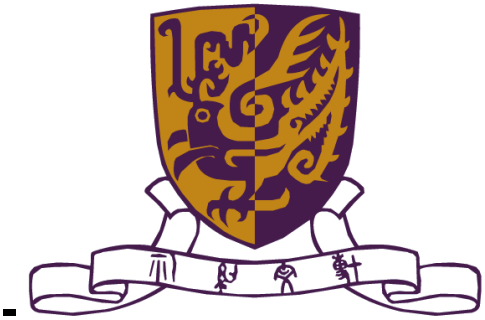


*Department of Computer Science & Engineering*

**The Chinese University of Hong Kong**

**2005 / 2006 Final Year Project**

# First Term Report

**LYU 0503**

# Document Image Reconstruction on Mobile Using Onboard Camera

***Supervisor***

**Professor Michael R. Lyu**

**Leung Man Kin**

**Ng Ying Kit**

**Prepared by Leung Man Kin**
**2nd, December, 2005**

# Abstract

The report would summarize all of our work in the final year project which is entitled Document Image Reconstruction on Mobile Using On board Camera. The objective of our project is to reconstruct a whole document using the onboard camera to take photo of different parts of document. This report would state the motivation, background information, experiment result and difficulties when we are participating in the final year project.

Firstly, we would introduce the idea of our final year project. Then we would describe Symbian, which is the major operating system in mobile phone. After that, we would discuss about the testing platform on PC. Next, we would discuss about what algorithm is used to implement to reconstruct the document. The algorithm is image alignment and stitching. There are many different of such algorithms. In our project, we are focusing on feature-based registration and use "Scale Invariant Feature Transform" to extract the features. Following is our implement of our project and some results. At last, we would finish the report by stating the difficulties encountered and our future works.

# Content

# Chapter 1 Introduction

## 1.1 Motivation

Nowadays, mobile phones become essential handheld devices for every body. According to a statistic at the end of 2005, each person in Hong Kong owns average of 1.25 mobile phones.

Traditionally, mobile phones only enable people to communicate with others by voice. However, as technology is developing rapidly, many phones are embedded with process and the processing power of mobile phone is increasing. Current mobile phones are embedded with more and more functions, including video conferencing, music jukebox, video player, photo-taking, web surfing, games etc. At the same time, the processing power of mobile phones and the quality of the onboard cameras are improving continuously.

We think the use of camera should not be limited to these functions and the market of software on mobile phone is potentially large, so we think of a way to add more values to the camera of phones.


*Fig.1 Nokia N90 mobile phone*

## 1.2 Project Objective

Our project is to develop a program on Symbian phone. The program can use the camera to take photos of different parts of a document. Then the program would make use of suitable algorithms to stitch the picture and output a full document image.

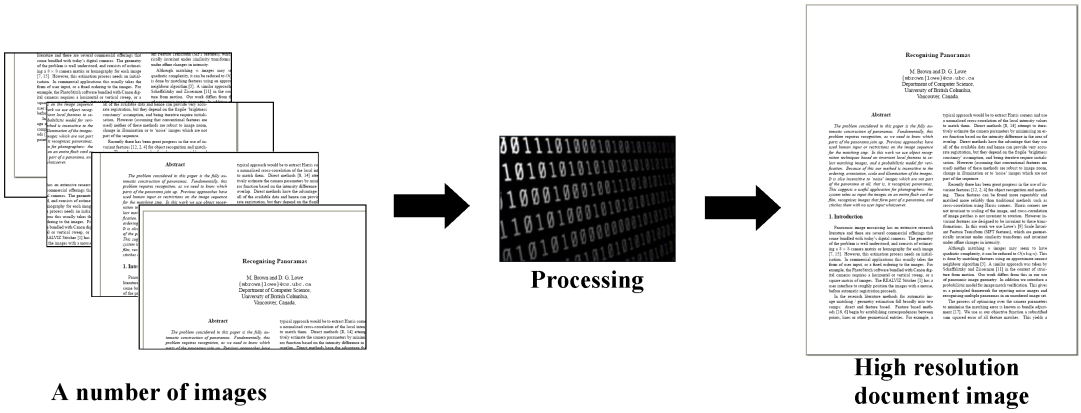



*Fig.2 Overview of project*

The reason, which we need to take photo of different part instead a whole document is quality of the photo of a whole document is poor. You cannot even identify the character on the document.

After the document has been created, the document can be transmitted to other people. The transmission can be by means of fax, MMS and even email, depended on the phone supported itself.

As a result, our program can make the phone be a document scanner and transmit it by different means.

## 1.3 Project Equipment

The equipment we use is Nokia 6600 and Nokia 6630. They are built-in with different version of Symbian OS. The operation system of Nokia 6600 is Symbian OS 7.0. The operation system of Nokia 6630 is Symbian OS 8.0. They are mainly used to do testing for Symbian API. For example, we have done testing for image accessing, image decoding, image encoding, camera controlling and faxing.

However, we have not made use of them to do the document restoration part so far. But we have done, tested the restoration part on PC first. The picture comes from the photo taken by Nokia N90. The operation system of Nokia N90 is Symbian OS 8.1.


*Fig.3 Project equipments – Nokia 6600, 6630 & N90*

# Chapter 2 Symbian OS

## 2.1 Introduction

Symbian OS is a mobile operating system from Symbian Ltd. It is licensed by the world's leading mobile phone manufacturers. Symbian OS is designed for 2G, 2.5G and 3G mobile phones. It is a 32-bit preemptive multitasking operating System. It includes a robust multi-tasking kernel, mobile telelphony support, communications protocols, data management and synchronization, international support, multimedia support, graphics support, a low-level graphical user interface framework, etc.

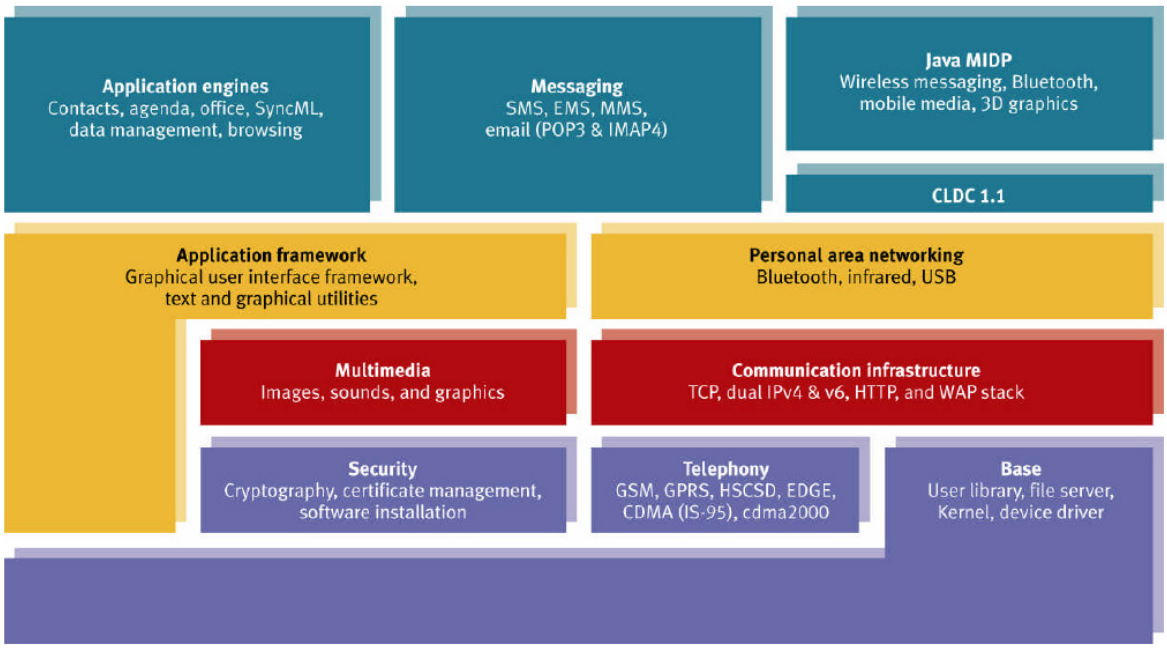



*Fig.4 Architecture of Symbian OS 8.0*

The following focuses on how to write programs for Symbian OS and limitations of mobile phone. In additional, we would focus on how we make use of libraries so that we can perform the following jobs:

- Image processing;
- Image decoding and encoding; and
- Fax function

## 2.2 Development Environment

To develop program on Symbian phone, we can use either Java or C++ to implement programs. However, the library for Java is not as complete as C++. For example, it cannot support getting raw frame data from onboard camera continually. Further, as Java is run on Java Virtual Machine. As a result, the speed would be slow. However, our product focuses on image processing. It needs a lot of computation power. So Java is not suitable for us.

On the contrary, The C++ library allows access to numeral application engines, such as fax function, camera and image decoding and encoding. So we choose this as our programming language for the Symbian OS.

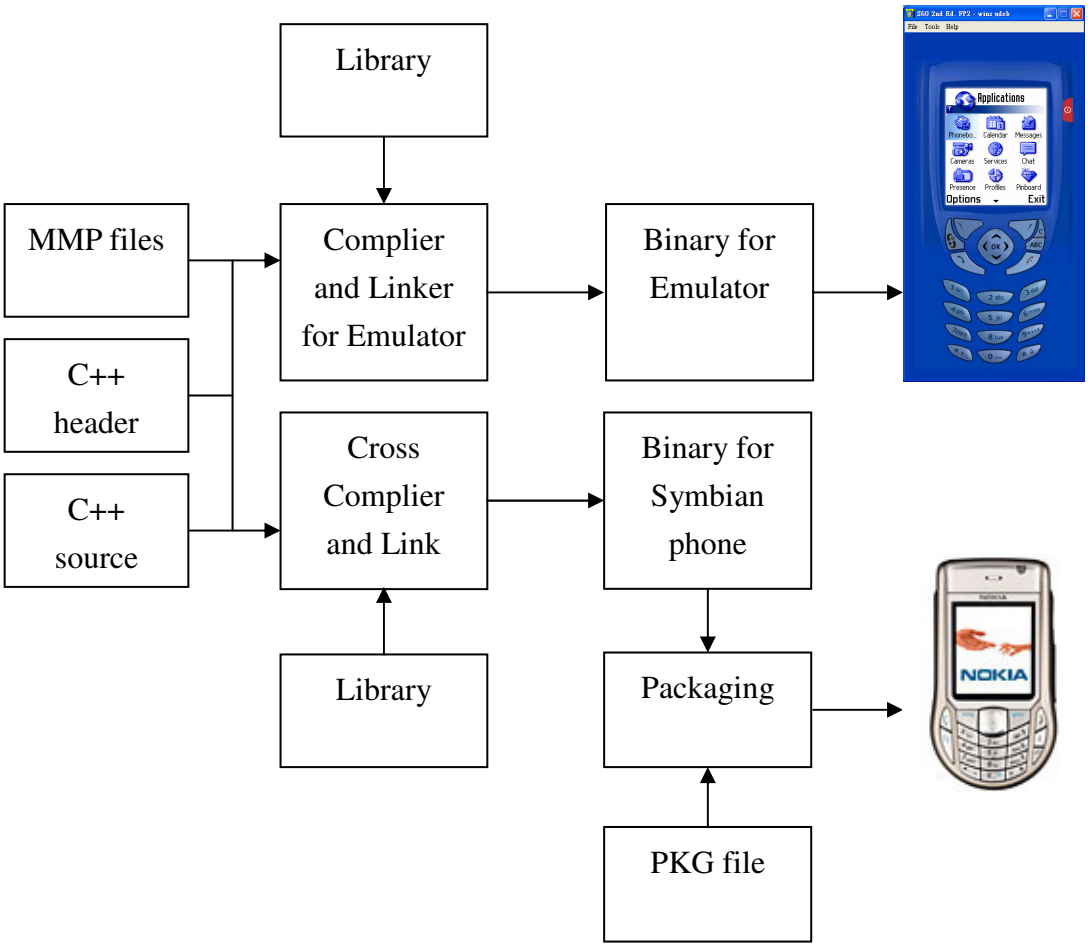



*Fig.5 Procedures to make application for Symbian phone*

To develop applications for Symbian phone, we need to use Series 60 SDK for Symbian OS. There are many versions for this. Each would be suitable for Symbian phone with specific version of Symbian OS. However, there is only little different. In our case, we use "Series 60 2nd Edition SDK for Symbian OS, Supporting Feature Pack 2, For C++" as SDK and "Visual Studio .Net" as IDE to edit and compile our program.

Firstly, we can use the application wizard provided in SDK to create a simple sample program. It would create all corresponding files for compiling the application.

Secondly, we can edit C++ source file and C++ header file to suit our need. Sometimes, we also need to edit MMP (project definition) file. The file specifies the properties of a project component in a platform and compiler independent. It specifies what files and library need to be included to compiler the application.

Finally, we can compile the program and link related library to produce the application. There are two different compilers. One is used to compile for the emulator. It is run on PC for us to do testing and debugging. The other is used to compile for the phone. It is run on Symbian phone.

To install the application program on the Symbian phone, we need to supply a PKG file and package the program into an installation file (SIS file).

As we have emulator on PC, the development cycle would be:

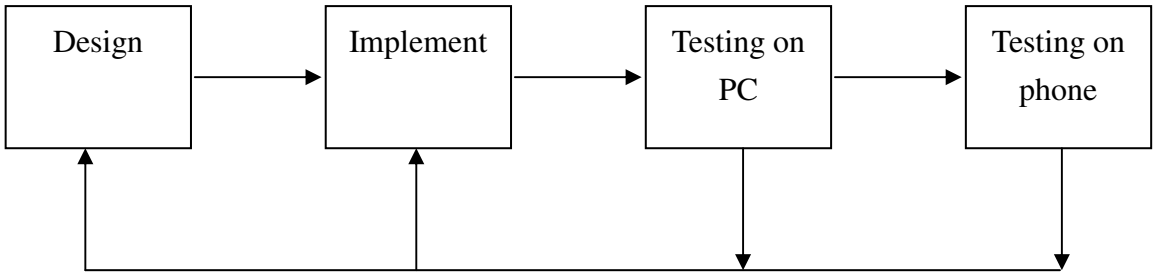



*Fig.6 Development cycle*

Before testing on the phone, we test the program on PC first. However, we should not rely heavily on it, as even if the program can be run on PC, it does not guarantee that the program can be run on the Symbian phone.

## 2.3 Limitation in Symbian phone

Here is the specification of some Symbian phones.

| Nokia 6600 Technical Specification | | |
|---|---|---|
| Operating System | Symbian OS v7. 0s |  |
| Developer Platform | Series 60 Developer Platform 2.0 | |
| Memory | Heap: 3MB<br>Shared Memory for Storage: 6MB<br>Unlimited Jar Size | |
| Camera | VGA Camera with 2x digital zoom | |
| Processor | 32-bit RISC CPU based on ARM-9 series, 104 MHz | |

| Nokia 6630 Technical Specification | | |
|---|---|---|
| Operating System | Symbian OS v8.0a |  |
| Developer Platform | Series 60 2nd Edition, Feature Pack 2 | |
| Memory | Heap: Unlimited<br>Shared Memory for Storage: 10MB<br>Unlimited Jar Size | |
| Camera | 1.3 megapixel camera with 6x digital zoon | |
| Processor | 32-bit RISC CPU based on ARM-9 series, 220 Mhz | |

| Nokia N90 Technical Specification | | |
|---|---|---|
| Operating System | Symbian OS v8.1a |  |
| Developer Platform | Series 60 2nd Edition, Feature Pack 3 | |
| Memory | Heap: Unlimited<br>Shared Memory for Storage: 30MB<br>Unlimited Jar Size | |
| Camera | 2 megapixel camera with 20x digital zoon<br>Carl Zeiss optics | |
| Processor | 32-bit RISC CPU based on ARM-9 series, 220 Mhz | |

It is obviously that the processor power of all these phones is low. They cannot be compared to the power of PC. As a result, the efficient of the program should be optimized carefully so it can be run on phone smoothly. Our program is about image alignment and stitching. It may involve lots of computation power. We should investigate ways to improve the efficient of our program.

Furthermore, there is no special hardware to process floating point operation. So when there is floating point operation, the phone would need much to perform floating point operation.

Besides, the memory of the phone is limited. In Nokia 6600, there is only 3MB heap size. Although the specification writes that there is unlimited heap size, it still bounded by the total memory. So the memory space would not be as large as that of PC. So we should be careful to handle the memory when design and implement application program for Symbian phone.

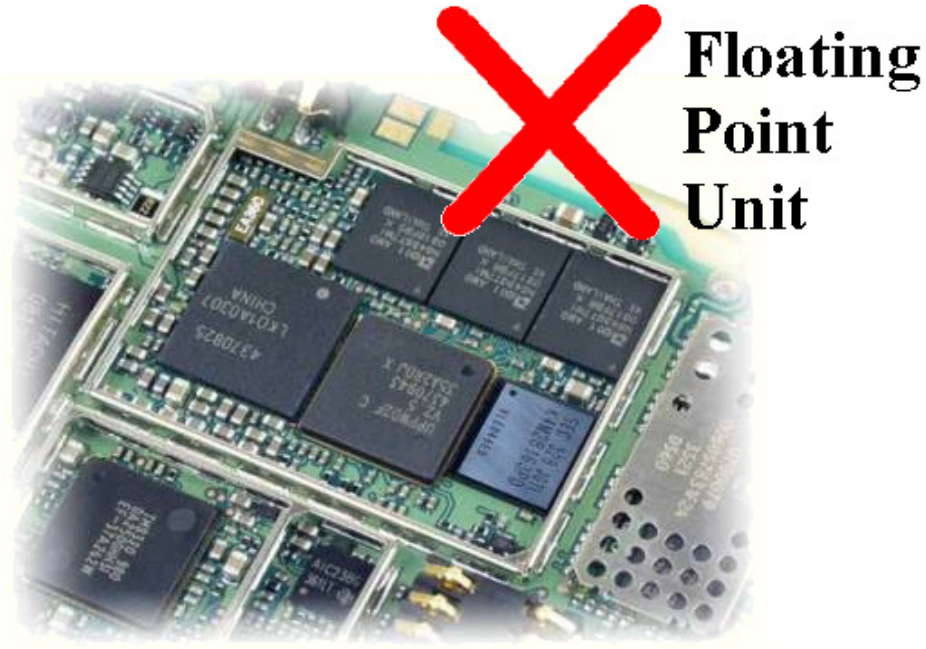



*Fig.7 Nokia mobile phone circuit board. It got no floating point unit.*

## 2.4 Overview of our Focus Library

In the following section, we would discuss about library of image processing, image decoding and encoding, controlling onboard camera and faxing.

To process image data, we first need to access the raw data of an image. That means it is essential to access the RGB or YUV value of a specific coordinate. In Symbian OS, there is an object called CFbsBitmap, which enable us to edit the image.

The image is actually stored as some format like JPEG, GIF, TIFF, etc. To get the image, the image files should be first decoded to raw data first. To store the image, the image should be encoded first. In Symbian OS, decoding can be done using object called CImageDecoder and encoding can be done using object called CImageEncoder.

As we should get the picture of the document, we should know to use the onboard camera on the Symbian phone to take photo. In the Symbian OS, an object called CCamera is used to control the object. In addition, we should implement a MCameraObserver-derived object to receive call back of the camera.

Faxing is a popular transmission method for document. As a result, we want to implement a faxing on the Symbian phone. A related object is CFaxTransfer. It is used to send and retrieve the fax. Before sending fax, a fax file should be first created using a object called CWriteFaxFile.

### 2.4.1 Image processing

CFbsBitmap is class provided by Symbian C++ API. Using this class, we can get the pixel value and set the pixel value.

Firstly, we should get the CFbsBittmap object, this object can be got easily from the output of the image decoding and encoding and the camera. Besides, we need to create a empty image with specific size. This can be done by creating an object using the constructor:

```
CFbsBitmap*bitmap = new CFbsBitmap();
```

Then create the bitmap with the size and display mode using the following member function:

```
TInt CFbsBitmap::Create(const TSize& aSizeInPixels, TDisplayMode
aDispMode);
```

After we have successfully create a bitmap, we get the pixel value using the following member function:

```
Void CFbsBitmap::GetPixel(TRgb& aColor, const TPoint& aPixel) const;
```

And we can set the pixel value by modify the object TRgb.

