Masters Project Final Report

December 2012

Project Title	**Removing Camera Shake Effect from a Photograph**			
Student Name	J. P. Gayan Saminda			
Registration No. & Index No.	2008/MCS/016 *08440162*			
Supervisor's Name	Dr. Anuja Dharmaratne			
Please Circle the appropriate	**Masters Program**		**Type**	
	MIT	**MCS**	**Research**	*Implementation*

For Office Use ONLY

Removing Camera Shake Effect from a Photograph

J. P. Gayan Saminda

2013

Removing the Camera Shake Effect from a Photograph

A dissertation submitted for the Degree of Master of Computer Science

J. P. Gayan Saminda

University of Colombo School of Computing

2013

Declaration

This thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Students Name: J. P. Gayan Saminda

_____

Signature: Date:

This is to certify that this thesis is based on the work of

Mr J. P. Gayan Saminda

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Dr A. T. Dharmaratne

_____

Signature: Date:

Abstract

This project develops a method to remove the blur effect due to camera shake in a photograph. All the tasks that carried out during the project can be characterized into two areas. First is the development of a software framework for the algorithm development and testing. It includes tools to visualize pixel patterns better and support Implement, experiment, and evaluate various algorithms.

The second Item is the development of the main methodology, which is the invention of this project, to remove the camera shake effect from a photograph. The methodology was an evolutionary development. It involved a great deal of experiments, analysis of results and feedback. The mechanism includes Pre Processing, Blurred Edge Detection, Edge Correction and Post Processing. All these functions have got evolved throughout the project, in order to produce better results.

In order to support all the experiments done for the main algorithm development, a rich set of tools and components were needed in the supporting framework. Standard graphs such as color histogram, gradient histogram, together with some custom graphs, are implemented. Standard image processing algorithms like Blur kernels, Median Filtering and Frequency Domain Analysis was implemented. Lot more other tools were developed to visualize the pixel patterns better.

Table of Contents

List of figures

Chapter 1

Introduction

1.1 Background

An image that is taken by a camera can be blurred due to many reasons. The common reasons are, out of focus, shaking the camera, object motion. Etc. Obviously all these issues can be completely eliminated by handling the camera carefully. But somehow there are various situations that it is completely unavoidable. There are situations where we don't get enough time to handle the camera. We may miss the scenery if we didn't take the picture in a hurry. Or else the object is moving such as we are taking a photograph of a moving vehicle. Or else we are (or camera is) on top of a shaking surface.

Quite a lot of techniques are used in modern digital cameras in order to eliminate the shake effect. Powerful processors inside the cameras are running intelligent software which can detect the shake and adjust the shutter speed and other parameters in order to minimize the blur due to shake. Other common technique they use is, when user press the shot button in a shaking environment, it waits a while and takes the shot at the best moment that is with the minimum camera shake. Some softwares are intelligent enough to digitally shift the captured values and correct the image while capturing. Hardware based stabilizers also there, inside the modern day's cameras. They try to physically stabilize the lens and the sensor up to some extent.

Along with all these techniques, modern cameras are capable of removing great amount of camera shake disturbance. But somehow all these technologies have their own limitations. Even latest cameras with all the technologies, generates images with destroyed quality, depending on the situation. On the other hand, the potable cameras that we use in our day to day life, does not have all these sophisticated technology to prevent the shake effect. The phone cameras definitely. Therefore we find lot of photographs including very valuable ones in our day to day life, destroyed due to camera shake.

Software based correction for such image after image was taken, is a research area that never ends up with some straight forward mechanism which is simple and works for any image. Various solutions are there in the industry, developed by various people. The common property of all the methods is, they are only good for certain categories of images or certain blur patterns. Therefore there is lot more areas to research on this field.

1.2 Focus of the project

The main focus of this project is to develop a methodology to remove the camera shake effect from a photograph. This was done as an evolutionary development with continuously trying various techniques and checking the results and fine tuning the methodologies.

The main problem was divided into modules or sub problems. Each module can be individually evolved to make its output better. Always the basic workflow was, Imagine a technique to improve, Implement, Test and decide whether to go with the technique or not. If decided to go with the technique then, find various ways to improve and fine-tune it. Otherwise throw it away.

For this purpose, there was a requirement for a software framework that can be used as a Test Bed for quickly develop and test any image processing algorithm. So the project was started with developing such tool with basic image viewing features. Throughout the project, that tool got evolved by adding more and more features and finally became a test bed for any image processing algorithm development and testing.

The main image correction methodology, which is the final outcome of the project, was developed on top of the Test Bed Framework. Therefore the output of the project is a single software program that has the Image Shake Correction functionality and all the testing functionalities.

1.3 The Image Shake Correction Mechanism

The main mechanism developed to correct blurred images, assumes that the edges are the areas that the shake effect is clearly visible. If we correct all the blurred edges, the blurred look and feel of the image will go off and image will look nice.

Basically the algorithm navigates throughout the image line wise and corrects the blurred edges in individual lines. A novel approach was developed for blurred edge detection. Other functionalities like image Preprocessing, Post Processing, and Edge Location Purification are also involved in the process. The entire process will be described in the relevant chapters.

Chapter 2

Literature Review

2.1 Introduction

This chapter contains the information about research papers on the topic of Image Shake Correction, their approaches and some evaluations on them.

The evaluations were made based on various experiments that had been carried out about the assumptions and methods that the researcher(s) have done.

2.2 Literature Review 1

The research paper, Removing Camera Shake from a Single Photograph (Rob Fergus et al. [1]) describes a method of image shake correction based on set of assumptions and a parametric model. Some testing and evaluation that had been carried out about this research paper is described in Appendix.

2.2.1 Assumptions:-

- Any Camera shake can be modeled as a blur kernel. Applying reverse blur kernel will restore the image.
- Any natural image will have their Gradient Vs. Log(2)(number of pixels) histogram in unique shape.
- Gradient Vs. Log(2)(number of pixels) histogram of a blurry image is narrower than a good image(figure 1.2). Therefore making the histogram wider and adhering to the parametric model that they found, is making the image better.

2.2.2 Approach:-

This algorithm selects a small portion of the image and calculates the blur kernel that makes the portion of image more adhering to natural image properties, and applies that kernel to the entire image.

They assume that any natural image will have its gradient vs. Log(2)(number of pixels) histogram in unique shape. And they have mapped that shape into parametric model (figure1.1). They start with some blur kernel and recursively improve it by applying it to the sample portion and making sure the gradient histogram of the resulting image is wider and more adhering to the predefined parametric model.

