



***Department of Computer Science & Engineering***

**The Chinese University of Hong Kong**

***2005 / 2006 Final Year Project***

**Final Report**

***LYU 0503***

**Document Image Reconstruction on Mobile**

**Using Onboard Camera**

***Supervisor***

**Professor Michael R. Lyu**

***Group Members***

**Leung Man Kin**

**Ng Ying Kit**

**Prepared by Leung Man Kin  
20<sup>th</sup>, April, 2006**

## **Abstract**

This report summarizes all of our works in the final year project which is entitled Document Image Reconstruction on Mobile Using Onboard Camera. The objective of our project is to reconstruct a high resolution document image using photos of different parts of the document, which are taken by onboard camera of mobile phone. This report would state the motivation, background information, experimental results 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 discuss the platforms we have used for project development. The first one is the PC platform for testing of proto-type. The second one is the Symbian Operating System, which is the actual platform on mobile phone to implement our project. Next, we would discuss about the algorithms used to implement to reconstruct the document, namely, image alignment and stitching. There are many different types of such algorithms. In our project, we are focusing on feature-based registration with manually assistance. Following is the implement of our project in details and some experimental results. At last, we would finish the report by stating the difficulties encountered and improvements.

# **Content**

<b>Abstract .....</b>	<b>2</b>
<b>Content.....</b>	<b>3</b>
<b>Chapter 1 Introduction .....</b>	<b>6</b>
1.1 Motivation.....	6
1.2 Project Objective.....	7
1.3 Project Equipments .....	8
<b>Chapter 2 Testing on PC .....</b>	<b>9</b>
2.1 Introduction.....	9
2.2 Cygwin Platform & Mono .....	10
2.3 C# Programming.....	12
2.4 Matlab Programming .....	14
2.5 Conclusion .....	16
<b>Chapter 3 Symbian OS.....</b>	<b>17</b>
3.1 Introduction.....	17
3.2 Development environment.....	18
3.3 Limitation in Symbian phones.....	21
3.4 Overview of our focus library.....	23
3.5 Testing on N90 .....	36
3.6 Conclusion .....	39

<b>Chapter 4</b>	<b>Concept of Image Alignment .....</b>	<b>40</b>
4.1	Introduction.....	40
4.2	Direct Alignment .....	41
4.3	Feature-based Registration .....	45
4.4	Conclusion .....	47
<b>Chapter 5</b>	<b>Implementation in Our Project .....</b>	<b>48</b>
5.1	Overview.....	48
5.2	Take photos of the document .....	50
5.3	Do lens correction of the photos .....	59
5.4	Mark matching points of the photos manually .....	69
5.5	Stitch the photos.....	75
5.6	Optimize the Result .....	81
5.7	Experimental Results .....	86
5.8	Conclusion .....	107
<b>Chapter 6</b>	<b>Project Progress .....</b>	<b>110</b>
<b>Chapter 7</b>	<b>Difficulties.....</b>	<b>112</b>
7.1	Limited speed of Symbian phone .....	112
7.2	Limited memory of Symbian phone .....	112
7.3	Difficult to debug.....	113
7.4	Lack of clear documentation for fax API .....	113
7.5	Unfamiliar with digital image process.....	114
<b>Chapter 8</b>	<b>Contribution of work .....</b>	<b>115</b>



8.1	Introduction.....	115
8.2	Preparation stage.....	115
8.3	Implementation stage.....	116
8.4	Conclusion .....	118
 <b>Chapter 9 Conclusion .....</b>		<b>119</b>
 <b>Chapter 10 Further Improvements .....</b>		<b>120</b>
10.1	Optimization for Symbian Platform .....	120
10.2	Restoration based on video .....	120
10.3	Automatic lens correction .....	121
10.4	A better blending technique .....	121
 <b>Chapter 11 Acknowledgement .....</b>		<b>122</b>
 <b>Chapter 12 Reference .....</b>		<b>123</b>

# **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.

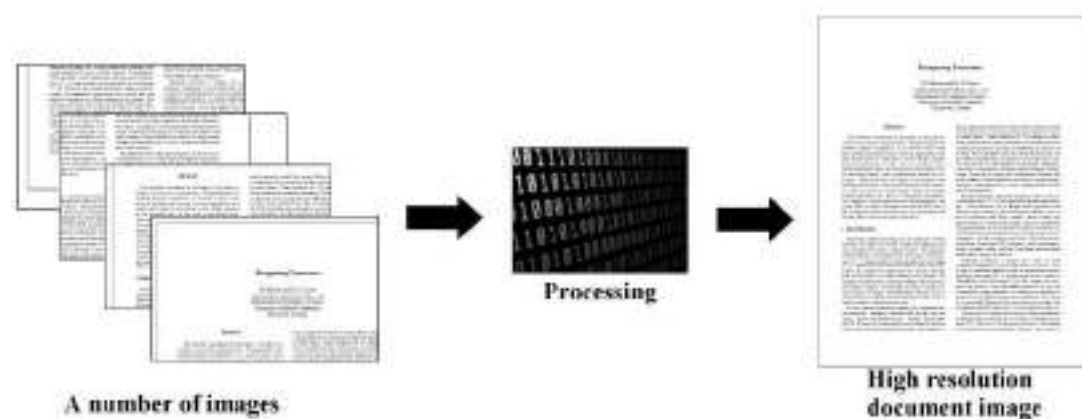
We think the use of camera should not be limited to these functions and the market of software on mobile phone is potentially large, so we think of a way to add more values to the camera of phones.



