

ABSTRACT

Various nonlinear, fixed-neighbourhood techniques based on local statistics have been proposed in the literature for filtering noise in colour images. We present adaptive vector filtering (AVF) techniques for noise removal in colour images. The main idea is to find for each pixel (called the “seed” when being processed) a variable-shaped, variable-sized neighbourhood that contains only pixels that are similar to the seed. Then, statistics computed within the adaptive neighbourhood are used to derive the filter output. Results of the AVF techniques are compared with those given by a few multivariate fixed-neighbourhood filters: the double-window modified trimmed-mean filter, the generalized vector directional filter – double-window – α -trimmed mean filter, the adaptive hybrid multivariate filter, and the adaptive nonparametric filter with Gaussian kernel. It is shown that the AVF techniques provide better visual results, effectively suppressing noise while not blurring edges; the results are also better in terms of objective measures (such as normalized mean-squared error and normalized colour difference) than the results of the other methods.

CONTENTS

ABSTRACT	iv
LIST OF FIGURES	v
LIST OF TABLES	vi
CHAPTER 1:	PAGE.NO
Introduction to MATLAB	1
1.1 MATLAB	1
1.2 The Matlab Environment	2
1.3 Features of Matlab	3
1.4 Applications of Matlab	4
CHAPTER 2:	5
Introduction to image processing	5
2.1 Image processing	5
2.2 purpose of image processing	6
2.3 operation of image processing	6
2.4 Image in Matlab	6
2.5 Data types in Matlab	7
2.6 Image display function	7
2.7 Image conversion function	7
2.8 File I/O MATLAB	8
CHAPTER 3:	9
Introduction to GUI	9
3.1 GUI	9
3.2 Create a UI using GUIDE	10
3.3 Layout the simple GUIDE UI	11
3.4 Align the components	11
CHAPTER 4:	12
Introduction to project	12
4.1 Project propose	13
4.2 Hardware and Software required	14
CHAPTER 5:	15
Introduction to Image restoration	15
5.1 Image restoration	15
5.2 Gaussian filter	15
5.3 Multi Thresholding	16
5.4 Pixel classification	17

5.5 Histogram	18
5.6 Process	19
5.7 Binary mask	20
CHAPTER 6:	21
Introduction to Image recognition	21
6.1 Image recognition	21
6.2 Colour detection (flow chart)	22
6.3 Shape detection (flow chart)	23
6.4 Process	24
CHAPTER 7:	25
Advantages and application	25
CHAPTER 9:	26
Conclusion and References	26
CHAPTER 10:	27
Outputs	28

APPENDICES

A. LIST OF FIGURES	PAGENO
Figure.1.1MATLAB Environment(window)	3
Figure.2.1Reading image in Matlab	8
Figure.3.1Matlab (GUI window)	9
Figure.3.2Matlab(GUI Operation window)	10
Figure.5.1Multithresholding	15
Figure.5.2Histogram(Plot)	18
Figure.5.3Colorbasedsegmentation	20
Figure.6.1Colordetection	24
Figure.6.2Shapedetection	24
Figure.9.1Preprocessing output	27
Figure.9.2Multithresholding output	27
Figure.9.3RGB color based of image with RGB mask	28
Figure.9.4Distribution of the areas BLOBS	28
Figure.9.5Histogram to size of the picture to occupy screen	29
Figure.9.6Binary masking filling small holes in image	29
Figure.9.7Masked original image	30
Figure.9.8Masked original image with area coverage in (%)	30
Figure.9.9Resorted image (Final)	31

B. LIST OF TABLES

Table.5.1RGB colors with decimal code	1
---------------------------------------	---

CHAPTER 1

INTRODUCTION TO MATLAB

1.1 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.

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 fraction of the time it would take to write a program in a scalar noninteractive 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 LINPAC and EISPACK projects. Today, Matlab engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation. MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, Matlab is the tool of choice for high-productivity research, development, and analysis.

Matlab features a family of add-on application-specific solutions called toolboxes. Very important to most users of Matlab, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of Matlab functions (M-files) that extend the Matlab environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

1.2 THE MATLAB ENVIRONMENT

When you start Matlab, a special window called the Matlab desktop appears. The desktop is a window that contains other windows. The major tools within or accessible from the desktop are

- **The Command Window**

Matlab is an interactive program for numerical computation and data visualization. You can enter a command by typing it at the Matlab prompt '>>' on the Command Window.

- **The Command History**

It displays a log of statements that you ran in the current and previous Matlab sessions. These statements include those you run using the Evaluate Selection item on context menus in tools such as the Editor, Command History window, and Help browser.

- **The Workspace**

It displays the Workspace browser, a graphical user interface that allows you to view and manage the contents of the Matlab workspace. It provides a graphical representation of the display, and allows you to perform the equivalent of the clear, load, open, and save functions.

- **The Current Directory**

The Current Folder browser enables you to interactively manage files and folders in Matlab. Use the Current Folder browser to view, create, open, move, and rename files and folders in the current folder.

- **The Help Browser**

Matlab automatically installs the documentation and demos for a product when you install that product. The Help browser is an html browser integrated with the Matlab desktop. To open the Help browser, click the Help button in the desktop toolbar, type help browser in the Command Window, or use the Help menu in any tool.

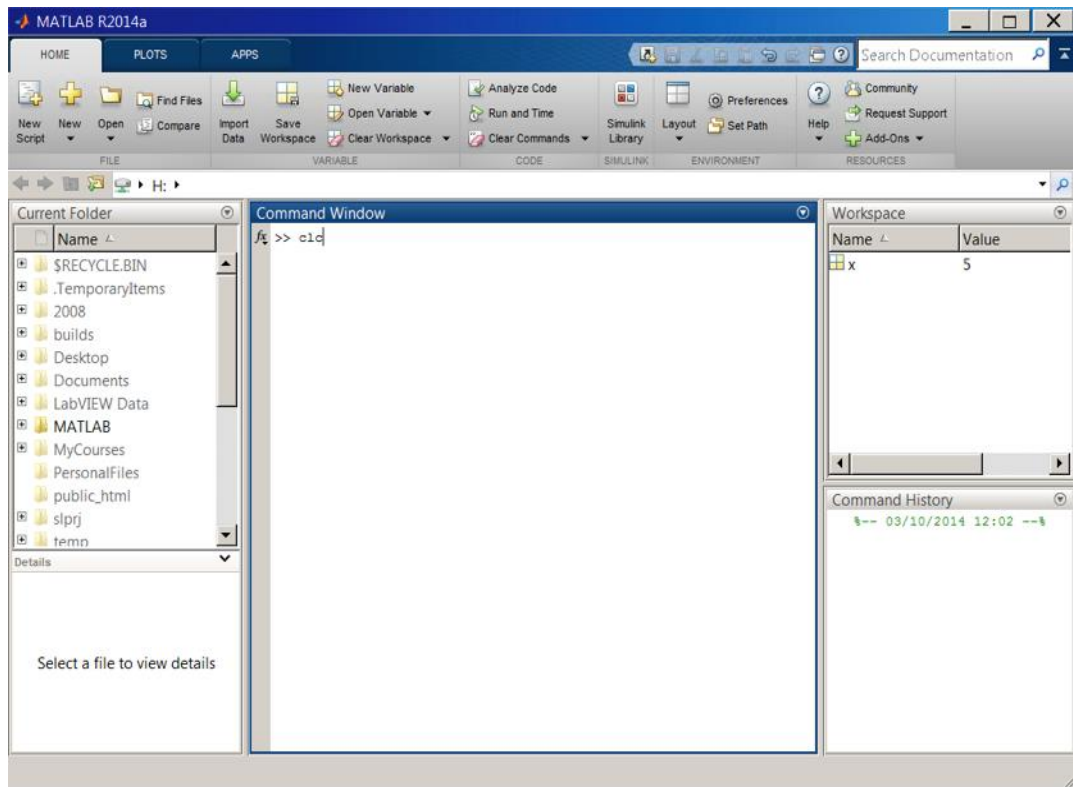


Fig.1.1 MATLAB Environment(window)

1.3FEATURES OF MATLAB

- It is a high-level language for numerical computation, visualization and application development.
- It also provides an interactive environment for iterative exploration, design and problem solving.
- It provides vast library of mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration and solving ordinary differential equations.
- It provides built-in graphics for visualizing data and tools for creating custom plots.
- MATLAB's programming interface gives development tools for improving code quality, maintainability, and maximizing performance.
- It provides tools for building applications with custom graphical interfaces.

- It provides functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET and Microsoft Excel.

1.4 APPLICATION OF MATLAB

Matlab is widely used as a computational tool in science and engineering encompassing the fields of physics, chemistry, math and all engineering streams. It is used in a range of applications including

- Image Acquisition Toolbox
- Image Processing Toolbox
- Signal Processing Toolbox
- DSP System Toolbox
- Antenna Toolbox
- Audio System Toolbox
- Automated Driving System Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- Database Toolbox
- Robotics System Toolbox

CHAPTER 2

IMAGE PROCESSING TOOLBOX

2.1 IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps

- Importing the image via image acquisition tools
- Analysing and manipulating the image
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

In this lecture we will talk about a few fundamental definitions such as image, digital image, and digital image processing. Different sources of digital images will be discussed and examples for each source will be provided. The continuum from image processing to computer vision will be covered in this lecture. Finally, we will talk about image acquisition and different types of image sensors.

2.2 PURPOSE OF IMAGE PROCESSING

The purpose of image processing is divided into 5 groups. They are

- **Visualization** - Observe the objects that are not visible.
- **Image sharpening and restoration** - To create a better image

- **Image retrieval** - Seek for the image of interest.
- **Measurement of pattern** – Measures various objects in an image.
- **Image Recognition** – Distinguish the objects in an image.

2.3 OPERATIONS OF IMAGE PROCESSING

The Image Processing Toolbox is a collection of functions that extend the capabilities of the MATLAB's numeric computing environment. The toolbox supports a wide range of image processing operations, including:

- Geometric operations
- Neighbourhood and block operations
- Linear filtering and filter design
- Transforms
- Image analysis and enhancement
- Binary image operations
- Region of interest operations

2.4 IMAGES IN MATLAB

MATLAB can import/export several image formats

- BMP (Microsoft Windows Bitmap)
- GIF (Graphics Interchange Files)
- HDF (Hierarchical Data Format)
- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- XWD (X Window Dump)
- raw-data and other types of image data

2.5 DATA TYPES IN MATLAB

- Double (64-bit double-precision floating point)
- Single (32-bit single-precision floating point)
- Int32 (32-bit signed integer)
- Int16 (16-bit signed integer)
- Int8 (8-bit signed integer)
- Uint32 (32-bit unsigned integer)

2.6 IMAGE DISPLAY FUNCTIONS

- Imread – to read an image
- image - create and display image object
- imagesc - scale and display as image
- imshow - display image
- colourbar - display colour bar
- getimage - get image data from axes
- truesize - adjust display size of image
- zoom - zoom in and zoom out of 2D plot

2.7 IMAGE CONVERSION FUNCTIONS

- gray2ind - intensity image to index image
- im2bw - image to binary
- im2uint8 - image to 8-bit unsigned integers
- im2uint16 - image to 16-bit unsigned integers
- ind2gray - indexed image to intensity image
- mat2gray - matrix to intensity image
- rgb2gray - RGB image to grayscale
- rgb2ind - RGB image to indexed image.

2.8 File I/O MATLAB

It allows you to save matrices and read them in later. The simplest way to do this is using the commands “save” and “load”. Typing in “save A” saves matrix A to a file called A.mat. If you want to read in matrix A later, just type “load A”. You can also use the load command to read in ASCII files, as long as they are formatted correctly. Formatted correctly means that the number of columns in each line is the same and the columns are delimited with a space.

- fopen – open a file for reading or writing
- fread – read a binary file
- fscanf – read an ASCII file
- fwrite – write a binary file
- fprintf – write an ASCII file
- fclose – close a file (when you are done with it)

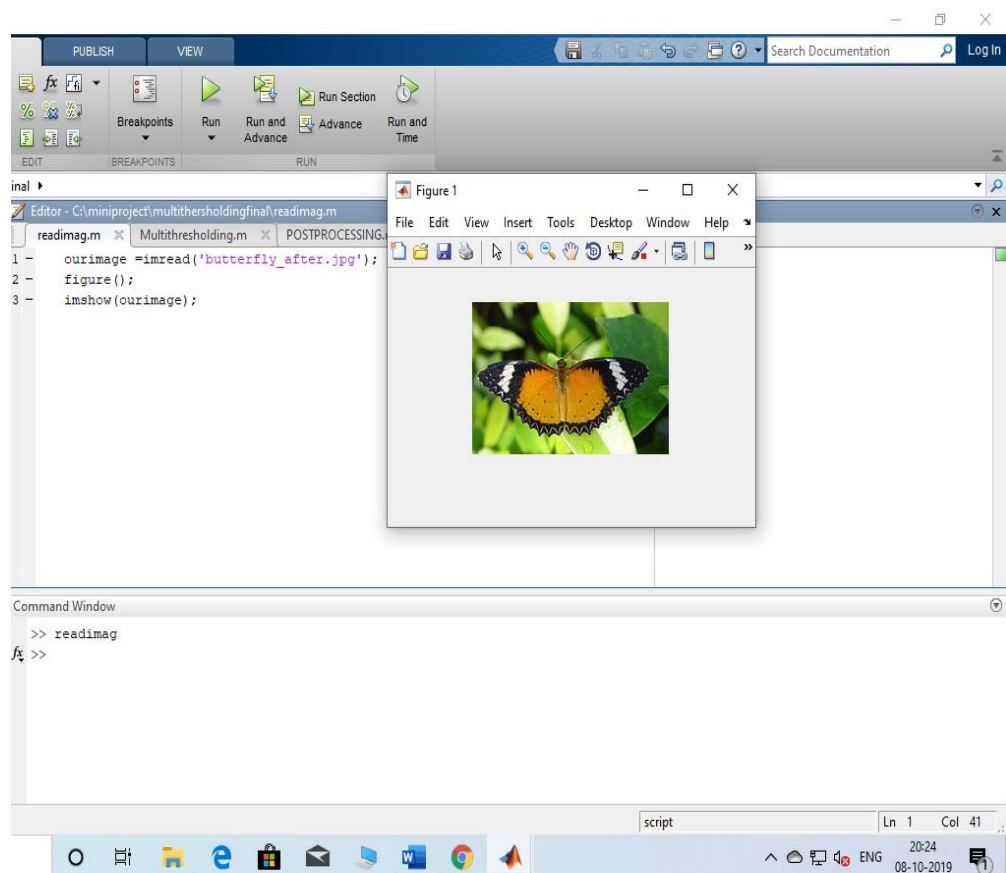


Fig.2.1 Reading image in matlab

CHAPTER 3

INTRODUCTION TO GUI

3.1 GUI (Graphical User Interface)

A user interface (UI) is a graphical display in one or more windows containing controls, called components, that enable a user to perform interactive tasks. The user does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user does not need to understand the details of how the tasks are performed. UI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders just to name a few. UIs created using matlab tools can also perform any type of computation, read and write data files, communicate with other UIs, and display data as tables or as plots. The following figure illustrates a simple UI that you can easily build yourself.

