Image Processing in MATLAB with Graphical User Interface (GUI) Environment

A Thesis Report

By

Abdulla Al Mamun

Roll No: ASH1201009M

Session: 2011-2012

&

S.M. Shafkat Islam

Roll No: ASH1201064M

Session: 2011-2012

Department of Computer Science & Telecommunication Engineering

Submitted in partial fulfillment of the requirements

For the degree of

(B.Sc Engineering in CSTE)

Under The Guidance

Of

ABHIJIT CHAKRABORTY

Assistant Professor, Department of CSTE

NOAKHALI SCIENCE & TECHNOLOGY UNIVERSITY

SONAPUR, NOAKHALI-3802

February 2017

The undersigned have examined the thesis entitled '**Image Processing in MATLAB with Graphical User Interface(GUI) mechanisms**' presented by Roll No: ASH1201009M & Roll No: ASH1201064M candidates for the degree of Bachelor of Science in Computer Science & Telecommunication Engineering as B.Sc. Engineering (CSTE) and hereby certify that it is worthy of acceptance.

Board of Examiners

1.

Department of Computer Science &
Telecommunication Engineering
Noakhali Science & Technology University

Chairman

2.

Department of Computer Science &
Telecommunication Engineering
Noakhali Science & Technology University

1st Examiner

3.

Department of Computer Science &
Telecommunication Engineering
Noakhali Science & Technology University

2nd Examiner

Statement of Originality

It is hereby declared that the content of this project is original and any part of it has not been submitted elsewhere for the award of any degree or diploma of the university or other institute of higher learning**.**

Signature of the Supervisor
Date:

Signature of the Candidate
Date:

Signature of the Candidate
Date:

ACKNOWLEDGEMENT

First of all, we would like to show our gratitude to Almighty ALLAH who gives us strength and opportunity to complete the project '**Image Processing in MATLAB with Graphical User Interface (GUI) environment'.**

We are very grateful to our project supervisor Abhijit Chakraborty, Assistant Professor, Department of Computer Science & Telecommunication Engineering, NSTU, for his endless support, inspiration, encouragement, contractive criticism and suggestion throughout this work. His advice and overall guidance helped us a lot in advance of the project.

Further, we would like to thank all of our teachers, Department of Computer Science & Telecommunication Engineering, NSTU, for their encouragement.

At last we would like to thank our parents and our friends who helped us through their appreciation, encouragement and criticism.

ABSTRACT

Image processing applications are widely used day by day. This paper presents information on wide aspects of the computer graphics, introduction to MATLAB and its Image Processing Toolbox.

Later, the thesis focuses on object detection from different images with training & Vanish Object. An algorithm for the design object detection & removing object systems is presented. Cascade classifier is utilized as the algorithms for this object detection system. Development of object detection is categorized into two phase which are training phase and execution phase. And also, Patch based in paint is utilized as the algorithms for removing object system. All theoretical studies are followed by an implementation of an image processing applications. Here MATLAB with GUI environment is used to find & remove the different features of the object of interest from random images.

LIST OF CONTENTS

Chapter Page Number

Appendix

CHAPTER 1

INTRODUCTION

1.1 Introduction

Recently people got interested in the possibilities of information technology and they have noticed that computer can help them with daily tasks. This need has motivated software programmers to create new systems that would incorporate with the effectiveness of work. Those big changes have influenced also computer related fields, including computer graphics as well as an image processing.

Image Processing involves changing the nature of an image for interprets image data to human understanding. The Image Processing Application is the largest one and therefore offers the greatest number of image modifications. We can also improve the quality of an image by image processing. A huge number of applications have been developed due to fulfill our daily basis needs such as motion estimation and tracking for traffic systems, number of affected cancer cell detection for treatment, pixel values analysis for animation etc. This paper focuses on image processing applications besides edge detection, object detection and recognition and finally introducing with GUI environment for creating application interface.

There are a number of skills applied for image processing such as Matlab, OpenCV, C# etc. Among them Matlab is most user interactive tool for doing numerical computations with matrices and vectors. It can also display information's graphically. The Image Processing Toolbox (IPT) provides a comprehensive set of functions for image manipulation, analysis, digital imaging, and computer graphics. It provides engineers and scientists with an extensive suite of robust digital image processing and analysis functions. GUIDE or the Graphical User Interface development environment is a tool for laying out and Programming GUI's. A GUI uses graphics and text input to make using Matlab much more user friendly for people who are unfamiliar with it .

1.2 Why do we need image processing?

Image processing refers to processing image by means of a computer. Image is composed of a finite number of elements, each of which has a particular location & value. These elements are referred to as picture elements, image elements & pixels. The key element that distinguishes image processing (or digital image processing) from computer graphics is that image processing generally begins with images with image spaces and performs pixel based operations on them to produce new image that exhibits certain desired features. Images can be represented in different ways and formats. For example a RGB image is consists of three colors; red, green, blue. A grayscale image consists of two colors; black and white. The formats of image representation are JPEG, PNG, GIF, PSD etc. Image processing involves all types of images. But sometimes it may causes problems with specific types of image formats.

For the graphical representation of three dimensional [3D] image & make the image as real image, Image processing is highly used. It is also used for better pictorial views .

1.3 Edge Detection Techniques

Image Edge detection significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image. Since edge detection is in the forefront of image processing for object detection, it is crucial to have a good understanding of edge detection algorithms.

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The Discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There are an extremely large number of edge detection operators available, each designed to be sensitive to certain types of edges. They are Sobel, Prewitt, Robert, Canny etc. In this paper we have discussed on Sobel and Prewitt Operator [2].

1.3.1 Canny operator

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.

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hystresis Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

1.4 Object detection Techniques

Object detection is commonly referred to as a method that is responsible for discovering and identifying the existence of objects of a certain class. An extension of this can be considered as a method of image processing to identify objects from digital images.

