CERTIFICATION OF APPROVAL

Reconstruction of Digital Document Images from Scribbled Sketches

by

Mohd Ridhwan bin Abu Hassan

A dissertation submitted to the
Business Information Systems Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
Bachelor of Technology (Hons)
Business Information Systems

Approved by,

(JALE BIN AHMAD)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
July 2009

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own, except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources of person.

(MOHD RIDHWAN BIN ABU HASSAN)

ABSTRACT

The presence of artifacts and defects in digital images compromise their homogeneity. This problem however can be unraveled by having the defective regions to be recognized manually, removed and filled. Comes image inpainting, this predicament is no longer an intricate task. Still, the need for human intervention in identifying defective regions curtails its autonomy and robusity. Thus, the author intends to discuss some fundamental ideas behind image inpainting, removal and an ingenious way of filling those blank holes dubbed "texture synthesis". This report concisely reviews the concept of image inpainting, right from the semi-automatic removal of defective regions by simulating Photoshop's Magic Wand tool using color segmentation and filling-in the blank holes as the results of removing non-textual objects with a plausible result to the human eye. This project focuses on reconstructing digital images with textual components. To achieve this, the author implements software prototyping in constructing the software program to realize the objectives using MATLAB and C++. Finally, the author conducted various analyses on the accuracy and performance of the software program.

ACKNOWLEDGEMENTS

I am deeply indebted to my supervisor, Jale bin Ahmad, for the making the past few months one of the best periods of my life. For each effort I put into my work, I think he put two. I arrived at my senior year quite without a clue, and rather worried of what kind of research I would indulge into. Fortunately, it was difficult to stray too far afield when striving to follow his examples and exemplary advices. His dexterity in our mutual research interest makes things even easier. He was a better supervisor than I imagined possible.

I owed a bunch of thanks to fellow friends, Sarhanah, Hawa, Khushairy and Zaimal, for listening to my gibberish babbling, and even pretending to enjoy them, while I am under pressure from piles of works and endless datelines. My gratitude also conveys to fellow supervisees and former coursemates for attending tirelessly to my ceaseless curiosities on work matters.

I would never forget the fact that my family had provided tremendous supports while I am a student, and for that, my hefty gratitude.

I am also indebted to the faculty members of Computer and Information Sciences department of Universiti Teknologi PETRONAS and the executives as well, who never neglect helping me with my trouble on my final year project matters. For that, I owed my many thanks.

As an ending note, my thanks impart to those I have not mentioned here but whose assistance during all this while had helped me a lot and thus, very much appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background of Study

According to (Shen, 2003) and (Inpainting - Wikipedia, the free encyclopedia, 2009), "Image Inpainting" is the artistic synonym for Image Interpolation, known to artists working for archiving in museums, and by definition is the process of reconstructing lost or deteriorated parts of images and videos. First coined by the authors in (Bertalmio, 2001), the authors revolutionized the classical methods of image inpainting by proposing inpainting method based on Partial Differential Equations. However, these classical methods rely on users to define and recognized the defective areas to be removed and inpainted.

1.2 Problem Statement

The author realizes a significant problem on which image inpainting may come into use. Given a paper with blocks of handwritten text and diagrams and arbitrary sketches and smudges are inflicted by ink on top of those. The texts and diagrams will be rendered difficult to read as they are occluded by the defects. This scenario can be analogized by an exam paper ready to be marked by a teacher, which she is unfortunately unable to do so when her daughter of 2-years-old practices her flair to abstract drawings on that exam paper using conventional ball-point pen. The author aims to propose a framework for detecting and removing the defects and inpainting the empty regions. With this, the teacher will be able to continue marking the exam papers. This project will greatly improve the readability of digital images with textual components that have been spoilt by defects.

1.3 Objectives

This research aims to:

1. Provide a semi-automatic detection of defective regions in still images.
2. Fill empty holes of removed regions by synthesizing neighboring textures.
3. Develop a prototype, for which image acquisition, region removal and inpainting can be done in one place.

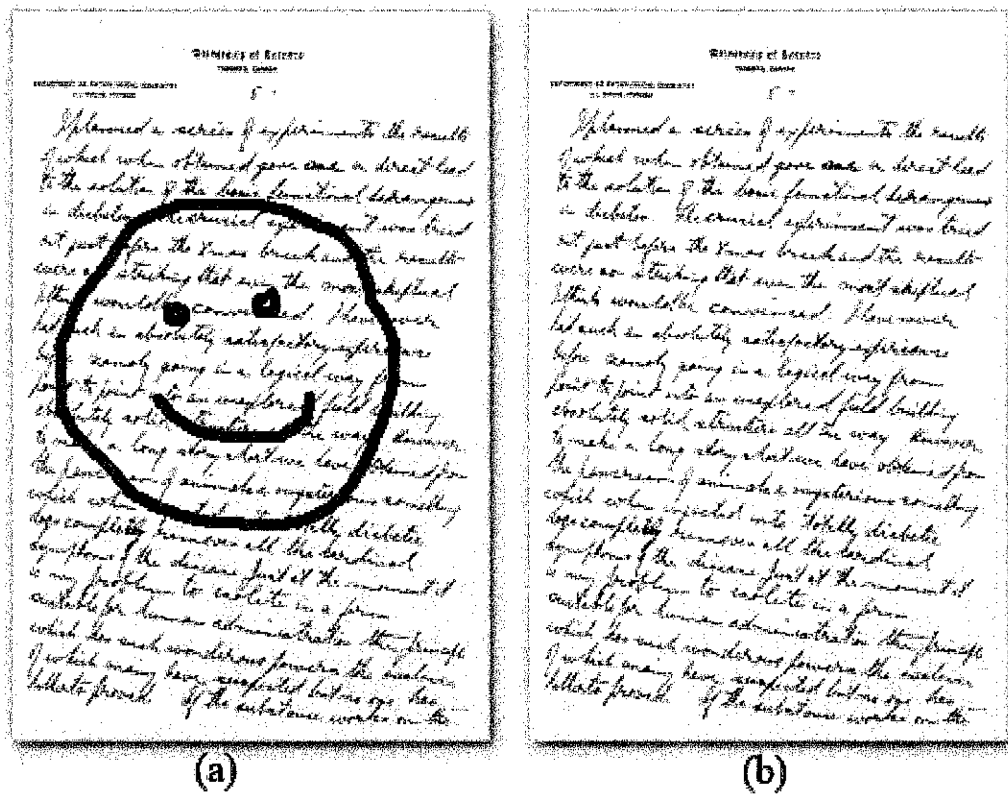


Figure 1 *A sample intended result. The Image is captured from a handwritten paper acquired from a conventional digital scanner. (a): Image with defective region, marked in red ink. (b): Defective region removed and inpainted.*

1.4 Scope of Study

This research bases its roots under the premise that incomplete images can be filled by taking a texture sample of neighboring regions and propagates through the missing regions. This research intends to provide partial automation to the whole process by performing color segmentation and will be focusing on images with textual components. Color segmentation will distinguish the defective regions and non-defective regions. The implementation of the system will include two (2) parts: command-line execution and GUI. However, command-line execution will be given priority to completion. The focus of this research will be to correctly identify and remove the defective regions from still single image with minimum human interventions and inpaint those removed regions in a way that looks "reasonable" to the human eye.

CHAPTER 2

LITERATURE REVIEW

2.1 Defect Detection

Defects, in a nutshell, are any foreign entities on the image which will destroy the homogeneity of the image (Ye Q, 2003). Examples of defects on an image includes ink spray, scratches, dust, varying illumination and geometric distortion, most of which may be procured from normal image acquisition through scanners or cameras. The defects detection, ink in particular, relies on the property of the ink (e.g., ink color, shape and decadence of gradient). Defects are hard to be modeled explicitly due to its arbitrary nature and the effects of occlusion with the underlying objects.