The UI contains these components:

- An axes component.
- A pop-up menu listing three data sets that correspond to MATLAB functions.
- A static text component to label the pop-up menu.

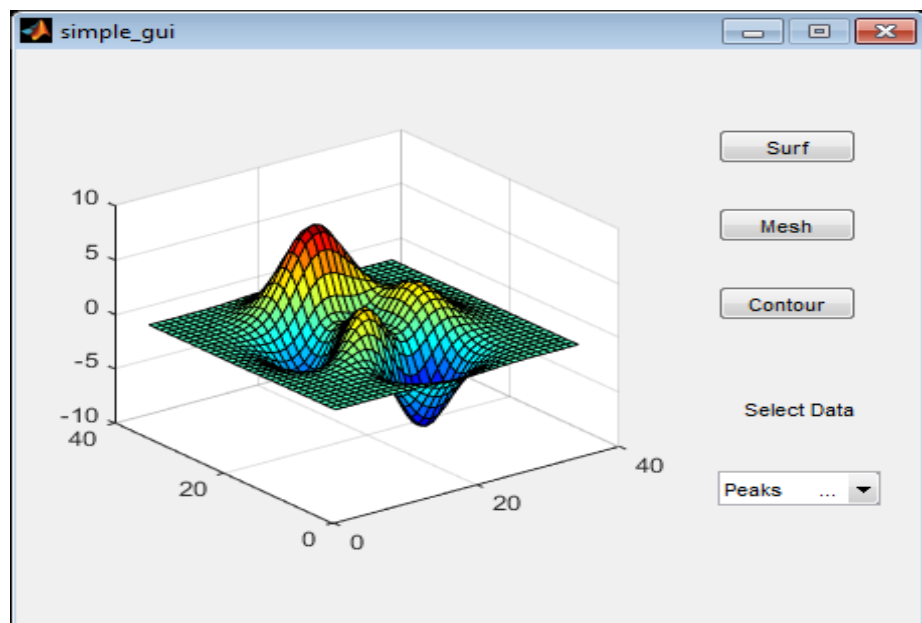


Fig.3.1 MATLAB (GUI window)

3.2 Create a UI Using GUIDE

This example shows how to use GUIDE to create a simple user interface (UI), such as shown in the following figure. Subsequent topics guide you through the process of creating this UI. To run the UI, on the Editor tab, in the Run section, click Run.

Open a New UI in the GUIDE Layout Editor

- Start GUIDE by typing `guide` at the MATLAB prompt.
- In the GUIDE Quick Start dialog box, select the Blank GUI (Default) template, and then click OK.
- 3 Display the names of the UI components in the component palette:
 1. Select FILE > PREFERENCES > GUIDE.
 2. Select Show names in component palette.
 3. Click OK.

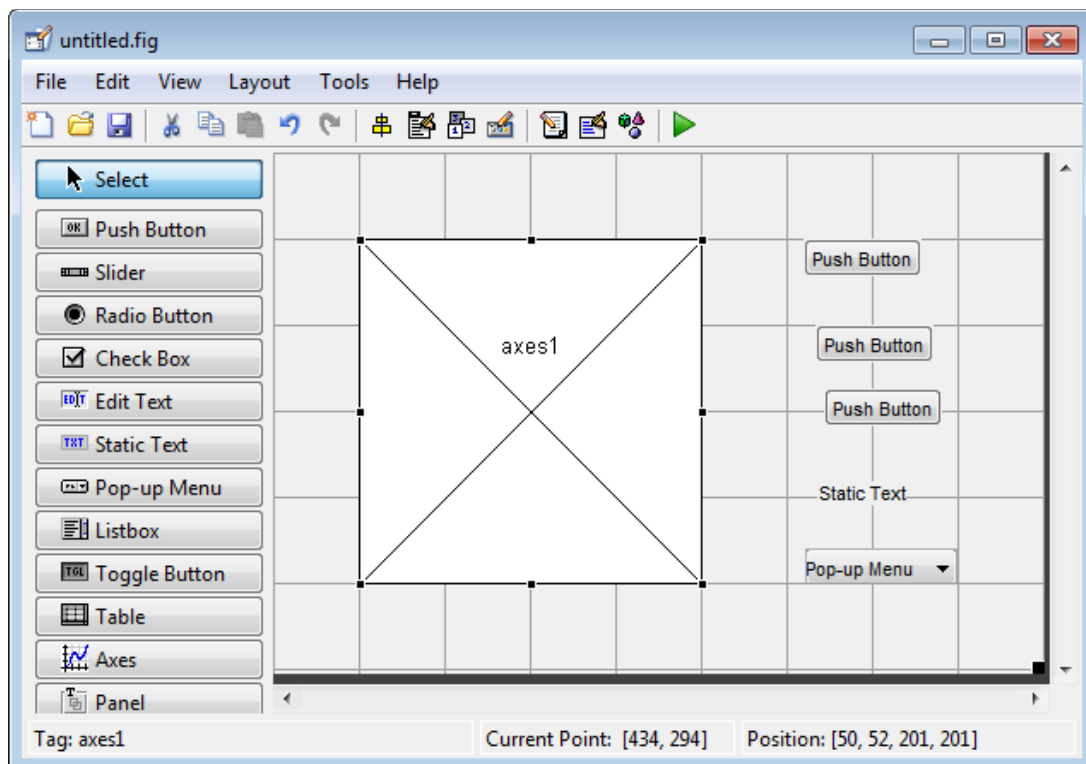


Fig .3.2 MATLAB (GUIOperation window)

3.3 Layout the Simple GUIDE UI

Add, align, and label the components in the UI.

1 Add the three push buttons to the UI. Select the push button tool from the component palette at the left side of the Layout Editor and drag it into the layout area. Create three buttons, positioning them approximately as shown in the following figure.

2 Add the remaining components to the UI.

- A static text area
- A pop-up menu
- An axis

Arrange the components as shown in the following figure. Resize the axes component to approximately 2-by-2 inches.

3.4 Align the Components

If several components have the same parent, you can use the Alignment Tool to align them to one another. To align the three push buttons:

1. Select all three push buttons by pressing Ctrl and clicking them.
2. Select Tools > Align Objects.
3. Make these settings in the Alignment Tool:
 - Left-aligned in the horizontal direction.
 - 20 pixels spacing between push buttons in the vertical direction.
4. Click OK.

CHAPTER 4

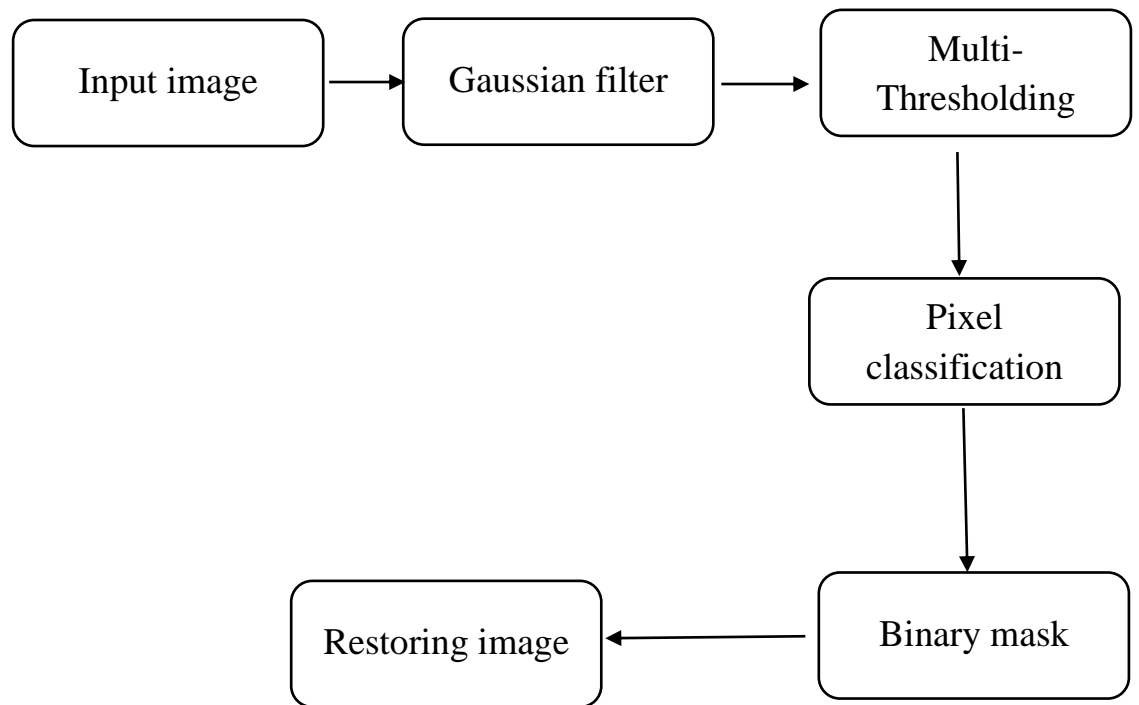
NATURAL COLOUR IMAGE RESTORATION AND IMAGE RECOGNITION

4.1 PROJECT PROPOSE

A robust structure-adaptive hybrid vector filter is proposed for digital colour image restoration in this paper. At each pixel location, the image vector (i.e., pixel) is first classified into several different signal activity categories by applying modified quad tree decomposition to luminance component (image) of the input colour image. A weight-adaptive vector filtering operation with an optimal window is then activated to achieve the best trade-off between noise suppression and detail preservation. Through extensive simulation experiments conducted using a wide range of test colour images, the filter has demonstrated superior performance to that of a number of well-known benchmark techniques, in terms of both standard objective measurements and perceived image quality, in suppressing several distinct types of noise commonly considered in colour image restoration, including Gaussian noise, impulse noise, and mixed noise.

A multichannel (vector-valued, multicomponent, or multispectral) image is characterized at each point (pixel) by a vector of relative spectral intensities. In the classical case of colour images, each pixel is a three-component vector, with the components being the relative amounts of red, green, and blue (RGB) that compose the local colour. Other representations of colour include the HSV (hue, saturation, value) and the CIE standard representations. For this reason, separate processing of the component images using common techniques meant for grey-scale images is not appropriate. In most colour image filtering algorithms, the observed noise is modelled as additive, white Gaussian noise that is independent of the signal. However, impulsive noise, modelled as sparse “spikes” that appear in the images, is likely to corrupt colour images. Hence, in many papers in the literature, the images are assumed to be corrupted by a combination of these two types of noise. Although simple to implement, this type of filtering seems to have lost its appeal, mainly due to its global and nonadaptive nature. Various nonlinear filtering techniques have been proposed to process colour images

BLOCK DIAGRAM



HARDWARE AND SOFTWARE REQUIRED

HAREWARE REQUIRED

- Computer or laptop
- with minimum of 4GB RAM
- Approximately with 20-30GB hard disk space
- GPU capability 3.0 or higher

SOFTEWRE REQUIRED

- Operating software (windows 7,8,10)
- MATLAB software

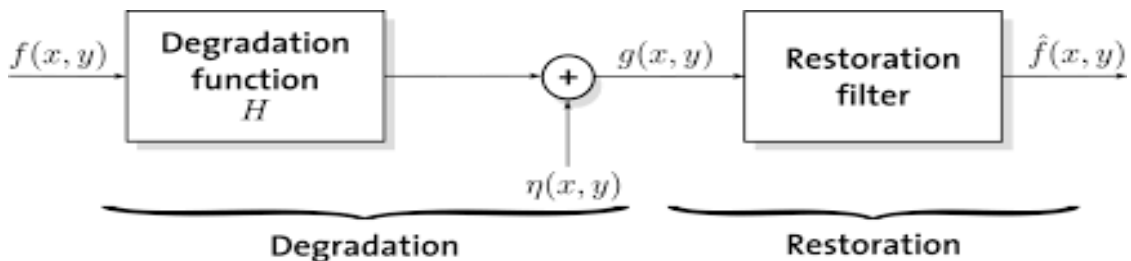
CHAPTER 5

INTRODUCTION TO IMAGE RESTORATION

5.1 IMAGE RESTORATION

Image Restoration is the operation of taking a corrupt/noisy image and estimating the clean, original image. Corruption may come in many forms such as motion blur, noise and camera mis-focus. Image restoration is performed by reversing the process that blurred the image and such is performed by imaging a point source and use the point source image, which is called the Point Spread Function (PSF) to restore the image information lost to the blurring process.

