

PRACTICAL 1

Aim: Introduction to Digital Image Processing and MATLAB

Digital Image Processing

Digital Image Processing means processing digital image by means of a digital computer. We can also say that it is a use of computer algorithms, in order to get enhanced image either to extract some useful information.

Image processing mainly include the following steps:

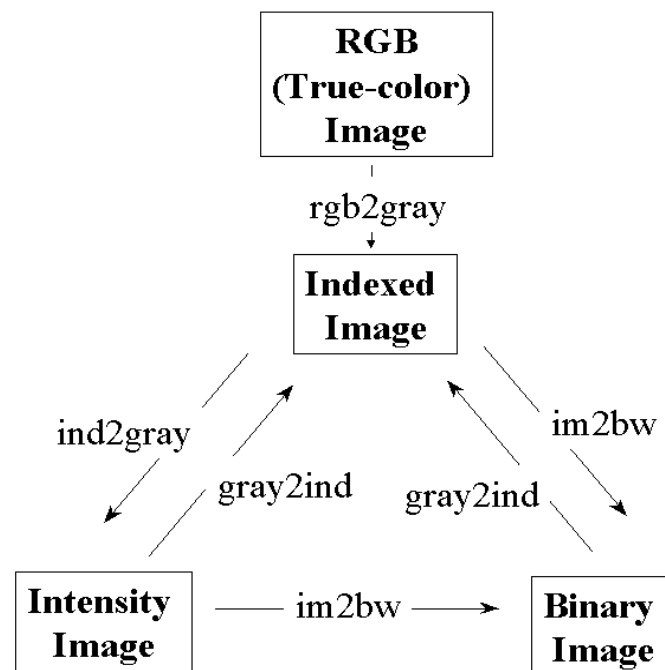
- 1.Importing the image via image acquisition tools;
- 2.Analysing and manipulating the image;
- 3.Output in which result can be altered image or a report which is based on analysing that image.

What is an image?

An image is defined as a two-dimensional function, $F(x,y)$, where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x,y) is called the **intensity** of that image at that point. When x,y , and amplitude values of F are finite, we call it a **digital image**. In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns. Digital Image is composed of a finite number of elements, each of which elements have a particular value at a particular location. These elements are referred to as *picture elements*, *image elements*, and *pixels*. A *Pixel* is most widely used to denote the elements of a Digital Image.

Types of an image

1. **BINARY IMAGE**– The binary image as its name suggests, contain only two pixel elements i.e. 0 & 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.
2. **BLACK AND WHITE IMAGE**– The image which consist of only black and white color is called BLACK AND WHITE IMAGE.
3. **8 bit COLOR FORMAT**– It is the most famous image format. It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for grey.
4. **16 bit COLOR FORMAT**– It is a color image format. It has 65,536 different colors in it. It is also known as High Color Format. In this format the distribution of color is not as same as Grayscale image.



1. TIFF (also known as TIF), file types ending in .tif

TIFF stands for Tagged Image File Format. TIFF images create very large file sizes. TIFF images are uncompressed and thus contain a lot of detailed image data (which is why the files are so big) TIFFs are also extremely flexible in terms of color (they can be grayscale, or CMYK for print, or RGB for web) and content (layers, image tags).

TIFF is the most common file type used in photo software (such as Photoshop), as well as page layout software (such as Quark and InDesign), again because a TIFF contains a lot of image data.

2. JPEG (also known as JPG), file types ending in .jpg

JPEG stands for Joint Photographic Experts Group, which created this standard for this type of image formatting. JPEG files are images that have been compressed to store a lot of information in a small-size file. Most digital cameras store photos in JPEG format, because then you can take more photos on one camera card than you can with other formats.

A JPEG is compressed in a way that loses some of the image detail during the compression in order to make the file small (and thus called “lossy” compression).

JPEG files are usually used for photographs on the web, because they create a small file that is easily loaded on a web page and also looks good.

JPEG files are bad for line drawings or logos or graphics, as the compression makes them look “bitmappy” (jagged lines instead of straight ones).

3. GIF, file types ending in .gif

GIF stands for Graphic Interchange Format. This format compresses images but, as different from JPEG, the compression is lossless (no detail is lost in the compression, but the file can't be made as small as a JPEG).

GIFs also have an extremely limited color range suitable for the web but not for printing. This format is never used for photography, because of the limited number of colors. GIFs can also be used for animations.

4. PNG, file types ending in .png

PNG stands for Portable Network Graphics. It was created as an open format to replace GIF, because the patent for GIF was owned by one company and nobody else wanted to pay licensing fees. It also allows for a full range of color and better compression.

It's used almost exclusively for web images, never for print images. For photographs, PNG is not as good as JPEG, because it creates a larger file. But for images with some text, or line art, it's better, because the images look less "bitmappy."

When you take a screenshot on your Mac, the resulting image is a PNG—probably because most screenshots are a mix of images and text.

PHASES OF IMAGE PROCESSING:

1.**ACQUISITION**— It could be as simple as being given an image which is in digital form. The main work involves:

- a) Scaling
- b) Color conversion (RGB to Gray or vice-versa)

2.**IMAGE ENHANCEMENT**— It is amongst the simplest and most appealing in areas of Image Processing it is also used to extract some hidden details from an image and is subjective.

3.**IMAGE RESTORATION**— It also deals with appealing of an image but it is objective(Restoration is based on mathematical or probabilistic model or image degradation).

4.**COLOR IMAGE PROCESSING**— It deals with pseudocolor and full color image processing color models are applicable to digital image processing.

5.**WAVELETS AND MULTI-RESOLUTION PROCESSING**— It is foundation of representing images in various degrees.

6.**IMAGE COMPRESSION**-It involves in developing some functions to perform this operation. It mainly deals with image size or resolution.

7.**MORPHOLOGICAL PROCESSING**-It deals with tools for extracting image components that are useful in the representation & description of shape.

8.SEGMENTATION PROCEDURE-It includes partitioning an image into its constituent parts or objects. Autonomous segmentation is the most difficult task in Image Processing.

9.REPRESENTATION & DESCRIPTION-It follows output of segmentation stage, choosing a representation is only the part of solution for transforming raw data into processed data.

10.OBJECT DETECTION AND RECOGNITION-It is a process that assigns a label to an object based on its descriptor.

MATLAB

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

As of 2018, MATLAB has more than 3 million users worldwide. MATLAB users come from various backgrounds of engineering, science, and economics.

The MATLAB application is built around the MATLAB scripting language. Common usage of the MATLAB application involves using the Command Window as an interactive mathematical shell or executing text files containing MATLAB code.

Getting started with MATLAB:

It is both a programming language as well as a programming environment. It allows the computation of statements in the command window itself.

- **Command Window:**

In this window one must type and immediately execute the statements, as it requires quick prototyping. These statements cannot be saved. Thus, this is can be used for small, easily executable programs.

- **Editor (Script):**

In this window one can execute larger programs with multiple statements, and complex functions. These can be saved and are done with the file extension '.m'.

- **Workspace:**

In this window the values of the variables that are created in the course of the program (in the editor) are displayed.

MATLAB Library comes with a set of many inbuilt functions. These functions mostly perform mathematical operations like sine, cosine and tangent. They perform more complex functions too like finding the inverse and determinant of a matrix, cross product and dot product.

Although MATLAB is encoded in C, C++ and Java, it is a lot easier to implement than these three languages. For example, unlike the other three, no header files need to be initialised in the beginning of the document and for declaring a variable, the data type need not be provided. It provides an easier alternative for vector operations. They can be performed using one command instead of multiple statements in a for or while loop.

Writing a MATLAB program:

1. **Using Command Window:**

Only one statement can be typed and executed at a time. It executes the statement when the enter key is pressed. This is mostly used for simple calculations.

Note: ans is a default variable created by MATLAB that stores the output of the given computation.

2. **Using Editor:**

Multiple lines of code can be written here and only after pressing the run button (or F5) will the code be executed. It is always a good practice to write `clc`, `clear` and `close` all in the beginning of the program.

Note: Statements ending with a semicolon will not be displayed in the command window, however, their values will be displayed in the workspace.

Any statement followed by % in MATLAB is considered as a comment.

3. **Vector Operations:**

Operations such as addition, subtraction, multiplication and division can be done using a single command instead of multiple loops.

PRACTICAL 2

Aim: (a) Reading and displaying images in defined format using different color models

Reading an image using imread()

`A = imread(filename)` reads the image from the file specified by filename, inferring the format of the file from its contents.

- If filename is a multi-image file, then imread reads the first image in 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.
- If imread cannot find a file named filename, it looks for a file named filename.*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.

Displaying image using imshow()

`imshow(I)` displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.

Algorithm:

1. Find the images in different formats
2. Read the image using Matlab functions
3. Display the image read

Screenshots:

Input:

JPEG Image



PNG image



TIFF Image

**Code:**

```
Editor - D:\DIP_LAB\Experiment2.m
Experiment2.m x +
1 % JPG Image
2 - img_jpg = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
3 - imshow(img_jpg);
4
5 % PNG Image
6 - img_png = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.png');
7 - imshow(img_png);
8
9 % TIFF Image
10 - img_tiff = imread('C:\Users\shiva\OneDrive\Documents\DIP\minionstiff.tiff');
11 - imshow(img_tiff);
```

Aim: (b) To display image using different color models

Colour Models:

Colour models provide a standard way to specify a particular colour, by defining a 3D coordinate system, and a subspace that contains all constructible colours within a particular model. Any colour that can be specified using a model will correspond to a single point within the subspace it defines. Each colour model is oriented towards either specific hardware (RGB, CMY, YIQ), or image processing applications (HSI).

The RGB Model:

In the RGB model, an image consists of three independent image planes, one in each of the primary colours: red, green and blue. Specifying a particular colour is by specifying the amount of each of the primary components present. The grayscale spectrum, i.e. those colours made from equal amounts of each primary, lies on the line joining the black and white vertices.

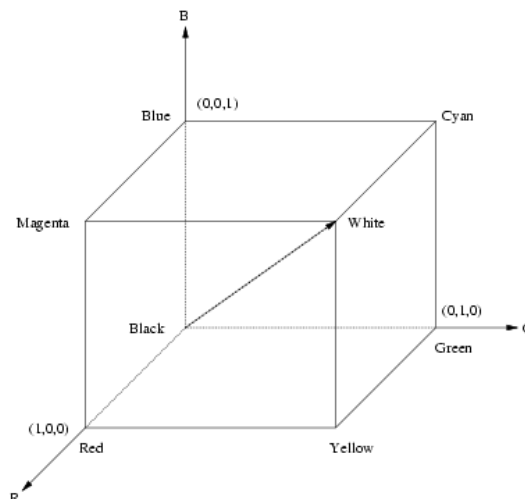


Fig 1: The RGB colour cube. The greyscale spectrum lies on the line joining the black and white vertices.

The Grayscale Model:

Grayscale is a range of monochromatic shades from black to white. Therefore, a grayscale image contains only shades of gray and no color. This model removes all color information, leaving only the luminance of each pixel.

The HSI Model :

Colour may be specified by the three quantities hue, saturation and intensity. This is the HSI model, and the entire space of colours that may be specified in this way is shown in figure.

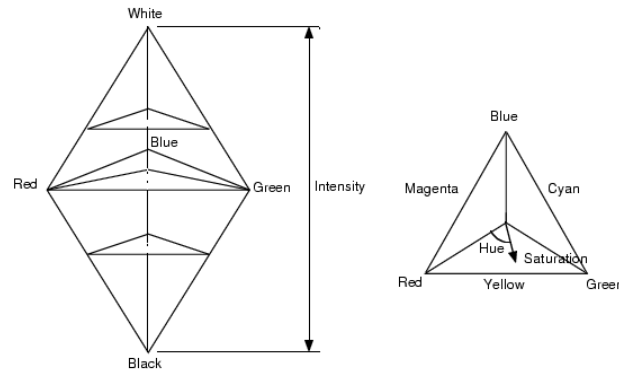


Fig 2: The HSI model, showing the HSI solid on the left, and the HSI triangle on the right, formed by taking a horizontal slice through the HSI solid at a particular intensity. Hue is measured from red, and saturation is given by distance from the axis. Colours on the surface of the solid are fully saturated, i.e. pure colours, and the greyscale spectrum is on the axis of the solid. For these colours, hue is undefined.

