Detection and Removal of Cracks in Digitized Paintings *Via* Digital Image Processing

By

**Nawafil Abdul Wahab Farajalla Ali**

B.Sc. in Computer Science
Faculty of Science and Engineering
University of Science and Technology (2001)

Post Graduate Diploma in Computer Sciences
Faculty of Engineering and Technology
University of Gezira (2011)

**A Dissertation**

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science

in

Computer Sciences

Faculty of Mathematical and Computer Sciences

**Supervisor: Dr. Mohammed Abdalla Elmaleeh**

**Co-Supervisor: Dr. Saif Eldin Fatooh Osman**

*June 2013*
Detection and Removal of Cracks in Digitized Paintings Via Digital Image Processing

Nawafil Abdul Wahab Farajalla Ali

Supervisors Committee:

Supervisor: Dr. Mohammed Abdalla Elmaleeh .................................

Co-Supervisor: Dr. Saif Eldin Fatooh Osman .................................

Date of Examination: 22/6/2013
Detection and Removal of Cracks in Digitized Paintings *Via* Digital Image Processing

Nawafil Abdul Wahab Farajalla Ali

Examination Committee:

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Mohammed Abdulla Elmaleeh</td>
<td>Chairman</td>
<td></td>
</tr>
<tr>
<td>Dr. Mohammed Hussien Mohammed</td>
<td>External Examiner</td>
<td></td>
</tr>
<tr>
<td>Dr. Majdi Bakr Mohammed</td>
<td>Internal Examiner</td>
<td></td>
</tr>
</tbody>
</table>

**Date of Examination:** 22/6/2013
Dedication

To My Beloved Mother
Acknowledgment

I would like to thank Dr. Mohammed Abdalla Elmaleeh, my supervisor or his patience, support and his positive way of guiding me through the dissertation.

I would like to thank my beloved sister for her selfless support and all my relatives and friends who gave me aid.

Finally I need to express my thanks and appreciation, to my special person in my life who’s constantly gives me full support and patience through my life.
Detection and Removal of Cracks in Digitized Paintings Via Digital Image Processing

Nawafil Abdul Wahab Farajalla Ali
Master of Science in Computer Sciences (22 June 2013)
Faculty of Mathematical and Computer Sciences
University of Gezira

Abstract

Many paintings, especially old ones, suffer from breaks in the substrate, the paint, or the varnish. These patterns are usually called cracks and be caused by aging, drying and mechanical factors. The appearance of cracks on painting deteriorates the perceived paintings quality. One can use digital image processing techniques to detect and eliminate the cracks on digitized paintings. The main objective of this study is to present the digital image processing technique can be applied to the virtual restoration of artistic paintings which serves many purposes. The methods implemented on this study are based on studying the digital image processing technique used for cracks identification and removal. One of the most important result focuses on separating cracks and applying interpolations techniques for the restoration of digitized painting. A possible solution would be using artificial intelligence to establish rules to identify the background by analyzing the pattern of the image and rate the cracks accordingly can be a better means of identifying cracks.
كشف وإزالة الشقوق في اللوحات الفنية ذات التنسيق الرقمي عبر معالجة الصور الرقمية

نوافل عبد الوهاب فرج الله
ماجستير العلوم في علوم الحاسوب (22 يونيو 2013)
كلية العلوم الرياضية والحاسوب
جامعة الجزيرة

الملخص

العديد من اللوحات الفنية القديمة، خاصة القديمة، تعاني من التشققات من الطلاء أو من الصور. تلك الأبواب دائما تدعو تشغيل وكيون سبب الشارقة، الجفاف والعوامل الميكانيكية. إن ظهور التشغيل على اللوحات الفنية يظهر من جودة اللوحات الفنية المصورة يمكن للمرء استخدام تقنية معالجة الصور الرقمية لكشف وإزالة الشقوق التي على اللوحات الفنية. الهدف الرئيسي لهذه الدراسة هو إمكانية تطبيق تقنية معالجة الصور الرقمية لدراست تشغيل التشغيل واللطخة من اللوحات الفنية ذات التنسيق الرقمي التي تقدم أغراض عديدة. الطرق التي ف מחieved لهذه الدراسة استندت على دراسة تقنية معالجة الصور الرقمية المستخدمة على الشقوق وإزالتها. واحدة من أهم النتائج التحكم على فصل الشقوق وتطبيق تقنيات زيادة لاستعادة اللوحات الفنية الرقمية. أحد الحلول الممكنة سوف تستخدم المشاكل الإضافية لبناء قواعد لتحديد الخصائص من خلال تحليل نظام الصورة ومعدل الشقوق وفقاً لذلك يمكن أن يكون وسيلة أفضل لتحديد الشقوق وإزالتها.
# Table of Contents

Dedication .................................................................................. i
Acknowledgement ...................................................................... ii
Abstract ..................................................................................... iii
Abstract in Arabic ....................................................................... iv
List of Figures .............................................................................. Vi

## Chapter 1: Introduction
1.1 Introduction ........................................................................... 1
1.2 Problem Statement .............................................................. 2
1.3 Objectives ............................................................................. 3
1.4 Methodology .......................................................................... 3
1.5 Organization of the Dissertation ........................................... 3

## Chapter 2: Literature Review
2.1 Introduction ........................................................................... 4
2.2 Crack Detection with Three Majeure Steps ......................... 4
   2.2.1 Using Modified Trimmed Mean Filter ......................... 8
   2.2.2 Using Weighted Median Filter ................................. 8
   2.2.3 Doing Iteration ....................................................... 9
2.3 The Radial Basis Function network ..................................... 9
2.4 Edge Detection technique .................................................... 10
2.5 Implementation Environment ............................................. 12
2.6 User Friendly Graphical User Interface ............................ 13
2.7 Common Knowledge .......................................................... 16
2.8 Buttons and Sliders ............................................................. 16
2.9 Axes .................................................................................... 17
2.10 Creating Menu ................................................................... 18
    Summery ............................................................................. 19