*Fig1.1 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.



*Fig1.2 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 Equipments**

The equipment we use is Nokia 6600, Nokia 6630 and Nokia N90. 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 8.0a. The operation system of Nokia N90 is Symbian OS 8.1. Nokia 6600 and Nokia 6630 are only used to do testing for Symbian API. For example, we have done testing for image accessing, image decoding, image encoding, camera controlling and faxing.

The document restoration is implemented on Nokia N90.



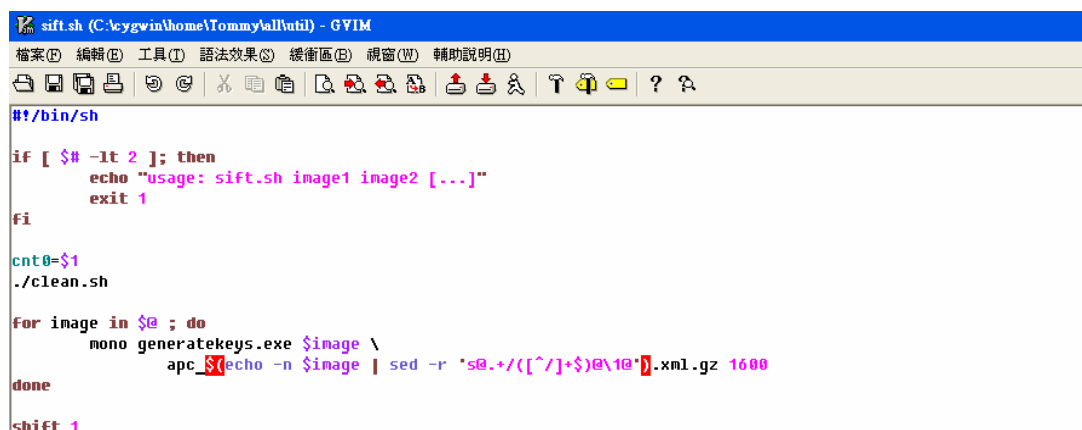
*Fig1.3 Project equipments – Nokia 6600, 6630 & N90*

## Chapter 2 Testing on PC

### 2.1 Introduction

Before implementing our project on the mobile phone platform, we tested our algorithms on the PC platform. We made use of proto-type of our program to evaluate the feasibilities, efficiencies and performance of different algorithms. It saved our time because the processing power of a PC is much higher than a mobile phone.

We used C# programming languages to write the program of image alignment and use mono to compile the code under Cygwin platform. In order to compile and run the programs efficiently, we wrote some shell scripts to handle the process. We also used Matlab programming language to write the program of image stitching.



```
sift.sh (C:\cygwin\home\Tommy\all\util) - GYIM
檔案(F) 編輯(E) 工具(T) 語法效果(S) 緩衝區(B) 視窗(W) 輔助說明(H)
#t/bin/sh

if [ $# -lt 2 ]; then
    echo "usage: sift.sh image1 image2 [...]"
    exit 1
fi

cnt0=$1
./clean.sh

for image in $@ ; do
    mono generatekeys.exe $image \
        apc echo -n $image | sed -r "s@.+/(^[^/]+)$@\\1@''.xml.gz 1600
done

shift 1
```

*Fig2.1 Shell scripts to run the programs*

## **2.2 Cygwin Platform & Mono**

### **2.2.1 Cygwin Platform**

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

- A DLL (cygwin1.dll), which can act 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.

### **2.2.2 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 libraries.
- A compiler for the C# programming language.

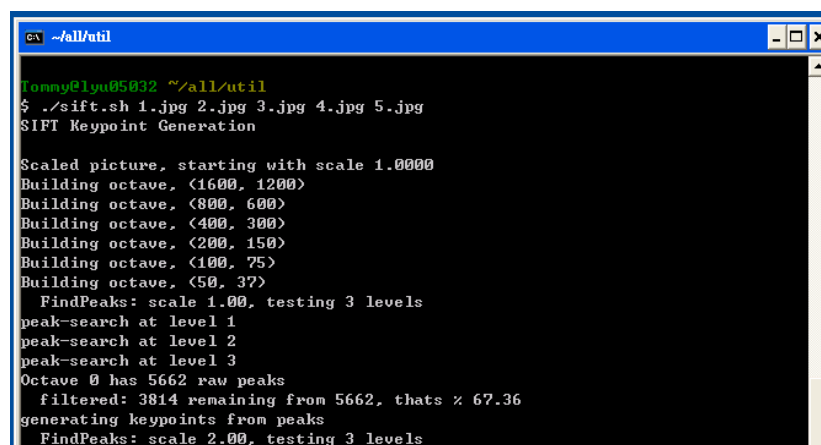
Windows has compilers, which target the virtual machine from a large number of programming languages. The languages include Managed C++, Java Script, Eiffel,

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 matrix 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.