1.5 Introduction of Haar-like features

A more sophisticated method is therefore required. One such method would be the detection of objects from images using features or specific structures of the object in question.

However, there was a problem. Working with only image intensities, meaning the RGB pixel values in every single pixel in the image, made feature calculation rather computationally expensive and therefore slow on most platforms.

This problem was addressed by the so-called Haar-like features, developed by Viola and Jones on the basis of the proposal by Papageorgiou et. al in 1998.

A Haar-like feature considers neighbouring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image.

An example of this would be the detection of human faces. Commonly, the areas around the eyes are darker than the areas on the cheeks. One example of a Haar-like feature for face detection is therefore a set of two neighbouring rectangular areas above the eye and cheek regions.[3]

1.6 Cascade Classifier

The cascade classifier consists of a list of stages, where each stage consists of a list of weak learners. The system detects objects in question by moving a window over the image. Each stage of the classifier labels the specific region defined by the current location of the window as either positive or negative –positive meaning that an object was found or negative means that the specified object was not found in the image.[4] [5]

Problem Definition

- The cascade classification method have no limitation of kind of objects it can detect as long as there is some pattern which cascade can figure out during its training phase. It is these features or pattern which trained cascade will try searching in image, when performing object detection. Definately, having more distinct features aid the cascade perform classification task better.

- The variation in intensity has its own advantages and disadvantages. Including images with intensity variation depends also on the environment in which I want my cascade to be able to detect objects. Eg: If I want cascade to detect objects in varied environments such as dark/light background, during day/night and other such factors it would be advisable to have intensity variations included. Again having large intensity variations may result in cascade not being able to detect patterns specific to object resulting in poor performance. However, if the cascade is expected to detect objects under a specific condition, training it under that specific condition without including much intensity variation will prove more fruitful.

- Having individual samples using different background will make cascade more robust i.e it will better ability to detect object in complex backgrounds. Generating positive images from videos wont provide cascade with background variations, though it will serve purpose of having more positive samples required for training cascade.

1.7 Object Removing Techniques

The idea of Removal is remove object(s) from digital photographs, and then fill the hole with information extracted from the surrounding area. Filled region should look “reasonable” to the human eyes. Recovering lost part of an image plays a great role in image processing. Inpainting is a technique that helps in recovering lost pixels from an image. From the existing techniques of Inpainting, Exemplar Inpainting is one of the fast and better techniques that help in restoring the lost part of an image. Exemplar based method chooses a patch similar to the lost patch from the known area to fill in the occluded surface. This paper proposes a modified exemplar based Inpainting algorithm for restoring the unknown pixels in a lost region in an image.[6]

1.8 Exemplar based inpainting

One of the important works in Exemplar based image inpainting is carried out by Criminisi algorithm. This algorithm combines the advantage of texture synthesis method and PDE-based inpainting method. This method proposed in this work consists of the following major steps for filling the missing region.

- Counts the known pixels that belongs to the edge and perform edge detection.
- As a second step, the distance is computed between the patch of missing region and the entire known region is calculated.

Figure 1 gives an introduction to the terms used in the algorithm.

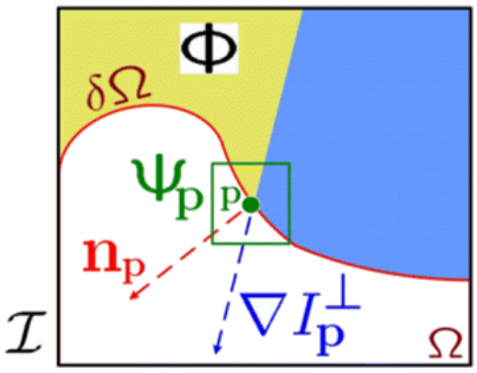

Fig 1.1: Demonstration of the terms used to represent elements of occluded region and known region.

Consider I as the whole image that includes both the known region and the occluded region that is to be filled. Let Ω represent the target region where the pixels are missing and need to be filled. Φ represents the source region from which a most similar patch is selected to fill the target region. δΩ represent the contour of the region to be filled. This can also be known as fill front. The pixels are selected from the fill front and are diffused inwards. Consider the following figure, Figure 2 that clearly explains the procedure followed in filling pixels using the above method.

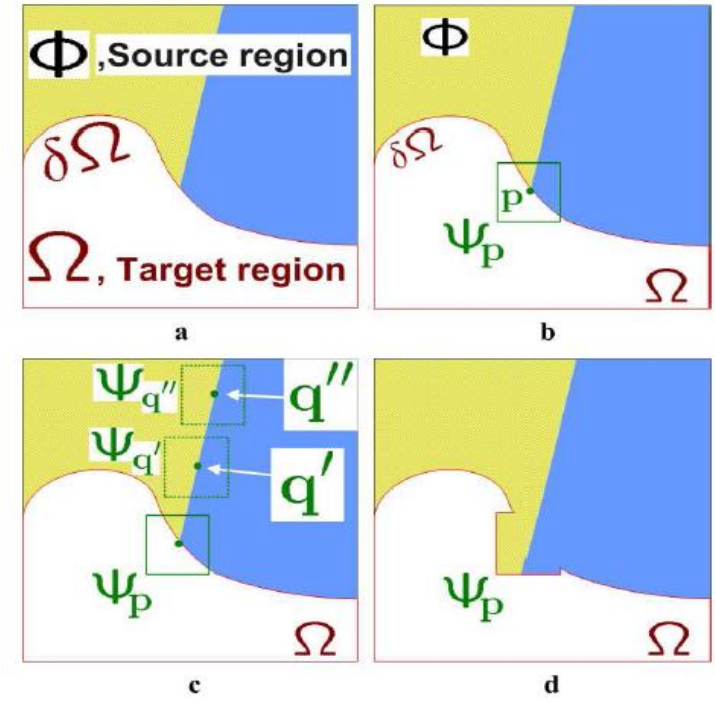

Fig 1.2: Basic steps to show the process of filling the occluded region using Criminisi's algorithm.