However, accessing value using this member function would involve context switching. So the overhead would be very large as GetPixel needed to be called for each pixel. As a result, we should use other method to access the pixel value.

Instead of the above method, we access the pixel value directly using the pointer. The follow member function would return the starting address of the image data:

```
TUint32* CFbsBitmap::DataAddress() const;
```

After we get the address, we can modify the value directly. Thus it would be faster.

### 2.4.2 Image Decoding and Encoding

Symbian provide object called CImageDecoder. It can be used to decode the file. Firstly, we should create the object using:

```
static CImageDecoder* CImageDecoder::FileNewL(RFs& aFs, const
TDesC& aSourceFilename, const TDesC8& aMIMEType, const TOptions
aOptions=EOptionNone);
```

We need to specify which file should be decoded and which image format is used. Then we can use the following function to do the decoding:

```
void CImageDecoder::Convert(TRequestStatus* aRequestStatus,
CFbsBitmap& aDestination, TInt aFrameNumber = 0);
```

After that, we obtain a CFbsBitmap, which is the raw data of the image from the function.

Symbian provide object called CImageEncoder. It can be used to encode the raw image to a file. Firstly, we should create the object using:

```
static CImageEncoder* CImageEncoder::FileNewL(RFs& aFs, const
TDesC& aDestinationFilename, const TDesC8& aMIMEType, const
TOptions aOptions = EOptionNone);
```

Then we can use the following member function to do the encoding:

```
void CImageEncoder::Convert(TRequestStatus* aRequestStatus, const
CFbsBitmap& aSource, const CFrameImageData*
aFrameImageData=NULL);
```

**Image Decoding**

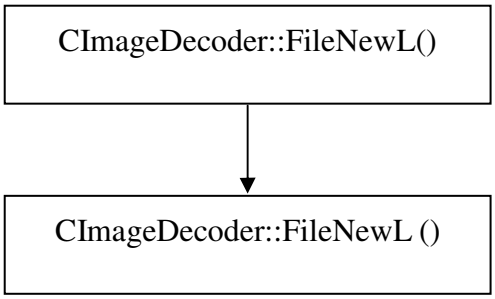



**Image Encoding**

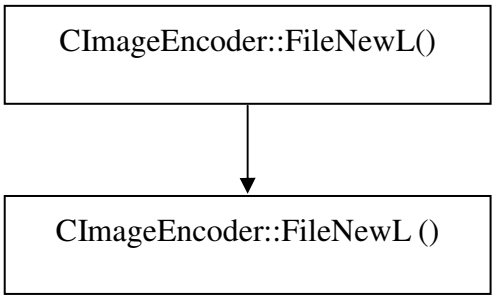



*Fig.8 Objects of Image decoding & encoding*

### 2.4.3 Controlling onboard camera

In the project, we should be able to use the Camera to capture images. Firstly, we should use

**static CCamera* CCamera::NewL(MCameraObserver& aObserver, TInt aCameraIndex);**

to get a CCamera object. We should also supply an MCameraObserver-derived object to receive callbacks.

Then we should the two following functions reserve and power on the camera:

**void CCamera::Reserve();**

**void Camera::PowerOn();**

When the power on is completed, it is informed by

**void MCameraObserver::PowerOnComplete()**

Next, we can specify the required image format and start the view finder.

Before we capture the image, we need to set image format, size:

**void CCamera::PrepareImageCaptureL(TFormat aImageFormat, TInt aSizeIndex);**

Then we can use the following function to capture a image:

**void CCamera::CaptureImage();**

The image date would be return in the

**void MCameraObserver::ImageReady(CFbsBitmap* aBitmap, HBufC8* aData, TInt aError);**

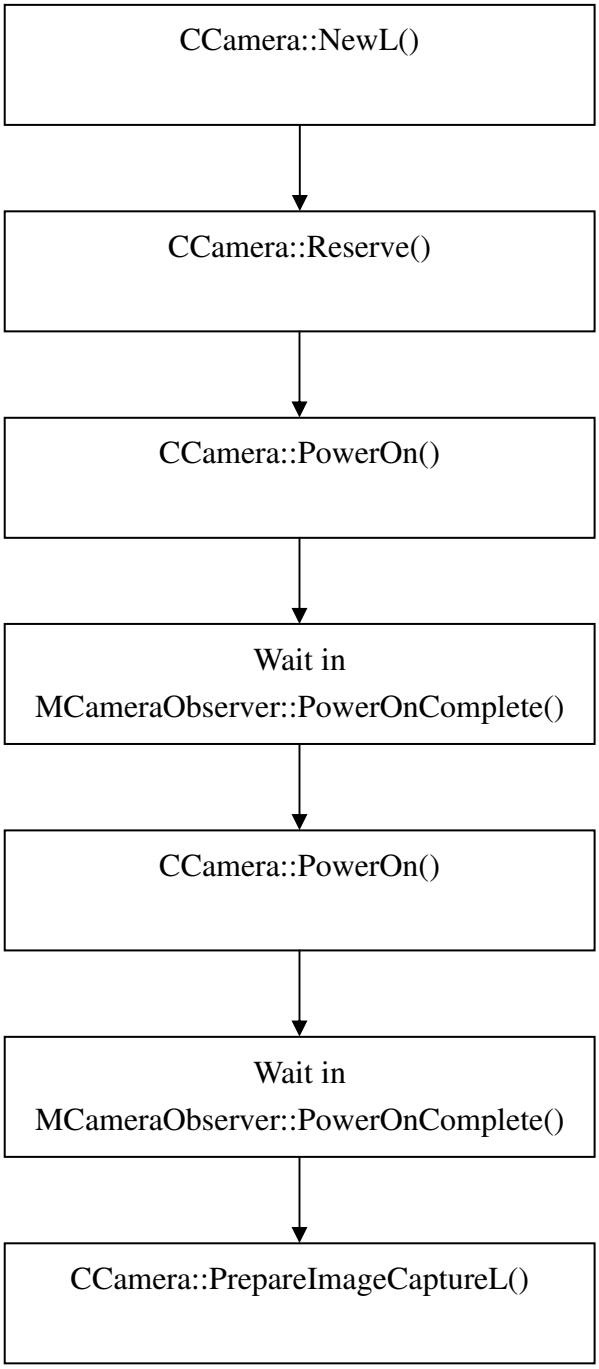



*Fig.9 Flow of controlling onboard camera*

### 2.4.4 Faxing

In Symbian OS, there are two methods (API) to send fax originally. They are CFaxMtmClient object and CFaxTransfer object. However, CFaxMtmClient has been deleted in Symbina OS v8.0 or later. As a result, we would only introduce CFaxTransfer object below:

Firstly, we need to create the CFaxTransfer by the following class function:

```
static CFaxTransfer* CFaxTransfer::NewL(const TFaxSettings&
aFaxSettings);
```

Then we need to set the mode of the operating during the fax session and the phone number by:

```
void CFaxTransfer::SetMode(TFaxMode aMode);
void CFaxTransfer::SetPhoneNumber(TDesC8& aNumber);
```

We also need to specify which fax file need to be included in the faxing by using the following member function:

```
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore);
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore,
TInt aStartPage);
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore,
TInt aStartPage, TInt aEndPage);
```

After all these thing has be set, we can start faxing by the member function:

```
TInt CFaxTransfer::Start(TRequestStatus& aThreadStat);
```

Each starting to fax must be paired with the following member function:

```
void CFaxTransfer::Stop();
```

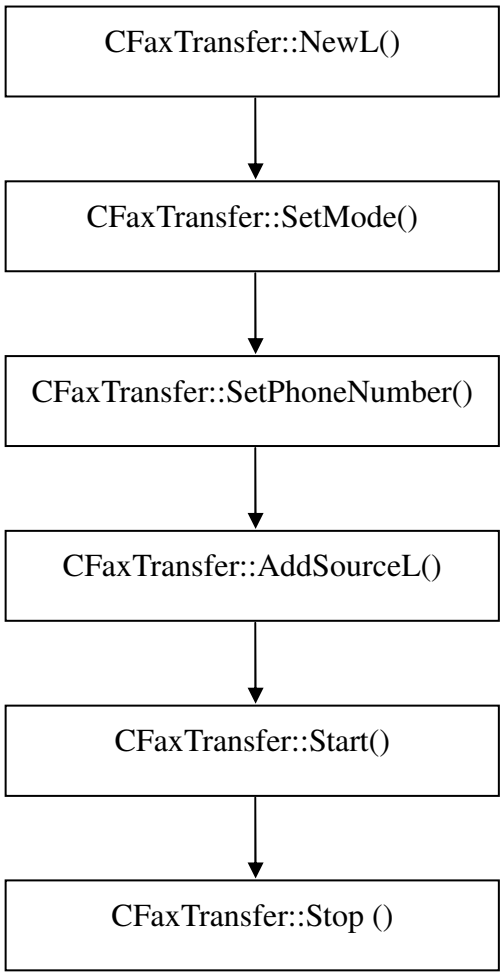



*Fig.10 Flow of Faxing*

Although we knew the basic procedure of how to send a fax, we still cannot implement this function properly on the Symbian phone.

## 2.5 Conclusion

This chapter introduces the overview of Symbian OS and describes the limition of Symbina phone. In addition, we have successful implement program to use Symbian C++ API to take photo using onboard camera, decode and encode image, access raw data of image. However, we still be fail to implement program to send fax properly.

# Chapter 3 Testing on Windows

## 3.1 Cygwin Platform & Mono

### 3.1.1 Cygwin Platform

Cygwin can simulate Linux environment on Windows. It consists of two main parts:

- A DLL (cygwin1.dll), which can acts as a Linux API emulation layer. The layer can provide substantial Linux API functionality. The Cygwin DLL can work with any x86-32bit versions of Windows. However, using Cygwin cannot run linux applications on Windows.
- A collection of tools, which can provide the appearance of Linux User.

### 3.1.2 Mono

Mono in Spanish means 'monkey'. The Mono Project is an open development initiative sponsored by Novell. It develops an open source, which is UNIX version of the Microsoft .NET development platform. The objective is to enable .NET Application be built by UNIX developers. The project implements many technologies developed by Microsoft.

Mono provides a number of components for building software:

- A Common Language Infrastructure virtual machine. It includes a class loader, just-in-time compiler and a garbage collecting runtime.
- A class library. The library can work with any programming language which works on the Common Language Infrastructure. It includes both .NET compatible class libraries and Mono-provided class librarie.
- A compiler for the C# programming language.

Windows has compilers, which target the virtual machine from a large number of programming language. The languages include Managed C++, Java Script, Eiffel, Component Pascal, APL, Cobol, Perl, Python, Scheme, Smalltalk, Standard ML, Haskell, Mercury and Oberon.

The Common Language Infrastructure and the Common Type System allows application programs and libraries to be written in a collection of different supported programming languages, which target the byte code. For example, if there is a class to do maxtrix computation in C#. That class can be reused using any other supported language. All the supported languages can share a single object system, threading system, class libraries, and garbage collection system.

### 3.1.3 Advantages of Cygwin Platform & Mono

1. Cygwin Platform provides a Linux-like development platform. As we had already been familiar with these type of platform, we can develop our program more convenient without learning a new development platform.

2. Mono provides platform for us to develop .NET program. As a result, combined with Cygwin and Mono, we can develop .NET program with learning Visual Studio .Net. It would enable us to develop our program faster.

3. The sample code of SIFT is developed by Mono. As a result, we can directly make use of the sample code without facing other technical problems.

## 3.2 C# programming

### 3.2.1 C# programming language

C# is an object-oriented programming language developed by Microsoft as part of their .NET Framework. C# is base on C++ and Java. As a result, the syntax of C# is similar to both Java and C++. The feature can enable C++ and Java programmer to come up with C# with less difficulties.

All .NET program runs on .NET Framewrok. C# is the programming language which can mostly reflects the Framework and it depends strongly on this framework. There is no unmanaged C# programs. The datatypes of C# are objects of the corresponding .NET types. C# show many key features of .NET runtime. For example, it is garbage-collected. The classes , interfaces, delegates, exception of C# can show the features.

When C# is compared to C and C++, It is restricted or enhanced in the following ways:

- Unlike C and C++, Raw pointers can only be used in an unsafe mode. Most object is accessed through safe references, which cannot be invalid. Most arithmetic operations are checked for overflow. Pointers can only be pointed to value types
- Objects cannot be explicitly freed. When no more references to them exist , they are garbage collected.
- C# only alow single inheritance. However, a class can implement any number of abstract interfaces. It can simplify the implementation of runtime.
- C# is more typesafe than C++. There is only one default implicit conversion called safe conversions. It includes widening of integers and

conversion from derived types to base types. There are no implicit conversions between booleans and integers as well as enumeration members and integers. There are also no void pointers. Any user-defined implicit conversion must be explicitly marked.

- Different syntax for array declaration. In C#, it is 'int[] a = new int[10]' where as it is "int a[10]" in C or C++).
- Enumeration members are placed in their own namespace.
- There is no template in C#

### 3.2.2 Advantages of C# programming

1. C# is a object-oriented program which is similar to C++ and C++ is the native programming language to develop program on Symbian platform.

2. C# is more convenient to write than C++. We need no to touch too much pointer variable which would be very troublesome sometimes.

3. The sample code of Scale Invariant Feature Transform (SIFT) is written in C#. In order to make use of the sample code, it is more convenient to use C# to write the programs.

## 3.3 Matlab programming

### 3.3.1 Matlab programming language

Matlab is a high-level technical computing language. It provides interactive environment for algorithm development, data visualization, data analysis and numeric computation. Solving technical computing problems using Matlab could be faster than using traditional programming languages, like C, C++, etc.

There are five main features of Matlab:

- Desktop Tools and Dvelopment Environment: It provides set of tools and facilities to help user to get familiar with Matlab. Most of them has graphic user interfaces. So it is very user-friendly. The tools include the Matlab desktop,command window, command history, editor and debugger, and browsers for viewing help, the workspace, files. The interface can be changed as the user like.
- The Matlab Mathematical Function Library: The library provide many elemental functions, like sum, sine, cosine, complex arithmetic, etc. Besides, there are also other complex functions like matrix inverse, matrix eigenvalues, Bessel functions, fast Fourier transforms, etc.
- The Matlab Language: Matlab is a language which support many matrix and array operations. It also have control flow statements, functions, data structures, input/output, and object-oriented programming features. Matlab program can be used only for testing algorithm as well as developing application programs.
- Graphics: Matlab provides extensive facilities for displaying vectors and matrices as graphs which can be annotated and printed. Matlab provides high-level functions for two-dimensional and three-dimensional data

visualization, image processing, animation, and presentation graphics. In addition, there are also low-level functions that support customizing the appearance of graphics and building graphical user interfaces on Matlab applications.

- The Matlab External Interfaces/API: Using this libray, C and Fortran programs can interact with Matlab. It includes facilities for calling routines in C or Fortran from Matlab, considering Matlab as a computational engine, and for reading and writing of MAT files

Matlab can be used in variety of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. There are also Add-on toolboxes separately which can extend the Matlab environment to solve speicific classes of problems

### 3.3.2 Advantages of Matlab programming

1. It provides many matrix operations. Matrix operations are essential in image processing. It enable us to develop image processing program easier. It provides a convenient development platform so that we can test our algorithm faster.

2. There are many functions which are very convenient for us. For example, we just show a picuture using a function. It help to avoid many technical problems because in C or C++, we need to write GUI program to suit our use. As a result, Matlab save us a lot of time to develop our programs.

# Chapter 4 Concept of Image Alignment

## 4.1 Introduction

The following concept mainly comes from [1].

Image Alignment algorithm is also called image registration algorithm. It is used to find the correspondence relationships among different images with different degrees of overlap. Image stitching algorithm would use the alignment parameters generated by an image alignment algorithm to blend the images so that the images can be merged in seamless manner.

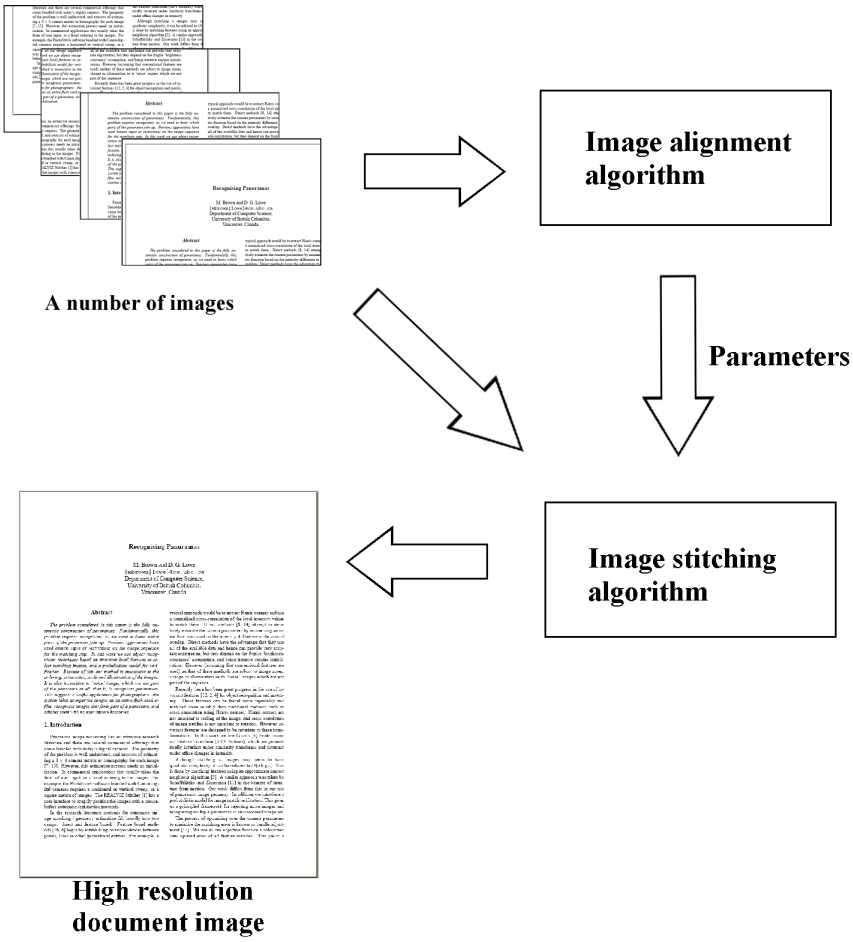



*Fig.11 Overview of image alignment and stitching*

Generally, there are two different approaches for image alignment algorithm. They are direct alignment and feature-based alignment.

## 4.2 Direct alignment

Direct alignment is an approach that uses pixel-to-pixel matching. In direct method, there are several error metric used to compare the image. Then the error metric would be combined with search technique to do the searching. For example, we can try all possible alignments. However, full search would be too slow. So there would be some techniques to enhance the efficiency. For example, hierarchical coarse-to-fine techniques and Fourier transforms can used to speed up the computation. Incremental methods would be used to get sub-pixel precision in the alignment. Parametric motion models can be also applied. More detail of all these things would be mentions below.

### 4.2.1 Error metrics

One simple way is to shift one image relative to the other. Then several types of Error metrics can be constructed. Some of variables need to be defined first.

$x, y$ is pixel coordinates

$u, v$ is displacement coordinates

$I_o(x, y)$ is template image sampled at $(x, y)$

$I_1(x, y)$ is another image sampled at $(x, y)$

One of the methods is to find the minimum of the sum of squared differences (SSD) function.

$$E_{SSD}(u,v) = \sum_i \left[ I_1(x_i + u, y_i + v) - I_0(x_i, y_i) \right]^2$$

Another way is to find the minimum of the absolute value of differences (SAD) function.

$$E_{SAD}(u,v) = \sum_i \left| I_1(x_i + u, y_i + v) - I_0(x_i, y_i) \right|$$

Instead of finding the minimum of the differences, we can perform correlation to intensity differences. In the correlation, we need to maximize the product of the two aligned image.

$$E_{CC}(u,v) = \sum_i I_0(x_i, y_i) I_1(x_i + u, y_i + v)$$

## 4.2.2 Hierarchical motion estimation

In the hierarchical motion estimation, an image pyramid is first constructed. Then search would be performed on the coarser levels first. Then search would be performed on the finer levels. But the search is only performed on the area, which is satisfied by the search on the coarser levels. So most of the candidates would be filtered in the coarser levels. The amount of search on the finer levels would be greatly reduced. As a result, the algorithm would be more efficient.