Image restoration is different from image enhancement in that the latter is designed to emphasize features of the image that make the image more pleasing to the observer, but not necessarily to produce realistic data from a scientific point of view. Image enhancement techniques (like contrast stretching or de-blurring by a nearest neighbor procedure) provided by imaging packages use no *a priori* model of the process that created the image.



5.2 GAUSSIAN FILTER

It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. Since the Fourier transform of a Gaussian is another Gaussian, applying a Gaussian blur has the effect of reducing the image's high-frequency components; a Gaussian blur is thus a low pass filter. This is to make sure no energy is added or removed from the image after the operation. Specifically, a Gaussian kernel (used for Gaussian blur) is a square array of pixels where the pixel values correspond to the values of a Gaussian curve (in 2D).

5.3 MULTI THERSHODLING

Multilevel thresholding is a process that segments a grey level image into several distinct regions. This technique determines more than one threshold for the given image and segments the image into certain brightness regions, which correspond to one background and several objects. It is a way to create a binary image from a grayscale or full-colour image. This is typically done in order to separate "object" or foreground pixels from background pixels to aid in image processing. To split an image into smaller segments, or junks, using at least one colour or grey scale value to define their boundary. The advantage of obtaining first a binary image is that it reduces the complexity of the data and simplifies the process of recognition and classification.



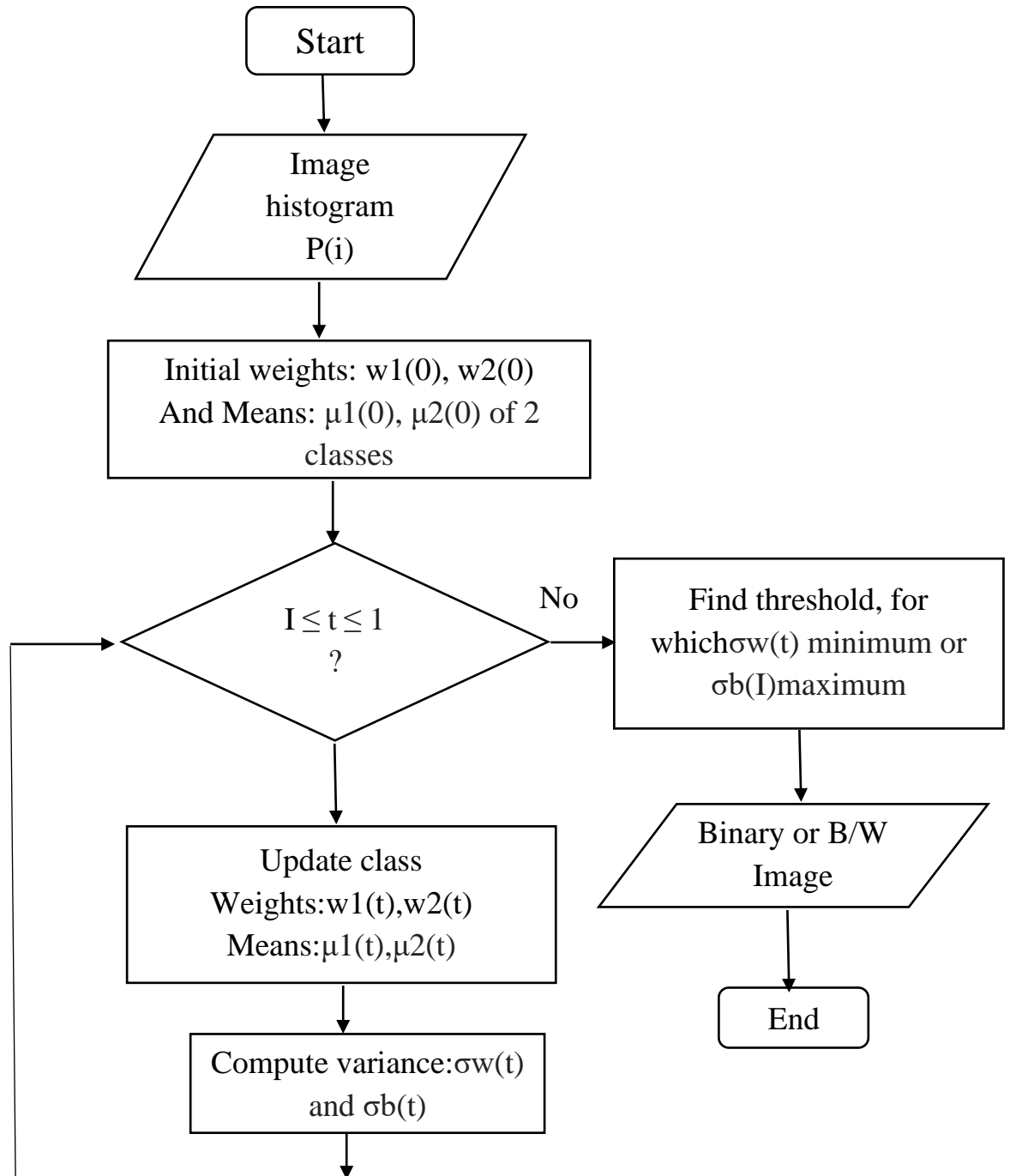
Figure.5.1 Multi Thresholding

OSTU MULTITHRESHOLDING

In computer vision and image processing, **Otsu's method**, named after Nobuyuki Otsu is used to perform automatic image thresholding. In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background. This threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance ^[2]. Otsu's method is a one-dimensional discrete analogy of Fisher's Discriminant Analysis, is related to Jenks optimization method, and is equivalent to a globally optimal k-means^[3] performed on the intensity histogram. The extension to multi-level

thresholding was described in the original paper^[2], and computationally efficient implementations have since been proposed.¹⁵

MULTI THRESHOLDING



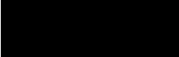










5.4 PIXEL CLASSIFICATION

Accordingly, the trained classification algorithm would output a class prediction for each individual pixel in an image. colour-based classification: Classification is done on a localized group of pixels, taking into account the spatial properties of each pixel as they relate to each other.

COLOR BASED CLASSIFICATION

RGB colour space or RGB colour system, constructs all the colors from the combination of the Red, Green and Blue colours. The red, green and blue use 8 bits each, which have integer values from 0 to 255. This makes $256 \times 256 \times 256 = 16777216$ possible color. Each pixel in the LED monitor displays colors this way, by combination of red, green and blue LEDs (light emitting diodes). When the red pixel is set to 0, the LED is turned off. When the red pixel is set to 255, the LED is turned fully on. Any value between them sets the LED to partial light emission.

RGB COLOR TABLE

Color	HTML / CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Green	#008000	(0,128,0)
	Purple	#800080	(128,0,128)
	Teal	#008080	(0,128,128)
	Navy	#000080	(0,0,128)

BLOBS

A Binary Large Object (BLOB) is a collection of binary data stored as a single entity in a database management system. Blobs are typically images, audio or other multimedia objects, though sometimes binary executable code is stored as a blob. Short for binary large object, a collection of binary data stored as a single entity in a database management systems (DBMS). BLOBs are used primarily to hold multimedia objects such as images, videos, and sound, though they can also be used to store programs or even fragments of code. A binary large object is a varying-length binary string that can be up to 2,147,483,647 characters long.

5.5 HISTOGRAM

The histogram function uses an automatic binning algorithm that returns bins with a uniform width, chosen to cover the range of elements in X and reveal the underlying shape of the distribution. histogram displays the bins as rectangles such that the height of each rectangle indicates the number of elements in the bin. Histograms are a type of bar plot for numeric data that group the data into bins. After you create a Histogram object, you can modify aspects of the histogram by changing its property values. This is particularly useful for quickly modifying the properties of the bins or changing the display.

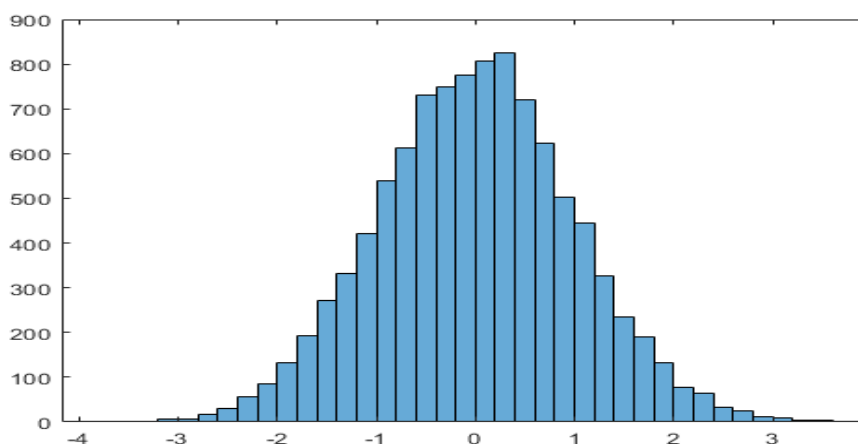


Figure.5.2Histogram (Plot)

5.6 PROCESS

Step 1: Acquire Image

Read in the `image.png` image, which is an image of `colourfullimage`. Instead of using `image.png`, you can acquire an image using the following functions in the Image Acquisition Toolbox.

Step 2: Calculate Sample Colors in L*a*b* Colour Space for Each Region

You can see major colors in the image: the background colors, red, green, purple, yellow, and magenta. Notice how easily you can visually distinguish these colors from one another. The L*a*b* colorspace (also known as CIELAB or CIE L*a*b*) enables you to quantify these visual differences. The L*a*b* colors space is derived from the CIE XYZ tristimulus values. The L*a*b* space consists of a luminosity 'L*' or brightness layer, chromaticity layer 'a*' indicating where colors falls along the red-green axis, and chromaticity layer 'b*' indicating where the colors falls along the blue-yellow axis. Your approach is to choose a small sample region for each colors and to calculate each sample region's average colors in 'a*b*' space. You will use these colors markers to classify each pixel.

Step 3: Classify Each Pixel Using the Nearest Neighbor Rule

Each colors marker now has an 'a*' and a 'b*' value. You can classify each pixel in the lab fabric image by calculating the Euclidean distance between that pixel and each colors marker. The smallest distance will tell you that the pixel most closely matches that colors marker. For example, if the distance between a pixel and the red colors marker is the smallest, then the pixel would be Labeled as a red pixel.

Step 4: Display Results of Nearest Neighbor Classification

The label matrix contains a colors label for each pixel in the fabric image. Use the label matrix to separate objects in the original fabric image by colors.

Step 5: Display 'a*' and 'b*' Values of the Labeled Colors

You can see how well the nearest neighbor classification separated the different colors populations by plotting the 'a*' and 'b*' values of pixels that were classified into separate colors. For display purposes, label each point with its colors label.

19

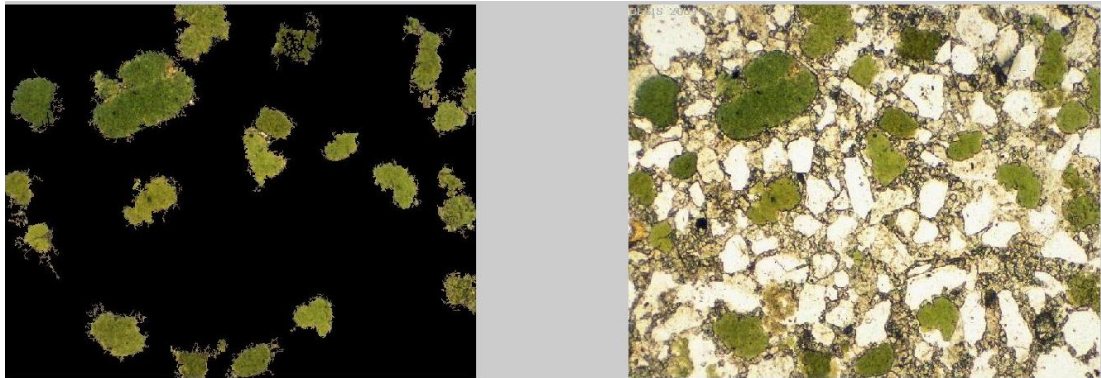


Figure.5.3colorbased segmentation

5.7 BINARY MASK

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The colors used for the objects in the image is the foreground colors while the rest of the image is the background colors. In the document-scanning industry, this is often referred to as "bi-tonal". Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit—i.e., a 0 or 1. The names black-and-white, B&W, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images. In Photoshop parlance, a binary image is the same as an image in "Bitmap" mode. Binary images often arise in digital image processing as masks or as the result of certain operations such as segmentation, thresholding, and dithering.

Create Binary Mask Based on Colour Values

A region of interest (ROI) is a portion of an image that you want to filter or operate on in some way. The toolbox supports a set of ROI objects that you can use to create ROIs of many shapes, such circles, ellipses, polygons, rectangles, and hand-drawn shapes. After creation, you can use ROI object properties to customize their appearance and functioning. In addition, the ROI objects support object functions and events that you can use to implement interactive behaviour

(Roicolor)Select region of interest (ROI) based on color.

