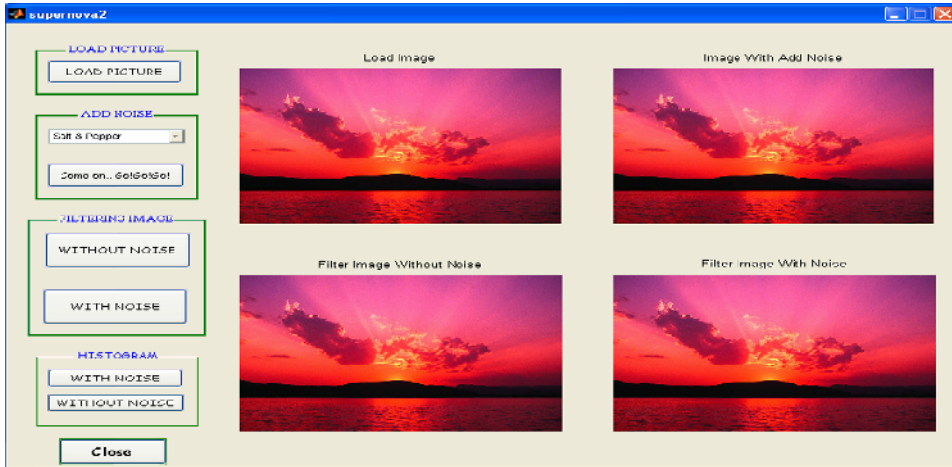


Figure 5: Loading picture

Add noise

There are four types of noise that user can choose which are Salt & peper, poisson, Gaussian and speckle and can add to original image by clicking *Come on..Go!Go!Go!.*

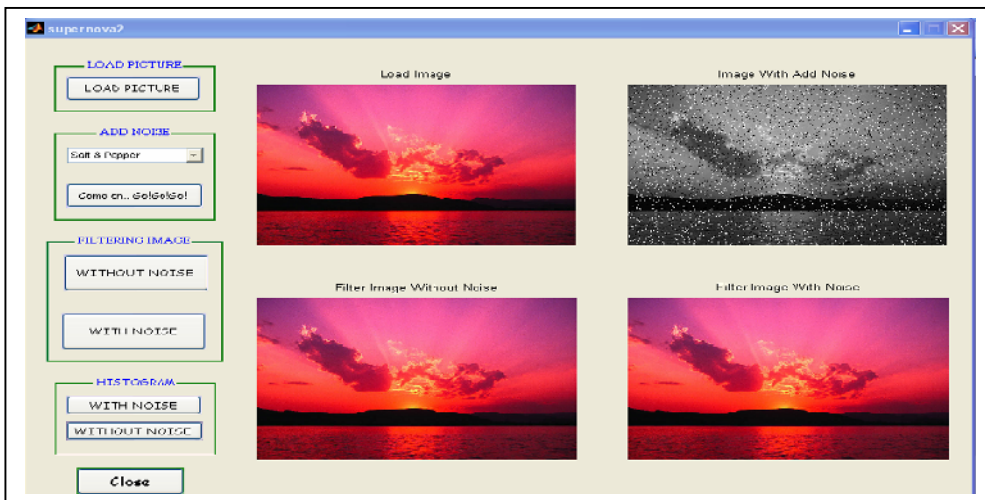


Figure 6: Image add with noise (Salt & pepper)

Filtering image

User can prefer either to filter an image without noise or an image add with noise by clicking button *WITHOUT NOISE* or *WITH NOISE* in Filtering Image menu.

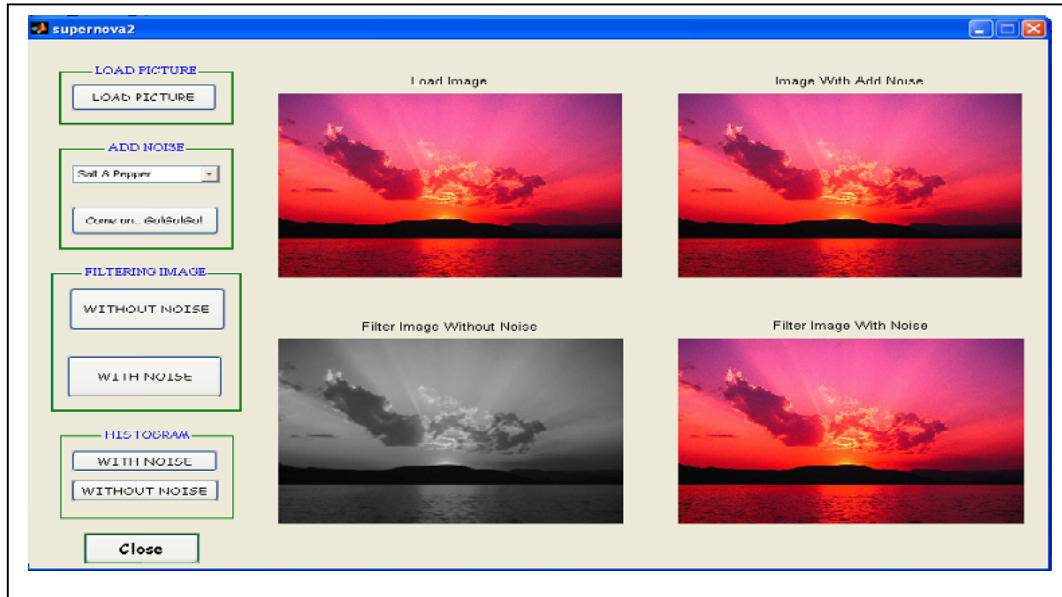


Figure 7: Filtering image without noise

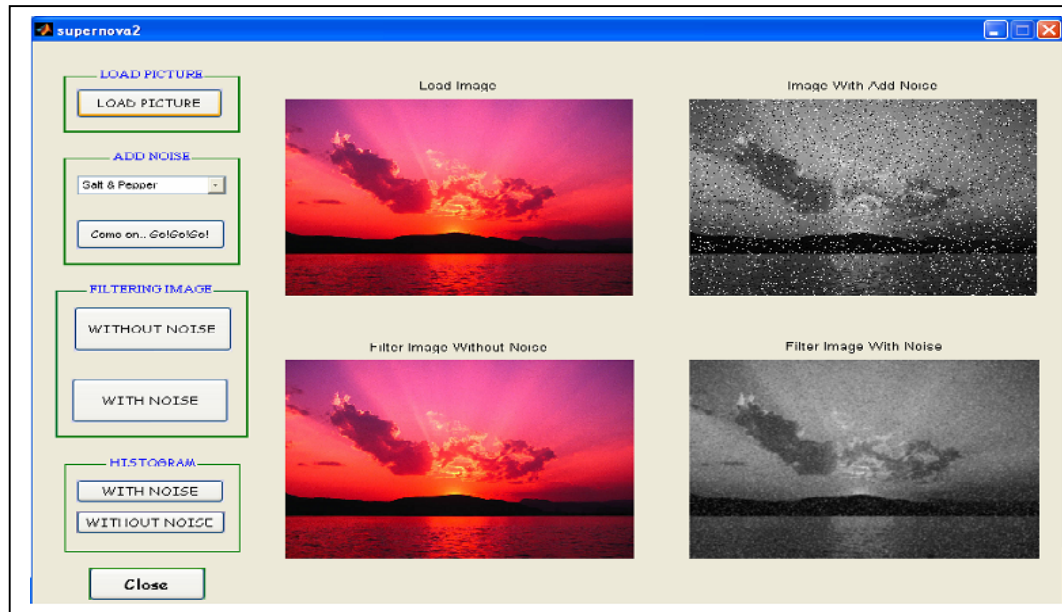


Figure 8: Filtering image with noise

Histogram

To view image's histogram, user can click the button on Histogram menu and the program will show the histogram of image either for image WITHOUT NOISE or WITH NOISE.