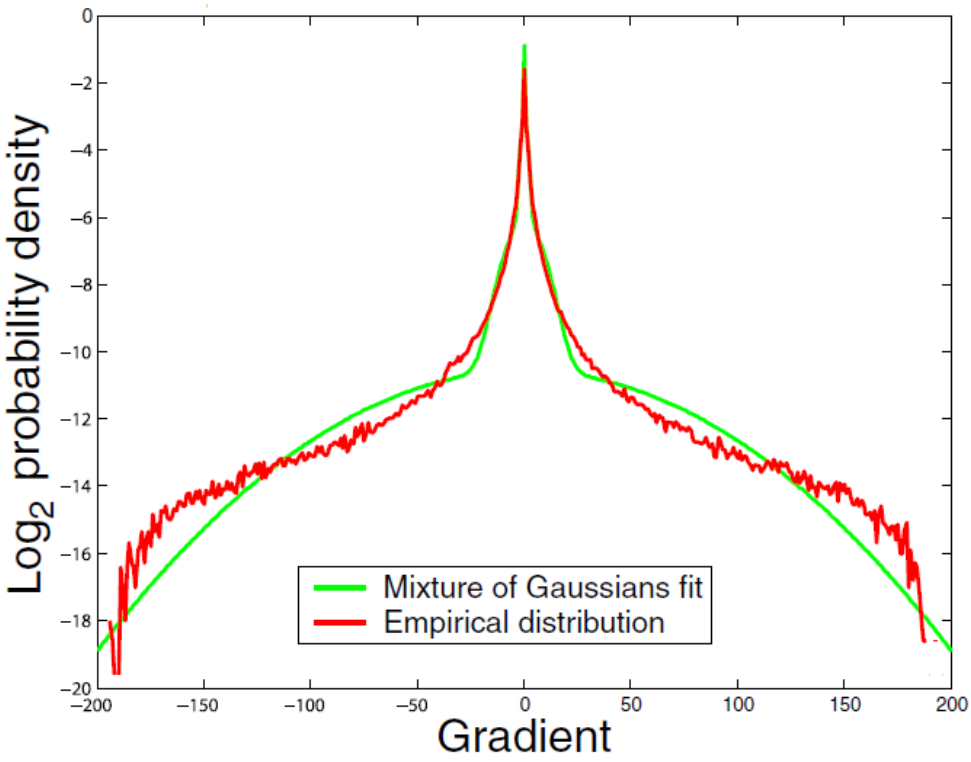


Figure 1.1 – Gradient Graph

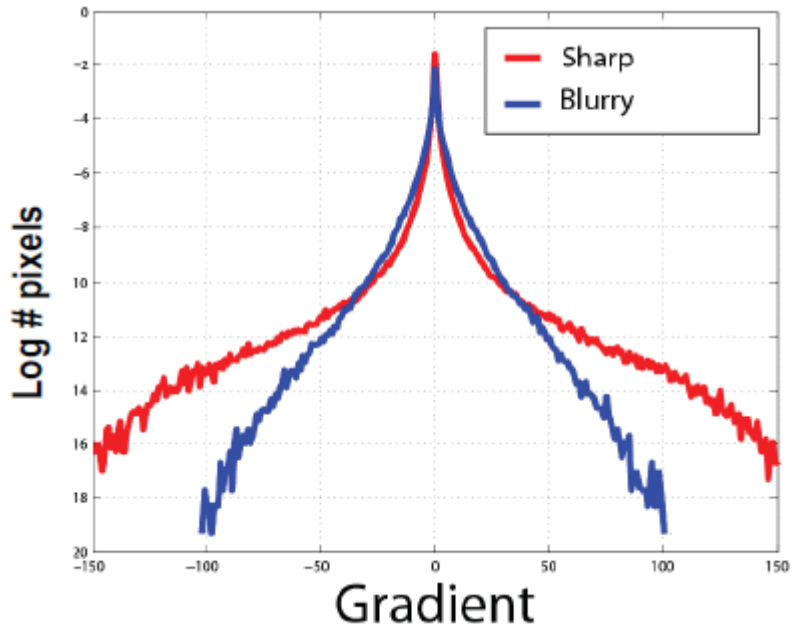


Figure 1.2 - Gradient Graph, Sharp image vs. Blurred image

2.2.3 Evaluation of the research paper

The hart of their algorithm is the ability to correct a blurred image by a reverse blur kernel. That assumption was experimented and found out to be not hundred percent correct but over seventy five percent correct. (Appendix describes all the experiments done for checking the assumptions) The interesting second assumption is that, every natural image has their gradient histogram in a common shape which is mapped to a parametric model. And they use the shape of the histogram to determine whether image is corrected by the selected kernel or not. According to the experiments it was found that those assumptions are not dependable enough to use in an automatic image correction algorithm. It is correct that all the natural images have their gradient histogram in a common shape. But the width and the skewedness of the histogram will be varied depending on the image. The other point is that, a completely unnatural looking image can have the histogram well adhering to their parametric model.

Therefore it will be hard for a computer program to determine the effectiveness of a blur kernel by analyzing the effect it made in the gradient histogram. They have stated that user supervision is needed in the process. It seems to be a considerable amount of feedback from use is required, in order to come to a good result. Otherwise there is a probability that it can end up with a completely weird looking image which adheres to the parametric model.

2.3 Literature Review 2

The paper, Image Shake Correction Image Processing apparatus and Program (Hiroshi Shimizu. [2]) is based on correcting shake effect in a video stream. But this is directly related this project because it corrects the stream frame by frame. A single frame is considered as a single image.

The software contains following components.

Feature detector: - Detects features of the image based on the image data and calculates amount of the features.

Block Selector: - Selects blocks that contains amount of features, larger than a predetermined amount of features.

Motion Calculator: - Calculates a range of magnitudes of motion vectors from various blocks and filer only some vectors based on some threshold and construct a total motion vector.

Displacement corrector: - Corrects the displacements, cased on the motion vector.

Their process is illustrated in the following diagram.

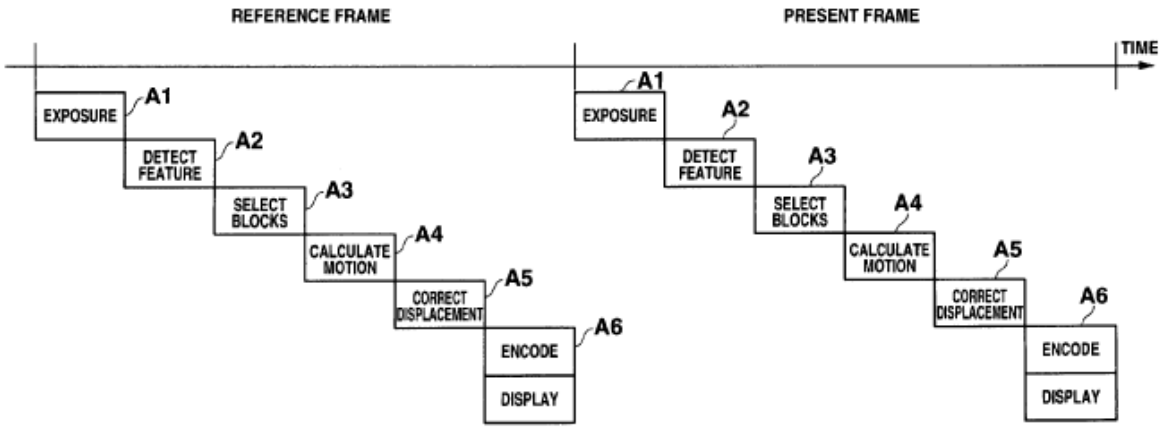


Figure 1.3 - Design

2.3.1 Evaluation of the research paper

This approach divides image into blocks and each block is corrected individually. Each bock can have its own motion vector. Because of that, it supports object movements and camera rotation effect as well.

Features are detected from the image. Image is divided into blocks based on the features. That is a good approach rather than divide the image space.

Step by step process flow and the software architecture are good. Some ideas were taken for the development of the research work.

Chapter 3

Supporting Software Framework

3.1 Introduction

The main development of the project started with a simple tool that has the basic features to visualize an image and its properties. That tool got evolved with the requirements for algorithm development and finally ended up as a software framework like a Test Bed for any image processing algorithm development and testing.

It has features to view the image better, view the slandered graphs that are commonly used in image processing, and some special graphs for the blurred image correction domain. It also has the ability to apply common filters to the image, as well as the basic frequency domain operators.

The tool was developed in C#, with a component based architecture so that any algorithm can be plugged in and tested with the common image processing and analyzing features.

This section explains about the common utilities. The Figure 3.1 illustrates the main UI. Please note that it also contains the UI components that are only relevant to the main algorithm, which are not described in this chapter.

3.2 Common features in the Framework

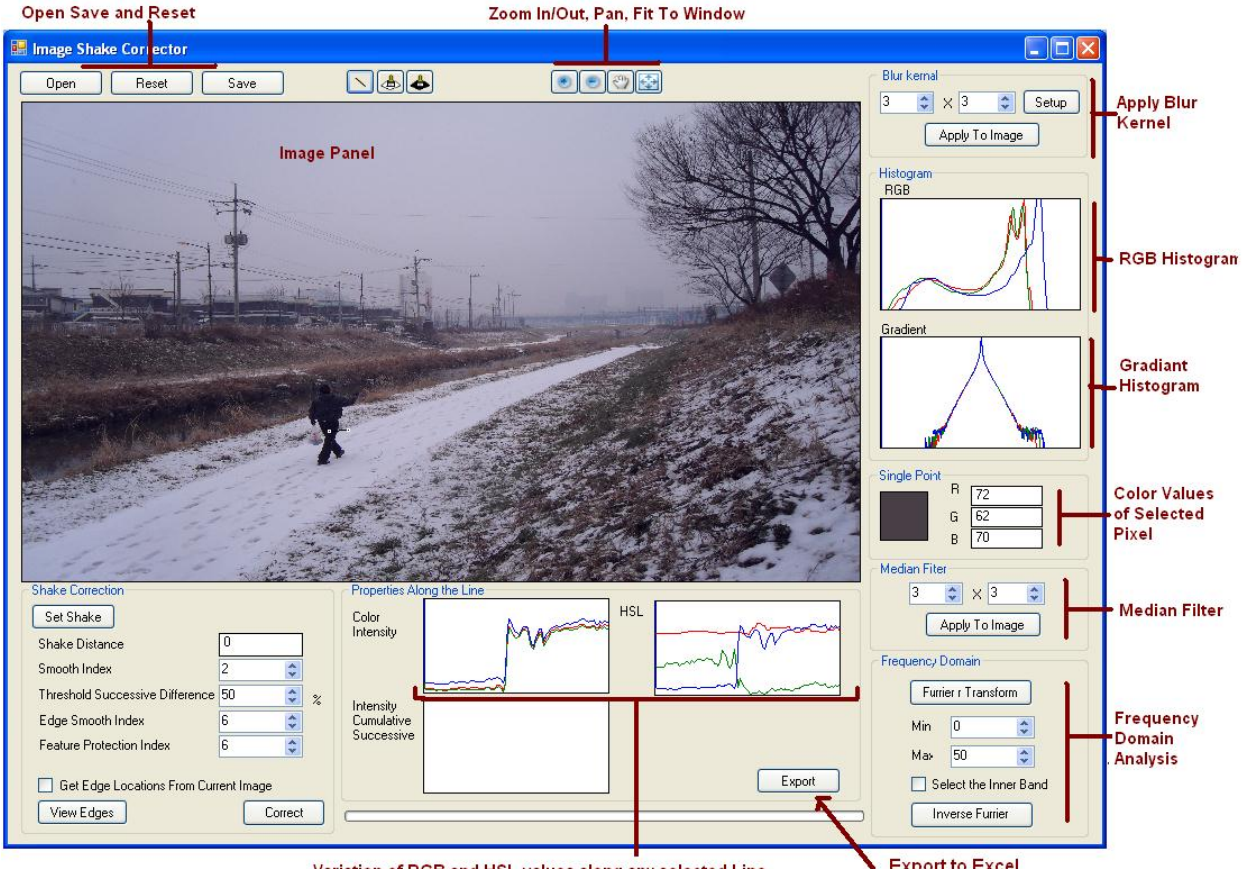


Figure 3.1 The main Software UI

3.2.1 Open, Save, Reset:-

Allows opening image file in any image format supported by windows, save the resulting Image after one or more operations, and reset to original image, after one or more operations.

3.2.2 Zoom In/Out, Pan, Fit To Window:-

Allows viewing fine details of the image by zooming and navigating to locations.

3.2.3 Apply Blur Kernel:-

Allows to apply any blur kernel, with any width and height, with any values.

3.2.4 Image Panel:

Displays the original image initially. After any operation, displays the resulting Image. A new operation can be applied to the image that is in the Image Panel. The image can be bigger or smaller than the dimensions of the image panel. The image will be stretched to display in the image panel, but the operations will be applied to the image which is in the original size.