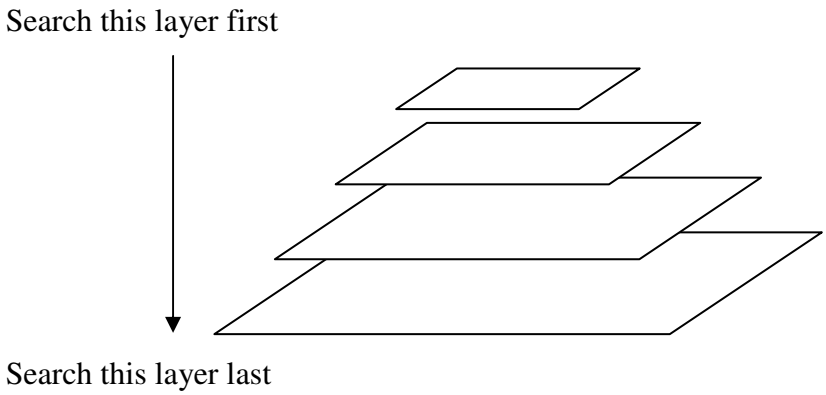



*Fig.12 Hierarchical motion estimation*

### 4.2.3 Fourier-based alignment

The hierarchical motion estimation approach may not work well when the search range is significant fraction of a large image. It is because the coarser level of the image pyramid would be blurred too much. Then Fourier-based alignment would work better in this situation.

Using Fourier transform, the cross-correlation function $E_{cc}$ can written as,

$$\Im\{E_{CC}(u,v)\} = \Im\{\sum_i I_0(x_i, y_i) I_1(x_i + u, y_i + v)\}$$
$$= \Im\{I_0(u,v) \bar{*} I_1(u,v)\} = \Im(I_0(u,v))\Im^*(I_1(u,v))$$

As a result, $E_{cc}$ can be computed in the following way. Firstly, both image $I_0(x,y)$ and $I_1(x,y)$ need to be taken Fourier transform. Then multiply them together with the second taken conjugated. At last, $E_{cc}$ can be obtained by take inverse transform of the product. As there exists Fast Fourier Transform algorithm, it is more efficient to compute the result than the usual one.

The $E_{SSD}$ can also be computed in similar way.

$$\Im\{E_{SSD}(u,v)\} = \Im\{\sum_i [I1(xi + u, yi + v) - I0(xi, yi)]^2\}$$
$$= \Im\{\delta(u,v)\}\sum_i [I_0{}^2(x_i, y_i) + I_1{}^2(x_i, y_i)] - 2\Im(I_0(u,v))\Im^*(I_1(u,v))$$

### 4.2.4 Incremental refinement

All the techniques mentioned above can only estimate translational alignment to the nearest pixel. However, the accuracy is not enough.

To obtain better result, sub-pixel estimates can be used. Sveral discrete values of (u, v) around the best value found is evaluated. Then the matching score is interpolated to find an analytic minimum. There is a more common approach

proposed by Lucas and Kanade. It uses a Taylor Series expansion of the image function to do gradient descent on the sum of squared difference SSD energy function.

$$
\begin{aligned}
& E_{LK-SSD}(u+\Delta u, v+\Delta v) \\
&= \sum_i [I_1(x_i+u+\Delta u, y_i+v+\Delta v) - I_0(x_i, y_i)]^2 \\
&\approx \sum_i [I_1(x_i+u, y_i+v) + J_1(x_i+u, y_i+v)(\Delta u, \Delta v) - I_0(x_i, y_i)]^2
\end{aligned}
$$

where

$$
\begin{aligned}
& J_1(x_i+u, y_i+v) \\
&= \nabla I_i(x_i+u, y_i+v) \\
&= (\frac{\partial I_i}{\partial x}, \frac{\partial I_i}{\partial y})(x_i+u, y_i+v)
\end{aligned}
$$

It is the image gradient at ($x_i$+$u$, $y_i$+$v$).

The above least squares problem can be minimizing by solving the following equations.

$$\mathbf{A}(\Delta u, \Delta v) = \mathbf{b}$$

where

$$A = \sum_i J_1^T(x_i+u, y_i+v) J_1(x_i+u, y_i+v)$$

$$b = -\sum_i (I_1(x_i+u, y_i+v) - I_0(x_i+u, y_i+v)) J_1^T(x_i+u, y_i+v)$$

## 4.2.5 Parametric motion

Many image alignment job need to use more complex motion models. These models would have more parameters than pure translation. As a result, full is not practical. The above algorithm can be generalized to parametric motions models combined with hierarchical search algorithm.

## 4.3 Feature-Based Registration

Beside direct alignment, there is other method. It is feature-based registration. In this kind of alignment, the first step is to extract distinctive features from images. Secondly, the features extracted need to matched in the different images. After that,

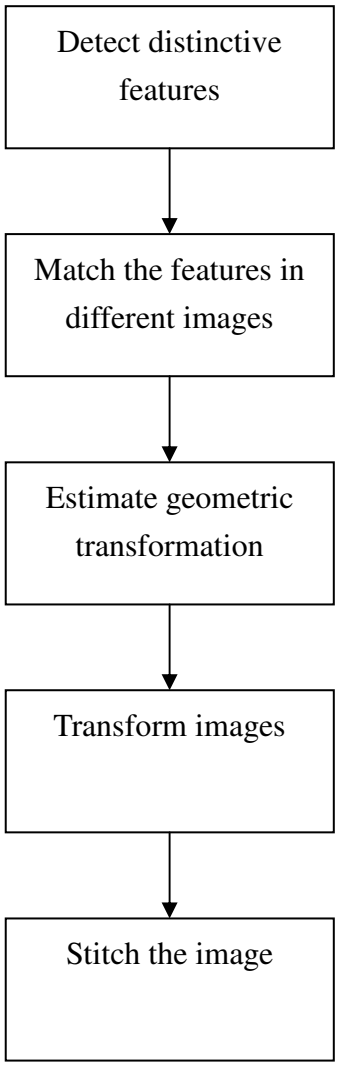



*Fig.13 Flow diagram of feature-based registration*

we can estimate geometric transformation for the images. At last, the images need to be transformed and stitch together.

### 4.3.1 Interest point detectors

To do feature-based registration, detecting interest point is the first step. The interest point should have the some properties. They must be distinguished in many ways so that they can correctly match in the images. The interest points are expected to be invariant to many things so that the interest points can be matched in different images no matter how the image is transformed or noise is added. They are called stable points. As a result, the interest point should be invariant to the following properties:

1. Scales
2. Rotations
3. Change in illumination
4. 3D camera viewpoint
5. Noise
6. Occlusion
7. Clutter

Nowadays, there are several techniques that can detect such interest points. They include Harris points, Harris-Laplace points, DoG points, Harris-Affine points, etc.

### 4.3.2 Feature matching

After the feature points are detected, they must be matched. It is to find similar feature from different image. After that, we can estimate the geometric transform.

Before the matching, each feature point should be stored in the descriptor. The descriptors should also be invariant to many properties like the feature point itself.

Then the actual match can be done. One way to match feature point is to compare all features in one image against them in the other image. However, it is not efficient. Instead, some indexing schemes can be applied to speed up the matching. Most of these are based on the concept of find nearest neighbours. To support these types of match, k-d trees data structure can be used. Furthermore, Best-Bin-First algorithm, locality-sensitive hashing can be used to speed up the matching.

## 4.4 Conclusion

We have studied two major kinds of image alignment. And in each kind of image alignment, there are many methods to do it. So far, we have general understanding of these methods. Between these two major methods, we choose to use feature-based registration. In the next chapter, we would describe what method we use to extract the features. It is Scale Invariant Feature Transform.

# Chapter 5 Implementation in Our Project

## 5.1 Introduction

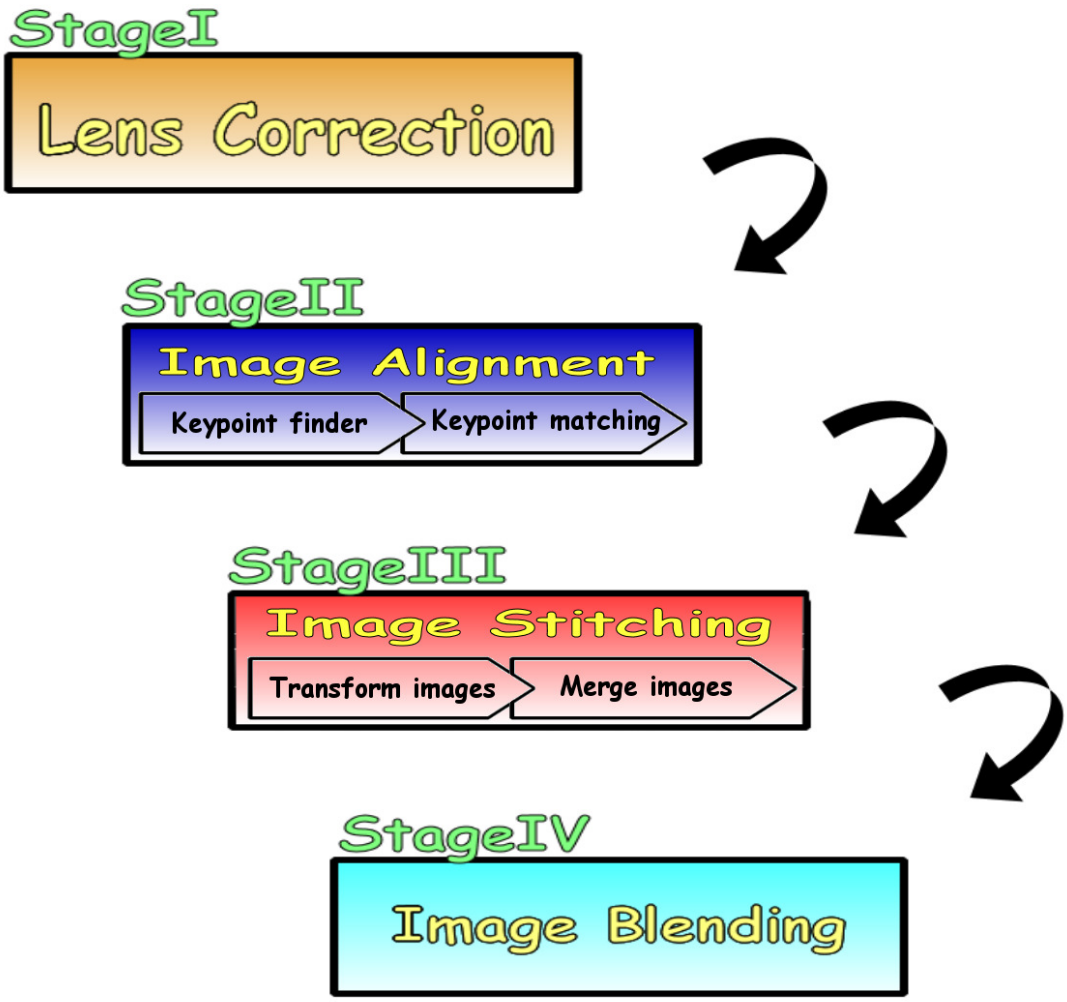



*Fig.14 Flow diagram of our implementation.*

Our implementation of document image reconstruction consist of 4 stages:

*Stage I*: **Lens Correction**

*Stage II*: **Image Alignment (Keypoint finder & Keypoint matching)**

*Stage III*: **Image Stitching (Transform images & Merge images)**

*Stage IV*: **Image Blending**

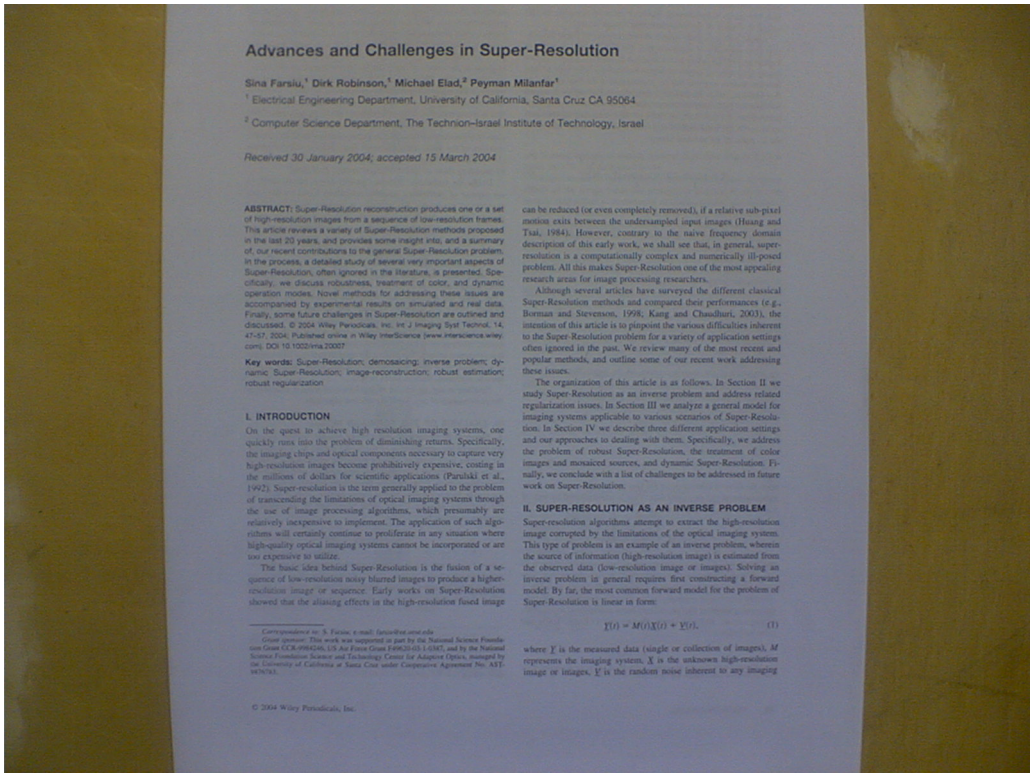
In the following, we will go deep into the designs of each stage.

# 5.2 Stage I. Lens Correction

## 5.2.1 Introduction

After taking photos, we find that all the photos suffered from lens distortion (Fig. 15). It can be obviously seen the bending at the sides of the document.

To overcome the problems of lens distortion, we will first perform lens correction before image alignment. This technique is used to ensure the accuracy of matching the sub-images and have correct transform parameters.



Advances and Challenges in Super-Resolution

Sina Farsiu,[1] Dirk Robinson,[1] Michael Elad,[2] Peyman Milanfar[1]

[1] Electrical Engineering Department, University of California, Santa Cruz CA 95064

[2] Computer Science Department, The Technion–Israel Institute of Technology, Israel

*Received 30 January 2004; accepted 15 March 2004*

**ABSTRACT:** Super-Resolution reconstruction produces one or a set of high-resolution images from a sequence of low-resolution frames. This article reviews a variety of Super-Resolution methods proposed in the last 20 years, and provides some insight into, and a summary of, our recent contributions to the general Super-Resolution problem. In the process, a detailed study of several very important aspects of Super-Resolution, often ignored in the literature, is presented. Specifically, we discuss robustness, treatment of color, and dynamic operation modes. Novel methods for addressing these issues are accompanied by experimental results on simulated and real data. Finally, some future challenges in Super-Resolution are outlined and discussed. © 2004 Wiley Periodicals, Inc. Int J Imaging Syst Technol, 14, 47–57, 2004; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/ima.20007

**Key words:** Super-Resolution; demosaicing; inverse problem; dynamic Super-Resolution; image-reconstruction; robust estimation; robust regularization

**I. INTRODUCTION**

On the quest to achieve high resolution imaging systems, one quickly runs into the problem of diminishing returns. Specifically, the imaging chips and optical components necessary to capture very high-resolution images become prohibitively expensive, costing in the millions of dollars for scientific applications (Parulski et al., 1992). Super-resolution is the term generally applied to the problem of transcending the limitations of optical imaging systems through the use of image processing algorithms, which presumably are relatively inexpensive to implement. The application of such algorithms will certainly continue to proliferate in any situation where high-quality optical imaging systems cannot be incorporated or are too expensive to utilize.

The basic idea behind Super-Resolution is the fusion of a sequence of low-resolution noisy blurred images to produce a higher-resolution image or sequence. Early works on Super-Resolution showed that the aliasing effects in the high-resolution fused image can be reduced (or even completely removed), if a relative sub-pixel motion exits between the undersampled input images (Huang and Tsai, 1984). However, contrary to the naive frequency domain description of this early work, we shall see that, in general, super-resolution is a computationally complex and numerically ill-posed problem. All this makes Super-Resolution one of the most appealing research areas for image processing researchers.

Although several articles have surveyed the different classical Super-Resolution methods and compared their performances (e.g., Borman and Stevenson, 1998; Kang and Chaudhuri, 2003), the intention of this article is to pinpoint the various difficulties inherent to the Super-Resolution problem for a variety of application settings often ignored in the past. We review many of the most recent and popular methods, and outline some of our recent work addressing these issues.

The organization of this article is as follows. In Section II we study Super-Resolution as an inverse problem and address related regularization issues. In Section III we analyze a general model for imaging systems applicable to various scenarios of Super-Resolution. In Section IV we describe three different application settings and our approaches to dealing with them. Specifically, we address the problem of robust Super-Resolution, the treatment of color images and mosaiced sources, and dynamic Super-Resolution. Finally, we conclude with a list of challenges to be addressed in future work on Super-Resolution.

**II. SUPER-RESOLUTION AS AN INVERSE PROBLEM**

Super-resolution algorithms attempt to extract the high-resolution image corrupted by the limitations of the optical imaging system. This type of problem is an example of an inverse problem, wherein the source of information (high-resolution image) is estimated from the observed data (low-resolution image or images). Solving an inverse problem in general requires first constructing a forward model. By far, the most common forward model for the problem of Super-Resolution is linear in form:

$$\underline{Y}(i) = M(i)\underline{X}(i) + \underline{V}(i), \qquad (1)$$

where $\underline{Y}$ is the measured data (single or collection of images), $M$ represents the imaging system, $\underline{X}$ is the unknown high-resolution image or images, $\underline{V}$ is the random noise inherent to any imaging

*Fig.15 Photo taken using Nokia N90 of resolution 1600x1200.*

## 5.2.2 Basic concepts

### 5.2.2.1 Lens distortions

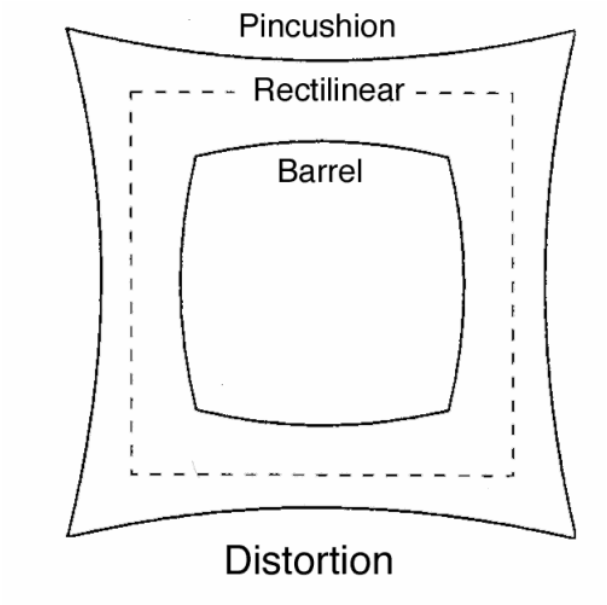



*Fig.16 Common types of lens distortion*

Lens design has been important for photography since its beginnings. Fortunately lenses had been designed for optical instruments such as telescopes and microscopes for a great many years before the invention of photography.

Ideal lens would give even illumination over the whole image, and excellent image sharpness for all colors of light across the whole frame, and it would have good drawing (if designed as a rectilinear lens would represent any straight line in reality by an equally straight line in the image). But there is no such thing as a perfect lens, all lens have defects (or sometimes called aberrations).

Distortion is the most easily recognized aberration as it deforms the image as whole. Since straight lines in the object space are rendered as curved lines on the film, the name curvilinear distortion is frequently encountered. There are two fundamental manifestations of the aberration, barrel and pincushion distortion. Straight lines in the undistorted subject (rectilinear) bulge in the characteristic barrel

fashion or bend inward in the pincushion representation. Straight lines running through the image center remain straight and a circle concentric with the image center remains a circle, although its radius is affected.