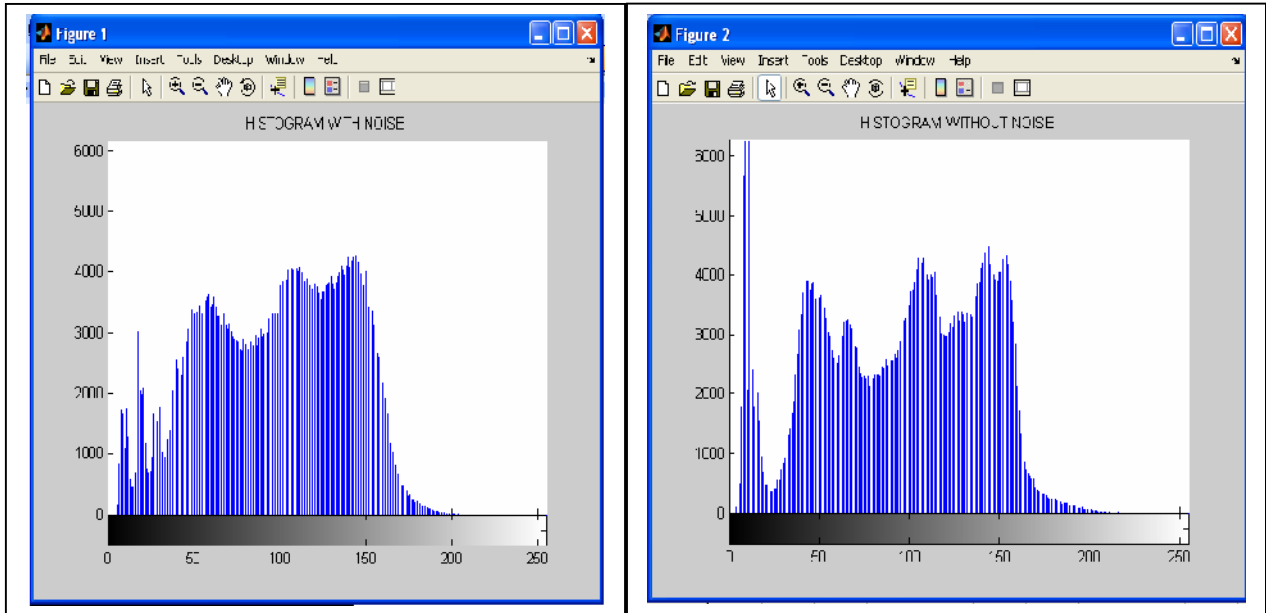


Figure 9: Image's histogram

Procedure can be repeated to view the effect of other noises and user can see the result to an image before and after using averaging filter.

4.3 AVERAGING FILTER

There are two common types of smoothing methods: filtering (averaging) and local regression. Each smoothing method requires a span. The span defines a window of neighboring points to include in the smoothing calculation for each data point. This window moves across the data set as the smoothed response value is calculated for each predictor value. A large span increases the smoothness but decreases the resolution of the smoothed data set, while a small span decreases the smoothness but increases the resolution of the smoothed data set. The optimal span value depends on your data set and the smoothing method, and usually requires some experimentation to find. The Curve

Fitting Toolbox supports these smoothing methods: Moving average filtering -- Lowpass filter that takes the average of neighboring data points.

A moving average filter smoothes data by replacing each data point with the average of the neighboring data points defined within the span. This process is equivalent to lowpass filtering with the response of the smoothing given by the difference equation

$$y_s(i) = \frac{1}{2N+1} (y(i+N) + y(i+N-1) + \dots + y(i-N))$$

Where $y_s(i)$ is the smoothed value for the i th data point, N is the number of neighboring data points on either side of $y_s(i)$, and $2N+1$ is the span. The moving average smoothing method used by the Curve Fitting Toolbox follows these rules:

- The span must be odd.
- The data point to be smoothed must be at the center of the span.
- The span is adjusted for data points that cannot accommodate the specified number of neighbors on either side.
- The end points are not smoothed because a span cannot be defined.

Note that you can use MATLAB's `filter` function to implement difference equations such as the one shown above. However, because of the way that the end points are treated, the toolbox moving average result will differ from the result returned by `filter`. Refer to *Difference Equations and Filtering* in the MATLAB documentation for more information. For example, suppose you smooth data using a moving average filter with a span of 5. Using the rules described above, the first four elements of y_s are given by

$$\begin{aligned}y_s(1) &= y(1) \\y_s(2) &= (y(1)+y(2)+y(3))/3 \\y_s(3) &= (y(1)+y(2)+y(3)+y(4)+y(5))/5 \\y_s(4) &= (y(2)+y(3)+y(4)+y(5)+y(6))/5\end{aligned}$$

Note that $ys(1), ys(2), \dots, ys(end)$ refer to the order of the data after sorting, and not necessarily the original order. The smoothed values and spans for the first four data points of a generated data set are shown below.

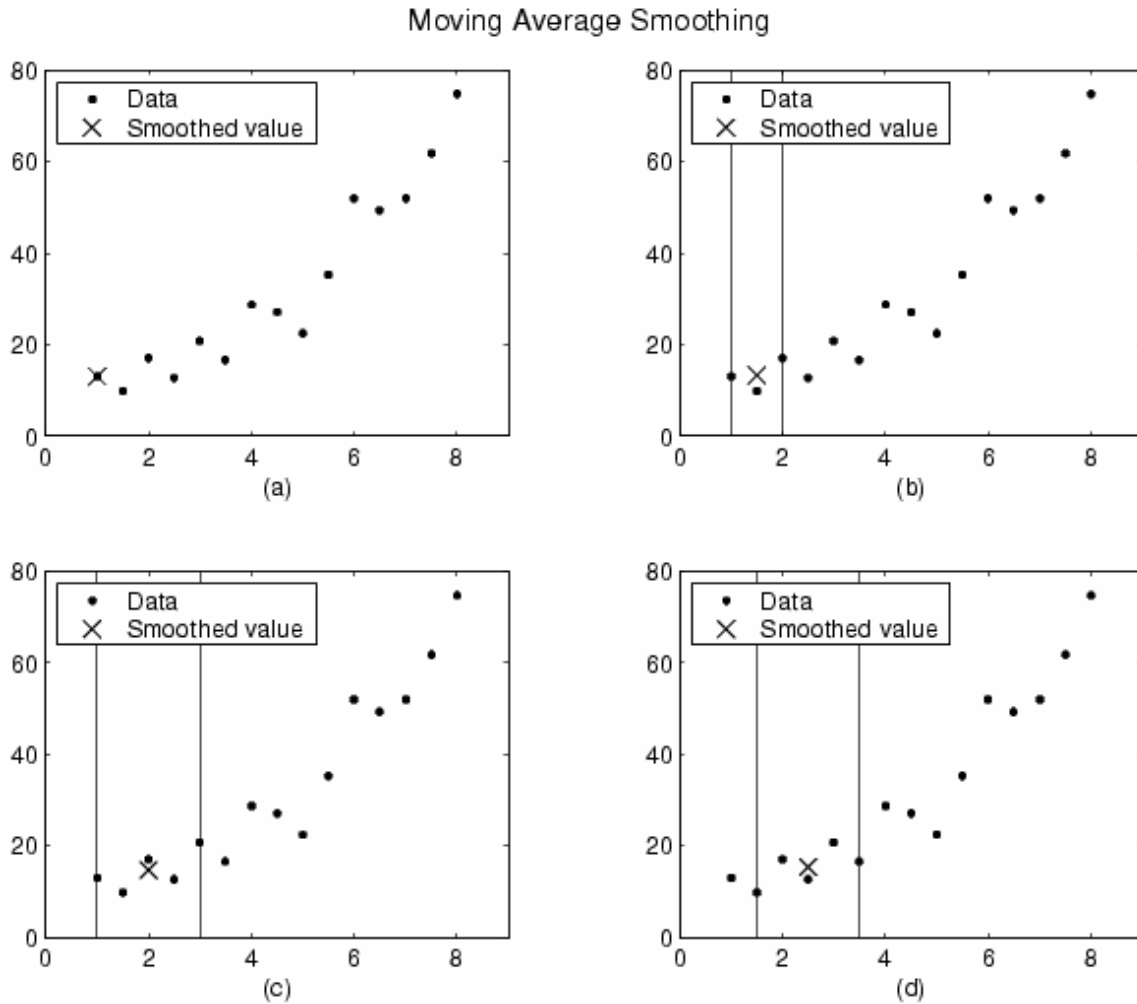


Figure10: Moving smoothing Averaging

Plot (a) indicates that the first data point is not smoothed because a span cannot be constructed. Plot (b) indicates that the second data point is smoothed using a span of three. Plots (c) and (d) indicate that a span of five is used to calculate the smoothed value. Smoothing Data Lowness and Loess: Local Regression Smoothing

Filtering of images, either by correlation or convolution can be performed using the toolbox function `imfilter`. This example filters the image in the file `blood1.tif` with a


```

5-by-5 filter containing equal weights. Such a filter is often called an averaging filter. I =
imread('blood1.tif');
h = ones(5,5) / 25;
I2 = imfilter(I,h);
imshow(I), title('Original image')
figure, imshow(I2), title('Filtered image')

```

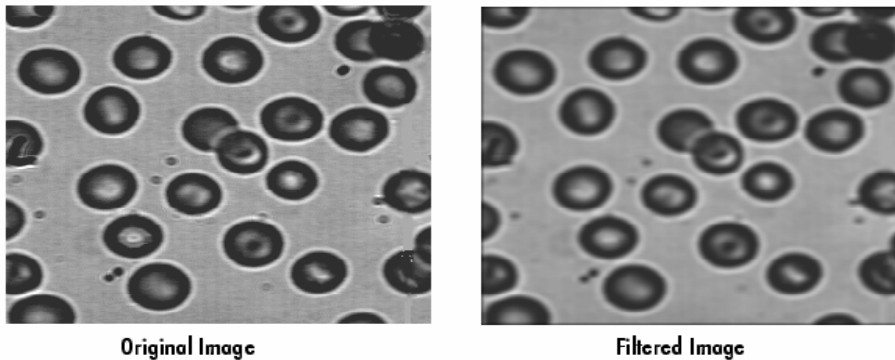


Figure 11: Original & filtered image

Data Types The `imfilter` function handles data types similar to the way the image arithmetic functions do, as described in Image Arithmetic Truncation Rules. The output image has the same data type, or numeric class, as the input image. The `imfilter` function computes the value of each output pixel using double-precision, floating-point arithmetic. If the result exceeds the range of the data type, the `imfilter` function truncates the result to that data type's allowed range. If it is an integer data type, `imfilter` rounds fractional values. Because of the truncation behavior, you may sometimes want to consider converting your image to a different data type before calling `imfilter`. In this example, the output of `imfilter` has negative values when the input is of class `double`

```
A = magic(5)
```

```
A =
```

```

17 24  1  8 15
23  5  7 14 16
 4  6 13 20 22