### **2.2.3 Advantages of Cygwin Platform & Mono**

1. Cygwin Platform provides a Linux-like development platform. As we had already been familiar with these types 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.



```

Cygwin - ~/all/util
Tommy@lyu05032 ~/all/util
$ ./sift.sh 1.jpg 2.jpg 3.jpg 4.jpg 5.jpg
SIFT Keypoint Generation

Scaled picture, starting with scale 1.0000
Building octave, (1600, 1200)
Building octave, (800, 600)
Building octave, (400, 300)
Building octave, (200, 150)
Building octave, (100, 75)
Building octave, (50, 37)
FindPeaks: scale 1.00, testing 3 levels
peak-search at level 1
peak-search at level 2
peak-search at level 3
Octave 0 has 5662 raw peaks
filtered: 3814 remaining from 5662, thats x 67.36
generating keypoints from peaks
FindPeaks: scale 2.00, testing 3 levels

```

*Fig2.2 Outlook of Cygwin Platform*

## **2.3 C# programming**

### **2.3.1 C# programming language**

C# is an object-oriented programming language developed by Microsoft as part of their .NET Framework. C# is based 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 Framework. C# is the programming language which can mostly reflect the Framework and it depends strongly on this framework. There are no unmanaged C# programs. The data types of C# are objects of the corresponding .NET types. C# shows 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 objects are 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 allows single inheritance. However, a class can implement any number of abstract interfaces. It can simplify the implementation of runtime.
- C# is more type safe than C++. There is only one default implicit conversion called safe conversions. It includes widening of integers and



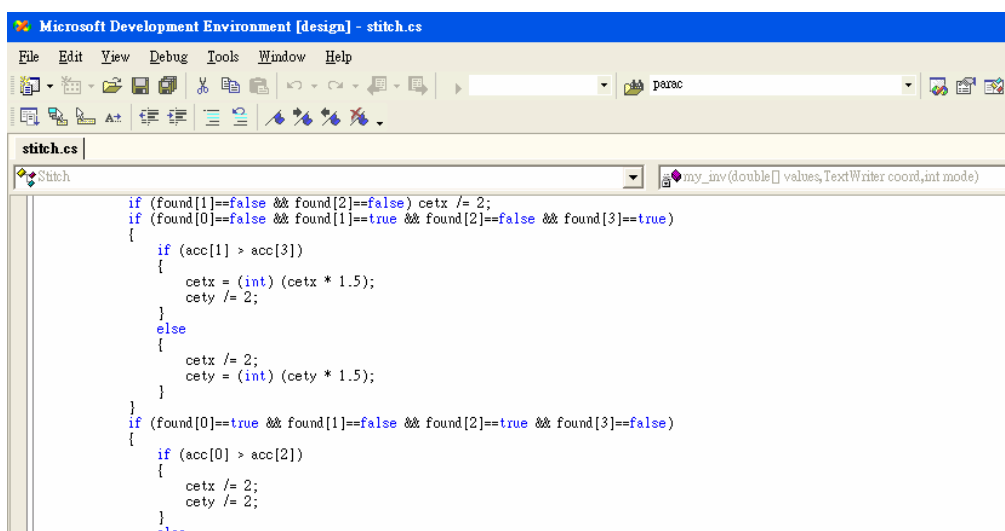
conversion from derived types to base types. There are no implicit conversions between Booleans and integers as well as enumeration members and integers. There are also no void pointers. Any user-defined implicit conversion must be explicitly marked.

- 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#

### **2.3.2 Advantages of C# programming**

1. C# is an 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.



*Fig2.3 C# programming languages*

## **2.4 Matlab programming**

### **2.4.1 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 Development Environment: It provides set of tools and facilities to help user to get familiar with Matlab. Most of them have 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, and files. The interface can be changed as the user like.
- The Mat lab 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 supports many matrix and array operations. It also has 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 library, 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 specific classes of problems

### **2.4.2 Advantages of Matlab programming**

1. It provides many matrix operations. Matrix operations are essential in image processing. It enables 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 picture using a function. It helps 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.