The square template Ψp is the patch centered at p, belongs to Ω whose center is at P that is to be filled. If Ψp lies in the continuation of an edge, the most similar match Ψq ϵ Φ also lies in the same edge.[7][8][

1.9 Objectives

- We will work with data representation & their library in Matlab environment.
- Object detection can be also used for people counting, it is used for analysing store performance or crowd statistics during festivals.
- Object detection is also used in industrial processes to identify products. Say if we want our machine to only detect circular objects. Hough circle detection transform can be used for detection.
- We can be used object removing system for removing unwanted image or recover pixel loss in images.
- In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quadcopter).

1.10 Summery

Image processing operates on images and results in images, while image analysis obtains numerical data from images. In this chapter we have introduced with objectives of our works. Next chapter we will discuss on methodology. MATLAB is built around a programming language, and as such it's really designed with tool-building in mind. Guide extends MATLAB's support for rapid coding into the realm of building GUIs. With the help of MATLAB we have developed GUI interface to detect objects from image. Detection offers numerous application in our daily basis life.

Chapter 2

METHODOLOGY

Methodology refers to the study of methods. Methodology is the systematic, theoretical analysis of the methods applied to a field of study or the theoretical analysis of the body of methods & principals associated with the branch of knowledge.

Initializing MATLAB

- Matlab's library Image Processing Toolbox has mostly found usefulness in mathematical problems.
- The Command Window is the window on the right hand side of the screen. This window is used to both enter commands for MATLAB to execute, and to view the results of matlab commands.
- An M-file is a MATLAB document the user creates to store the code they write for their specific application. Creating an M-file is highly recommended, although not entirely necessary. An M-file is useful because it saves the code the user has written for their application. It can be manipulated and tested until it meets the user's specifications. The advantage of using an M-file is that the user, after modifying their code, must only tell MATLAB to run the M-file, rather than reenter each line of code individually.

Read Images

- The function **imread** is used to read an image file with a specified format. The function **imshow** displays an image, while figure tells MATLAB which figure window the image should appear in. The function imtool for image tools.

- To initialize an image on which the operators (prewitt, sobel) will be applied needs to store in c://matlab/work. This operator is vastly used for edge detection. If the image that you have is in color, but color is not important for the current application, then you can change the image to grayscale. This makes processing much simpler since then there are only a third of the pixel values present in the new image.

Train Object Detector

- Cascade classifier training requires a set of positive samples and a set of negative images.
- Need to provide a set of positive images with regions of interest specified to be used as positive samples. And also need to provide a set of negative images from which the function generates negative samples automatically. To achieve acceptable detector accuracy, set the number of stages, feature type, and other function parameters.
- **trainCascadeObjectDetector** is a function that export a XML file of object structure model.[4]

Detect Object

- The cascade object detector uses the Viola-Jones algorithm to detect object using output xml file.

- **vision.CascadeObjectDetector** creates a System object, detector, and configures it to use the custom classification model specified with the XMLFILE input.

- **BBOX** is the function to find out the boundary of detected object from the images.

- Given input image, determine which objects it contains, and possibly a transformation of the object, based on predetermined interest points.

Display the detected object

- Get the bounding polygon of the reference image. Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

Exemplar based object removal using Inpainting

- Before removing the area of the image first we need to select the area. Here we use **ginput** function for selection.
- Initial occluded region is detected and are represented with the data structures that are appropriate to the region.
- A predefined function is used to compute the filling order for all the missing pixels as the first step in each of the iterations.
- From the source region, the most matching block is searched to fix the given occluded regions block with a pixel.
- Boundary of the target region and the information for computing filling priorities are updated. [7]

Introducing GUI's

- GUI's can be created without using GUIDE but laying out the design of the window can be very time consuming. To open GUIDE we have to click on the start button in the bottom left corner of Matlab and select START! MATLAB! GUIDE or by clicking.

- We first lay out the basic controls for our research, selected from the menu along the left side: axes, static text, edit box, and a button.

- We have to define a callback for the button so it will plot the function when we press it.

- When you run the program, it creates two files.

  1)gui.fig -- contains the layout.

  2)gui.m -- contains code that defines a callback function.

CHAPTER 3

DESIGN & IMPLEMENTATION

3.1 For the development of the project we've considered so many tools they are in the following:

3.2 Tools

- ✓ Matlab R20013a
- ✓ Matlab Compiler
- ✓ Image Processing Toolbox
- ✓ Graphical User Interface(GUI)

3.2.1

Matlab R20013a: MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by Math Works, 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++, Java, and Fortran.

3.2.2

Matlab Compiler: The MATLAB Compiler Runtime (MCR) is a standalone set of shared libraries that enables the execution of compiled MATLAB applications or components on computers that do not have MATLAB installed. When used together, MATLAB, MATLAB Compiler, and the MCR enable you to create and distribute numerical applications or software components quickly and securely.

3.2.3

Image Processing Toolbox: Image Processing Toolbox provides a comprehensive set of reference-standard algorithms, functions, and apps for image processing, analysis, visualization, and algorithm development. We can perform image enhancement, image deblurring, feature detection, noise reduction, image segmentation, geometric transformations, and image registration. Many toolbox functions are multithreaded to take advantage of multicore and multiprocessor computers.

Image Processing Toolbox supports a diverse set of image types, including high dynamic range, gigapixel resolution, embedded ICC profile, and tomographic. Visualization functions let you explore an image, examine a region of pixels, adjust the contrast, create contours or histograms, and manipulate regions of interest (ROIs). With toolbox algorithms you can restore degraded images, detect and measure features, analyze shapes and textures, and adjust color balance.

