Shanghai Jiao Tong University

Software Engineering Report

Image Inpainting Software

Yesheng Ma, Hu Hu, Yikai Zou

supervised by
Prof. Bin Sheng
TA Saleha

January 6, 2017

Chapter 1

Introduction

Abstract

In this part, we will briefly discuss the purpose of this project, the scope of this project, overview of this document, and some introduction on the development tools and teamwork integration.

1.1 Purpose of the project

This project is the course project of software engineering at Shanghai Jiao Tong University. The purpose of this project is to build a working software for image inpainting. In this project specification, we will mainly give the requirements specification, which includes 1. functional requirements 2. non-functional requirements 3. domain requirements; the software design, which includes 1. software model 2. software development tools 3. architectural design 4. object identification; testing, which includes 1. test plan 2. test-design specification 3. test-case specification 4. test-procedure specification; as well as conclusion and references.

1.2 Scope of the project

1.2.1 Project goal and task

The goal and purpose of this project is shown as follows:

1. Implement image in-painting algorithm. When it is given an image and a chosen area, it can remove the object in the area and inpaint the image.
2. Both implement the project in a traditional manner and with a probabilistic graphical model, these two models should exhibit their benefits in different situations.
3. Finally translate the prototype program into a GUI application with feasible user interaface, so that users can easily inpaint a target image.

Major Mission	Date	Finished or Not
Paper research, UML design, Requirement analysis	9/15 - 11/16	finished
Prototype design, paper proposal	11/17 - 11/30	finished
Backend development of exemplar-based algorithm	12/1 - 12/11	finished
Backend development of Markov random field	12/11 - 12/25	finished
Frontend development and user interface	12/25 - 12/28	finished
Test and refactoring	12/28 - 1/3	finished
final presentation	1/4 - 1/4	finished
final report and paper writing	1/4 - 1/6	finished

Table 1.1: Time schedule of this project

1.2.2 Project cost

In this project, there is no too much money cost needed for the exemplar-based method. However, in order to train that probabilitic graphical model, i.e. Markov random field model, we need some number of dataset. As we searched on the internet, there are not large dataset for image inpainting and the only existing one is a dataset provided by UC Berkeley with 300 inpainted images and we will use this dataset to train the model.

1.2.3 Schedule for this project

The concreate timeline for this project is shown in the Table 1, the deadline of this project is Jan 7, 2017.

1.2.4 Hardware and software resources

In this project, we have to deal with hardware and software resources since in a software development, these resources are critical.

Hardware:

1. Lab cluster to train MRF model
2. 3 laptops for software development

Software:

1. use Python/MATLAB as backend programming language
2. use Python Tkinter cross platform GUI framework and MATLAB builtin GUI utility to build user interface

1.3 Overview

In this final report, we will first talk about the requirement of image inpainting. Image inpainting itself is as old as human art and it is really a topic worth hard working. Actally, the image inpainting is quite useful for ancient art museums.

Next, we will discuss the architecuture of this project in a software enginnering manner. We will discuss why we design the image inpainting system in that way and how we overcome the difficulties during developing this system. We will also present some UML diagrams to illustrate our design formally.

Also, we will talk about how our users can use this image inpainting software and what is the common work flow to use our image inpaiting software. Although our user interface is quite user-friendly, users may still be confused by the software since the technique we used to implement image inpainting is quite complex.

Moreover, test specification is also given. Any modern reliable software should be tested over multiple test cases and only in this way can our software present good reliability.

At last, we will conclude this project and give several topics for future work since we find there might be more things we can discover in this research field and we will give some references of this project.

1.4 Development environment & teamwork integration

1.4.1 Development environment

We implemented two different algorithms: exemplar-based algorithm and Markov random field in two platforms. We implemented the exemplar-based algorithm on Windows 10 operating system and the MRF-based algorithm is implemented on Ubuntu 16.04 Linux OS.

The proramming languages we use are carefully chosed. We use Python for the exemplar-based algorithm since Python code is easy to write and there are useful libraries like Tkinter, PIL, numpy, and scipy. We implement the Markov random field algorithm in MATLAB since it is a machine learning, to be specific a probabilistic graphical model, problem and we can use MATLAB to easily implement matrix operations and some optimization algorithms like SGD(stochastic gradient descent).

1.5 Teamwork integration

Since this is a quite large programming task and we work in a team to contribute to this project, we decide to use Git version control system to help us develop our project and actually we find we can further save our source code on cloud for convenience and safety.

The most convenient component of GitHub might be the issue tracker. Once a contributor discovers an issue, he can write a issue and another contributor can work on this issue and fix that bug. A sample issue is shown as follows: Altogether we make about 50 commits and roughly 15000 lines of code.

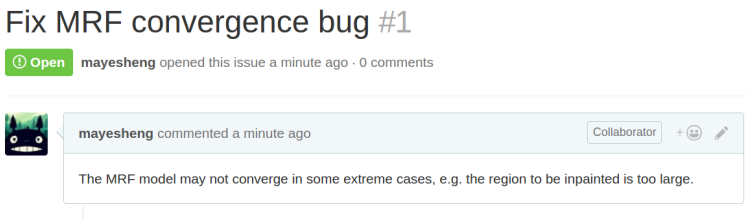


Figure 1.1: A sample issue tracker

Chapter 2

Requirement Analysis

Abstract

In this document, we will mainly discuss the requirements for our image inpainting software. We discuss both the functional and non-functional requirements as well as a specific hardware and software environment requirement and other necessary parts for our image inpainting ysstem.