## Chapter 3: Design and Implementation
3.1 Introduction .......................................................................... 20
3.2 Background ........................................................................... 20
   3.2.1 Crack Detection ....................................................... 21
3.2.2 Separation of the Thin Dark Brush Strokes, which have been Misidentified as Cracks ........................................... 22
3.2.2.1 Semi-Automatic Crack Separation ........................................... 22
3.2.3 Crack Filling ........................................................................... 24
3.3 Methodology ........................................................................... 26

Chapter 4: Results and Discussions
4.1 Introduction ........................................................................... 29
4.2 The Maintenance ...................................................................... 29
4.3 System Testing ......................................................................... 29
   4.3.1 Module Testing ..................................................................... 29
   4.3.2 Integration Testing ................................................................. 30
   4.3.3 Validation Testing ................................................................. 30
4.4 Studies Case ............................................................................ 30
   4.4.1 Case Study1 ......................................................................... 31
   4.4.2 Case Study2 ......................................................................... 34
   4.4.3 Case Study3 ......................................................................... 36
   4.4.4 Case Study4 ......................................................................... 38
   4.4.5 Case Study5 ......................................................................... 40

Chapter 5: Conclusion and Recommendation
5.1 Conclusion ............................................................................. 42
5.2 Recommendation ..................................................................... 42

References ................................................................................ 44
Appendix .................................................................................. 46
List of Figures

2.1 Original Image with Cracks[14] ................................. 6
2.2 Original Image after top-hat Transforms [14] .................. 6
2.3 Stroke Image before using Gaussian Classifier ................. 7
2.4 Stroke Image after using Gaussian Classifier (the Circle Shows the Increasing Pixel) [14] ................................. 7
2.5 Removal of Dark Brush Strokes after using Gaussian classifier. 7
2.6 Sharpened Image [14] ........................................ 9
2.7 Radial Basis Function Networks[18] .......................... 10
2.8 Sharpened Image [18] ........................................ 11
2.9 Edge Detection Image [18] .................................. 11
2.10 Edge Strength Classification Map. .......................... 12
2.11 Example of graphical user interface with some of the components .................................................. 15
3.1 Semi-Automatic Crack Separation .............................. 24
3.2 Before Crack Filling ........................................ 25
3.3 After 1st Crack Filling Iteration .............................. 25
3.4 After n th Crack Filling Iteration ............................ 26
3.5 Data Flow Diagram ........................................ 27
4.1 Original Image[13] ........................................ 31
4.2 Crack Detection[13] ........................................ 32
4.3 Crack Filling [13] ........................................ 33
4.4 Original Image[14] ........................................ 34
4.5 Crack Detection[14] ........................................ 34
4.6 Crack Filling [14] ........................................ 35
4.7 Original Image[15] ........................................ 36
4.8 Crack Detection[15] ........................................ 36
4.9 Crack Filling [15] ........................................ 37
4.10 Original Image[16] ........................................ 38
4.11 Crack Detection[16] ........................................ 38
4.12 Crack Filling [16] ........................................ 39
4.13 Original Image[17] ........................................ 40
4.14 Crack Detection[17] ........................................ 40
4.15 Crack Filling [17] ........................................ 41
Chapter 1

Introduction
Chapter 1

Introduction

1.1 Introduction

This chapter presents a background of image definition, digital image, digitized painting and detection and removal of cracks, problem statement, objectives, methodology, and thesis structure.

An image is the representation of a two dimensional image as a finite set of digital value, called picture elements, each of which has a particular location. For each pixel, there is an associated number known as digital number or sample, which dictates the color and brightness for that particular pixel. So image can be defined as a two- dimensional function, \( f(x,y) \), where \( x \) and \( y \) are spatial(plane)” coordinates, and the amplitude of \( f \) at any pair of coordinates\((x,y)\) is called the intensity or gray level of the image at that point[1].

A digital image is composed of a finite number of elements called pixel, each of which has a particular location and value. This mean when \( x, y \) and the intensity values of \( f \) are all finite and discrete quantities [2].

Almost all graphics software deals with some ‘real or painted’ images that are captured using digital cameras or flatbed scanners. We need the image in digital form; to transform a continuous tone painted picture into digital form requires a digitizer. The two functions of digitizer are sampling and quantizing. Sampling captures evenly spaced data points to represent an digitized image. Since these data points are to be stored in a computer, they must be converted to a binary form. Quantization assigns each value a binary number [3].
Computer image have been “digitized”, a process which converts the real word color painted picture to be numeric computer data consisting of rows and columns of millions of colors samples measured from the original painted picture. In either case the image quality, color, brightness and the darkness of each tiny area seen by a sensor is “sampled” meaning the color value of each value areas is mastered and recorded as a numeric value which represents the color there. This process is called digitized the paint image. The data is organized into the same rows and columns to retain the location of each actual tiny paint picture area [4].

The main cause of crack is ageing, drying and mechanical factors, digital image processing technique that can detects and removes the cracks in paintings, through applying a method which identifies of missing or damaged pixel painting areas by filling in information from the neighboring pixel areas [5].

### 1.2 Problem Statement

The appearance of cracks on paintings deteriorates the perceived image quality. However, one can use digital image processing techniques to detect and eliminate the cracks on digitized paintings. These techniques consist of the following stages:

- Crack detection.
- Separation of the thin dark brush strokes, which have been misidentified as cracks.
- Crack filling (interpolation).
1.3 Objectives

The primarily objectives of this study are:

- To detect, identify and remove the cracks in digitized painting using digital image processing technique.
- To identified the cracks by thresholding the output of the top-hat transform.
- To detect the breaks which are wrongly identified as cracks and separate them using a semi-automatic procedure based on region growing.
- To restore the image using the order statistics filters.