Due to this vexing fact, most defects detection algorithm rely on users to manually identify them, although there have been increasing literature on automatic defects identification, such as those of (Jyotirmoy Banarjee, 2009), which implements probabilistic context model and learned relationship using Markov Random Field.

2.2 Image Segmentation

Image segmentation is the process of dividing an image into non-overlapping, connected image areas called, regions, on the basis of criteria governing similarity and homogeneity. Similarly, color image segmentation describes the process of extracting from the image domain one or more connected regions satisfying uniformity. Linda Shapiro of Washington University defined a good image segmentation as "having regions of an image that are uniform and homogeneous with respect to some characteristic such as gray tone or texture. Region interiors should be simple and without many small holes. Adjacent regions of segmentation should have significantly different values with respect to the characteristic on which they are uniform. Boundaries of each segment should be simple, not ragged, and must be spatially accurate.

There are many segmentation methods. One of them is Amplitude Segmentation which is based on the fact that some images can be categorized based on their luminance. Many images can be characterized as containing some object of interest of reasonably uniform brightness placed against a background of differing brightness. Typical examples include handwritten images in black over white background. It is a trivial task to set a mid-gray threshold to segment the object from the background. However, practical problems occur when the image is subject to noise and when both the object and background are in some broad range of gray scales and when the background is not uniform. Understanding that sketches on an handwritten image may come in different color, the author proposes color-based segmentation.

2.3 Color Segmentation

Color-based segmentation methods merge pixels and regions together based on their color similarity. Adjacent pixels P_i and P_j are merged into the same region if the color distance D_{ij} between the pixels is smaller than the pixel grouping threshold $D(T_p)$. The parameter T_p is a percentage of the number of pixels within the neighborhood of P_i and P_j. $D(T_p)$ is a color distance, such that T_p is the amount of adjacent pairs of pixels in the neighborhood having color distances smaller than or equal to $D(T_p)$. Empirical studies by (Libo Fu, 2005) shows that $D(T_p)$ of 80% consistently produces good initial segmentation. The color distance is defined by the Euclidean distance between two pixels, P_i and P_j, as

$$D_{ij} = \sqrt{[(i_R - j_R)^2 + (i_G - j_G)^2 + (i_B - j_B)^2]}$$

These regions are then joined into bigger cluster where they are likely to belong to the same object if their D_{ij} of the two regions are smaller or equal to $D(T_p)$ and the two regions are geographically close. Later, bigger clusters are formed by joining clusters which have D_{ij} similar or equal to $D(T_p)$. For each merging of pixels into bigger clusters, the $D(T_p)$ are relaxed, that is setting it to a higher value, to allow more pixels to be grouped together.

The author of (Swee-Seong Wong, 2000), proposed non-color-based grouping to further improve the initial color segmentation described below:

1. Hole Filtering

   Removal of small regions that are insignificant compared to its neighboring regions. A region R_i is insignificant if its area S_i is smaller than or equal to $14(A_i)^{\frac{1}{4}}$ where A_i is the total size of its neighboring regions. A hole is removed by absorbing it into the adjacent region that shares the longest boundary.

2. Compact Grouping

   Combining adjacent clusters to form more compact clusters. The compactness M_i of cluster C_i is defined as the ratio of the square of the cluster's perimeter and its area. The smaller M_i is the more compact C_i will be.

2.4 Image Inpainting

According to (Criminisi, 2003), inpainting is the process of reconstructing lost or deteriorated parts of images and videos. This technique can be used in falsifying digital image or video data. For example, person A removed a region in image X where Person B, someone person A hated so much, which is standing next to person C, person A's best friend. Person A then used inpainting technique to reconstruct image X as if person B is not in it in the first place. In the case of a valuable painting, this task would be carried out by a skilled image restoration artist. In the digital world, inpainting refers to the application of sophisticated algorithms to recover lost or corrupted parts of image or video data.

Mathematically, given Ω which denote a complete image domain and a subset β of Ω which is a result of certain factors such as object occlusion or packet loss in data transmission, the goal of image inpainting is to recover the original ideal image ω on the entire domain Ω, based on the partial observation, $n_o\,|_{\Omega\backslash\beta}$.

The authors of (Criminisi, 2003) categorized image inpainting into three general ways of solving them:

1. Statistical-based Method

   Extraction of input image statistics via compact parametric statistical models, and to synthesize a new texture, an output image with purely noise will be iteratively pertubated until its statistics match the estimated statistics of the input texture. This method is only applicable to texture synthesis, not to image inpainting in general.

2. Partial Differential Equation (PDE)-based Method

   Introduced by (Bertalmio, 2001), this method smoothly propagates information from the boundary of source region towards the interior of the target region (empty region), simulated by solving PDE which is typically non-linear and of higher-order. It replicates the way professionals restore valuable drawings. This method is only suitable for empty holes which are small, thin, and elongated. Bigger regions to be inpainted resulted in over-smoothing and blurring artifacts.

3. Exemplar-based Method

   This method, of which is the most successful among the aforementioned methods, fills unknown region by copying content from the observed part of the image. It implements Markov Random Field as described in (Alexie A. Efros, 1999).

2.4.1 Exemplar-based Image Inpainting

Exemplar-based Image Inpainting is based on (Criminisi, 2003). The method the author chose to implement for this dissertation is described in the aforementioned paper. The core of their algorithm is an isophote-driven (linear structures) image sampling processes.

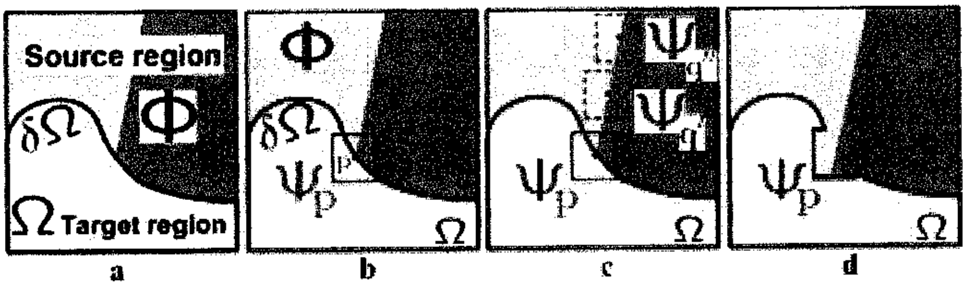


Figure 2: Structure propagation by exemplar-based texture synthesis. (a) Original image, with the *target region* Ω, its contour $\delta\Omega$ and the *source region* Φ clearly marked. (b) We want to synthesize the area delimited by the patch $\Psi_{\mathbf{p}}$ centred on the point $\mathbf{p} \in \delta\Omega$. (c) The most likely candidate matches for $\Psi_{\mathbf{p}}$ lie along the boundary between the two textures in the source region, *e.g.*, $\Psi_{\mathbf{q}'}$ and $\Psi_{\mathbf{q}''}$. (d) The best matching patch in the candidates set has been copied into the position occupied by $\Psi_{\mathbf{p}}$, thus achieving partial filling of Ω. The target region Ω has, now, shrank and its front has assumed a different shape. See text for details.

Figure 2 *Structure propagation by exemplar-based texture synthesis. (An excerpt from (Criminisi, 2003)).*

- Extract the manually selected initial front $\delta\Omega^0$.
- Repeat until done:

  1a. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.

  1b. Compute priorities $P(\mathbf{p})$ $\forall \mathbf{p} \in \delta\Omega^t$.

