ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (36-47), Month: October - December 2015, Available at: www.researchpublish.com

Identification and Removal of Cracks in Digitized Paintings

[1]C. SURESH, [2]V. VIDHYA, [3]E. SHAMLI, [4]P. MAHALAKSHMI, [5] G. SUMITHRA

[1]B.TECH Information Technology (Final year) & Mailam Engineering College, India

[2],3,5 B.E Computer Science Engineering (Final year)& Mailam Engineering College, India

[4]B.E Civil Engineering (Final year)& Mailam Engineering College, India

Abstract: **In proposed System, the detection and removal of cracks in digitized painting is presented. Our system will restore the old cracked painting or images. Detect crack using morphological high pass operator called top-hat transformation. Then filling procedure will be applied on cracks. Order statistics filter are used for crack filling which deals with mean and median filter. In previous system, this things were done manually. Our proposed system going to overcome all our problems of existing because it is automated tool to detect and remove the cracks from the cracked images.**

Keywords: **Image processing, Digital image Processing, Top hat Transformation, bottom hat transformation.**

1. INTRODUCTION

Digital image processing techniques can be used in the restoration of digitized paintings which contains cracks. Old Paintings suffer from breaks in the paint, or the varnish. These patterns are usually called cracks and can be caused by aging, drying, and other mechanical factors. This cracks can be rectangular, circular, spider-web, unidirectional, tree branches and random. Cracks also depend on the materials used for the painting, the painting technique of the artist, the atmospheric variations and storage conditions. The appearance of cracks on paintings deteriorates the legibility of image.

A virtual restoration can provide clues to art historians, museum curators and the general public on how the painting would look like in its initial state.

1.1 Motivation:

Paintings which are made by paint or other material suffer from break due to some factor; it makes some pattern like structure is called cracks. Because of this cracks paintings are damaged. Original effect of painting is lost. Craquelure is the fine pattern of dense "cracking" formed on the surface of materials, either as part of the process of ageing or of their original formation or production. The term is most often used to refer to tempera or oil paintings, where it is a sign of age that is also sometimes induced in forgeries and ceramics. It can also develop in old ivory carvings, and painted miniatures on an ivory backing are prone to craquelure.

So in our project we are trying to restore paintings virtually so that people can get idea of that painting i.e. how the paintings usually look like.Our proposed system is automated to detect and remove the cracks from the cracked images.It can be used in museums in case of expensive paintings.

1.2 Problem Statement:

Among many proposed system crack were detected manually where user needs to specify the initial crack location for filling of cracks. The Crack filling algorithm gives the blurring effect on image.

Proposed system will detect crack automatically. It will find low luminance intensity point and apply top hat transformation. And then fill these crack pixels using order statistics filter.

1.3 Scope:

Old cracked painting will be scanned and converted to digital image by converting it to jpeg or jpg format.

Proposed system will detect only low luminance pixel by Top Hat Transform and it will not detect high intensity pixel values.

The Top Hat transformation generates a grayscale output image with no background.

Threshold is applied to generate crack map. Crack Filling is done by order statistic Filters.

2. LITERATURE SURVEY

1. Digital image processing techniques for the detection and removal of cracks in digitized paintings.[1]

In this paper, An integrated methodology for the detection and removal of cracks on digitized paintings is presented. The cracks are detected by thresholding the output of the morphological top-hat transform. Afterwards, the thin dark brush strokes which have been misidentified as cracks are removed using either a Median Radial Basis Function (MRBF) neural network on hue and saturation data or a semi-automatic procedure based on region growing.

Disadvantage- For crack filling anisotropic diffusion method is used which introduces blurring effect.

2. Image Processing Methods for the Restoration of Digitized Paintings.[2]

In this paper, a morphological methodology (MAO) is proposed which is a variant of recently published morphological methods to identify cracks. After detecting cracks, a modified adaptive median filter (MAMF) is used to fill the cracks The order of the median filter to be applied on crack pixels is computed on the basis of the number of crack pixels in its neighborhood .Main Disadvantage- For detection bottom-hat transform(BHT) is used which is less efficient.

3. Image In painting.[3]

In this paper, we introduce a novel algorithm for digital inpainting of still images that attempts to replicate the basic techniques used by professional restorators. 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 .No limitations are imposed on the topology of the region to be inpainted. One of the main problems is the reproduction of large textured regions.

Disadvantage-The user needs to provide the region to be in painted.

Existing system:

One of the main problems is the reproduction of large textured regions.

The user needs to provide the region to be in painted.

Bottom-hat transform (BHT) is used which is less effective. If multiple image channels are present, Bottom hat transform operates on each of them separately.