3.2.4

Graphical User Interface (GUI): A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called components that enable a user to perform interactive tasks. The user of the GUI 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 of a GUI need not understand the details of how the tasks are performed.

GUI components can include menus, toolbars, push buttons, axes, guinput, label, and panel—just to name a few. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots.

3.3 Canny Edge Detection Operator

An improved algorithm based on frame difference and edge detection is presented for moving object detection. First of all, it detects the edges of each two continuous frames by Canny detector and gets the difference between the two edge 8 images. And then, it divides the edge difference image into several small blocks and decides if they are moving areas by comparing the number of non-zero pixels to a threshold. At last, it does the block-connected component labeling to get the smallest rectangle that contains the moving object.[9]

3.3.1 Snapshot of BW image with pixel values after applying Canny Operator:

Fig 3.1– Black and white Image

Fig 3.2– Apply Canny edge detection.

3.4 Design of the project in the matlab environment

Since we are going to identify the object from any images, we have to consider some image where we can find trained object. First we have to train detector. After finding the values of the pixel we are going to identify their distances of special pixel values so that we can identify the object in the images. Then remove object using exemplar base image inpainting Therefore in the following we have shown the details that how can we train detector, object detection , remove and so on.

3.4.1: Snapshot of simple layout using GUI:

Fig 3.3– A simple layout in GUI environment

Figure 3.3 shows simple layout using GUI. When you run the program, it creates two files.

gui.fig -- contains the layout.

#gui.m -- contains code that defines a callback function.

3.5 Train detector using Cascade Trainer

- The cascading(multiple stages of classifying an ROI) classifier is trained such that it detects a specified region that bounds the object of interest. Cascade trainer requires positive and negative image database.
- The positive images are those images that contain the object (e.g. face or eye), and negatives are
- those ones which do not contain the object.
- Train the classifiers with image database, here we use 'trainCascadeObjectDetector' function and feed images into the proper arguments (Positive, negative images). The output classifier will be 'outputXMLFilename' as in traincascadeobjectdetector.

3.5.1: Snapshot of train detector (Sohid Minar)

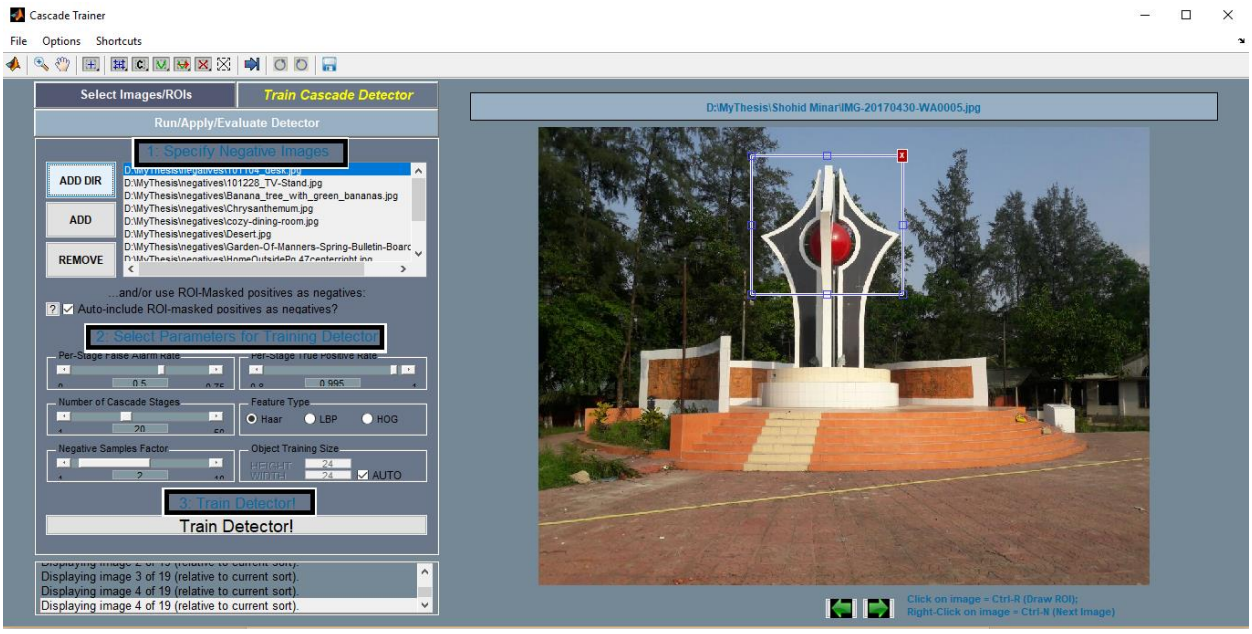

Fig 3.4– Train object (Sohid Minar)

Figure 3.4 shows the training process of object (sohid minar).

3.6 Design Object Detection System using Cascade Classifier

- After training Cascade Classifier use 'outputXML' file to detect Object.
- **`detector = vision.CascadeObjectDetector(XMLFILE)`** creates a System object, `detector`, and configures it to use the custom classification model specified with the `XMLFILE` input. We have to specify a full or relative path to the `XMLFILE`, if it is not on the MATLAB® path.
- **`BBOX = step(detector,img_to_test)`** returns **`BBOX`**, an *M*-by-4 matrix defining *M* bounding boxes containing the detected objects. This method performs multi-scale object detection on the input image, **`img_to_test`**.
- After detecting object we show it on axes.