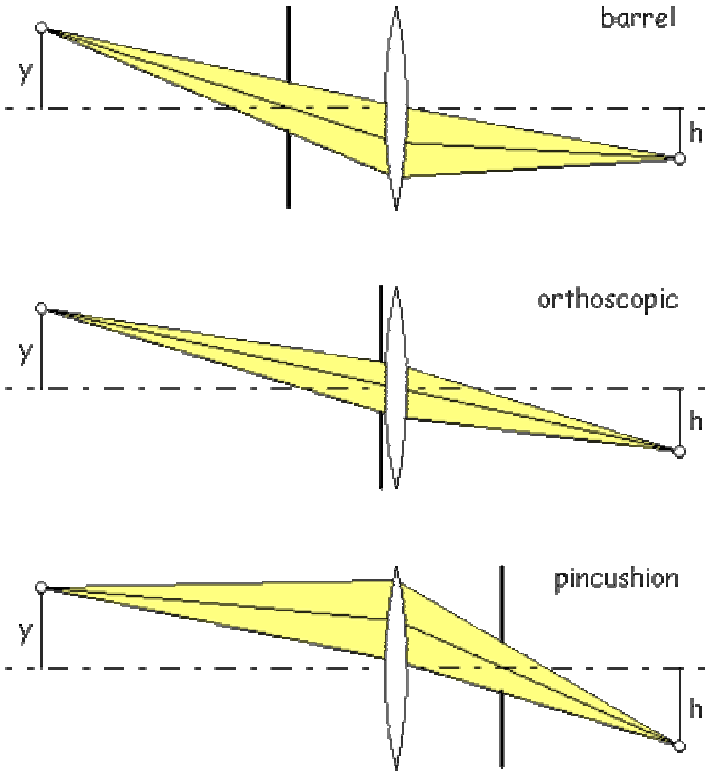



*Fig.17 Causes of different type of distortion*

A common cause of distortion is the introduction of a stop in a system of thin lenses, e.g. to reduce spherical aberration or astigmatism. The position of such a stop determines the amount and the sign of the distortion, as illustrated above.

The position of the image point is determined by the chief ray (solid line in the diagram), which is the ray that passes through the center of the stop. The chief ray is characteristic for the cone of light marked by the margins of the stop.

### 5.2.2.2 Image Transformation

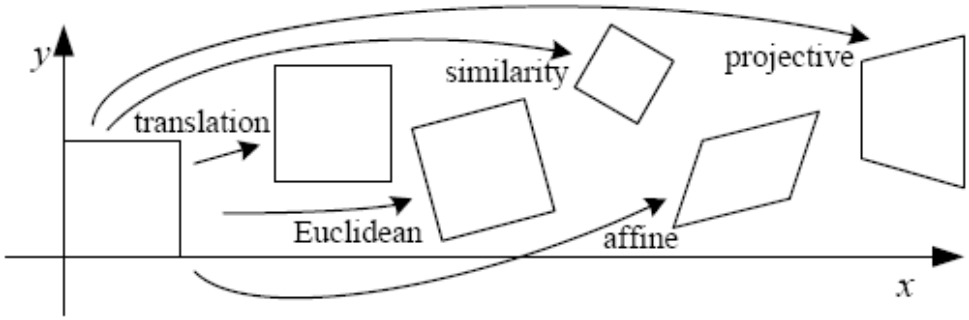



*Fig.18 Common types of image transformation*

The manipulation of an image in any graphics system is considered as an inherent feature of the system. Both vector- and raster-based system provide transformation, but with differing effect. Transformation is applying an operator to a graphics entity to produce a different entity. Here we introduce some of the common image transformations:

(a) **Translation** - A common requirement in the graphics application is to move a picture to a new position, as in figure st-1a. This is achieved by means of a translation or shift transformation. In order to translate a point, constant values are added to the x and y coordinates of the point. The new co-ordinates of the point (x',y') are given by:

$$x' = y + tx$$

$$y' = y + ty$$

(b) **Scaling** - A scaling transformation is used to change the size of an object. Scaling about the origin (0,0) is achieved by multiplying the co-ordinates of a point by x- and y-scale factors:

$$x' = (Sx)x$$

$$y' = (Sy)y$$

- If |Sx| and |Sy| > 1, the effect is increasing the size of an object.
- If |Sx| and |Sy| < 1, the size is reduced.

For a symmetric or uniform scaling transformation in which the x and y scale factors are the same (Sx = Sy), the object is expanded by the same amount in each axis direction. For asymmetric or non-uniform scaling transformations in which the x and y scale factors are not equal (Sx ≠ Sy), the object changes its size by different values in the x- and y-axis directions.

(c) **Rotation (Euclidean)** - Another common type of transformation is rotation, this is used to orientate objects. An object and a point are rotated by an angle α about the origin. A line joining the point with the origin makes an angle β with the x-axis and has length R. Hence,

$$x = R\cos\beta$$

$$y = R\sin\beta$$

After rotation the point has co-ordinates x' and y' with values

$$x' = R\cos(\alpha + \beta)$$

$$y' = R\sin(\alpha + \beta)$$

Expanding these formulae for cos(α+β) and sin(α+β), and rearranging the x' and y' is expressed as:

$$x' = R\cos\alpha\cos\beta - R\sin\alpha\sin\beta$$

$$y' = R\sin\alpha\cos\beta + R\sin\beta\cos\alpha$$

Finally substituting for Rcosβ and Rsinβ, the co-ordinate of new point (x', y') is simplified to:

$$x' = y \cos\alpha - x \sin\alpha$$

$$y' = x \sin\alpha + y \cos\alpha$$

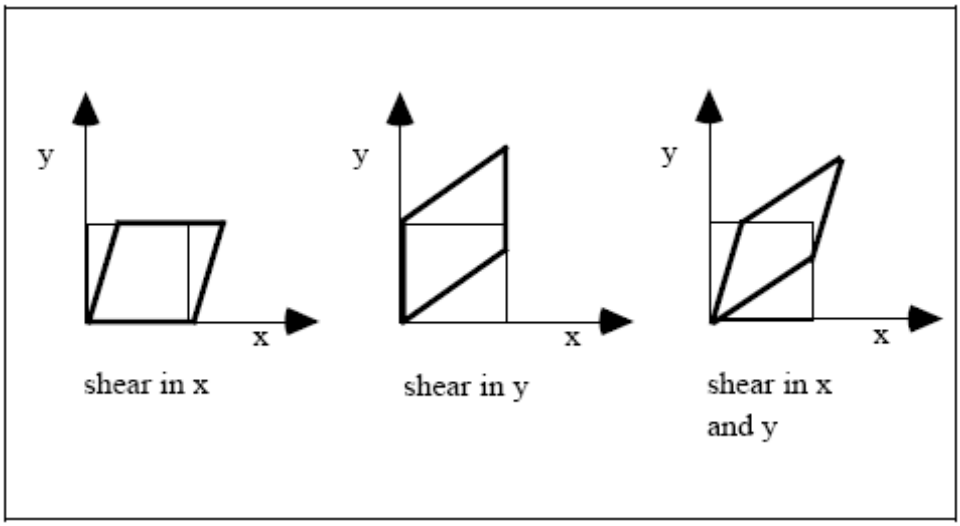



*Fig.19 Shear along different axis*

(d) **Shearing (Affine)** - A shear transformation has the effect of distorting the shape of an object. Figure above illustrates several different kinds of shear transformation applied to a unit square: a shear along x-axis (the x co-ordinates of points are displaced as a function of their height.), a shear in y (the y co-ordinates are displaced according to the x co-ordinate.), a shear in both x and y is shown. The new x and y co-ordinates of a point after shearing are given by:

$$x' = x + y.a$$

$$y' = x.b + y$$

If $a \neq 0$ then a shear along the x-axis is obtained and similarly if $b \neq 0$ then a shear in y-axis is obtained.

We can determine the transform parameters of a region by 4 pair of tiepoints (corners), then calculate the new coordinates of the pixels inside the region bounded by the 4 tiepoints. The diagram below illustrates such transformation:

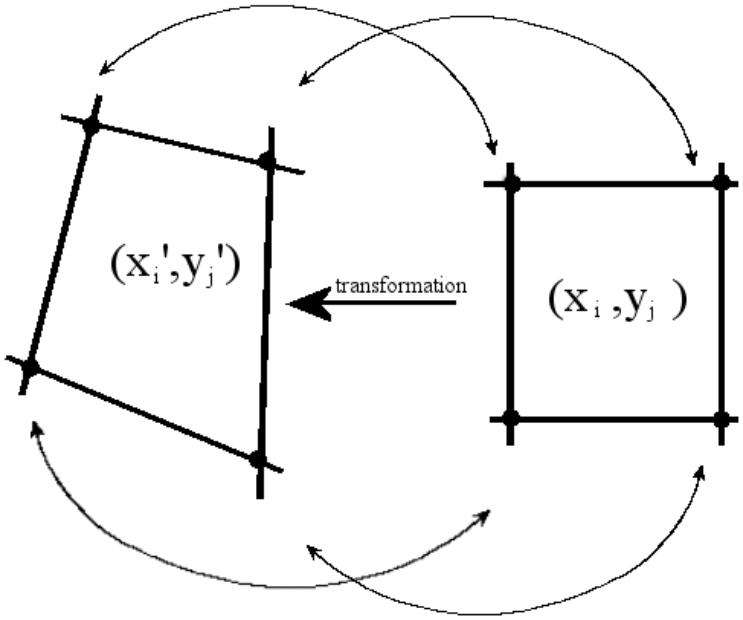



*Fig.20 Image transformation using 4 pairs of tiepoints*

$$\begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix} \begin{bmatrix} c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} = \begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix}$$

where,

$(x_i , y_j)$ are the original coordinates before transformation

$(x_i' , y_j')$ are the new coordinates after transformation

$c_i$ are the transform parameters

## 5.2.3 Implementation

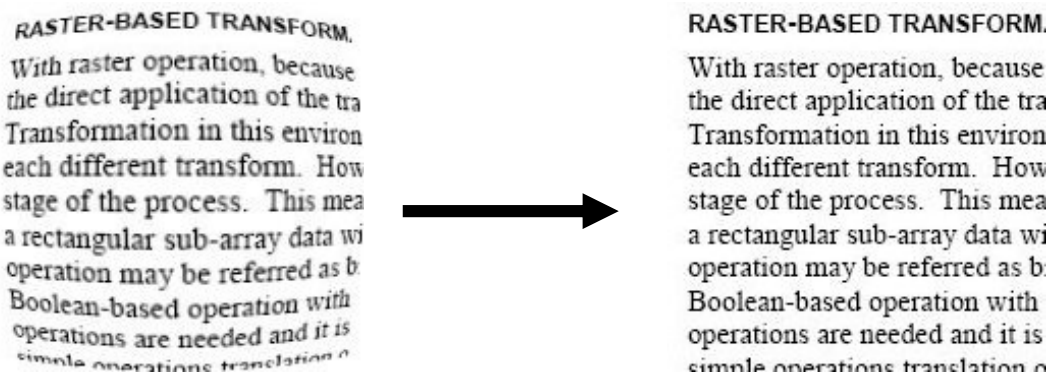



*Fig.21a Barrel distortion correction*

Lens correction is basically one specific type of image transformation, in which it is always with fixed correction factors for the same lens of same camera. At present in our project, the transform parameters of lens correction are fixed manually for lens on a particular mobile phone (N90 model). The values are obtained by trial and error which the transformed image is acceptable.

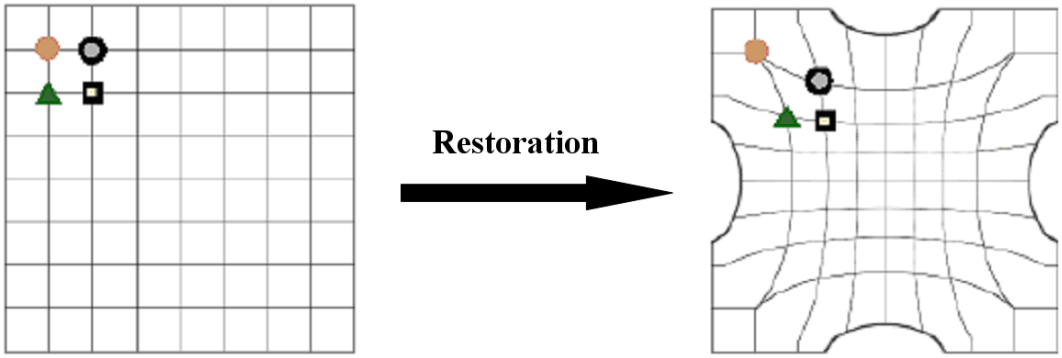



*Fig.21b Barrel distortion correction implementation*

The best way to perform lens correction is to divide the images into grids and transform the pixels inside that grid according to the parameter obtained at the 4 corners of the grid. Thus, smaller the grid is, better the restoration can be obtained.

However, we used another way to perform lens correction. We use the normalized coordinates and treat the center of photos as origin (0, 0). (Fig. ).

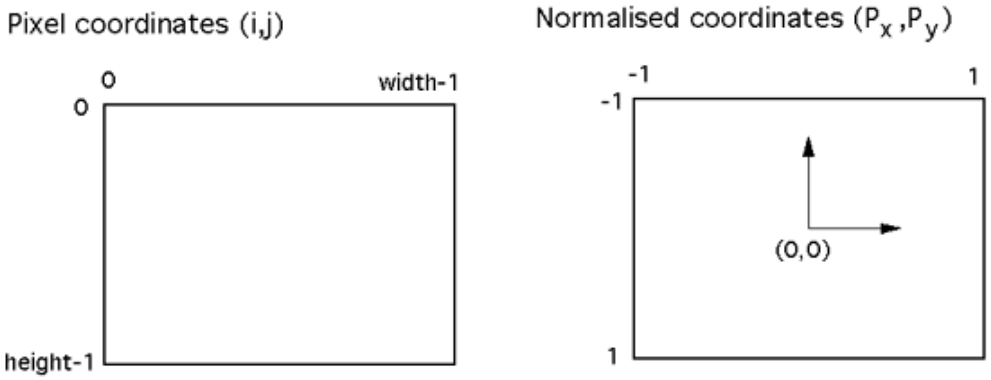



*Fig.22 Pixel coordinate system & normalized coordinate system*

Thus, further the pixel from the origin, longer the distance it moves towards the origin. And this is done by the following equations:

$$P'_x = P_x \ (1 - a_x \ \| P \|^2)$$

$$P'_y = P_y \ (1 - a_y \ \| P \|^2)$$

where

$(P_x', P_y')$ are coordinates at the corrected images

$(P_x, P_y)$ are coordinates at the original images

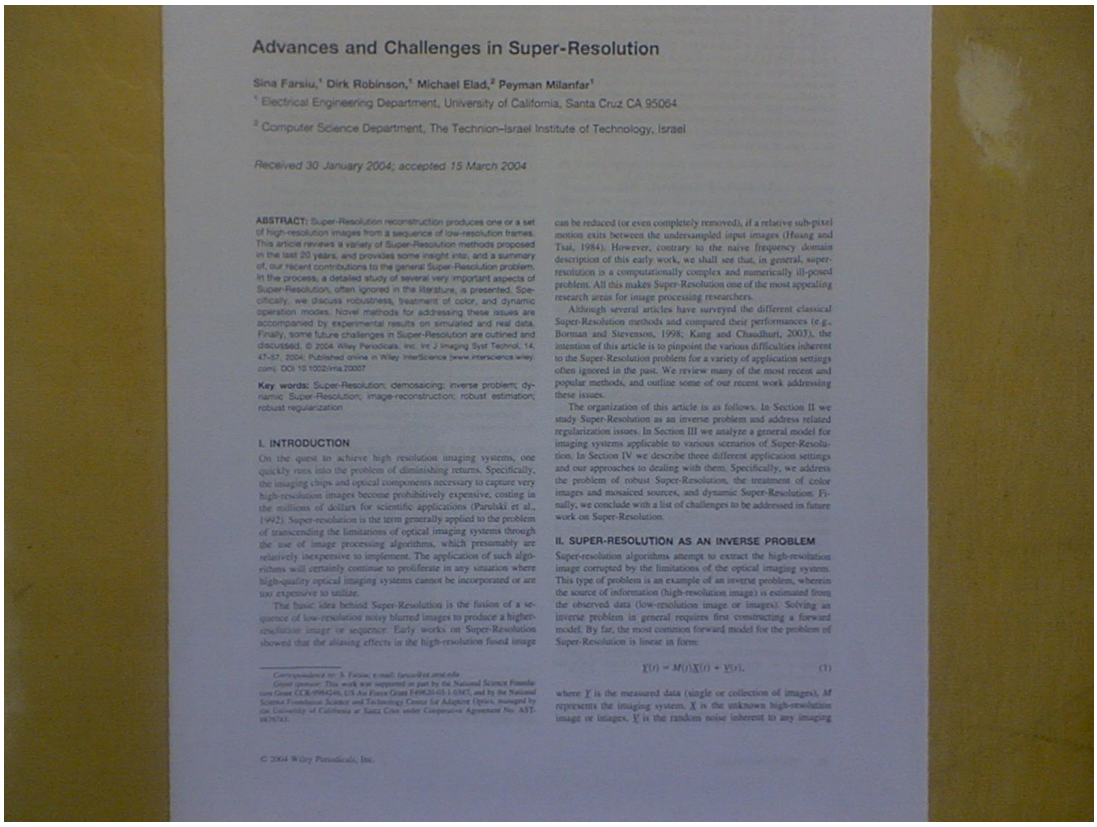
$| P |$ is distance of $(P_x, P_y)$ from origin

$a_x$, $a_y$ are correction factor along x-direction and y-direction respectively

We follow the equations and directly copy the pixel values (in RGB format) to the new coordinates. Since the target coordinates $(P_x', P_y')$ calculated from the equations may not be an integer, we will copy the pixel value to its nearest integral coordinate.

## 5.2.4 Conclusion

After the lens correction, all the sub-images are corrected with the same set of correction factors. All the sub-images look more rectangular (Fig. 23), but there are still rooms for improvement as the correction factors should be slightly different for different images.



Advances and Challenges in Super-Resolution

Sina Farsiu,[1] Dirk Robinson,[1] Michael Elad,[2] Peyman Milanfar[1]
[1] Electrical Engineering Department, University of California, Santa Cruz CA 95064
[2] Computer Science Department, The Technion–Israel Institute of Technology, Israel

Received 30 January 2004; accepted 15 March 2004

**ABSTRACT:** Super-Resolution reconstruction produces one or a set of high-resolution images from a sequence of low-resolution frames. This article reviews a variety of Super-Resolution methods proposed in the last 20 years, and provides some insight into, and a summary of, our recent contributions to the general Super-Resolution problem. In the process, a detailed study of several very important aspects of Super-Resolution, often ignored in the literature, is presented. Specifically, we discuss robustness, treatment of color, and dynamic operation modes. Novel methods for addressing these issues are accompanied by experimental results on simulated and real data. Finally, some future challenges in Super-Resolution are outlined and discussed. © 2004 Wiley Periodicals, Inc. Int J Imaging Syst Technol, 14, 47–57, 2004; Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/ima.20007

**Key words:** Super-Resolution; demosaicing; inverse problem; dynamic Super-Resolution; image-reconstruction; robust estimation; robust regularization

I. INTRODUCTION

On the quest to achieve high resolution imaging systems, one quickly runs into the problem of diminishing returns. Specifically, the imaging chips and optical components necessary to capture very high-resolution images become prohibitively expensive, costing in the millions of dollars for scientific applications (Parulski et al., 1992). Super-resolution is the term generally applied to the problem of transcending the limitations of optical imaging systems through the use of image processing algorithms, which presumably are relatively inexpensive to implement. The application of such algorithms will certainly continue to proliferate in any situation where high-quality optical imaging systems cannot be incorporated or are too expensive to utilize.