2.1 Introduction to requirement analysis

As Wikipedia says: in software engineering, requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.

Thus requirement analysis is critical to the success of our image inpainting software. Also, the requirements should be well documented, actionable, measurable, traceable, and defined in detail.

In this following sections, we will discuss the requirements of this image inpainting software.

2.2 System Overview

2.2.1 Definitions

First, we should give a definition of image inpainting and this is a first solid step to the success of our image inpainting software.

Most of you will have some old damaged photos at your home with some black or gray spots on it. It is really a real-world requirement to fix that image in an acceptable way. We can't simply erase them in a single color because

it will simply replace the damaged region which is useless. In these cases, a technique called image inpainting is used. Look at the image bellow:

Figure 2.1: An example of image inpainting

We can see that the left picture is damaged and we want to fix those damaged regions. The right-hand-side picture is the in painted image, which is what we desire to get.

2.2.2 History of image inpainting

Inpainting, the technique of modifying an image in an undetectable form, is as ancient as art itself. Actually there are many professional image restorers in museums of ancient arts. Though the history of image restoration is quite long, computer-aided image inpainting appears quite late.

To our surprise, research on image inpainting emerged very late. In SIGGRAPH 2000 [1], Bertalmio proposed an algorithm to inpaint a image: after the user selects the regions to be restored, the algorithm automatically fills in these regions with information surrounding them. The fill-in is done in such a way that isophote lines arriving at the regions boundaries are completed inside.

Roth and Black [5] later come up with a more general framework called *Field of Experts* which is a Markov random field model. This work relies on the result of Hinton [4], where Hinton discovers that a factor in MRF can be modeled by a field of "expert" distributions. In this work, they first learn the model and then apply the learned model to Bertalmio's propagation method. Although this work in some sense captures image structure, the blurring problem is also serious after our experiment.

Another different way to tackle this problem is proposed by Criminisi et al. [2, 3]. They note that exemplar-based texture synthesis contains the essential process requred to replicate the structure of image. They introduce a priority for each exemplar and propagate according to the priority of each exemplar. This technique does not have the problem of blurring and can restore texture well, but may still fail for large object removal.

All these previous works shows that image inpainting is actually a really challenging job and to the best of our knowledge, there haven't been any practical software for image inpainting on market. Thus we will work hard on this

project and finally open source this work on web so that other people can make use of it.

2.3 Detailed system requirements

2.3.1 Computation platforms

The recommended computation platform is shown as follows:

Hardware	Requirement
CPU	Intel Core i5 2GHz
Memory	1GB
Disk	100MB
GPU	N.A.

Table 2.1: Hardware requirement spec

Since to train the MRF model is CPU bound, we need the CPU to be at least Intel Core i5, which has a typical 2 GHz frequency. Also, our model is not typically designed for GPU and thus there is no limitation to GPU. For memory, the computation of MRF model requires much matrix computation and thus a 1GB memory is required. At last, we need to store the Berkeley image inpainting database, which contains 300 inpainted images and 100MB disk storage is required.

2.3.2 Runtime environment requirement

We want our image inpainting software to work cross-platform, thus a Windows OS or Linux OS will make sense. An example OS is shown in the following figure:

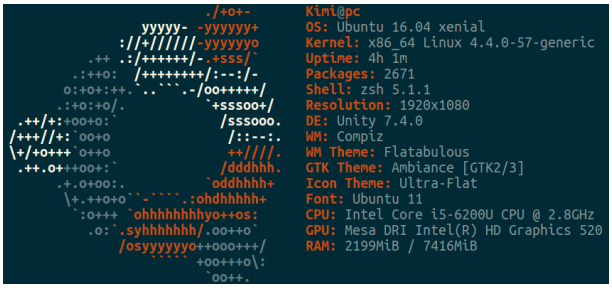


Figure 2.2: Example operating system specification

As regard to programming language environment, we need a minimal MATLAB 2014 version in order to run our code in Windows OS. We recommend you use Python 2.7 instead of Python 3 since the libraries we use might not work

fine with Python 3. Also, if you are using Linux, please make sure you have installed some cross-platform libraries such is python-tkinter so that our code can work properly.

2.4 Software Architecture requirements

Software architecture is the the fundamental structures of a software system, the discipline of creating such structures, and the documentation of these structures. Software architecture is so important that it may influence our development in all aspects.

We design the architecture of our image inpainting software to be a branch architecture. On the one hand, we implement a exemplar-based algorithm, which works fine for those images which explicit texture feature; on the other hand, we implement a MRF-based algorithm, which is based on probabilistic graphical model and works better for small damage inpainting. To combine these two components together, we have a unified user interface and it is easy for users to operate.