  2a. Find the patch $\Psi_{\hat{\mathbf{p}}}$ with the maximum priority, *i.e.*, $\Psi_{\hat{\mathbf{p}}} \mid \hat{\mathbf{p}} = \arg\max_{\mathbf{p}\in\delta\Omega^t} P(\mathbf{p})$

  2b. Find the exemplar $\Psi_{\hat{\mathbf{q}}} \in \Phi$ that minimizes $d(\Psi_{\hat{\mathbf{p}}}, \Psi_{\hat{\mathbf{q}}})$.

  2c. Copy image data from $\Psi_{\hat{\mathbf{q}}}$ to $\Psi_{\hat{\mathbf{p}}}$.

  3. Update $C(\mathbf{p})$ $\forall \mathbf{p} \mid \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$

Figure 3 *Algorithm for region filling. (An excerpt from (Criminisi, 2003)).*

CHAPTER 3

METHODOLOGY

3.1 Software Prototyping

Before developing a system, a developer should have a methodology to structure, plan, and to control the process of developing an information system. One type of methodology may be perfect for a particular projects, while for others, may be a disaster. Among the numerous types of methodologies available, the author chooses software prototyping methodology. The main goal of using software prototyping is to build a very robust prototype in a structured manner and constantly refine it.

According to (Software Prototyping, 2009), software prototyping is an activity in software development for the creation of prototypes that represents the incomplete, partially functional software program being developed. A prototype typically simulates only a few aspects of the features of the eventual program, and may be completely different from the eventual implementation.

The conventional purpose of a prototype is to allow users of the software to evaluate developer's proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. The benefits of software prototyping are as follows:

1. The software developer can obtain feedback from the users early in the project development, thus allowing the client and developer to compare if the software made matches the software specifications.
2. It allows the software developer to get some insight into the accuracy of the initial project estimates and whether the deadlines and milestones proposed can be successfully met.

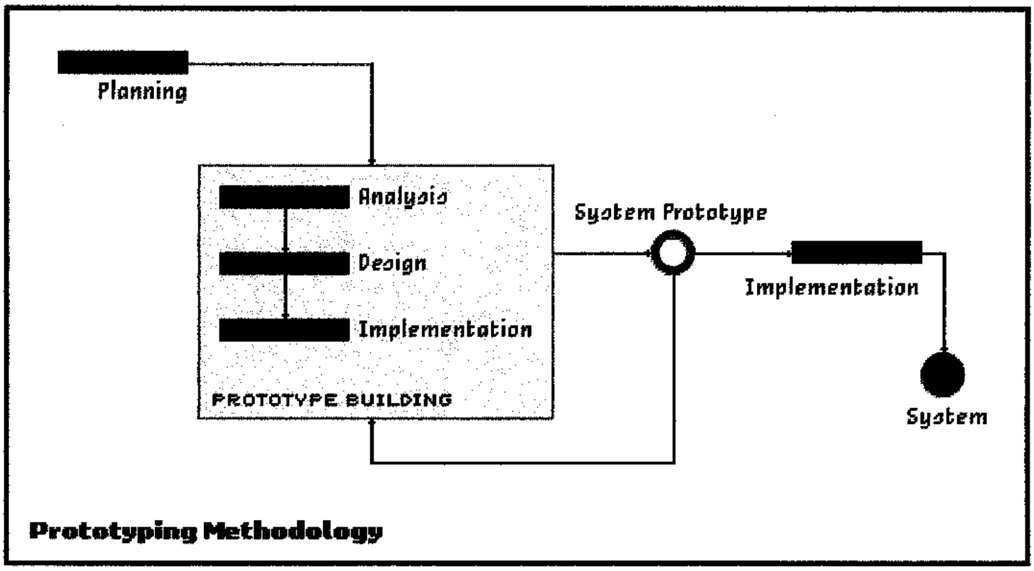


Figure 4 *The software prototyping methodology*

3.2 Equipment and Tools

The author makes use of the following tools for developing the system:

1. Matrix Laboratory (MATLAB)
   a. Version 7.7 (R2008b)
   b. Usage: as the backbone for the software implementation.
   c. MATLAB is a numerical programming environment, which allows high-performance matrix computation, data analysis and plotting, user interface development, interfacing with other programming languages and easy expansion with collection of MATLAB codes for specific purpose programming called Toolboxes. For the purpose of this project, the author used Image Processing toolbox.
2. Minimalist GNU for Windows (MingW) C++ Compiler
   a. Version 3.4.5
   b. Usage: to compile C++ code portion to search for best texture exemplars
3. IAM-Online Handwritten Text Database
   a. Usage: for measuring the accuracy of document reconstruction containing handwritten text
   b. Developed by the computer vision researchers at the University of Bern, this database contains form of handwritten English text acquired on a whiteboard.
4. Hewlett-Packard (HP) Deskjet All-in-One
   a. Model F4280
   b. Usage: for digitizing the handwritten document

3.3 Proposed Graphical User Interface (GUI) for the prototype

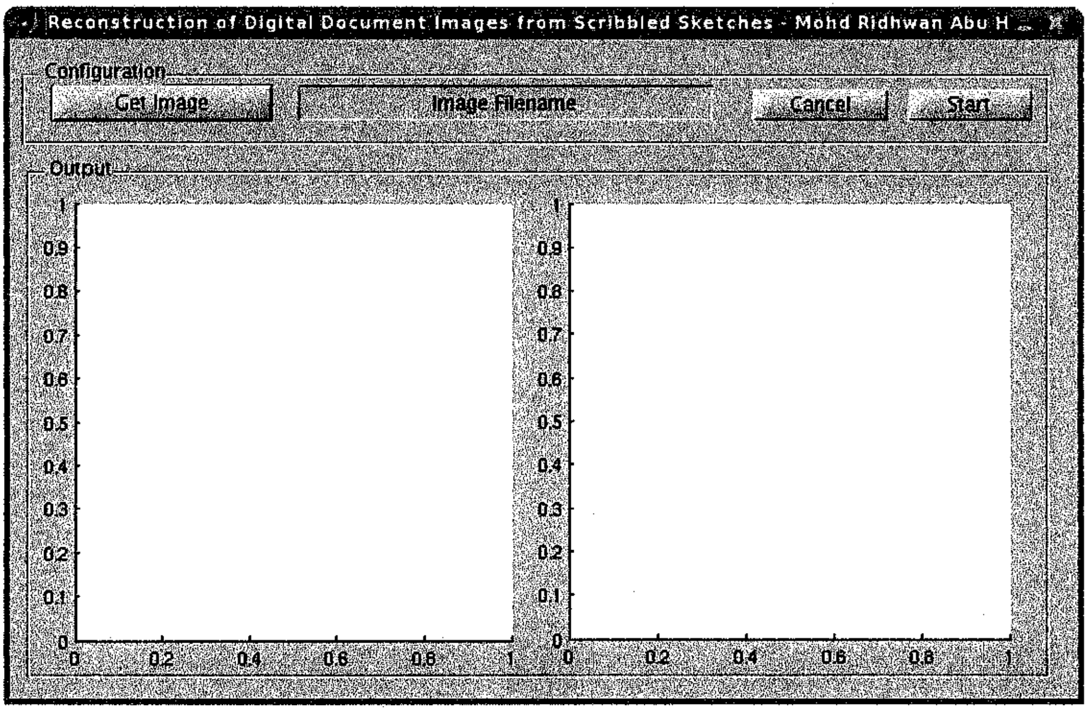


Figure 5 *Layout of the proposed prototype GUI that will be implemented in MATLAB.*

Button GET IMAGE will procure the digitized input image from local storage. User will perform manual identification of defective regions by single clicking on a sample pixel of defective regions on the left panel before clicking the button START, upon which the reconstruction process will begin. Button CANCEL will stop ongoing reconstruction process. A context menu, available with right-clicking on the right panel will allow the user to export the reconstructed output image into variety of image formats or to the printer.