```

```
10 12 19 21 3
11 18 25 2 9
```

```
h = [-1 0 1]
```

```
h =
```

```
-1 0 1
```

```
imfilter(A,h)
```

```
ans =
```

```
24 -16 -16 14 -8
5 -16 9 9 -14
6 9 14 9 -20
12 9 9 -16 -21
18 14 -16 -16 -2
```

Notice that the result has negative values. Now suppose A was of class uint8, instead of double. `A = uint8(magic(5));`

```
imfilter(A,h)
```

```
ans =
```

```
24 0 0 14 0
5 0 9 9 0
6 9 14 9 0
12 9 9 0 0
18 14 0 0 0
```

Since the input to `imfilter` is of class `uint8`, the output also is of class `uint8`, and so the negative values are truncated to 0. In such cases, it may be appropriate to convert the image to another type, such as a signed integer type, `single`, or `double`, before calling `imfilter`

Correlation and Convolution Options. The `imfilter` function can perform filtering using either correlation or convolution. It uses correlation by default, because the filter design functions, described in Filter Design, and the `fspecial` function, described in Using Predefined Filter Types, produce correlation kernels. However, if you want to perform filtering using convolution instead, you can pass the string 'conv' as optional input argument to `imfilter`. For example, `A = magic(5);`

```
h = [-1 0 1]
```

```
imfilter(A,h) % filter using correlation
```

```
ans =
```

```
24 -16 -16 14 -8
 5 -16  9  9 -14
 6  9 14  9 -20
12  9  9 -16 -21
18 14 -16 -16 -2
```

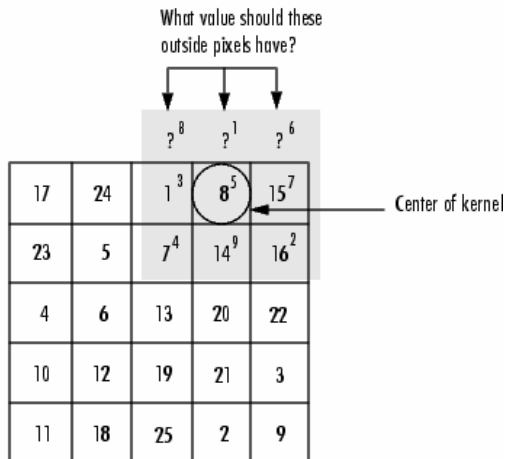
```
imfilter(A,h,'conv') % filter using convolution
```

```
ans =
```

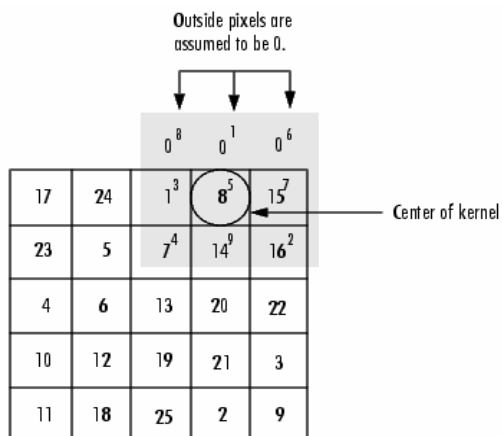
```
-24 16 16 -14  8
-5 16 -9 -9 14
-6 -9 -14 -9 20
-12 -9 -9 16 21
-18 -14 16 16  2
```

Boundary Padding Options

When computing an output pixel at the boundary of an image, a portion of the convolution or correlation kernel is usually off the edge of the image, as illustrated in the figure below.



The imfilter function normally fills in these "off-the-edge" image pixels by assuming that they are 0. This is called zero-padding and is illustrated in the figure below.



When filtering an image, zero-padding can result in a dark band around the edge of the image, as shown in this example. `I = imread('blood1.tif');`

`h = ones(5,5)/25;`

`I2 = imfilter(I,h);`

`imshow(I), title('Original image')`

`figure, imshow(I2), title('Filtered image')`

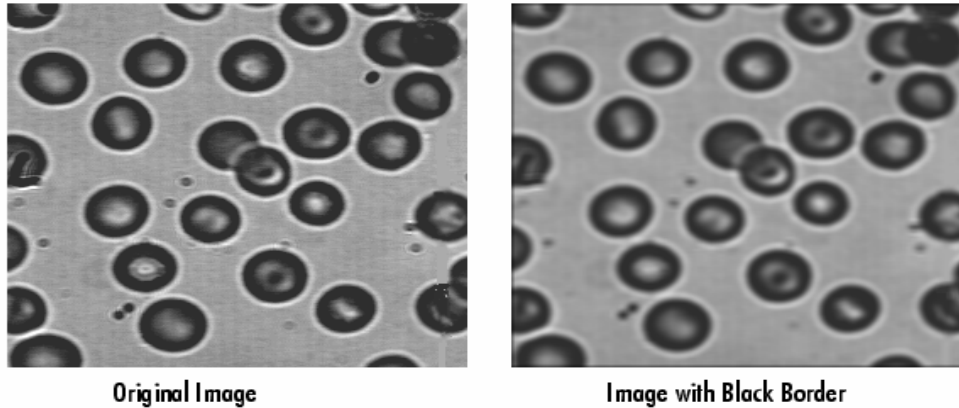
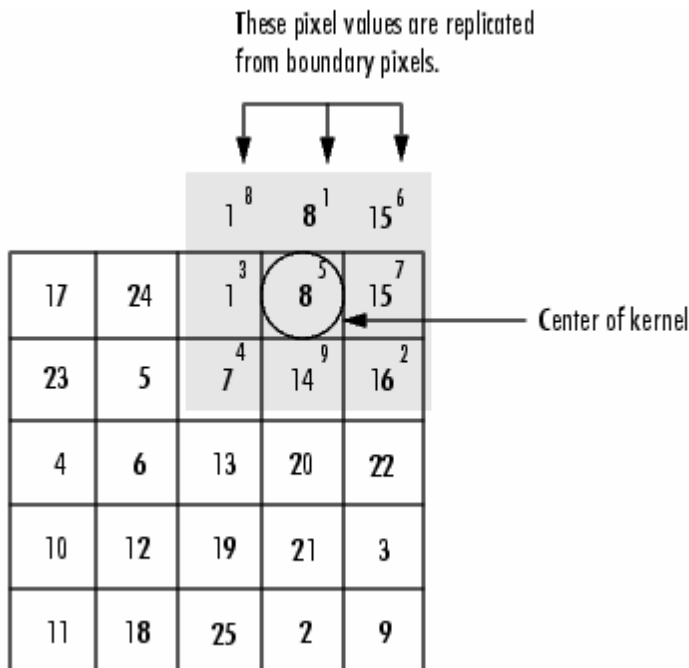
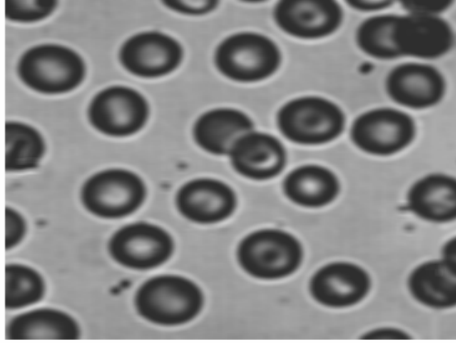


Figure 12: Original image & Image with black border

To eliminate the zero-padding artifacts around the edge of the image, `imfilter` offers an alternative boundary padding method called border replication. In border replication, the value of any pixel outside the image is determined by replicating the value from the nearest border pixel. This is illustrated in the figure below.



To filter using border replication, pass the additional optional argument 'replicate' to `imfilter`. `I3 = imfilter(I,h,'replicate');`
`figure, imshow(I3), title('Filtered with border replication')`



1. **Image Border with Replication**

Figure 13 : Image border with replication

Multidimensional Filtering

The `imfilter` function can handle both multidimensional images and multidimensional filters. A convenient property of filtering is that filtering a three-dimensional image with a two-dimensional filter is equivalent to filtering each plane of the three-dimensional image individually with the same two-dimensional filter. This property makes it easy, for example, to filter each color plane of a truecolor image with the same filter. `rgb = imread('flowers.tif');`

```
h = ones(5,5) / 25;
```

```
rgb2 = imfilter(rgb,h);
```

```
imshow(rgb), title('Original image')
```

```
figure, imshow(rgb2), title('Filtered image')
```



Figure 14: Original & Filtered Image

5.0 METHODOLOGY

Before starting on this assignment, there were discussion that been made by the group members on how the flow of our work must be done. First of all the group members must done some research on averaging filters. After the researches have been done, all the information will be collected and studied. Then after that the work of designing the filters MATLAB program will be executed. After the programming have finished the designing of the Graphic User Interface (GUI) will be done. Then from this point onward the programs will undergo some trial and error to find whether the programs are right and the result is as expected. If the programs occurs error, so the designing will be done again until the desired result is accomplish. Then after the trial and error has been done, the analysis of our work will be done and then the accomplished result will be submitted.