2.5 Requirement analysis models

2.5.1 Functional requirements

In this section, we will talk about the requirement of a real-world user. What's the requirements of a user to easily inpaint a image? We will list the requirements as follows:

- Input image: first, the user need to specify which image he/she is going to inpaint. In this case, the requirement is that we should provide a extra window to help the user browse in the file system to find the picture to be inpainted.
- Input mask: similarly, the user also need to specify the input mask so that our image inpainting software can know which part it need to inpaint.
- Inpainting progress: since image inpainting is not such an easy task, it may take minutes or even hours to finish inpainting a single image. Thus we provide the user with a window showing the progress of computation. This is a diagram that shows the convergence of our model, once the value of RGB channels all comes below 1, the image inpainting is finished.
- Inpainting result: once the task of image inpainting is finished, a window will display both the original image and the inpainted version. Also, we will save the inpainted image to local file system so that user can further take the inpainted image for future use.
- Error recovery: our model might not work for all images, in case our model does not fit well with the image given by user, we provide the user with

an amount describing how "good" our model performs on this image and it is up to the user himself whether to proceed to inpaint this image.

2.5.2 Non-functional requirements

Non-functional requirements can vary from software to software, we will only list some of the key non-functional requirements for our image inpainting software:

- Platform compatibility: our image inpainting software works well both on Windows OS and Linux, which are two major operating systems in use.
- Open source: since this is a course project and we do not want earn money from it, we will open source this project on GitHub.
- Maintainability: this project is highly modularized and we believe the code will be easy for other community contributors to read and modify.
- Response time: the execution time to inpaint an image may take 30 seconds to several minutes depending on the size to be inpainted. However, we believe our model works better than most of existing ones.
- Documentation: we mainly provide 4 documentation: 1. this report talks the design of this image inpainting software in detail; 2. a README file to instruct developers explore in this project; 3. a user guide for users who might not have a computer vision background; 4. a paper discussing the contribution of our work for those computer vision enthusiasts.

2.5.3 Domain requirements

We think the image inpainting software may apply to many different domains. We come up with some domains but applications should include but not restricted to these:

- Image restoration in museums of ancient arts
- Help police recover damaged images
- Help artists to remove texts from a image
- Help people fix damaged old family photographs

Chapter 3

Software Architecture

Abstract

This document is used to describe the software architecture of our image inpainting software. It is the guideline of our software developing and structure. We will illustrate clearly how we design and implement our image inpainting software. In this document , we will describe the detailed design idea and reader can have a clear picture with referring to the rational rose design model.

Keywords Image Inpainting Software, Architecture, Design, Software Structure

3.1 Introduction

3.1.1 Purpose

This document is the guideline of our software developing and structure. Tt provides a comprehensive architectural overview of our image inpainting software. It uses a number of different architectural views to depict different aspects of the system to help illustrate the software architecture. In this document , we will describe the detailed design idea and reader can have a clear picture with referring to the rational rose design model. It is intended to capture and convey the significant architectural decisions.

3.1.2 Scope

This document is an overview of the architecture and how it should be modeled. Decisions in this document affect how the system is modeled. Implementations of software is supposed to accord to this document.

3.2 Architectural Representation

In this document, the architecture of the application is represented following the recommendations of the Rational Unified Process and the Rational Architecture Practice guidelines.

Our image inpainting software implemented two different algorithm and users can choose different method to satisfy their requirement. In other document, we will illustrate clearly the detail algorithm of these two method. And in this document, we will describe the software architecture of our image inpainting software.

This document presents the architectural as a series of views:

1. Use-Case View
2. Logical View
3. Process Vied
4. Implement View
5. Deploy View

3.3 Use-Case View

The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of an iteration.

Our image inpainting software provide users freedom to choose the input image and the input mask. Users can choose their wanted area in the input image and our image inpainting software will inpaint the target area.There are different applicable scenario for out software including image text removal, object removal and image detail restoration etc. We also provide the users the UI to satisfy their requirements.

All implemented use cases also have an associated Us-Case Specification document. You can refer to the Use-Case Specification document and get more details.

The functionality of our image inpainting software is captured in the Use-Case graph below:

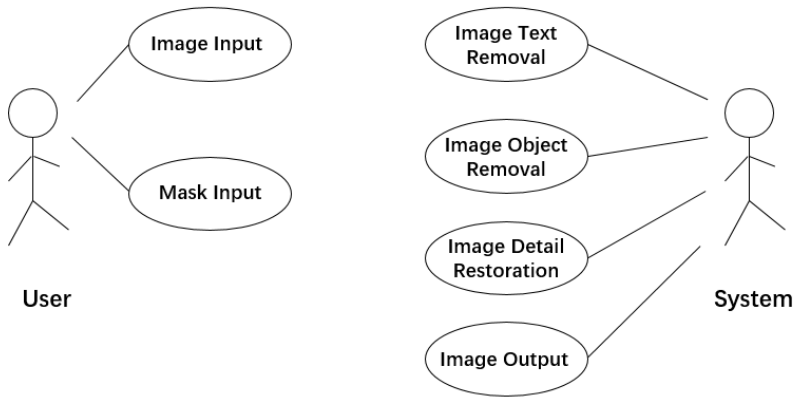


Figure 3.1: Use-Case View

3.4 Logical View

In this section, we will describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. This section will illustrate clearly the logical structure of the system. It presents its key structure, behavioral elements and and related structure.