3.4.1: Snapshot of object detection

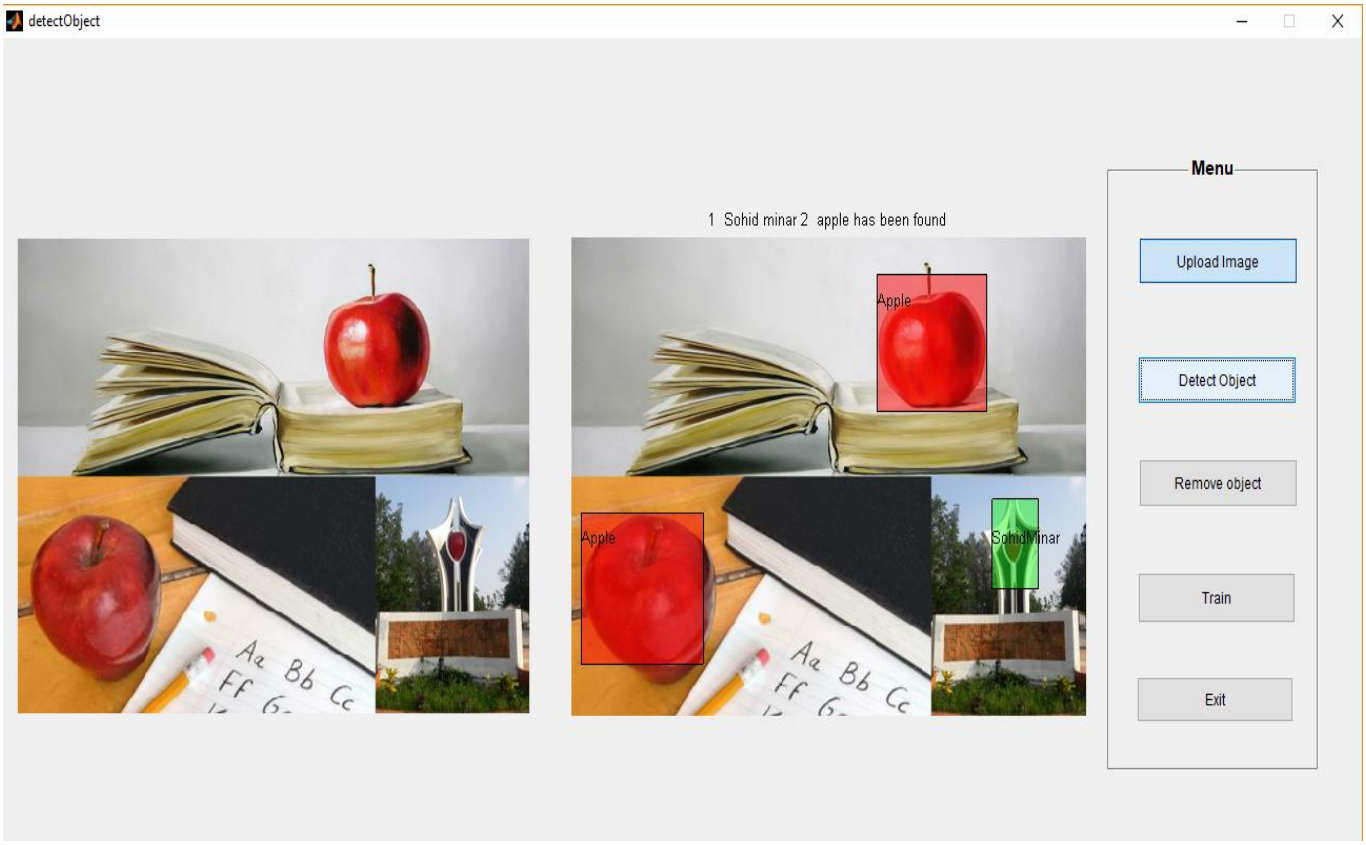

Fig 3.5– Object Detection (Shohid Minar & apple)

Figure 3.5 shows the object detection. Here 2 apple and 1 sohid minar is detected. Before detection we first train Cascade Trainer with **shohidminar** & **apple** images. After that we get two output xml file (shohidminarXml and appleXml). Computer Vision System Toolbox offers functionality for object detection that include cascadeObjectDetector, BBOX etc. cascadeObjectDetector take those xml as input to detect apple and shohid minar. And BBOX return number of detected object and also the region of the detected object. And Patch the detected region with green rectangle with 0.4 opacity. Here detector find 2 apple and 1 sohid minar. Then we show that on axes.

3.7 Design Object Removing System using inpainting

Inputs: Image and exemplar of the object to be removed

Fig 3.6– Input Image for object removing

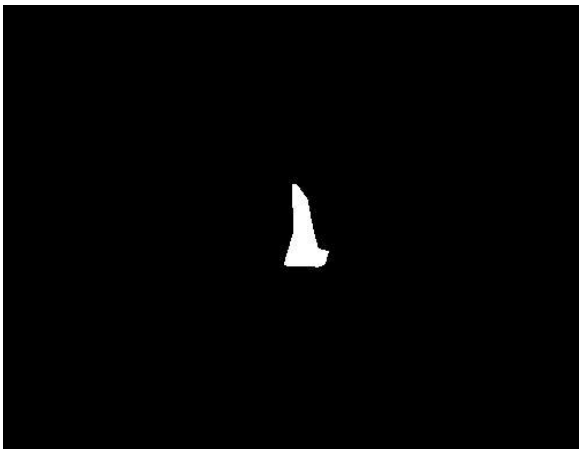

Fig 3.7– a Exemplar of object

Figure 3.6 shows that input image. Here we use **uigetfile** to input images and shows it on axes.

In figure 3.7 shows the target mask. Here we use **ginput** for selecting the desired object. **ginput** return the axis point. Then we use **poly2mask** to make target mask . After that we send it to inpaint function as parameter to remove the object.

Remove the object (exemplar) from the original Image.

Once we remove the object of interest from the input image, there will be unfilled pixels which are to be filled using inpainting.

Find the fill front.

Fill front can be found by using any edge detection algorithm like “Laplacian based edge detection” or “canny”.

Find the priority of the pixels along the fill front using confidence and data terms.