1.4 Methodology

The methods implemented on this study are based on studying the digital image processing technique used for cracks identification and removal. It considers the classification of cracks into paints to aide in damage assessment. The implementations has been applied is MATLAB.A code is designed with in that language.

1.5 Organization of the Dissertation

This dissertation consists of five chapters, chapter one is the introduction, chapter two is the literature review, chapter three is about design and implementation, chapter four is about results and discussion, chapter five presents the conclusion and recommendations.
Chapter 2

Literature Review
Chapter 2

Literature Review

2.1 Introduction

This chapter presents a study, analysis and evaluates the method that used previous researches detecting and removal of cracks in digitized painting via digital image processing.

2.2 Crack Detection with Three Major Steps

Later work by Siddharth Kherada and Varun Varshney, their study focuses on Paintings which suffer from breaks. These patterns are usually called cracks and be caused by aging, drying and mechanical factors. Age cracks can result from non-uniform contraction in the canvas or wood-panel support of the painting, which stresses the layers of painting. Drying cracks are usually caused by the evaporation of volatile paint components and the consequent shrinkage of the paint. Finally, mechanical cracks result from painting deformations due to external causes, e.g. vibrations and impacts [2-3]. The appearance of cracks on painting deteriorates the perceived image quality. However, one can use digital image processing techniques to detect and eliminate the cracks on digitized paintings. A technique that decomposes the image to textured and structured areas and uses appropriate interpolation techniques depending on the area where the missing information lies has also proposed [4]. The results obtained by these techniques are very good. A methodology for the restoration of cracks on digitized paintings. Which adapts and integrates a number of images processing and analysis tools is proposed in this paper.
The methodology is an extension of the crack removal framework presented in [5]. The technique consists of the following steps /methods:

There are three basic steps in above problem namely:

- Crack detection.
- Crack classification.
- Crack Filling.

**Step 1: Crack Detection**

Concept: Cracks usually have low luminance and thus can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack detection procedure based on top-hat transform is proposed on this paper.

Technique top-hat transform defined as following:

\[ y(x) = f(x) - f_{nB}(x) \]  

(1)

where \( f(y) \) luminance component of the image, \( f(x) \) is original negated image \( f_{nB}(x) \) is opening of the image \( f(x) \), \( B \) is the structuring element and \( n \) represents the number of times we do dilation:

\[ nB = B(dilate) \ B(dilate) \ B(dilate) \ldots (n \text{ times}) \]  

(2)

The opening ‘fnB (x)’ of function is low-pass nonlinear filter that erases all peaks (local maxima) in which the structuring element ’nB’ cannot fit, Thus, the image ‘f- fnB’ contains only those peaks and no background at all, hence, the cracks which are the local minima segmented by talking the top hat transform
of the negated image. The top-hat transform result is the parameters are chosen as used by the paper:

![Image](image1.jpg)

**Figure 2.1. Original Image with Cracks [14].**

![Image](image2.jpg)

**Figure 2.2. Original Image after top-hat Transforms [14].**

- **Step 2: Crack Classification**

  In some paintings, certain areas exist where brush strokes have almost the same thickness and luminance future as cracks. The hair of a person in a portrait could be such an area. Therefore, the top-hat transform might misclassify these dark brush strokes as original image, in order to avoid any undesirable alterations to the original image, it is important to separate these brush strokes from the actual cracks, before the implementation of cracks filling procedure. Hence we need to classify the undecided white pixels of the top-hat transformed image. This can be done by technique call ‘Gaussian Classifier’ to classify between brush storks and cracks.
Figure 2.3. Brush stroke Image before using Gaussian Classifier [14].

Figure 2.4. Brush stroke Image after using Gaussian Classifier (The circle shows the increasing pixel [14].

Figure 2.5. Removal of Dark brush Strokes after using Gaussian Classifier [14].

- **Step 3: Crack Filling Methods**

  After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information
from neighboring pixels) to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB (red, green, black) channel independently and affect only those pixels which belong to cracks. Therefore, provided that the identified crack pixels are indeed crack pixels, the filling procedure does not affect the ”useful” content of the image.

2.2.1 Using Modified Trimmed Mean Filter

A variation of the modified trimmed mean (MTM) filter which excludes the samples in the filter window, which are considerably smaller from the local median and averages the remaining pixels.

2.2.2 Using Weighted Median Filter

For this filter, smaller filter windows (e.g. windows that are approximately 30% wider than the widest crack appearing on the image) can be used since the probability that a color value corresponding to a crack is selected as the filter output (a fact that would result in the crack pixel under investigation not being filled effectively by the filter) can be limited by using small weights (e.g. 1) for the pixels centrally located within the window (which are usually part of the crack) and bigger ones (e.g. 2 or 3) for the other pixels.

2.2.3 Doing Iteration:

As after doing iterations the image gets smoothed, so we apply a high boost filter to sharpen the image as shown in figure[2.6]
2.3 The Radial Basis Function networks

Ioannis Giakoumis, Nikos Nikolaidis and Ioannis Pitas presented an integrated strategy for crack detection and filling in digitized paintings. Cracks are detected by using top-hat transform, whereas the thin dark brush strokes, which are misidentified as cracks, are separated either by an automatic technique Two methods to achieve this goal are described in the following subsections.