There are three dominant structures in the application design model:

1. Logical decomposition of the system into three layers.
2. The structure of the use case realizations derived from model templates of architectural mechanisms.
3. The design mechanisms package contains the a pre-designed solution to a common problem.

The high-level diagram of above is showed in the following graph:

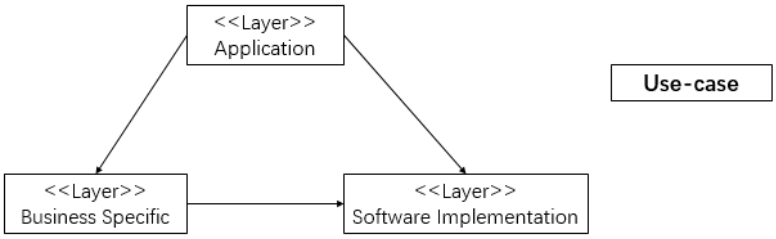


Figure 3.2: Logical View

In the logical view graph, It include three soft ware layers and the Use-Case realizations. These are illustrate below:

1. Application: This layer is for special logic. It has a close relation to the presentation logic. The boundary classes are contained in this layer. It includes our two different implementation algorithm: Markov random field and exemplar-based algorithm.

2. Business Specific: This layer includes business components and one common elements and services component. The components in this layer have a high reusability. It can be reuse in other situation or development process.

3. Software Implementation: This layer includes some basic implementation of our image inpainting software. It include the platform and the detail language of our software. In our image inpainting software, we use MatLab and Python as our development language, and use Ubuntu 16.04 and Windows 10 as our development platform.

4. Use-Case: It include the detail use-case situation of our image inpainting software.

3.5 Process View

In our image inpainting software, there are total one process. Users choose the image that they want to inpaint and the mast which covers the target area, then they can click the button in our UI. Then they will get the result.

The detail process will describe in the following graph:

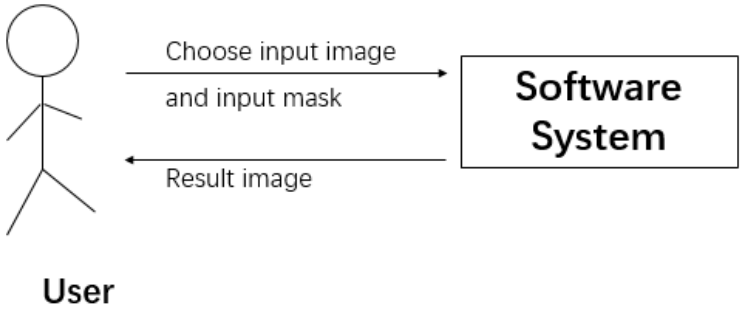


Figure 3.3: Process View

In the process, it includes the following steps:

1. Users choose the image that they want to inpaint and the mast
2. Our image inpainting software system will process the input image and input mask.
3. Software system presents the result image to users.

3.6 Development View

The deployment view of a system shows the physical nodes on which the system executes and the assignment of the system processed to the nodes.

The diagram below shows the most typical deployment configuration used by the development team.

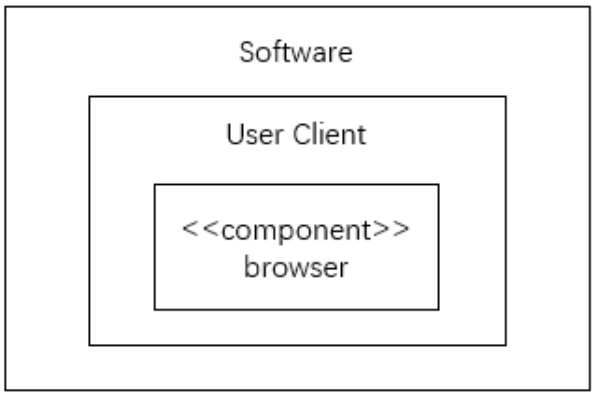


Figure 3.4: Development View

3.7 Implement View

This section include our development platform and our development language. Due to we implement two different algorithm, so we use different platform and development tools.

The detail is shown in the following graph:

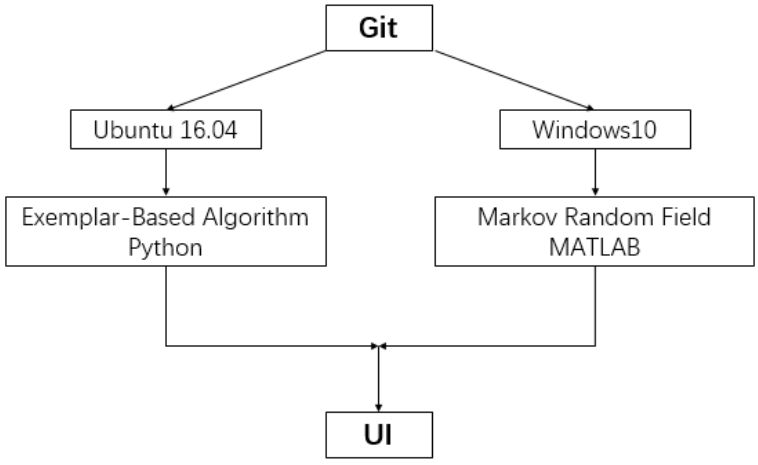


Figure 3.5: Implement View

As is shown in the graph, we use different tools to help our development including:

1. Git: It is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. We use Git to manage our software version and and tracking changes, which can support our development in different platform.