Algorithm:

1. Read the image.
2. Convert the image to grayscale image using `rgb2gray()`
3. Convert the original image to HSV using `rgb2hsv()`
4. Display all the images

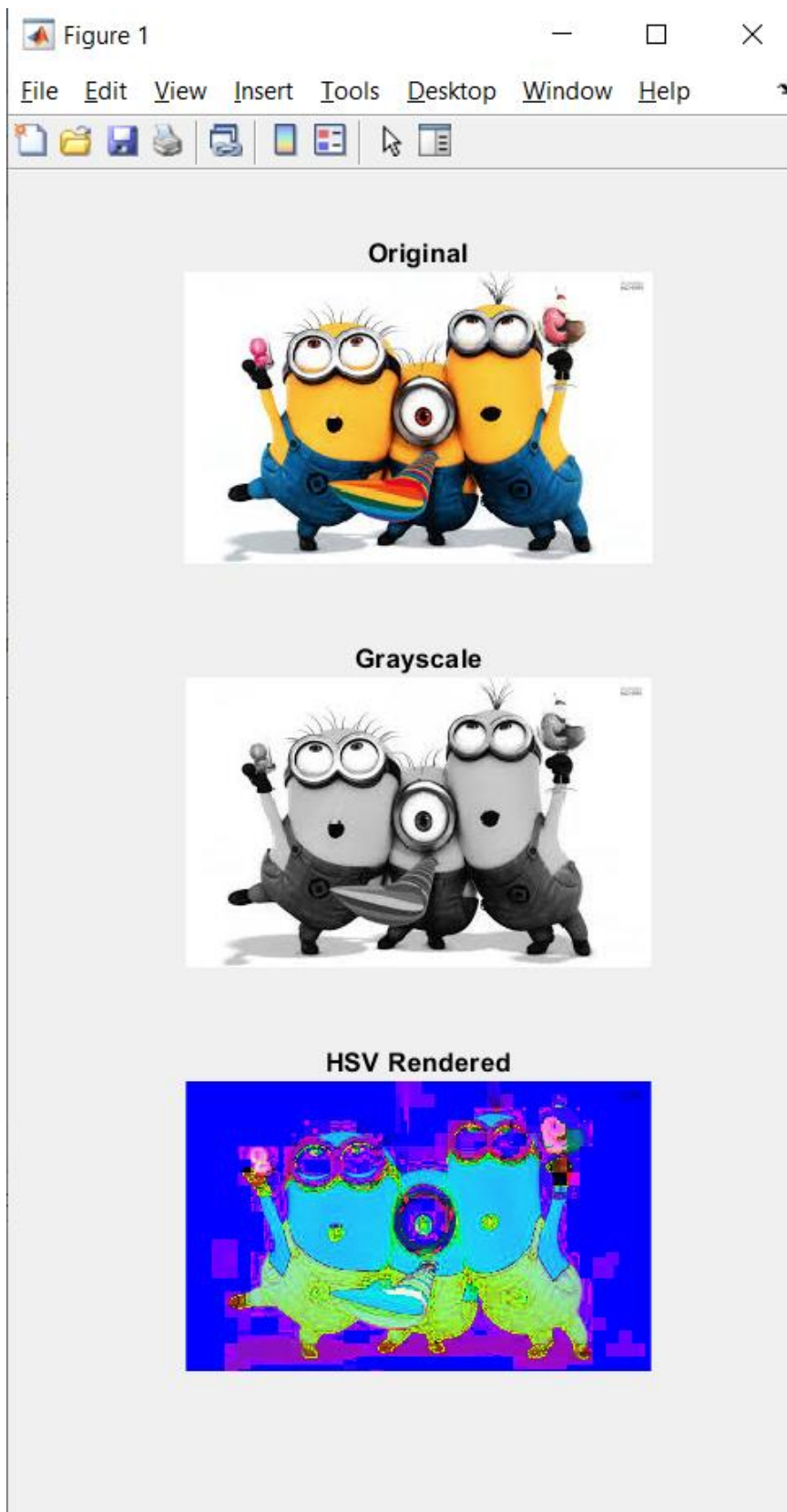
Screenshots:

Code:

```

Editor - D:\DIP_LAB\Experiment3.m
Experiment2.m Experiment3.m +
1 - i = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2
3 % RGB to Grayscale
4 - img_gs = rgb2gray(i);
5 - imshow(img_gs);
6
7 % RGB to HSV
8 - img_hsv = rgb2hsv(i);
9 - imshow(img_hsv);
10
11 % Subplots
12 - subplot(3, 1, 1), imshow(i), title('Original');
13 - subplot(3, 1, 2), imshow(img_gs), title('Grayscale');
14 - subplot(3, 1, 3), imshow(img_hsv), title('HSV Rendered');

```

Output Image:

PRACTICAL 3

Aim: Conversion of a RGB image to monochrome Image.

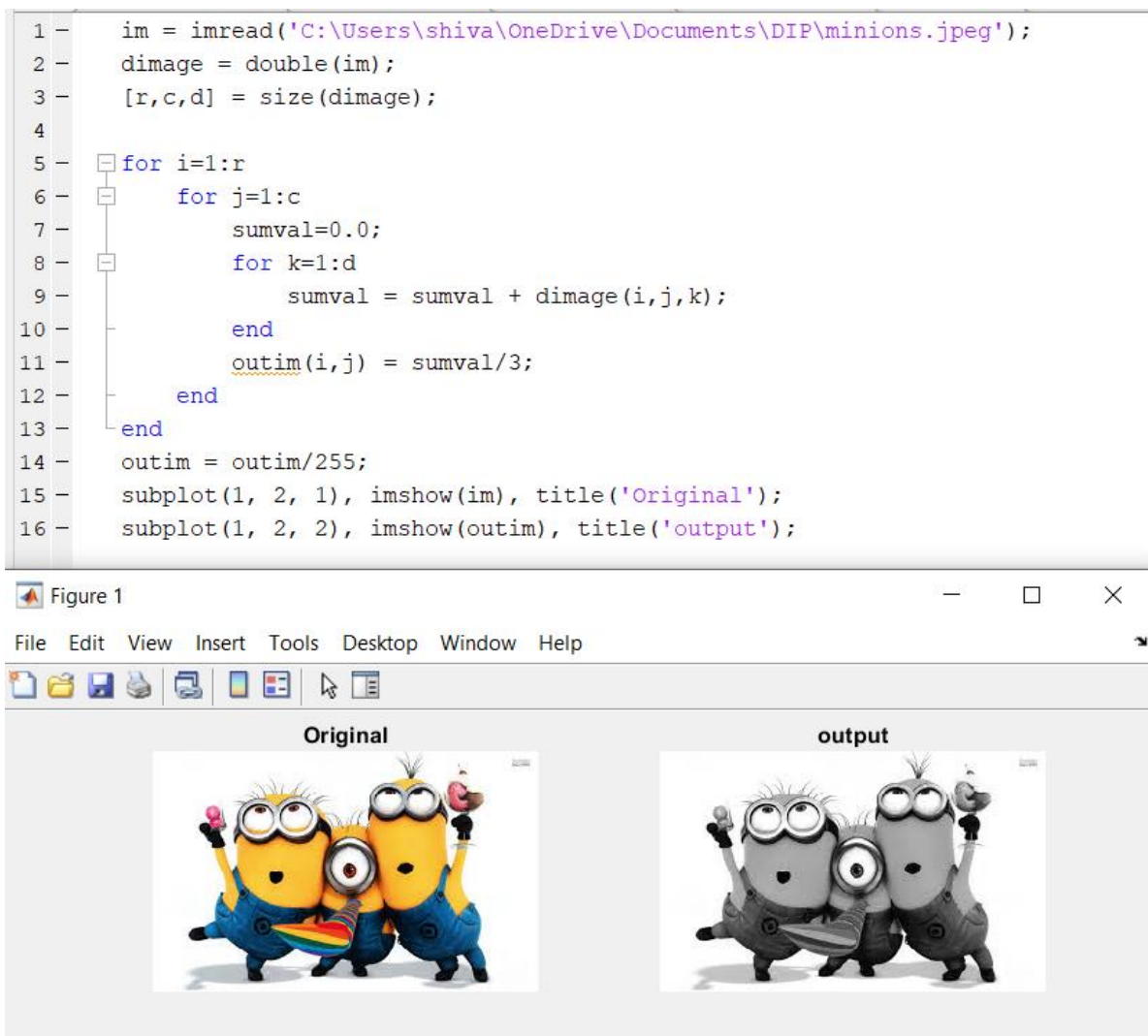
Algorithm:

1. Read the image
2. For each pixel f of the image
 - a. Convert to monochrome value
 - b. Store new values
3. Display the image

Screenshots

Converting RGB image to grayscale image using algorithm.

Code and output



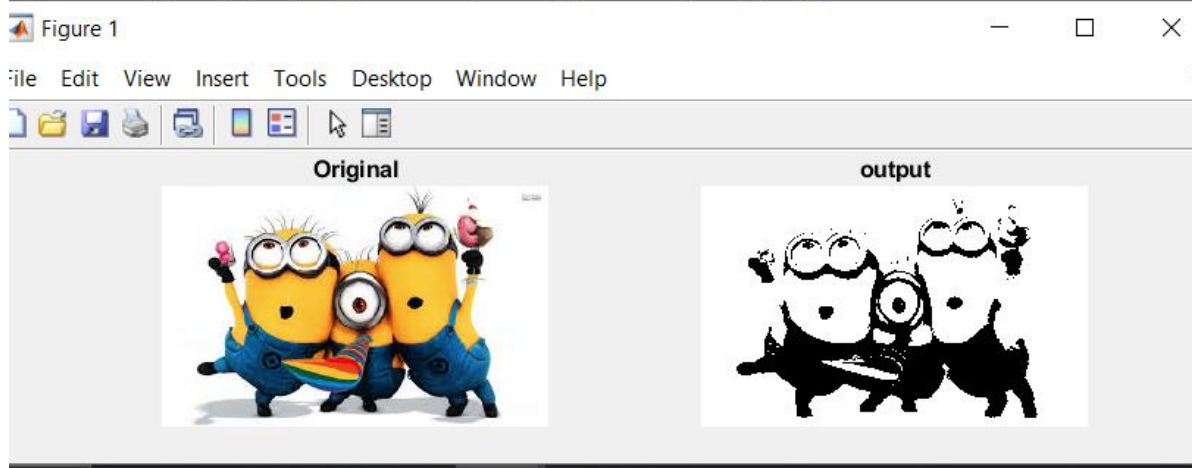
Converting RGB image to Black and white image using algorithm

Code and output

```

1 - im = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2 - dimage = double(im);
3 - [r,c,d] = size(dimage);
4
5 - for i=1:r
6 -     for j=1:c
7 -         sumval=0.0;
8 -         for k=1:d
9 -             sumval = sumval + dimage(i,j,k);
10 -        end
11 -        if(sumval/3)>127
12 -            outim(i,j)=1;
13 -        else
14 -            outim(i,j)=0;
15 -        end
16 -    end
17 - end
18 - subplot(1, 2, 1), imshow(im), title('Original');
19 - subplot(1, 2, 2), imshow(outim), title('output');

```



PRACTICAL 4

Aim: Image enhancement using gray level transformation and spatial and frequency domain filters.