The basic idea behind Super-Resolution is the fusion of a sequence of low-resolution noisy blurred images to produce a higher-resolution image or sequence. Early works on Super-Resolution showed that the aliasing effects in the high-resolution fused image can be reduced (or even completely removed), if a relative sub-pixel motion exits between the undersampled input images (Huang and Tsai, 1984). However, contrary to the naive frequency domain description of this early work, we shall see that, in general, super-resolution is a computationally complex and numerically ill-posed problem. All this makes Super-Resolution one of the most appealing research areas for image processing researchers.

Although several articles have surveyed the different classical Super-Resolution methods and compared their performances (e.g., Borman and Stevenson, 1998; Kang and Chaudhuri, 2003), the intention of this article is to pinpoint the various difficulties inherent to the Super-Resolution problem for a variety of application settings often ignored in the past. We review many of the most recent and popular methods, and outline some of our recent work addressing these issues.

The organization of this article is as follows. In Section II we study Super-Resolution as an inverse problem and address related regularization issues. In Section III we analyze a general model for imaging systems applicable to various scenarios of Super-Resolution. In Section IV we describe three different application settings and our approaches to dealing with them. Specifically, we address the problem of robust Super-Resolution, the treatment of color images and mosaiced sources, and dynamic Super-Resolution. Finally, we conclude with a list of challenges to be addressed in future work on Super-Resolution.

II. SUPER-RESOLUTION AS AN INVERSE PROBLEM

Super-resolution algorithms attempt to extract the high-resolution image corrupted by the limitations of the optical imaging system. This type of problem is an example of an inverse problem, wherein the source of information (high-resolution image) is estimated from the observed data (low-resolution image or images). Solving an inverse problem in general requires first constructing a forward model. By far, the most common forward model for the problem of Super-Resolution is linear in form:

$$\underline{Y}(t) = M(t)\underline{X}(t) + \underline{V}(t), \qquad (1)$$

where $\underline{Y}$ is the measured data (single or collection of images), $M$ represents the imaging system, $\underline{X}$ is the unknown high-resolution image or images, $\underline{V}$ is the random noise inherent to any imaging

Correspondence to: S. Farsiu, e-mail: farsiu@ee.ucsc.edu

Grant sponsor: This work was supported in part by the National Science Foundation Grant CCR-9984246, US Air Force Grant F49620-03-1-0387, and by the National Science Foundation Science and Technology Center for Adaptive Optics, managed by the University of California at Santa Cruz under Cooperative Agreement No. AST-9876783.



*Fig.23 Corrected image of Fig.15*

## 5.3 Stage II. Image Alignment - SIFT

### 5.3.1 Introduction

In this section, Scale Invariant Feature Transform (SIFT) would be introduced. SIFT algorithm is developed by David Lowe in University of British Columbia. The university has applied for a patent on the algorithm in the United States. As a result, the algorithm is restricted by patents in the United States. The SIFT is discussed in detail in [11]. Some figures are also come from this paper.

The features extracted from this method are invariant to image scaling and rotation. It is also partially invariant to change in illumination and 3D camera viewpoint. In addition, the features extracted also have the following properties:

- **Locality:** The features extracted are local so that they are robust to clutter and occlusion.
- **Distinctiveness:** The features are very distinctive so the probability of matching features wrongly is very low.
- **Quantity:** Many features can be generated even from a small image.
- **Efficiency:** The method to extract is very efficient

Extracting these features take a cascade filtering approach so that the cost can be minimized. The following is the key stages to extract the feature:

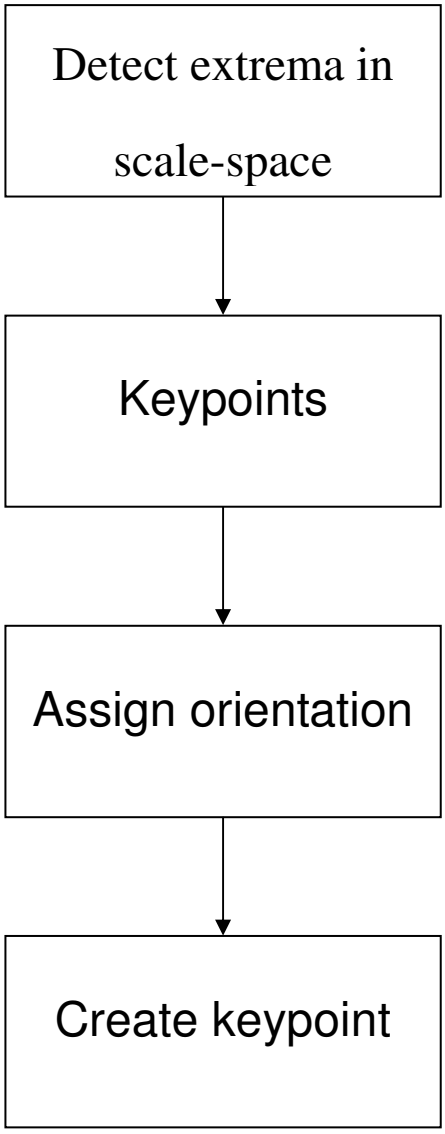



*Fig.24 Flow diagram of keypoints generation*

**Scale-space extrema detection**

Search over all scale and location of the image. Use a difference-of-Gaussian function to identify candidate keypoints. The points are invariant to scale and orientation.

**Keypoint localization**

Determine location and scale at each candidate location using a detailed model. Based on the stability of the candidate location, keypoints are selected.

**Orientation assignment**

Based on local image gradient directions, orientations assigned to each keypoint location. The image would be transformed relative to the assigned orientation, scale and location for each keypoints. So the points are invariant to orientation, scale and location.

**Keypoint descriptor**

Around each keypoint, the local image gradients are measured. They are transformed so that it is invariant to distortion and change in illumination.

In the following section, detail information of how to generate keypoints would be described.

### 5.3.2 Detect extrema in scale-space

In the cascade filtering approach, an efficient algorithm would be used to identify candidate locations first. Then the candidate locations would then be examined in further detail.

Firstly, it is to identify locations that are invariant to scale. It can be done by searching features in all possible scales using function of scale space.

Koenderink and Lindeberg show that the only possible scale-space kernel is Gaussian function shown in the following

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

The scale space of an image is

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Lowe has proposed that detection of scale-space extrema in the difference-of-Gaussian function convolved with the original image to detect keyopoint locations in scale space efficiently.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

There are two advantages to choose this function.

1. It is efficient to compute.
2. The function is approximately equal to scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$ as

   $$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

Lindeberg (1994) showed the above function $\sigma^2\nabla^2 G$ can be used for true scale invariance. Mikolajczyk (2002) have further show that t he extrema of the function can produce the very stable image features.

Fig.25 shows the calculation of $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$

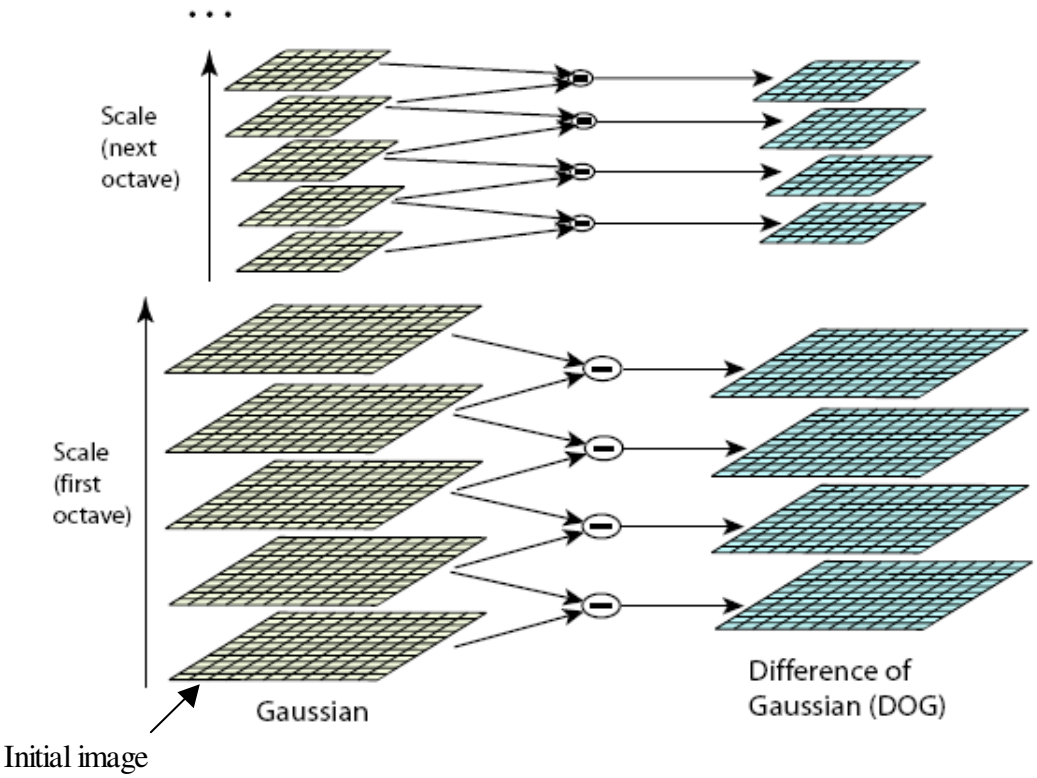



*Fig.25 calculation of* $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$

The initial image is convolved with the Gaussian function to produce the blurred images. Each neighbour blurred image would be taken Gaussioan with different $\sigma$ by factor of k. The Difference of Gaussian would be obtained by subtracting the neighbors blurred image.

According to Lowe, the suitable of k is square root of 2 and there should 3 scales for each octave. The initial value of $\sigma$ is 1.6.

After an octave is created, another octave would be needed to contructed. The

Gaussian blurred image with twice initial value of $\sigma$ is sub-sampled by taking every alternative pixel in the horizontal and vertical direction.

After we have computed all the difference-of-Gaussian of the image. We need to detect the extrema of it. By comparing a pixel to its 26 neighbours in 3x3x3 pixels, maxima and minima of the difference-of-Gaussian would be detected. The idea is illustrated in the following figure.

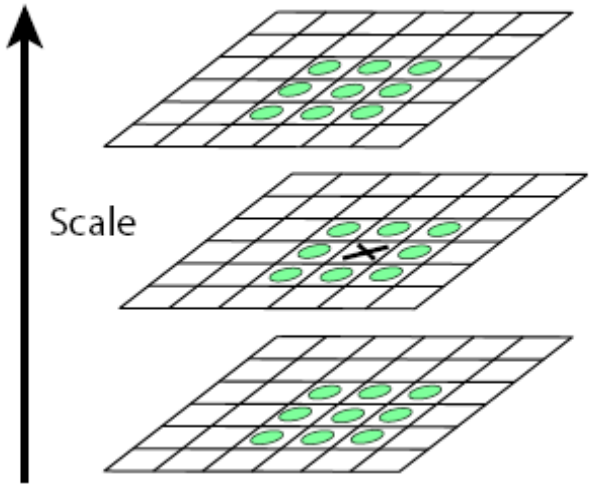



*Fig.26 detection of maxima and minima with 26 neighbours*

After the computation, candidate feature points are obtained.

## 5.3.3 Keypoint localization

After candidate keypoints have been detected, a detail model is use to fit the nearby data for location, scale and ratio. It is used to remove unstable keypoints.

The unstable keypoint includes low contrast and edge reponse. A suitable function is to detect such unstable keypoint and remove them from the candidate keypoints.

### 5.3.3.1 Remove keypoint with low contrast

To remove keypoint with low contrast, we need use Taylor expansion of the DoG function.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

where D is shifted so that the origin is at the candidate keypoint, $\mathbf{x} = (x,y,\sigma)^T$ is the offset from that point.

Take derivative of the above function with respect to x and set it zero, we can find the location of the extremum

$$\hat{\mathbf{x}} = \frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

By substitution, we obtains

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}$$

As a result, we would remove low contrast keypoint by removing keypoint with $D(\hat{\mathbf{x}})$ smaller than a suitable value.

#### 5.3.3.2 Remove keypoint having strong edge response

The DoG function has large principal curvature on edges. The principal curvature can be found using Hessian function **H**

$$\mathbf{H} = \begin{bmatrix} Dxx & Dxy \\ Dxy & Dyy \end{bmatrix}$$

and

$$Tr(\mathbf{H}) = Dxx + Dyy$$

$$Det(\mathbf{H}) = DxxDyy - Dxy^2$$

Then checking using suitable value of r.

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r+1)^2}{r}$$

If the above equation is satisfied, the keypoint is stable enough. Otherwise, The candidate keypoint need to be removed.

The following figures are shown in [11]. They show how the keypoints occur during localization.

Fig.27a shows the initial image.

*Fig.27a Initial image*

Fig.27b shows the keypoints after detecting extrema of Difference-of-Gaussian of the image. The keypoints are shown using the arrow. The arrow shows the image gradient of the keypoint.

*Fig.27b Image gradient of keypoints*

Fig.27c shows the keypoints after eliminating low contrast keypoints.

*Fig.27c high contrast keypoints*

Fig.27d shows the keypoints after removing keypoints with strong edge response.

*Fig.27d keypoints with weak edge response*

### 5.3.4 Orientation assignment

After the keypoints have been localized, each keypoint would be assigned to a consistent orientation base on the local image around the keypoint. Then the keypoint descriptor could be described relative to the consistent orientation to achieve rotational invariant.

At each pixel, the image gradients can be computed using the following equation.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) + L(x, y-1))^2}$$

At each pixel, the image orientations can be computed using the following equation.

$$\theta(x, y) = \tan^{-1}((L(, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

The consistent orientation of the keypoint is found by detecting the peak of in the orientation histogram of the local image gradient. The orientation histogram in fact is created by using a Gaussian window with $\sigma = 1.5$ times of the scale of the keypoint.

### 5.3.5 Keypoint Descriptor

After the above step, each keypoint is assigned an image location, scale and orientation. Then a distinctive descriptor for the local image is needed to compute and the descriptor need to invariant to other variations like change in illumination.

The first thing to do is to compute the gradient magnitude and orientation around the keypoint locations. The gradient magnitudes are weighted by a Gaussian window with $\sigma = 1.5$ times the width of the descriptor indicated by the circle.

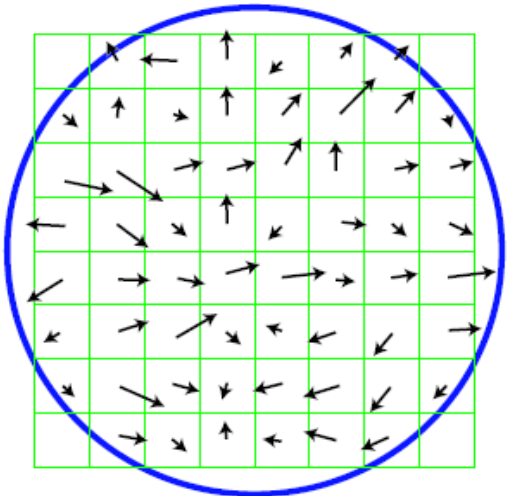


*Fig.28 Image gradient*

The samples are then accumulated into orientation histograms. It summaries the content over 4x4 subregions.

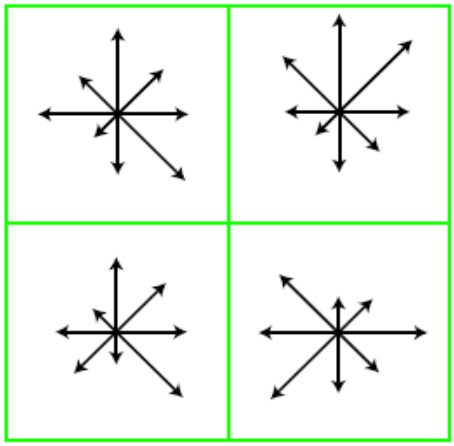


*Fig.29 Keypoint descriptor*

The length of the arrows indicates the sum of the gradient magnitudes around the local image region in that direction. The above figure just shows2x2 descriptor array computed from an 8x8 set of samples. There are 2x2x8 = 32 elements feature vector. But in the real situation, there would be 4x4 descriptors computed from 16x16 sample array. As there are 8 orientations in each descriptor, the keypoints would have 4x4x8 = 128 elements feature vector.

### 5.3.6 Keypoint Matching

In fact, there is no specific technique of keypoint matching for SIFT. The keypoint can be matched using the technique mentioned in the previous chapter. But generally, all the keypoints would be stored in KD tree.

When searching, in stead of using the KD tree searching. An algorithm called Best Bin First modified by KD tree searching is used because the dimension of the KD tree is too large (128 dimension). Using this algorithm can be more efficient. However, it is still not very efficient.

After searching, successful matches are got. In order to obtain good parameters of geometric transformation, RANSAC (RANdom Sample Consensus) are used to select matched keypoint to generate the parameters of geometric transformation.

## 5.3.7 Implementation - SIFT Sample Code

### 5.3.7.1 Introduction

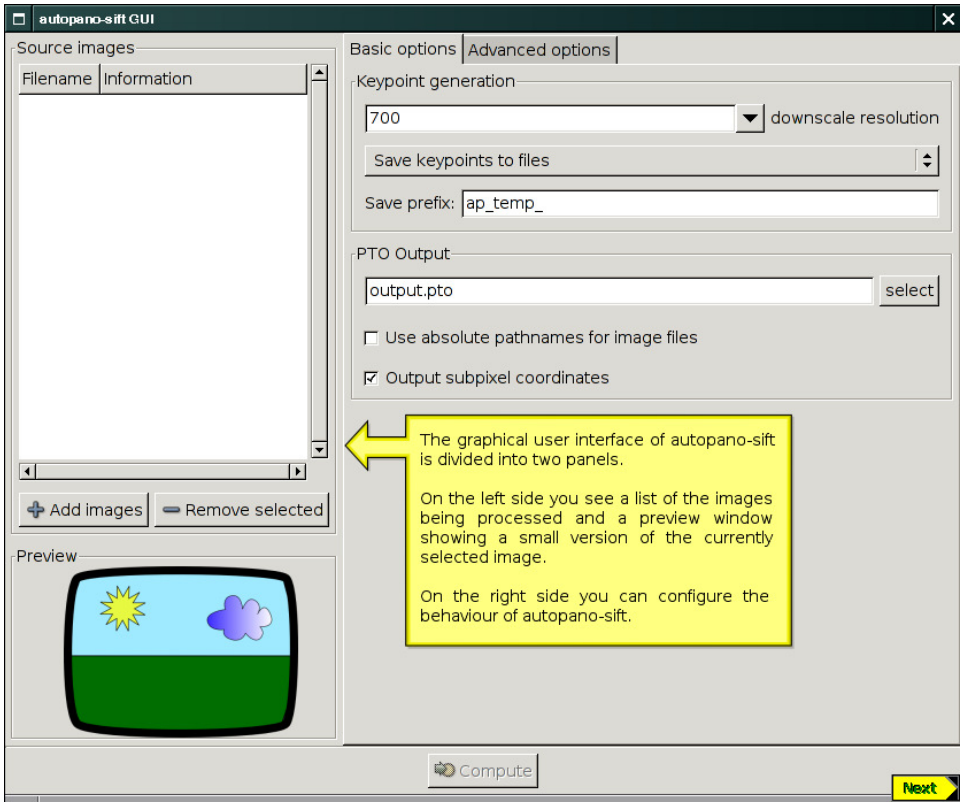



*Fig.30 GUI of autopano-sift package*

The autopano-sift package (http://user.cs.tu-berlin.de/~nowozin/autopano-sift/) is a program used to align images. This is done by using control points which give information about shared image features. The manual work of creating these control points can be immense. Especially for panorama images holding more than one line of images there is a huge number of possible overlaps.

The autopano-sift package can automatically create control point pairs by using a combination of sophisticated algorithms and models. For most feature-rich images it works very well, often outperforming humans in coverage, precision and speed.

There are several components in the package:

***Main Modules:***

(a) **autopano:** The key-point matching program. It can take SIFT key-point files as input and produce PTO output files.

(b) **generatekeys:** The SIFT key-point extraction program. Takes an image and gives key-points back.

***GUI Modules:***

(c) **autopanog:** The GTK# GUI front-end for generating and matching SIFT key-points. As a user wanting to stitch panorama images, this is all you will ever need.

(d) **showone:** Show the SIFT key-points overlayed over the source image. Use includes debugging and tuning of SIFT parameters and to get an impression how well spread the key-points are.

(e) **showtwo:** Do simplistic matching of SIFT key-points between two images, without geometric model. Use includes representing general matching quality.