- A Semi-automatic crack separation and filling

Is simple interactive approach for the separation of cracks from brush strokes is to apply a region growing algorithm on the thresholded output of the top-hat transform, starting from pixels (seeds) on the actual cracks. The growth mechanism that was used implements the well-known grassfire algorithm that checks recursively for unclassified pixels with value 1 in the 8-neighborhood of each crack pixel. At the end of this procedure, the pixels in the binary image, A

Median Radial Basis Function neural network

A great portion of the dark brush strokes, falsely detected by the top-hat transform, can be separated from the cracks. This separation can be achieved by classification using a Median Radial Basis Function (MRBF) neural network as shown in Figure 2.7
After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information from neighboring pixels) to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB channel independently and affect only those pixels which belong to cracks. The methodology has been applied for the virtual restoration of images and was found very effective by restoration experts. However, there are certain aspects of the proposed methodology that can be further improved.

2.4 Edge Detection Technique

Vegard Foldy Thorsen, Nataraj K.R, PhD. They have implemented edge detection to the process of identifying and locating sharp discontinuities in an image. Edge detection is very basic research field in process of image analysis and measurement, the latter must rely on processing the information it provides and edge extraction directly affects the follow-up to the accuracy and ease of handling. The edge of the image is a set of pixels which spatial image intensity or brightness of the direction of mutation or mutation carrier's degree. It is a vector which includes magnitude and direction, in the image it shows the
mutation of gray scale. Edge detection is to detect non continuity of a gray image of and to determine their exact position in the image.

Figure 2.8. Original Image [18].

Figure 2.9. Edge Detection Image[18].

- Thresholded
In first-derivative-based edge detection, the gradient image should be thresholded to eliminate false edges produced by noise. With a single threshold \( t \), some false edges may appear if \( t \) is too small and some true edges may be missed if \( t \) is too large. The final stage involves Thresholding the result from the nonmaximal suppression stage to create a binary image using one threshold value. A straightforward approach would include running a first pass over the image to compare all the pixels with the threshold value. Pixels with gradient values above are marked 0 → 255, classified as an edge. The rest of the pixels are left as zeros, or non-edge. Figure 2.10 shows an example of this pixel mapping approach. This section of the algorithm alone can contribute to more than 20 operations per pixel.
The application can be optimized in multiple ways for detecting more accurate the edges of an image or vision in real time biomedical application. The object as it enters the image, when it is less clear. Another improvement is to combine the results from images to preserve the class consistency of the detected edges.

2.5 Implementation Environment

The name ‘Matlab’ comes from two words: matrix and laboratory. According to the MathWorks (producer of Matlab), which developed primarily Cleve Moler in the 1970’s Derived from FORTRAN subroutines LINPACK and EISPACK. Matlab is a technical computing language used mostly for high-performance numeric calculations and visualization. It integrates computing, programming, signal processing and graphics in easy to use environment, in which problems and solutions can be expressed with mathematical notation. Basic data element is an array, which allows for computing difficult mathematical formulas, which can be found mostly in linear algebra. But Matlab is not only about math problems. It can be widely used to analyze data, modeling, simulation and statistics. Mat-lab high-level programming language finds implementation in
other fields of science like biology, chemistry, economics, medicine and many more.

2.6 User Friendly Graphical Interface

A graphical user interface can be defined as set of techniques and mechanisms, used to create interactive communication between a program and a user. The aim is to give the user pleasant working environment. Colors, alignment and simplicity of look should be considered carefully. Every function, button or any other object should have its meaning, simple and understandable by an average program user. Similar components should have analogous looks and usage. Functions ought to perform quickly and result with wanted outcome. Flexibility can be perceived in this topic as being sensitive to each user’s knowledge, skills, experience, personal performance and other differences that may occur.

A good interface is simple, limits the number of actions and do what it is expected to do. It is not an easy task to design an efficient and user-friendly graphical interface. Luckily, Matlab provides a helpful tool called ‘GUIDE’. After typing guide into Mat-lab’s command line, a quick start window appears. From the choice of exemplary positions it is recommended to pick ‘Blank GUI’. In the new window it is possible to drag and drop each object into the area of the program. On the left side of the created figure there is a list of possible components. The list includes a push button, slider, axes, static and edit texts – which will be described in details in the next paragraph. It also contains objects that will be briefly explained below:
• Toggle Button – once pressed stays depressed and executes an action, after the second click it returns to the raised state and performs the action again;
• Check Box – generates an action when checked and indicates its state (checked or not checked), many options might be ticked in the same time;
• Radio Button – similar to the check box, but only one option can be selected at any given time, function starts working after the radio button is clicked;
• Listbox – displays a list of items and enables user to select one or more from them;
• Pop-up Menu – open a list of choices when the arrow is pressed;
• Panel – groups all components what makes interface easy and understandable, positions of all objects are relative to the panel and do not change while moving the whole panel;
• Button Group – similar to the panel but able to manage specific behavior of radio and toggle buttons that are logically grouped;
• ActiveX Component – allows displaying ActiveX controls that are interactive technology extensions of html. They enable sound, Java applets and animations to be integrated in a Web page.

An example of GUIDE with random components is presented in Figure 3.6:
After the first time saving, GUIDE stores the interface in two files: .fig file, where the description of whole graphic part is placed and .m file, where the code that controls the actions can be found. Each object properties are kept in the .fig file and can be set directly from GUIDE tool, thanks to ready-built Property Inspector. All actions, usually called ‘callbacks’ can be modified and changed in the .m file. Every single component has ‘Tag’ property, which is used while creating the name of the callback reference. To get access to each attribute, Matlab offers command set. It requires reference to the object that is about to be changed and the name of the property, followed by its value.

Among other characteristics, there is an action trigger - callback operation. It is important to know, that any element can have its own specific implementation of this function. Besides operations responsible for actions of objects, there are two additional functions implemented in .m file:

- **Opening function** – executes tasks before the interface becomes visible to the user;
- **Output function** – if needed, it returns variables to the command line.
There is much more behind mechanisms and techniques of programming GUI but this topic will be explained closely in the next chapter.