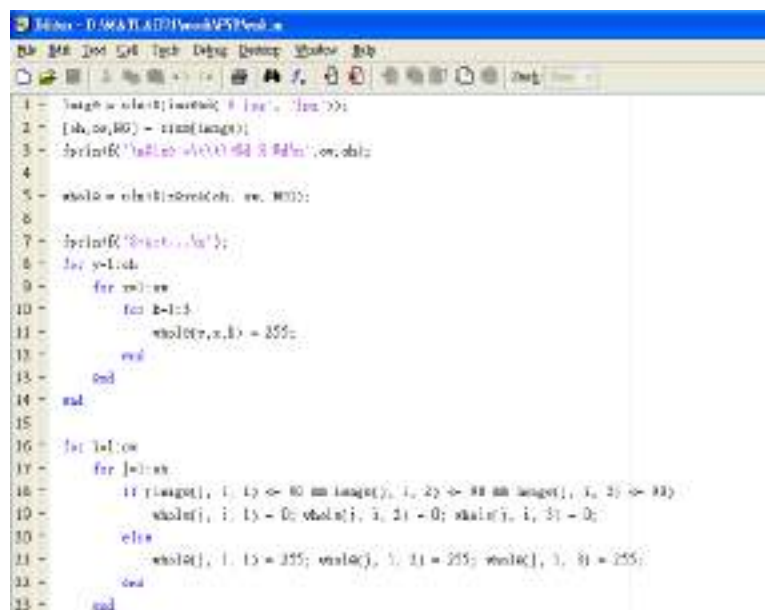


Fig2.4 Matlab programming language

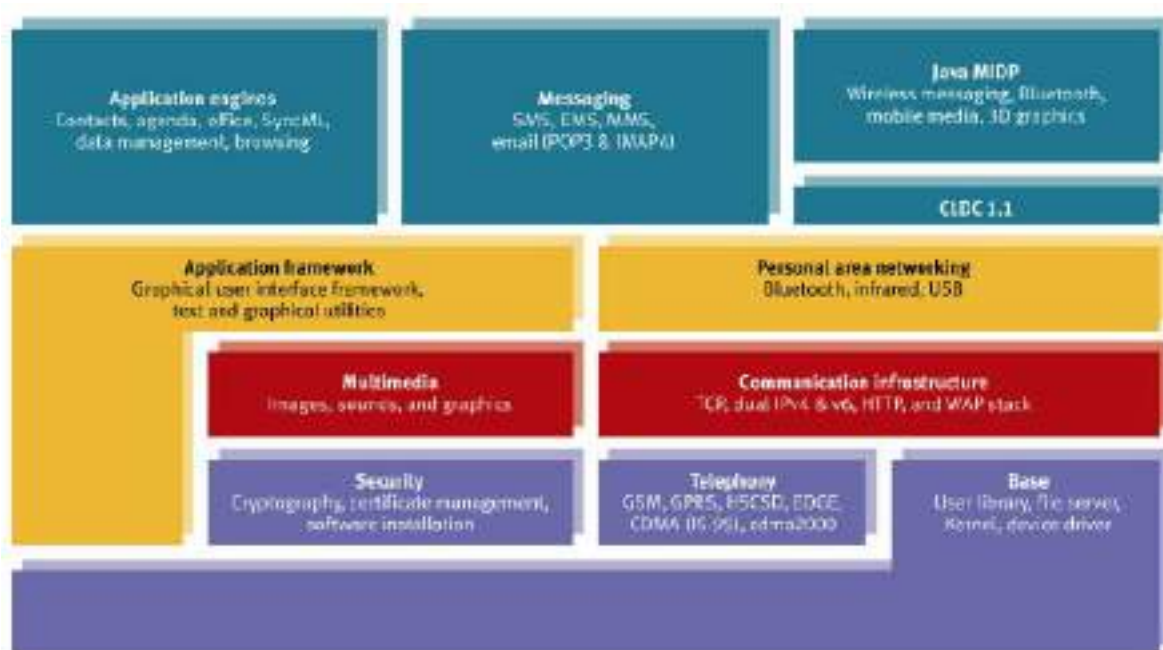
## 2.5 Conclusion

In this chapter, we introduced the testing environment on PC. And after the testing of our proto-type on PC platform, we have chosen the most suitable image alignment and stitching algorithms for our project (Details of algorithms tested will be discussed in later chapters). The next chapter we will focus on the actual platform we want to implement our project, i.e. Symbian Operating System.

## **Chapter 3 Symbian OS**

### **3.1 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 telephony support, communications protocols, data management and synchronization, international support, multimedia support, graphics support, a low-level graphical user interface framework, etc.



*Fig3.1 Architecture of Symbian OS 8.0*

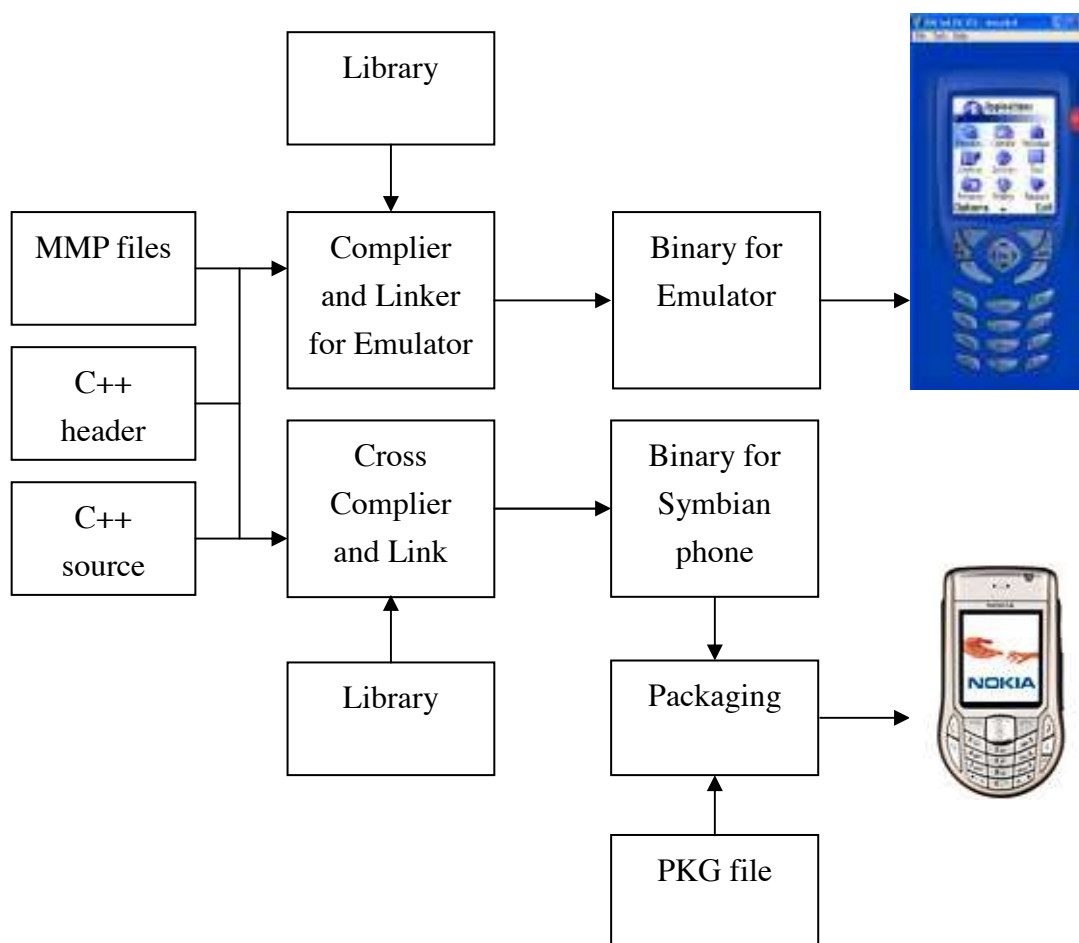
The following focuses on how to write programs for Symbian OS and limitations of mobile phone. In addition, we would focus on how we make use of libraries so that we can perform the following jobs:

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

## 3.2 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.3.2 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 2<sup>nd</sup> 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.

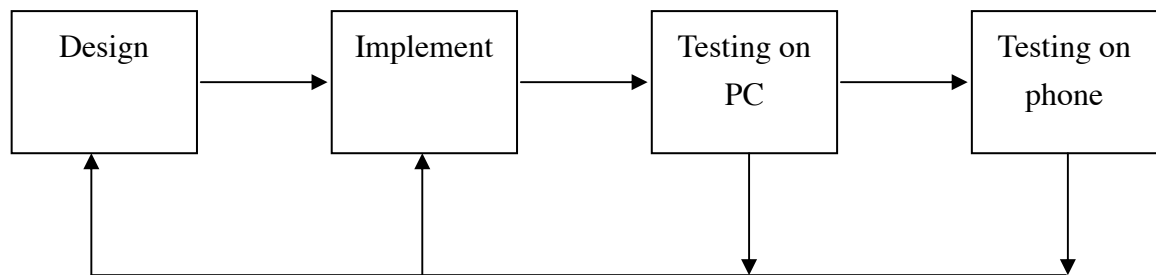
Firstly, we can use the application wizard provided in SDK to create a simple sample program. It would create all corresponding files for compiling the application.

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 compile 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:




*Fig3.3 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.




### **3.3 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 Shared Memory for Storage: 6MB Unlimited Jar Size	
Camera	VGA Camera with 2x digital zoom	
Processor	32-bit RISC CPU based on ARM-9 series, 104 MHz	

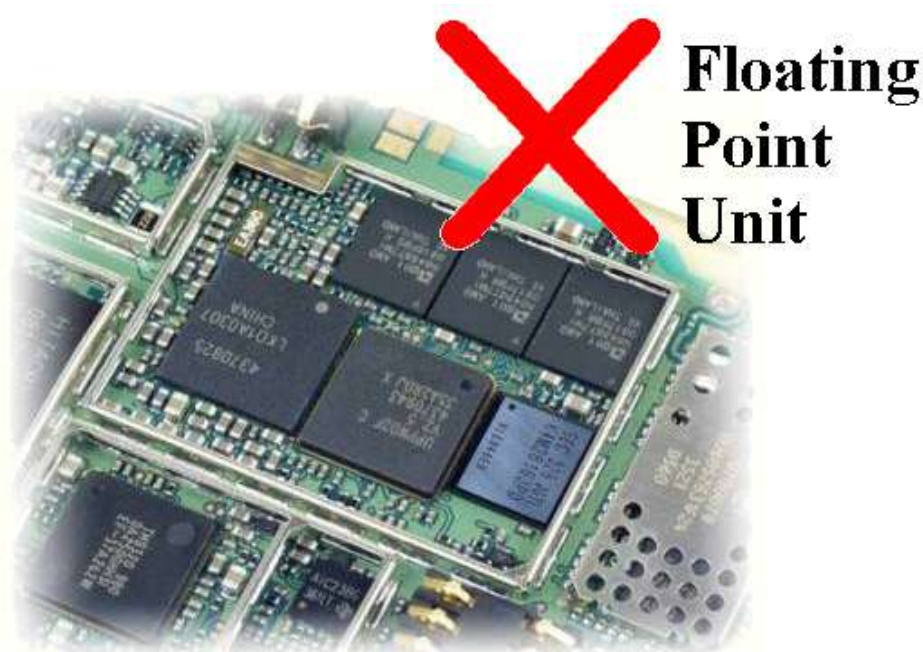
Nokia 6630 Technical Specification		
Operating System	Symbian OS v8.0a	
Developer Platform	Series 60 2 <sup>nd</sup> Edition, Feature Pack 2	
Memory	Heap: Unlimited Shared Memory for Storage: 10MB Unlimited Jar Size	
Camera	1.3 mega pixel camera with 6x digital zoom	
Processor	32-bit RISC CPU based on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

Nokia N90 Technical Specification		
Operating System	Symbian OS v8.1a	
Developer Platform	Series 60 2 <sup>nd</sup> Edition, Feature Pack 3	
Memory	Heap: Unlimited Shared Memory for Storage: 30MB Unlimited Jar Size	
Camera	2 mega pixel camera with 20x digital zoom 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 time 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 in Nokia 6630 and Nokia N90, it is 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.