3.2.5 RGB Histogram:-

Always shows the RGB Histogram of the current displaying image.

3.2.6 Gradient Histogram:-

Always Displays the Gradient vs. Number of pixels histogram of the current displaying image.

3.2.7 Color Values Of Selected Pixel:-

Always shows the RGB values of the pixel respective to the mouse pointer, if the mouse pointer is on the Image Panel. It also shows the actual color of that pixel in a bigger square.

3.2.8 Frequency Domain Analysis:-

Allows the image to be transferred to frequency domain and apply low pass filter and high pass filter, and transfer back to spatial domain.

3.2.9 Variation of RGB and HSL values along the selected line:-

User can draw a line on the Image Panel by clicking two spots. First graph shows the RGB value variation along that line, from starting point to end. Second graph shows variation of calculated HSL values in the same pixels. These are very useful graphs to visualize the shake effect and the effectiveness of any operation that is being tested.

3.2.10 Export to Excel

The color intensity values along the line, can be written to a .csv file which can be opened from Microsoft Excel. The data can be used for lots of experiments with all the data analyzing features that Excel has.

Chapter 4

Definition of the problem and Approach

4.1 Selecting the problem domain

Like any other image processing approaches, the exact algorithm which corrects the image, will be depending on the domain of the image and the shake pattern. In other words, an algorithm that is best for landscape photos will not be best for close up photos.

By analyzing background and shake patterns of lots of shacked images, it is decided to narrow down the area of this research into simple liner shake patterns. Figure 4.1 shows a complex motion and 4.2 shows a simple motion.

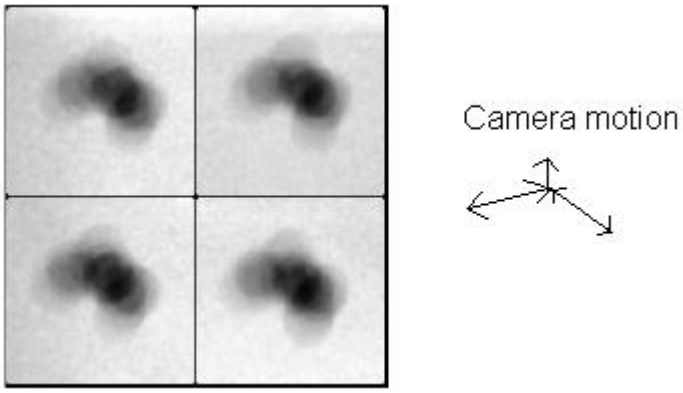


Figure 4.1 - Complex Motion

Figure 4.2 - Simple Motion

4.2 Assumptions:-

- Camera shake is linear and unidirectional (selected Domain).

- Sharp edge is a rapid change in color intensities, and blurred edge is gradual change in the intensities.
- Sharked image can be corrected by converting blurred edges (gradual intensity changes) into sharp edges (rapid intensity changes)
- The distance of the shake (Figure 4.4) is unique throughout the image.
- User can input the Direction and the Distance of the shake.

4.3 Approach:-

Figure 4.3 shows the main tasks that carried out in order to correcting the image.

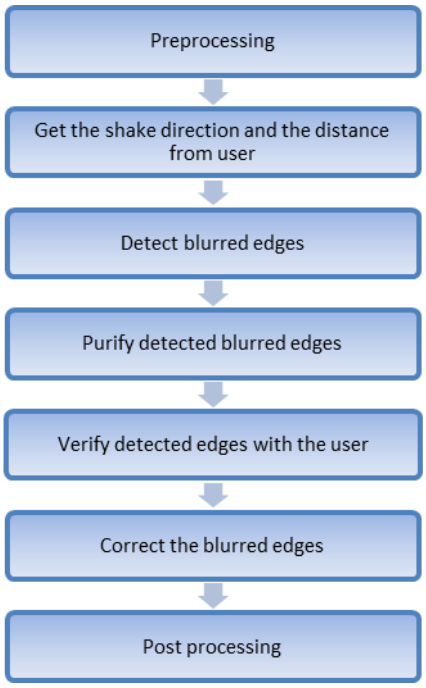


Figure 4.3 Process Floor

Initially it applies some Preprocessing to the image in order to make color values friendlier for the calculations. Next task is getting the shake direction and distance from the user. It is assumed that the user can see the edges and the shake direction manually. Convenient tools are provided for the user to enter that information correctly.

Next is the blurred edge detection process. It detects the blurred edges according to the provided parameters. Detected edges need to be purified in order to make sure the edge locations are correct. The edge purification is done programmatically. User is also given chance to alter the programmatically detected and purified edges in order to verify those are correct as well as all the edges got selected. Edge correction is done next. And then there are post processing techniques to make the corrected image looking more natural.

4.4 Processes in Details

4.4.1 Preprocessing

Initially 5x5 mean filter is applied to the entire image. It removes the noise so that the edge detection algorithm will get more clear data than the actual data.

Following graphs shows the RGB variation and the variation of cumulative successive difference, along a line across a blurred edge, before and after preprocessing.

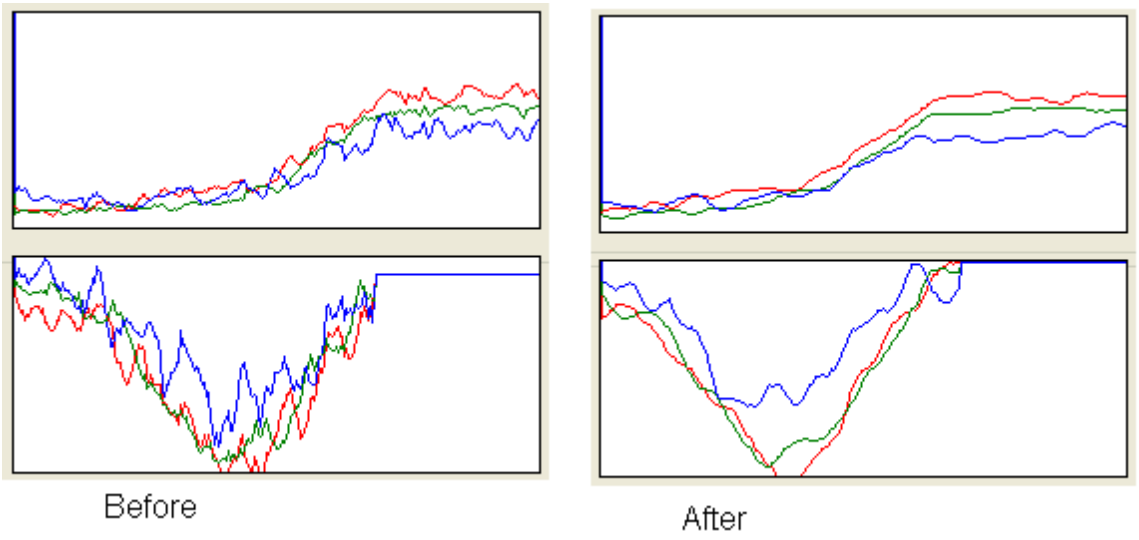


Figure 4.3 – Effectiveness of preprocessing

4.4.2 Getting the shake direction and the distance from user.

An assumption is made that the user has the capability of detecting the blurred edges manually. User has to find a clear blurred edge and draw a line across that edge. The RGB color Intensity graph under “Properties along the Line” will display the RGB variation along the line. If the selected line is across an edge, user can clearly see a ramp in one or more colors in the graph. Then User has to click on the graph and select the start of the ramp (edge) and end of the ramp.

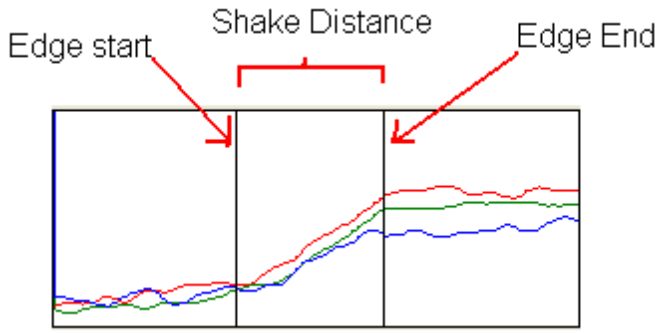


Figure 4.4

By doing so, User is entering a very important parameter "Shake Distance" to the system. That value will be used to detect all the blurred edges in the entire image.

4.4.3 Blurred Edge Detection

Any industry standard edge detection approach does not produce good results for the blurred edges because they are designed for normal images. They fail mainly because the color intensity ramp is too gradual and wide for an algorithm to detect as an edge. In shaked images, sometimes the intensity ramp (edge) can be about 40-50 pixels wide in one direction (Shake Direction) and only about 2-3 pixels wide in the other direction. The width of the intensity ramp (Shake Distance) is varied form image to image.

In this situation, a novel approach was needed to detect such blurred edges. The approach that developed in this project, is based on a factor called "Cumulative Successive Difference"

Successive Difference, in other words color intensity difference between two successive pixels, is used in most of the edge detection approaches. Higher successive difference often indicates a sign of an edge. But with the blurred edges, successive difference between two pixels is not enough for detecting an edge.

The novel approach that developed in this project is as follows. The algorithm knows the shake distance in pixels (say 10). It navigates through the entire image, line by line and culminates the successive differences of 10(shake distance) successive pixels. Likewise it calculates a cumulative successive difference value for each and every pixel in the image.

e.g

Arr[] = *Array of color intensities of a line in the image.*
D = *Shake Distance.*
N = *Current processing location.*

Thus,

$$Cumulative\ Successive\ Difference[N] = Arr[N] + Arr[N+1] + Arr[N+2] + .. + Arr[N+D]$$

Since we have cumulated the number of pixels equal to the Shake Distance, We get a maximum or minimum for the cumulative successive difference value for the pixel at the beginning of the edge.