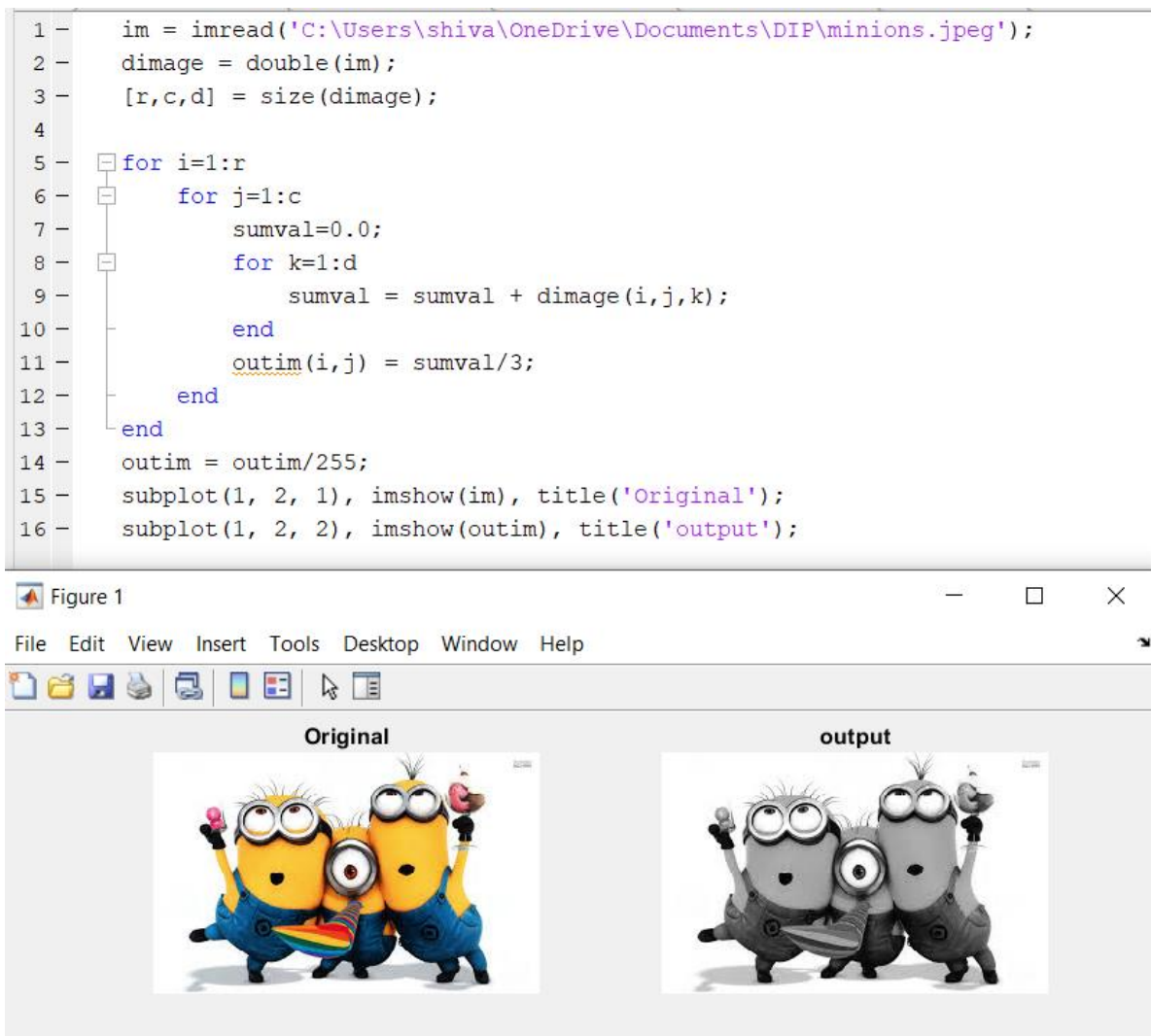
Algorithm:

1. Read the image
2. For each pixel f of the image
 - a. Apply different grey level transformation function like image negative, contrast stretching, brightness enhancement log powerlog to each pixel value.
 - b. Store new values.
3. Display the image

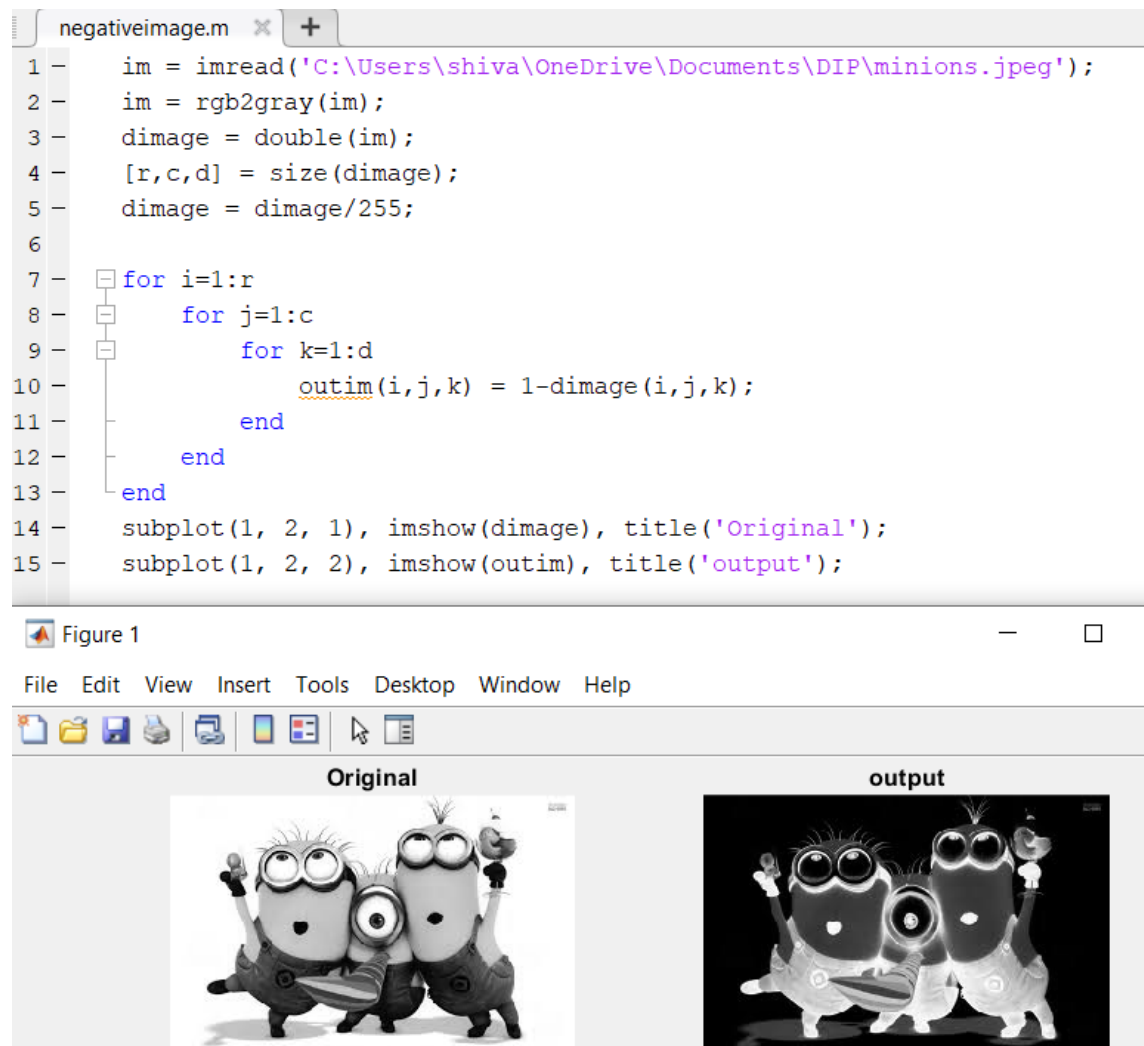
Screenshots

Gray Level Transform

Code and output



Negative of Image



Contrast Stretching

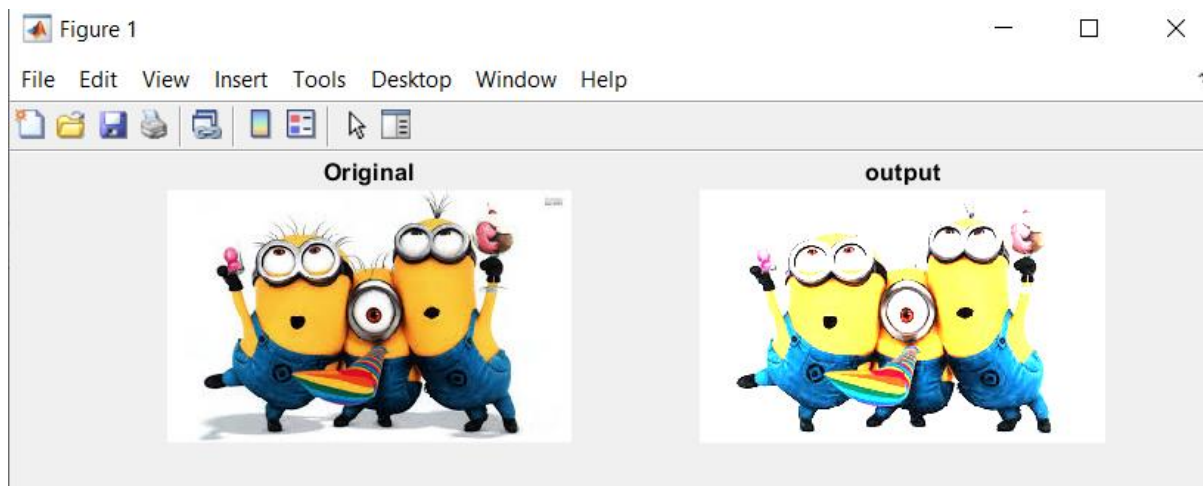
Code

```

1 -  im = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2
3 -  dimage = double(im);
4
5 -  [r,c,d] = size(dimage);
6
7 -  for i=1:r
8 -      for j=1:c
9 -          for k=1:d
10 -              outim(i,j,k) = 2*dimage(i,j,k);
11 -          end
12 -      end
13 -  end
14 -  outim = outim/255;
15 -  subplot(1, 2, 1), imshow(im), title('Original');
16 -  subplot(1, 2, 2), imshow(outim), title('output');