CHAPTER 6

INTRODUCTION TO IMAGE RECOGNITION

6.1 IMAGE RECOGNITION

As we progress and develop technology changes rapidly, New methods are being employed in industrial manufacture and inspection every day. There was a time when manual labour was in much demand in the industries for jobs like packaging inspection etc. which are a repetitive process. Now it is time for automated machines to take over these simple jobs and make them more efficient. The sensors are one of the most important part of automation. They provide inputs for the controllers to take action based upon the inputs received. Various kinds of sensors like temperature sensor, humidity sensor, infrared ultrasonic sensor, camera etc. are easily available in market, High precision is required for better feedback.

Cameras are used for Image Processing which finds huge application in the field of automation industry. It captures the images or streams live video and then they are processed as per the needs of the application. This project attempts at demonstrating the shape and colour recognition of an object using an algorithm which will be explained in detail. This algorithm is realized with the help of matlab.

COLOR DETECTION

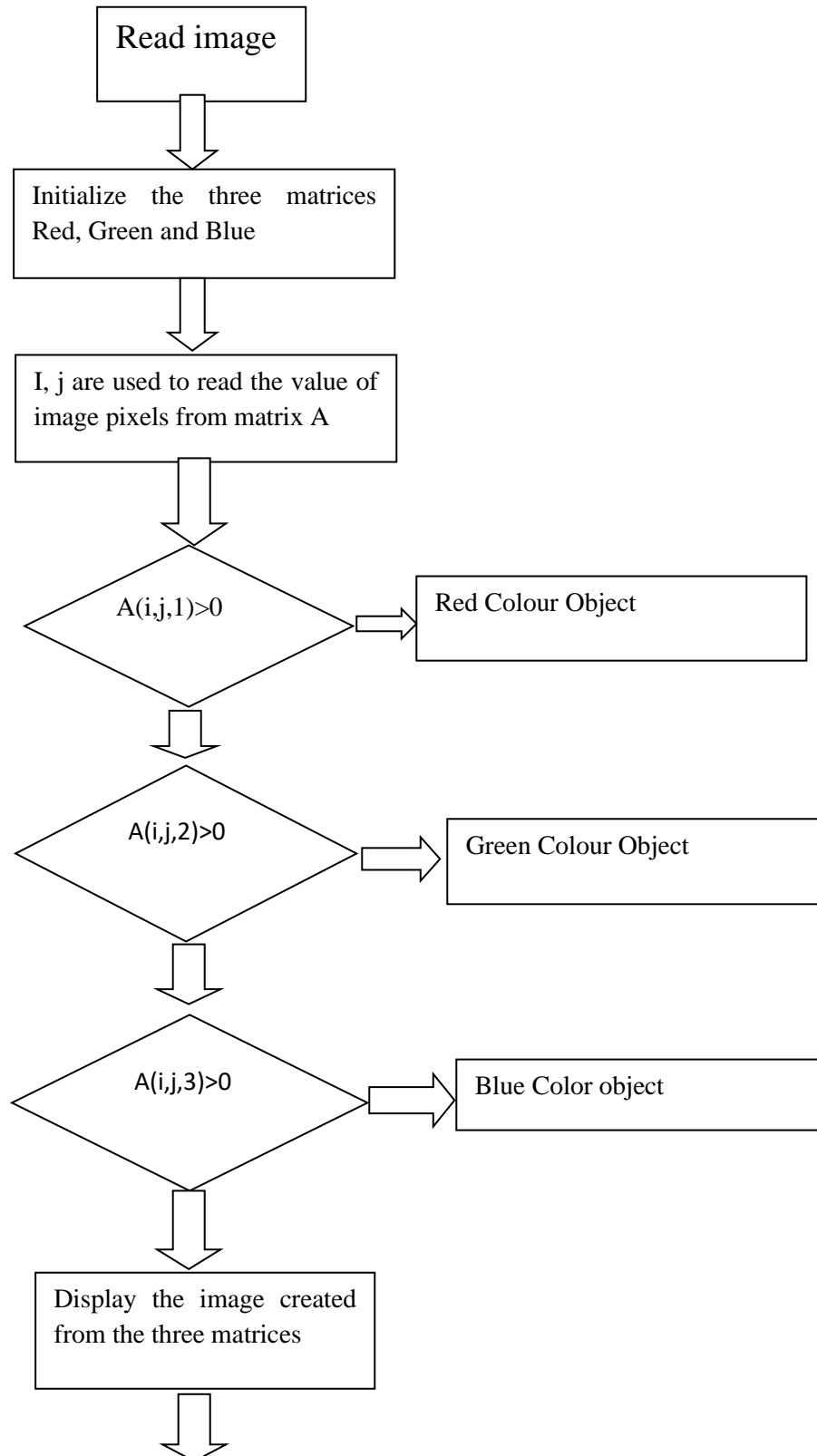
A colour detection algorithm identifies pixels in an image that match a specified colour or colour range. The colour of detected pixels can then be changed to distinguish them from the rest of the image.

SHAPE DETECTION

Shape recognition can be defined as “the ability to identify and name basic shapes”. For preschool children this skill is normally focused on basic shapes: square, triangle, circle, rectangle, diamond and oval. Shape recognition can actually be broken down into separate skills – matching, identifying, and naming. When matching, a child can find the matching

shape when shown an example. This is the first step in shape recognition. The child perceives the differences between the shapes.

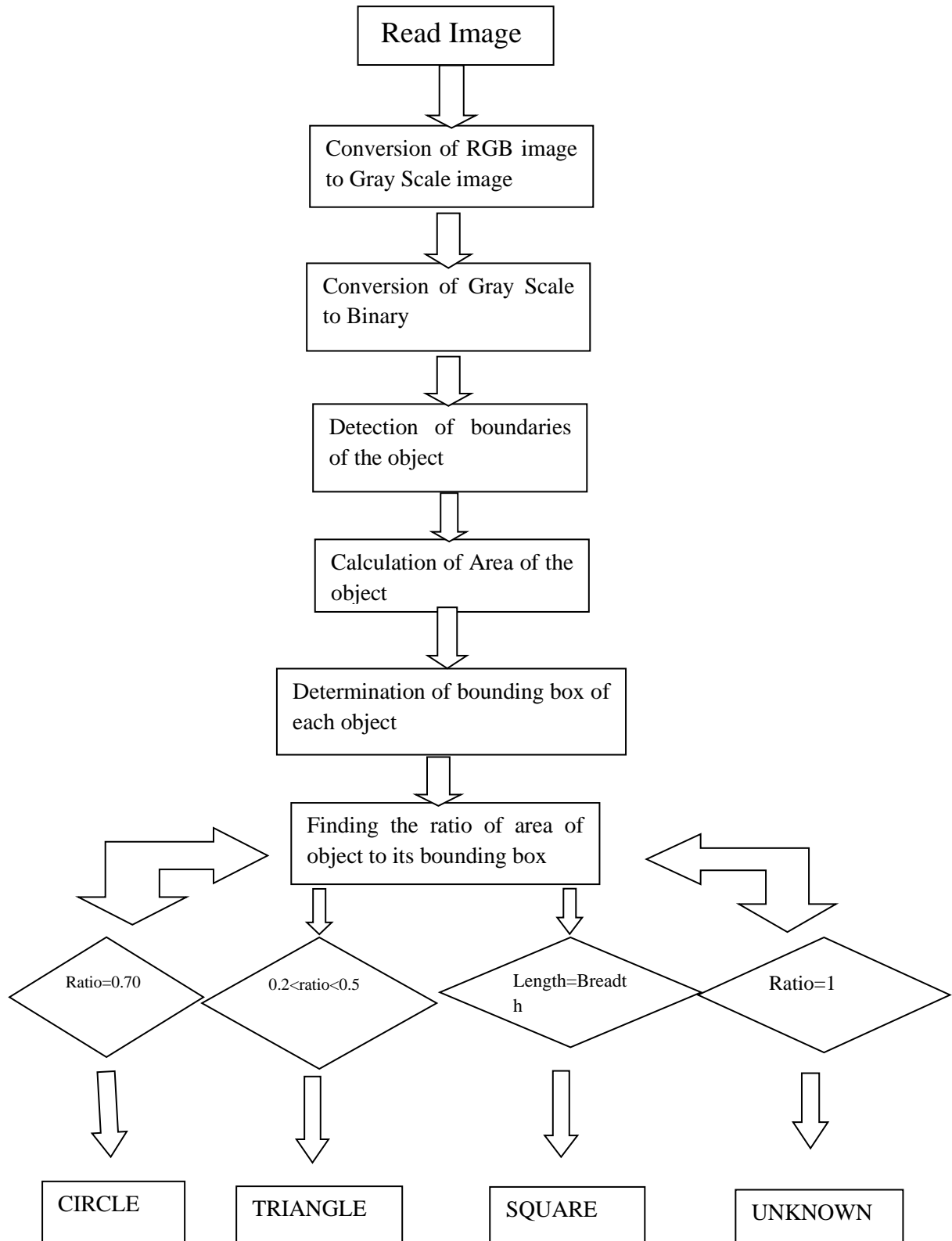
COLOR DETECTION



Stop

22

SHAPE DETECTION



6.4 PROCESS

- **Read the input image:** Any RGB Image is captured and loaded in MATLAB. Each pixel of the RGB image is in the form of element of the matrix.
- Conversion of RGB image to grey scale and then to binary image using Thresholding Process.
- **Detection of the boundary of the object:** The 3D input image which has already been converted into 2d array. One of the pixel is taken as a reference and moving in a fixed direction to detect other object pixels. Hence the boundary of the object is detected.
- **Finding bounding box of the given object:** Bounding box is an imaginary rectangle enclosing the given object. Because of inclination of the object, the size of the bounding box changes. If the Object is inclined at some angle the box is rotated by angle of inclination and made parallel to the object axis.
- **Area of the object:** The area of the object can be calculated by the summation of pixels within the boundary of the object.
- **Ratio:** The ratio of area of the object to the area of bounding box is calculated.

Ratio = $\frac{\text{Area of object}}{\text{Area of bounding index}}$

- **Comparison of the calculated ratio:** The calculated ratio is compared with the predefined values to determine the shape of the given object.
- **Colour detection:** Read the image and construct the matrices for Red, Green and Blue. All the pixels are compared with each other. If the red colour matrix

has positive value and others are zero then that object is detected as red. Similarly, for green and blue objects. Finally, image is displayed separately.

24

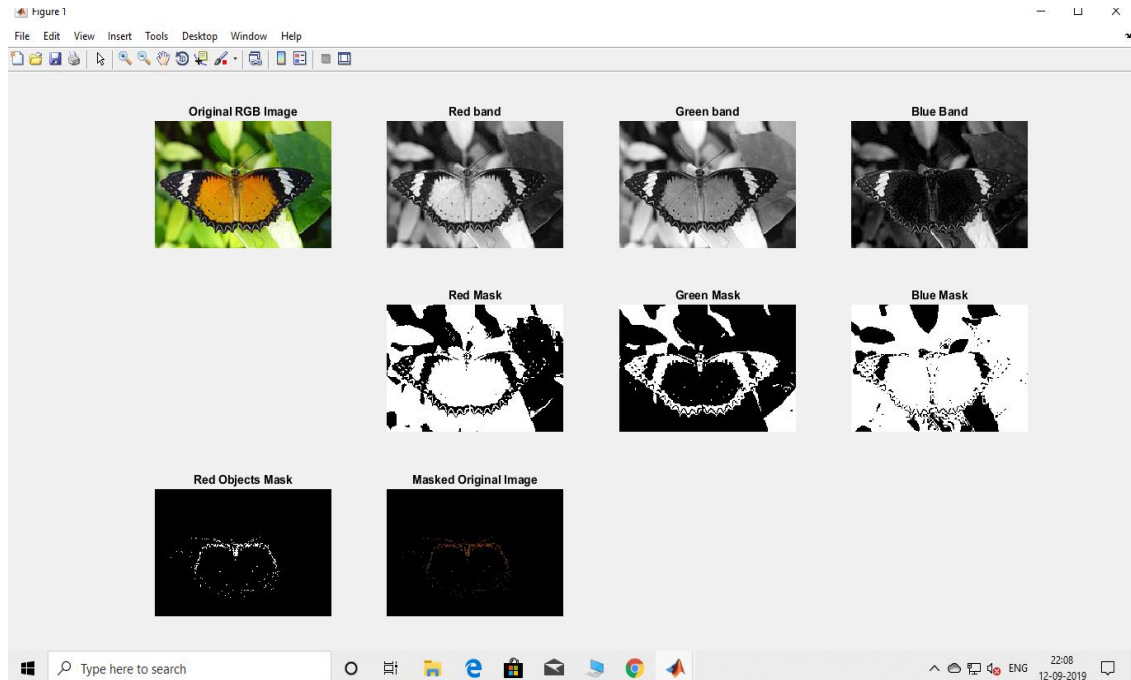


Figure.6.1 Colour detection

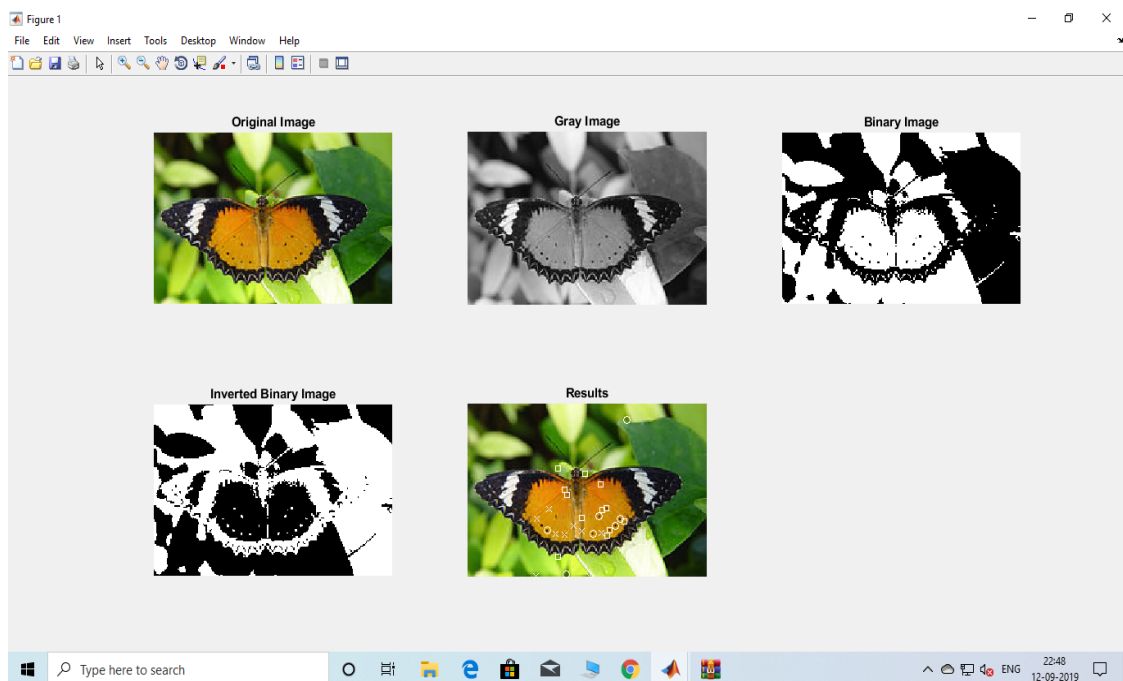


Figure.6.2 Shape detection

CHAPTER 7

ADVANTAGES AND APPLICATIONS

ADVANTAGES

- High Efficiency
- Low Complexity
- Predefined functions
- Independent platform
- Ease of use