Figure 4.5 shows the color RGB variation and the Cumulative Successive Difference variation along a line.

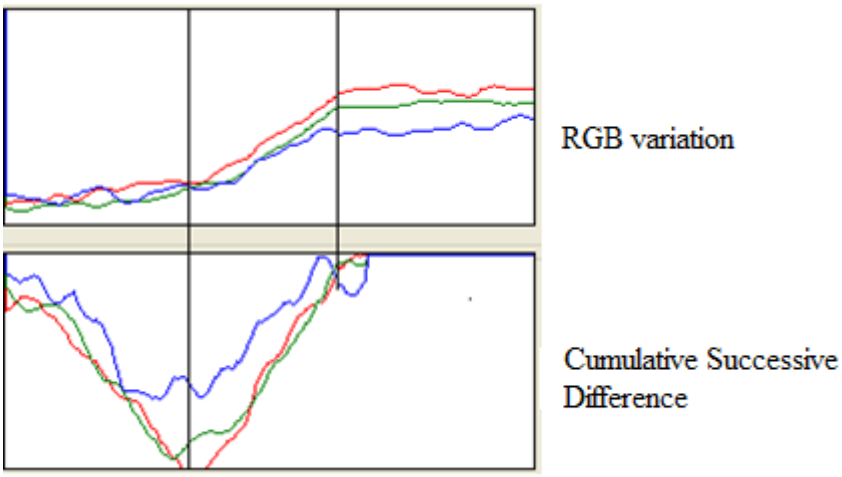


Figure 4.5 Color variation and Cumulative Successive Difference Variation

If the Abstract of the Cumulative Successive Difference value of a pixel is greater than some threshold value, that pixel is detected as a 'probable' start of an edge.

E.g. If the value is greater than 50% of the maximum value found for Cumulative Successive Difference, that pixel is selected for farther processing.

In the UI it is allowed user to enter a percentage value for "Threshold Successive Difference". The value is default to 50%. Higher values will detect fewer edges and lower values will detect more edges. Figure 4.6 shows the effect of the threshold.

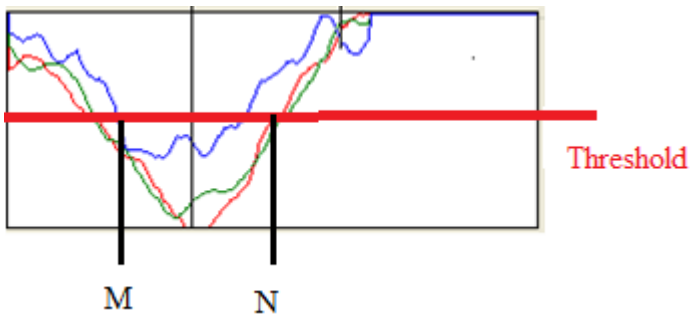


Figure 4.6 Applying a Threshold

For a single edge, there can be a set of pixels selected, which has the abstract cumulative successive difference pass the threshold. In Figure 4.6, all the pixels in the area from M to N have passed the threshold. But we want to find out the exact position respect to the exact 'Start Location' of the edge. A set of clues will be used to determine the actual edge start position from the area from M to N.

There are some occasions that all three colors have the Cume. Successive Difference passes the threshold, and some other occasions where only one or two out of three colors pass the threshold. Even if a single color passes the threshold, it will be considered as an edge.

The actual edge location will be selected as follows.

Arr[] = Array of cumulative successive differences of a line in the image.
M = Start location of the area which passes the threshold (Figure4.6).
N = End location of the area which passes the threshold
Color1 = A color (One form R.G, B)

Clue 1 = (M+N) / 2
Clue 2 = Max value of the Arr[] form location M to N

Clue form Color1 = (Clue 1 + Clue 2) / 2

If this is the only color which passes the threshold in between M and N then
Edge Location = Clue form Color1

If Color1 and Color2 has passed the threshold
Edge Location = (Clue form Color1 + Clue form Color2) / 2

If all three colors has passed the threshold
Edge Location = (Clue form Color1 + Clue form Color2 + Clue form Color3) / 3

From this method, it finds the best edge starting location from the selected values.

4.4.4 Purify detected blurred edges

The output of the edge detection module is an array of edge locations. But somehow, due to noise and anomalies in image data, the detected edge locations are not perfect. Normal shape of the detected edge locations will be as in figure 4.7..

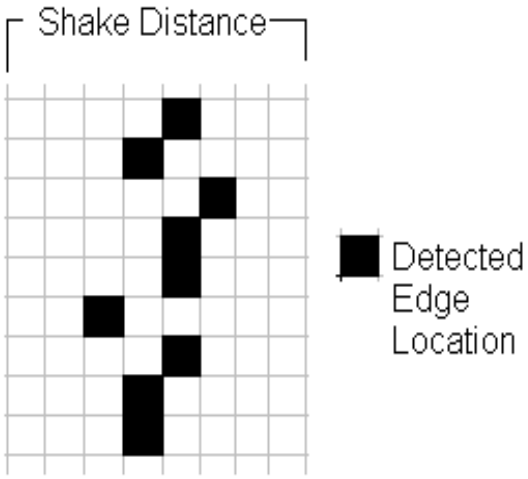


Figure 4.7 Normal shape of the detected edge Locations

Edge location purification is done in two steps. The first step is averaging the locations. It will average 5 consecutive lines and gets the average edge position.

Averaging happens as follows.

The array of detected edges is traversed line by line. When edge location found, checks whether upper two lines and lower two lines has edge locations at the same position or closer to it (within the distance of half of "Shake Distance").

If all the lines have edge location marked closer to that, then take the average location of all 5 lines.

If any other line does not have the edge. Remove the edge location from this line.

If one out of 5 lines doesn't have the edge location marked. Mark the average edge location on that line as well.

Pseudo code of the algorithm is as follows.

N = Current processing line index
Loc1 = Edge location at the line index N-2
Loc2 = Edge location at line index N-1
Loc3 = Edge location at line index N+1
Loc4 = Edge location at line index N+2

If (Loc1, Loc2, Loc3, Loc4) >0 // This line plus four surrounding lines has the edge
NewEdgeLovation[N] = (CurrentEdgeLoc[N] + Loc1 + loc2 + Loc3 + loc4)/5

If (N, Loc1, Loc2, Loc4) >0 and (Loc3=0) // Location3 doesn't have the edge location and all other locations have
NewEdgeLovation[N] = (CurrentEdgeLoc[N] + Loc1 + loc2 + loc4)/4
Loc3 = NewEdgeLovation[N]

If (Loc1, Loc2, Loc3, Loc4) = 0 // Location1, 2 3 and 4 does not have the edge location
Remove current edge location

Although averaging makes the edge locations more accurate, it is not accurate enough to use. Therefore a second step of edge purification was implemented.

In the second step, it marks all the detected edges on the image, from the detected edge starts location, for the distance of “Shake Distance” in white color. After all the edges are marked, it applies a Median Filter on the image. Then it selects the white areas as edges.

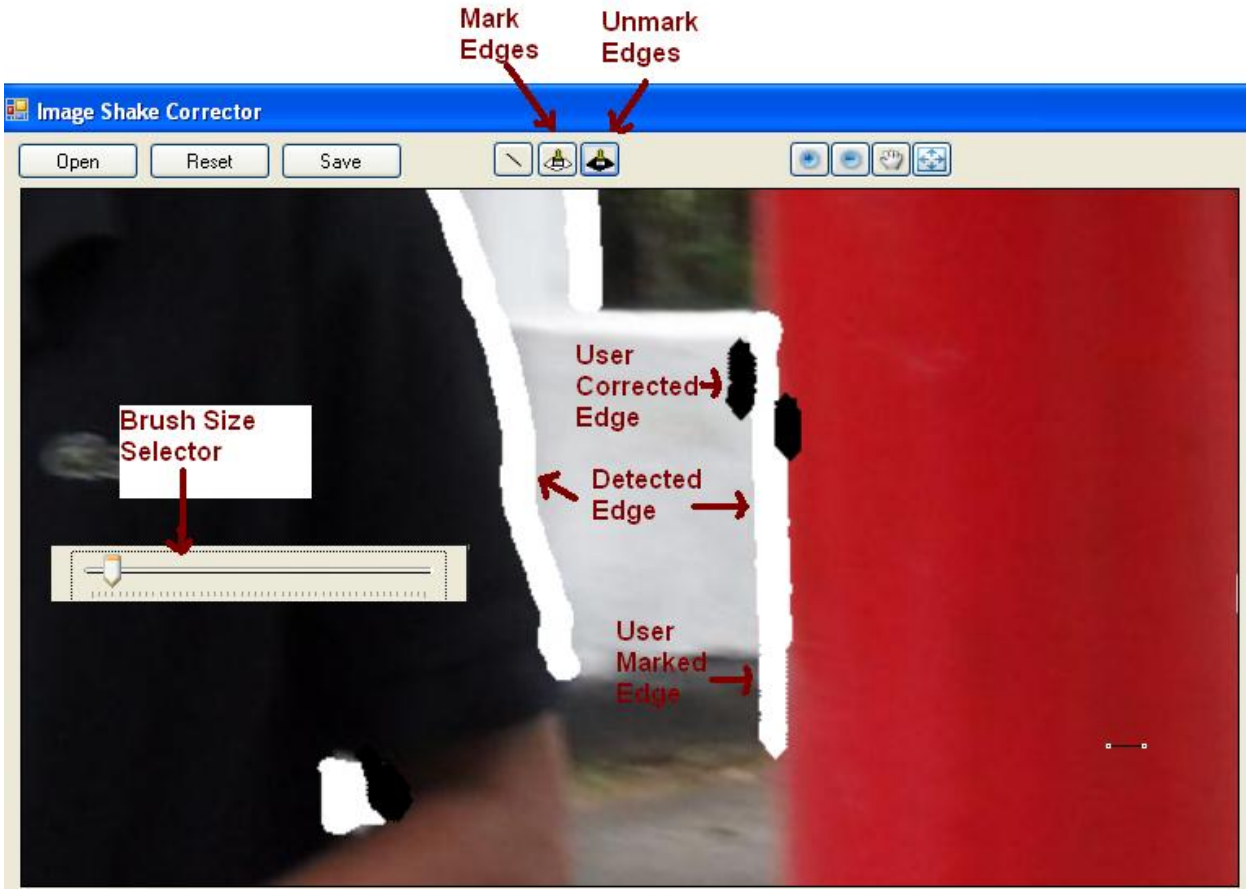
After this step, most of the anomalies get disappeared and we get a smoothed set of edges. Figure 4.8 shows the results from each step.