3.4 Project Planning

The author divided the planning for this project into two (2) phases.

1. The first part involves extensive research on image inpainting and defects detection. The author makes use of available online library such as IEEExplore, ACM Digital Library and indexing website such as Google Scholar. The author also utilizes the available time to learn programming using MATLAB. Throughout this phase, the author produces various documentations representing his progress with this project.
2. The second part involves software program development. The author codes the program mostly in MATLAB and some in C++. The GUI for the software program is developed in MATLAB as well.

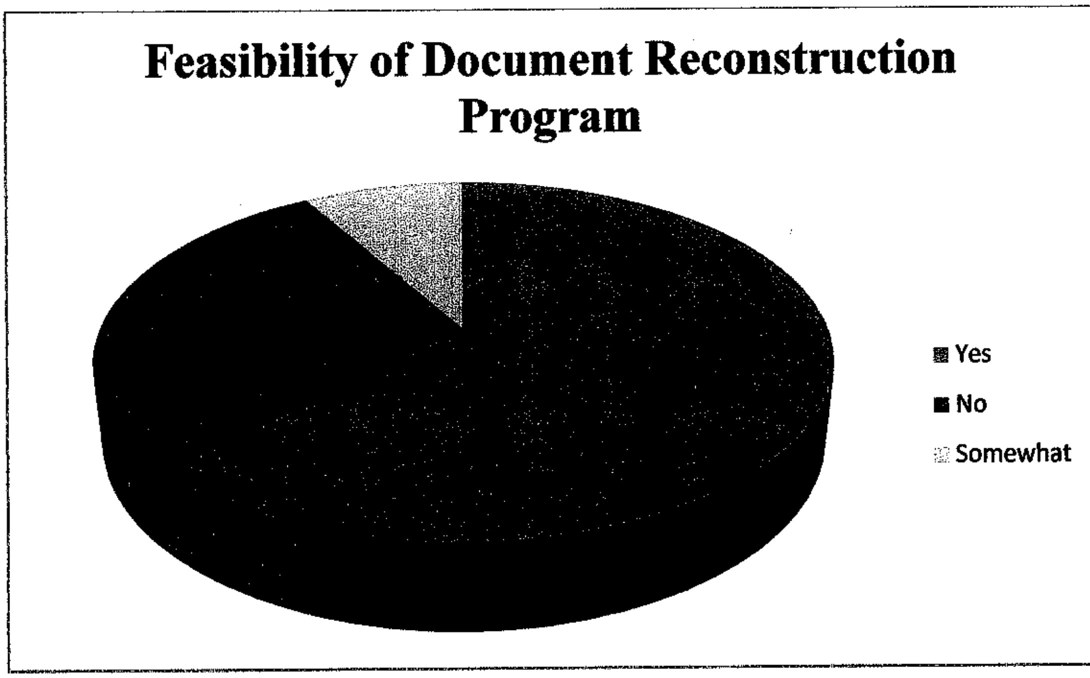
For references to project activities, key milestones and period allocated for each tasks, refer to the appendix.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Project Feasibility

Based on the feedback from 50 correspondents who gave their view on whether this project will greatly solve their problem, should they encounter a particular situation where, given a piece of paper (document) with handwriting overlaid by sketches which render the document to be difficult to read, will the digitized piece of paper (digital document images) help them solve this predicament?



4.2 Project Deliverables

The software program is completed with some minor features exclusion from the proposed GUI:

1. The context menu for reconstructed output image is not implemented. However, the reconstructed output image will be stored in the same folder where the input image with defects resides, with the standard format that is PNG.
2. The CANCEL button to stop the ongoing reconstruction process will not be implemented. However, in future where the input image will be bigger in size and resolution, or it will be in video format, which will render the

reconstruction process to be marginally longer, then only the CANCEL button will be implemented.

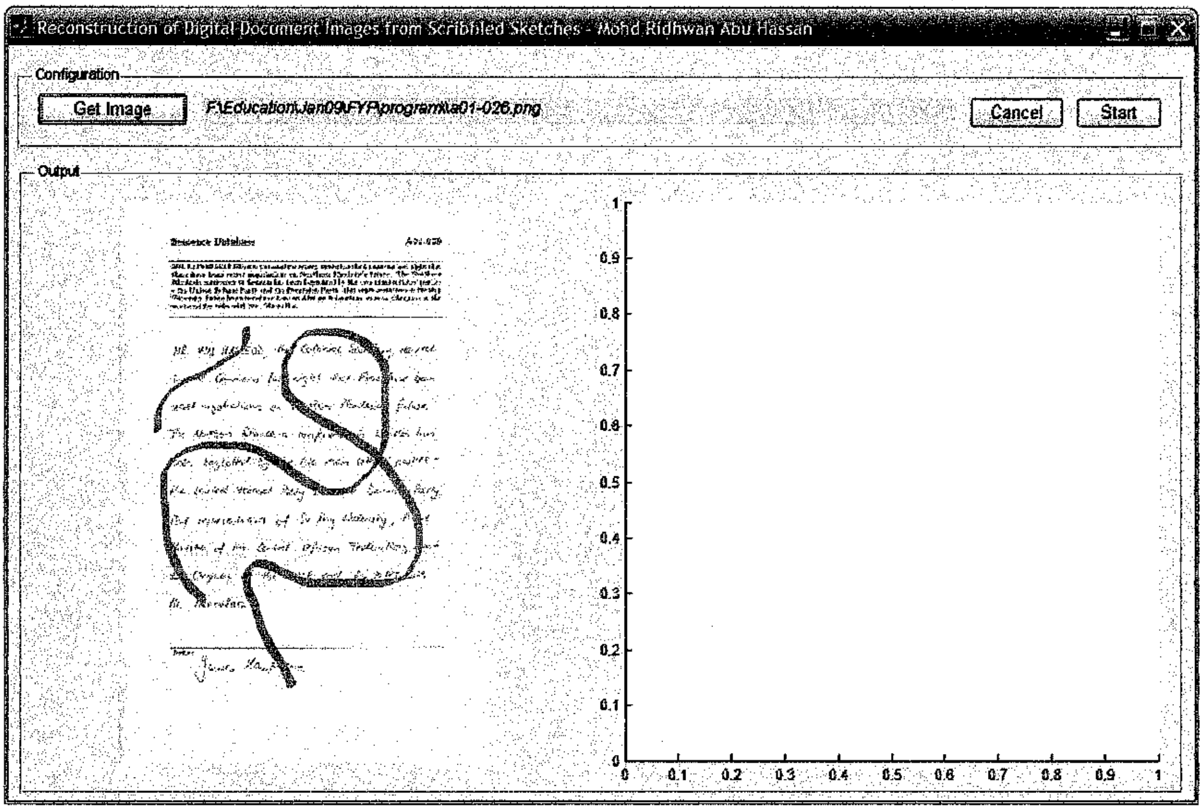


Figure 6 *The software program GUI shown with the input image with defects on the left panel. This software program can be distributed into various platforms without having to have MATLAB installed.*

The software program can also be executed through command line with the following output upon successful reconstruction.