*Fig3.4 Nokia mobile phone circuit board. It got no floating point unit.*

### **3.4 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 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 an object called CWriteFaxFile.

### **3.4.1 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 an 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 created 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.

However, there are several display modes for the bitmap. When the pixel value is accessed using the pointer directly. The format of data stored would depend heavily on the display modes. There are three common display mode used in our project. They are listed in the following table.

Display mode	Description
EGray256	256 gray scales display mode (8 bits per pixel)
EColor64K	64,000 colour display mode (16 bits per pixel)
EColor16M	True colour display mode (24 bits per pixel)

Display mode EGray256 is used when gray scale image is being processed. 8 bits is used to represent gray level for each pixel. Data type **char** is used to access the data. Here is the example to access pixel value at position (x, y).

```
char *data = (char *)image->DataAddress();

Tint pos = y * width + x;

GrayLevel = data[pos];
```

Display mode EColor64K is used when the screen is being processed. 16 bits is used to represent colour components for each pixel. Data type **TUint16** is used for color components to access the data. 5 bits are allocated to red. 6 bits are allocated to green. 5 bits are allocated to blue. Here is the example to access pixel value at position (x, y).

```
TUint16 *data = (TUint16 *)image->DataAddress();

Tint pos = y * width + x;

Red = (data[pos] >> 11) & 0x001F;

Green = (data[pos] >> 5) & 0x003F;
```

```
Blue = data[pos] & 0x001F;
```

Display mode EColor16M is used when color image is being processed. 24 bits is used to represent colour components for each pixel. Data type **char** is used for each color component to access the data. Here is the example to pixel value at position (x, y).

```
char *data = (char *)image->DataAddress();  
Tint pos = y * width + x;  
Red = data[pos * 3 + 2];  
Green = data[pos * 3 + 1];  
Blue = data[pos * 3 + 0];
```

### **3.4.2 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);
```

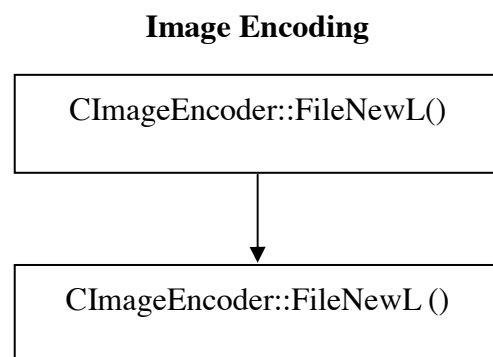
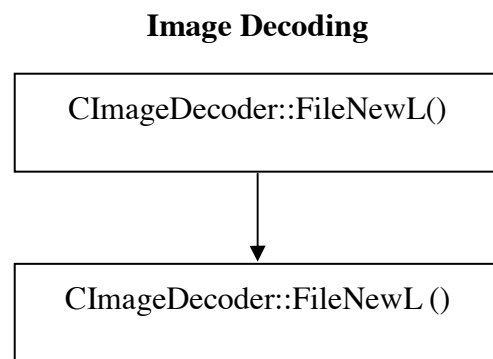
After that, we obtain a CFbsBitmap, which is the raw data of the image from the function.

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);
```



*Fig3.5 Objects of Image decoding & encoding*



### **3.4.3 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 a 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:

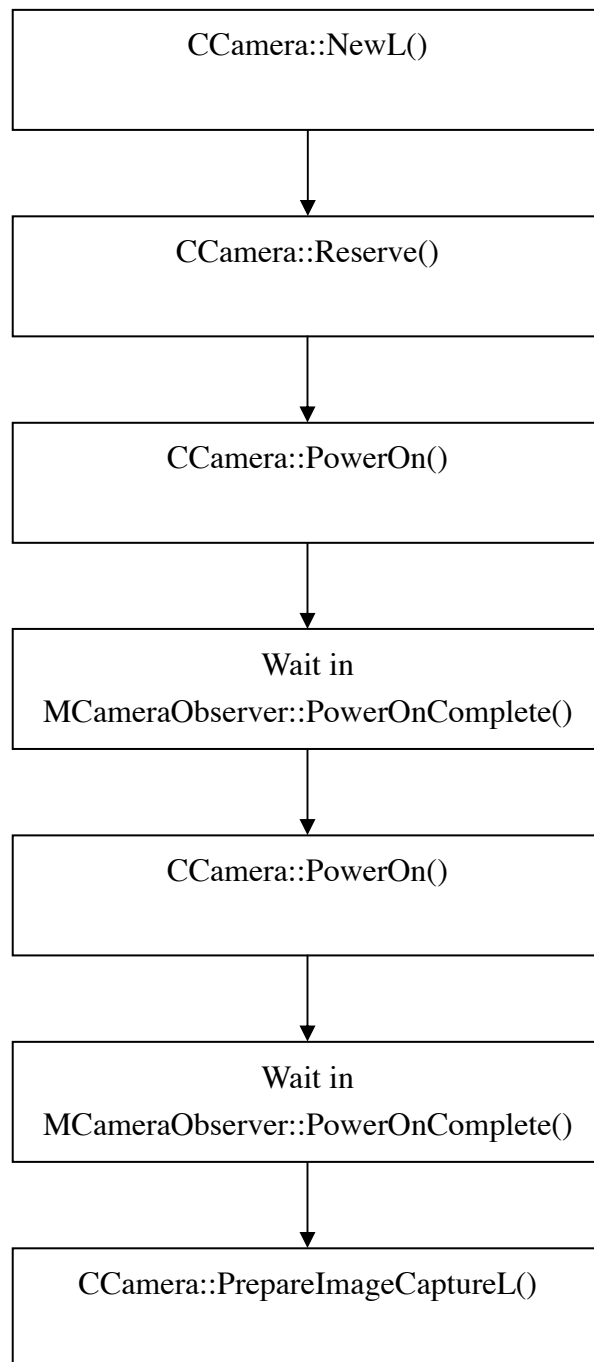
```
void CCamera::PrepareImageCaptureL(TFormat aImageFormat, TInt  
aSizeIndex);
```