Figure 4.8 Edge Location Purification step by step

4.4.5 Verify Detected edges with user

There is an option for the user to view and alter the edges that are detected by the algorithms. There is a button in the UI call “View Edges”. When user clicks that, Edge detection module will detect blurred edges and Edge Location Purification module will purify the detected edges and those edges will be marked in white color on the image and displayed.

User can manually check whether all the edges are detected. If it is not, user can use the ‘Mark Edges’ toot to draw edges on the image in white color.



Unmark Edges tool is to delete unnecessary edges that are programmatically selected or drawn by the user. Those two tools can be used as normal paint brush tools in any painting application. With all zooming and panning features, User can carefully draw the undetected edges and correct any wrong edge or mistake made while drawing.

Brush Size Selector will appear when user right clicks on the image panel. There user can select the required size of the brush

4.4.6 Blurred Edge Correction

Edge correction process narrows down the color intensity ramp at the edge. Figure 4.9 illustrates the edge correction process.

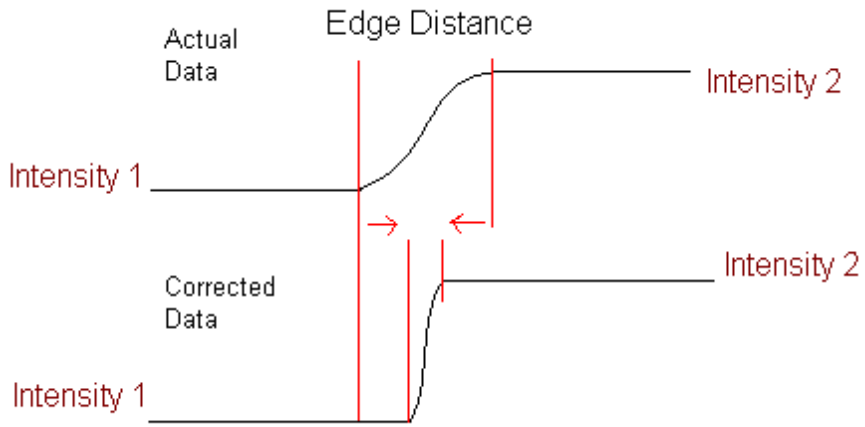


Figure 4.9 Edge Correction

When single color (Either R G or B) and single line is concern, at an edge location, we find following parameters.

Intensity 1 :- Intensity at one side of the edge.

Intensity 2 :- Intensity at the other side of the edge.

Start and end of the edge :- Pixel locations where the ramp starts and ends.

Edge Distance :- Distance of the ramp. This is different from earlier Shake Distance, which was a global parameter. After edges are purified and edited by the user, each edge has its own distance.

Edge Middle location :- The midpoint of the edge.

Methodology:

The corrected (new) intensity of the pixel location inside the edge, will be depending on following parameters.

- Intensity 1,
- Intensity 2,
- Liner interpolation value corresponding to the pixel location.
- Actual (current) intensity of the pixel.

The corrected intensity will be the weighted sum of all these parameters. The liner interpolation value is given a fixed weight in calculation and the weights given to other parameters are made user editable.

Edge Smooth Index: - The weight given to fixed intensity (either Intensity 1 or Intensity 2) depending on which one is closer to the current location. This parameter determines how smooth the edge is, after correction.

Feature Protection Index: - The weight given to the current intensity of the pixel. Even though we need to change the intensity according to the intensities at the two sides of the edge, the new value will be depending upon the current intensity of the pixel as well. Otherwise the image will lose its naturalness as well as it will lose any special shapes or features at the edge locations.

4.4.7 Post Processing

Just after edge correction, image doesn't look good. Edge correction algorithm adds some unnatural features to the image, such as rapid changes in the intensities which are not percent in natural images. Sometimes the colors at the corrected edges are not natural, because each color (R,G,B) is processed separately and the resulting color after all the operations can be quite different from the colors at the two sides of the edge.

So that the post processing procedure applies a 3 X 3 mean filter only for the pixels which are edited at the edge correction stage. After that operation, image looks better and more natural.

Chapter 5

Evolution of the Methodology

This project is a Research and Development project which evolved around the base goal of developing a better methodology for removing the camera shake effect from an image. Entire evolution of the software was based on: - imagine, Implement and try, Decide to use or not, fine-tune.

The basic approach was navigating line by line throughout the image, Detecting blurred edges and correcting them. The problem was divided into following sub problems (modules) from the very beginning of the project.

- Common Utilities
- Pre Processor
- Blurred Edge Detector
- Edge Location Purifier
- Edge corrector
- Post Processor

Each module got evolved separately, in order to make its task better. Success of the entire process is the success of each module. Therefore each module got evolved more or less throughout the project in order to make total process better.

5.1 Common Utilities

The software contains almost all the basic image processing and visualizing utilities they are the fundamental building blocks of most of the image processing algorithms. Namely, Color Histogram, Gradient Histogram, Blur kernels, Median Filtering, Fourier Transformation, low pass and high pass filters, Zooming, Panning and Visualizing the RGB values in a single pixel.

Apart from that, it has another very useful feature to view the color intensity variation and HSL value variation along any user defined line.

The common utilities were started with just Image Panel that only shows the image. Throughout the project, Common Utilities got evolved by adding more and more utilities to make the software rich and able to test the progress of any algorithm.

The color histogram was the first tool that was developed. That is a very common tool used in image processing, in order to visualize the distribution of the color levels of an image. The gradient histogram was implemented as the second utility, in order to evaluate one research paper. The blur kernel was developed to try the edge sharpening kernels that are used in image processing.

The graph that shows the color variation along a line, is the most useful feature for the domain. From that, we can see the color variation at the edges and we can visualize the shape of the ramp graphically. Along with this feature, it has ability to export the color values to .csv file and it can be open from Microsoft Excel. That also was a very useful feature. Second derivative edge detection techniques and some other functions were tested with the variety of graphical and algorithmic features that excel provides.

The most recently added feature is the frequency domain analysis. It was implemented to try getting any help from frequency domain techniques, for the main problem.

5.2 Preprocessor

Preprocessor is the item that didn't got evolved at all. Initially the edge detector was detecting the edges without any preprocessing in the image. Later it was seemed that, edge detection results were getting better if small blur kernel was applied to the image.

Then the preprocessing utility was added to the main process. It started with 5x5 mean filter. And it was found to be the best preprocessing and never wanted to change it later.

5.3 Blurred Edge Detector

This module got evolved a lot. Started with very common edge detection technique called "Successive Difference". There after it moved to "Sobel" operator. Both of these operators generated very poor results in detecting blurred edges.

The shaked image normally has two types of edges. Edges that are perpendicular to the shake direction are with a gradual color ramp. The edges that are parallel to the shake direction are rapid ramps. The normal edge detection techniques are good in detecting rapid ramps but poor in detecting gradual ramps. But for the shake correction problem, what needed is a good algorithm that detects gradual edges.

Finally it was decided to implement a Novel approach using "Cumulative Successive Difference" the details about the approach is in chapter 4.

The novel approach was also started with fixed threshold and considering every pixel that passes the threshold, is an edge. Later, a requirement came to make the edge thinner and the edge location should be very much accurate. With these requirements, algorithm evolved to calculate one single location, from a set of locations that passes the threshold.

5.4 Edge Location Purifier

This is the module that had the maximum evolution. Since the edge detection algorithm detects the edges in line basis, there is a huge probability of having a considerable distance between the detected edge locations correspond to the same edge, in two consecutive lines. That cannot be avoided from the edge detection module because of the noise.

A lot of trial and error experiments were done on finding a way to purify the edge locations and eventually came to a process with three steps. Step 1 and 2 are described in the above captor and the third step is to show the detected edges to the user and allow user to edit and verify the edges.

For that step, In order for user to edit the edges, Two paint brush-like tools were developed in the UI.

5.5 Edge Corrector

Edge Corrector module was started with simply narrowing the edge width unto zero by applying the colors at the both sides of the edge, till the middle of the edge.

Obviously it didn't produce good results. The second approach was narrowing down the edge unto pre-defined size and applying the linear interpolation, throughout the new edge distance as in the figure 5.1.