For crack filling -Controlled Anisotropic Diffusion is been used which give blur effect to the image.(At each position and iteration, diffusion is controlled by the conduction (or diffusion) coefficients c(x, y, t). Since diffusion should be inhibited across regions separated by discontinuities, the conduction coefficients should obtain small values in pixels with large intensity gradient magnitude.)

Among many proposed system crack were detected manually.

3. SCOPE

Old cracked painting will be scanned and converted to digital image by converting it to jpeg or jpg format.

Proposed system will detect only low luminance pixel by Top Hat Transform and it will not detect high intensity pixel values.

The Top Hat transformation generates a grayscale output image with no background.

Threshold is applied to generate crack map. Crack Filling is done by order statistic Filters.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (36-47), Month: October - December 2015, Available at: www.researchpublish.com

4. REQUIREMENT GATHERING & ANALYSIS

4.1 Requirement Elicitation:

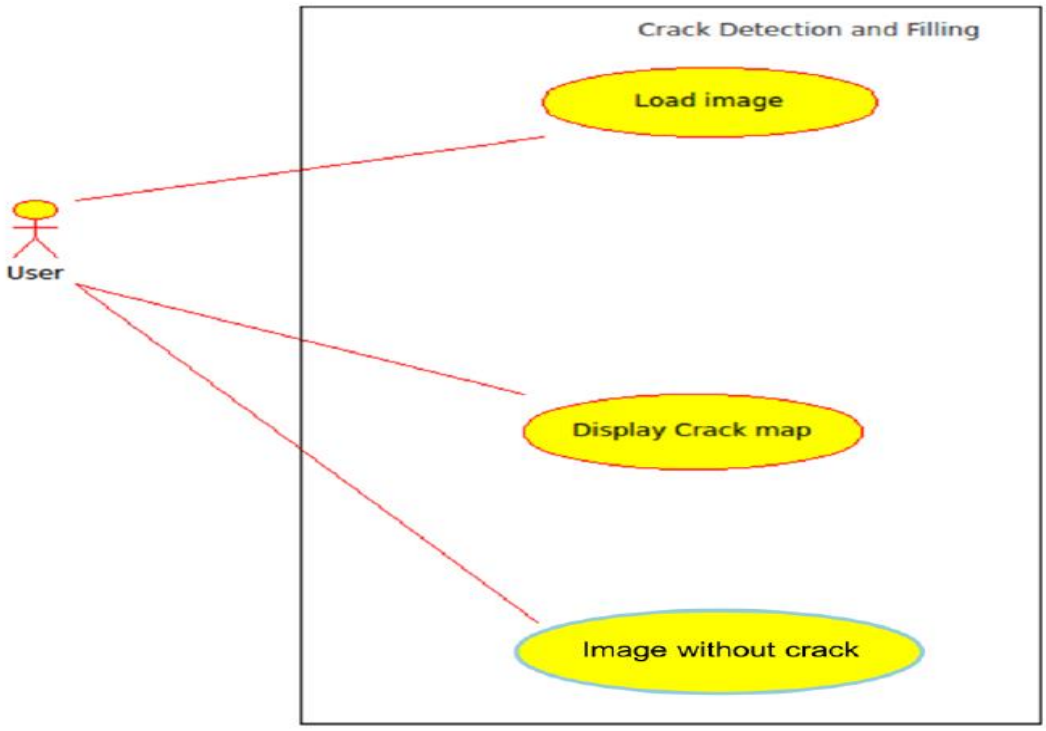

Fig.4.1 Use-case diagram for crack detection and filling

Template:

Sr. No.	ACTOR	DESCRIPTION
1	User	1.User loads the image 2.Observes crack map

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

Purpose:

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and Statechart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modelled to present the outside view.

So in brief, the purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.

- Show the interacting among the requirements are actors.

4.2 Feasibility Study :

The very first phase in any system developing life cycle is preliminary investigation. The feasibility study is a major part of this phase. A measure of how beneficial or practical the development of any information system would be to the organization is the feasibility study.

4.2.1 Technical feasibility:

- At least 166 MHz Pentium Processor or Intel compatible processor.
- At least 16 MB RAM
- A video graphics card
- MATLAB R2014a
- A mouse or other pointing device.
- At least 3 MB free hard disk space.

4.2.2 Economic feasibility:

Once the hardware and software requirements get fulfilled, there is no need for the user of our system to spend for any additional overhead.

For the user, the application will be economically feasible in the following aspects:

- The application will reduce a manual work. Hence the work will be reduced.
- Our application will reduce the time that is wasted in manual processes.
- The storage and handling problems of the image will be solved.