#### 5.3.7.2 Overview of sample codes

We will now briefly describe the implementation of the sample codes.

The source codes of these programs are located in /src/util. These programs would create objects and do corresponding things. The source code of related class in declared and implemented in /src. For example, the *generatekeys* would make use of GaussianConvolution.cs, ScaleSpace.cs, etc. The autopanog would make make use of MatchKeys.cs, RANSAC.cs.

While compilation of the codes, a libsift.dll file is generated from the codes located in /src, and this file is the reference for the codes in /src/util. Therefore it is needed when compile the main modules in /src/util. After compilation of main modules, 2 executable files are generated, namely, generate-key.exe and autopano.exe. The generate-keys program will input the images and generates xml files for each image. The xml file stores the keypoint descriptor of that image for later use. The autopano program will input those xml files and carry out the keypoint matching. The program will each time pick 2 xml files for matching, i.e. it will performs 6 times if there is 3 xml files (= 3 images). The resulting matched coordinates are stored in .pto files.

#### 5.3.7.3 Modification to suit our use

We first discarded those GUI modules and write our own "Makefile". We also wrote our own shell script files to suit our use of those programs. The generate-key program is kept used as before but the autopano program is used in a different way. We modified it so that it would only match keypoints of first image with other images. In other words, it performs 6 times of matching for 7 images, where as the first image is an overall image and the others are the sub-images. The reason why we do so is related to the stitching method we proposed, details will be explained in **part 5.4** ***Stage II. Image Stitching***.

### 5.3.8 Conclusion

Scale Invariant Feature Transform (SIFT) algorithm is very powerful. The algorithm can find and match keypoints with images of different scales and orientations. And this is the reason why we chose SIFT algorithm for our image alignment. And after understood the sample codes, we can easily use it to suit our need. Thus it shortens our time to implement the SIFT algorithm.

# 5.4 Stage II. Image Stitching

## 5.4.1 Introduction

The second stage of our project is to stitch the sub-images together to form a high resolution document image. To overcome the problems different scales and orientations of sub-images, we have used the techniques of image transformation.

After image alignment, those sub-images are still of different scales and orientations, thus we cannot simply combine them together to form a image. Therefore we employed the technique of image transformation to adjust the scale and orientation of the sub-images. Since we have taken a photo of overall view of the document, the sub-images are basically transformed according to it.

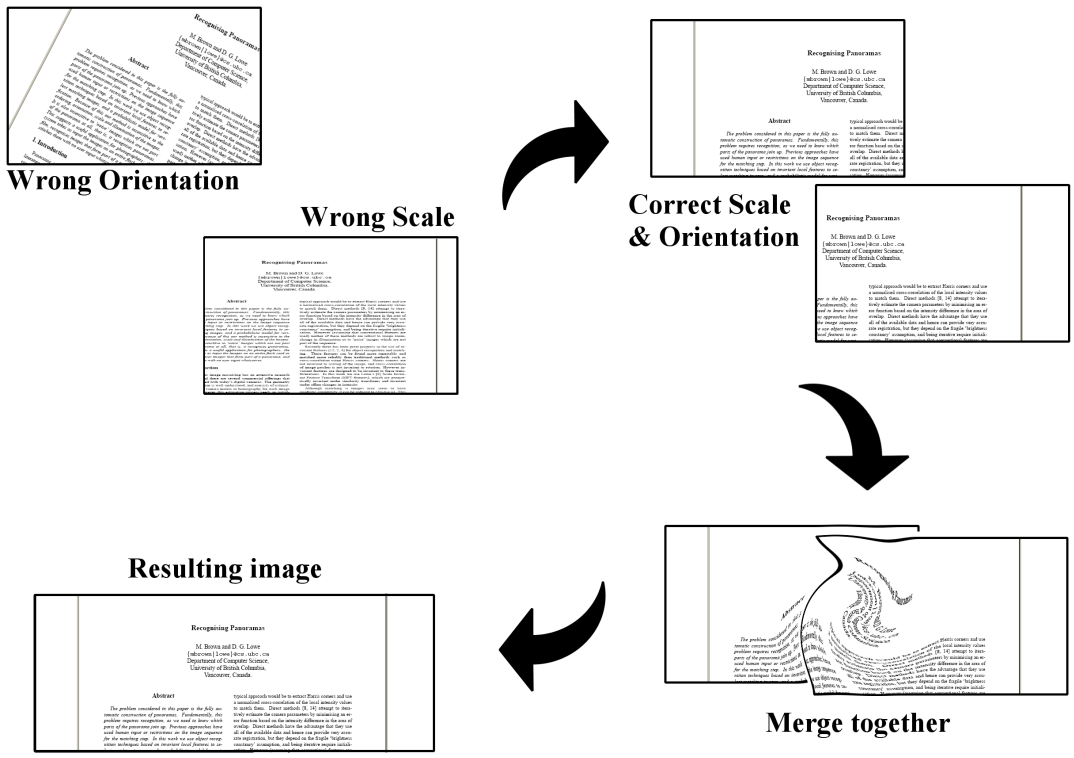



*Fig.31 Flow diagram of image stitching*

## 5.4.2 Implementation

*Fig.32a Image1* – Overall Image (Resolution: 1600x1200 pixels)

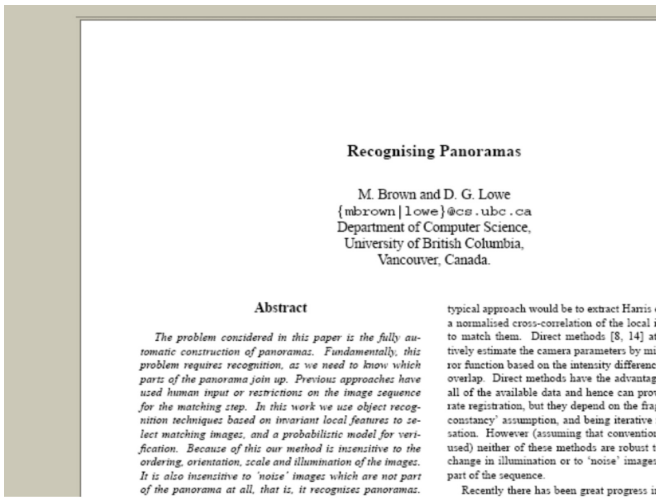


*Fig.32b Image2* – Any one of the sub-images (Resolution: 1600x1200 pixels)

*Fig.32c Image3* – All sub-images other than *Image2* (Resolution: 1600x1200 pixels)

**Step 1:**

Find the farthest 4 points among all matching points in Image2, i.e. the 4 tiepoints that covers the most area inside the image. We implemented a simple routine that finds the points that are farthest from the so-called starting point. The original starting point is the center of the image, i.e. x = width/2 & y = height/2. Then the image is divided into 4 regions.

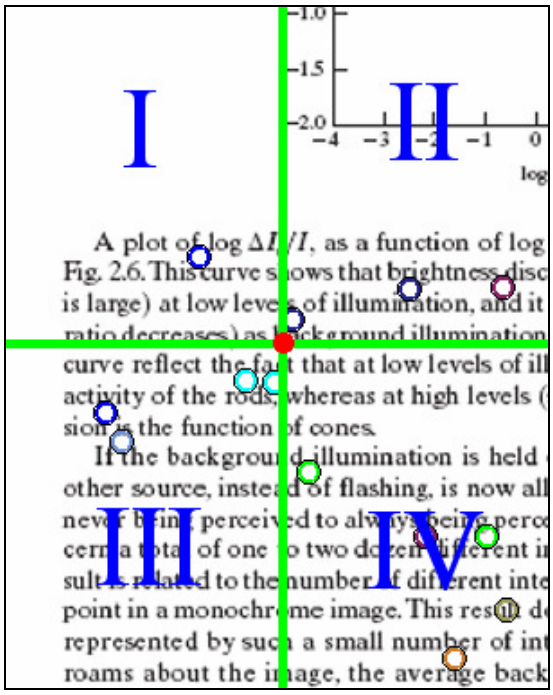



*Fig.33 sub-image with 4 regions*

Red dot – Starting point

I, II, III & IV – the 4 regions divided by the starting point

Small circles – matching points

In each region, the distance of each point from the starting point is calculated, and the farthest one is recorded. As illustrated below, the 4 corners are the points that are farthest in each region, the blue line is the distance of each point from the starting point.

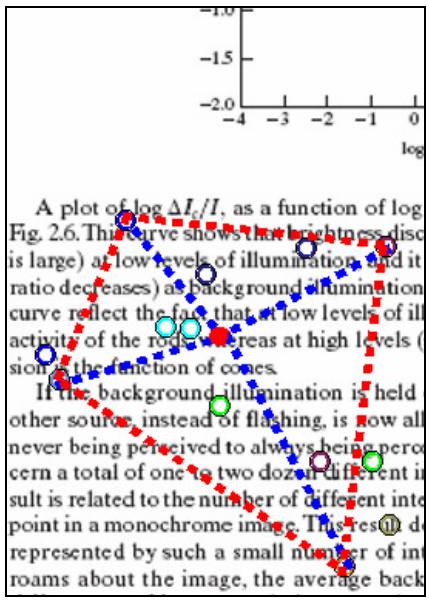


*Fig.34 Farthest 4 matching keypoints*

However, if there is no matching point in a region, the starting point will adjust automatically and perform the routine again, so that exactly 4 points are recorded in the image. For example, if the region I and III do not contain matching points, the starting point will shift right by 50%. If only region III does not contain matching points, the starting point would not only shift right by 50% but also shift up by 50%.

**Step 2:**

Denote the 4 recorded tiepoints in Image2 as $(x_1', y_1')$ to $(x_4', y_4')$, we can then calculate the transform parameters $C_{Ai}$ with the corresponding matching tiepoints in Image1 (from $(x_1, y_1)$ to $(x_4, y_4)$) by the equation shown below:

$$\begin{bmatrix} c_{A1} \\ c_{A2} \\ c_{A3} \\ c_{A4} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix}$$

$$\begin{bmatrix} c_{A5} \\ c_{A6} \\ c_{A7} \\ c_{A8} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix}$$

**Step 3:**

We calculate the new transformed coordinate (from $(x_1', y_1')$ to $(x_4', y_4')$) of the 4 tiepoints in Image1 (from $(x_5, y_5)$ to $(x_8, y_8)$) by the transform parameters $C_{Ai}$.

$$\begin{bmatrix} x_5' \\ x_6' \\ x_7' \\ x_8' \end{bmatrix} = \begin{bmatrix} c_{A1} \\ c_{A2} \\ c_{A3} \\ c_{A4} \end{bmatrix} \begin{bmatrix} x_5 & y_5 & x_5y_5 & 1 \\ x_6 & y_6 & x_6y_6 & 1 \\ x_7 & y_7 & x_7y_7 & 1 \\ x_8 & y_8 & x_8y_8 & 1 \end{bmatrix}$$

$$\begin{bmatrix} y_5' \\ y_6' \\ y_7' \\ y_8' \end{bmatrix} = \begin{bmatrix} c_{A5} \\ c_{A6} \\ c_{A7} \\ c_{A8} \end{bmatrix} \begin{bmatrix} x_5 & y_5 & x_5y_5 & 1 \\ x_6 & y_6 & x_6y_6 & 1 \\ x_7 & y_7 & x_7y_7 & 1 \\ x_8 & y_8 & x_8y_8 & 1 \end{bmatrix}$$

**Step 4:**

Denote the 4 tiepoints in *Image3* as ($x_1$'', $y_1$'') to ($x_4$'', $y_4$''), which are corresponding to tiepoints from ($x_5$', $y_5$') to ($x_8$', $y_8$'). We calculate another set of transform parameters $C_{Bi}$ with the 4 tiepoints pair:

$$\begin{bmatrix} c_{B1} \\ c_{B2} \\ c_{B3} \\ c_{B4} \end{bmatrix} = \begin{bmatrix} x_1'' & y_1'' & x_1''y_1'' & 1 \\ x_2'' & y_2'' & x_2''y_2'' & 1 \\ x_3'' & y_3'' & x_3''y_3'' & 1 \\ x_4'' & y_4'' & x_4''y_4'' & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_5' \\ x_6' \\ x_7' \\ x_8' \end{bmatrix}$$

$$\begin{bmatrix} c_{B5} \\ c_{B6} \\ c_{B7} \\ c_{B8} \end{bmatrix} = \begin{bmatrix} x_1'' & y_1'' & x_1''y_1'' & 1 \\ x_2'' & y_2'' & x_2''y_2'' & 1 \\ x_3'' & y_3'' & x_3''y_3'' & 1 \\ x_4'' & y_4'' & x_4''y_4'' & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_5' \\ y_6' \\ y_7' \\ y_8' \end{bmatrix}$$

**Step 5:**

Transform the whole *Image3* with the parameters $C_{Bi}$.

**Step 6:**

Since after Step 5, resulting image of *Image3* is already of the same orientation and same scale with *Image2*, therefore we can directly merge it with *Image2* to form an intermediate image.

**Step 7:**

Repeat Step 4 to Step 6 with all other sub-images.

*Note:*

*Images in Fig.35 are not drawn to scale,*

*They all are resolution 1600x1200, except "High Resolution Document Image"*

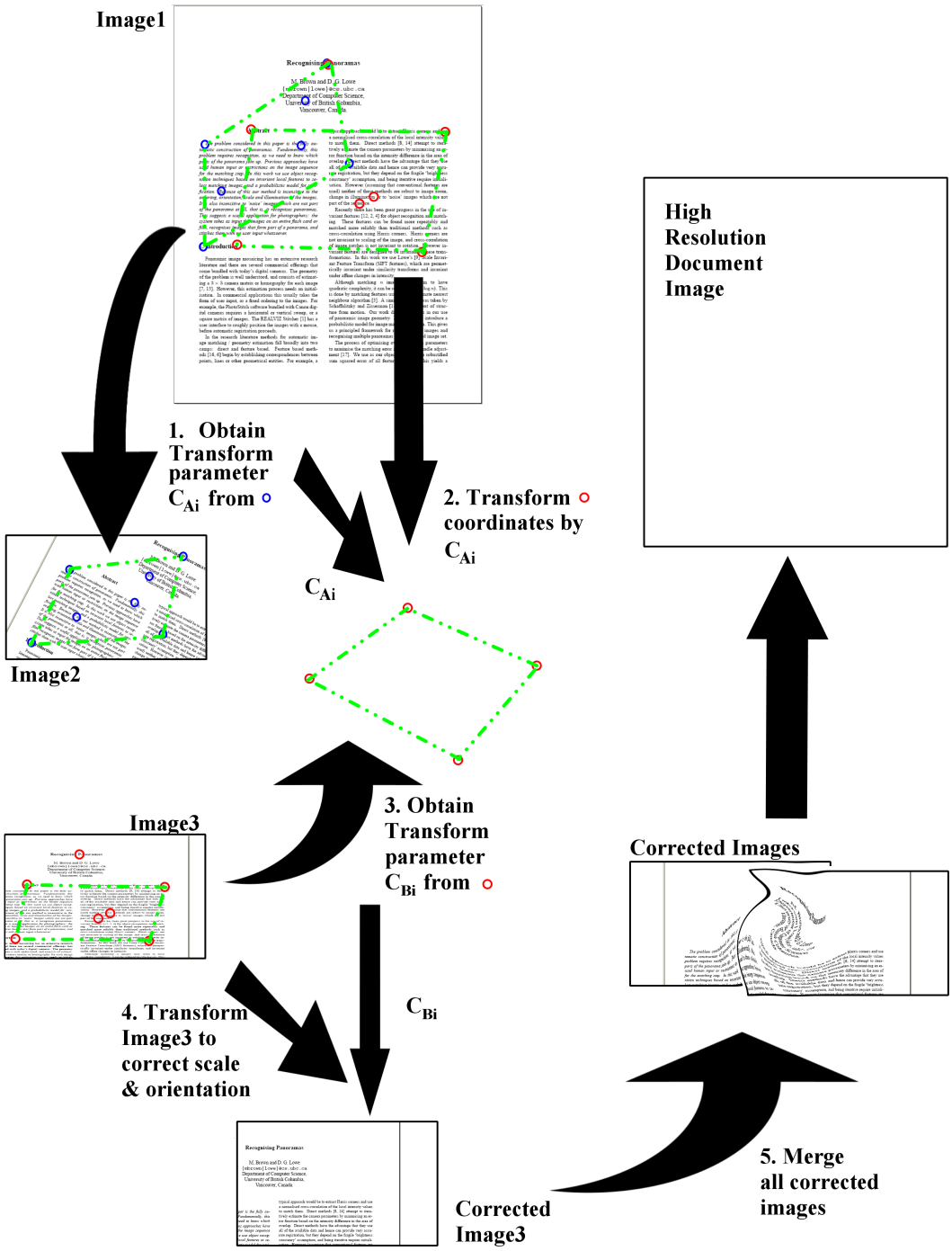



*Fig.35 Flow diagram of image stitching*

### 5.4.3 Conclusion

The normal way to stitch panorama is to match the keypoints with consecutive images. For example, suppose the 4 sub-images of a panorama from left to right are named *Image1*, *Image2*, *Image3* and *Image4*. The image alignment will match the keypoints on *Image1* and *Image2*, then on *Image2* and *Image3*, and finally on *Image3* and *Image4*. Thus some of the relationships between those sub-images are ignored, such as relationship between *Image1* and *Image3*.

Our proposed method of image stitching is basically relies on the overall image of the document. The relationships between any two sub-images are resided on that overall image. Thus the stitching of sub-images can be more reliable. Furthermore, since the matching of keypoints depends on the overall image, but not the consecutive sequence of those sub-images, user can take photos of sub-images in any order!

It can also be extended that, after an overall photo of a document is taken, we may provide an overlay image of requesting sub-image on the screen of mobile phone. So that the user may follow the best way to take photos of sub-images and get a much more accurate stitching result.

## 5.5 Stage III. Image Blending

### 5.5.1 Introduction

After image transformation is performed, some of the sub-images are zoomed or shrank in scale. Thus the resulting high resolution image will have defects such as checkerboard effects. In order to improve the image quality, image blending is applied.

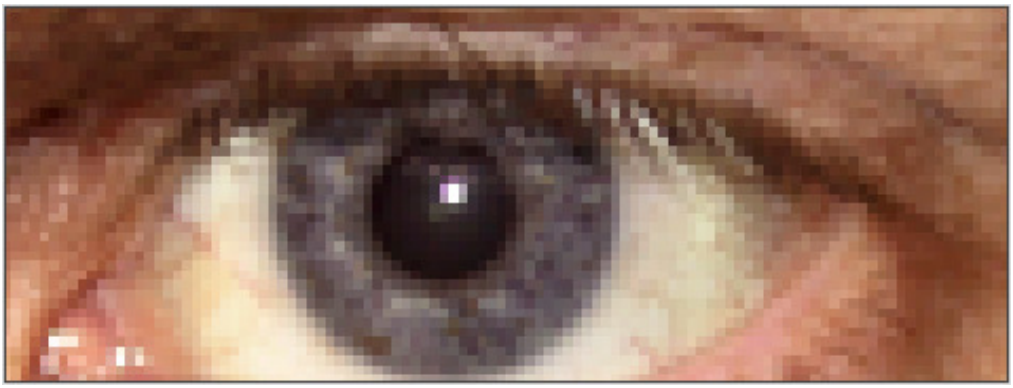


*Fig.36 Image with checkerboard effect*

### 5.5.2 Basic concepts - Interpolation

Interpolation (sometimes called resampling) is an imaging method to increase (or decrease) the number of pixels in a digital image. Some digital cameras use interpolation to produce a larger image than the sensor captured or to create digital zoom. Virtually all image editing software support one or more methods of interpolation. How smoothly images are enlarged without introducing jaggies depends on the sophistication of the algorithm. There are of course many methods of interpolation:

(a) **Nearest Neighbor Interpolation -** It is the simplest method and basically makes the pixels bigger. The color of a pixel in the new image is the color of the nearest pixel of the original image. If you enlarge 200%, one pixel will be enlarged to a 2 x

2 area of 4 pixels with the same color as the original pixel. Most image viewing and editing software use this type of interpolation to enlarge a digital image for the purpose of closer examination because it does not change the color information of the image and does not introduce any anti-aliasing. For the same reason, it is not suitable to enlarge photographic images because it increases the visibility of jaggies.