APPLICATIONS

- Object Detection
- Space research
- Video surveillance
- Security analysis
- VFX graphics
- Research centres

CHAPTER 8

CONCLUSION AND REFERENCES

CONCLUSION

In this paper, the adaptive filtering approach was extended to the case of colour (multichannel) images. Procedures for region growing and estimation of uncorrupted pixels using statistics computed within adaptive regions were developed for the vectoral case. It was shown that results obtained by the adaptive-neighbourhood approach are better, both in objective and subjective terms, than the results obtained using fixed filters. The adaptive Vector filtering approach may be easily extended to the case of multispectral images, which makes it a powerful tool for vectoral image processing.

REFERENCES

www.mathworks.com

www.Github.com

www.ieee.org

www.wikipedia.org

CHAPTER 9

OUTPUTS

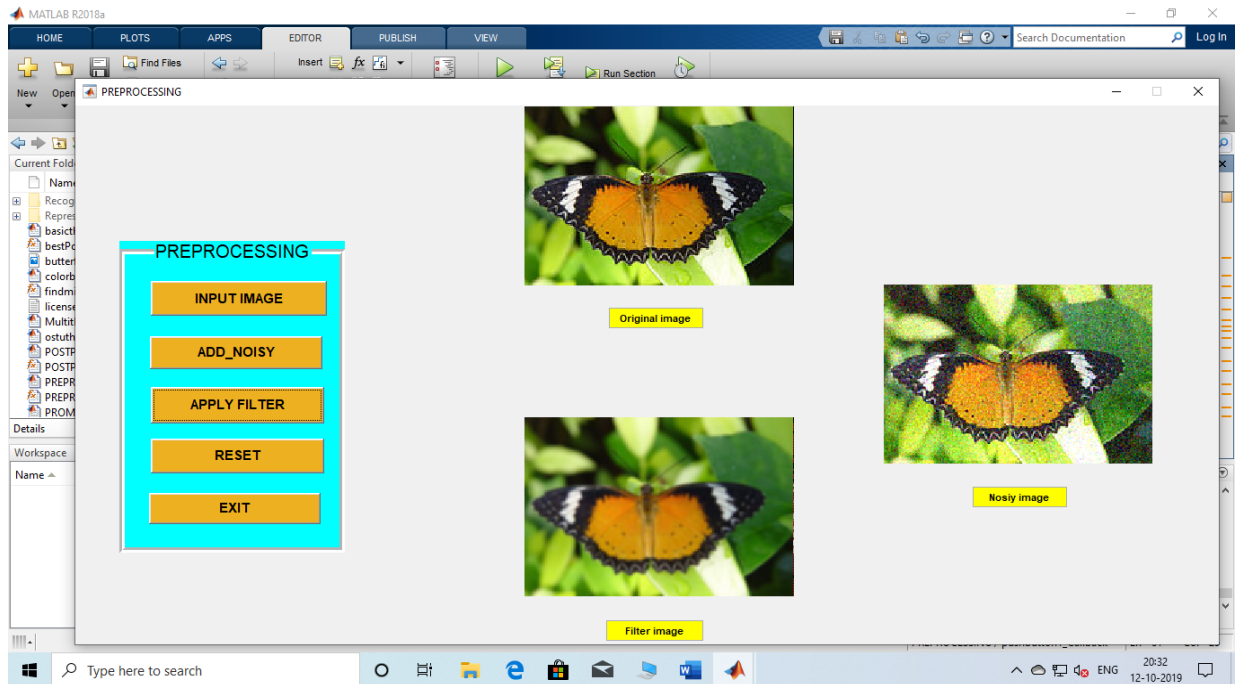


Figure.9.1Preprocessing output.

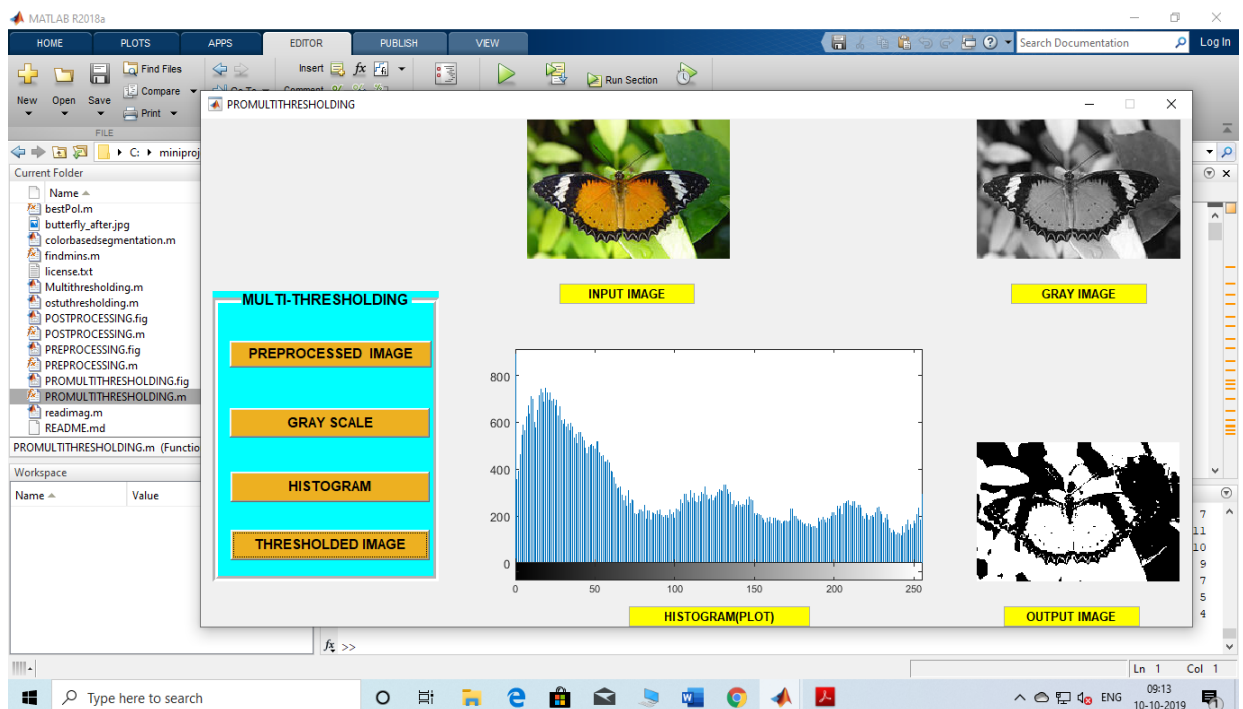


Figure.9.2Multithresholding output.

28

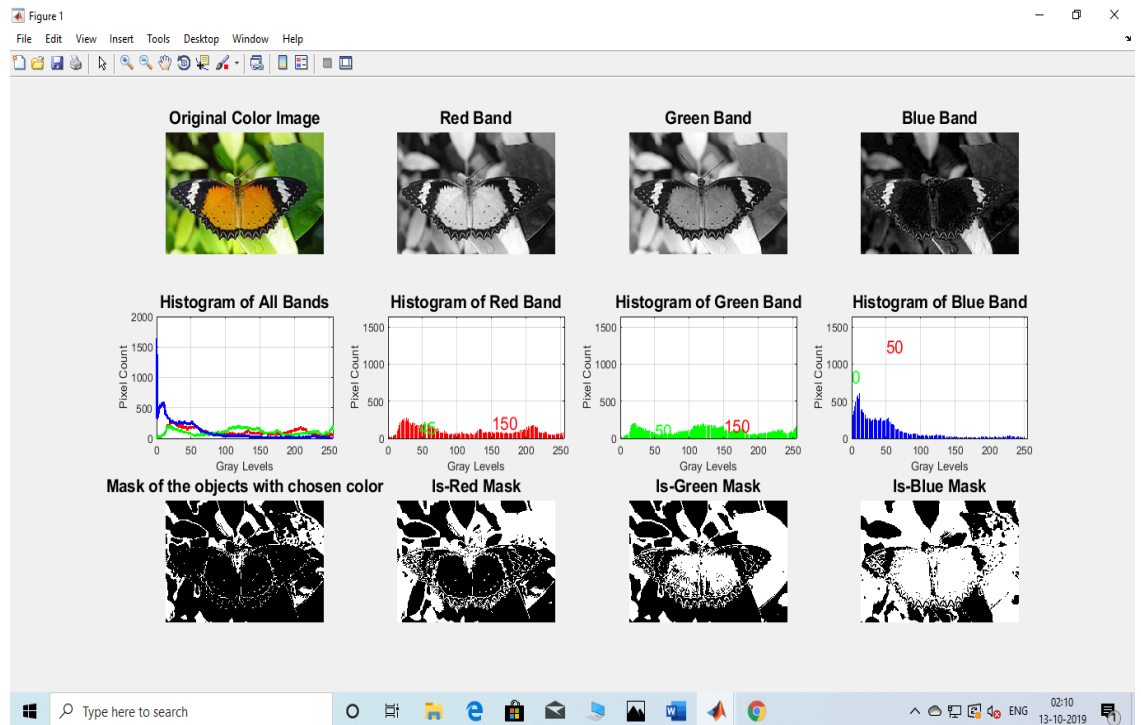


Figure.9.3RGB color band of image with RGB mask.