5. DESIGN

5.1 UML Diagram:

The Unified Modeling Language (UML) is a standard visual modeling language intended to be used for

- modeling business and similar processes,
- analysis, design, and implementation of software-based systems

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.

UML can be applied to diverse application domains (e.g., banking, finance, internet, aerospace, healthcare, etc.) It can be used with all major object and component software development methods and for various implementation platforms (e.g., J2EE, .NET).

UML is a standard modeling language, not a software development process. UML 1.4.2 Specification explained that process:

- provides guidance as to the order of a team's activities,
- specifies what artifacts should be developed,
- directs the tasks of individual developers and the team as a whole, and
- Offers criteria for monitoring and measuring a project's products and activities.

UML is intentionally process independent and could be applied in the context of different processes. Still, it is most suitable for use case driven, iterative and incremental development processes. An example of such process is Rational Unified Process (RUP).

UML is not complete and it is not completely visual. Given some UML diagram, we can't be sure to understand depicted part or behavior of the system from the diagram alone. Some information could be intentionally omitted from the diagram, some information represented on the diagram could have different interpretations, and some concepts of UML have no graphical notation at all, so there is no way to depict those on diagrams.

For example, semantics of multiplicity of actors and multiplicity of use cases on use case diagrams is not defined precisely in the UML specification and could mean either concurrent or successive usage of use cases. Name of an abstract classifier is shown in italics while final classifier has no specific graphical notation, so there is no way to determine whether classifier is final or not from the diagram.

5.1.1 Activity Diagram:

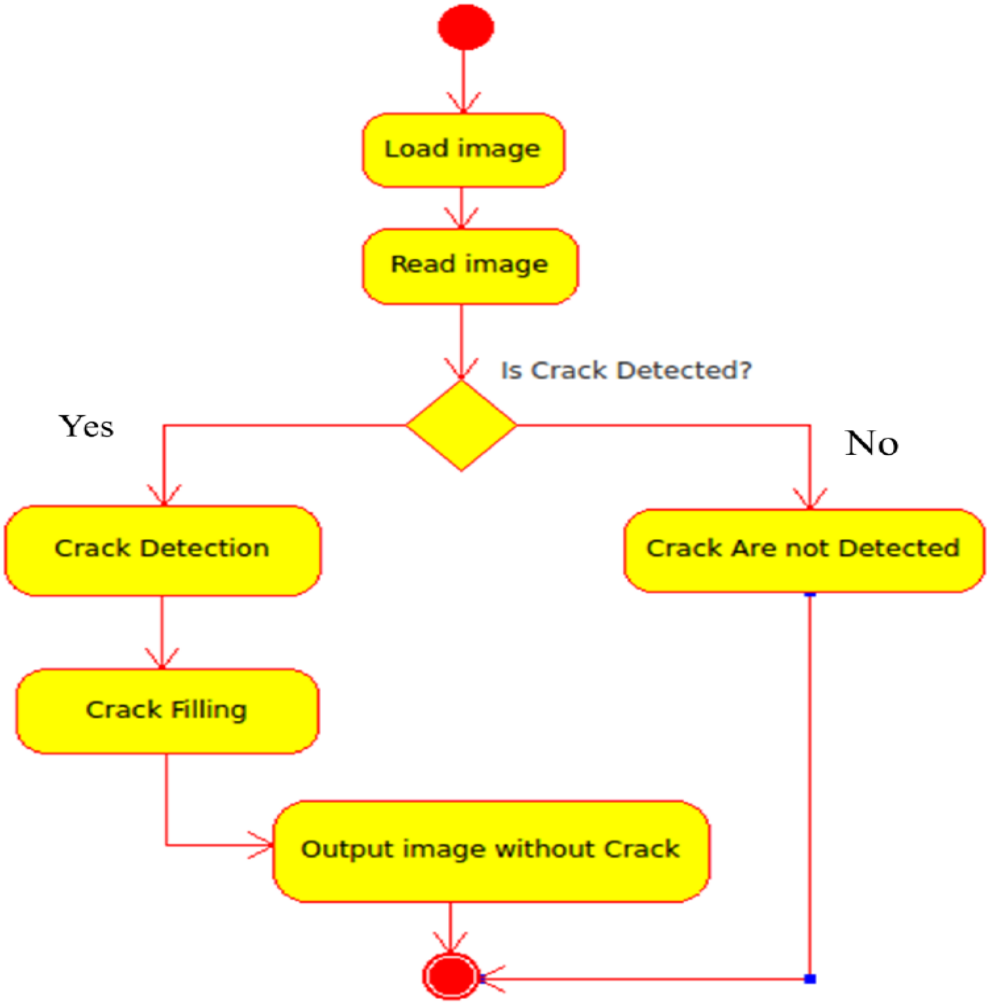

Fig. 5.1.1 Activity Diagram of Crack detection and Filling

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows) Activity diagrams show the overall flow of control.

Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

- rounded rectangles represent actions;
- diamonds represent decisions;
- bars represent the start (split) or end (join) of concurrent activities;

- a black circle represents the start (initial state) of the workflow;
- An encircled black circle represents the end (final state).

Arrows run from the start towards the end and represent the order in which activities happen.

Activity diagrams may be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrent However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

While in UML 1.x, activity diagrams were a specialized form of state diagrams, in UML 2.x, the activity diagrams were reformalized to be based on Petri net-like semantics, increasing the scope of situations that can be modeled using activity diagrams These changes cause many UML 1.x activity diagrams to be interpreted differently in UML 2.x.

UML activity diagrams in version 2.x can be used in various domains, i.e. in design of embedded systems. It is possible to verify such a specification using model checking technique.

5.1.2 State Diagram:

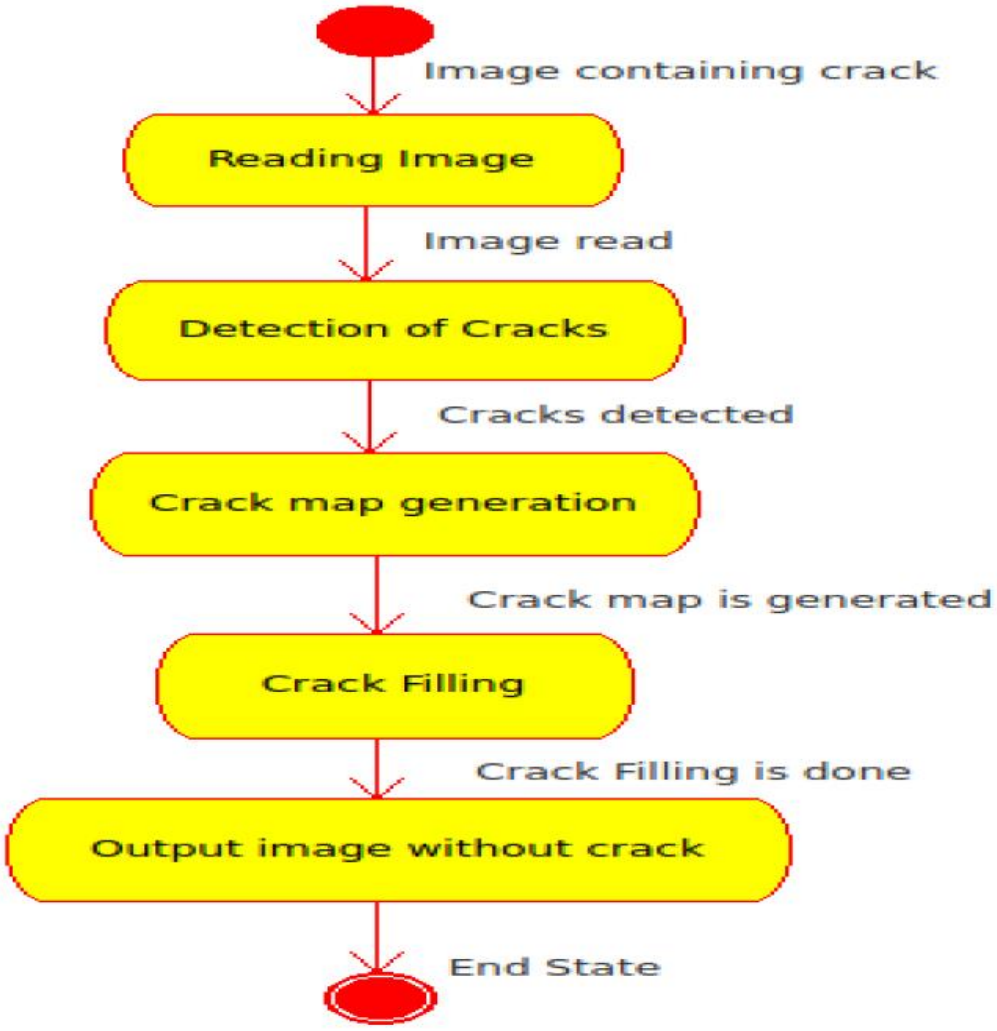

Fig. 5.1.2 State Diagram of Crack detection and Filling

The state diagram in the Unified Modeling Language is essentially a Harel statechart with standardized notation which can describe many systems, from computer programs to business processes. In UML 2 the name has been changed to *State Machine Diagram*. The following are the basic notational elements that can be used to make up a diagram:

- Filled circle, pointing to the initial state

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (36-47), Month: October - December 2015, Available at: www.researchpublish.com

- Hollow circle containing a smaller filled circle, indicating the final state (if any)
- Rounded rectangle, denoting a state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activities that are done in that state are indicated
- Arrow, denoting transition. The name of the event (if any) causing this transition labels the arrow body. A guard expression may be added before a "/" and enclosed in square-brackets (*eventName*[guardExpression]), denoting that this expression must be true for the transition to take place. If an action is performed during this transition, it is added to the label following a "/" (*eventName*[guardExpression]/action).
- Thick horizontal line with either x>1 lines entering and 1 line leaving or 1 line entering and x>1 lines leaving. These denote join/fork, respectively.

5.1.3 Sequence Diagram:

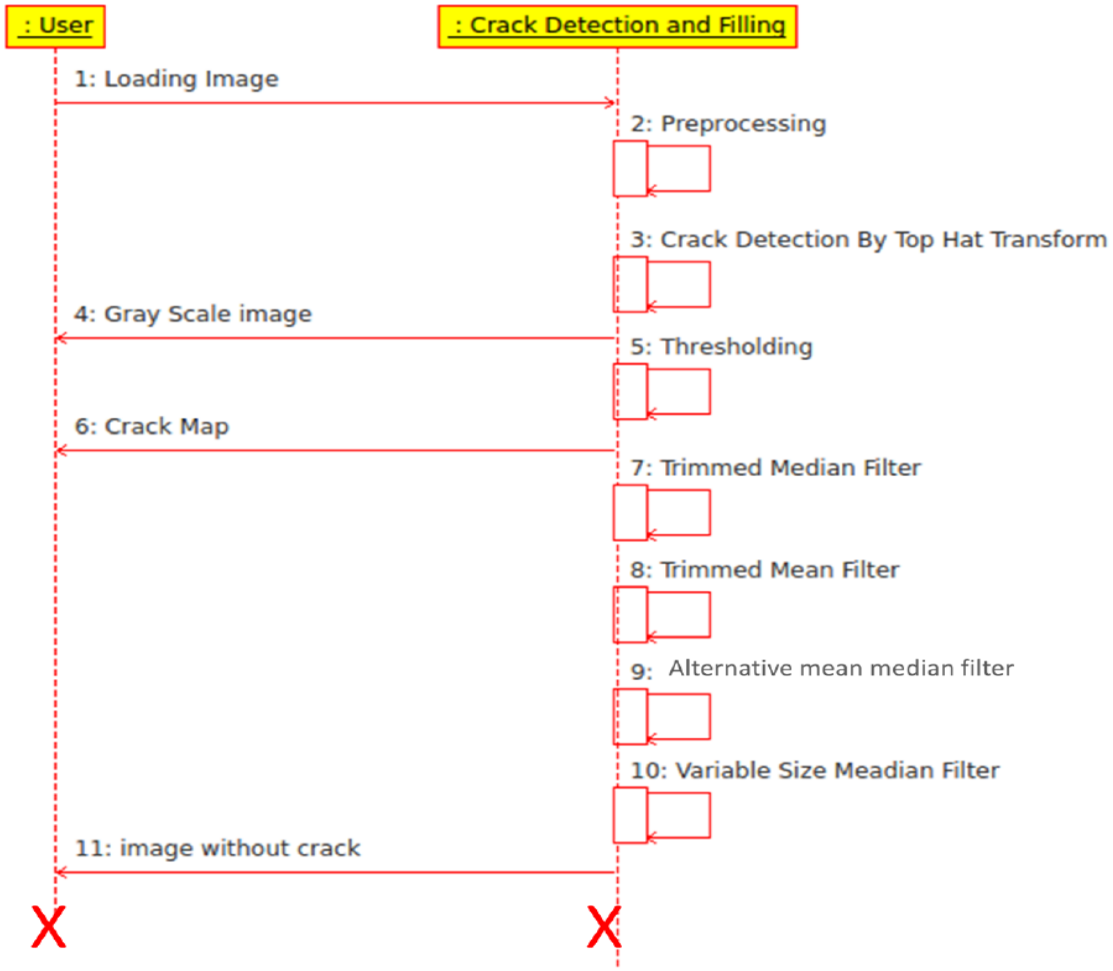

Fig. 5.1.3 Sequence Diagram of Crack detection and Filling

A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (36-47), Month: October - December 2015, Available at: www.researchpublish.com