2.7 Common Knowledge

All operative user interface components of Matlab GUIDE are called ‘uicontrols’. They all contain various selections of properties to be set. After a programmer double-clicks an object created in GUIDE, a window of Property Inspector appears. It is a list of all changeable traits of the component.

2.8 Buttons and Sliders

Push buttons are important components because they allow a user to interact with the program on a visual and simple level. Usually buttons are suggestive and they convey their main purpose. When it comes to sliders, they are not less valuable than buttons. Thanks to sliders, users can change for example brightness or contrast of the image, with some certain steps. Field ‘Style’ takes argument pushbutton or slider, dependable from the type of uicontrol. There are four parameters, connected together. ‘Min’ and ‘Max’ specify the minimum and maximum slider values. Defaults are 0 for minimum and 1 for maximum. Matlab will not allow defining the lowest number bigger than expected utmost numeral. Using both properties, ‘SliderStep’ attribute can be determined. As the name suggest, this characteristic calculates the size of the step which a user may modify, by clicking arrows on this component. The step of the slider is a two element vector. By default it equals the bracket [0,01 0,1], which sets one percent change for clicks on the arrow button and ten percent modification for clicks in the middle. Also feature ‘Value’ relies on previous numbers. It is set to the point, indicated by the slider bar and a programmer can access it with get
function. To get access to each attribute, Matlab offers command set. It requires reference to the object that is about to be changed and the name of the property, followed by its value. Among other characteristics, there is an action trigger - callback operation. It is important to know, that any element can have its own specific implementation of this function. Besides operations responsible for actions of objects, there are two additional functions implemented in .m file:

- Opening function – executes tasks before the interface becomes visible to the user;
- Output function – if needed, it returns variables to the command line.

There is much more behind mechanisms and techniques of programming GUI but this topic will be explained closely in the next chapter.

2.9 Axes

Axes component contains several additional attributes. ‘Box’ property defines whether the region of the axes will be enclosed in two-dimensional or three-dimensional area. Options ‘XTick’, ‘XTickLabel’ and ‘YTick’, ‘YTickLabel’ allow a programmer to define what values will be displayed along the horizontal and vertical axis. As a separator, the easiest way is to use this line ‘|’. Also the location of both lines can be set with help of ‘XAxisLocation’ and ‘YAxisLocation’ features. ‘XGrid’ and ‘YGrid’ creates the grid that might be useful while cropping or resizing processed image. Besides all graphical attributes responsible for outer look of the axes, this object contains also all features common for different components. A lot of properties will not be described here because they refer to appearance of graphs, drawn with plot
command, while this paper treats about image processing. Therefore, axes will be used as an area of picture input and display

2.10 Creating Menu

Every decent application should have the menu bar. An average computer user is accustomed to possibility of getting most things done with the help of the menu. That is why Matlab enables programmers to create two kinds of menus:

- Menu bar objects – drop-down menus whose titles are situated on the top of the figure.
- Context menu objects – pop-down menus that appear after a user right-click one of the components.

To create both of them, GUIDE offers Menu Editor. They are implemented with two objects – uimenu and uicontextmenu. After entering GUIDE Menu Editor it is possible to create a hierarchical menu, without any limitations of items amount. This tool helps programmers on many levels. Process of making menu becomes intuitive and simple. It enables setting of menu properties with Property Inspector, for every menu and submenu element. Creating context menu requires changing the tab into ‘Context Menus’. Then the process goes similarly to the menu bar building. There are several properties that can be set just after new menu is generated. ‘Label’ defines the name of the item that will be displayed to the user. ‘Tag’ value determines the name, needed to identify the callback function. ‘Separator above this item’ is responsible for a slim line between logically divided menu elements. Another attribute ‘Check mark this item’ displays a check next to the menu item and indicates the current state of this item. To ensure that users can select any option, property ‘Enable this item’ has to be marked.[11,12].
2.11 Summary

In this paper, the detection and removal filling in digitized paintings.

- Cracks are detected by using top-hat transform, whereas the thin dark brush strokes, which are misidentified as cracks, are separated using a Gaussian classifier.

- Crack interpolation is performed by appropriately modified filters.

- Crack detection stage is not very efficient in detecting cracks located on very dark image areas, since in these areas the intensity of crack pixels is very close to the intensity of the surrounding region. A possible solution to this shortcoming would be to apply the crack detection algorithm locally on this area and select a low threshold value.

- Another situation where the system (more particularly, the crack filling stage) does not perform as efficiently as expected is in the case of cracks that cross the border between regions of different color. In such situations, it might be the case that part of the crack in one area is filled with color from the other area, resulting in small spurs of color in the border between the two regions. Use of image in painting techniques could also improve results.
Chapter 3

Design and Implementation
Chapter 3

Design and Implementation

3.1 Introduction
The digital image processing technique can be applied to the virtual restoration of artistic paintings. Using an integrated methodology for the detection and removal of cracks on digitized paintings is presented in this chapter.

3.2 Background
Many paintings, especially old ones, suffer from breaks in the substrate, the paint, or the varnish. These patterns are usually called cracks or craquelure. The cracks are the breaks mainly influenced by ageing, drying and mechanical factors. With the help of digital image processing technique, cracks are detected and then eliminated which provide help to artwork historian, museum curators and normal public on how the artwork would look like in earlier [1]. Age cracks can result from non-uniform contraction in the canvas or wood-panel support of the painting, which stresses the layers of the painting. Drying cracks are usually caused by the evaporation of volatile paint components and the consequent shrinkage of the paint. Finally, mechanical cracks result from painting deformations due to external Causes, e.g. vibrations and impacts [2-3].