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

```
void CCamera::CaptureImage();
```

The image data would be return in the

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



*Fig3.6 Flow of controlling onboard camera*

### **3.4.4 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 Symbian 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:

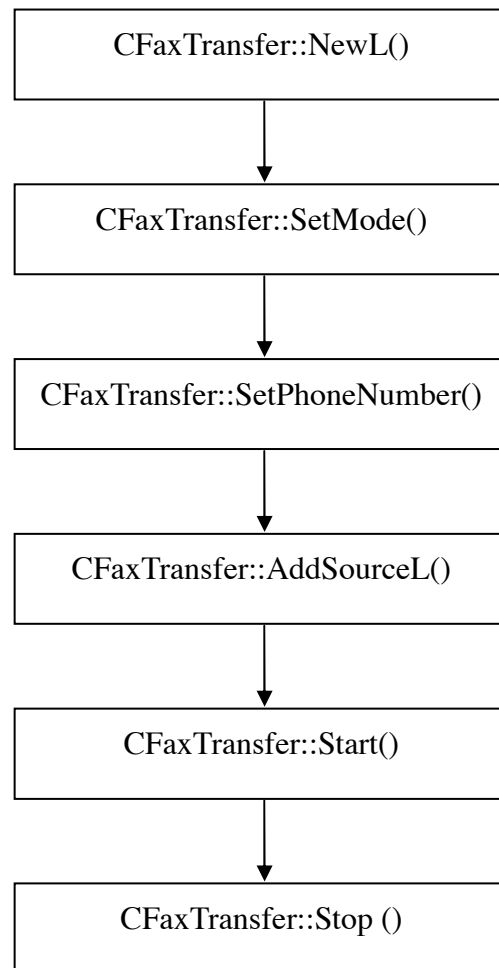
```
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore);
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore,
                              TInt aStartPage);
void CFaxTransfer::AddSourceL(const TFileName& aFaxPageStore,
                              TInt aStartPage, TInt aEndPage);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

After all these things have 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();
```



*Fig3.7 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.

### **3.4.5 File operation**

In Symbian OS, before the file is read or written, it is essential to connect to the file server using class RFs. File server is connected by:

```
TInt RFs::Connect(TInt  
aMessageSlots=KFileServerDefaultMessageSlots);
```

After file server is connected, file can be opened using class RFile. The file is opened via the file server by:

```
TInt RFile::Open(RFs& aFs,const TDesC& aName,TUint  
aFileMode);
```

If the file does not exist, the above method would be failed. Then the file should be created and opened by

```
TInt RFile::Create(RFs& aFs,const TDesC& aName,TUint  
aFileMode);
```

For the above two methods, the parameter aFileMode is specified for the mode for opening file. The two important parameters are

<b>Parameter</b>	<b>Description</b>
EFileRead	The file is used to read.
EFileWrite	The file is used to read and write.

After a file is opened, we want to read and write text for each line. Therefore, we make use of the class TFileText, which enable us to do this. The opened should be set to using class TFileText by

```
void TFileText::Set(RFile& aFile);
```

To read a line from the file, the parameter aFileMode in opening and creating the file should be set accordingly. The following method is used

```
TInt TFileText::Read(TDes& aDes);
```

To write a line from the file, the parameter aFileMode in opening and creating

the file should be set accordingly. The following method is used.