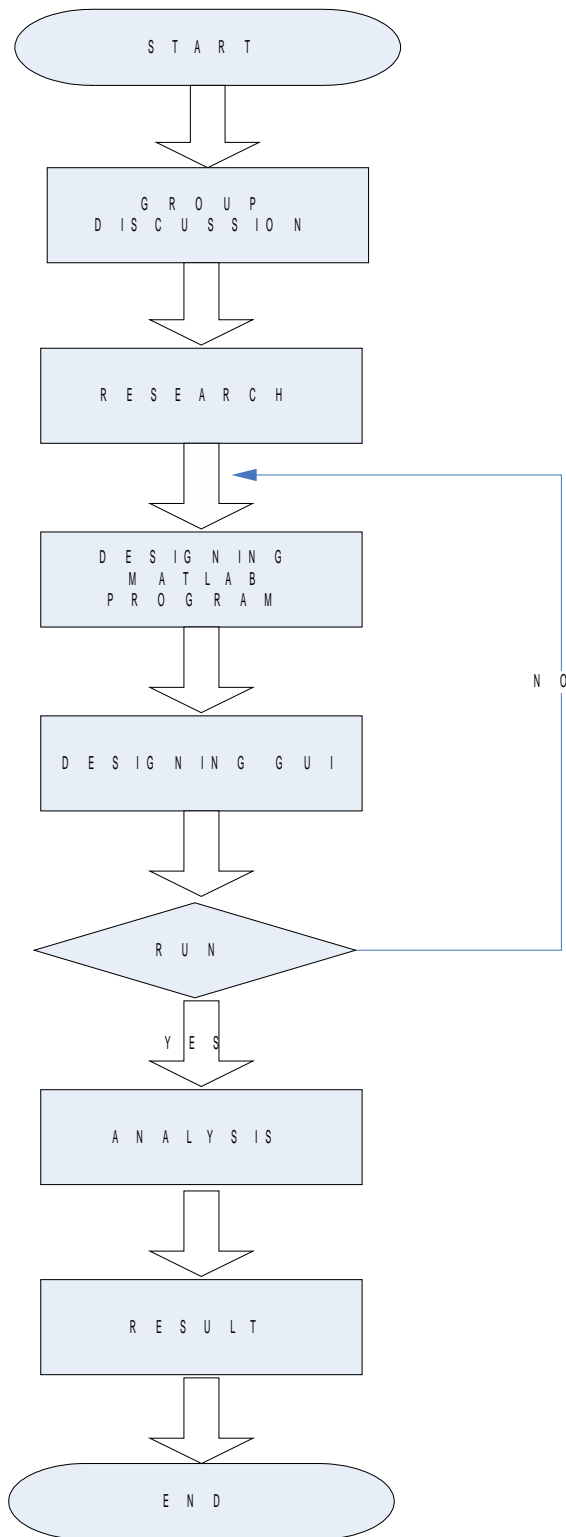
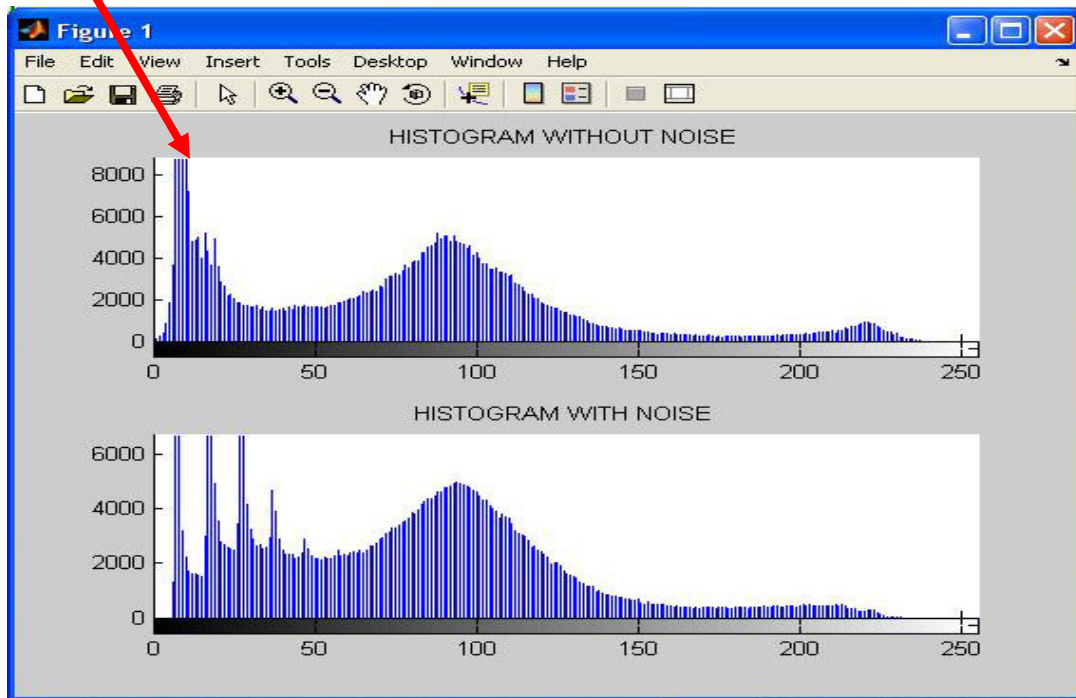
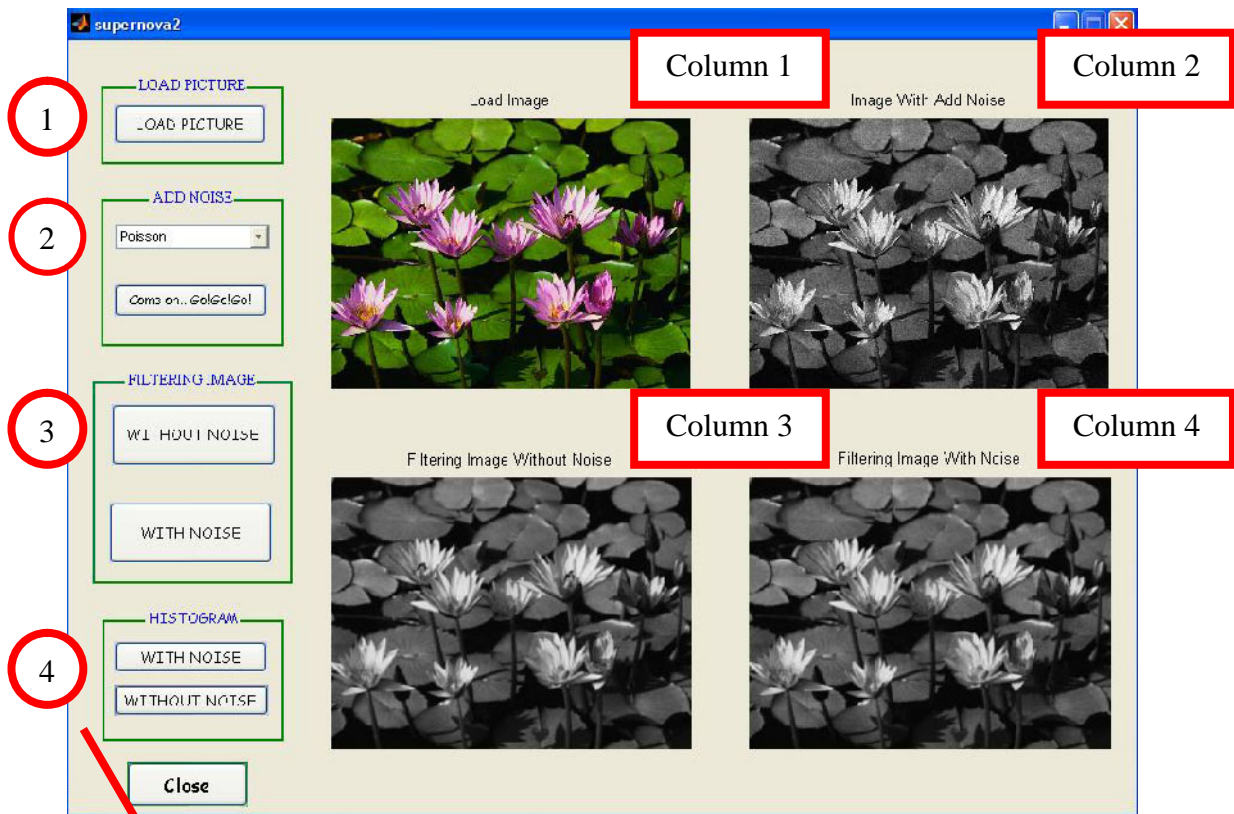


Chart 1: The Procedure Flow Chart

BASIC INSTRUCTION ON THE GUI



[1] Picture is being uploaded into the GUI interface. There are 4 columns of the GUI and each column represents the process that will take place in the GUI. The first will display the original image.