Once the fill front is found each pixel on the fill front is given a priority value that is the order in which they are to be filled. To calculate the priority value at each pixel two terms are to be computed.

Fill the unfilled pixels from the similar patch obtained.

Once the priorities are calculated take the highest priority pixel and extract a 9x9 patch around the pixel and look for similar patches in the source region (ϕ) and fill the unfilled pixels using the similar patch found.

Update the confidence value at the pixel which is filled.

3.7.1: Snapshot after removing Object

Fig 3.8– After Removing Object

Figure 3.8 shows the inpainted image. Where selected is removed. Here **inpaintingfw** function take **input image** and **target mask** as parameter. Then find the priority of the pixels along the fill front using confidence and data terms. Fill the unfilled pixels and update the confidence value.

4 FUTURE PLAN

In this paper we tried to detect various types of objects from any images. Our future plan is to detect undefined objects from randomly selected image and make self-decision using neural network.

5 CONCLUSION

A simple image processing application can become a complex piece of a new software. After wide studies of topic-related areas the prototype of an image processing application was designed and implemented. During the first stage of the development, the Graphical User Interface was created; it required careful consideration of the purpose of the program, in order to place elements correctly. The proposed algorithm is for detecting a specific object based on finding point correspondences between the reference and the random image. This method of object detection works best for rectangular object of interest. Inpainting based object removal doesn't work well if the object is large.

There are still a lot of possibilities for future development of our work.

Appendix

i. List of Figures

ii. Source codes

a) Edge Detection by MATLAB

```
Img=imread('shohidMinar.jpg');
imshow(b);
BW2 = edge(b,'canny');
b=im2bw(Img);figure,imshow(BW2);
```

b) User Interface

```
function varargout = detectObject(varargin)
warning off;
%DETECTOBJECT M-file for detectObject.fig
%      DETECTOBJECT, by itself, creates a new DETECTOBJECT or raises the
existing
%      singleton*.
%
%      H = DETECTOBJECT returns the handle to a new DETECTOBJECT or the
handle to
%      the existing singleton*.
%
%      DETECTOBJECT('Property','Value',...) creates a new DETECTOBJECT
using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to detectObject_OpeningFcn.  This calling syntax produces
a
%      warning when there is an existing singleton*.
%
%      DETECTOBJECT('CALLBACK') and DETECTOBJECT('CALLBACK',hObject,...)
call the
%      local function named CALLBACK in DETECTOBJECT.M with the given
input
%      arguments.
%
%      *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 detectObject

% Last Modified by GUIDE v2.5 15-May-2017 12:19:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @detectObject_OpeningFcn, ...
                   'gui_OutputFcn',  @detectObject_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 detectObject is made visible.
function detectObject_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   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = detectObject_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 uploadPushbutton.
function uploadPushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to uploadPushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[FileName,PathName] = uigetfile('*','Select Image to Test');
image_path = strcat(PathName,FileName);
img_to_test = imread(image_path);
[m n]=size(img_to_test);
```

```
if or(m>700,n>700)
if(m>n)
img_to_test=imresize(img_to_test,1500/m);
else
img_to_test=imresize(img_to_test,1500/n);
end
end
axes(handles.axes4);
imshow(img_to_test);
title('Image has been uploaded successfully');
cla(handles.axes5,'reset');
handles.img=img_to_test;
guidata(hObject,handles);


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


% --- Executes on button press in objectDetectPushbutton.
function objectDetectPushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to objectDetectPushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isfield(handles,'img')
% figure,imshow(handles.img);
b=uint8(handles.img);

% axes(handles.axes5);
imshow(b);
TestClassifier1(handles,b);
end


% --- Executes on button press in removeObjectpushbutton.
function removeObjectpushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to removeObjectpushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes5,'reset');
set(handles.axes5,'xtick',[],'ytick',[]);

Img_in=uint8(handles.img);
b=num2str(rand);
pathname='D:\MyThesis\input\input';
name= strcat(pathname,b,'.jpg');
imwrite(Img_in,name);
% Img_in=imresize(Img_in,[200,200]);
% figure,imshow(Img_in);
[x, y] = ginput;
```

```
% Img_in=imresize(Img_in,[200,200]);
target_mask = poly2mask(x, y, size(Img_in, 1), size(Img_in, 2));

I = im2uint8(target_mask);
template = cat(3, I, I, I);

pathname='D:\MyThesis\input\mask\mask';
name= strcat(pathname,b,'.jpg');
imwrite(template,name);
% imwrite(I,'Test12.tif');
% template = imread('Test12.tif');

[inpaintedimg] = inpaintingfw(handles,Img_in,template);


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

c) Cascade Object Detection

```
function [detected_object] = TestClassifier1(handles, img_to_test )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
sohidMinar_xml_file='SohidMinar.xml';
sohidMinar_detector=vision.CascadeObjectDetector(sohidMinar_xml_file);

% Find bbox of any detected objects
sohidMinar_bbox = step(sohidMinar_detector,img_to_test);

% number of detected sohid minar
sohidMinar_nDetected = size(sohidMinar_bbox,1);
% figure;
% imshow(img_to_test);
% hold on;

axes(handles.axes5);
imshow(img_to_test);


% imshow(img_to_test,handles.axes5);


for ii = 1:sohidMinar_nDetected
patch([sohidMinar_bbox(ii,1),sohidMinar_bbox(ii,1)+sohidMinar_bbox(ii,3),s
ohidMinar_bbox(ii,1)+sohidMinar_bbox(ii,3),sohidMinar_bbox(ii,1),...
sohidMinar_bbox(ii,1)],[sohidMinar_bbox(ii,2),sohidMinar_bbox(ii,2),sohidM
inar_bbox(ii,2)+sohidMinar_bbox(ii,4),...
sohidMinar_bbox(ii,2)+sohidMinar_bbox(ii,4),sohidMinar_bbox(ii,2)],'g','fa
cealpha',.4);
text(sohidMinar_bbox(ii,1),sohidMinar_bbox(ii,2)+30,'SohidMinar')
```