5.1.4 Collaboration Diagram:

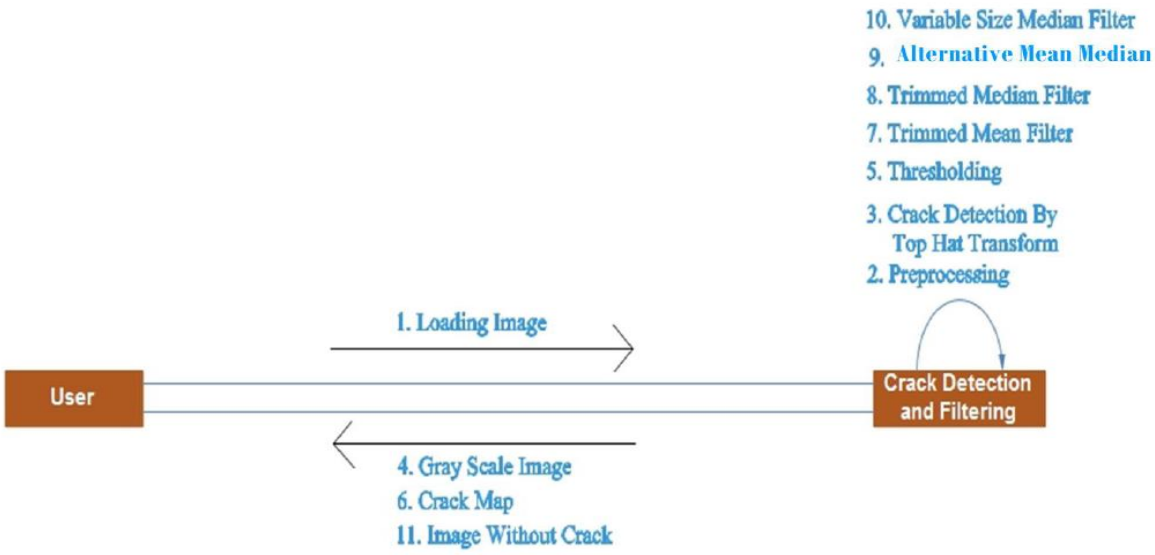

Fig. 5.1.4 Collaboration Diagram of Crack detection and Filling

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

Class roles: Class roles describe how objects behave. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Association roles: Association roles describe how an association will behave given a particular situation. You can draw association roles using simple lines labeled with stereotypes.

Messages

Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time and instead number messages in order of execution. Sequence numbering can become nested using the Dewey decimal system. For example, nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on. The a condition for a message is usually placed in square brackets immediately following the sequence number. Use a * after the sequence number to indicate a loop.

6. PROPOSED SYSTEM

6.1 Architecture Diagram:

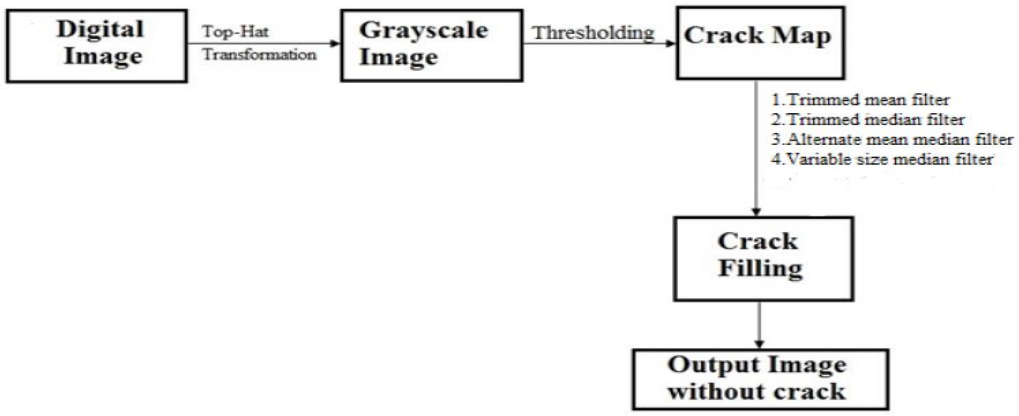

Fig. 6.1 Block Diagram of Crack detection and Filling

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (36-47), Month: October - December 2015, Available at: www.researchpublish.com

7. METHODOLOGY

Our project will be built on the waterfall mode. This model suggests work cascading from step to step like a series of waterfalls. It consists of the following steps in the following manner