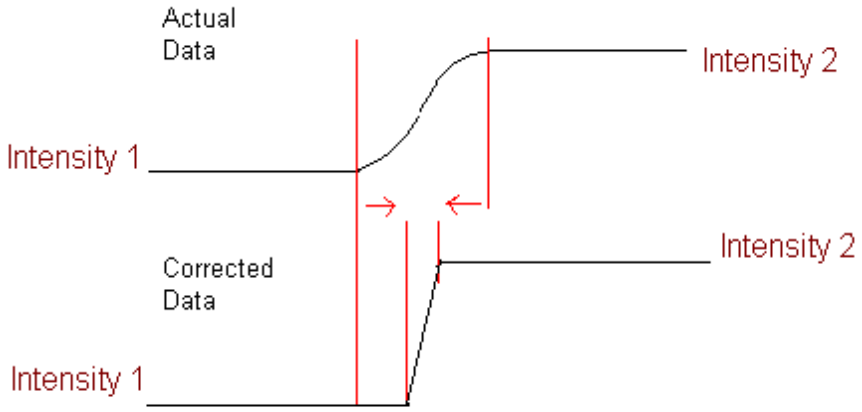


Figure 5.1 Second Approach

That approach was little better, but the resulting image from that algorithm was not looking natural.

Third approach was introducing the image smooth index and makes the curve shape more natural. That illustrated in Figure 5.2

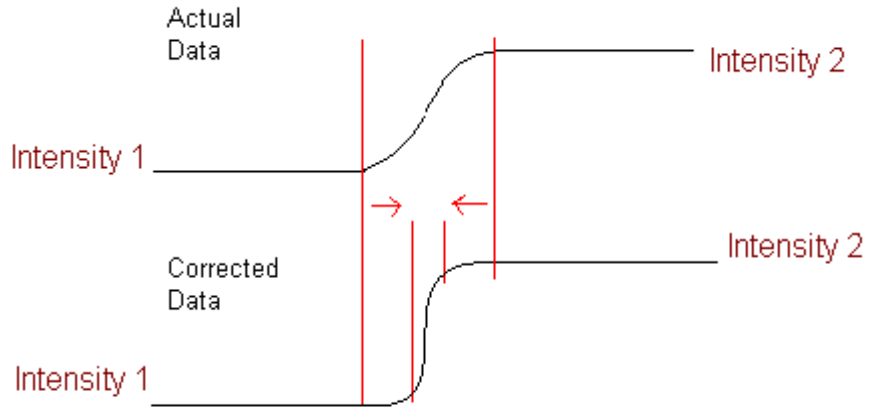


Figure 5.2 Effect of Smooth Index

Even though the ideal graph looks OK, this algorithm had the problem of losing image features. If some special color pattern (feature) excise near to the edge, this algorithm deletes it completely, because algorithm does not care about the actual color value of the edge pixel, rather it applies a calculated value, which is only derived by the colors at the two sides of the edge. After considering the fact that the new color value of the pixel should be dependent on its old value as well, a parameter called “Feature Protection Index” was introduced.

Now in the latest implementation, the new value of the pixel is calculated depending upon the values at the both side of the edge, the distance from current pixel to the both sides, as well as the old value of the pixel. Figure 5.3 illustrates that.

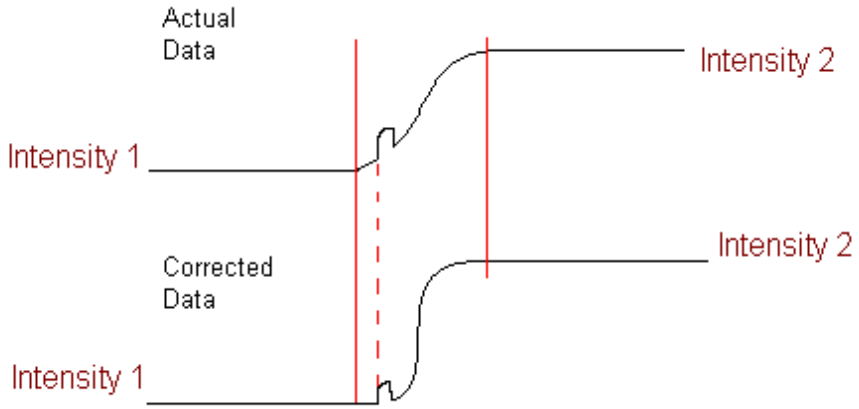


Figure 5.3 Feature Protection

5.6 Post Processor

Intention of the post processor module is to make the image look natural at the end of processing. Since we are applying completely new values for the edge pixels, even though they were calculated with good algorithms, there is a huge chance to have the edge pixels look unnatural. After correcting the image, if all R, G and B values are not marching each other at the edge, it is clearly visible as an unnatural color at the edge. The second problem is when the edge made thinner; it was clearly visible as an unnatural edge.

Post processor is introduced to remove those effects. Initially tried with 3X3 mean filter, for the entire image. It did a great job but since it affect to the entire image, it blurs the image, as well as it takes time.

Then it was made in a way that it applies only for the edited pixels. And it is producing good results.

Chapter 6

Results and Evaluation

6.1 Experiment

Figure 6.1 is a blurred image due to camera shake, opened by the application.

First User has to draw a draw a line across a blurred edge. Then the shake distance has to be marked on the “Color Intensity along the line” graph.

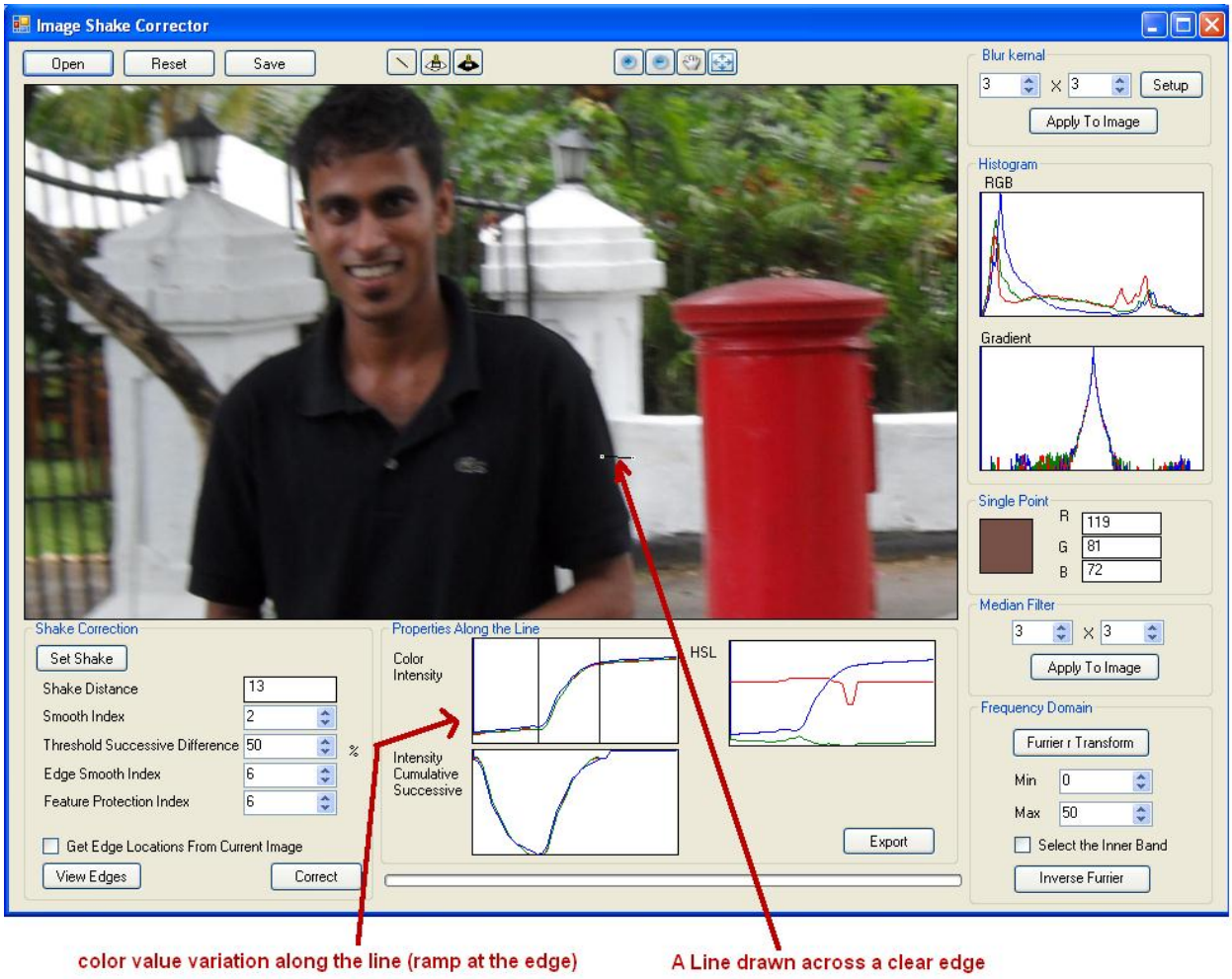


Figure 6.1 Blurred image, opened with the application

Then user has to click “View Edges” to view the selected edges and verify them. The selected edges will be shown in white color. If the edge detection is not seems to be good, user can reset the image and decrees the “Threshold successive difference”.