```
end
sohidttl='';
if sohidMinar_nDetected>0
sohidttl=strcat(num2str(sohidMinar_nDetected),'  Sohid minar ');
end

% --happy DETECTOR -- %
% XML File name
apple_xml_file = 'apple.xml';
apple_detector = vision.CascadeObjectDetector(apple_xml_file,'MinSize',[100
100]);

% Find bbox of any detected objects
apple_bbox = step(apple_detector,img_to_test);

% remove any bounding boxes where the red is not the dominant color
apple_nDetected = size(apple_bbox,1);
counter = 0;
for i = 1:apple_nDetected
roi = img_to_test(apple_bbox(i,2):apple_bbox(i,2)+apple_bbox(i,4),...
      apple_bbox(i,1):apple_bbox(i,1)+apple_bbox(i,3),:);

roi = (rgb2ycbcr(roi));
roi=roi(:,:,3);
% YCBCR_mask = YCBCR(:,:,3)>143;

[row, col] = size(roi);
amount_red = sum(sum(roi>156)); %amount of red pixels
red_ratio = amount_red/(row*col);
if red_ratio < 0.54 ;
      %remove bounding box if less than 0.58 the pixels are red
      counter = counter+1;
      to_delete(counter) = i;
end
end
% delete columns that don't meet red threshold
try
apple_bbox(to_delete,:) = [];
end
apple_nDetected = size(apple_bbox,1);


%Show the results
% figure();
% imshow(img_to_test);
% hold on;

for ii = 1:apple_nDetected
patch([apple_bbox(ii,1),apple_bbox(ii,1)+apple_bbox(ii,3),apple_bbox(ii,1)
+apple_bbox(ii,3),apple_bbox(ii,1),apple_bbox(ii,1)],...
[apple_bbox(ii,2),apple_bbox(ii,2),apple_bbox(ii,2)+apple_bbox(ii,4),apple
_bbox(ii,2)+apple_bbox(ii,4),apple_bbox(ii,2)],...
'r','facealpha',0.5);
text(apple_bbox(ii,1),apple_bbox(ii,2)+20,'Apple')
end
```

```
applettl='';
if apple_nDetected>0
applettl=strcat(' ',num2str(apple_nDetected),'  apple ');
end
ttl=strcat(sohidttl,{' '},applettl,' has been found');
title(ttl);
end
```

d) Remove Object

```
function [inpaintedimg] =
inpaintingfw(handles,InputImg,TemplateImg)%fillfront approach
psize = 9;%%default patch size used
%InputImg = imread('Input1.png');
%TemplateImg = imread('template2.tif');

% colorTransform = makecform('srgb2lab');
% lab = applycform(InputImg, colorTransform);
% InputImg = lab;

Template_gray = rgb2gray(TemplateImg);
Template_bin = im2bw(Template_gray);
cntr = edge(Template_bin,'canny');%%or use log
Sourceimg = InputImg - TemplateImg;
SourceImg2 = Sourceimg;
% figure,imshow(SourceImg2)
for i=1:size(SourceImg2,1)
for j=1:size(SourceImg2,2)
      if cntr(i,j)==1
              SourceImg2(i,j,1)=255;
              SourceImg2(i,j,2)=0;
              SourceImg2(i,j,3)=0;
      end
end
end
% figure,imshow(SourceImg2)
%title('fill front in red')

%%Algorithm: This how the algorithm the proceeds
%The object which has to be removed from the original image has been
%removed and the Sourceimg has empty pixel which has to be filled with
best
%possible pixels so that human eye cannot make any difference. Treat the
%problem as evolving a fill front by giving a temporary priority value for
% each pixel in the fill front until all the pixels are filled.
```

```
%% create confidence image
disp('creating confidence Image')
padedimage = padarray(Sourceimg,[psize psize],'post');
NumofUnFildPix = numel(find(TemplateImg(:,:,:)==255));

disp('calculating isophotes')
Ix = zeros(size(InputImg));
Iy = zeros(size(InputImg));
[Ix(:,:,1) Iy(:,:,1)] = gradient(double(InputImg(:,:,1)));
[Ix(:,:,2) Iy(:,:,2)] = gradient(double(InputImg(:,:,2)));
[Ix(:,:,3) Iy(:,:,3)] = gradient(double(InputImg(:,:,3)));

Ix = sum(Ix,3)/765;%%summing along 3rd dimension
Iy = sum(Iy,3)/765;
temp = Ix;
Ix = -Iy;%%isophote in x
Iy = temp;%%isophote in y

% temp = Iy;
% Iy = -Ix;%%isophote in x
% Ix = temp;%%isophote in y
%figure,imshow(Ix)
%figure,imshow(Iy)
%%initializing confience image
confidenceImage = zeros(size(padedimage));
for i=1:size(confidenceImage,1)-psize
for j=1:size(confidenceImage,2)-psize
      if TemplateImg(i,j)==255
              confidenceImage(i,j,:)=0;
      else
              confidenceImage(i,j,:)=1;
      end

end
end
itrn = 1;
p_confid = cast(confidenceImage,'double');
%%%data image
dataimage = zeros(size(confidenceImage));

disp('inpaint start')
while NumofUnFildPix ~=0
%1.a Identify the fill front

fillfront_xlist = [];%%list of x locations of the fill front
fillfront_ylist = [];%% list of y locations of the fill front
in = 1;
in1 = single(rgb2gray(TemplateImg));
in2 = im2bw(in1);
cntr = edge(in2,'canny');
in3 = ~in1;
cntr2 = bwdist(in3);%%distance transform
for i=1:size(in1,1)
      for j=1:size(in1,2)
              if cntr(i,j)~=0
              fillfront_xlist(in) = i;
```