2. MatLab: MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. We use MatLab to implement our Markov Random Field Algorithm, which is much more convenient doing matrix operation.

3. Python: Python is a widely used high-level programming language used for general-purpose programming. It can provide us some basic image process library.We use python to implement our Exemplar-Based Algorithm and our final UI.

Chapter 4

Use Case Analysis

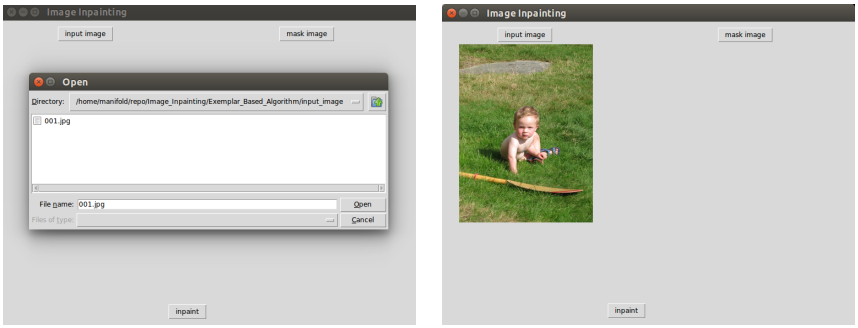
Abstract

In this part, we will mainly discuss how to use this software and we will demonstrate this process with a rich bunch of pictures. We will separately discuss the usage of the exemplar-based algorithm and the Markov random field method

4.1 Use case: exemplar-based algorithm

4.1.1 Input image use case

As is shown in the following figures, when we click the left button, we can select the input image from the file system of current user and after we selected the input image, the selected image will be immediately shown on the window, which provides perfect user interaction experience.

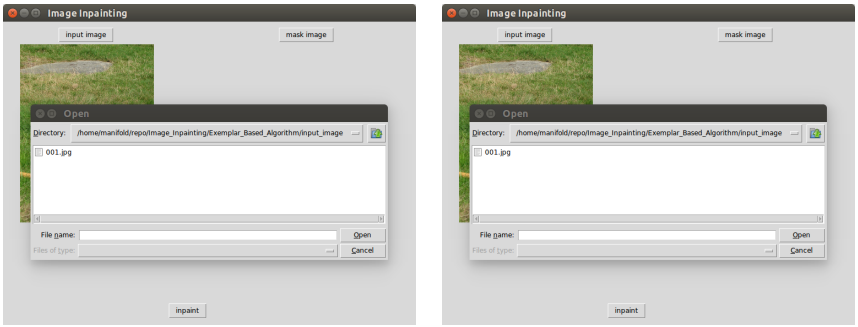


(a) Select input image (b) Display input image

Figure 4.1: Input image use case

4.1.2 Input mask use case

use not only need to provide an input image, but also an input mask so that the computer can get aware of which part should e inpainted. As well as the case of input image, we can select the desired mask from user file system and the mask will be shown on the screen as soon as the user chooses proper input mask.



(a) Select input mask (b) Display input mask

Figure 4.2: Input mask use case

4.1.3 Execution use case

Once we have inputed both the target image and input mask, we can click the "inpaint" button to start execution. During the execution of the inpainting program, the image inpainting software will regularly save the partially inpainted image to disk and we can further look at that image to checkout how much have been inpainted.

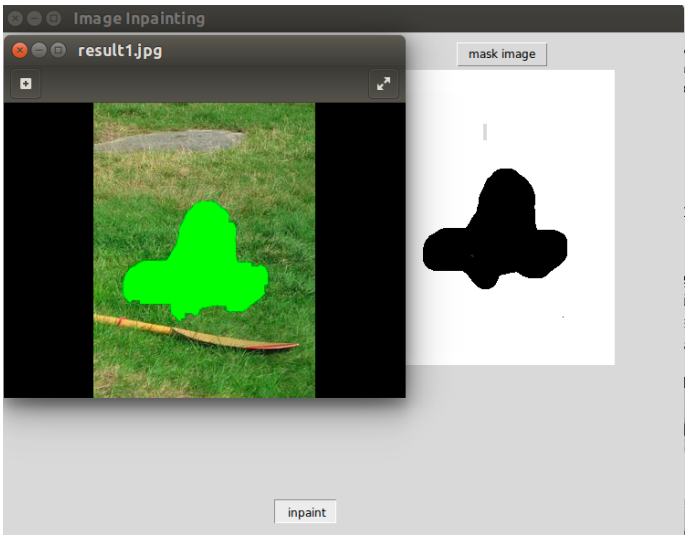


Figure 4.3: Image inpainting process

From the above image we can clearly see that some part of the image has already been inpainted

4.1.4 Result checkout use case

Once the inpainting process finishes, we can checkout the result. The following figure show the result of inpainting.