With the help of digital image processing technique, cracks are detected and then eliminated which provide help to artwork historian, museum curators and normal public on how the artwork would look like in earlier state.

The technique consists of the different stages:

- Crack detection.
- Separation of the thin dark brush strokes, which have been
misidentified as cracks.

- Crack filling (interpolation).

### 3.2.1 Crack Detection

A crack detector is applied on the luminance component of an image and should be able to identify such minima. Top-hat transform is proposed in this research for crack detection procedure. The top-hat transform is filter defined as

$$y(x) = f(x) - fnB(x)$$  \(^{(1)}\)

where \(y(x)\) is the luminance component of image, \(fnB(x)\) is the opening of the function \(f(x)\) with the structuring set, defined as:

$$nB = B(dilate) \ B(dilate) \ B(dilate) \ldots \ (n \ times)$$ \(^{(2)}\)

In the previous equation, the dilation operation. The final structuring set is evaluated only once using equation (2) and is used subsequently in the opening operation of equation (1). The opening of a function is a low-pass nonlinear filters that erases all local maxima for which the structuring element cannot fit. Hence, the digital picture contains only those local maxima and no background at all. Since cracks are local minima rather than local maxima, the top-hat transform should be applied on the inverse luminance image. The output of the detection procedure can be controlled by selecting the suitable values for the user can control the result of the crack-detection procedure by choosing appropriate values for the following quantities:

- Structuring element type;
- Structuring element size and the number of dilations in equation (2).
These quantities produce an effect on the size of the resultant structuring element and must be selected according to the broadness of the cracks to be detected. The top-hat transform produces a grayscale resultant image where pixels with a high grey value are potential crack or crack-like elements. Therefore, a thresholding operation on resultant image is required to separate cracks from the rest of the image. The number of image pixels which are separated as cracks decreases as the threshold value increases. Hence certain cracks, especially in dark image areas where the local minimum condition may not be achieved, can remain undetected. Therefore it is better to select the threshold value so that some cracks remain undetected than to select a threshold that would result in the detection of all cracks but will also falsely identify as cracks, and subsequently modify, other image structures.

3.2.2 Separation of the Thin Dark Brush Strokes, Which have been Misidentified as Cracks

Some artistic paintings contain certain breaks where they have almost the same broadness and luminance features as cracks. Therefore, the top-hat transform might misclassify these breaks as cracks. Thus, in order to avoid any undesirable changes to the original digital painting, it is very important to separate these breaks from the actual cracks, before to put into the effect of the crack filling method. A procedure to accomplish this aim is described in the following subsection.

3.2.2.1 Semi-Automatic Crack Separation

An easy user friendly technique for the separation of cracks from breaks is to apply a region growing algorithm on the thresholded result of the top-hat transform, starting from pixels (seeds) on the actual cracks. The pixels are
chosen by the user in an interactive way. At least one seed per connected crack element should be selected. In the similar way, the user can apply the method on the breaks, if this is more appropriate. The growth mechanism checks recursively for unclassified pixels with value 1 in the 8-neighborhood of each crack pixel. At last phase of this procedure, the pixels in the binary digital picture, which correspond to breaks that are not 8-connected to cracks, will be cleared. An example is shown in figure1. Cracks are normally considered as darker than the background and that they are characterized by a uniform gray level, tracking is accomplished on the basis of two main features: absolute gray level and crack uniformity. Once the system knows some pixels belong to a crack, it assigns to the crack new pixels if their gray levels lie in a given range and do not differ significantly from those of the pixels already classified as belonging to the crack. Figure1 shows three iterations of the tracking procedure. Point A is the starting crack point that the user selected. First, the 8-neighborhood of pixel A is considered (pixels B1 through B5). For each pixel Bi of the neighborhood the system tests the following conditions:

\[ |f(A) - f(Bi)| \leq \Box T \]  
\[ f(Bi) \in \Box [T1, T2] \]

where \( f(Bi) \) represents the gray level of Bi, and T, T1 and T2 are adaptive thresholds calculated on the basis of the crack pixels previously classified as such.
By referring to Figure 1, the system observes that only pixels B1, B2, and B3 belong to the crack. The process continues by referring all the pixels adjacent to B1, B2 and B3, namely C1 through C6. The system checks conditions 3 and 4 for each of them. More importantly, the validity of condition 3 is verified for each pair of pixels belonging to the same 8-neighborhood. Thus, considering an example, a pixel Ci is supposed to belong to the crack if, in support to condition 4, condition 3 is verified for at least one Bj in the neighborhood of Ci (in Figure 2, only point C5 is identified as a crack pixel). The above process iterates until the system can’t get a pixel with the appropriate features. An interesting feature of the process is that it traverse cracks by fronts (\{B1, B2, B3\} at iteration 1, \{C5\} at iteration 2, and \{D2, D3\} at iteration 3).

3.2.3 Crack Filling

Last step of the crack extraction process consists of simple comparisons between gray scale values. Every crack pixel has a corresponding gray scale value found in the underlying gray scale image. Up to this point all crack pixels are a direct result of edge detection. Hence, all crack pixels are in fact binary edge pixels where only one side of a pixel is considered valid as far as being a part of the actual crack. Therefore, filling the appropriate gap between two corresponding binary pixels (Figure 3.2) is a twofold iterative approach where
the first iteration differs from the rest. Furthermore, the marking of new crack pixels is done separately. For that reason, during a new iteration, only pixels marked in the previous iteration is considered. That way the amount of pixels to process becomes smaller for every additional iteration. The first iteration can be thought of as “moving one pixel away from the edge and thus into the crack”. During the first iteration, a crack (binary edge) pixel compares its neighbors gray scale values against each other in a total of four ways. The gray scale value of its left neighbor is compared to the gray scale value of its right neighbor. The neighbor that has the smallest value are marked as a new crack pixel if, and only if, its value is also smaller than the current pixels gray scale value. The same reasoning applies to the other opposite neighbors. As a result, all binary edges are expanded by one pixel into the crack (Figure 3.2). The remaining iterations is somewhat different. Like in edge thresholding, percentiles is here used in order to find a sufficiently small value, based on the values in the gray scale image, to use as the definition of a crack pixel. More specifically, in order to be marked as a crack pixel, the gray scale value of a pixel needs to be lower than this percentile value. That way,