```


Output



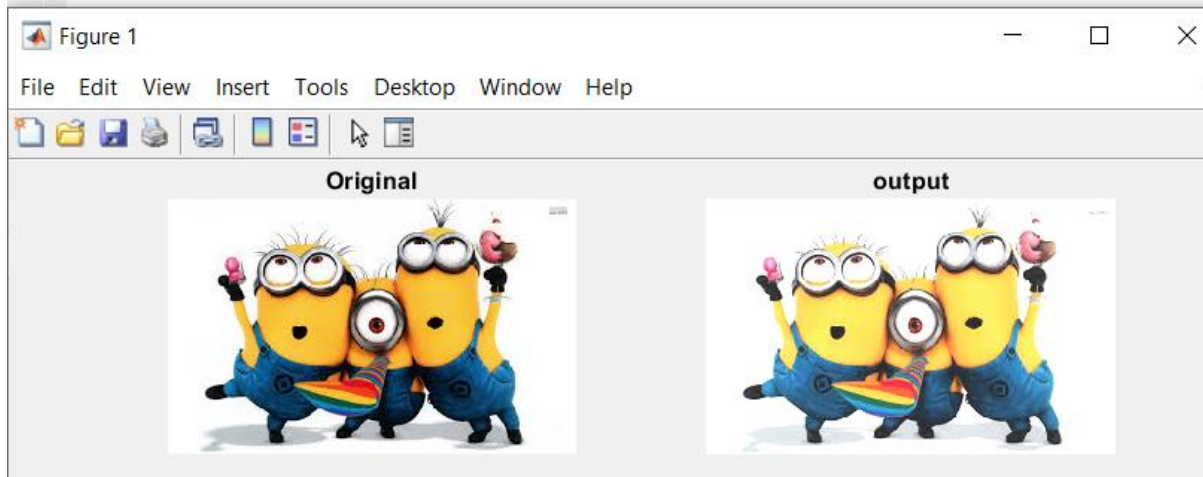
Brightness Enhancement

Code and Output

```

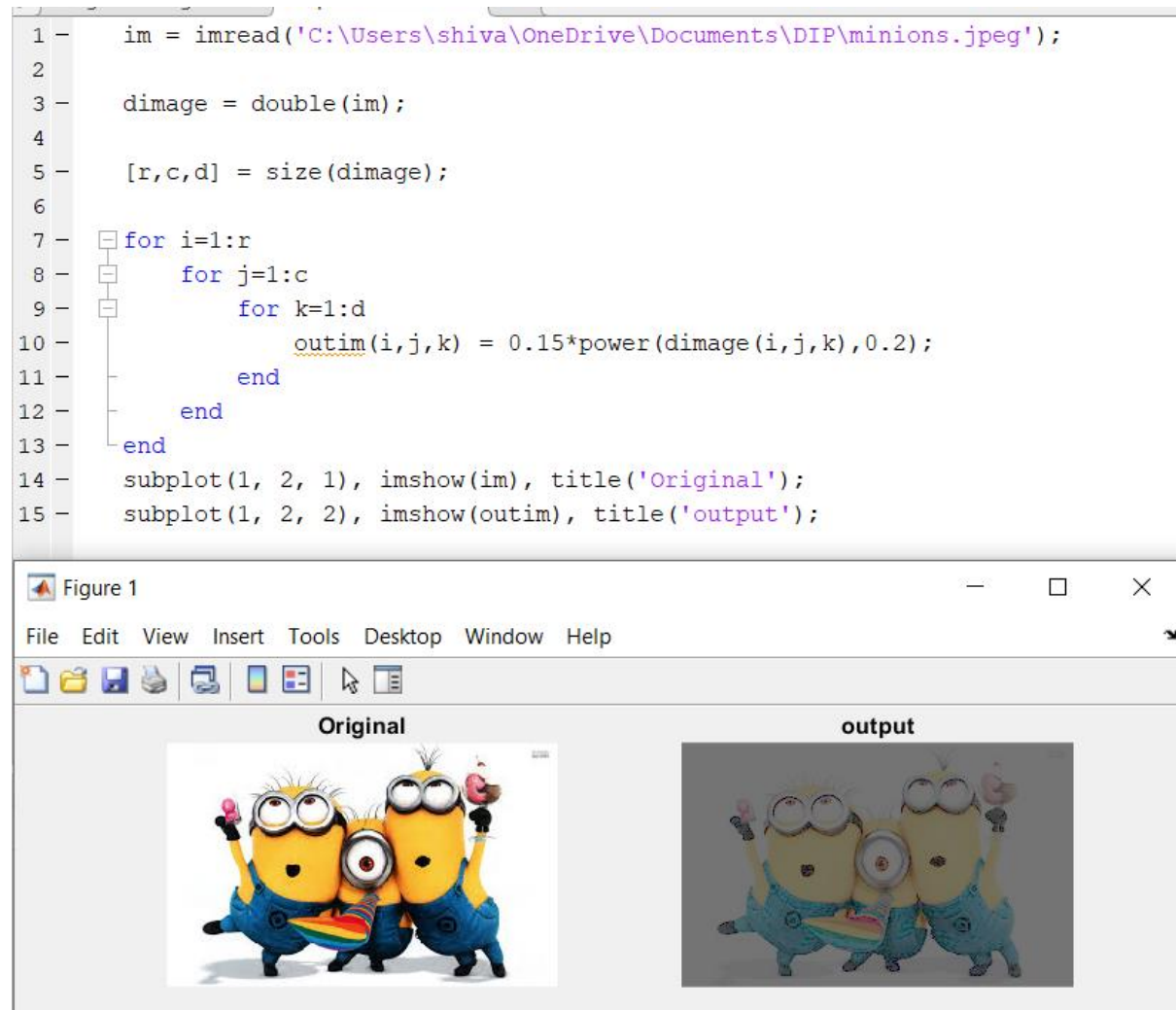
1 - im = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2
3 - dimage = double(im);
4
5 - [r,c,d] = size(dimage);
6
7 - for i=1:r
8 -     for j=1:c
9 -         for k=1:d
10 -             outim(i,j,k) = 40+dimage(i,j,k);
11 -         end
12 -     end
13 - end
14 - outim = outim/255;
15 - subplot(1, 2, 1), imshow(im), title('Original');
16 - subplot(1, 2, 2), imshow(outim), title('output');

```



Power Transform

Code and Output



Log Transform

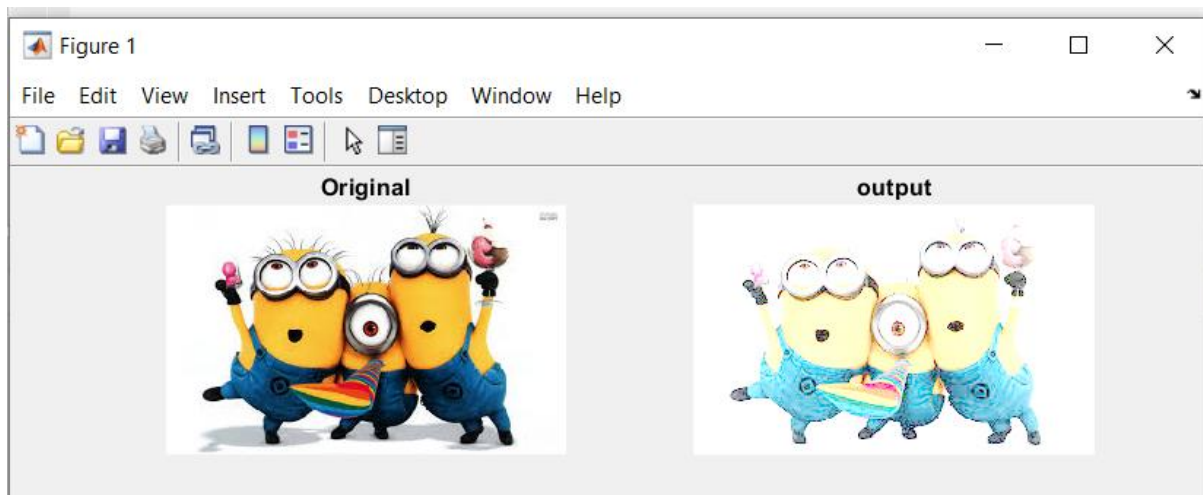
Code

```

1 - im = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2
3 - dimage = double(im);
4
5 - [r,c,d] = size(dimage);
6
7 - for i=1:r
8 -     for j=1:c
9 -         for k=1:d
10 -             outim(i,j,k) = 0.2*log(1+dimage(i,j,k));
11 -         end
12 -     end
13 - end
14 - subplot(1, 2, 1), imshow(im), title('Original');
15 - subplot(1, 2, 2), imshow(outim), title('output');

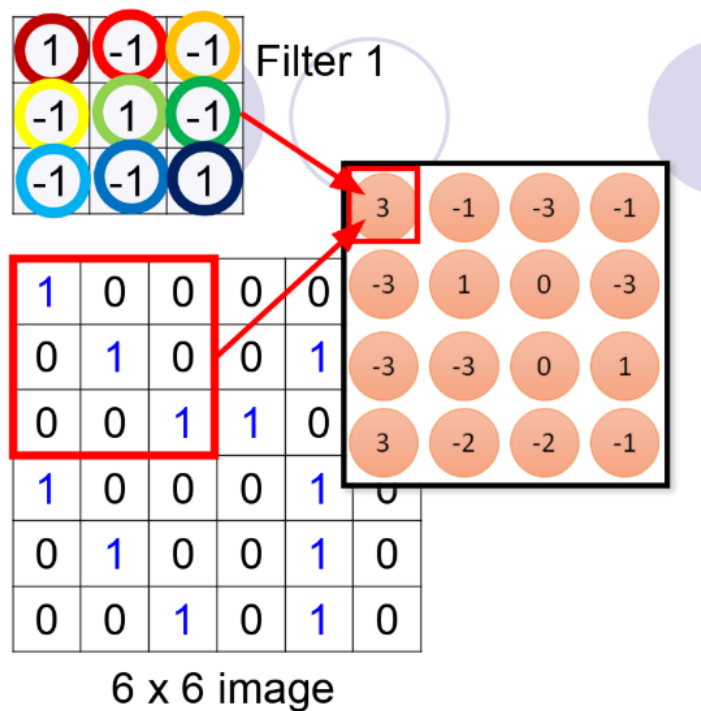
```


Output



Spatial Filtering

Spatial filtering is an image processing technique for changing the intensities of a pixel according to the intensities of the neighbouring pixels. Using spatial filtering, the image is transformed (convolved) based on a kernel H which has certain height and width (x,y) , defining both the area and the weight of the pixels within the initial image that will replace the value of the image. The corresponding process is to convolve the input image $I(i,j)$ with the filter function $H(x,y)$, to produce the new filtered image



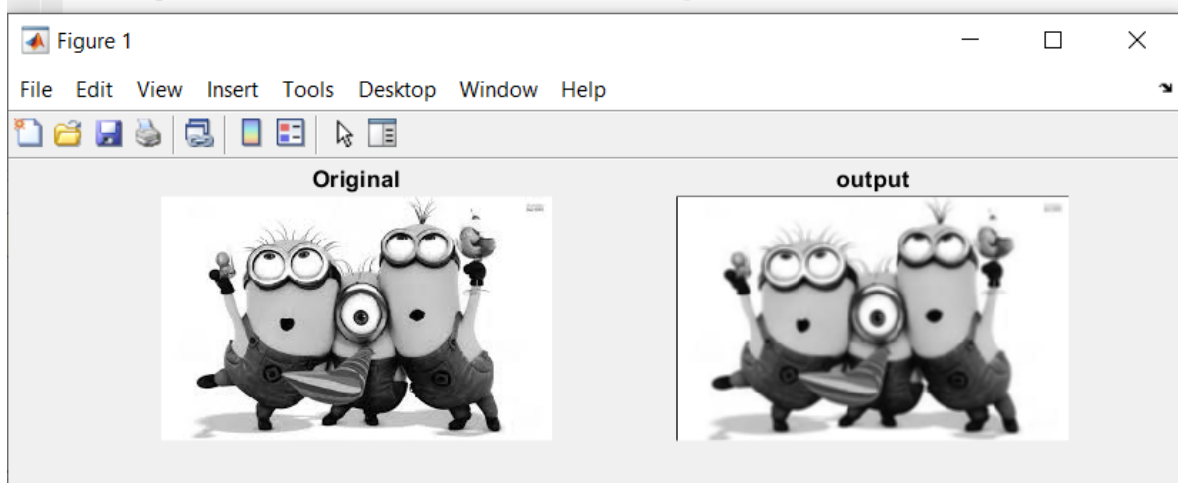
Median Filter Application

Code and Output

```

1 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
2 - I = double(im);
3 - [x,y] = size(I);
4 - for i = 2:x-1
5 -     for j = 2:y-1
6 -         sum = double(0);
7 -         for ii = i-1:i+1
8 -             for jj = j-1:j+1
9 -                 sum = sum + I(ii,jj);
10 -            end
11 -        end
12 -        I2(i,j) = ceil(sum/9);
13 -    end
14 - end
15 - I2 = I2/255;
16 - subplot(1, 2, 1), imshow(im), title('Original');
17 - subplot(1, 2, 2), imshow(I2), title('output');

```



Gaussian Low Pass Filter

Code

```

1 - Img = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2 - Im = rgb2gray(Img);
3 - I = double(Im);
4 -
5 - sigma = 1.76;
6 - sz = 4;
7 - [x,y]=meshgrid(-sz:sz,-sz:sz);
8 -

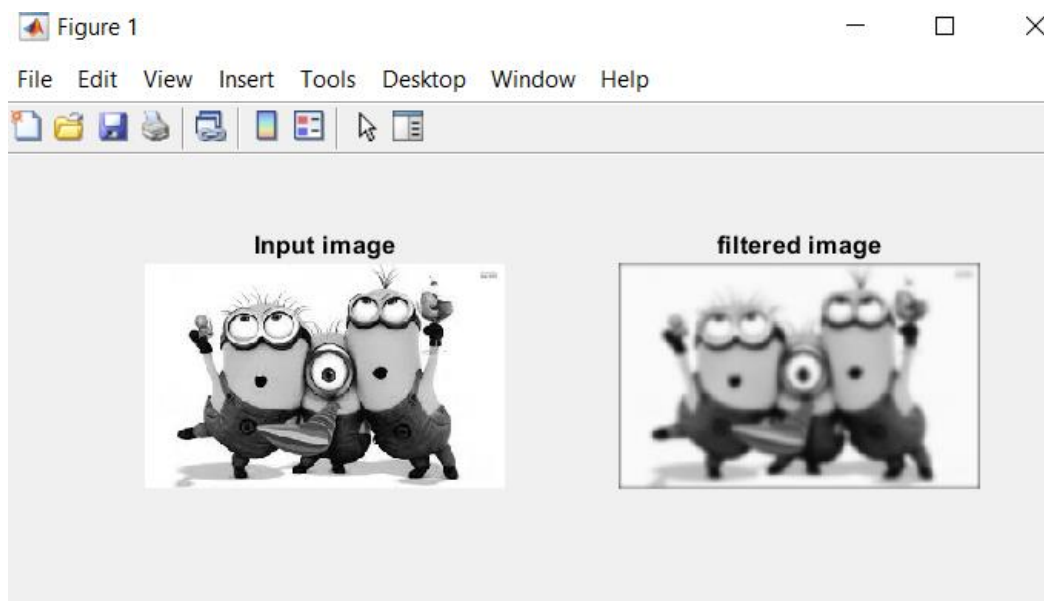
```

```

9 - M = size(x,1)-1;
10 - N = size(y,1)-1;
11 - Exp_comp = -(x.^2+y.^2)/(2*sigma*sigma);
12 - Kernel= exp(Exp_comp)/(2*pi*sigma*sigma);
13 - Output=zeros(size(I));
14 - I = padarray(I,[sz sz]);
15 - for i = 1:size(I,1)-M
16 -     for j =1:size(I,2)-N
17 -         Temp = I(i:i+M,j:j+M).*Kernel;
18 -         Output(i,j)=sum(Temp(:));
19 -     end
20 - end
21 - Output = uint8(Output);
22
23 - subplot(1,2,1); imshow(I); title('Input image');
24 - subplot(1,2,2); imshow(Output); title('filtered image');
25