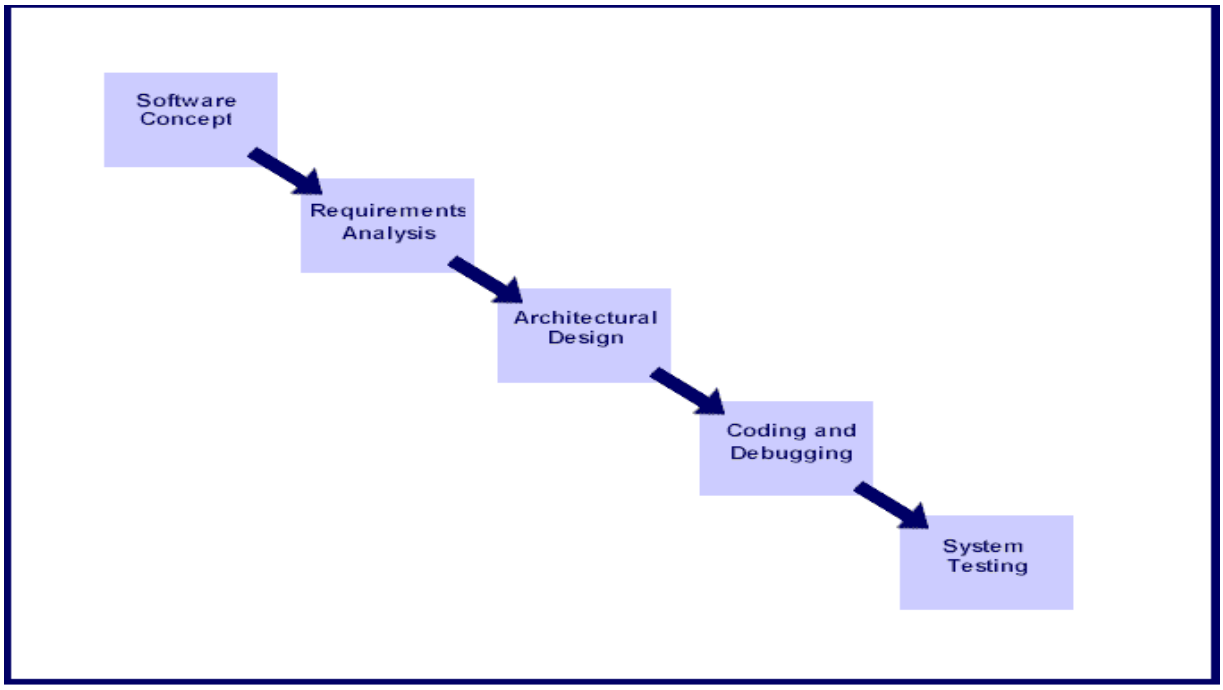

Fig.7.1 methodology

7.1 Steps involved in the System Development Life Cycle:

Below are the steps involved in the System Development Life Cycle. Each phase within the overall cycle may be made up of several steps.

Step 1: Software Concept

The first step is to identify a need for the new system. This will include determining whether a business problem or opportunity exists, conducting a feasibility study to determine if the proposed solution is cost effective, and developing a project plan.

This process may involve end users who come up with an idea for improving their work. Ideally, the process occurs in tandem with a review of the organization's strategic plan to ensure that IT is being used to help the organization achieve its strategic objectives. Management may need to approve concept ideas before any money is budgeted for its development.

Step 2: Requirements Analysis

Requirements analysis is the process of analyzing the information needs of the end users, the organizational environment, and any system presently being used, developing the functional requirements of a system that can meet the needs of the users. The requirements documentation should be referred to throughout the rest of the system development process to ensure the developing project aligns with user needs and requirements.

Professionals must involve end users in this process to ensure that the new system will function adequately and meets their needs and expectations.

Step 3: Architectural Design

After the requirements have been determined, the necessary specifications for the hardware, software, people, and data resources, and the information products that will satisfy the functional requirements of the proposed system can be determined. The design will serve as a blueprint for the system and helps detect problems before these errors or problems

are built into the final system. Professionals create the system design, but must review their work with the users to ensure the design meets users' needs.

Step 4: Coding and Debugging

Coding and debugging is the act of creating the final system. This step is done by software developer.

Step 5: System Testing

The system must be tested to evaluate its actual functionality in relation to expected or intended functionality. Some other issues to consider during this stage would be converting old data into the new system and training employees to use the new system. End users will be key in determining whether the developed system meets the intended requirements, and the extent to which the system is actually used.

Step 6: Maintenance

Inevitably the system will need maintenance. Software will definitely undergo change once it is delivered to the customer. There are many reasons for the change. Change could happen because of some unexpected input values into the system. In addition, the changes in the system could directly affect the software operations. The software should be developed to accommodate changes that could happen during the post implementation period.

7.2 Technology used:

MATLAB R2014a:

The MATLAB language provides native support for the vector and matrix operations that are fundamental to solving engineering and scientific problems, enabling fast development and execution.