[2] Then the ADD NOISE command will be initiated, the image in the second column will display the image that has been generated with noise. There are 4 types of noise that can select in the command there are:

1. Gaussian noise
2. Salt and Pepper noise
3. Speckle noise
4. Poisson noise

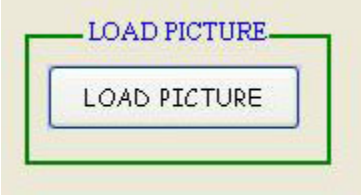

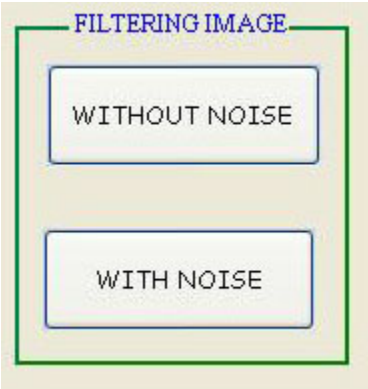
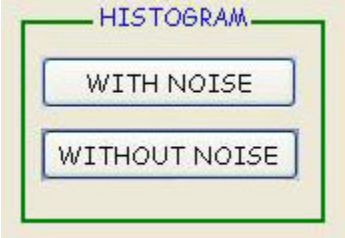
Each noise have their own characteristic.

[3] Then the image will be filtered using average filter command (FILTERING IMAGE). The third column will generate the without noise command meaning that the image is filtered without the presence of the noise and the forth column will generate the with noise command meaning that the image is filtered with the presence of noise.

[4] Then the histogram command is use to displayed the histogram graph of the image. There 2 type of histogram that is:

1. The histogram of the image without the presence of noise.
2. The histogram of the image with the presence of noise.

THE GUI PUSH BUTTON COMMANDS AND INSTRUCTIONS.

Command	Instructions
	<p>The LOAD PICTURE command is use to load image from the picture file in the computer.</p>
	<p>The ADD NOISE command is use to insert noise to image that have been load into the GUI. There are 4 types of noise that can be generate onto the image:</p> <ul style="list-style-type: none"> [1] Poisson [2] Salt and Pepper [3] Speckle [4] Gaussian
	<p>The FILTERING IMAGE command is use to filter the image that have been uploaded into the GUI. There 2 command in the push buttons that is:</p> <ul style="list-style-type: none"> [1] WITHOUT NOISE use to filter the image without the present of the noise [2] WITH NOISE use to filter the image with the present of the noise
	<p>The HISTOGRAM is use to show the histogram graph of the filtered image to see how the filter effects on the image. There are 2 types of histogram graph that been displayed:</p> <ul style="list-style-type: none"> [1] with noise histogram image [2] without noise histogram image

6.0 GANTT CHART

ID	Task Name	Start	Finish	Duration	Aug 2007					Sep 2007					Oct 2007				
					29/7	5/8	12/8	19/8	26/8	2/9	9/9	16/9	23/9	30/9	7/10	14/10	21/10		
1	First Meeting	7/25/2007	7/25/2007	1d															
2	Information searching	7/25/2007	8/6/2007	9d	■■■■■														
3	Second Meeting	8/6/2007	8/6/2007	1d															
4	Creating soft code using MATLAB GUI	8/6/2007	8/27/2007	16d	■■■■■■■■■■														
5	Third meeting	8/27/2007	8/27/2007	1d															
6	Preparation for proposal	8/27/2007	9/6/2007	9d	■■■■■														
7	Submit proposal	9/6/2007	9/6/2007	1d															
8	Fifth Meeting	9/10/2007	9/10/2007	1d															
9	Build Image Filtering use MATLAB GUI	9/10/2007	9/28/2007	15d	■■■■■■■■■■■														
10	Sixth Meeting	10/1/2007	10/1/2007	1d															
11	Preparation for report	10/1/2007	10/19/2007	15d	■■■■■■■■■■■														
12	Submit Final Report	10/22/2007	10/22/2007	1d															
13	Preparation for presentation	10/22/2007	10/25/2007	4d	■■■■														
14	Presentation	10/26/2007	10/26/2007	1d															

7.0 SOFTCODE – AVERAGING FILTER

```
function varargout = LastStand(varargin)
% LASTSTAND M-file for LastStand.fig
%   LASTSTAND, by itself, creates a new LASTSTAND or raises the existing
%   singleton*.
%
%   H = LASTSTAND returns the handle to a new LASTSTAND or the handle to
%   the existing singleton*.
%
%   LASTSTAND('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in LASTSTAND.M with the given input
arguments.
%
%   LASTSTAND('Property','Value',...) creates a new LASTSTAND or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before LastStand_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to LastStand_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help LastStand

% Last Modified by GUIDE v2.5 20-Oct-2007 22:18:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @LastStand_OpeningFcn, ...
                  'gui_OutputFcn', @LastStand_OutputFcn, ...
                  'gui_LayoutFcn', [], ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before LastStand is made visible.
function LastStand_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LastStand (see VARARGIN)

% Choose default command line output for LastStand
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes LastStand wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = LastStand_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% *****
% *****
% ***** Push button4 untuk LOAD FILE *****
% *****
% *****

```

```

function pushbutton4_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

global grayfidz;
global colorfidz;

hehe = pwd;
% PWD displays the current working directory.
% PWD returns the current directory in the string buffer

[file, pathname] = uigetfile('*.jpg','Select An Image File');

cd(pathname); % CD sets the current working directory to chosen PATHNAME.

colorfidz = imread(file); % imread read image from chosen FILE

cd(hehe); % CD sets the current working directory to wacha.

subplot(handles.axes1);
imshow(colorfidz);
title('Load Image');

subplot(handles.axes2);
imshow(colorfidz);
title('Image With Add Noise');

subplot(handles.axes3);
imshow(colorfidz);
title('Filtering Image Without Noise');

subplot(handles.axes4);
imshow(colorfidz);
title('Filtering Image With Noise');

% imshow show image from colorfidz

grayfidz = rgb2gray(colorfidz); % show gray image for colorfidz

save colorfidz;
% save file in M-File format.
% The data may be retrieved with LOAD

```

```

save grayfidz;
% save file in M-File format.
% The data may be retrieved with LOAD

% *****
% *****
% ***** Push button4 untuk LOAD FILE
% *****
% *****
% *****
% *****
% ***** popupmenu1 untuk ADDNOISE
% *****
% *****

function popupmenu1_Callback(hObject, eventdata, handles)
% hObject handle to popupmenu1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
% contents{get(hObject,'Value')} returns selected item from popupmenu1

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject handle to popupmenu1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
end

```



```

% *****
% *****
% ***** popupmenu1 untuk ADDNOISE
% *****
% *****
% *****
% *****
% ***** pushbutton9 untuk EXECUTE ADDNOISE
% *****
% *****

function pushbutton9_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton9 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

load colorfidz;
load grayfidz;

A = colorfidz;
B = grayfidz;

d = imnoise(B,'salt & pepper',0.09);
e = imnoise(B,'poisson');
f = imnoise(B,'gaussian',0.0,0.1);
g = imnoise(B,'speckle',0.04);

axes(handles.axes2);
cla;
% cla reset deletes from the current axes all graphics objects regardless
% of the setting of their HandleVisibility property and resets all
% axes properties, except Position and Units, to their default values.

dari = get(handles.popupmenu1, 'Value');

switch dari
    case 1
        imshow(d);
        title('Image With Add Noise');
    case 2
        imshow(e);
        title('Image With Add Noise');

```

```

case 3
    imshow(f);
    title('Image With Add Noise');
case 4
    imshow(g);
    title('Image With Add Noise');

end

save colorfidz;
% save file in M-File format.
% The data may be retrieved with LOAD

save grayfidz;
% save file in M-File format.
% The data may be retrieved with LOAD

% *****
% *****
% ***** pushbutton9 untuk EXECUTE ADDNOISE
% *****
% *****
% *****
% *****
% ***** pushbutton10 utk FILTER WITHOUT NOISE
% *****
% *****

function pushbutton10_Callback(hObject, eventdata, handles)
% hObject   handle to pushbutton10 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

load grayfidz;

B = grayfidz;

axes(handles.axes3);
filt = fspecial('average', [5, 5]);
L = imfilter(B, filt, 'symmetric');
imshow(L);
title('Filtering Image Without Noise');