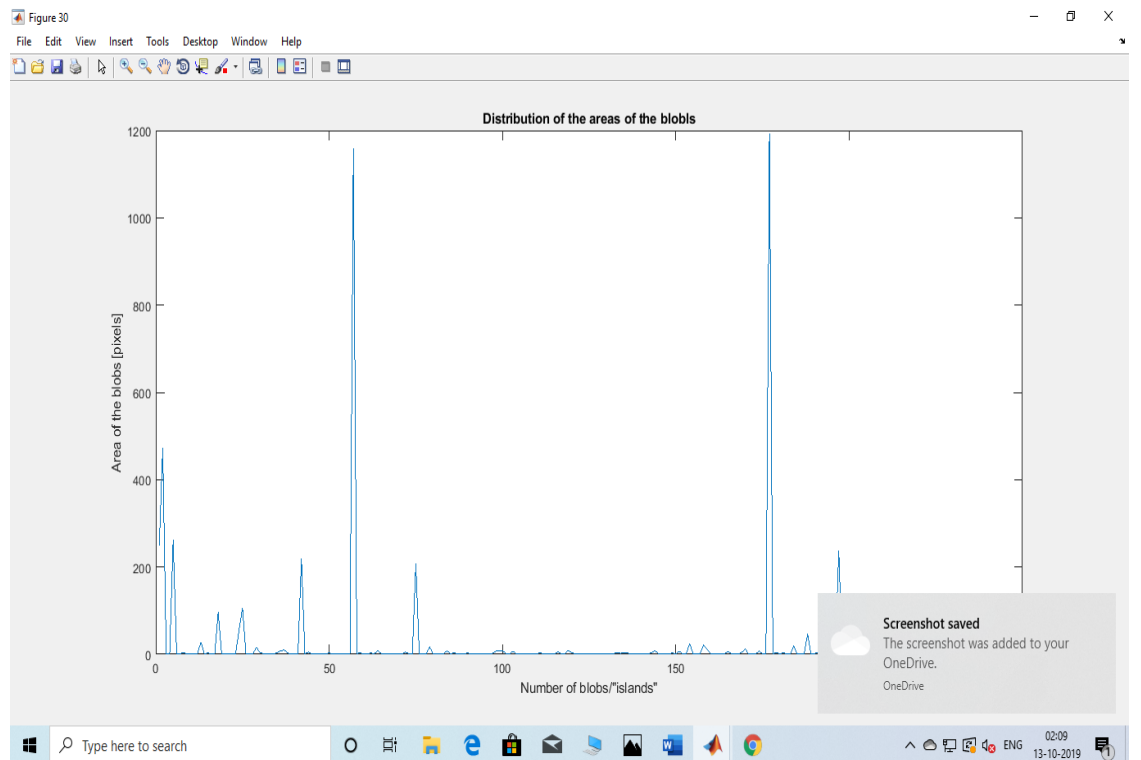


Figure.9.4Distribution of the area's blobs

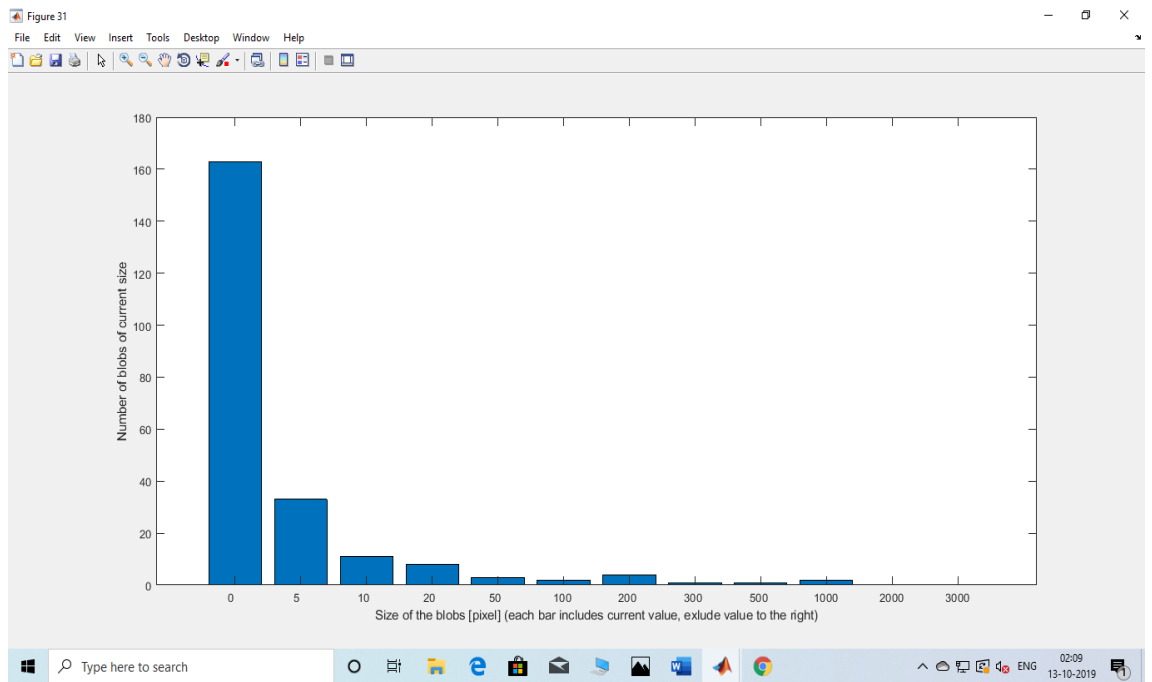


Figure.9.5 Histogram to Size of the picture to occupy screen.

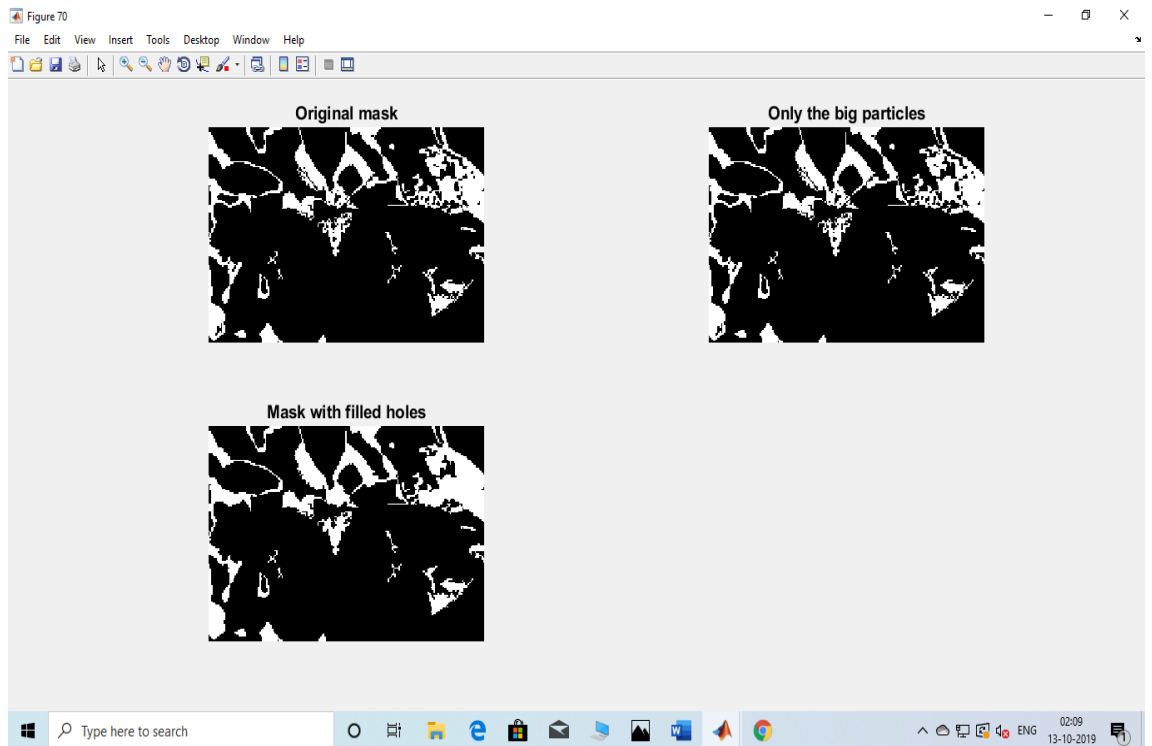


Figure.9.6 Binary mask filling small holes in image.

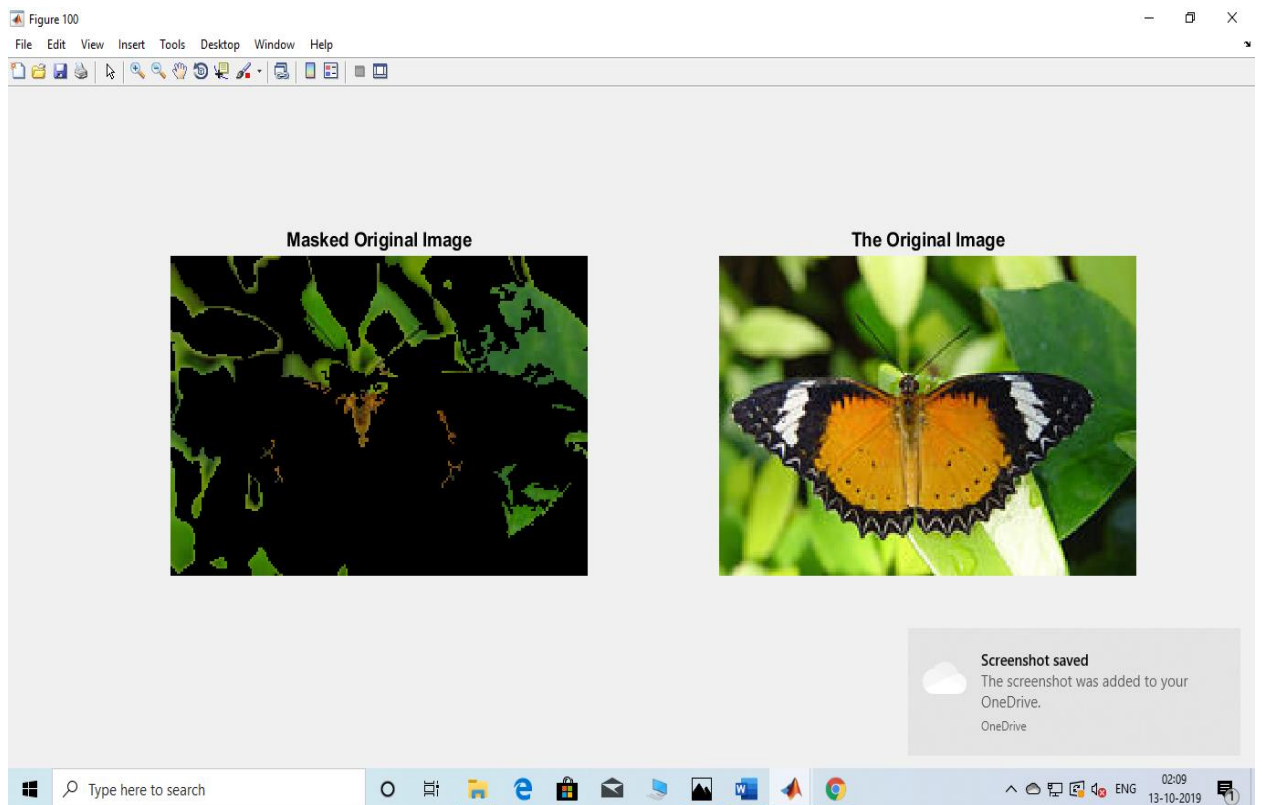


Figure.9.7Masked Original image.

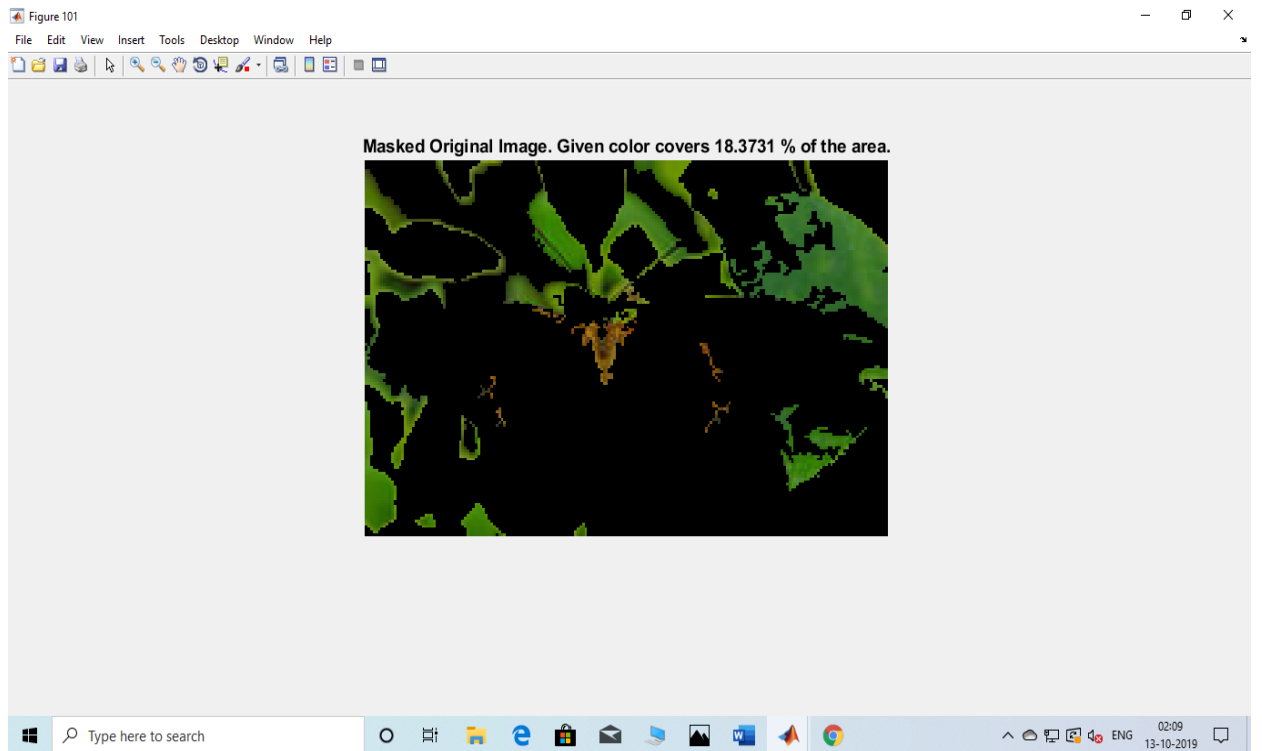


Figure.9.8Masked Original Image with area coverage in (%).