![Figure 3.2. Before Crack Filling.](image-url)

![Figure 3.3. After 1\textsuperscript{st} Crack Filling Iteration.](image-url)
3.3 Methodology

The methods implemented on this study are based on studying the digital image processing technique used for cracks identification and removal. It considers the classification of cracks into paints to aid in damage assessment. The implementation have been used is high-performance and modern programming language environment for technical computing called Matlab.

The methods include the followings:

- Input module.
- Gray scale conversion module.
- Cracks detection module.
- Crack filling module.
- Output module
- **Data Flow Diagram**
  The data flow diagram shown represents the complete steps required in this dissertation.

![Diagram](image)

**Figure 3.5. Data Flow Diagram.**

- **Input module**
  It is used to give the input image such as cracked image.

- **Gray scale conversion module**
  If the image is colored then it should be converted into the common color format like a gray colored image. This work will be performed by using the gray scale algorithm.

- **Cracks Detection module**
  This module is used to find the cracks in the cracked image via the surrounded pixels.
- **Crack Filling module**
  This module used to fill the color by using the median filter and cracks removal algorithm. The cracks will be filled by the surrounded pixel color.

- **Output module**
  The output will be presented by this module. All changes in this project will be displayed from the separate forms.
Chapter 4
Results and Discussion
Chapter 4
Results and Discussion

4.1 Introduction

This chapter discus the results of the project through showed how the maintenance and testing implemented to the image crack detection for each module by using Matlab with the same preprocessing technique, then we got all the results together.

4.2 The Maintenance

The objectives of this maintenance work are to make sure that the system gets into work all time without any bug. This system able to accept any modification after its implementation, also has been designed to favor all new changes which will not affect the system’s performance or its accuracy.

4.3 System Testing

System testing is aimed to ensuring that the system works accurately and efficiently. The main objective of testing is to uncover errors from the system. Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions.

4.3.1 Module Testing

Module testing verification efforts on the smallest module of software design. The modules are tested separately. This testing is carried out during
programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

4.3.2 Integration Testing
Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole. In the integration-testing step, all the error uncovered is corrected for the next testing steps.

4.3.3 Validation Testing
To uncover functional errors, that is, to check whether functional characteristics confirm to specification or not.

4.4 Study Cases
This section presents five case studies that are implemented on detection and removal of cracks in digitized paintings via digital image processing, through applying a method which identifies of missing or damaged pixel painting areas by filling in information from the neighboring pixel areas. It considers the classification of cracks into paints to aide in damage assessment. The implementations has been applied is a high-performance and modern programming language environment for technical computing.
4.4.1 Study Case 1

- **Input Image**
This function is used to give the original image such as cracked image from this module and display it on the GUI as followed:

![Original Image](image)

**Figure 4.1 Original Image[13].**

```matlab
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject handle to OpenMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global im;
file = uigetfile('*.jpg');
im = imread(file);
axes(handles.axes1);
imshow(im);
axis off;
```
• Crack Detection
This function is used to find the cracked image via the surrounded pixels and display on the GUI as followed:

![Crack Detection Image](image-url)

Figure 4.2. Crack Detection [13].

```matlab
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)
    global im;
    global thres;
    global im2out;
    global nbn;
    im2out = crack_detect(im, thres/255.0, nbn);
    axes(handles.axes2);
    imshow(im2out);
    axis off;
```
• Crack Filling
This function is used to fill the color by using the median filter and cracks removal algorithm. The cracks will be filled by the surrounded pixel color and display on the GUI as followed:

```
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global im;
global im2out;
C = crack_fill_A(im,im2out,5);
figure,imshow(C);axis off;
```

Figure 4.3. Crack Filling[13].
4.4.2 Study Case 2

An original image are used to give as (cracked image). If the image is color image, it should have to convert into the common color format like a gray colored image. This work will done the use of gray scale algorithm as shown in figure 4.4

![Figure 4.4. Original Image[14].](image)

Cracks usually have low luminance and thus can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack detection procedure based on top-hat transform technique defined as shown in figure 4.5

![Figure 4.5. Crack Detection[14].](image)

After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information from neighboring pixels) to fill (interpolate) the cracks. Two classes of
techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB (red, green, black) channel independently and affect only those pixels which belong to cracks. The filling procedure does not affect the "useful" content of the image as shown in figure 4.6.

Figure 4.6: Crack Filling[14].
4.4.2 Study Case 3

An original images are used to give as (cracked image). If the image is color image, it should have to convert into the common color format like a gray colored image. This work will done the use of gray scale algorithm as shown in figure 4.4

![Figure 4.7. Input Image[15].](image)

Cracks usually have low luminance and thus can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack detection procedure based on top-hat transform technique defined as shown in figure 4.5

![Figure 4.8. Crack Detection[15].](image)
After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information from neighboring pixels) to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB (red, green, black) channel independently and affect only those pixels which belong to cracks. The filling procedure does not affect the "useful" content of the image as shown in figure 4.6.