```

```

% fspecial Create 2-D special filters
% fspecial(type) creates a two-dimensional filter h of the specified type.
% fspecial returns h as a correlation kernel,
% which is the appropriate form to use with imfilter.

% *****
% *****
% ***** pushbutton10 utk FILTER WITHOUT NOISE
% *****
% *****
% *****
% *****
% ***** pushbutton10 utk FILTER WITH NOISE
% *****
% *****

function pushbutton7_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton7 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

load grayfidz;

d = imnoise(B,'salt & pepper',0.09);
e = imnoise(B,'poisson');
f = imnoise(B,'gaussian',0.0,0.1);
g = imnoise(B,'speckle',0.04);

axes(handles.axes4);
cla;

filt = fspecial('average', [5, 5]);
h = imfilter(d, filt, 'symmetric');
i = imfilter(e, filt, 'symmetric');
j = imfilter(f, filt, 'symmetric');
k = imfilter(g, filt, 'symmetric');

% symmetric is Input array values outside the bounds of the array
% are computed by mirror-reflecting the array across the array border.

daripada = get(handles.popupmenu1, 'Value');
switch daripada

```

```

% ***** Push button8 untuk HISTOGRAM XDAK noise
% *****
% *****
% *****

% ***** Push button8 untuk HISTOGRAM ada noise
% *****
% *****

function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load grayfidz;

d = imnoise(B,'salt & pepper',0.09);
e = imnoise(B,'poisson');
f = imnoise(B,'gaussian',0.0,0.1);
g = imnoise(B,'speckle',0.04);

figure(1)
subplot(2,1,2)
cla;

filt = fspecial('average', [5, 5]);
h = imfilter(d, filt, 'symmetric');
i = imfilter(e, filt, 'symmetric');
j = imfilter(f, filt, 'symmetric');
k = imfilter(g, filt, 'symmetric');

from = get(handles.popupmenu1, 'Value');

switch from
    case 1
        imhist(h);
        title('HISTOGRAM WITH NOISE')
    case 2
        imhist(i);
        title('HISTOGRAM WITH NOISE')

```

```

case 3
    imhist(j);
    title('HISTOGRAM WITH NOISE')
case 4
    imhist(k);
    title('HISTOGRAM WITH NOISE')
end

% *****
% *****
% ***** Push button8 untuk HISTOGRAM ada noise
% *****
% *****
% *****
% *****
% ***** pushbutton6 utk CLOSE
% *****
% *****
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close;

% *****
% *****
% ***** pushbutton6 utk CLOSE
% *****
% *****
% *****

```

8.0 DATA ANALYSIS AND RESULTS

EXPLANATION AND FUNTION OF COMMAND

1) IMNOISE

Add noise to an image

Syntax

`J = imnoise(I,type)`

`J = imnoise(I,type,parameters)`

Description

`J = imnoise(I,type)` adds noise of a given type to the intensity image I. type is a string that can have one of these values.

Value and Description

a)'gaussian' for Gaussian white noise.

- `J = imnoise(I,'gaussian',m,v)` adds Gaussian white noise of mean m and variance v to the image I. The default is zero mean noise with 0.01 variance.

b)'localvar' for Zero-mean Gaussian white noise with an intensity-dependent variance.

- `J = imnoise(I,'localvar',V)` adds zero-mean, Gaussian white noise of local variance V to the image I. V is an array of the same size as I. `J = imnoise(I,'localvar',image_intensity,var)` adds zero-mean, Gaussian noise to an image I, where the local variance of the noise, var, is a function of the image intensity values in I. The image_intensity and var arguments are vectors of the same size, and `plot(image_intensity,var)` plots the functional relationship between noise variance and image intensity. The image_intensity vector must contain normalized intensity values ranging from 0 to 1.

c)'poisson' for Poisson noise.

- `J = imnoise(I,'poisson')` generates Poisson noise from the data instead of adding artificial noise to the data. In order to respect Poisson statistics, the intensities of uint8 and uint16 images must correspond to the number of photons (or any other quanta of information). Double-precision images are used when the number of photons per pixel can be much larger than 65535 (but less than 10^{12}); the intensity values vary between 0 and 1 and correspond to the number of photons divided by 10^{12} .

d)'salt & pepper' for On and off pixels.

- $J = \text{imnoise}(I, \text{'salt \& pepper'}, d)$ adds salt and pepper noise to the image I , where d is the noise density. This affects approximately $d \cdot \text{prod}(\text{size}(I))$ pixels. The default is 0.05 noise density.

e)'speckle' for Multiplicative noise.

- $J = \text{imnoise}(I, \text{'speckle'}, v)$ adds multiplicative noise to the image I , using the equation $J = I + n \cdot I$, where n is uniformly distributed random noise with mean 0 and variance v . The default for v is 0.04.

- The mean and variance parameters for 'gaussian', 'localvar', and 'speckle' noise types are always specified as if the image were of class double in the range [0, 1]. If the input image is of class uint8 or uint16, the imnoise function converts the image to double, adds noise according to the specified type and parameters, and then converts the noisy image back to the same class as the input.

$J = \text{imnoise}(I, \text{type}, \text{parameters})$ accepts an algorithm type plus additional modifying parameters particular to the type of algorithm chosen. If you omit these arguments, imnoise uses default values for the parameters. Here are examples of the noise types and their parameters:

2) IMSHOW

$\text{imshow}(I, n)$ displays the intensity image I with n discrete levels of gray. If you omit n , imshow uses 256 gray levels on 24-bit displays, or 64 gray levels on other systems. $\text{imshow}(I, [\text{low high}])$ displays I as a grayscale intensity image, specifying the data range for I . The value low (and any value less than low) displays as black; the value high (and any value greater than high) displays as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels. If you use an empty matrix ($[]$) for [low high], imshow uses $[\text{min}(I(:)) \text{max}(I(:))]$; that is, the minimum value in I is displayed as black, and the maximum value is displayed as white. $\text{imshow}(BW)$ displays the binary image BW . imshow displays pixels with the value 0 (zero) as black and pixels with the value 1 as white. $\text{imshow}(X, \text{map})$ displays the indexed image X with the colormap map . $\text{imshow}(RGB)$ displays the true-color image RGB . $\text{imshow}(\dots, \text{display_option})$ displays the image, where display_option specifies how imshow handles the sizing of the image. display_option is a string that can have either of these values. Option strings can be abbreviated.

3) IMREAD

The imread function supports four general syntaxes, described below.

$A = \text{imread}(\text{filename}, \text{fmt})$ reads a greyscale or color image from the file specified by the string filename , where the string fmt specifies the format of the file. If the file is not in the current directory or in a directory in the MATLAB path, specify the full pathname of the location on your system.

For a list of all the possible values for fmt , see Supported Formats. If imread cannot find a file named filename , it looks for a file named $\text{filename}. \text{fmt}$.

imread returns the image data in the array A. If the file contains a grayscale image, A is a two-dimensional (M-by-N) array. If the file contains a color image, A is a three-dimensional (M-by-N-by-3) array. The class of the returned array depends on the data type used by the file format. See Class Support for more information.

SUPPORT FORMAT

Format : 'bmp'

Full name : window 'bitmap'(BMP)

Variants : 1-bit, 4-bit, 8-bit, 16-bit, 24-bit, and 32-bit uncompressed images and 4-bit and 8-bit run-length encoded (RLE) images.

Format : 'gif'

Full name : Graphics Interchange Format (GIF)

Variants : 1-bit to 8-bit images

Format : 'jpg' or 'jpeg'

Full name : Joint Photographic Experts Group (JPEG)

Variants : Any baseline JPEG image or JPEG image with some commonly used extensions, including:

Image Type, Bitdepth, and Compression.

For most file formats, the color image data returned uses the RGB color space. For TIFF files, however, imread can return color data that uses the RGB, CIELAB, ICCLAB, or CMYK color spaces. If the color image uses the CMYK color space, A is an M-by-N-by-4 array. See the TIFF-Specific Syntax for more information.

[X,map] = imread(filename,fmt) reads the indexed image in filename into X and its associated colormap into map. The colormap values are rescaled to the range [0,1]. [...] = imread(filename) attempts to infer the format of the file from its content. [...] = imread(URL,...) reads the image from an Internet URL. The URL must include the protocol type (e.g., http://).

4) ISPC

Syntax

tf = ispc

Descriptiontf = ispc returns logical true (1) for the PC version of MATLAB and logical false (0) otherwise.

5) Fspecial

Create 2-D special filters Syntax

h = fspecial(type)

h = fspecial(type,parameters)

DESCRIPTION

`h = fspecial(type)` creates a two-dimensional filter `h` of the specified type. `fspecial` returns `h` as a correlation kernel, which is the appropriate form to use with `imfilter`. `Type` is a string having one of these values.

Value : 'gaussian'

Description: Gaussian lowpass filter

Value : 'average'

Description: Averaging filter

Value : 'sobel'

Description: Sobel horizontal edge-emphasizing filter

Value : 'laplacian'

Description: Filter approximating the two-dimensional Laplacian operator

`h = fspecial(type,parameters)` accepts a filter type plus additional modifying parameters particular to the type of filter chosen. If you omit these arguments, `fspecial` uses default values for the parameters.

The following list shows the syntax for each filter type. Where applicable, additional parameters are also shown.