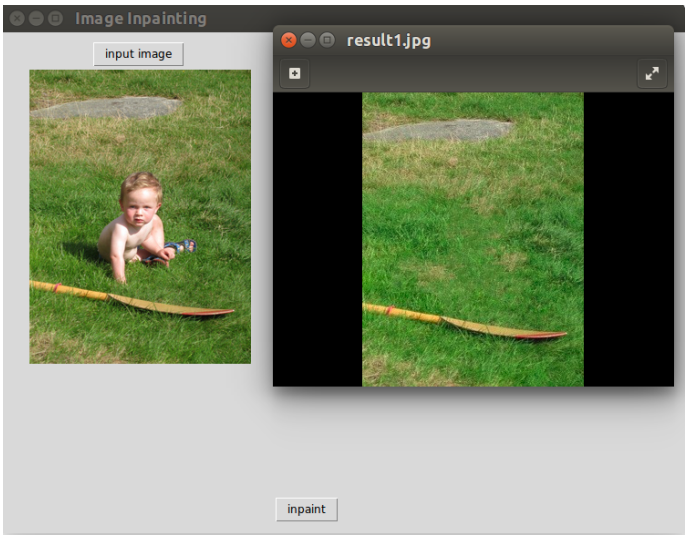


Figure 4.4: Image inpainting result

As we can see from this figure, the result of inpainted image is quite good.

4.2 Use case: Markov random field

In this section, we will discuss the use case of another completely different algorithm: Markov random field. Since it is convenient to read in a file in MATLAB, we will no longer discuss file IO in detail with respect to MRF method. We will mainly talk about the convergence here.

4.2.1 Monitor status

Once we hit the start key, the Markov random field model will begin to compute on this image. The computation will continue until the model converges.

However, the user may not be aware of how much the model has been fitted. So we provide a window to display the extent of convergence and we will update this window to keep it up-to-date.

At beginning, the trend of convergence might not seem obvious. As is shown in the following figure, there are three lines in the figure, which has colors of red, green, and blue, and you might cleverly discover that represents the convergence three different convergence trend of RGB(red, green, blue) channels.

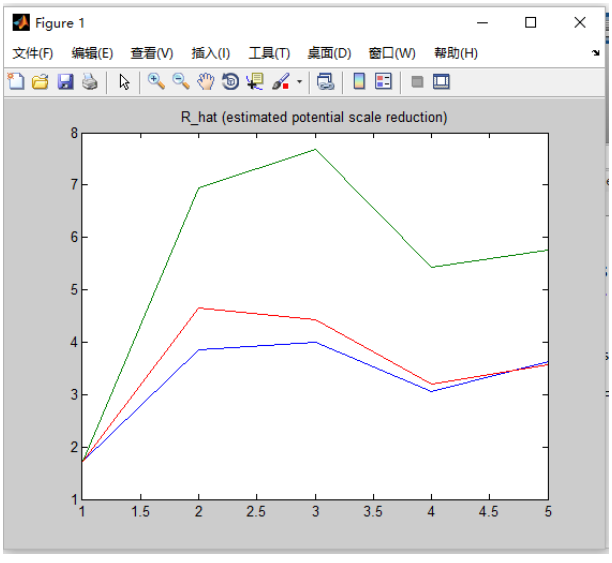


Figure 4.5: Beginning of convergence

As the process of computation continues, we will see that the value $\hat{R}$ will drop dramatically since this value represents the estimated potential scale reduction, where the potential is a terminology in Markov random field representing the probabilistic distribution over a small region of pixels.

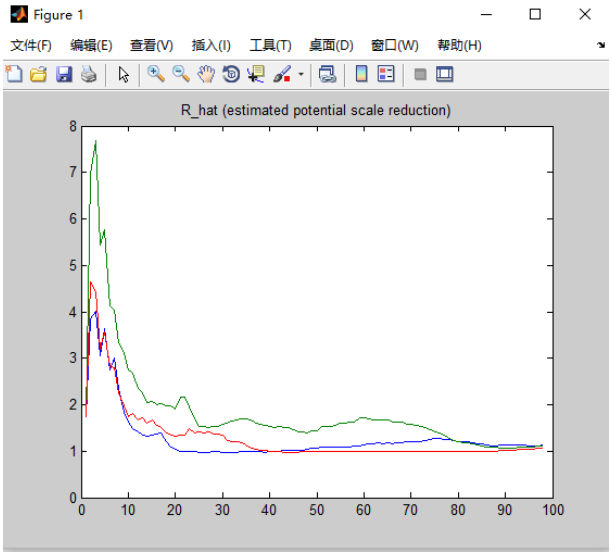


Figure 4.6: Near the end of convergence

As we can see from the above figure, both the RGB channels are near the end of convergence (when we say end of convergence, we mean that the scale reduction of RGB both become 0).

4.2.2 Result demonstration

After the final convergence, a window of both the original figure and the inpainted figure will show on the screen. The following figure shows the result of inpainted image, which is a inpainted image of one of our teammates:

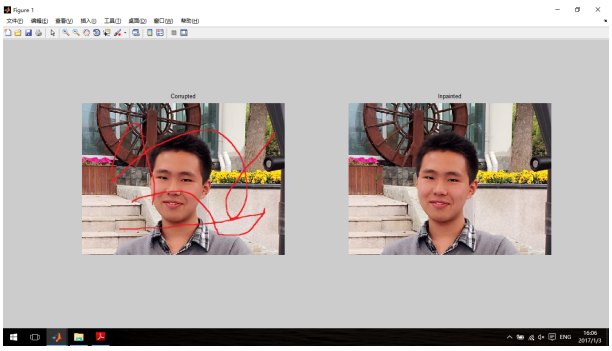


Figure 4.7: Result of Markov random field model

As we can see from the above figure, we have a damaged image on the left, after the model has converged, we have the fixed image on the right and the

fixed image restores the fixed image quite well.