Figure 4.9. Crack Filling[15].
4.4.4 Study Case 4

An original images are used to give as (cracked image). If the image is color image, it should have to convert into the common color format like a gray colored image. This work will done the use of gray scale algorithm as shown in figure 4.4

![Figure 4.10. Input Image[16].](image)

Cracks usually have low luminance and thus can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack detection procedure based on top-hat transform technique defined as shown in figure 4.5

![Figure 4.11. Crack Detection[16].](image)

After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information from neighboring pixels) to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are
proposed for this purpose. Both are implemented on each RGB (red, green, black) channel independently and affect only those pixels which belong to cracks. The filling procedure does not affect the "useful" content of the image as shown in figure 4.6

Figure 4.12. Crack Filling[16].
4.4.5 Study Case 5

An original images are used to give as (cracked image). If the image is color image, it should have to convert into the common color format like a gray colored image. This work will done the use of gray scale algorithm as shown in figure 4.4

![Figure 4.13: Original Image[17]](image)

Cracks usually have low luminance and thus can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack detection procedure based on top-hat transform technique defined as shown in figure 4.5

![Figure 4.14: Crack Detection[17]](image)
After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information (i.e., information from neighboring pixels) to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB (red, green, black) channel independently and affect only those pixels which belong to cracks. The filling procedure does not affect the "useful" content of the image as shown in figure 4.6

Figure 4.15: Crack Filling[17].
Chapter 5

Conclusion and Recommendation
Chapter 5

Conclusion and Recommendation

5.1 Conclusion

In this Dissertation, an integrated strategy have been presented for crack detection and filling in digitized paintings. Cracks are detected by using top-hat transform, whereas the thin dark brush strokes, which are misidentified as cracks. The methodology has been applied for the virtual restoration of images and was found very effective by restoration experts. However, there are certain aspects of the proposed methodology that can be further improved. For example, the crack-detection stage is not very efficient in detecting cracks located on very dark image areas, since in these areas the intensity of crack pixels is very close to the intensity of the surrounding region.

5.2 Recommendation

- A possible solution would be to perform edge detection or segmentation on the image and confine the filling of cracks that cross edges or region borders to pixels from the corresponding region.
- Future work will focus on separating breaks which are misidentified as cracks and applying interpolation technique for the restoration of digitized artistic pictures.
- Another improvement of the crack filling stage could aim at using properly adapted versions of nonlinear multichannel filters (e.g., variants of the vector median filter) instead of processing each color channel independently.
• Using artificial intelligence to establish rules to identify the background by analyzing the pattern of the image and rate the cracks accordingly can be a better means of identifying cracks.

• A possible solution to this shortcoming would be to apply the crack-detection algorithm locally on this surrounding region area and select a low threshold value.
References


[18] http://www.google.com/imgres?imgurl=http://1.bp.blogspot.com/-lzIhy7uMVYs/UFXkn68gAgI/AAABnE/TaFLub7tDo8/
Appendix
Matlab Source Code

This appendix presents the code of the Matlab language.

function varargout = main(varargin)
% MAIN M-file for main.fig
% MAIN, by itself, creates a new MAIN or raises the existing
% singleton*.
% H = MAIN returns the handle to a new MAIN or the handle to
% the existing singleton*.
% MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in MAIN.M with the given input
% arguments.
% MAIN('Property','Value',...) creates a new MAIN or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before main_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to main_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
% one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDELS
% Edit the above text to modify the response to help main
% Last Modified by GUIDE v2.5 17-Jan-2013 13:26:47
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @main_OpeningFcn, ...)
'gui_OutputFcn', @main_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 main is made visible.
function main_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 main (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);
axes(handles.axes1);
axis off;
axes(handles.axes2);
axis off;
global nbn;
global thres;
thres = 23;
nbn=1;
% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = main_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)

global im;
global thres;
global im2out;
global nbn;
im2out = crack_detect(im,thres/255.0,nbn);
axes(handles.axes2);
imshow(im2out);
axis off;

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

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

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

global im;
file = uigetfile('*.jpg');
im = imread(file);
axes(handles.axes1);
imshow(im);
axis off;

% ---------------------------------------------------------------------------

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

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

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

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

% --- Executes on slider movement.
function thres_slider_Callback(hObject, eventdata, handles)
    % hObject    handle to thres_slider (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,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global thres;
global im;
global im2out;
global nbn;
thres = get(hObject,'Value');
im2out = crack_detect(im,thres/255.0,nbn);
axes(handles.axes2);
imshow(im2out);
axis off;

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

    % Hint: slider controls usually have a light gray background.
    set(hObject,'Max',255);
    set(hObject,'Min',0);
    if isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
    set(hObject,'Value',23);

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
    % hObject    handle to popupmenu2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
    %    contents {get(hObject,'Value')} returns selected item from popupmenu2 contents
    contents = cellstr(get(hObject,'String'));
    global nbn;
nbn = str2num(contents{get(hObject,'Value')});
% --- Executes during object creation, after setting all properties.
% --- Executes just before main is made visible.
function main_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 main (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);
axes(handles.axes1);
axis off;
axes(handles.axes2);
axis off;
global nbn;
global thres;
thres = 23;
nbn=1;
% UIWAIT makes main wait for user response (see UIRESUME)
% uwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = main_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 popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'Value',1.0);

% --- 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)
global im;
global im2out;
C = crack_fill_A(im,im2out,5);
figure,imshow(C);axis off;

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
% pushbutton4.
function pushbutton4_ButtonDownFcn(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 pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global im;
[filename, pathname] = uigetfile( 'C:\Work\myfile.jpg')
im = imread(filename);
axes(handles.axes1);
imshow(im);
axis off;

% --- Executes just before main is made visible.
function main_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 main (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);
axes(handles.axes1);
axis off;
axes(handles.axes2);
axis off;
global nbn;
global thres;
thres = 23;
nbn=1;
% UIWAIT makes main wait for user response (see UIRESUME)
% uiswait(handles.figure1);

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