```

Output



Gaussian High Pass Filter

Code

```

+1 experiment5.m x medianfilter.m x meanFilter.m x weiner.m x poissonnoise.m x Gaus
1 - Img = imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
2 - Im = rgb2gray(Img);
3 - I = double(Im);
4 - sigma = 1.7/6;
5
6 - sz = 4;
7 - [x,y]=meshgrid(-sz:sz,-sz:sz);
8

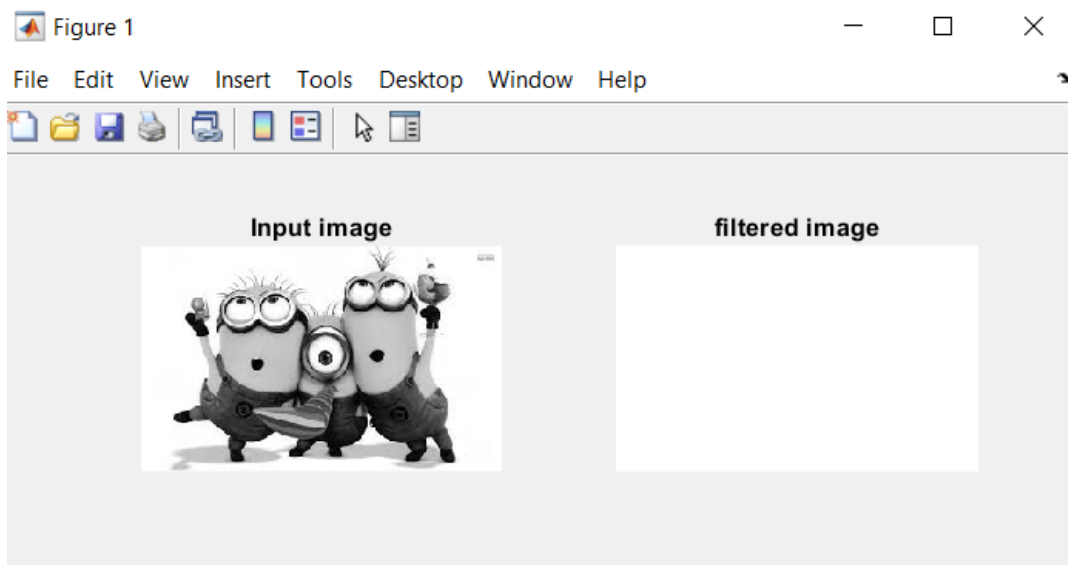
```

```

9 - M = size(x,1)-1;
10 - N = size(y,1)-1;
11 - Exp_comp = -(x.^2+y.^2)/(2*sigma*sigma);
12 - Kernel= 1-exp(Exp_comp)/(2*pi*sigma*sigma);
13 - Output=zeros(size(I));
14 - I = padarray(I,[sz sz]);
15
16 - for i = 1:size(I,1)-M
17 -     for j =1:size(I,2)-N
18 -         Temp = I(i:i+M,j:j+M).*Kernel;
19 -         Output(i,j)=sum(Temp(:));
20 -     end
21 - end
22 - Output = uint8(Output);
23
24 - subplot(1,2,1); imshow(Im); title('Input image');
25 - subplot(1,2,2); imshow(Output); title('filtered image');
26

```

Output



Butterworth Low Pass Filter

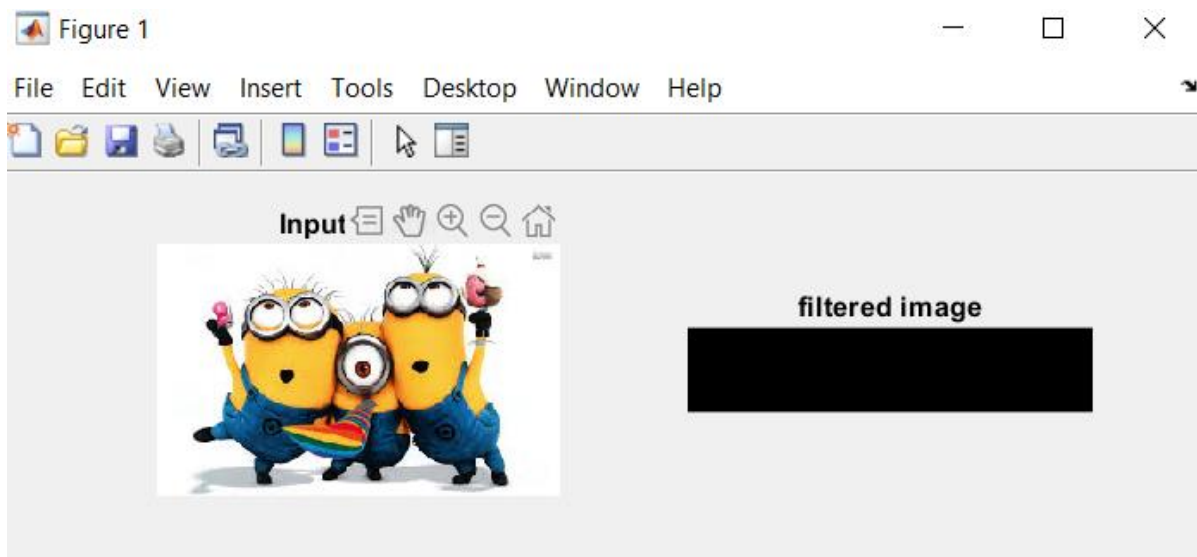
Code

```

1 - n=1;
2 - D0=50; % change the name for d0, d is usually the  $(u^2+v^2)^{1/2}$ 
3 - d0=10;
4 - A=1.5; % normally the amplitude is 1
5 - im=imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
6 - [M,N]=size(im); % is easy to get the h and w like this
7
8 - H=zeros(M,N);
9 - for u=0:(M-1)
10 -     for v=0:(N-1)
11 -         dist=(i-M/2)^2 + (j-N/2)^2;
12 -         H(i,j)= ( 1 + (dist/d0)^(2*n) )^(-1);
13 -     end
14 - end
15 - subplot(1,2,1); imshow(im); title('Input image');
16 - subplot(1,2,2); imshow(H,[ ]); title('filtered image');
17

```

Output



Butterworth High Pass Filter

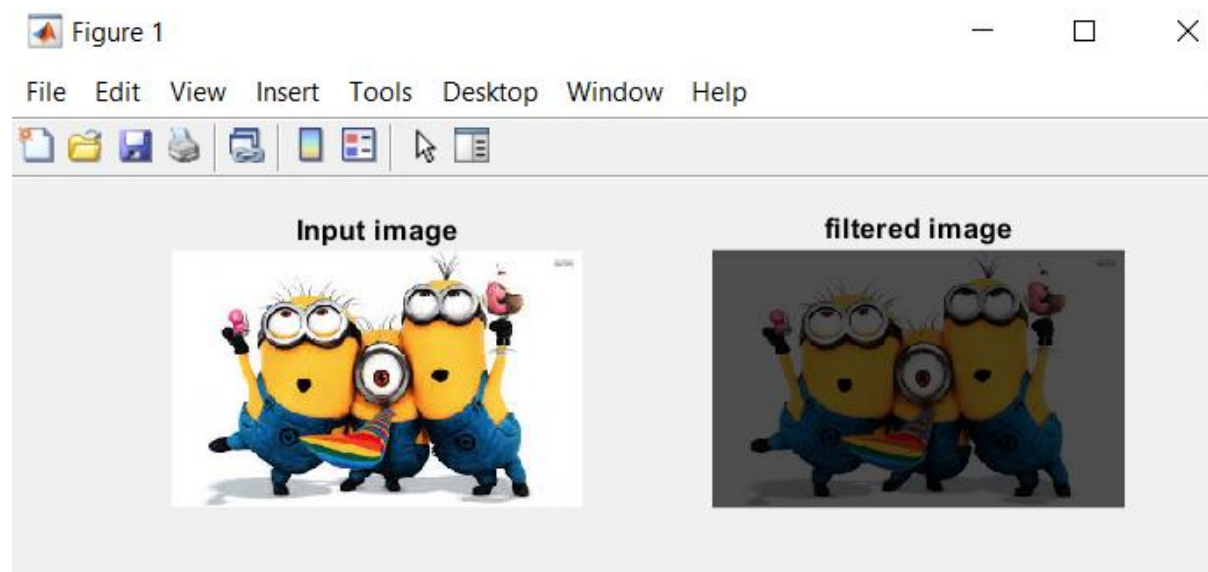
Code

```

1 - n=1;
2 - D0=50; % change the name for d0, d is usually the  $(u^2+v^2)^{1/2}$ 
3 - A=1.5; % normally the amplitude is 1
4 - im=imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg');
5 - [M,N]=size(im); % is easy to get the h and w like this
6 - F=fft2(double(im));
7
8 - for u=0:(M-1)
9 -     for v=0:(N-1)
10
11 -         idx=find(u>M/2);
12 -         u(idx)=u(idx)-M;
13 -         idy=find(v>N/2);
14 -         v(idy)=v(idy)-N;
15 -         [V,U]=meshgrid(v,u);
16 -         D=sqrt(U.^2+V.^2);
17 -         H =A * (1./(1 + (D0./D).^ (2*n)));
18
19 -     end
20 - end
21 - G=H.*F;
22 - g=real(ifft2(double(G)));
23 - subplot(1,2,1); imshow(im); title('Input image');
24 - subplot(1,2,2); imshow(g,[ ]); title('filtered image');
25

```

Output



PRACTICAL 5

Aim: Image noise removal and inverse filtering of images

Algorithm:

1. Read the image.
2. Add different types of noise using 'imnoise' command.
3. Remove the noise using appropriate filters.
 - a. **For salt and pepper noise** use median filter
 - b. **For Gaussian noise** use Weiner filter.
 - c. **For Poisson noise** use median filter.

Screenshots:

Salt and pepper Noise

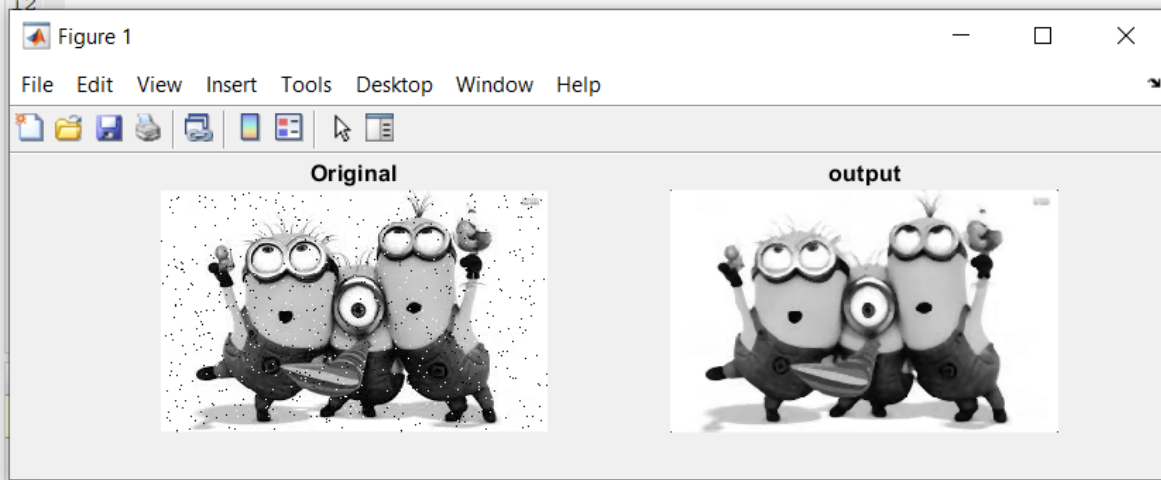
Salt-and-pepper noise is a form of noise sometimes seen on images. It is also known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels. An effective noise reduction method for this type of noise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

Code and Output

```

1 - clear all;
2 - clc;
3
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
5
6 - I = double(im);
7 - J = imnoise(im, 'salt & pepper',0.02);
8 - k=medfilt2(im);
9
10 - subplot(1, 2, 1), imshow(J), title('Original');
11 - subplot(1, 2, 2), imshow(k), title('output');
12

```



Gaussian Noise

Principal sources of Gaussian noise in digital images arise during acquisition e.g. sensor noise caused by poor illumination and/or high temperature, and/or transmission e.g. electronic circuit noise. In digital image processing Gaussian noise can be reduced using a spatial filter, though when smoothing an image, an undesirable outcome may result in the blurring of fine-scaled image edges and details because they also correspond to blocked high frequencies. Conventional spatial filtering techniques for noise removal include: mean (convolution) filtering, median filtering and Gaussian smoothing.