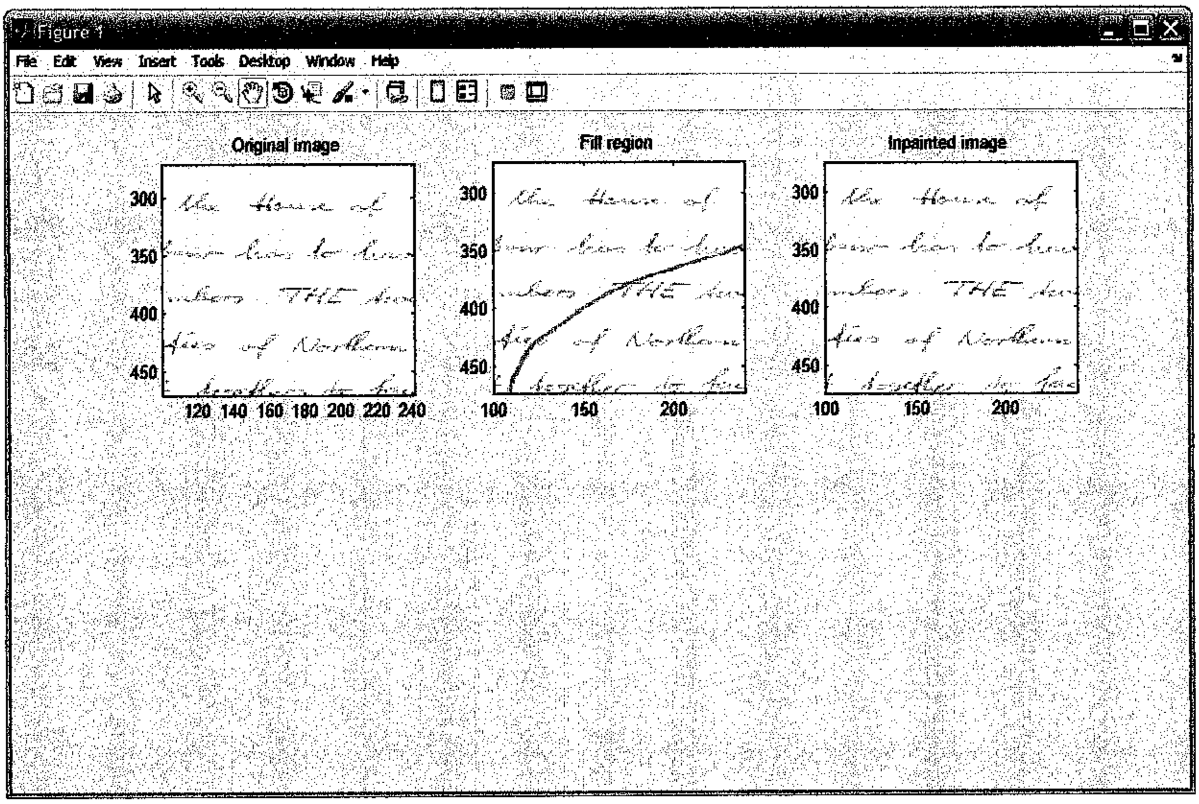


Figure 7 *The output of the reconstruction program from command line in MATLAB environment. This can only be achieved with MATLAB installed on the platform.*

4.3 Measurement of Performance and Accuracy

The authors performed analysis of the accuracy and performance of the software program as described below:

1. Measurement of Performance

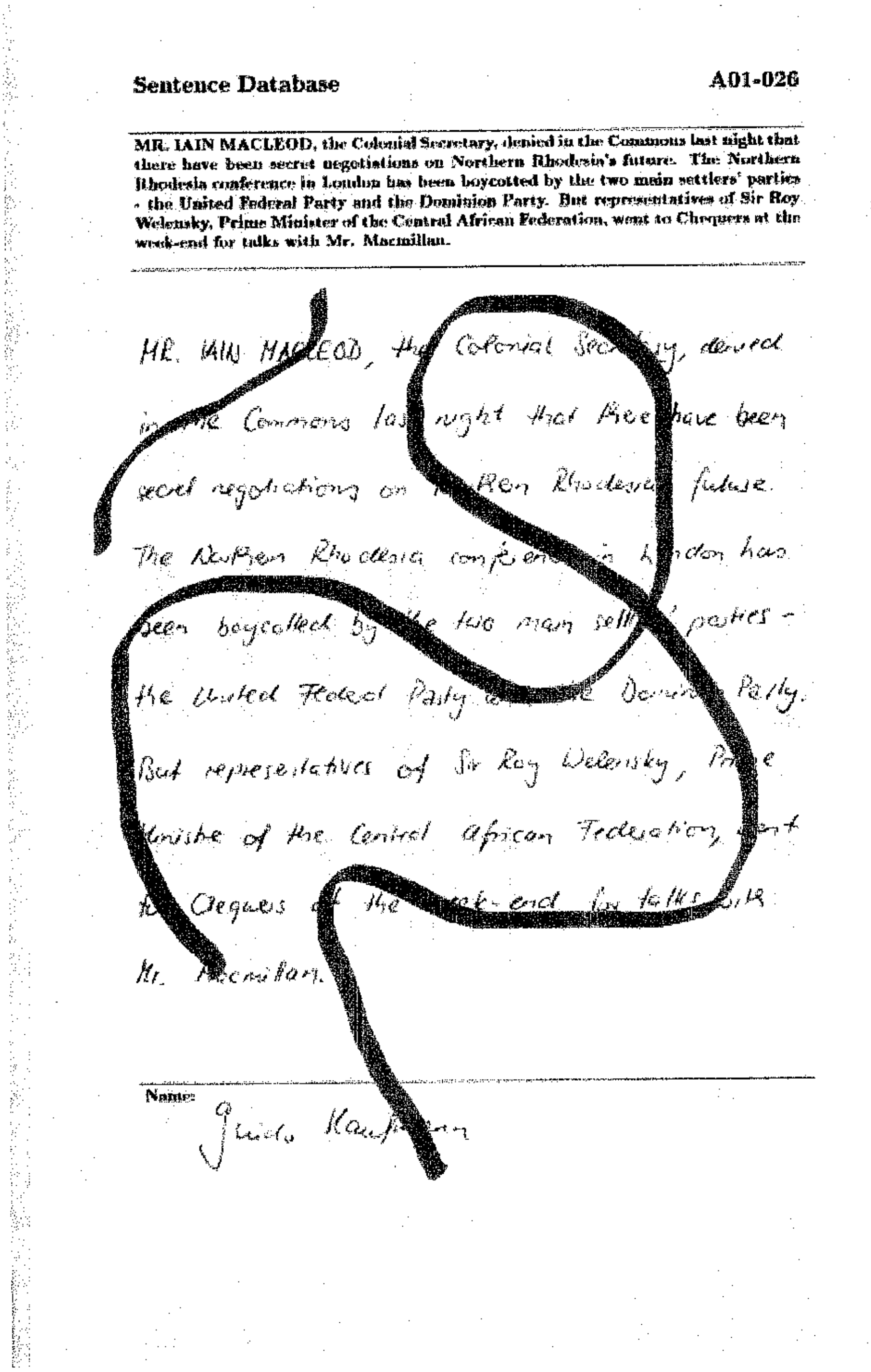
   At the end stage of software development, the author noticed that for images with higher resolution and dimension, the time taken to complete the reconstruction process will be longer. The analysis done is described in the following table.