FINAL RESORTED IMAGE

CODE SECTION

PREPROCESSING

```
function varargout = PREPROCESSING(varargin)
% PREPROCESSING MATLAB code for PREPROCESSING.fig
%     PREPROCESSING, by itself, creates a new PREPROCESSING or
%     raises the existing
%     singleton*.
%
%     H = PREPROCESSING returns the handle to a new PREPROCESSING or
%     the handle to
%     the existing singleton*.
%
%     PREPROCESSING('CALLBACK', hObject,eventData,handles,...) calls
%     the local
%     function named CALLBACK in PREPROCESSING.M with the given
%     input arguments.
%
%     PREPROCESSING('Property','Value',...) creates a new
%     PREPROCESSING or raises the
%     existing singleton*. Starting from the left, property value
%     pairs are
%     applied to the GUI before PREPROCESSING_OpeningFcn gets
%     called. An
%     unrecognized property name or invalid value makes property
%     application
%     stop. All inputs are passed to PREPROCESSING_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

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

% Last Modified by GUIDE v2.5 11-Sep-2019 01:30:29

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

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

% --- Executes just before PREPROCESSING is made visible.
```

```

function PREPROCESSING_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 PREPROCESSING (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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

% --- Outputs from this function are returned to the command line.
function varargout = PREPROCESSING_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;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
a1=imread('butterfly_after.jpg');
axes(handles.axes1);
imshow(a1)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
b=imread('butterfly_after.jpg');
a2=imnoise(b,'gaussian',0.08);
% gaussian noise, mean = 0, variance = 0.08
axes(handles.axes2);
imshow(a2);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
a1 = imread('butterfly_after.jpg');

```

```

a3 = imgaussfilt(a1,1);
imwrite(a3,'t1.jpg');
axes(handles.axes3);
imshow(a3)

% --- Executes on button press in pushbutton4.
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)

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clearstr = 'clear all'
avalin('base',clearstr);

%delete the figure
delete(handles.figureExitButton);

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4
%        as a double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
%            called

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

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5
%        as a double

```

```
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

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

```
function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of edit6 as text
%       str2double(get(hObject, 'String')) returns contents of edit6
as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

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

```
% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject, 'Value') returns toggle state of togglebutton1
```

```
% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```


MULTITHRESHOLDING

```
function varargout = PROMULTITHRESHOLDING(varargin)
% PROMULTITHRESHOLDING MATLAB code for PROMULTITHRESHOLDING.fig
%     PROMULTITHRESHOLDING, by itself, creates a new
PROMULTITHRESHOLDING or raises the existing
%     singleton*.
%
%     H = PROMULTITHRESHOLDING returns the handle to a new
PROMULTITHRESHOLDING or the handle to
%     the existing singleton*.
%
%     PROMULTITHRESHOLDING('CALLBACK',hObject,eventData,handles,...)
calls the local
%     function named CALLBACK in PROMULTITHRESHOLDING.M with the
given input arguments.
%
%     PROMULTITHRESHOLDING('Property','Value',...) creates a new
PROMULTITHRESHOLDING or raises the
%     existing singleton*. Starting from the left, property value
pairs are
%     applied to the GUI before PROMULTITHRESHOLDING_OpeningFcn gets
called. An
%     unrecognized property name or invalid value makes property
application
%     stop. All inputs are passed to
PROMULTITHRESHOLDING_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

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

% Last Modified by GUIDE v2.5 11-Sep-2019 03:09:09

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

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

```
% --- Executes just before PROMULTITHRESHOLDING is made visible.
function PROMULTITHRESHOLDING_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 PROMULTITHRESHOLDING (see
VARARGIN)
```

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

```
% Update handles structure
guidata(hObject, handles);
```

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

```
% --- Outputs from this function are returned to the command line.
function varargout = PROMULTITHRESHOLDING_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;
```

```
function edit1_Callback(~, ~, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit1 as text
%         str2double(get(hObject,'String')) returns contents of edit1
as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, ~, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```

function edit2_Callback(~, ~, ~)
% hObject      handle to edit2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%         str2double(get(hObject,'String')) returns contents of edit2
%         as a double

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edit2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
%              called

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

function edit3_Callback(hObject, eventdata, handles)
% hObject      handle to edit3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%         str2double(get(hObject,'String')) returns contents of edit3
%         as a double

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edit3 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
%              called

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

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

```

```

a=imread('t1.jpg');% Read the Image
axes(handles.axes1);
imshow(a);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a=imread('butterfly_after.jpg');% Read the Image
Y=rgb2gray(a);
imwrite(Y,'t2.jpg');
axes(handles.axes2);
imshow(Y);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
B =imread('t2.jpg')
axes(handles.axes3);
imhist(B)

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4
%        as a double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
%            called

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

% --- Executes on button press in pushbutton4.
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)
a=imread('t2.jpg');% Read the Image
n=imhist(a); % Compute the histogram
N=sum(n); % sum the values of all the histogram values

```

```

max=0; %initialize maximum to zero
for i=1:256
    P(i)=n(i)/N; %Computing the probability of each intensity level
end

for T=2:255 % step through all thresholds from 2 to 255
    w0=sum(P(1:T)); % Probability of class 1 (separated by threshold)
    w1=sum(P(T+1:256)); %probability of class2 (separated by
threshold)
    u0=dot([0:T-1],P(1:T))/w0; % class mean u0
    u1=dot([T:255],P(T+1:256))/w1; % class mean u1
    sigma=w0*w1*((u1-u0)^2); % compute sigma i.e variance(between
class)
    if sigma>max % compare sigma with maximum
        max=sigma; % update the value of max i.e max=sigma
        threshold=T-1; % desired threshold corresponds to maximum
variance of between class
    end
end

bw=im2bw(a,100/255);
% Convert to Binary Image
imwrite(bw,'t3.jpg');
axes(handles.axes4);
imshow(bw)

```

COLOR BASED SEGMENTATION

```

clear all; close all; clc
%% General settings
fontSize = 14;
%% Pick the image and Load the image
[filename, pathname] = uigetfile( ...
    {'*.jpg;*.tif;*.tiff;*.png;*.bmp', 'All image Files (*.jpg,
*.tif, *.tiff, *.png, *.bmp)'; ...
    '*.*', 'All Files (*.*)'}, ...
    'Pick a file');
f = fullfile(pathname, filename);
disp('Reading image')
rgbImage = imread(f);
[rows columns numberOfColorBands] = size(rgbImage);
%% Create directory for the results
k = findstr('.', filename); % find where the extension starts
FileToSave= filename(1:k-1); %
fileNameWrongCounter = 0;
% if the name of the input wav file is longer than 256 chrachters,
% which is the limit for windows folder name's length,
% the directory will be composed of the first 200 chraracters
% of the file + the "CounterNumber" will be added to see,
% how meny files has to long name
% However, because of the excel file names restriction, the file name
% cannot be 256, but only 218. Therefore, the folder name lengthe is
% restricted to 150
CreateNewFoldrSucces = 0;
while CreateNewFoldrSucces == 0
    if length(filename) < 150
        % the lenght of the name for the folder can be up to 255
        % characters, but later, the lenght of the file name (TOGETHER WITH
        % the path) of the excel/csv files with the results can be only 218
        DirectoryName = ['Res_', FileToSave];
        % Matalb does not like when the names beginns with a NUMBER!!!
    end
end

```

```

% F = folder
% dontappend the extension of the wav file: (.wav) = 4characters
% FileToSave is already adjusted with respect to the
% prescribed file name, and is without the extension
[SUCCESS,MESSAGE,MESSAGEID] = mkdir(DirectoryName);
% SUCCESS = 1 -> folder was created.
% what if the folder already exists?
else% if the input file name is longer than 150 characters
fileNameWrongCounter = fileNameWrongCounter + 1;
DirectoryName = strcat('Res_', FileToSave(1:30), '_',
num2str(fileNameWrongCounter));
[SUCCESS,MESSAGE,MESSAGEID] = mkdir(DirectoryName );
end

if SUCCESS == 1
CreateNewFoldrSucces = 1;
break
end
end
%% Diary - save content of teh Command Window to a txt file
CommantWindowText= strcat(pathname, DirectoryName, '\ComWind.txt');
diary(CommantWindowText)
% Everything that is printed in the Command window will be saved in a
% text file. End tag is "diary off" in the end of the script
%% If the image is monochrome (indexed), convert it to color.
% Check to see if it's an 8-bit image needed later for scaling).
ifstrcmpi(class(rgbImage), 'uint8')
% Flag for 256 gray levels.
eightBit = true;
else
eightBit = false;
end

ifnumberOfColorBands == 1
ifisempty(storedColorMap)
% Just a simple gray level image, not indexed with a stored color
map.
% Create a 3D true color image where we copy the monochrome image
into all 3 (R, G, & B) color planes.
rgbImage = cat(3, rgbImage, rgbImage, rgbImage);
else
% It's an indexed image.
rgbImage = ind2rgb(rgbImage, storedColorMap);
% ind2rgb() will convert it to double and normalize it to the range
0-1.
% Convert back to uint8 in the range 0-255, if needed.
ifeightBit
rgbImage = uint8(255 * rgbImage);
end
end
end

%% Display the color image
disp('Displaying color original image')
F1 = figure(1);
subplot(3,4,1);
imshow(rgbImage);

ifnumberOfColorBands> 1
title('Original Color Image', 'FontSize', fontSize);

```

```

else
    caption = sprintf('Original Indexed Image\n(converted to true
color with its stored colormap)');
title(caption, 'FontSize', fontSize);
end

%% Size of the picture - to occupy the whole screen
scnsize = get(0, 'ScreenSize'); % - - width height
position = get(F1, 'Position'); % x-pos y-pos width height
outerpos = get(F1, 'OuterPosition');
borders = outerpos - position;
edge = abs(borders(1))/2;
pos1 = [edge, ...
        1/20*scnsize(4), ...
        9/10*scnsize(3), ...
        9/10*scnsize(4)];
set(F1, 'OuterPosition', pos1)
%% Explore RGB
% Extract out the color bands from the original image
% into 3 separate 2D arrays, one for each color component.
redBand = rgbImage(:, :, 1);
greenBand = rgbImage(:, :, 2);
blueBand = rgbImage(:, :, 3);
% Display them.
subplot(3, 4, 2);
imshow(redBand);
title('Red Band', 'FontSize', fontSize);
subplot(3, 4, 3);
imshow(greenBand);
title('Green Band', 'FontSize', fontSize);
subplot(3, 4, 4);
imshow(blueBand);
title('Blue Band', 'FontSize', fontSize);
%% Compute and plot the red histogram.
hR = subplot(3, 4, 6);
[countsR, grayLevelsR] = imhist(redBand);
maxGLValueR = find(countsR > 0, 1, 'last');
maxCountR = max(countsR);
bar(countsR, 'r');
grid on;
xlabel('Gray Levels');
ylabel('Pixel Count');
title('Histogram of Red Band', 'FontSize', fontSize);
%% Compute and plot the green histogram.
hG = subplot(3, 4, 7);
[countsG, grayLevelsG] = imhist(greenBand);
maxGLValueG = find(countsG > 0, 1, 'last');
maxCountG = max(countsG);
bar(countsG, 'g', 'BarWidth', 0.95);
grid on;
xlabel('Gray Levels');
ylabel('Pixel Count');
title('Histogram of Green Band', 'FontSize', fontSize);
%% Compute and plot the blue histogram.
hB = subplot(3, 4, 8);
[countsB, grayLevelsB] = imhist(blueBand);
maxGLValueB = find(countsB > 0, 1, 'last');
maxCountB = max(countsB);
bar(countsB, 'b');
grid on;
xlabel('Gray Levels');

```

```

ylabel('Pixel Count');
title('Histogram of Blue Band', 'FontSize', fontSize);
%% Set all axes to be the same width and height.
% This makes it easier to compare them.
maxGL = max([maxGLValueR, maxGLValueG, maxGLValueB]);
ifeightBit
maxGL = 255;
end
maxCount = max([maxCountR, maxCountG, maxCountB]);
axis([hRhGhB], [0 maxGL 0 maxCount]);
%% Plot all 3 histograms in one plot.
subplot(3, 4, 5);
plot(grayLevelsR, countsR, 'r', 'LineWidth', 2);
    grid on;
xlabel('Gray Levels');
ylabel('Pixel Count');
    hold on;
plot(grayLevelsG, countsG, 'g', 'LineWidth', 2);
plot(grayLevelsB, countsB, 'b', 'LineWidth', 2);
title('Histogram of All Bands', 'FontSize', fontSize);
maxGrayLevel = max([maxGLValueR, maxGLValueG, maxGLValueB]);
% Trim x-axis to just the max gray level on the bright end.
ifeightBit
xlim([0 255]);
else
xlim([0 maxGrayLevel]);
end
%% Now select thresholds for the 3 color bands.
% pop-up window
prompt = {'RED color threshold LOWER:', 'RED color threshold UPPER:',
...
'GREEN color threshold LOWER:', 'GREEN color threshold UPPER:', ...
'BLUE color threshold LOWER:', 'BLUE color threshold UPPER:'};
dlg_title = 'Input';
num_lines = 1;
%def = {'0', '50', '150', '255', '0', '100'}; % only green
def = {'45', '150', '50', '150', '0', '50'}; % green and similar
%def = {'235', '255', '235', '255', '0', '200'}; % red
% def = {'250', '255', '250', '255', '0', '245'}; % yellow
% def = {'190', '255', '160', '245', '60', '165'}; % yellow + light brown
+ yellow-white
answer = inputdlg(prompt, dlg_title, num_lines, def);

redThresholdLow = str2num(answer{1});
redThresholdHigh = str2num(answer{2});
greenThresholdLow = str2num(answer{3});
greenThresholdHigh = str2num(answer{4});
blueThresholdLow = str2num(answer{5});
blueThresholdHigh = str2num(answer{6});
%% Show the thresholds as vertical red bars on the histograms.
PlaceThresholdBars(1, 3, 4, 6, redThresholdLow, redThresholdHigh,
fontSize, max(countsR));
PlaceThresholdBars(1, 3, 4, 7, greenThresholdLow,
greenThresholdHigh, fontSize, max(countsG));
PlaceThresholdBars(1, 3, 4, 8, blueThresholdLow,
blueThresholdHigh, fontSize, max(countsB));

%% Now apply each color band's particular thresholds to the color
band
redMask = (redBand >= redThresholdLow) & (redBand <= redThresholdHigh);