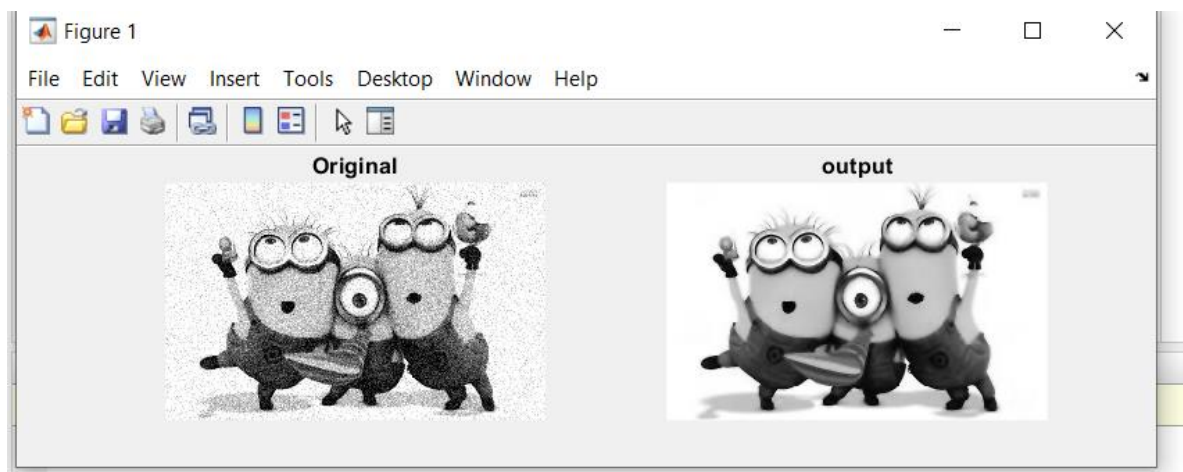
Code

```

negativeimage.m x experiment5.m x medianfilter.m x meanFilter.m x weiner.m x +
1 - clear all;
2 - clc;
3
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
5
6 - I = double(im);
7 - J = imnoise(im, 'gaussian',0.02);
8 - k=wiener2(im);
9
10 - subplot(1, 2, 1), imshow(J), title('Original');
11 - subplot(1, 2, 2), imshow(k), title('output');
12

```

Output



Poisson Noise

Shot noise or Poisson noise is a type of noise which can be modelled by a Poisson process. In electronics shot noise originates from the discrete nature of electric charge. Shot noise also occurs in photon counting in optical devices, where shot noise is associated with the particle nature of light.

Code

```
1 - clear all;  
2 - clc;  
3 -  
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));  
5 -  
6 - I = double(im);  
7 - J = imnoise(im, 'poisson');  
8 - k=medfilt2(im);  
9 -  
10 - subplot(1, 2, 1), imshow(J), title('Original');  
11 - subplot(1, 2, 2), imshow(k), title('output');  
12 -
```

Output



PRACTICAL 6

Aim: Point, Line and Edge detection.

Point Detection

Algorithm:

1. Read the image
2. Traverse to each pixel
 - a. Read the values of all the adjacent pixels
 - b. Evaluate whether the pixel is a point or not
 - c. Save the new pixel value in output image matrix
3. Display the output image

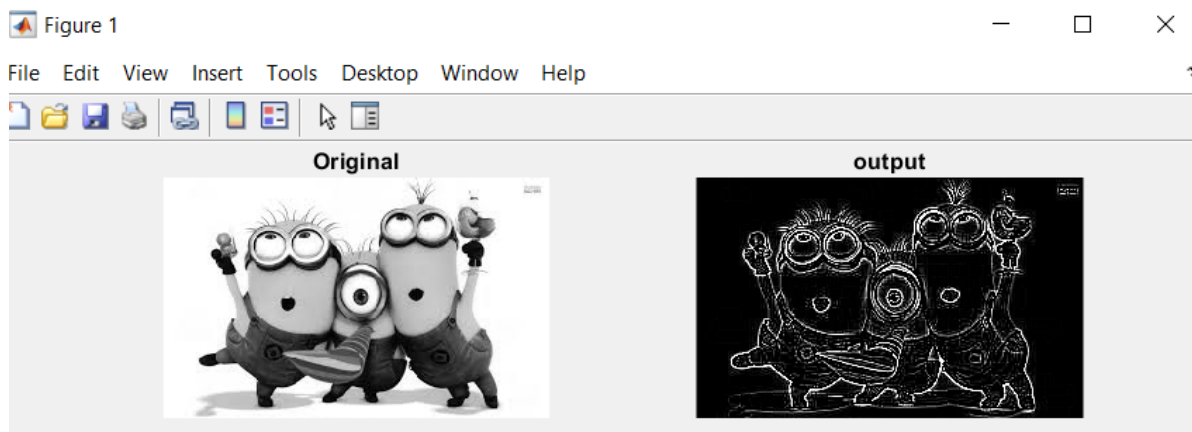
Screenshots

Code and Output

```

1 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
2 - I = double(im);
3 - [x,y] = size(I);
4 - for i = 2:x-1
5 -     for j = 2:y-1
6 -         sum = double(0);
7 -         for ii = i-1:i+1
8 -             for jj = j-1:j+1
9 -                 sum = sum - I(ii,jj);
10 -            end
11 -        end
12 -        I2(i,j) = sum + (9*I(i,j));
13 -    end
14 - end
15 - I2 = I2/255;
16 - subplot(1, 2, 1), imshow(im), title('Original');
17 - subplot(1, 2, 2), imshow(I2), title('output');

```



Line Detection

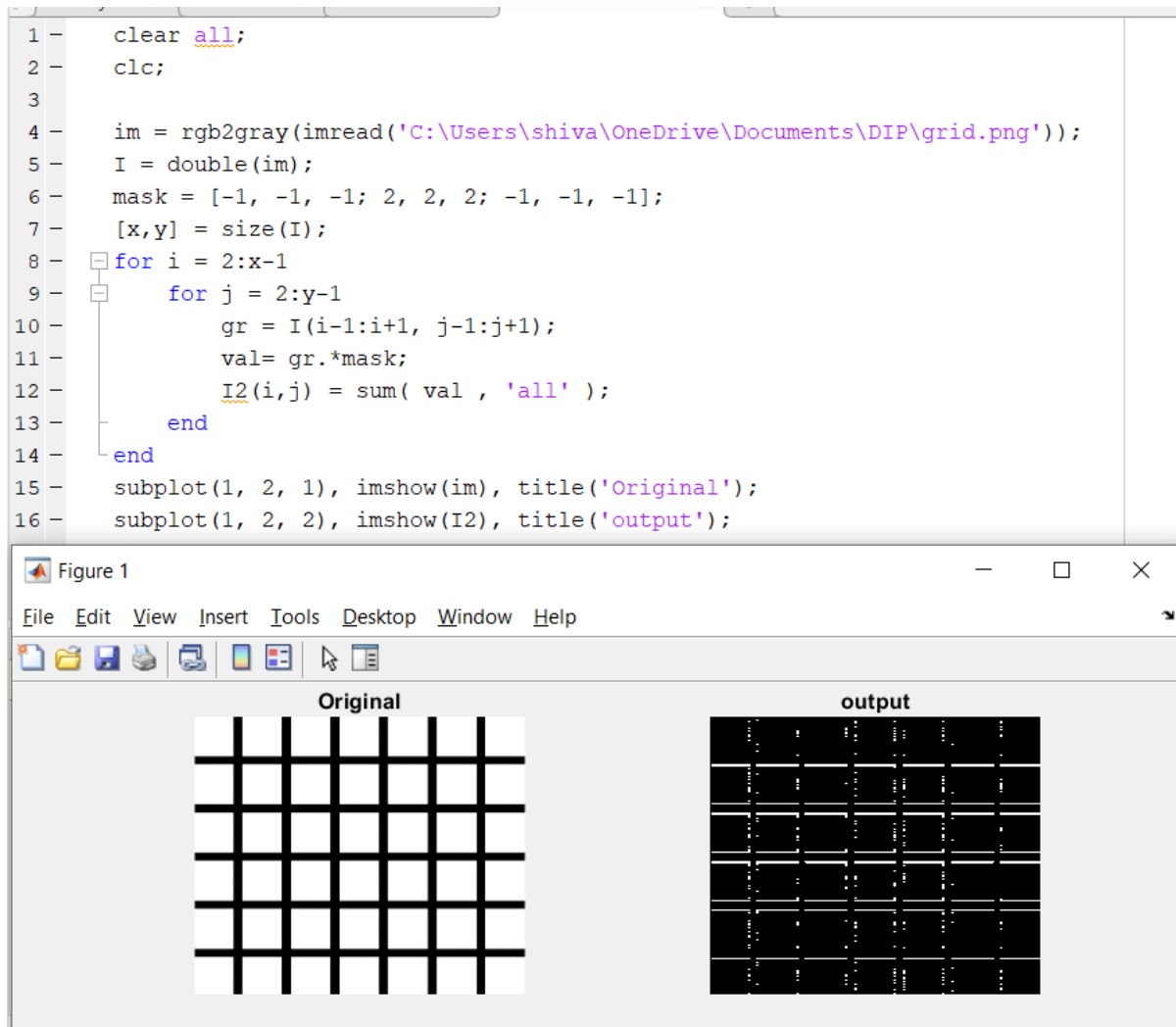
In a convolution-based technique, the line detector operator consists of a convolution masks tuned to detect the presence of lines of a particular width n and a θ orientation. Here are the two convolution masks to detect horizontal and vertical lines in an image.

a) Horizontal mask

-1	-1	-1
2	2	2
-1	-1	-1

Screenshots

Code and Output (Horizontal Line Detection)



(b) Vertical Line Detection

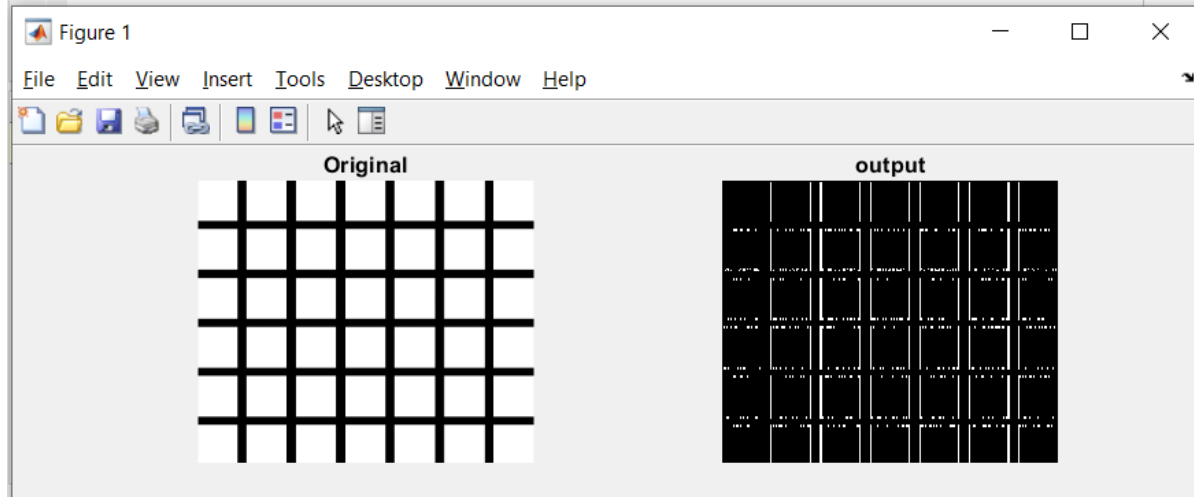
-1	2	-1
-1	2	-1
-1	2	-1

Code and Output

```

1 - clear all;
2 - clc;
3
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\grid.png'));
5 - I = double(im);
6 - mask = [-1, 2, -1; -1, 2, -1; -1, 2, -1];
7 - [x,y] = size(I);
8 - for i = 2:x-1
9 -     for j = 2:y-1
10 -         gr = I(i-1:i+1, j-1:j+1);
11 -         val= gr.*mask;
12 -         I2(i,j) = sum( val , 'all' );
13 -     end
14 - end
15 - subplot(1, 2, 1), imshow(im), title('Original');
16 - subplot(1, 2, 2), imshow(I2), title('output');

```



Edge Detection

Algorithm

1. Read the Image
2. Apply the edge detection functions
 - a. Canny edge detection
 - b. Prewitt edge detection
 - c. Sobel edge detection
3. Display the output image