Test Sample	Image I.D.	Time taken (seconds)
Sample A	A01-001.tiff	1.3
	A01-002.tiff	1.5
Dimension: 480x540px	A01-003.tiff	2.2
Resolution: 72 dpi	A01-004.tiff	1.1
Bit Depth: 24	A01-005.tiff	1.5
Sample B	B01-001.tiff	3.5
	B01-002.tiff	4.0
Dimension: 560x800 px	B01-003.tiff	4.1
Resolution: 80 dpi	B01-004.tiff	3.8
Bit Depth: 24	B01-005.tiff	4.1
Sample C	C01-001.tiff	4.5
	C01-002.tiff	4.8
Dimension: 850x900 px	C01-003.tiff	4.7
Resolution: 100 dpi	C01-004.tiff	4.8
Bit Depth: 24	C01-005.tiff	4.8
Sample D	D01-001.tiff	5.9
	D01-002.tiff	6.0
Dimension: 768x1024 px	D01-003.tiff	6.5
Resolution: 120 dpi	D01-004.tiff	4.3
Bit Depth: 24	D01-005.tiff	4.8
Sample E	E01-001.tiff	10.3
	E01-001.tiff	9.5
Dimension: 1200x1560 px	E01-001.tiff	8.5
Resolution: 150 dpi	E01-001.tiff	9.8
Bit Depth: 24	E01-001.tiff	8.8

Table 1 *Result from analysis of reconstruction performance over various test samples*

2. Measurement of Accuracy

   The author compares between the original transcript of handwritten document with the transcript of reconstructed document. For a sample of 10 document images, the average accuracy fluctuates between 70% to 80%.



Sentence Database A01-026

MR. IAIN MACLEOD, the Colonial Secretary, denied in the Commons last night that there have been secret negotiations on Northern Rhodesia's future. The Northern Rhodesia conference in London has been boycotted by the two main settlers' parties - the United Federal Party and the Dominion Party. But representatives of Sir Roy Welensky, Prime Minister of the Central African Federation, went to Chequers at the week-end for talks with Mr. Macmillan.

Name:

Figure 8 *The sample test image with defects*

```
MR. IAIN MACLEOD, the Colonial Secretary, denied
in the Commons last night that there have been
secret negotiations on Northern Rhodesia's future.
The Northern Rhodesia conference in London has
been boycotted by the two main settlers' parties -
```

the United Federal Party and the Dominion Party. But representatives of Sir Roy Welensky, Prime Minister of the Central African Federation, went to Chequers at the week-end for talks with Mr. Macmillan.

Table 2 *The transcript of the sample test image*

The accuracy is determined by comparing letters-by-letters of the input image with defects with the reconstructed image.

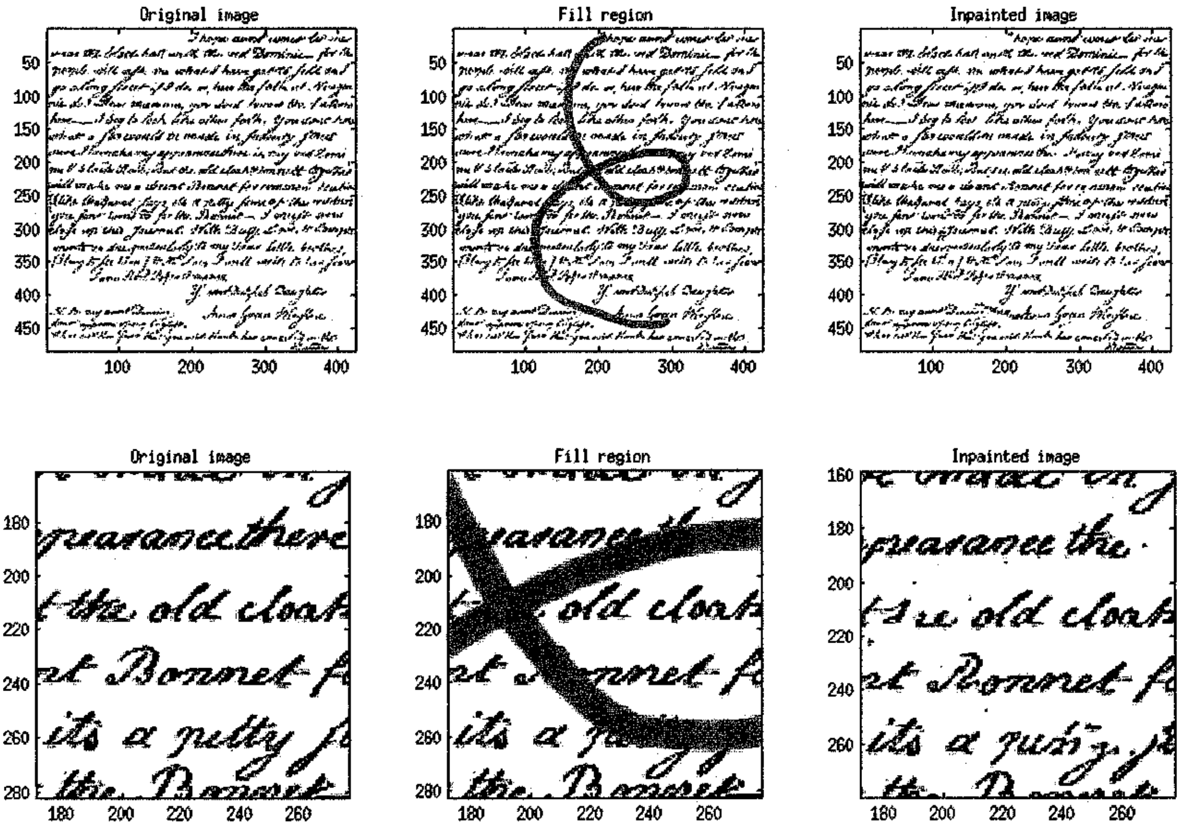


Figure 9 *The comparison between original image and reconstructed image to determine the percentage of accuracy*

4.4 Software Testing

Software testing is an empirical investigation conducted to provide stakeholders (users and developers) with the information about the quality of the product under test, with respect to the context in which it is intended to operate. It also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. The author conduct two (2) types of testing as described below:

1. Black box testing

   Treats the software as a "black box", that is, without any knowledge of the internal implementation. Black box testing has the advantage of an unaffiliated opinion of the software program. The author provides specification-based test cases for the users to follow through and provide their feedback. Ten (10) external users participated in the software testing procedures with 100% PASSED result. Refer to TABLE 3 for the test case used.

No.	**Test case name**	**Test procedure**	**Pre-condition**	**Expected result**	**Result**
1	Input_OK	Input image with defects through GUI	Digitized digital document image is available	Input image with defects will be displayed on the left panel	PASSED
2	Image_Defects	The user clicks the START button without first identifying the defective regions	Digitized digital document image has been successfully loaded	Prompt an error message. Reconstruction process does not start, pending user identification of defective regions	PASSED
3	Start_OK	The user clicks START button after identifying the defective regions	The defective regions have been identified manually by the user using left-mouse button	The GUI will display a "busy" text at the bottom panel. The reconstruction process will begin.	PASSED
4	Recon_OK	The user waits for reconstruction process to finish. Clicks on the prompt message box when it appears	The reconstruction process started successfully	The output result will display desired result with most if not all of the defects removed and reconstructed accurately making the handwriting readable	PASSED

Table 3 *Test Case for Black Box testing*

2. White box testing

   White box testing is a test procedure where the tester has access to the internal data structures and algorithms including the code that implement these. It can also be used to evaluate the completeness of the test suite that was created with black box testing methods. The authors conducted two (2) types of white box testing described below, which both yielded PASS result:

   a. Application Programming Interface (API) testing

      Tested the program for its robustness, in terms of, its function interfacing with C++

   b. Fault Injection

      Tested the program for its capability of handling faults to the test code paths. Here, the author introduced an invalid input image and evaluate how it handled this exception error.