- a) `h = fspecial('average',hsize)` returns an averaging filter `h` of size `hsize`. The argument `hsize` can be a vector specifying the number of rows and columns in `h`, or it can be a scalar, in which case `h` is a square matrix. The default value for `hsize` is `[3 3]`.
- b) `h = fspecial('disk',radius)` returns a circular averaging filter (pillbox) within the square matrix of side `2*radius+1`. The default radius is 5.
- c) `h = fspecial('gaussian',hsize,sigma)` returns a rotationally symmetric Gaussian lowpass filter of size `hsize` with standard deviation `sigma` (positive). `hsize` can be a vector specifying the number of rows and columns in `h`, or it can be a scalar, in which case `h` is a square matrix. The default value for `hsize` is `[3 3]`; the default value for `sigma` is 0.5.
- d) `h = fspecial('laplacian',alpha)` returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator. The parameter `alpha` controls the shape of the Laplacian and must be in the range 0.0 to 1.0. The default value for `alpha` is 0.2.
- e) `h = fspecial('log',hsize,sigma)` returns a rotationally symmetric Laplacian of Gaussian filter of size `hsize` with standard deviation `sigma` (positive). `hsize` can be a vector specifying the number of rows and columns in `h`, or it can be a scalar, in which case `h` is a square matrix. The default value for `hsize` is `[5 5]` and 0.5 for `sigma`.
- f) `h = fspecial('motion',len,theta)` returns a filter to approximate, once convolved with an image, the linear motion of a camera by `len` pixels, with an angle of `theta` degrees in a counterclockwise direction. The filter becomes a vector for horizontal and vertical

- motions. The default len is 9 and the default theta is 0, which corresponds to a horizontal motion of nine pixels.
- g) `h = fspecial('prewitt')` returns a 3-by-3 filter `h` (shown below) that emphasizes horizontal edges by approximating a vertical gradient. If you need to emphasize vertical edges, transpose the filter `h'`. `[1 1 1 0 0 0 -1 -1 -1]` To find vertical edges, or for x-derivatives, use `h'`.
 - h) `h = fspecial('sobel')` returns a 3-by-3 filter `h` (shown below) that emphasizes horizontal edges using the smoothing effect by approximating a vertical gradient. If you need to emphasize vertical edges, transpose the filter `h'`. `[1 2 1 0 0 0 -1 -2 -1]`
 - i) `h = fspecial('unsharp',alpha)` returns a 3-by-3 unsharp contrast enhancement filter. `fspecial` creates the unsharp filter from the negative of the Laplacian filter with parameter `alpha`. `alpha` controls the shape of the Laplacian and must be in the range 0.0 to 1.0. The default value for `alpha` is 0.2.

6) IMFILTER

Multidimensional image filtering

Syntax

`B = imfilter(A,H)`

`B = imfilter(A,H,option1,option2,...)`

Description

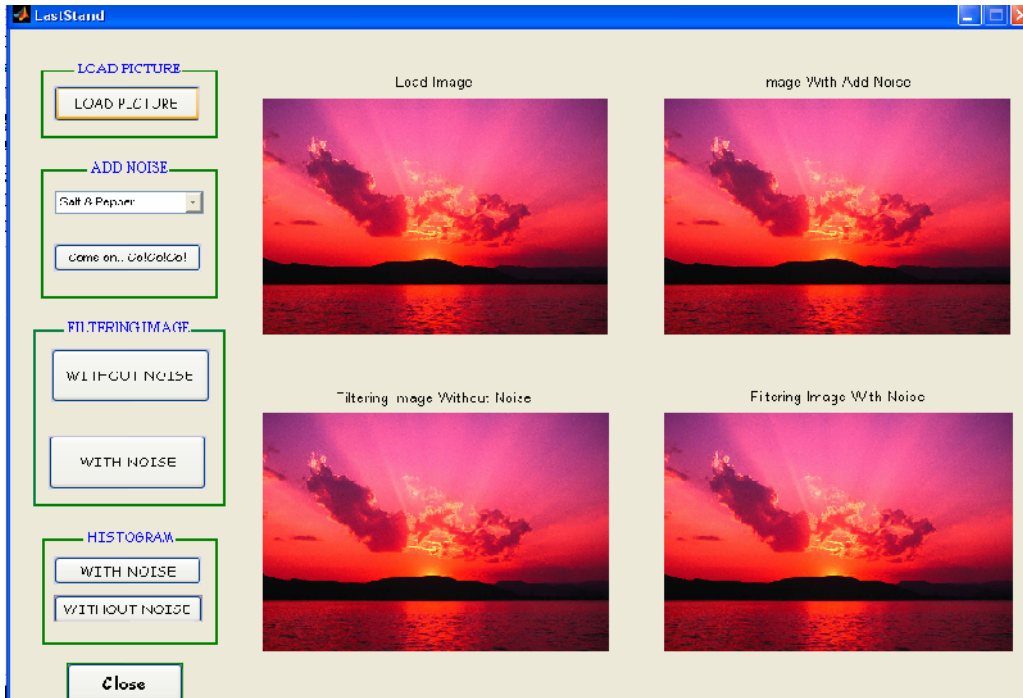
`B = imfilter(A,H)` filters the multidimensional array `A` with the multidimensional filter `H`. The array `A` can be a nonsparse numeric array of any class and dimension. The result `B` has the same size and class as `A`. Each element of the output `B` is computed using double-precision floating point. If `A` is an integer array, then output elements that exceed the range of the integer type are truncated, and fractional values are rounded. `B = imfilter(A,H,option1,option2,...)` performs multidimensional filtering according to the specified options. Option arguments can have the following values.

Boundary Options

Option : 'symmetric'

Description : Input array values outside the bounds of the array are computed by mirror-reflecting the array across the array border.

ANALYSIS



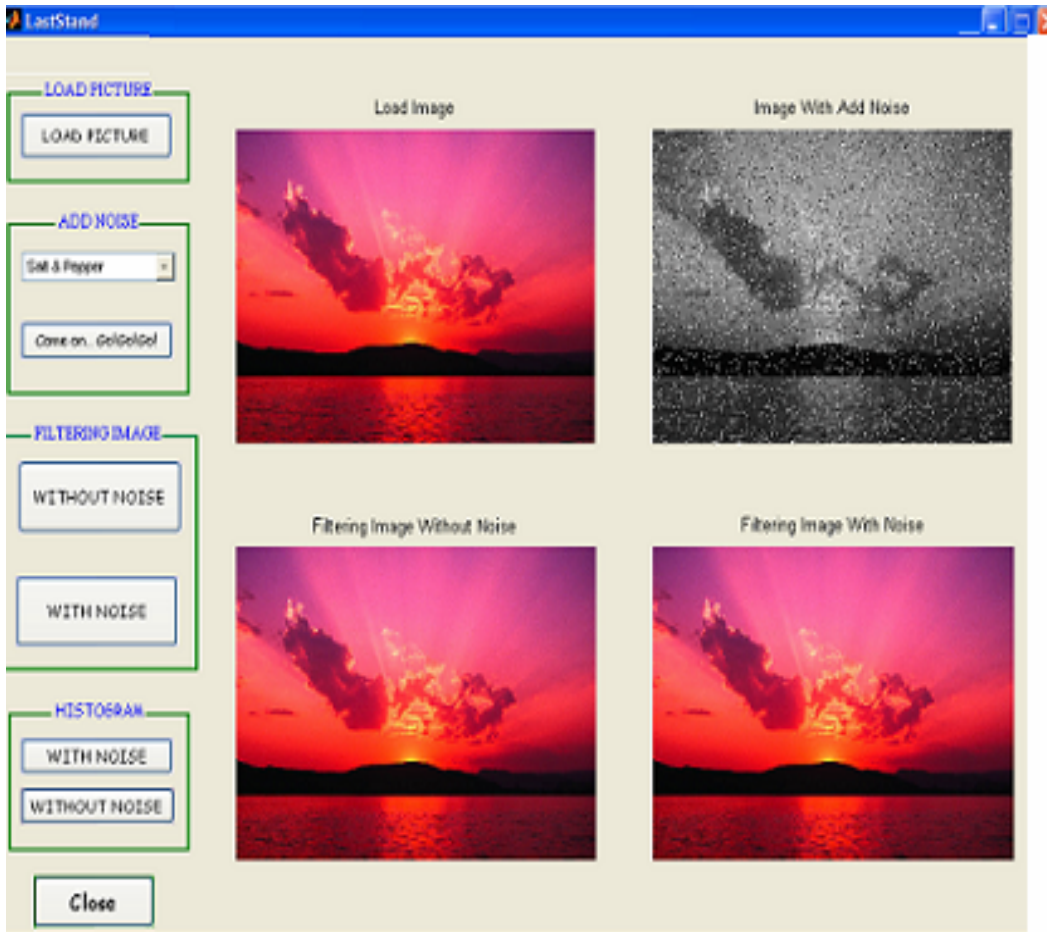
Step

- 1) Press load button picture, will appear place for select image file like figure below:
So, we select what a picture to we choose.