(b) **Bilinear Interpolation** – It determines the value of a new pixel based on a weighted average of the 4 pixels in the nearest 2 x 2 neighborhood of the pixel in the original image. The averaging has an anti-aliasing effect and therefore produces relatively smooth edges with hardly any jaggies.

(c) **Bicubic interpolation** - It is more sophisticated and produces smoother edges than bilinear interpolation. Notice for instance the smoother eyelashes in the example below. Here, a new pixel is a bicubic function using 16 pixels in the nearest 4 x 4 neighborhood of the pixel in the original image. This is the method most commonly used by image editing software, printer drivers and many digital cameras for resampling images. As mentioned in my review, Adobe Photoshop CS offers two variants of the bicubic interpolation method: bicubic smoother and bicubic sharper.

(d) **Fractal interpolation** – It is mainly useful for extreme enlargements (for large prints) as it retains the shape of things more accurately with cleaner, sharper edges and less halos and blurring around the edges than bicubic interpolation would do. An example is Genuine Fractals Pro from The Altamira Group.

## 5.5.3 Implementation

### 5.5.3.1 Bilinear interpolation

In our project, we used the techniques of bilinear interpolation and averaging. The actual implementation of bilinear interpolation is illustrated in the following graphs:

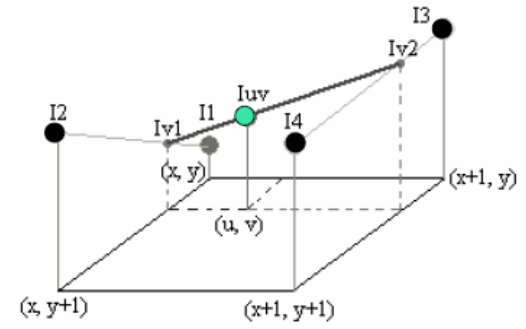
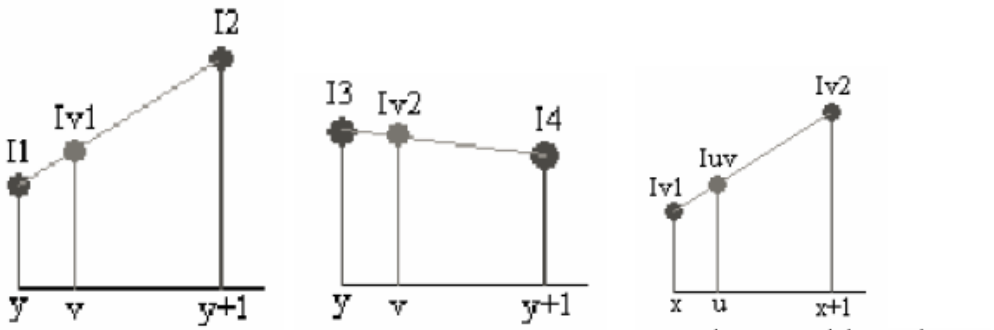



*Fig.37*
*Pixel value calculated by bilinear interpolation*




$$I_{v1} = (I_2 - I_1)(v - y) + I_1 \quad I_{v2} = (I_4 - I_3)(v - y) + I_3 \quad I_{uv} = (I_{v2} - I_{v1})(u - x) + I_{v1}$$

All vector components of RGB values are performed bilinear interpolation. Take R (red) component as an example. The value of a desired pixel is determined based on the weighted average of the 4 neighborhood pixels. First we calculate the pixel value along y-axis by the equations:

$$I_{V1} = (I_2 - I_1)(v - y) + I_1$$

$$I_{V2} = (I_4 - I_3)(v - y) + I_3$$

And finally find out the desired pixel value by:

$$I_{UV} = (I_{V2} - I_{V1})(u - x) + I_{V1}$$

It is similar cases for G (green) and B (blue) components.

### 5.5.3.2 Averaging mask

There are also cases that after shearing the sub-images, some pixels are missing its values, but bilinear interpolation is not suitable for these cases. Therefore we applied averaging mask instead of bilinear interpolation.

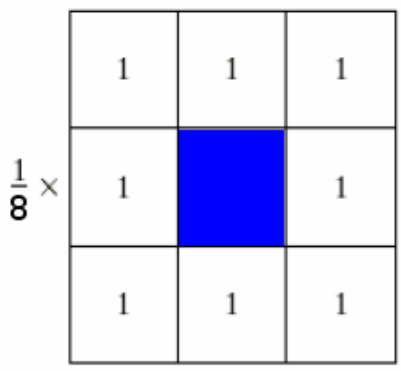



*Fig.38 Averaging mask*

The idea of averaging mask is simple: take the mean value of the nearest 8 pixels as the pixel value of the desired pixel. Since the "divided by 8" process can be replaced by "shift 3 binary digit", this technique is fast and efficient.

### 5.5.3.3 Thresholding

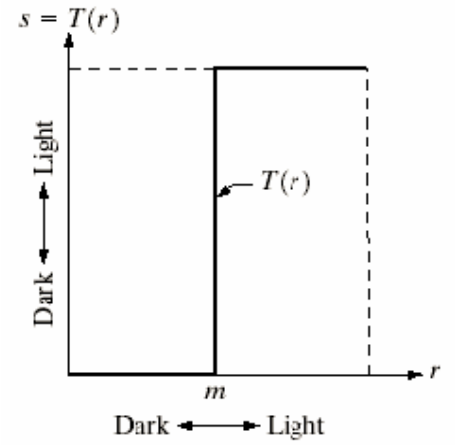



*Fig.39 Thresholding function*

This is an optional enhancement for documents contain only words in black and white. The idea of thresholding is to quantize the pixel values into 2 values only, i.e. black (represent by '0') or white (represent by '1'). It is performed on all RGB (Red, Green & Blue) components on every pixel.

Through some experiments, we find that the threshold should be set at around 90 out of 255 of RGB components. In other words, if both the R, B and G components are of values smaller than 90, we regard it as dark, and so assign it a black pixel value (RGB = #000000). Otherwise, it is regarded as bright and assign it to white pixel value (RGB = #FFFFFF). This technique can remove those noises from the resulting image. However, this is only applicable to black and white documents.

### 5.5.4 Conclusion

After applying the 3 techniques mentioned above, the defects after stitching the sub-images are minimized. Nevertheless, it is difficult to achieve a perfect stitched document, even commercial products would have flaws. What we can do is to employ more techniques to reduce those flaws and make them hard to see. Other techniques are still possible to employ but it is definitely that processing time will increase accordingly. Therefore, it would be hard to decide how sharp the image we should get if processing time is considered.

# 5.6 Experimental Results

*Note:*

*Images shown in Fig & are of resolution 1600x1200 (regular 2 Mega pixel),*

*Images shown in Fig are of resolution 2616x1870*

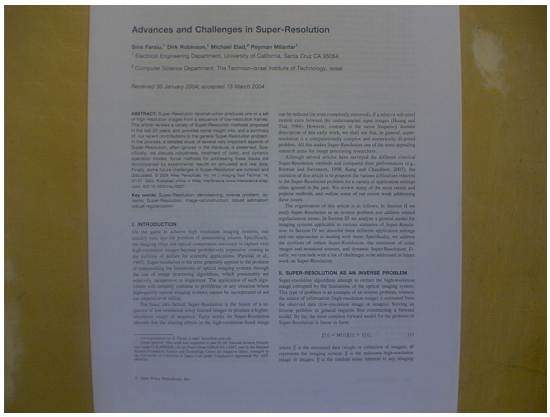


*Fig.40a Overall Image of document*

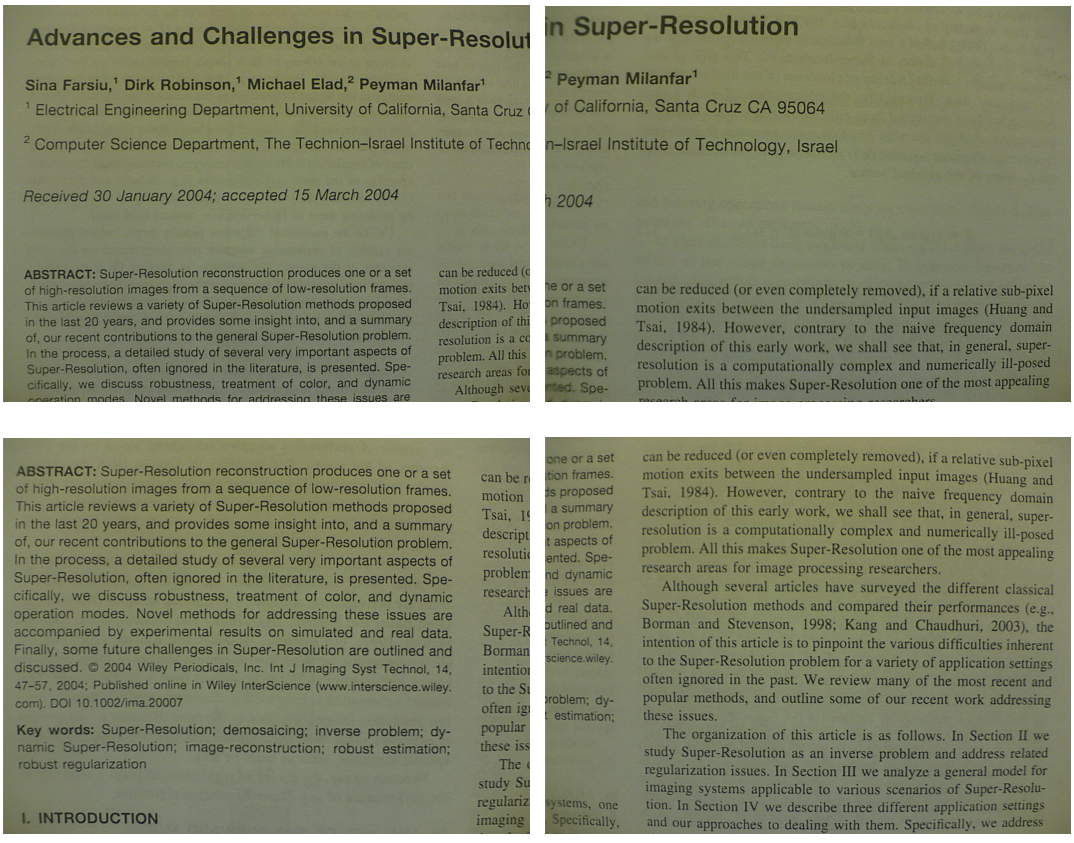



*Fig.40b Four sub-images of different parts of document*

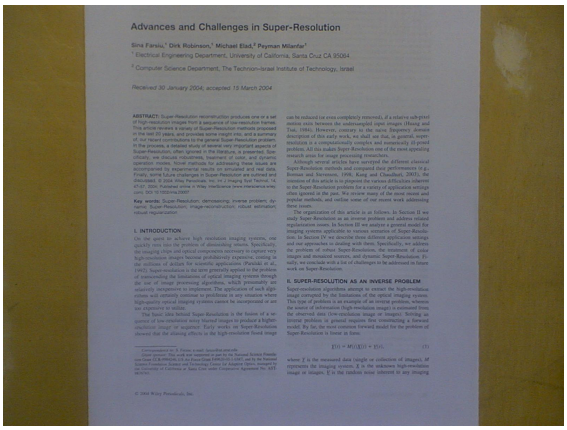


*Fig.41a Overall Image of document after lens correction*

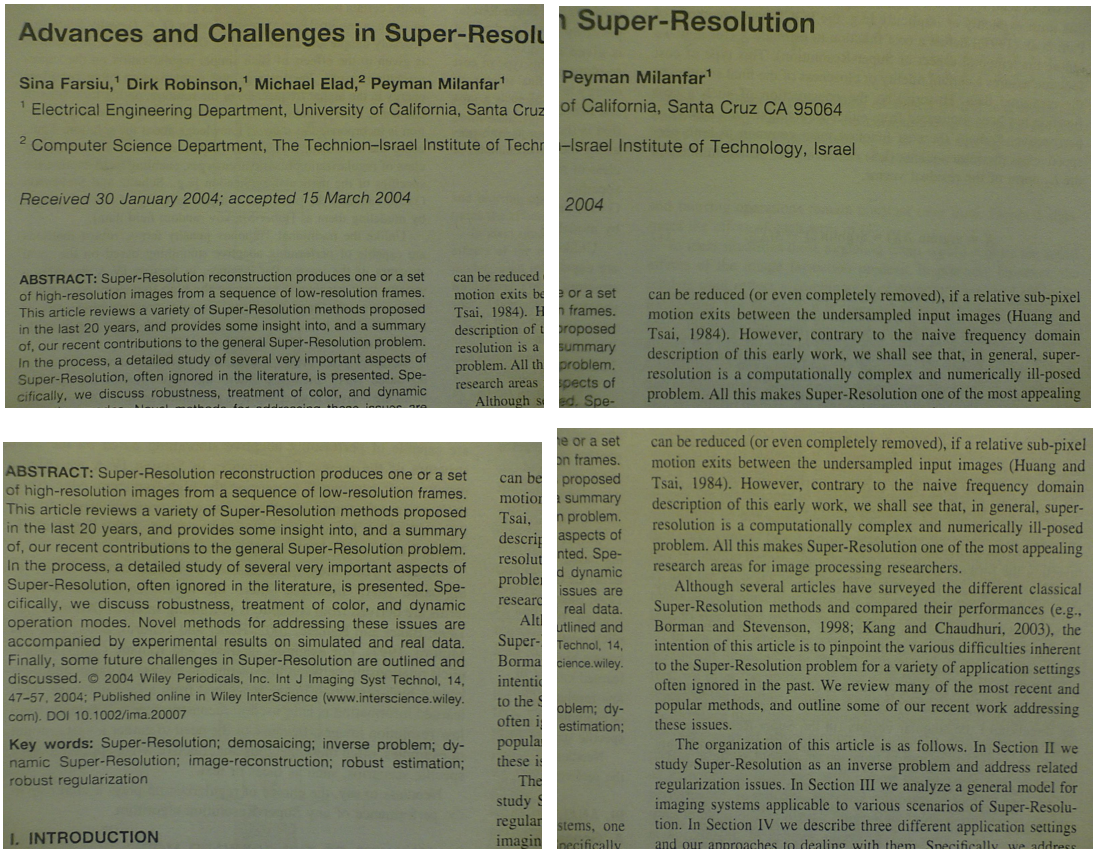
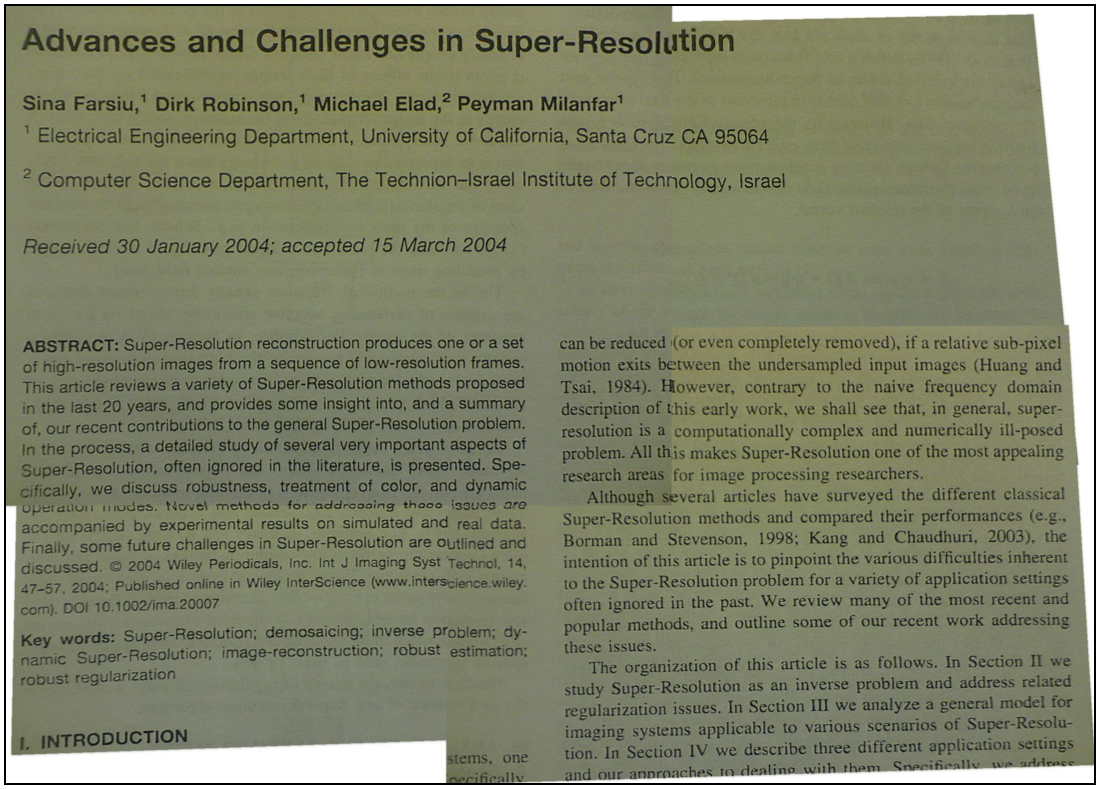



*Fig.41b Four sub-images of different parts of document after lens correction*



## Advances and Challenges in Super-Resolution

**Sina Farsiu,[1] Dirk Robinson,[1] Michael Elad,[2] Peyman Milanfar[1]**

[1] Electrical Engineering Department, University of California, Santa Cruz CA 95064

[2] Computer Science Department, The Technion–Israel Institute of Technology, Israel

*Received 30 January 2004; accepted 15 March 2004*

**ABSTRACT:** Super-Resolution reconstruction produces one or a set of high-resolution images from a sequence of low-resolution frames. This article reviews a variety of Super-Resolution methods proposed in the last 20 years, and provides some insight into, and a summary of, our recent contributions to the general Super-Resolution problem. In the process, a detailed study of several very important aspects of Super-Resolution, often ignored in the literature, is presented. Specifically, we discuss robustness, treatment of color, and dynamic operation modes. Novel methods for addressing these issues are accompanied by experimental results on simulated and real data. Finally, some future challenges in Super-Resolution are outlined and discussed. © 2004 Wiley Periodicals, Inc. Int J Imaging Syst Technol, 14, 47–57, 2004; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/ima.20007

**Key words:** Super-Resolution; demosaicing; inverse problem; dynamic Super-Resolution; image-reconstruction; robust estimation; robust regularization

### I. INTRODUCTION

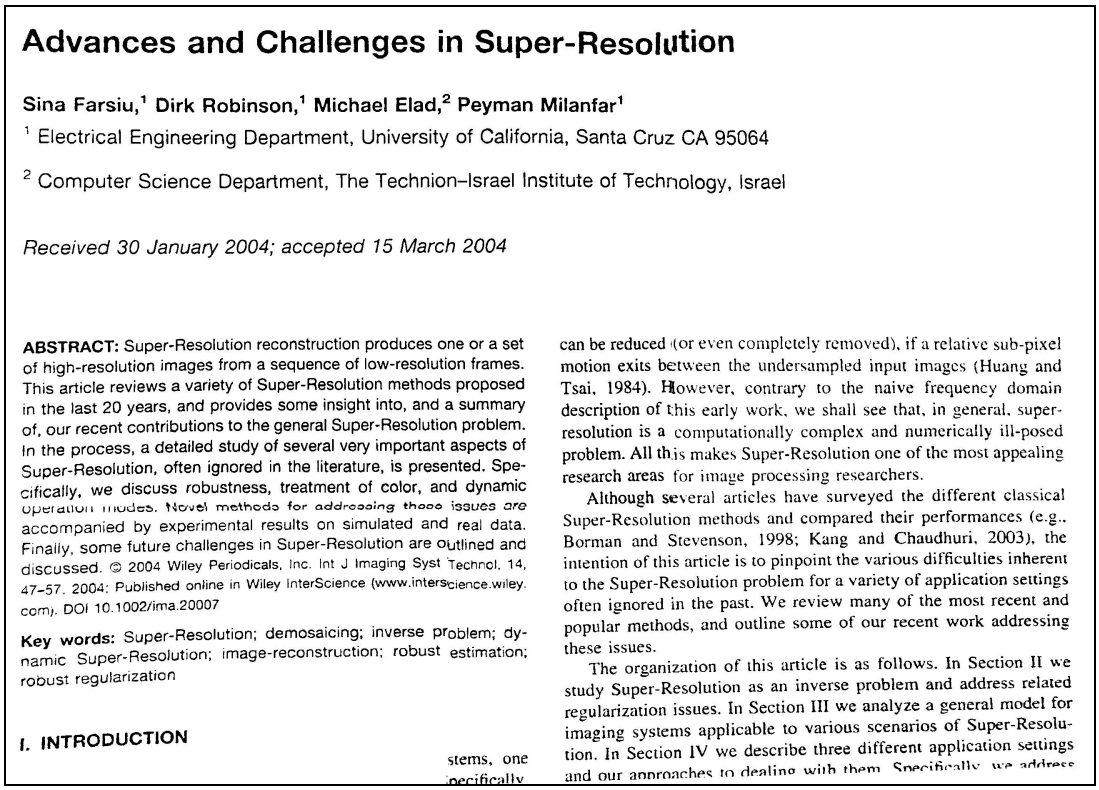
stems, one
ecifically

can be reduced (or even completely removed), if a relative sub-pixel motion exits between the undersampled input images (Huang and Tsai, 1984). However, contrary to the naive frequency domain description of this early work, we shall see that, in general, super-resolution is a computationally complex and numerically ill-posed problem. All this makes Super-Resolution one of the most appealing research areas for image processing researchers.