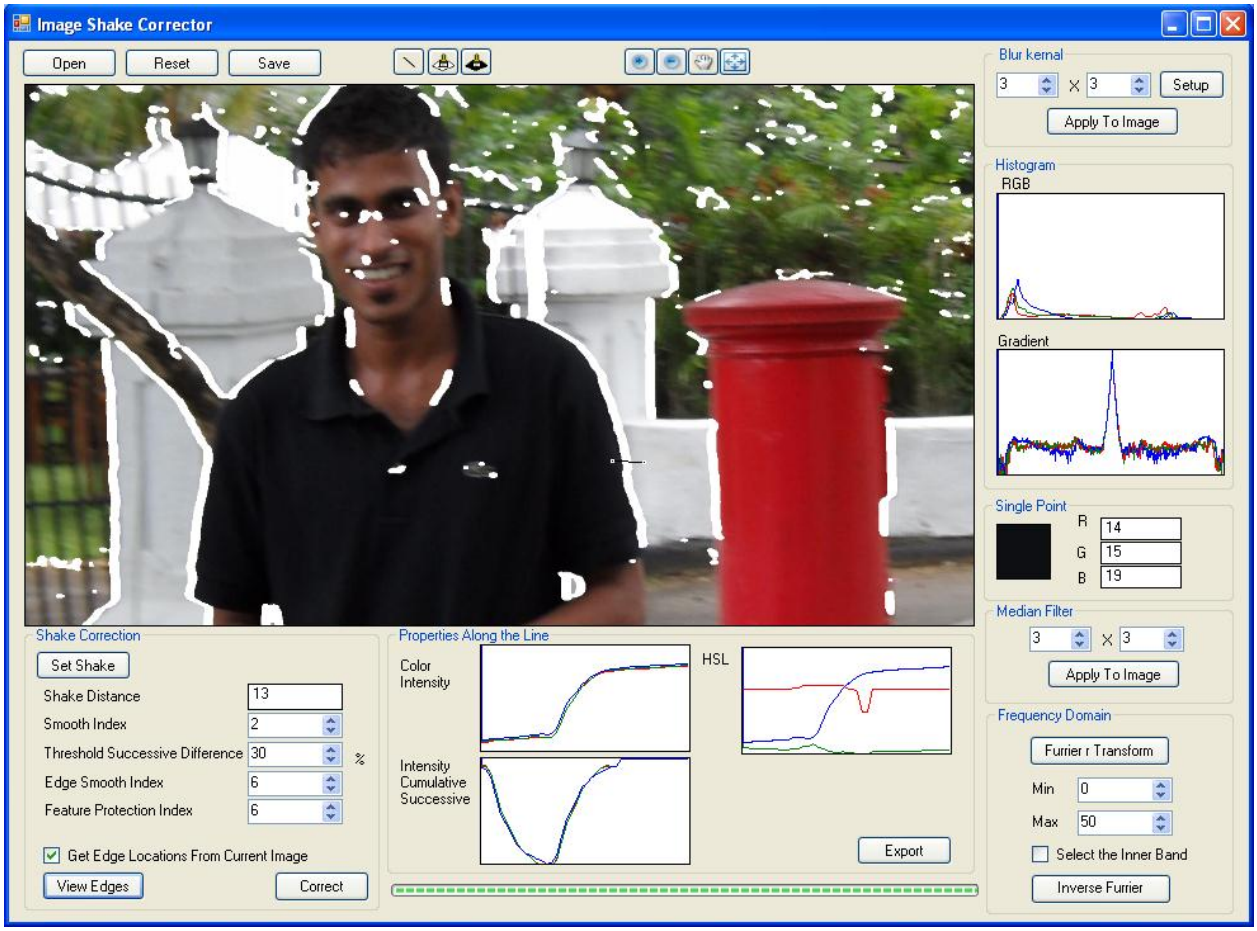


Figure 6.2 Edge Detected image

Figure 6.2 shows the image after edges are marked. Still if all the edges are not selected, or unwanted edges were selected, user can alter the selected edges by “Mark Edge” and “Unmark Edge” tools. Once the edge verification completed, user has to click “Correct” button to correct the image.

Figure 6.3 illustrates the image after correction. “Color Intensities along the line” graph shows the edge correction effect very clearly. User can change the “Edge Smooth Index” and “Feature Protection Index” until gets the best corrected image.

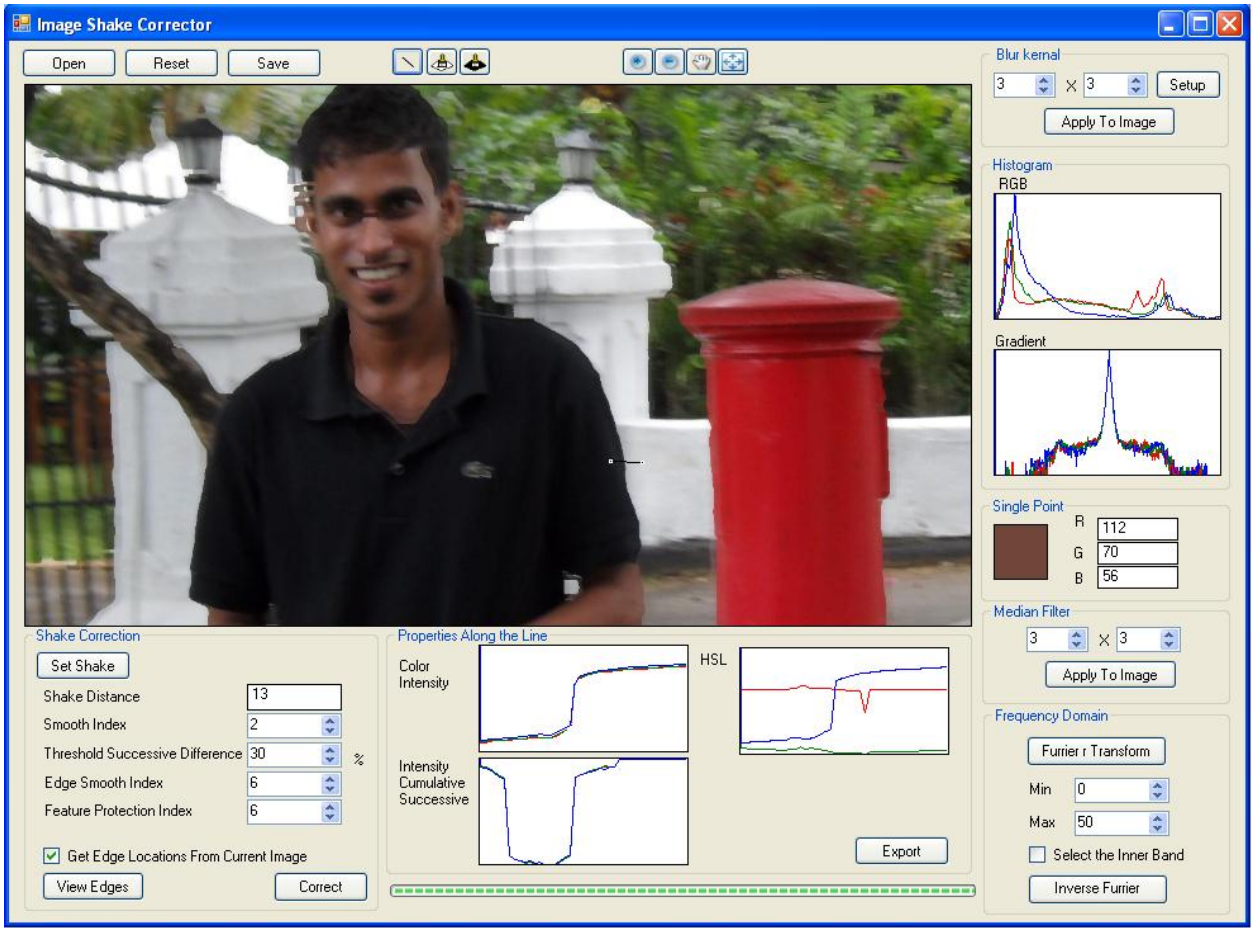


Figure 6.3 Corrected Image

6.1 Evaluation about the results

When concern about the above experiment, the results are great. The program does a great job with the images with following qualities.

- Images with clear edges: - Even though edges are blurred, it's better if they are clean and clear
- Simple Images: - Images with less number of objects on it. So that less number of edges.
- Images with flat colors: - Such as Close-up images or images of vehicles etc.
- Less blurred images: - There is some limit of blur that the algorithm can correct.

Even though image that we test don't have any of these qualities, the process can correct it up to some extent, but the result can be not very good as the above example.

Chapter 7

Conclusion and Future Work

7.1 Conclusion of the project

Goal of the project is to do research and develop a solution for the problem of Camera Shake Effect, in a photograph. There is no any straight forward solution for the problem in the industry. Therefore there is a huge space for the researchers to research and come up with solutions.

During this project, a great deal of research work was done and a methodology was invented to correct a blurred image due to camera shake. It was an evolutionary development. The basic concept was proved with the project, that is the invented methodology works and can be used in image shake correction. Like any other methods, this method also can be improved in lot of ways to make the output better.

The software tool, which is the output of the project, can correct blurred images in various domains. It can remove the blurred effect and make the image look better. But it has limitations as well. If the blurred effect is too high, the software cannot make the image better. Rather it makes the image worse. Knowledge about the research, methodology and the parameters is required up to some extent, in order to use the software. But the software was made user friendly as much as possible.

Finally the output of the project is a successful methodology of correcting the shaked images. It also includes a Software framework that can be used for any image processing algorithm development and testing. The methodology has a component based architecture. Each component can be improved individually in order to make its output better.

7.2 Future Work

Improving the output of the methodology is, improving the outputs of each module. Because of the component based architecture, any module can be improved without affecting to others.

Improving a module involves great deal of imagination, development, Testing and Fine-tuning. The Test Bed framework can be used for basic testing but when it comes to more advanced testing, the features will not be enough and it also need to be improved in parallel.

According to testing, the Edge Detection module and Edge Correction module are found to be the main components that need improvements. Other modules do quite a great job and don't need much improvement for the moment.

The Edge Detection module detects most of the edges but when the colors at the both sides of the edge are closer, it fails. Going along the edge, like 'Canny' operator, as well as getting some clues of edges from frequency domain techniques, was tried but given up due to time constrains. Those were seemed very good techniques to try in future. The results need to be analyzed carefully and the method needs to be fine-tuned in order to provide best results.

The edge correction process involves a set of parameters. Process itself is very good, but the results depend on the parameters. Currently there is no way to find the most accurate

parameters automatically. User has to manually enter them. Once parameters are entered, all the edges in the entire image will be corrected using them. Since the edges are different form each other, the parameters that corrects the edge best, can be different from each other. Therefore an algorithm is needed to analyze an edge and calculate the best values for the parameters used in edge correction.