CHAPTER 5

CONCLUSIONS

The final outcome of the project is able to adhere to the objectives which are:

1. Provide a semi-automatic identification of defective regions in digital images
2. Fill empty holes of the removed regions by synthesizing neighboring regions

The software program is able to accomplish all the required specifications and is robust after rigorous testing and further enhancement post-testing.

Although the exemplar-based inpainting yielded promising results with reasonably good accuracy and performance, implementation on digital images with handwriting exhibits some artifacts, which has been discussed in the previous section. The same problem also occurs in the implementation by the authors of (Criminisi, 2003) and (Bertalmio, 2001). This area of research which receive good research interest in the computer vision, image processing and machine learning research community over the past years, have a long way to go for the reconstruction to be perfect.

The author proposes the following for future works:

1. Extend the functions and capabilities of the program to include other image and video processing task. This will allow the software program to be commercializable, as what the premier Adobe© Photoshop proved so far.
2. Create a plug-in for Adobe© Photoshop or GNU Image Processing (GIMP) to extend their capability in image processing.
3. Implement a fully automatic defects detection using machine learning tools such as graphical models, support vector machine and others.

BIBLIOGRAPHY

Alexie A. Efros, T. K. (1999). Texture Synthesis by Non-Parametric Sampling. *International Conference on Computer Vision.* Greece: IEEE.

Bertalmio, M. (2001). Image Inpainting. *Graphics and Interactive Techniques (SIGGRAPH)* (pp. 417-424). Los Angeles: ACM.

Constructive Research - Wikipedia, the free encyclopedia. (n.d.). Retrieved April 25, 2009, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Constructive Research

Criminisi, A. (2003). Object Removal by Exemplar-Based Inpainting. *Computer Vision and Pattern Recognition (CVPR)* (pp. 721-728). Madison: IEEE.

Inpainting - Wikipedia, the free encyclopedia. (2009). Retrieved May 2, 2009, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Inpainting

Julinda Gllavata, B. F. (2005). Adaptive Fuzzy Text Segmentation in Images with Complex Background Using Color and Texture. *Computer Analysis of Images and Patterns.* Versailles.

Jyotirmoy Banarjee, A. M. (2009). Contextual Restoration of Severely Degraded Document Images. *International Conference on Computer Vision and Pattern Recognition.* Miami: IEEE.

Libo Fu, W. F. (2005). A Robust Text Segmentation Approach in Complex Background Based on Multiple Constraints. *Pacific-Rim Conference on Multimedia.* Korea: IEEE.

Pratt, W. K. (2007). *Digital Image Processing* (4th Edition ed.). Wiley-Interscience.

Shen, J. (2003). Inpainting and the Fundamental Problem of Image Processing. *SIAM News , 36* (5).

Software development process - Wikipedia, the free encyclopedia. (n.d.). Retrieved April 24, 2009, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Software development process

Software Prototyping. (2009, August 11). Retrieved August 11, 2009, from Wikipedia, the Free Encyclopedia: http://en.wikipedia.org/wiki/Software_prototyping

Swee-Seong Wong, W. K. (2000). Color Segmentation and Figure-Ground Segregation of Natural Images. *International Conference on Image Processing.*

Ye Q, G. W. (2003). A Robust Text Detection Algorithms in Images and Video Frames. *Pacific-Rim Conference on Multimedia.* Singapore: IEEE.

APPENDIX A

MATLAB CODE #1: RECONSTRUCTION FUNCTION

```
Function [inpaintedImg,origImg,fillImg,C,D,fillMovie] =
inpaint7(imgFilename,fillFilename,fillColor)

warning off MATLAB:divideByZero

[img,fillImg,fillRegion] =
loadimgs(imgFilename,fillFilename,fillColor);

img = double(img);
origImg = img;
ind = img2ind(img);
sz = [size(img,1) size(img,2)];
sourceRegion = ~fillRegion;

% Initialize isophote values
[Ix(:,:,3) Iy(:,:,3)] = gradient(img(:,:,3));
[Ix(:,:,2) Iy(:,:,2)] = gradient(img(:,:,2));
[Ix(:,:,1) Iy(:,:,1)] = gradient(img(:,:,1));
Ix = sum(Ix,3)/(3*255); Iy = sum(Iy,3)/(3*255);
temp = Ix; Ix = -Iy; Iy = temp;  % Rotate gradient 90 degrees

% Initialize confidence and data terms
C = double(sourceRegion);
D = repmat(-.1,sz);
iter = 1;
% Visualization stuff
if nargout==6
  fillMovie(1).cdata=uint8(img);
  fillMovie(1).colormap=[];
  origImg(1,1,:) = fillColor;
  iter = 2;
end

rand('state',0);

% Loop until entire fill region has been covered
while any(fillRegion(:))
  % Find contour & normalized gradients of fill region
  fillRegionD = double(fillRegion);
  dR = find(conv2(fillRegionD,[1,1,1;1,-8,1;1,1,1],'same')>0);

  [Nx,Ny] = gradient(double(~fillRegion));
  N = [Nx(dR(:)) Ny(dR(:))];
  N = normr(N);
  N(~isfinite(N))=0; % handle NaN and Inf

  % Compute confidences along the fill front
  for k=dR'
    Hp = getpatch(sz,k);
    q = Hp(~(fillRegion(Hp)));
    C(k) = sum(C(q))/numel(Hp);
  end

  % Compute patch priorities = confidence term * data term
```

```
  D(dR) = abs(Ix(dR).*N(:,1)+Iy(dR).*N(:,2)) + 0.001;
  priorities = C(dR).* D(dR);

  % Find patch with maximum priority, Hp
  [unused,ndx] = max(priorities(:));
  p = dR(ndx(1));
  [Hp,rows,cols] = getpatch(sz,p);
  toFill = fillRegion(Hp);

  % Find exemplar that minimizes error, Hq
  Hq = bestexemplar(img,img(rows,cols,:),toFill',sourceRegion);

  % Update fill region
  toFill = logical(toFill);
  fillRegion(Hp(toFill)) = false;

  % Propagate confidence & isophote values
  C(Hp(toFill))  = C(p);
  Ix(Hp(toFill)) = Ix(Hq(toFill));
  Iy(Hp(toFill)) = Iy(Hq(toFill));

  % Copy image data from Hq to Hp
  ind(Hp(toFill)) = ind(Hq(toFill));
  img(rows,cols,:) = ind2img(ind(rows,cols),origImg);

  % Visualization stuff
  if nargout==6
    ind2 = ind;
    ind2(logical(fillRegion)) = 1;
    fillMovie(iter).cdata=uint8(ind2img(ind2,origImg));
    fillMovie(iter).colormap=[];
  end
  iter = iter+1;
end

inpaintedImg=img;

function Hq = bestexemplar(img,Ip,toFill,sourceRegion)
m=size(Ip,1); mm=size(img,1); n=size(Ip,2); nn=size(img,2);
best = bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion);
Hq = sub2ndx(best(1):best(2),(best(3):best(4))',mm);

function [Hp,rows,cols] = getpatch(sz,p)
% [x,y] = ind2sub(sz,p);  % 2*w+1 == the patch size
w=4; p=p-1; y=floor(p/sz(1))+1; p=rem(p,sz(1)); x=floor(p)+1;
rows = max(x-w,1):min(x+w,sz(1));
cols = (max(y-w,1):min(y+w,sz(2)))';
Hp = sub2ndx(rows,cols,sz(1));

function N = sub2ndx(rows,cols,nTotalRows)
X = rows(ones(length(cols),1),:);
Y = cols(:,ones(1,length(rows)));
N = X+(Y-1)*nTotalRows;

function img2 = ind2img(ind,img)
for i=3:-1:1, temp=img(:,:,i); img2(:,:,i)=temp(ind); end;

function ind = img2ind(img)
s=size(img); ind=reshape(1:s(1)*s(2),s(1),s(2));
```

```
function [img,fillImg,fillRegion] =
loadimgs(imgFilename,fillFilename,fillColor)
img = imread(imgFilename); fillImg = imread(fillFilename);
fillRegion = fillImg(:,:,1)==fillColor(1) & ...
    fillImg(:,:,2)==fillColor(2) & fillImg(:,:,3)==fillColor(3);
```