4.3 Other Use Cases

Apart from fix image and object removal, there are many other use cases of image inpainting. Here we will give another use case to inspire the reader to come up with more interesting use cases. This example use case remove the unwanted text from a given image. The input image is as follows:

Figure 4.8: Image with unwanted text on it

We can apply either the exemplar-based algorithm or Markov random field on it. Here we apply the Markov random field algorithm on it since it work best on such small region restoration. The following figure shows the fixed version:

Figure 4.9: Image with unwanted text removed

Although making musk for this image might be a little tedious, it would be much more tedious if one uses software like PS to fix such an image.

Chapter 5

Test and Analysis

Abstract

This document is to record the test and the analysis of our image inpainting software. In this document, there are test analysis report and describe the test case and suggestions. We design the test case and the test situation for testing our image inpainting software. We put our software on the open source community on the Internet and receive some feedback from different users.

Keywords Image Inpainting Software, Tese, Test Case, Analysis

5.1 Introduction

5.1.1 Purpose

This document is to record the test and the analysis of our image inpainting software. It is our test analysis report for our software, which illustrates the details of test result according to the test context, test scope ,test standard design in the test plan document. We design the test case and the test situation for testing our image inpainting software.

Further more, we put our software on the open source community on the Internet and receive some feedback from different users.

5.1.2 Scope

This document will be the main reference for our testing. Therefore, the readers for this document are mainly the testers and the project manager of our image inpainting software. Test of our software is supposed to accord to this document.

5.1.3 Background

The image inpainting software is used for inpainting images. Users can choose their wanted area in the input image and our image inpainting software will inpaint the target area.There are different applicable scenario for out software including image text removal, object removal and image detail restoration etc.

The whole project began at September. After coding out the system and our testers master the testing knowledge and skills, we can do our test.

5.2 Test Design

Due to that we implement two different to solve the image inpainting problem, there are two test case that we design for our image inpainting software.

As for Markov Random Field Algorithm, it is used to inpaint image text and restore image trace. And as for exemplar-based algorithm, it is used to do big object removal. So we prepare different images to test the software performance.

We designed the test situation for different applicable scenario and different use-case. The test image is different from the training image. We will present some of the test image in this document to illustrate the result.

5.3 Test Result

5.3.1 Use-Case

Overview

We will present part of our test result in the following.

Use-Case	Action	Expected Result	Actual Result
Input Image1	Input a mask	Program goes well	Fit expectation
Input Image2	Input a mask	An error message happened	Fit expectation
Input Mask1	Input an image	Program goes well	Fit expectation
Input Mask2	Input an image	An error message happened	Fit expectation
Output Image1	Output an image	Program goes well	Fit expectation
Output Image2	Output an image	An error message happened	Fit expectation
User Interface	User interaction	Program works well	Fit expectation

Table 5.1: Use-Case Test Result Overview

Input Image

We use some correct image input and wrong image input to test our software. The software replay fits our expectation.

Input Mask

We use some correct mask input and wrong mask input to test our software. The software replay fits our expectation.

Output Image

We use some correct image output and wrong image output to test our software. The software replay fits our expectation.

User Interface

During our test process, the user interface works well.

5.3.2 Applicable Scenario

Overview

We tested our software in different applicable scenario, and we will put part of our test result in the following.

Applicable Scenario	Expected Result	Actual Result
Image Text Removal	Program goes well	Fit expectation
Image Object Removal	Program goes well	Fit expectation
Image Detail Restoration	Program goes well	Fit expectation

Table 5.2: Applicable Scenario Test Result Overview

Image Text Removal

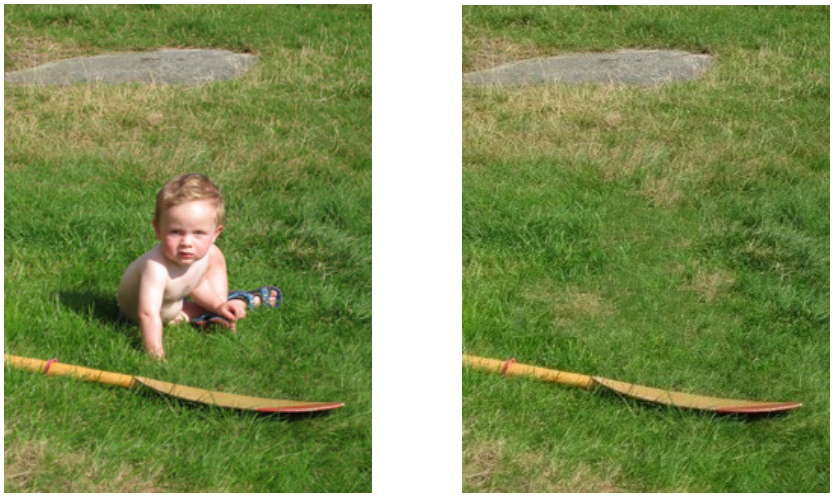
We used lots of image to test the image text removal function and it works well. The following is one of the result image:

(a) image before in-paintinging (b) image after in-paintinging

Figure 5.1: Image Text Removal

Image Object Removal

We used lots of image to test the image object removal function and it works well. The following is one of the result image:

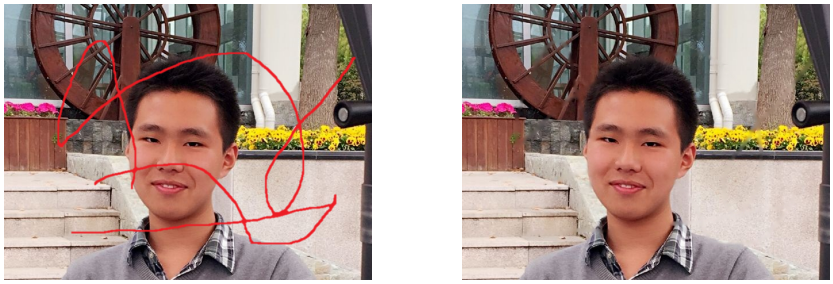


(a) image before in-paintinging (b) image after in-paintinging

Figure 5.2: Image Object Removal

Image Detail Restoration

We used lots of image to test the image detail restoration function and it works well. The following is one of the result image:



(a) image before in-paintinging (b) image after in-paintinging

Figure 5.3: Image Object Removal

5.4 Test Cost

The whole test is using our PC and get picture free from the Internet, so it did not cost any extra money. The test feedback is from our volunteer user and ourselves, it did not cost any extra money too.

5.5 Analysis

5.5.1 Performance Evaluation

The test result illustrates that the software works well. The image inpainting function (users can choose their wanted area in the input image and our image inpainting software will inpaint the target area.There are different applicable scenario for out software including image text removal, object removal and image detail restoration etc.) works well.

5.5.2 Distinguishing Feature

As for the two algorithm that we implemented, it presents different expert field. When do text removal and detail restoration, Markov Random Field works better and Exemplar-Based Algorithm works well when do big object removal. So we design the inner choice program to choose different method for users' input image.

To sum up, the software works well when do image inpainting(image text removal, image object removal, image detail restoration).

5.5.3 Limitations

When the input mask is large, it may cost a little long time to process the image and get the result.

5.5.4 Future Work

We can get quicker performance and more image process function. Further, in this AI age, we can apply deep learning model in this topic when there is enough data.

5.6 Users Feedback

We have put our image inpainting ware on the open source community on the Internet. We also invite some students to use our software. Therefore, we get some feedback from our volunteer users. We will present some feedback in the following.

Mike: *"That's a wonderful software! Ten days ago, I went on a holiday with*

my girlfriend at the seaside and took a lot of photos of landscape. But when I check the photos yesterday I find that there were always some people in our photos. That destroyed my mood. I feel worry. At that time, a co-worker of mine recommended the "Image Inpainting" software to me. I tried it with a photo at once. After some time, the miracle happened! There is only me and my girlfriend in the photo. It also saves the beautiful landscape. That's very convenient for me. I wont worry about someone in my favorite photos. "

Amy: *"This software is very convenient! I am a college student. I always search pictures on the internet. But nowadays a problem trouble me many times. When I download pictures from the internet. There are always some watermarks on the picture. It effects the beauty of the hole slides. One day, I find this Image Inpainting software on the internet. I am surprised that it can remove the watermarks automatically. It helps me a lot at study and life."*

Kimi: *"Favorable comment! Some days ago, I found some old photographs in the house. Those photos were taken fifty years ago. Those photos remind me of lots of things when I was young. I want to make these photos digital to save it forever. When I was scanning it into computer, I found that some cracks also scanning into the computer. I was not so good. I found this software in the APP Store. It can help me remove these cracks and make photos complete. Thanks to the software!"*

Timmy: *"How amazing it is! I like this software so much. I am a photographer. I always take some photos outside. But theres always something that I dont want in the lens. I need to take many time to make everything OK outside. I waste so many time on it. But a friend told me the Image Inpainting was useful for our photographers. It can remove what I dont want. How amazing! I can take photos very easy."*

Chapter 6

Conclusion

1. We accomplished the chosen task and solved the problem. The result works well.
2. The proposed ideas given by us successfully solve the Image Inpainting problem.
3. As for our Use-Case and three different applicable scenarios, our software works well and it successfully solve some image inpainting problem in our daily life.
4. Our algorithm is capable of different size of input mask and can apply to different kinds of scenarios including image text removal, image object removal and image detail restoration. Our method can successfully solve these problem efficiently.

Bibliography

[1] M. Bertalmío, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In J. R. Brown and K. Akeley, editors, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, pages 417–424. ACM, 2000.

[2] A. Criminisi, P. Pérez, and K. Toyama. Object removal by exemplar-based inpainting. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA*, pages 721–728. IEEE Computer Society, 2003.

[3] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Trans. Image Processing*, 13(9):1200–1212, 2004.

[4] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[5] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 860–867. IEEE Computer Society, 2005.