Screenshots

Canny Edge Detector

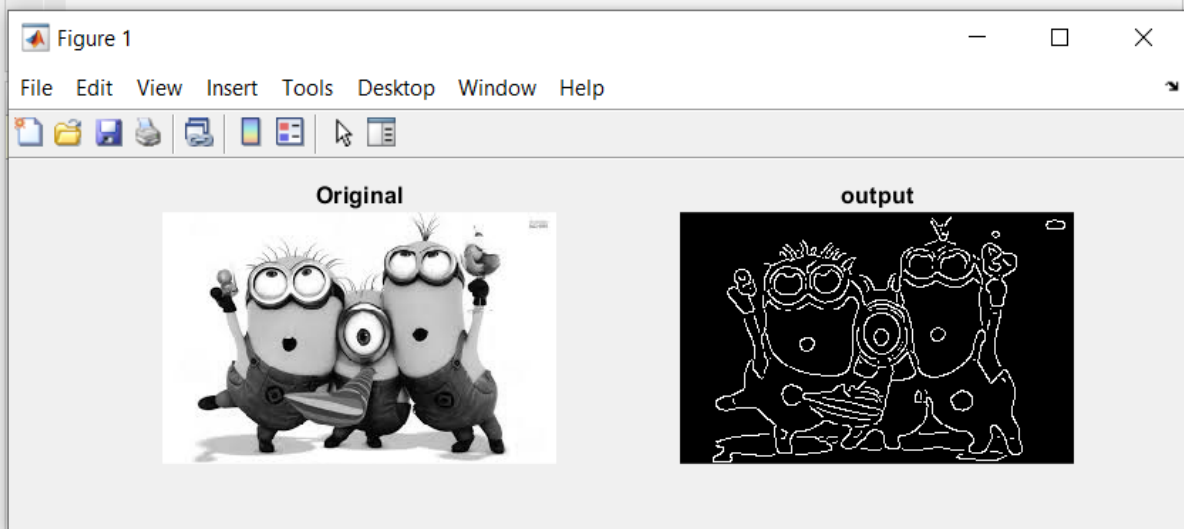
The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

Code and Output

```

1 - clear all;
2 - clc;
3
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
5 - I = double(im);
6 - outim = edge(I, 'Canny');
7
8 - subplot(1, 2, 1), imshow(im), title('Original');
9 - subplot(1, 2, 2), imshow(outim), title('output');
10

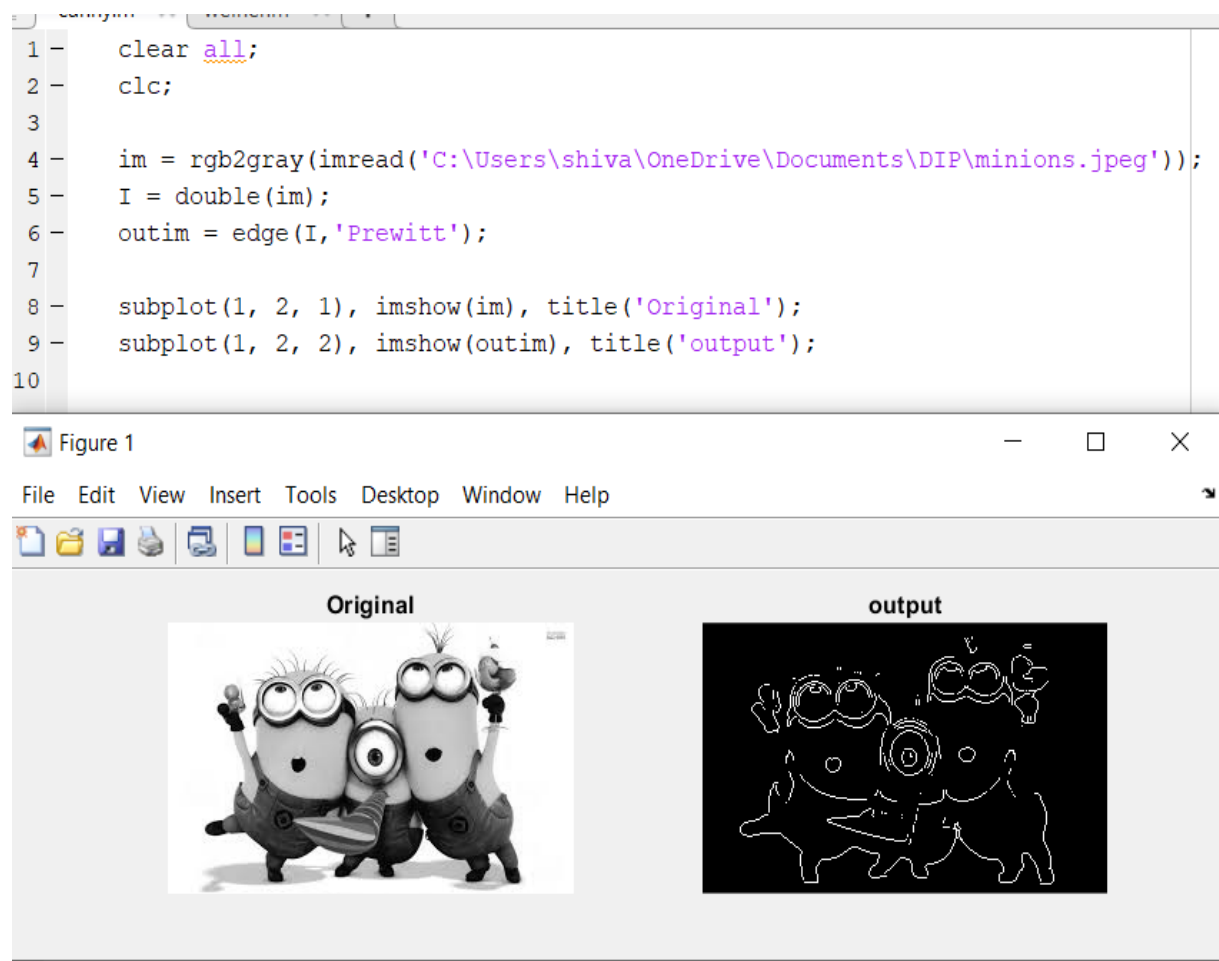
```



Prewitt Edge Detector

The **Prewitt operator** is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Prewitt operator is either the corresponding gradient vector or the norm of this vector. The Prewitt operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical directions and is therefore relatively inexpensive in terms of computations like Sobel and Kayyali operators. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image. The Prewitt operator was developed by Judith M. S. Prewitt.

Code and Output



Sobel Edge Detector

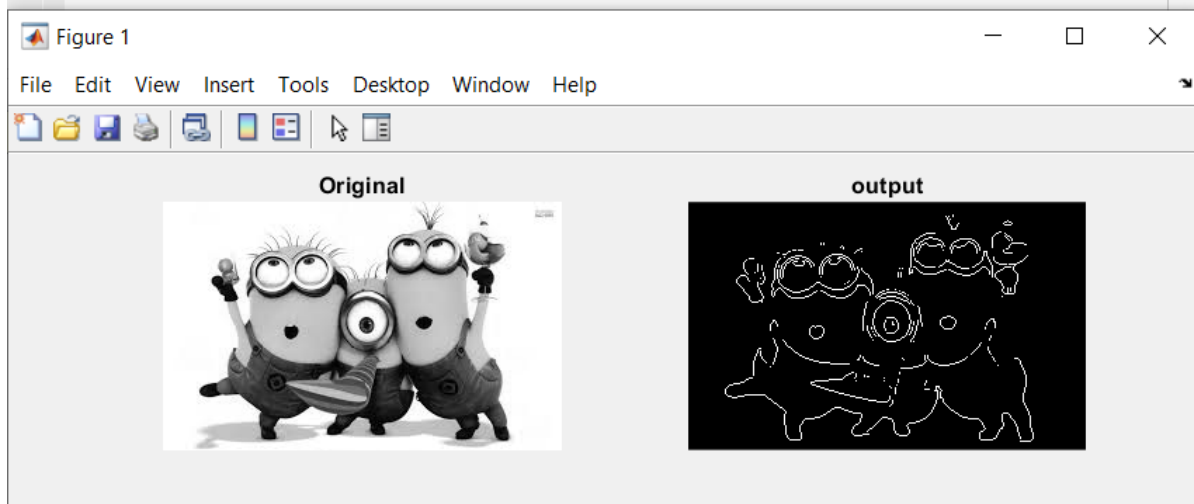
The **Sobel operator**, sometimes called the **Sobel–Feldman operator** or **Sobel filter**, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman, colleagues at the Stanford Artificial Intelligence Laboratory (SAIL). Sobel and Feldman presented the idea of an "Isotropic 3x3 Image Gradient Operator" at a talk at SAIL in 1968. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

Code and Output

```

1 - clear all;
2 - clc;
3
4 - im = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minions.jpeg'));
5 - I = double(im);
6 - outim = edge(I, 'Sobel');
7
8 - subplot(1, 2, 1), imshow(im), title('Original');
9 - subplot(1, 2, 2), imshow(outim), title('output');
10

```



PRACTICAL 7

Aim: Histogram matching

Algorithm:

1. Read the original image
2. Read the reference image
3. Match the histogram
4. Display matched image
5. Plot the histogram for the images

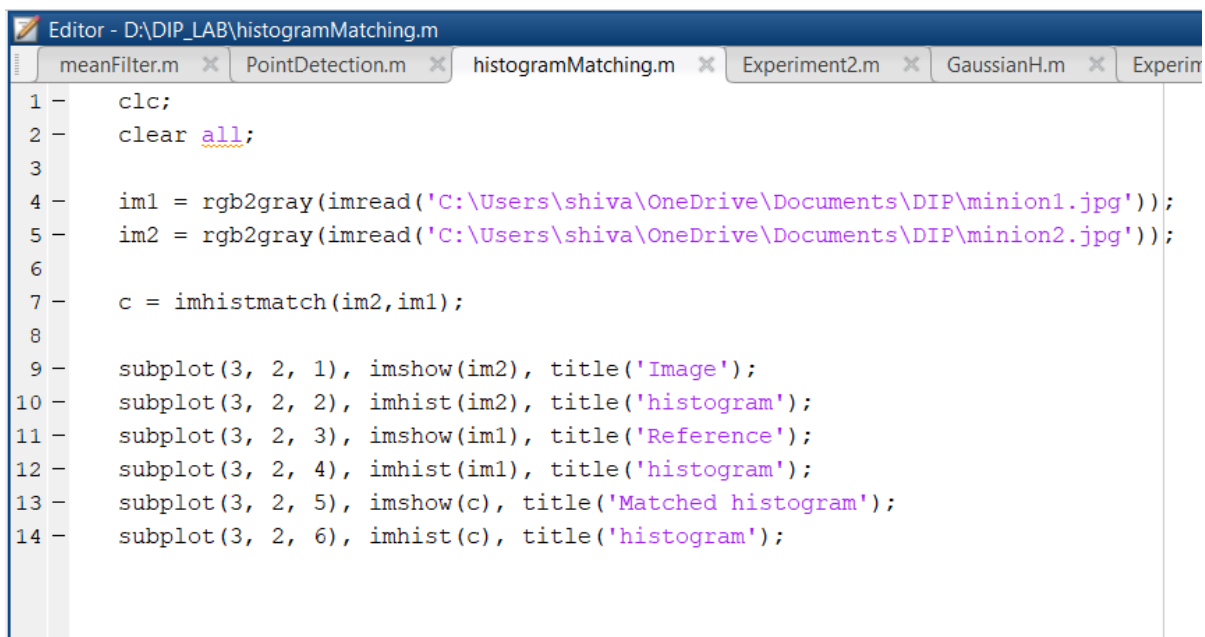
Histogram Matching

In image processing, histogram matching or histogram specification is the transformation of an image so that its histogram matches a specified histogram. The well-known histogram equalization method is a special case in which the specified histogram is uniformly distributed.