```



```

greenMask = (greenBand>= greenThresholdLow) & (greenBand<=
greenThresholdHigh);
blueMask = (blueBand>= blueThresholdLow) & (blueBand<=
blueThresholdHigh);
%% Display the thresholded binary images.
subplot(3, 4, 10);
imshow(redMask, []);
title('Is-Red Mask', 'FontSize', fontSize);
subplot(3, 4, 11);
imshow(greenMask, []);
title('Is-Green Mask', 'FontSize', fontSize);
subplot(3, 4, 12);
imshow(blueMask, []);
title('Is-Blue Mask', 'FontSize', fontSize);

%% Combine the masks to find where all 3 are "true."
% Then we will have the mask of only the chosen color parts of the
image.
ObjectsMask = uint8(redMask&greenMask&blueMask);
subplot(3, 4, 9);
imshow(ObjectsMask, []);
caption = sprintf('Mask of the objects with chosen color');
title(caption, 'FontSize', fontSize);

f2 = fullfile(pathname, DirectoryName);
fileNamePlot= strcat(f2, '\Figure_1.jpg');
saveas(gcf,num2str(char(fileNamePlot)))
%% Histogram small areas
% Measure the mean RGB and area of all the detected blobs.
[meanRGB, areas, numberOfBlobs] = MeasureBlobs(ObjectsMask,
redBand, greenBand, blueBand);
F30 = figure(30);
plot(areas(:,1))
title('Distribution of the areas of the blobs')

save('BlobAreas', 'meanRGB', 'areas', 'numberOfBlobs')
% Size of the picture - to occupy the whole screen
pos2 = [1/4*scnsize(3),...
1/20*scnsize(4), ...
2/3*scnsize(3),...
2/3*scnsize(4)];
set(F30, 'OuterPosition', pos2)
xlabel('Number of blobs/"islands"')
ylabel('Area of the blobs [pixels]')
fileNamePlot= strcat(f2, '\Figure_30.jpg');
saveas(gcf,num2str(char(fileNamePlot)))

figure(31)
XTickDescr = [0,5,10,20,50,100,200,300,500,1000,2000,3000];
N = hist(areas(:,1), XTickDescr);
bar(N)
set(gca, 'XTick', 1:length(XTickDescr))
set(gca, 'XTickLabel', XTickDescr)
xlhand = get(gca, 'xlabel');
set(xlhand, 'string', 'X', 'fontsize', 0.3)
xlabel('Size of the blobs [pixel] (each bar includes current value,
exlude value to the right)')
ylabel('Number of blobs of current size')
z1 = max(areas(:,1));
fileNamePlot= strcat(f2, '\Figure_31_histBlobs.jpg');

```

```

saveas(gcf,num2str(char(fileNamePlot)))
%% Insert the minimal area that will be counted
% Every blob smaller than this one will be ommited
prompt = {'Minimal area [pixels] of the blob that will be kept:'};
dlg_title = 'Min.area';
num_lines = 1;
def = {'10'};% yellow
answer2 = inputdlg(prompt,dlg_title,num_lines,def);
answer2 = str2num(answer2{1});

%% Ignore all small areas
F70 = figure(70);
subplot(2,2,1)
imshow(ObjectsMask, []);
title('Original mask', 'FontSize', fontSize)
set(F70,'OuterPosition',pos1)

ObjectsMask = uint8(bwareaopen(ObjectsMask,answer2));
figure(70)
subplot(2,2,2)
imshow(ObjectsMask, []);
title('Only the big particles', 'FontSize', fontSize)
%% Fill in any holes in the regions, since they are most likely red
also.
message = sprintf('Do you want to close the holes in the blobs?');
reply = questdlg(message, 'Close holes?', 'Yes','No', 'Yes');
ifstrcmpi(reply, 'Yes')
figure(70)
subplot(2,2,1);
imshow(ObjectsMask, []);
title('Original mask', 'FontSize', fontSize)
subplot(2,2,3);
ObjectsMask = uint8(imfill(ObjectsMask, 'holes'));
imshow(ObjectsMask, []);
title('Mask with filled holes', 'FontSize', fontSize);
end

fileNamePlot= strcat(f2, '\Figure_70.png');
saveas(gcf,num2str(char(fileNamePlot)))
%% Use the object mask to mask out the portions of the rgb image.
maskedImageR = ObjectsMask .* redBand;
maskedImageG = ObjectsMask .* greenBand;
maskedImageB = ObjectsMask .* blueBand;
% Concatenate the masked color bands to form the rgb image.
maskedRGBImage = cat(3, maskedImageR, maskedImageG, maskedImageB);

%% Show the masked off and original image.
F100 = figure(100);
subplot(1,2,1);
imshow(maskedRGBImage);
caption = sprintf('Masked Original Image');
title(caption, 'FontSize', fontSize);
subplot(1,2,2);
imshow(rgbImage);
title('The Original Image', 'FontSize', fontSize);

set(F100,'OuterPosition',pos1)

Text_descr= strcat('R_l =', num2str(redThresholdLow), ', R_u = ',
num2str(redThresholdHigh), ...

```

```

', G_l =', num2str(greenThresholdLow), ', G_u = ',
num2str(greenThresholdHigh), ...
', B_l =', num2str(blueThresholdLow), ', B_u = ',
num2str(blueThresholdHigh))
    Text_descr2 = strcat('Filled holes = ', reply, ', minimal
counted area = ', num2str(answer2), ' [pixel]', ...
', max. blob size = ', num2str(max(areas(:,1))), ' [pixel], # of
counted blobs = ', num2str(length(areas(:,1))));

    text(-3000,2200,Text_descr) ;
    text(-3000,2500,Text_descr2) ;

f2 = fullfile(pathname, DirectoryName);
fileNamePlot= strcat(f2, '\Figure_100.jpg');
saveas(gcf,num2str(char(fileNamePlot)))

%% Measure the mean RGB and area of all the detected blobs.
clear meanRGB
clear areas
clear numberOfBlobs
    [meanRGB, areas, numberOfBlobs] = MeasureBlobs(ObjectsMask,
redBand, greenBand, blueBand);
if numberOfBlobs > 0
fprintf(1, '\n-----\n');
fprintf(1, 'Blob #, Area in Pixels, Mean R, Mean G, Mean B\n');
fprintf(1, '-----\n');
for blobNumber = 1 : numberOfBlobs
fprintf(1, '%5d, %14d, %6.2f, %6.2f, %6.2f\n', blobNumber,
areas(blobNumber), ...
meanRGB(blobNumber, 1), meanRGB(blobNumber, 2), meanRGB(blobNumber,
3));
end
else
% Alert user that no blobs were found.
uiwait(msgbox('No blobs of given color were found in the image'),
'Error', 'error')
end
%% Compute how many % of the image the result takes
OrigImageArea = rows*columns % pxls ... 100%
AreaOfChosenColor = sum(areas(:,1)) % ... x %
Area_Procent = (AreaOfChosenColor/OrigImageArea)*100
uiwait(msgbox(sprintf('%s %0.5g %s','The chosen color covers ',
Area_Procent, ' % of the whole image. '), 'Results', 'modal'))
%% Plot only the resulted masked image - change the mask color to
white
F100 = figure(101);
imshow(maskedRGBImage);
% caption = sprintf(['Masked Original Image. Given color covers ',
num2str(Area_Procent), ' % of the area']);
% title(caption, 'FontSize', fontSize);
    T101 = title(['Masked Original Image. Given color covers ',
num2str(Area_Procent), ' % of the area.']);
set(T101, 'FontSize', fontSize);
    set(F100, 'OuterPosition', pos1)

f2 = fullfile(pathname, DirectoryName);
fileNamePlot= strcat(f2, '\Figure_101_Res.png');
saveas(gcf,num2str(char(fileNamePlot)))

%% save into xls file

```

```

message = sprintf('Do you want to save the results?');
reply = questdlg(message, 'Save results?', 'Yes', 'No', 'Yes');
ifstrcmpr(reply, 'Yes')

XlsFilesave = fullfile(pathname, DirectoryName, 'BlobsResults');
RGBHeadings = {'Red band', 'Green band', 'Blue band'};

xlswrite(XlsFilesave, RGBHeadings, 'meanRGB', 'A1') ;
xlswrite(XlsFilesave, meanRGB, 'meanRGB', 'A2') ;

xlswrite(XlsFilesave, RGBHeadings, 'areas', 'A1') ;
xlswrite(XlsFilesave, areas, 'areas', 'A2') ;

GeneralSetting = [numberOfBlobs, redThresholdLow, redThresholdHigh,
greenThresholdLow, greenThresholdHigh, blueThresholdLow,
blueThresholdHigh];
Headings = {'numberOfBlobs', 'redThresholdLow', 'redThresholdHigh',
'greenThresholdLow', 'greenThresholdHigh', 'blueThresholdLow',
'blueThresholdHigh'};
xlswrite(XlsFilesave, GeneralSetting, 'GeneralSetting', 'A2') ;
xlswrite(XlsFilesave, Headings, 'GeneralSetting', 'A1') ;
xlswrite(XlsFilesave, {'Procents of the image covered by the chosen
color:'}, 'GeneralSetting', 'A4') ;
xlswrite(XlsFilesave, Area_Procent, 'GeneralSetting', 'A5') ;

xlswrite(XlsFilesave, {'Minimale area [pixel] of the
blobs'}, 'GeneralSetting', 'A7') ;
xlswrite(XlsFilesave, answer2, 'GeneralSetting', 'A8') ;
end
%% Displaying the end of the computation
disp('*****')
disp('*****          Analysis has finnished          *****')
disp('*****          Check the xls file for results          *****')
disp('*****')

%% Stop writing in the file
diary off

```

COLOR DETECTION

```

% Demo macro to very, very simple color detection in RGB color space
% by ImageAnalyst
clc;
close all;

% Read standard MATLAB demo image.
rgbImage = imread('butterfly_after.jpg');

% Display the original image.
subplot(3, 4, 1);
imshow(rgbImage);
title('Original RGB Image');

% Maximize figure.
set(gcf, 'Position', get(0, 'ScreenSize'));

% Split the original image into color bands.
redBand = rgbImage(:, :, 1);
greenBand = rgbImage(:, :, 2);
blueBand = rgbImage(:, :, 3);

```

```

% Display them.
subplot(3, 4, 2);
imshow(redBand);
title('Red band');
subplot(3, 4, 3);
imshow(greenBand);
title('Green band');
subplot(3, 4, 4);
imshow(blueBand);
title('Blue Band');

% Threshold each color band.
redthreshold = 68;
greenthreshold = 70;
bluethreshold = 72;
redMask = (redBand>redthreshold);
greenMask = (greenBand<greenthreshold);
blueMask = (blueBand<bluethreshold);

% Display them.
subplot(3, 4, 6);
imshow(redMask, []);
title('Red Mask');
subplot(3, 4, 7);
imshow(greenMask, []);
title('Green Mask');
subplot(3, 4, 8);
imshow(blueMask, []);
title('Blue Mask');

% Combine the masks to find where all 3 are "true."
redObjectsMask = uint8(redMask&greenMask&blueMask);
subplot(3, 4, 9);
imshow(redObjectsMask, []);
title('Red Objects Mask');
maskedrgbImage = uint8(zeros(size(redObjectsMask))); % Initialize
maskedrgbImage(:, :, 1) = rgbImage(:, :, 1) .* redObjectsMask;
maskedrgbImage(:, :, 2) = rgbImage(:, :, 2) .* redObjectsMask;
maskedrgbImage(:, :, 3) = rgbImage(:, :, 3) .* redObjectsMask;
subplot(3, 4, 10);
imshow(maskedrgbImage);
title('Masked Original Image');

```

SHPAE DETECTION

```

% Shapes Classifier
% Step 1: Read image Read in
% Step 2: Convert image from rgb to gray
% Step 3: Threshold the image
% Step 4: Invert the Binary Image
% Step 5: Find the boundaries Concentrate
% Step 6: Determine Shapes properties
% Step 7: Classify Shapes according to properties
% Square      = 3
% Rectangular = 2
% Circle      = 1
% UNKNOWN     = 0
%-----
----
function W = Classify(ImageFile)

```

```

% Step 1: Read image Read in
RGB = imread('butterfly_after.jpg');
figure();
subplot(2,3,1)
imshow(RGB)
title('Original Image');

% Step 2: Convert image from rgb to gray
GRAY = rgb2gray(RGB);
subplot(2,3,2)
imshow(GRAY)
title('Gray Image');

% Step 3: Threshold the image Convert the image to black and white in
order
% to prepare for boundary tracing using bwboundaries.
threshold = graythresh(GRAY);
BW = im2bw(GRAY, threshold);
subplot(2,3,3)
imshow(BW)
title('Binary Image');

% Step 4: Invert the Binary Image
BW = ~ BW;
subplot(2,3,4)
imshow(BW)
title('Inverted Binary Image');

% Step 5: Find the boundaries Concentrate only on the exterior
boundaries.
% Option 'noholes' will accelerate the processing by preventing
% bwboundaries from searching for inner contours.
[B,L] = bwboundaries(BW, 'noholes');

% Step 6: Determine objects properties
STATS = regionprops(L, 'all'); % we need 'BoundingBox' and 'Extent'

% Step 7: Classify Shapes according to properties
% Square = 3 = (1 + 2) = (X=Y + Extent = 1)
% Rectangular = 2 = (0 + 2) = (only Extent = 1)
% Circle = 1 = (1 + 0) = (X=Y , Extent < 1)
% UNKNOWN = 0

subplot(2,3,5)
imshow(RGB)
title('Results');
hold on
for i = 1 : length(STATS)
    W(i) = uint8(abs(STATS(i).BoundingBox(3)-STATS(i).BoundingBox(4)) <
0.1);
    W(i) = W(i) + 2 * uint8((STATS(i).Extent - 1) == 0 );
    centroid = STATS(i).Centroid;
    switch W(i)
    case 1
        plot(centroid(1),centroid(2), 'wO');
    case 2
        plot(centroid(1),centroid(2), 'wX');
    case 3
        plot(centroid(1),centroid(2), 'wS');
    end
end

```

```
end  
return
```