Although several articles have surveyed the different classical Super-Resolution methods and compared their performances (e.g., Borman and Stevenson, 1998; Kang and Chaudhuri, 2003), the intention of this article is to pinpoint the various difficulties inherent to the Super-Resolution problem for a variety of application settings often ignored in the past. We review many of the most recent and popular methods, and outline some of our recent work addressing these issues.

The organization of this article is as follows. In Section II we study Super-Resolution as an inverse problem and address related regularization issues. In Section III we analyze a general model for imaging systems applicable to various scenarios of Super-Resolution. In Section IV we describe three different application settings and our approaches to dealing with them. Specifically, we address

*Fig.42 Stitched high resolution document image (upper half of the document only)*



## Advances and Challenges in Super-Resolution

**Sina Farsiu,[1] Dirk Robinson,[1] Michael Elad,[2] Peyman Milanfar[1]**

[1] Electrical Engineering Department, University of California, Santa Cruz CA 95064

[2] Computer Science Department, The Technion–Israel Institute of Technology, Israel

*Received 30 January 2004; accepted 15 March 2004*

**ABSTRACT:** Super-Resolution reconstruction produces one or a set of high-resolution images from a sequence of low-resolution frames. This article reviews a variety of Super-Resolution methods proposed in the last 20 years, and provides some insight into, and a summary of, our recent contributions to the general Super-Resolution problem. In the process, a detailed study of several very important aspects of Super-Resolution, often ignored in the literature, is presented. Specifically, we discuss robustness, treatment of color, and dynamic operation modes. Novel methods for addressing these issues are accompanied by experimental results on simulated and real data. Finally, some future challenges in Super-Resolution are outlined and discussed. © 2004 Wiley Periodicals, Inc. Int J Imaging Syst Technol, 14, 47–57, 2004; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/ima.20007

**Key words:** Super-Resolution; demosaicing; inverse problem; dynamic Super-Resolution; image-reconstruction; robust estimation; robust regularization

### I. INTRODUCTION

stems, one
ecifically

can be reduced (or even completely removed), if a relative sub-pixel motion exits between the undersampled input images (Huang and Tsai, 1984). However, contrary to the naive frequency domain description of this early work, we shall see that, in general, super-resolution is a computationally complex and numerically ill-posed problem. All this makes Super-Resolution one of the most appealing research areas for image processing researchers.

Although several articles have surveyed the different classical Super-Resolution methods and compared their performances (e.g., Borman and Stevenson, 1998; Kang and Chaudhuri, 2003), the intention of this article is to pinpoint the various difficulties inherent to the Super-Resolution problem for a variety of application settings often ignored in the past. We review many of the most recent and popular methods, and outline some of our recent work addressing these issues.

The organization of this article is as follows. In Section II we study Super-Resolution as an inverse problem and address related regularization issues. In Section III we analyze a general model for imaging systems applicable to various scenarios of Super-Resolution. In Section IV we describe three different application settings and our approaches to dealing with them. Specifically, we address

*Fig.43 High resolution document image after image blending with thresholding (upper half of the document only)*

## 5.7 Conclusion

Advantages of our system:

- Stitch through any direction
    - Traditional panorama stitch only horizontal or vertical consecutive images. But the direction for stitching in our project is not limited, any direction is possible.

- Stitch images of different scales
    - When taking photos for any part of the document, the distance between the document and the camera is not important. Users just need to ensure the focusing is correct.

- Stitch images of different orientation
    - Users can take photos in any angles, the document image can still be recovered after stitching.

- Stitch images of any order
    - Users need not to take photos in order, just need to ensure every parts of the document have been taken photos.

- Optimize for black and white documents
    - Since for a normal fax document, it is most likely a black and white document, our system can optimize to make the high resolution image sharper.

# Chapter 6 Project Progress

Here is our progress of the final year project.

| | |
|---|---|
| June, 2005<br>To<br>August, 2005 | 1. Familiar with digital image processing<br>2. Familiar with development platform of Symbian OS. |
| Sep, 2005 | 1. Try the sample program of Symbian OS.<br>2. Thinking FYP topic |
| October, 2005 | 1. FYP topic decided.<br>2. Try to implement program to access image raw data on Symbian phone using related API.<br>3. Try to implement program to decode and encode image on Symbian phone using related API.<br>4. Try to implement program to facimile things on Symbian phone using related API.<br>5. Study concept of image alignment and stitching (mainly focus on direct method).<br>6. Find sample code of SIFT (extract and match keypoints). |
| November, 2005 | 1. Study concept of image alignment and stitching (mainly focus on SIFT).<br>2. Study sample code of SIFT.<br>3. Make use of the sample code of SIFT.<br>4. Implement geometric transformation to transform images and merge the document on PC.<br>5. Prepare project presentation and demonstration.<br>6. Writing FYP report |

# Chapter 7 Difficulties

## 7.1 Limited speed of Symbian phone

Our project mainly involves image processing, especially image alignment and transformation. Image alignment involves a of search work. Image transformation involves editing of image data. These works need some time to process even on personal computer. As a result, we need to optimize it as much as possible or make use of more efficient so that it can be run on Symbian phone.

In addition, we need to make some assumption of our project in order to make it efficient. At this stage, we assume that the image of document is planer and it is a black and white photo.

## 7.2 Limited memory of Symbian phone

The images we use to process are 2 Mega pixel photos. As a result, memory usage is also one of our concerns. Unlike personal computer, the Symbian phone would not have much memory. It does not have large memory space and virtual memory for our working space. We need to make careful use of memory. Otherwise, the program cannot be run on Symbian phone.

## 7.3 Lack of clear documentation for fax API

So far, I still cannot implement faxing on Symbian phone. One of reason is that there is no clear documentation of faxing API. The documentation of SDK tool just states about the usage of the related object and its function.

In addition, a class related to faxing API has been deleted in the new version of Symbian OS. Originally, two faxing related API is found. Now there is only one left. And it seems this one is more difficult to use.

## 7.4 Unfamiliar with digital image processing

Before the FYP, both of us do not have any knowledge of image processing. As a result, when we studied the concept of image alignment and stitching, we need to spend much more time to study the basic concepts and notations of digital image processing.


*Fig.44 Difficulties encountered cartoon*

# Chapter 8 Contribution of work

## 8.1 Introduction:

In the following, I would describe my contribution of work in the final year project. I would divide my contribution into two main stages: preparation stage and implementation stage.

## 8.2 Preparation stage

In this stage, I have spent much on it, as I had no idea on both Symbian OS and image processing before the project. Although the topic was finalized so late, I learn much thing on Symbian OS.

In the summer, I start to learn what Symbian OS is and how to development platform for Symbian OS. During the learning, I had tried the sample code provided in the SDK tool. Beside, I had also ported a program, which originally run on PC, to Symbian phone. The program is about video decoding and encoding with codec o H.263. However, about although the quality of the video produced by the program on Symbian phone is not good.

As a result, I had got familiar with the development platform of Symbian OS.

### 8.2.1 Testing of Symbian API

After the objective has been decided, I tried to implement related functions on the Symbian phone. The related function includes how to access and modify raw

data of image, how to decode and encode image to specified format like JPEG, how to take photo using the onboard camera and how to do faxing. The Symbian SDK provides related API for all of them. Among from faxing, I successfully implement program to use the functions. However, I still can't implement write code to implement faxing function on API. The main reason is that the documentation of the SDK does not provide enough information for me.

### 8.2.2 Learning of image alignment and stitching

Apart from trying the related function on Symbian phone, I also need to study image alignment and stitching together with my partner. As I did not have any experience of image processing before, I need to spend much time on it.

During studying, I first came across the direct method. Then when we know the existence of the sample code of SIFT. We change to study the feature base registration as SIFT can extract stable keypoints. At the same time, my partner try to make use of the sample code, I study the theory of SIFT to give support to my partner. However, until the end of the semester, I just get the general concept of it.

## 8.3 Implementation stage

After we have implement the stitching on PC, I try to port the program to the Symbian phone directly. However, I found that it cannot be done because the source code is too complex and it is written in C# where C++ is the native language of Symbian OS. Instead, we design to simple and optimize the code first.

## 8.4 Conclusion

In the final year project, I had learnt many things. I had learnt how to build application program for Symbian phone. I also studied many knowledge about image processing, particular for image alignment and stiching.

# Chapter 9 Conclusion

In this project, we have studied a lot of image alignment and stitching algorithms, and implemented them on PC. Through the implementation and testing on PC, we also learned the windows programming development environment.

The major difficulties we faced is the balance between accuracy and speed of alignment and stitching. In order to stitch and form a high-resolution sharp document image, we need to align the images correctly. More number of much detailed keypoint recorded, higher the chance to match the correct keypoints. However, the time taken is directly proportion to the number and details of keypoint recorded. Therefore, it is hard to get the balance.

We proposed a new method of stitching and our system has several advantages over traditional panorama. Images of different orientations and scales, or even they are not in order, aligned neither in horizontal direction nor vertical direction, those images can still be stitched correctly. Moreover, we have an optional blending optimization for making high resolution black and white documents.

Last but not the least, although our system is focusing on reconstruction of document images, documents that contain images or photographs are still possible to stitch using our system.

# Chapter 10 Future works

## 10.1 Optimization for Symbian Platform

We have already proven the feasibility of our project on PC platform. However, considering the processing power of mobile phones and the specialty of Symbian Platform, we will encounter more restrictions than writing program for PC. Therefore, we need to optimize our program so that it can run more efficiently on Symbian mobile phones. For example, mobile phones are weak at floating point operations, thus we would like to modify our program so that most of the operations are integral operations instead of floating point operations.

## 10.2 Restoration based on video

In our project, image alignment plays an important role. Thus if more constraints are given for matching the feature points, higher the accuracy can be achieved. Since the nature of video frames is basically a consecutive sequence of images, we may be able to use this nature to establish constraints in image alignment. In other words, our product may allow user to capture the document in a video clip and carry out restoration based on that video clip. However, there may be a trade off of quality and accuracy. Although image alignment may become more accurate if constraints are added, quality of video clips on mobile phones are still a concern.

## 10.3 Automatic lens correction

Since the lens distortion are different for different camera lens, it is better to have an automatic lens correction in our project, so that our project can be more adaptable to any models of mobile phone. It is found that in 1999, Mr Harri Ojanen already showed that even with inexpensive zoom lenses, near perfect images can be obtained by using a correction algorithm based on a simple mathematical model and without any special measuring tools. Therefore, it should be possible to make automatic lens correction available in our product.

## 10.4 A better blending technique

At present, we have used bilinear interpolation and averaging mask to blend the final image, so that fewer defects can be seen on the merged edge. However, there are still rooms for improvement. We are now seeking a better algorithm, or we may develop by ourselves a new algorithm to enhance the resulting image quality.

## 10.5 Study the feasibility of employing techniques of super-resolution

In our project, sharpness of the image is the most important thing that we concern. Technique of super resolution is one of the ways to increase sharpness and details of digital images but it is hard to achieve on mobile phones. Therefore, we will study first the feasibility of employing such technique and see if it is suitable for us.

## 10.6 Conclusion

In next semester, we will not just complete the implementation of our program on Symbian, but also optimize it to best suit the equippment of mobile phones. Furthermore, we would like to add more values to our project. We are going to study the feasibilities of many other ideas that are related to image processing, such as super-resolution technique.

# Chapter 11 Acknowledgement

We would like to express our thank to our final year project superviser, Professor Michael R. Lyu. He has given us many suggestion and provided equiment thoughout the project.

We would like to think Mr. Edward Yau, who give us a lot of innovative idea and suggestion for our project.

# Chapter 12 Reference

[1] R. Szeliski, "Image alignment and stitching: A tutorial". Technical Report MSR-TR-2004-92, Microsoft Research, 2004.

[2] T. Szeliski, "Image alignment and stitching. In Mathematical Models of Computer Vision: The Handbook".

[3] H. Y. Shum and R. Szeliski, "Construction and refinement of panoramic mosaics with global and local alignment". Microsoft Research.

[4] Y. Deng and T. Zhang, "Generating Panorama Photos", ITCom, 2003.

[5] K. Mikolajczyk, A. Zisserman and C. Schmid, "Shape recognition with edge-based features".

[6] J. S. Beis and D. G. Lowe, "Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces".

[7] M. Brown and D, Lowe. "Invariant Features from Interest Point Groups".

[8] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors". In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003.

[9] D.G Lowe, "Object recognition from local scale-invariant features", In Seventh International Conference on Computer Vision, pp.1150-1157, Kerkyra, Greece, 1999.

[10] M. Brown and D. G. Lowe. "Recognizing panoramas", In Ninth International Conference on Computer Vision, pp.1218-1225, Nice, France, 2003.

[11] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", In the International Journal of Computer Vision, 2004.

[12] H. P. A. Lensch, W. Heidrich and H-P. Seidel, "A Silhouette-Based Algorithm for Texture Registration and Stitching", In Elsevier Science, 2004.

[13] V. Rankov, R. J. Locke, R. J. Edens, P. R. Barber and B. Vojnovic, "An algorithm for image stitching and blending".

[14] D.G. Lowe. "Local Feature View Clustering for 3D Object Recognition", In the IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii 2001.

[15] M. Varma and A. Zisserman, "Texture Classification: Are Filter Banks Necessary?"

[16] S. Se, D. Lowe and J. Little, "Vision-based Mapping with Backward Corrections", In the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems EPFL, Lausanne, Switzerland, pp.153-158, 2002.

[17] S. Se, D. Lowe and J. Little, "Global Localization using Distinctive Visual Features", In the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems EPFL, Lausanne, Switzerland, pp.226-231, 2002.

[18] J. Sivic and A. Zisserman, "Video Google: A Text Retrieval Approach to Object Matching in Videos", In the Ninth International Conference on Computer Vision (ICCV 2003) 2-Volume Set, 2003.

[19] T. Lindeberg, "Scale-space: A framework for handling image structures at multiple scales", In Proc. CERN School of Computing, Egmond aan Zee, The Nether lands, pp.8-21, 1996.

[20] S. Nene and S. K. Nayar. "A simple algorithm for nearest neighbor search in high dimensions. IEEE Transactions on Patern Analysis and Machine Intelligence", 19(9), pp.989-1003,1997.

[21] S. Farsiu, D. Robinson, M. Elad, P. Milanfar, "Advances and Challenges in Super-Resolution".

[22] A. Zomet and Shmuel Peleg, "Applying Super-Resolution to Panoramic Mosaics".

[23] L. G. Brown, "Aurvey of image registration techniques", Computing Surveys,

24(4), pp.325-376, 1992.

[24] C. Harris and M. J. Stephens, “A combined corner and edge detector”, In Alvey Vision Conference, pp.147-152, 1988.

[25] K. Mikolajczyk and C. Schmid, “Indexing based on scale invariant interest points”, In Eighth International Conference on Computer Vision (ICCV 2001, pp.525-531, Vancouver, Canada, 2001.

[26] K. Mikolajczyk and C. Schmid, “An affine invariant interest point detector”, In Seventh European Conference on Computer Vision (ECCV 2002), pp.128-142, Springer-Verlag, Copenhagen, 2002.

[27] S. Nene and S. K. Nayar, “A simple algorithm for nearest neighbor search in high dimensions”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(9), pp.989-1003, 1997.

[28] S. Peleg, “Elimination of seams from photomosaics”, Computer Vision, Graphics, and Image Processing, 16, pp.1206-1210, 1981.

[29] C. Schmid, R. Mohr and C. Bauckhage, “Evaluation of interest point detectors”, International Journal of Computer Vision, 37(2), pp.151-172, 200.

[30] C. Schmid and R. Mohr, “Local Grayvalue Invariants for Image Retrieval”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(5), pp.530-535, May 1997.

[31] A. Baumberg, “Reliable Feature Matching Across Widely Separated Views”, In Proceedings of the International Conference on Comptuer Vision and Pattern Recognition, pp.774-781, 2000.

[32] R. C. Gonzalez and R. E. Woods, “Digital Image Processing”, Second Edition, Prentice Hall, 2002.

[33] R. Szeliski and H.-Y. Shum, “Creating full view panoramic image mosaics and texture-mapped models”, Computer Graphics (SIGGRAPH’ 97 Proceedings), pp.251-258, 1997.

[34] Q. Tian and M. N. Huhns, "Algorithm for subpixel registration", Computer Vision, Graphics, and Image Processing, 35, pp.220-233, 1986.

[35] G. Stein, "Lens distortion calibration using point corresponces", In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '97), pp.602-608, San Juan, Puerto Rico, 1997.

[36] T. Nakao, A. Kashitani and A. Kaneyoshi, "Scanning a document with a small camera attached to a mouse", In IEEE Workshop on Applications of Computer Vision (WACV '98), pp.63-68, IEEE Computer Society, Princeton, 1998.

[37] M. EL-Melegy and A. Farag, "Nonmetric lens distortion calibration: Closed-form solutions, robust estimation and model selection", In Ninth International Conference on Computer Vision (ICCV 2003), pp.554-559, Nice, France, 2003.

[38] D. Capel and A. Zisserman, "Super-resolution enhancements of text image sequences", In Fifteenth International Conference on Pattern Recognition (ICPR '2000), pp.600-605, IEEE Computer Society Press, Barcelona, Spain, 2000.

[39] G. Borgefors, "Distance transformations in digital images", Computer Vision, Graphics and Image Processing, 34(3), pp.227-248, 1986.

[40] J. R. Bergen, P. Anandan, K. J. Hanna and R. Hingorani, "Hierarchical model-based motion estimation", In Second European Conference on Computer Vision (ECCV '92), pp.237-252, Springer, Santa Margherita Liguere, Italy, 1992.

[41] E. Banissi, "Basics of Computer Graphics", 2002

[42] Michael R. Bax, "Real-Time Lens Distortion Correction: 3D Video Graphics Cars Are Good for More than Games", Stanford ECJ, 2004

[43] Xiang Wnag, Reinhard Klette, and Bodo Rosenhahn, "Geometric and Photometric Correction of Projected Rectangular Pictures", New Zealand

[44] Marco Winzker and Uwe Rabeler, "Electronic Image Correction for Projection Displays", Germany, 2002

[45] Harri Ojanen, "Automatic Correction of Lens Distortion by Using Digital Image Processing", July 10, 1999

[46] Jun-Wei Hsieh, "Fast Stitching Algorithm for Moving Object Detection and Mosaic Construction", Taiwan, 2003

[47] Luke Ledwich and Stefan Williams, "Reduced SIFT Features For Image Retrieval and Indoor Localisation", Australia, 2005

[48] Vladan Rankov, Rosalind J. Locke, Richard J. Edens, Paul R. Barber and Borivoj Vojnovic, "An algorithm for image stitching and blending", United Kingdom, 2005

[49] Patrick Vandewalle, Sabine Susstrunk, and Martin Vetterli, "A Frequency Domain Approach to Registration of Aliased Images with Application to Super-Resolution", 2005

[50] Clement Fredembach, Michael Schroder and Sabine Susstrunk, "Eigenregions for Image Classification", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, No. 12, December 2004