With the MATLAB language, you can write programs and develop algorithms faster than with traditional languages because you do not need to perform low-level administrative tasks such as declaring variables, specifying data types, and allocating memory. In many cases, the support for vector and matrix operations eliminates the need for for-loops. As a result, one line of MATLAB code can often replace several lines of C or C++ code.

MATLAB provides features of traditional programming languages, including flow control, error handling, and object-oriented programming (OOP). You can use fundamental data types or advanced data structures, or you can define custom data types.

8. RESULT

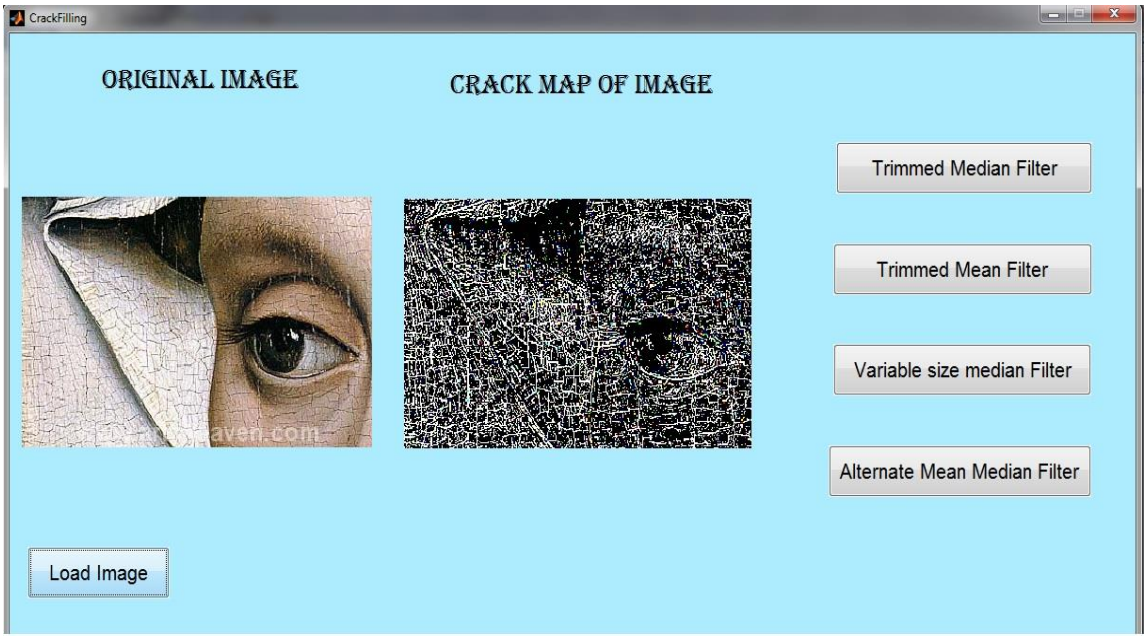

Fig 8.1: GUI with crack map

Fig 8.2: Result of Trimmed median filter

Fig 8.3: Result of Trimmed mean filter

Fig 8.4: Result of Variable size median filter

Fig 8.5: Result of Alternative mean median filter

9. CONCLUSION AND FURTHER WORK

9.1 Conclusion:

The results presented in this work regard the craquelure of old paintings, however, the same algorithms can be used for a much wider set of applications..

9.2 Further Work:

Image processing techniques used to detect and remove crack can be employed to inspection and/or diagnosis in many scientific field. There are certain aspects of the proposed methodology that can be further improved. For example; the crack detection algorithm can be applied locally which will select a low threshold value. Edge detection and segmentation can also be performed which will confine the filling of cracks that cross edges or region borders to pixels from the corresponding region. Use of image inpainting techniques given in Bertalmio et al. (2000) could also improve results in that aspect. These improvements will be the topic of future work on this subject

REFERENCES

[1] Giakoumis I., Nikolaidis N., Pitas I, 2006, "Digital image processing techniques for the detection and removal of cracks in digitized paintings", '*IEEE Transactions on Image Processing'*. 15(1), 178-88

[2] Gupta A., Khandelwal V., Gupta A., 2008,"Image Processing Methods for the Restoration of Digitized Paintings", '*Thammasat Int. J. Sc. Tech'*. 13(3), 66-72

[3] Bertalmio, M, Sapiro, G, Caselles, V, Ballester C, 2000, "Image inpainting", 'proceedings, *SIGGRAPH*'.417-424.

[4] Barni M., Bartolini F., Cappellini V., 2000, "Image processing for virtual restoration of artworks", '*IEEE Multimedia*'.7(2), 34-37.