Appendix

Experiments to evaluate the research paper, Rob Fergus et al. [1] (Removing Camera Shake from a Single Photograph)

Experiment 1: - Testing the effectiveness of sharpening, using a blur kernel.

1. Select an image and apply a known blur kernel to blur the image, and save the blurred image.
2. Take the blurred image and apply the image sharpening kernel, respective to the previous blur kernel and save the sharpened image.
3. Compare original image and sharpened image

Test results for 3 x 3 kernels

Figure 1 – Blurring and sharpening 3 x 3

Observation: - Sharpening is very good. The edges of the sharpen image is little bit over sharpen than the source image.

Test results with 9 x 9 kernels

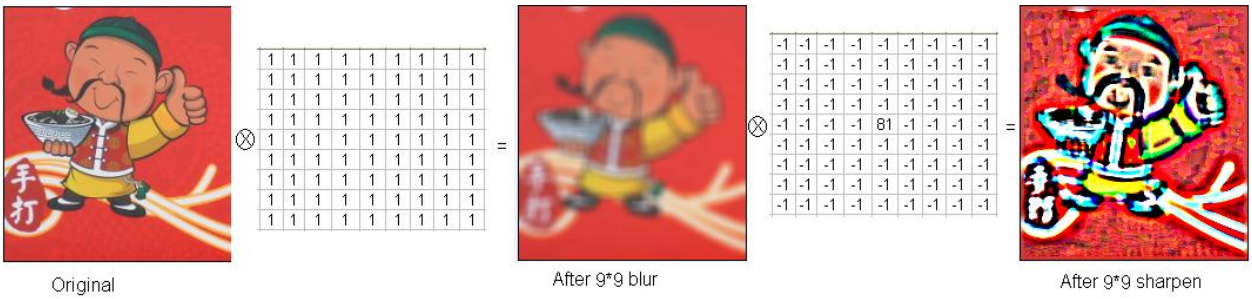


Figure 2 - Blurring and sharpening 9 x 9

Observation: - The edges are considerably over thick in the sharpen image.

Conclusion: - Even if we know the exact blur kernel, it is absolutely impossible to derive the exact original image, from a blurred image. What we can achieve is, something which is closer to the original. So that an image sharpening algorithm have to use various assumptions and properties of natural images, in order to come to a good result.

Experiment 2**:- Testing whether the gradient Vs. Log(2) probability density graph for various images takes the same pattern.**

For this, software was made to plot the gradient Vs. Log(2) probability graph for all R,G and B values.

Following are the results for some photographs. Four photographs are chosen, which are assumed to be in completely different domains.

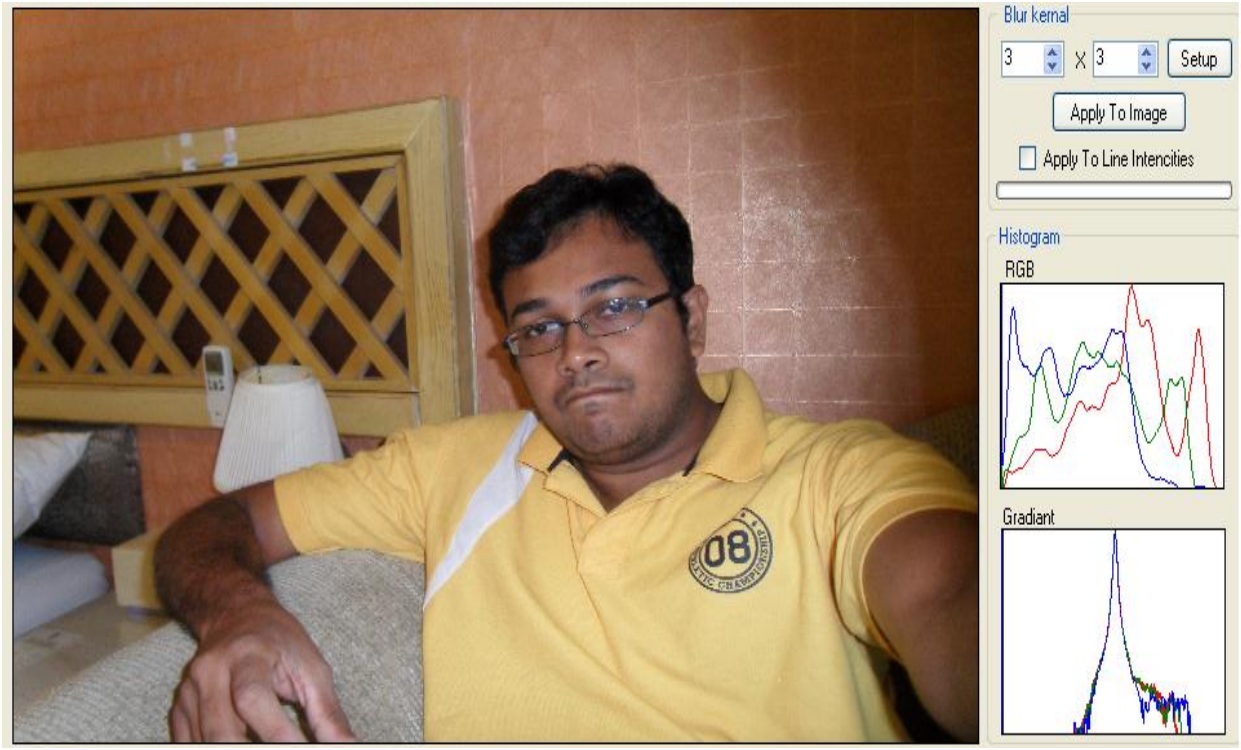


Figure 3 - Close-up image

Figure 4 -Dark Image

Figure 5 - Bright Image

Figure 6 - Text Image

Observation: - Images in the above examples are very different from each other. A close up image, a darker image, a brighter image and a text image. Although the color histograms (right middle) are very different, the gradient (successive difference) graph almost looks like same pattern. But somehow the patterns are not exactly same. Some are little bit wider and some are narrower. Bottom part of some are little bit skewed to one direction.

Conclusion: - By looking at the gradient graph. It is hard to determine whether it is a natural image or not.

Experiment 3**: - Testing whether the gradient graph is narrower, for blurry images**

For this following steps were taken.

1. Selected one image and plot the graph.
2. Then applied 5 x 5 blur kernel and plotted the Graph.

Following are the test results.

Figure 7 - Original Image

Figure 8 - Blurred Image

Observation: - Gradient graph is narrower, for blurred image.

Experiment 4 **: - Testing whether, making gradient graph wider, always corrects the image.**

I selected the blurred image form experiment 3, and applied some wrong sharpening kernel.

Following are the results.

Figure 9 - Blurred Image

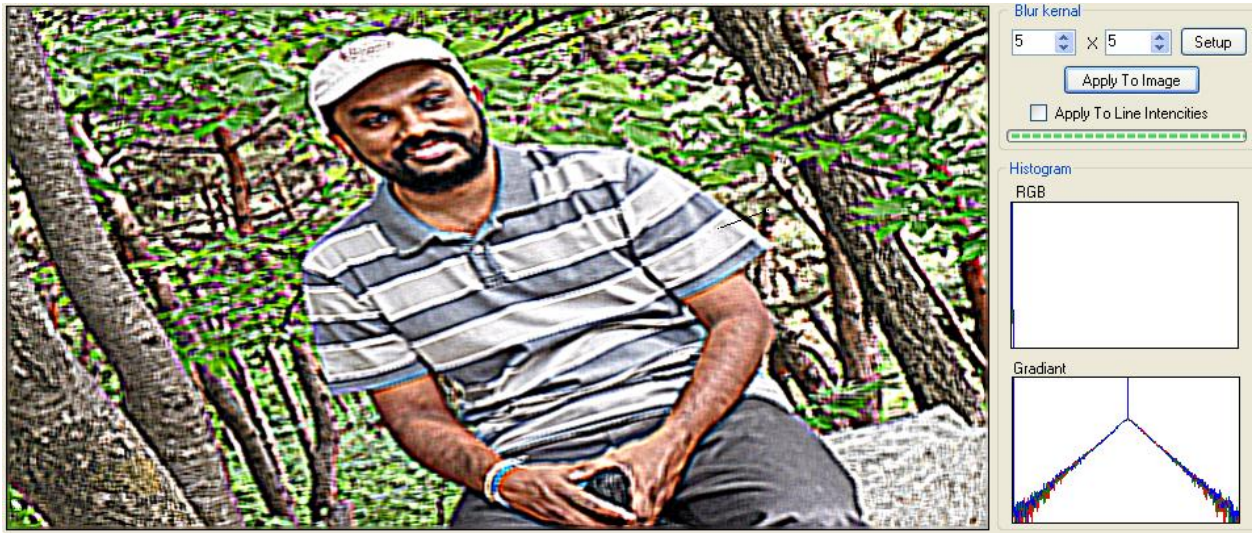


Figure 10 - image with Wrong Sharpening

Observation: - Making the gradient graph wider does not guaranties that we are getting the correct output.

References

[1] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman Removing Camera Shake from a Single Photograph

[2] Hiroshi Shimizu Image Shake Correction Image Processing apparatus and Program.

[3] AForge.NET open source framework[Online] 2011
http://www.codeproject.com/Articles/16859/AForge-NET-open-source-framework
(accessed September 2012)

[4] Image Processing Operator Worksheets[Online]
http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm
(accessed April 2012)

[5] Image Processing in the Frequency Domain[Online]
http://www.cse.lehigh.edu/~spletzer/rip_f06/labs/lab4.pdf
(accessed October 2012)