```
                fillfront_ylist(in) = j;
                in = in+1;
                end

        end
end
%compute the patch priorities
numoffillfrontpixls = numel(fillfront_xlist);
psin = (psize-1)/2;
cpixels = zeros(1,numoffillfrontpixls);
%confidence along the fill frontpixels
for i=1:numoffillfrontpixls
        confpatch = confidenceImage(fillfront_xlist(i)-
        psin:fillfront_xlist(i)+psin,fillfront_ylist(i)-
        psin:fillfront_ylist(i)+psin,:);
        val = sum(sum(sum(confpatch)))/numel(confpatch);
        cpixels(i)=val;
end
%%data term made up of the isotope values and normal values
ispopixels_x = zeros(1,numoffillfrontpixls);
ispopixels_y = zeros(1,numoffillfrontpixls);
for i=1:numoffillfrontpixls
        ispopixels_x(i)= Ix(fillfront_xlist(i),fillfront_ylist(i));
        ispopixels_y(i)= Iy(fillfront_xlist(i),fillfront_ylist(i));
end
nx = [];
ny = [];
[nx,ny] = gradient(double(in3));
normalpixels_x = zeros(1,numoffillfrontpixls);
normalpixels_y = zeros(1,numoffillfrontpixls);
for i=1:numoffillfrontpixls
        normalpixels_x(i)= nx(fillfront_xlist(i),fillfront_ylist(i))/255;
        normalpixels_y(i)= ny(fillfront_xlist(i),fillfront_ylist(i))/255;
end

datavalpixels_x = zeros(1,numoffillfrontpixls);
datavalpixels_y = zeros(1,numoffillfrontpixls);
datavalpixels_x = ispopixels_x.*normalpixels_x;
datavalpixels_y = ispopixels_y.*normalpixels_y;

Dval = abs(datavalpixels_x+datavalpixels_y)+0.001;
priorofpix = zeros(1,numoffillfrontpixls);
%cpixels = 0.5*cpixels+0.5;
%priorofpix = (cpixels+Dval)+0.001;
priorofpix = cpixels.*Dval+0.001;
idx = [];
ing = [];
[ing,idx] = sort(priorofpix,'descend');
%[ing,idx] = max(priorofpix(:));
for i=1:numoffillfrontpixls
toppixl = idx(i);
patch2 = padedimage(fillfront_xlist(toppixl)-
psin:fillfront_xlist(toppixl)+psin,fillfront_ylist(toppixl)-
psin:fillfront_ylist(toppixl)+psin,:);
patch3 = get_similar_patch(patch2,padedimage,TemplateImg,psize);
patch4 = zeros(size(patch2));
```

```
patch4 = cast(patch4,'uint8');
timage = [];
timage = padarray(TemplateImg,[9 9],'post');
patch_templ = timage(fillfront_xlist(toppixl)-
psin:fillfront_xlist(toppixl)+psin,fillfront_ylist(toppixl)-
psin:fillfront_ylist(toppixl)+psin,:);
for p=1:size(patch_templ,1)
      for q=1:size(patch_templ,2)
             if patch_templ(p,q,:)==0
             patch4(p,q,1)=patch2(p,q,1);
             patch4(p,q,2)=patch2(p,q,2);
             patch4(p,q,3)=patch2(p,q,3);
             else
             patch4(p,q,1)=patch3(p,q,1);
             patch4(p,q,2)=patch3(p,q,2);
             patch4(p,q,3)=patch3(p,q,3);
             end
      end
end
confidenceImage(fillfront_xlist(toppixl),fillfront_ylist(toppixl))=
cpixels(toppixl);

padedimage(fillfront_xlist(toppixl)-
psin:fillfront_xlist(toppixl)+psin,fillfront_ylist(toppixl)-
psin:fillfront_ylist(toppixl)+psin,:)= patch4;
end

%confidenceImage(fillfront_xlist(toppixl),fillfront_ylist(toppixl))=
cpixels(toppixl);
TemplateImg2 = zeros(size(InputImg));
for i=1:size(InputImg,1)
      for j=1:size(InputImg,2)
             if padedimage(i,j,1)==0 && padedimage(i,j,2)==0 &&
             padedimage(i,j,3)==0 && TemplateImg(i,j,1)==255 &&
             TemplateImg(i,j,2)==255 && TemplateImg(i,j,3)==255
             TemplateImg2(i,j,:)=255;
             end
      end
end
TemplateImg = TemplateImg2;
NumofUnFildPix = numel(find(TemplateImg(:,:,:)==255))

end
inpaintedimg = padedimage(1:size(InputImg,1),1:size(InputImg,2),:);
%imwrite(inpaintedimg,'birdsimage.tif')
% figure,imshow(inpaintedimg)
axes(handles.axes5);
imshow(inpaintedimg);
b=num2str(rand);
pathname='D:\MyThesis\Inpainted Images\Inpainted Images';
name= strcat(pathname,b,'.jpg');
imwrite(inpaintedimg,name);
title('After removing Object')
```

References

Books

[1] S. U. R. F. (SURF), “Herbet bay, andreas ess, tinne tuytelaars,luc van gool”, Elsevier preprint, 2008.

Helpful websites:

[2] https://www.mathworks.com/discovery/edge-detection.html

[3] https://en.wikipedia.org/wiki/Haar-like_features

[4] https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar Like_Classifiers_Step_by_Step.pdf

[5] https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html

[6] https://www.mathworks.com/help/images/ref/bwareaopen.html

[7] http://cs.brown.edu/courses/csci1950-g/results/final/eboswort

[8] http://ieeexplore.ieee.org/document/1323101

[9] https://en.wikipedia.org/wiki/Canny_edge_detector