It is possible to use histogram matching to balance detector responses as a relative detector calibration technique. It can be used to normalize two images, when the images were acquired at the same local illumination (such as shadows) over the same location, but by different sensors, atmospheric conditions or global illumination

Screenshots

Code

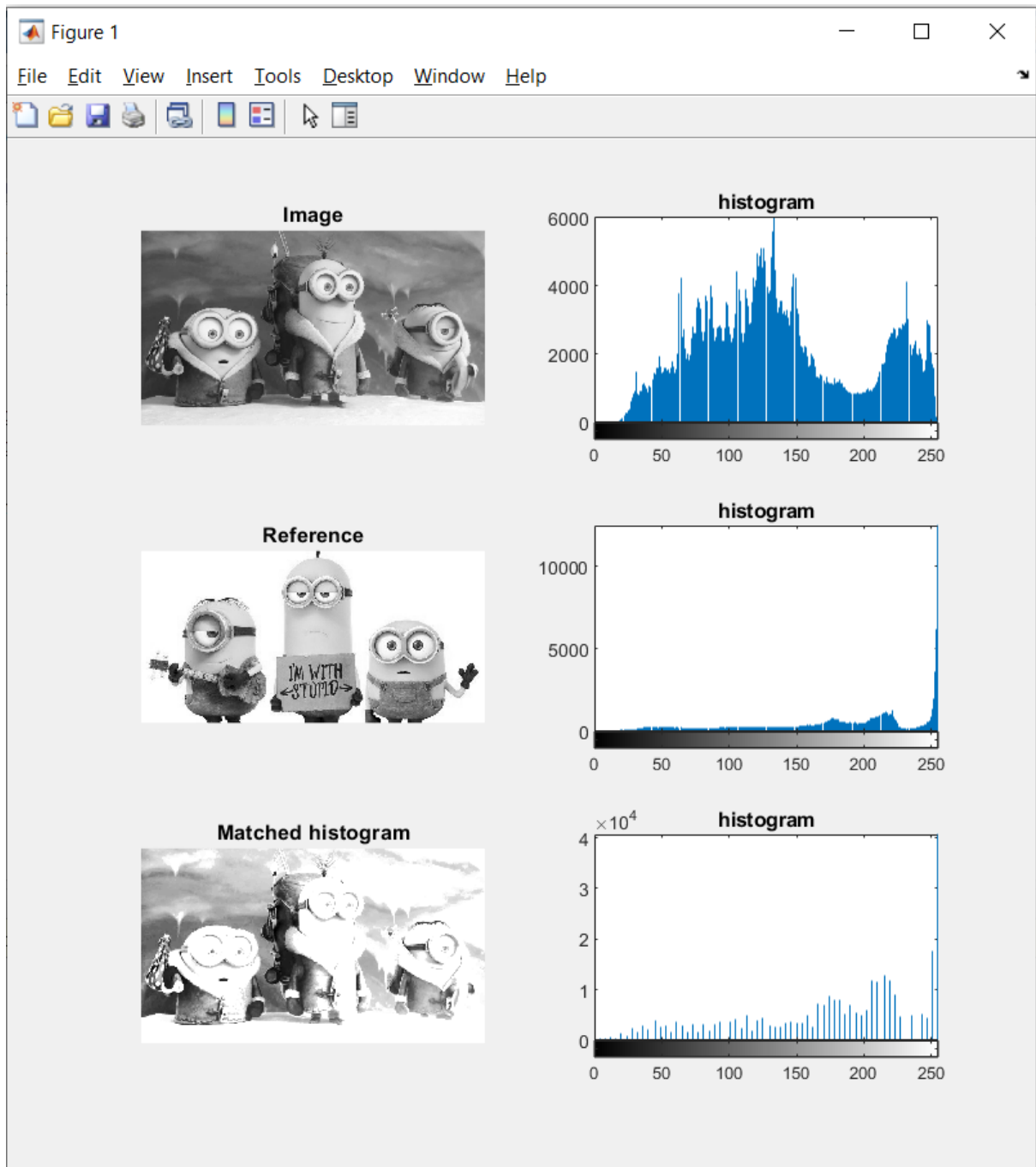


```

Editor - D:\DIP_LAB\histogramMatching.m
meanFilter.m  PointDetection.m  histogramMatching.m  Experiment2.m  GaussianH.m  Experim

1 -   clc;
2 -   clear all;
3
4 -   im1 = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minion1.jpg'));
5 -   im2 = rgb2gray(imread('C:\Users\shiva\OneDrive\Documents\DIP\minion2.jpg'));
6
7 -   c = imhistmatch(im2,im1);
8
9 -   subplot(3, 2, 1), imshow(im2), title('Image');
10 -  subplot(3, 2, 2), imhist(im2), title('histogram');
11 -  subplot(3, 2, 3), imshow(im1), title('Reference');
12 -  subplot(3, 2, 4), imhist(im1), title('histogram');
13 -  subplot(3, 2, 5), imshow(c), title('Matched histogram');
14 -  subplot(3, 2, 6), imhist(c), title('histogram');
  
```


Output



PRACTICAL 8

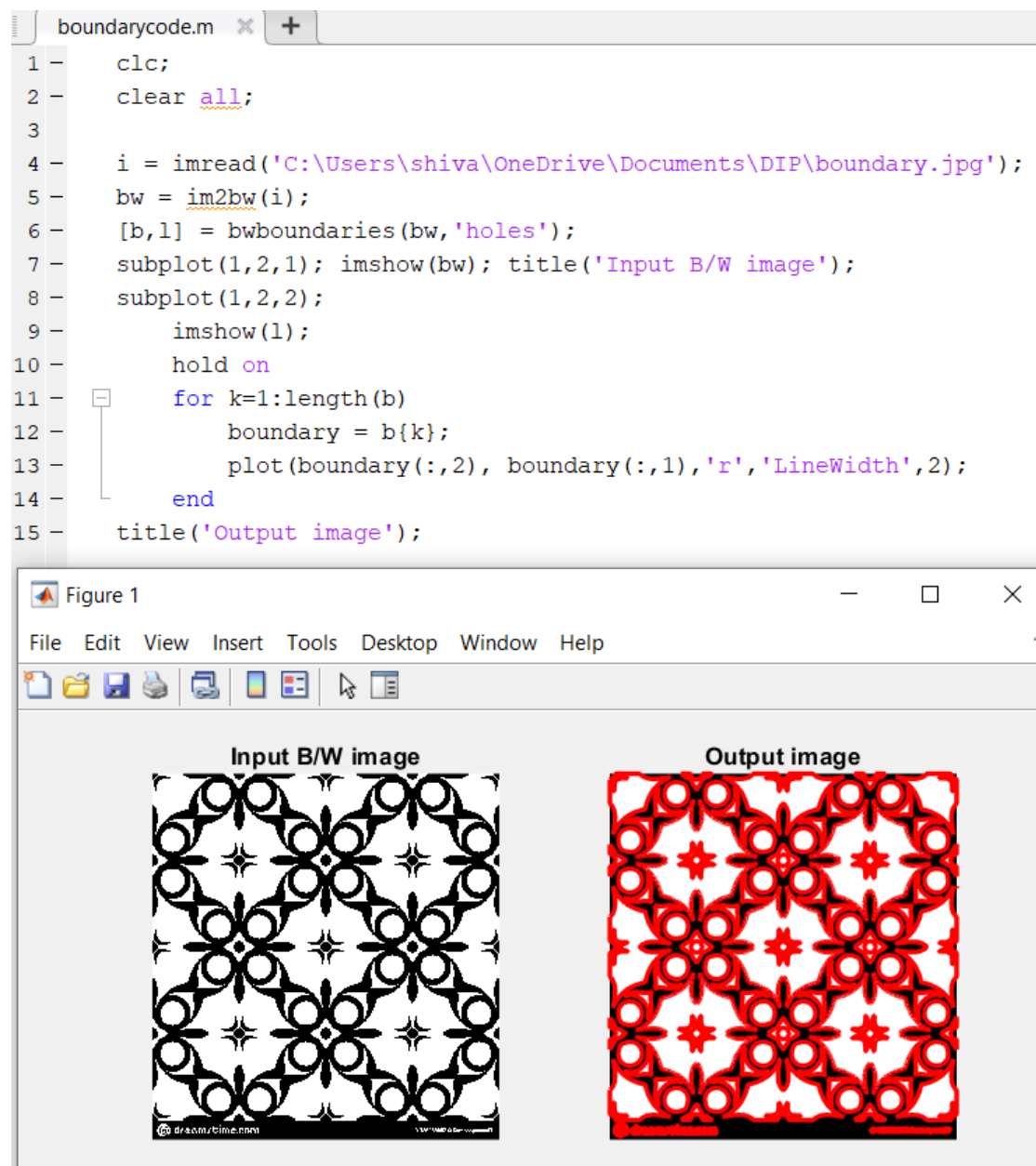
Aim: Boundary Linking, Representation and description techniques on images.

Algorithm

1. Read the image
2. Convert the image into black and white
3. Trace the exterior boundaries of the image, specifying the connectivity to use when tracing parent and child boundaries as 'holes'.
4. Overlay the boundaries on the image.
5. Display the original image and the image with the boundaries overlaid.

Screenshots

Code and Output (Boundary Linking)



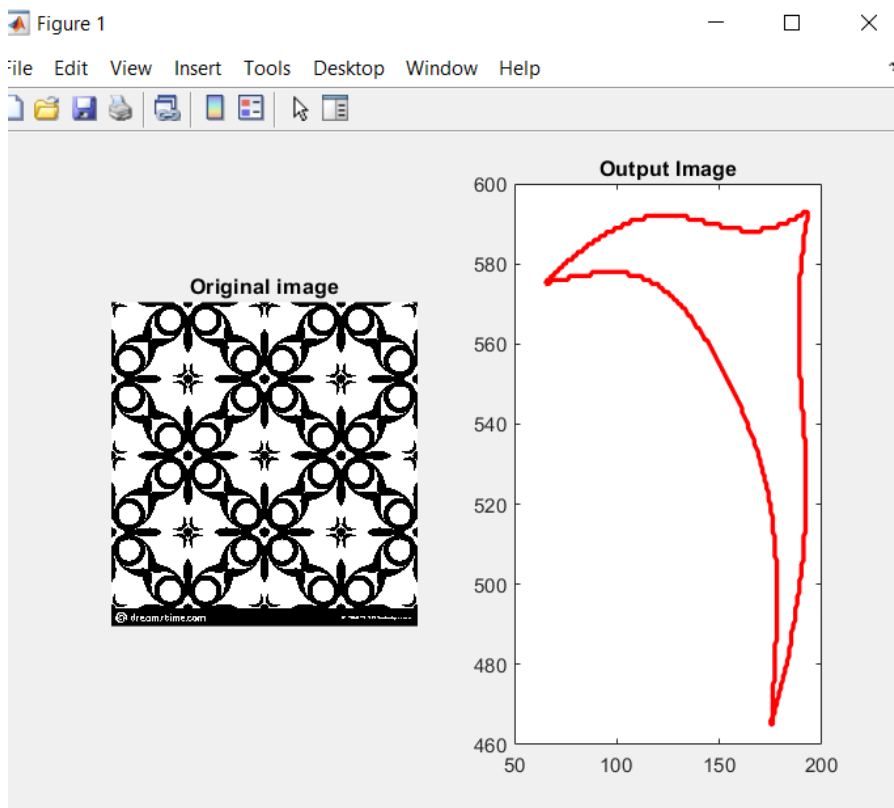
Code (Representation)

```

chainCode.m  x  boundarycode2.m  x  +
1 -   clc;
2 -   clear all;
3 -   I = imread('C:\Users\shiva\OneDrive\Documents\DIP\boundary.jpg');
4 -   BW = im2bw(I);
5 -   [B,L] = bwboundaries(BW,'noholes');
6 -   subplot(1,2,1); imshow(BW); title('Original image');
7 -   subplot(1,2,2);
8 -   b=B{12};
9 -   plot( b(:,2), b(:,1), 'r', 'LineWidth', 2);
10 -  title('Output Image');
11 -  sb = circshift(b, [-1,0]);
12 -  delta = sb-b;
13 -  if abs(delta(end,1))>1||abs(delta(end,2))>1
14 -      delta = delta(1:(end-1),:);
15 -  end
16 -  n8c = find(abs(delta(:,1))>1 | abs(delta(:,2))>1);
17 -  if size(n8c,1)>0
18 -      s = '';
19 -      for i=1:size(n8c,1)
20 -          s=[s sprintf(' idx -> %d \n', n8c(i))];
21 -      end
22 -      error('Curve isn''t 8-connencted in elements: \n %s',s);
23 -  end
24 -  idx=3*delta(:,1) + delta(:,2)+5;
25 -  cm([1 2 3 4 5 6 7 8 9])=[5 6 7 4 0 3 2 1];
26 -  cc.x0=b(1,2);
27 -  cc.y0=b(1,1);
28 -  cc.code=(cm(idx));
29 -  disp(cc.code);

```

Output



Algorithm (Boundary Description)

1. Read the binary image into workspace.
2. Calculate centroids for connected components in the image using regionprops.
3. Concatenate structure array containing centroids into a single matrix.
4. Display original image and image with centroid locations superimposed.

Code

```

description.m
1 -   clc;
2 -   clear all;
3
4 -   i = imread('C:\Users\shiva\OneDrive\Documents\DIP\boundary.jpg');
5 -   s = regionprops(i, 'centroid');
6 -   centroids = cat(1, s.Centroid);
7 -   subplot(1,2,1), imshow(i), title('Original Image');
8 -   subplot(1,2,2), imshow(i), title('Centroids Image');
9 -   hold on
10 -  plot(centroids(:,1), centroids(:,2), 'r*')
11 -  hold off
12

```

Output