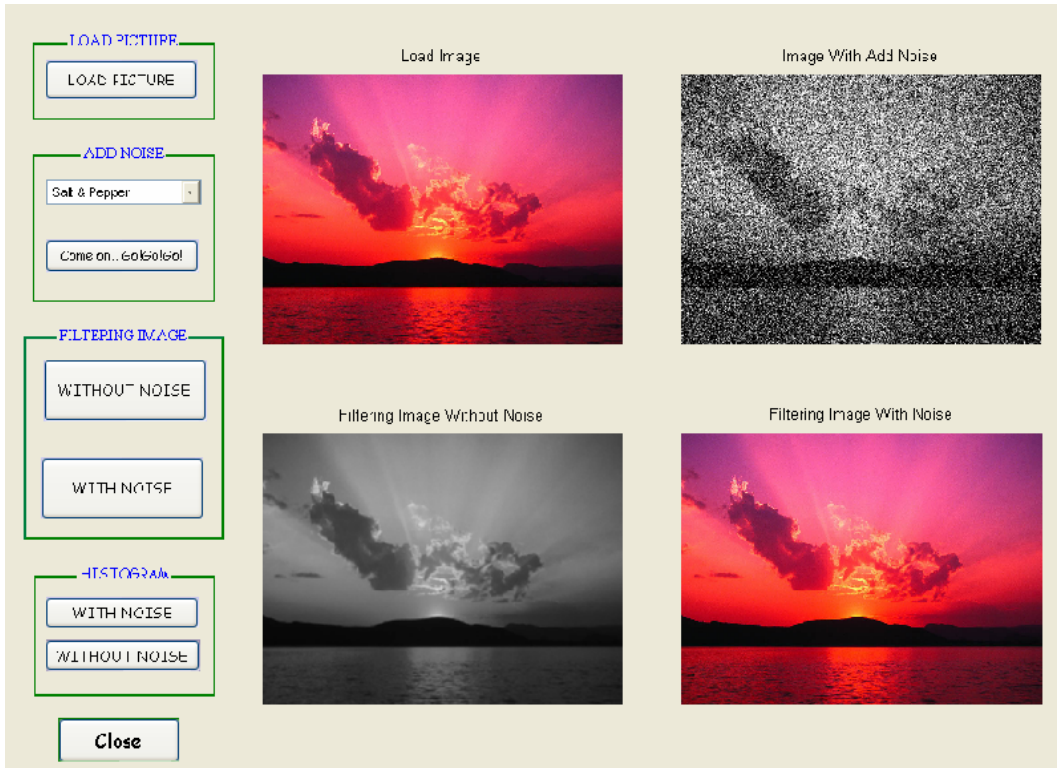


Figure below show when we:

- 1) Click adds noise part and we will find a four noise part like salt and pepper, poisson, Gaussian and speckle.
- 2) For figure below we choose a salt and pepper noise and we got one of from four figure Will change become blurring and not clear.



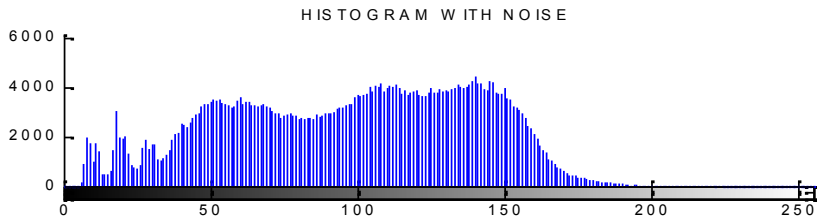
- 3) Figure below show when button for filtering image noise will push, a one of figure will change. A filtering image below show without noise. A color figure will disappear.



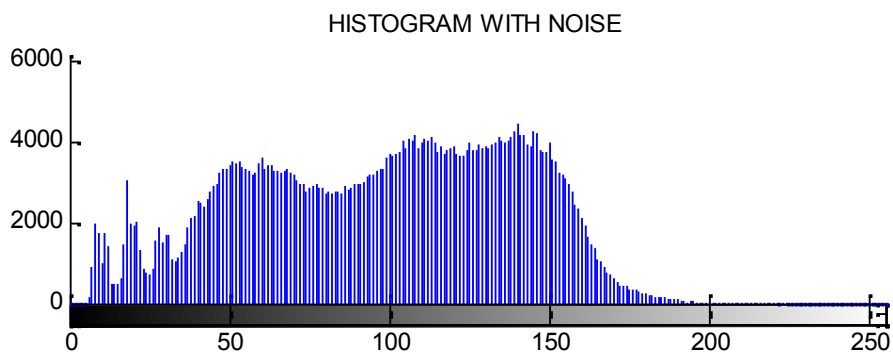
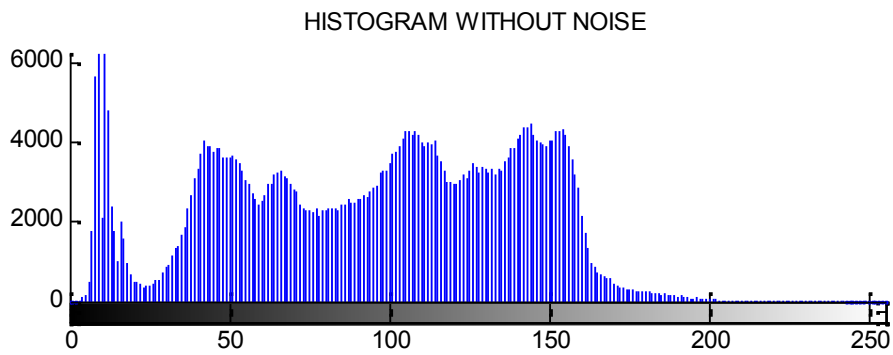
- 4) Figure below show when button for filtering image noise will push, a one of figure will change. A filtering image below show with noise. A color figure will disappear like without noise but have a some blur at figure not clear like without noise figure.



- 5) After filter image part, it will be have a histogram graph. Here have two histogram :
 - a) Histogram with noise.
 - b) Histogram without noise.
- 6) There have a histogram button; we click each one a histogram with noise or histogram without noise. Figure below show histogram with noise



7) Step 6 will repeat and choose for histogram without noise button part. Figure below is show a histogram without noise compared with histogram with noise. So we conclude for analysis, histogram without noise have more higher pixel (6000) from histogram with noise (2000). So that why a figure without noise more clear from a figure with noise.



9.0 DISCUSSION

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. By its typical use, an averaging filter with GUI is able to be created and use.

There are many stages to build the program. The main part is to build the main filtering method. The important command used in the averaging filter is `filt = fspecial('average', [5, 5])`. Here, Average filtering use mean value of array by using 2-D special filter (`fspecial`) by the averaging filter type and parameters 5 rows and 5 columns. After that, the process will continued to multidimensional image filtering between the gray image, the averaging filter with option of input array values outside the bounds of the array are computed by mirror-reflecting the array across the array border ('symmetric') by using the command of `L = imfilter(B, filt, 'symmetric')`.

A desired image is loaded to the program by any sources but only applicable in jpeg image. The image then is plotted to all picture viewers. There are four types of noise which is ready to be use that is Salt & Pepper, Poisson, Gaussian and Speckle. All the noise will give different effect of noise. Users are also able to filter the image with or without noise. Here, there will be different between both filtered images which is the filtered image with noise is smoothen by the noise reduction.

The next important step is to build GUI. The GUI is needed to ease the use of the program. By using push button to load images, the program is able to access any working directory to get images in 'jpeg' format. Pop up menu is used to make selection of noise. Here, the type of noise has been fixed which are Salt & Pepper, Poisson, Gaussian and Speckle. The user can only choose between the given options. To generate the selected noise, the push button 'Come on... Go!Go!Go!' is clicked. Switch case command is used in this stage. Then, images with noise will be generated.

Push buttons also used to generate filtered image with and without noise. Push buttons format used gradually because it is the simplest command to be used rather than other command type.

10.0 CONCLUSION

As a conclusion, we are required to use averaging filter technique to reduce noise in image filtering. Averaging is an operation that takes an image as input, and produces a new image as output. An averaging filter also works on the assumption that the noise in your image is truly random. This way, random fluctuations above and below actual image data will gradually even out as one average more and more images. Then, the image will perform in MATLAB Graphic User Interface.

The GUI can display data in tabular form or as plots, and can group related components. GUI is also a type of user interface which allows people to interact with a computer and computer-controlled devices which employ graphical icons, visual indicators or special graphical elements called "widgets", along with text, labels or text navigation to represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements. The GUI are tools simplify the process of laying out and programming GUIs. An result will show and original image that was added by noise and the filtered image using averaging filter. So we can compare an original images and filtered images. Graphic User Interface was build with all needed parameters to show an averaging filtering process

11.0 BIBLIOGRAPHY

- [1] Tinku Acharya and Ajoy K. Ray. *Image Processing Principles and Application*.A John Willey & sons,inc.,Publication. 2005.
- [2] R.C. Gonzalez and R.E. Woods, Digital Image Processing, Addison Wesley, Reading, MA, 1992
- [3] W.K. Pratt, Digital Image Processing, 2nd ed., Wiley, New York, 1991
- [4] http://en.wikipedia.org/wiki/Image_processing
- [5] <http://www.reindeergraphics.com>