APPENDIX B

MATLAB CODE #2: GUI

```
function varargout = gui(varargin)
% UNTITLED1 M-file for untitled1.fig
% UNTITLED1, by itself, creates a new UNTITLED1 or raises the
existing singleton*.
% H = UNTITLED1 returns the handle to a new UNTITLED1 or the handle
to the existing singleton*.
% UNTITLED1('CALLBACK',hObject,eventData,handles,...) calls the
local
% function named CALLBACK in UNTITLED1.M with the given input
arguments.
% UNTITLED1('Property','Value',...) creates a new UNTITLED1 or
raises the
% existing singleton*.  Starting from the left, property value pairs
are
% applied to the GUI before untitled1_OpeningFcn gets called.  An
% unrecognized property name or invalid value makes property
application
% stop.  All inputs are passed to untitled1_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help untitled1
% Last Modified by GUIDE v2.5 13-Sep-2009 11:54:34
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_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 untitled1 is made visible.

function gui_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 untitled1 (see VARARGIN)
% Choose default command line output for untitled1
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes untitled1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui_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 axes1_pushbutton.
function axes1_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to axes1_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% gets input file(s) from user
[filename,pathname] = uigetfile( ...
       {'BMP (*.bmp)', 'PNG (*.png)'; ...
        '*.*', 'All Files (*.*)'}, ...
        'Select files', ...
        'MultiSelect', 'on'); % select the input image to be
inpainted in axis1

 if ~ischar(filename)
     errordlg('Error!','No file selected');
     return
 end

 fullpathname = [pathname, '\' , filename];

 I = imread(fullpathname);
 set(handles.text1, 'String', fullpathname);
 axes(handles.axes1);
 imshow(I);

 guidata(hObject, handles);

% --- Executes on button press in axes2_pushbutton.
function axes2_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to axes2_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[i1,i2,i3,c,d,mov] = inpaint7('a01-007.tif','a01-007.png',[0 255
0]);
axes(handles.axes2);
imshow(i1);

guidata(hObject, handles);

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

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


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

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


% --- 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');
```

APPENDIX C

C++ CODE #1: EXEMPLARS SEARCH

```
#include "mex.h"
#include <limits.h>

void bestexemplarhelper(
      const int mm,
      const int nn,
      const int m,
      const int n,
      const double *img,
      const double *Ip,
      const mxLogical *toFill,
      const mxLogical *sourceRegion,
      double *best)
{
  register int i,j,ii,jj,ii2,jj2,M,N,I,J,ndx,ndx2,mn=m*n,mmnn=mm*nn;
  double patchErr=0.0,err=0.0,bestErr=1000000000.0;

  /* foreach patch */
  N=nn-n+1;  M=mm-m+1;
  for (j=1; j<=N; ++j) {
    J=j+n-1;
    for (i=1; i<=M; ++i) {
      I=i+m-1;
      /*** Calculate patch error ***/
      /* foreach pixel in the current patch */
      for (jj=j,jj2=1; jj<=J; ++jj,++jj2) {
      for (ii=i,ii2=1; ii<=I; ++ii,++ii2) {
        ndx=ii-1+mm*(jj-1);
        if (!sourceRegion[ndx])
          goto skipPatch;
        ndx2=ii2-1+m*(jj2-1);
        if (!toFill[ndx2]) {
          err=img[ndx      ] - Ip[ndx2    ]; patchErr += err*err;
          err=img[ndx+=mmnn] - Ip[ndx2+=mn]; patchErr += err*err;
          err=img[ndx+=mmnn] - Ip[ndx2+=mn]; patchErr += err*err;
        }
      }
      }
      /*** Update ***/
      if (patchErr < bestErr) {
      bestErr = patchErr;
      best[0] = i; best[1] = I;
      best[2] = j; best[3] = J;
      }
      /*** Reset ***/
    skipPatch:
      patchErr = 0.0;
    }
  }
}

/* best = bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion);
*/
```

```
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray
*prhs[])
{
  int mm,nn,m,n;
  double *img,*Ip,*best;
  mxLogical *toFill,*sourceRegion;

  /* Extract the inputs */
  mm = (int)mxGetScalar(prhs[0]);
  nn = (int)mxGetScalar(prhs[1]);
  m  = (int)mxGetScalar(prhs[2]);
  n  = (int)mxGetScalar(prhs[3]);
  img = mxGetPr(prhs[4]);
  Ip  = mxGetPr(prhs[5]);
  toFill = mxGetLogicals(prhs[6]);
  sourceRegion = mxGetLogicals(prhs[7]);

  /* Setup the output */
  plhs[0] = mxCreateDoubleMatrix(4,1,mxREAL);
  best = mxGetPr(plhs[0]);
  best[0]=best[1]=best[2]=best[3]=0.0;

  /* Do the actual work */
  bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion,best
```

APPENDIX

GANTT CHART

(Refer to next page)

PART ONE

ID	Task Name	Start	Finish	Duration
1	Preliminary Research	1/19/2009	2/20/2009	5w
2	Submit Preliminary Report	2/18/2009	2/18/2009	0w
3	Research on defect detection	2/20/2009	4/13/2009	7.4w
4	Seminar 1	2/23/2009	2/27/2009	1w
5	Submit Progress Report	3/11/2009	3/11/2009	0w
6	Seminar 2	3/16/2009	3/20/2009	1w
7	Construct MATLAB codes for defect detection and texture synthesis	4/13/2009	6/22/2009	10.2w
8	Submit Interim Report	5/22/2009	5/22/2009	0w
9	Oral Presentation	5/27/2009	5/27/2009	.2w

Feb 2009 · Mar 2009 · Apr 2009 · May 2009 · Jun 2009

1/25 · 2/1 · 2/8 · 2/15 · 2/22 · 3/1 · 3/8 · 3/15 · 3/22 · 3/29 · 4/5 · 4/12 · 4/19 · 4/26 · 5/3 · 5/10 · 5/17 · 5/24 · 5/31 · 6/7 · 6/14 · 6/21

PART TWO

ID	Task Name	Start	Finish	Duration
1	Continuation with software development	6/30/2009	7/22/2009	3.4w
2	Submit Progress Report 1	7/22/2009	7/22/2009	0w
3	Continuation with software development	7/22/2009	10/7/2009	11.2w
4	Submit Progress Report 2	9/24/2009	9/24/2009	.2w
5	Seminar 3	9/25/2009	9/25/2009	0w
6	Pre-EDX and Poster Exhibition	10/7/2009	10/7/2009	.2w
7	Finish software development	10/12/2009	10/12/2009	.2w
8	Submit Dissertation (soft-bound)	10/12/2009	10/12/2009	0w
9	Oral Presentation	10/28/2009	10/28/2009	.2w
10	Submit Finalized Dissertation (hard-bound)	11/16/2009	11/16/2009	.2w

Jul 2009 · Aug 2009 · Sep 2009 · Oct 2009 · Nov 2009

7/5 · 7/12 · 7/19 · 7/26 · 8/2 · 8/9 · 8/16 · 8/23 · 8/30 · 9/6 · 9/13 · 9/20 · 9/27 · 10/4 · 11/1 · 11/8