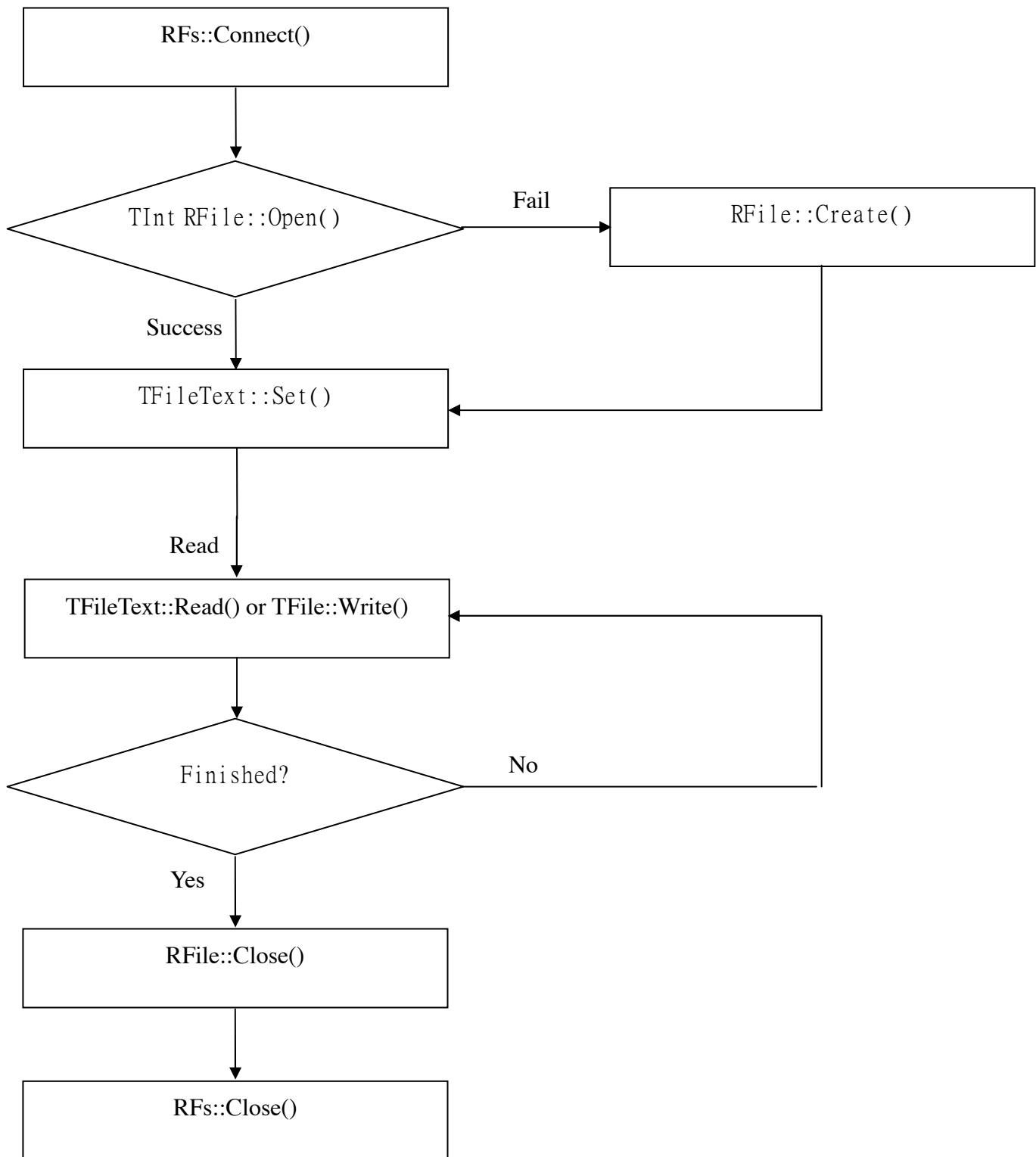
**TInt TFileText::Write(const TDesC& aDes);**

After the file is finished to read and write, the file needs to be closed by

**void RFile::Close();**

And the file server needs to be closed by

**void RFs::Close();**



*Fig3.8 Flow of File Read / Write*

### **3.5 Testing on N90**

Our project involves many operations on image processing. Therefore, memory is very important for us. There are 2 tests done to find how much memory can be allocated.

#### **Allocate memory using *malloc* or *User::Alloc***

In Symbian C++, *malloc* and *User::Alloc* in fact are the same things. They are used to allocate the memory of the current thread of the application. An experiment has been conducted to check how much memory can be allocated using these functions. Here is the pseudocode used in the experiment.

```
char *tmp
int i = 0;
while(1) {
    tmp = malloc(SIZE); //Allocate memory
    if(tmp == NULL) {
        break;
    }
    i++;
}
display i;
```

The product of **i** and **SIZE** is the amount of memory which can be allocated.

Different values of **SIZE** have been tried. It is obvious that **i** would decrease with **SIZE**. And we actually found out that the product of **i** and **SIZE** is 1,000,000. It means that the amount of memory which can be allocated is about 1 MB.

In fact, there is a parameter called **EPOCHEAPSIZE**, which can be set in **MMP**



file. It is expected to the heap size. However, we found out that the experiment result would not change whatever the value of the parameter is.

### **Memory used to open image**

The class CFbsBitmap is used to open or create an image. An experiment was conducted to find out how many image can be opened. In the experiment, we try to found out how many 1600 x 1200 grayscale image can be created. Here is the pseudocode used in the experiment.

```
imgSize.iWidth = 1600;
imgSize.iHeight = 1200;
for(i = 0; ; i++) {
    outputImage = new CFbsBitmap();
    //Allocate memory
    if(outputImage->Create(imgSize, EGray256) != KErrNone) {
        break;
    }
}
Display i;
```

The value of i is the number of the image which can be opened at the same. In the experiment, we found out that the value of i is about 8 – 10. It means that we can open 8 – 10 1600 x 1200 gray scale image. In addition, we can estimate the memory which can be used for image. It is about  $8 * 1600 * 1200 = 15$  MB.

### **Different between these two experiments**

We can see that there are big different between these two experiment. Only 1 MB can be allocated using malloc and User::Alloc. But about 15 MB can be

allocated using the class CFbsBitmap.

It is expected that the difference comes from the client-server architecture of the Symbian Operating System. When an application wants to do file operation, use camera and etc, the application need to act as a client and request server for their wanted services. This is the meaning of the client-server architecture. The client and server run in different threads. The client can only request a specific service using API which is defined by the server.

In the first experiment, the memory is allocated using malloc() and User::Alloc(). These functions are used to allocate the memory in the current thread. As a result, the memory should be in the client side.

In the second experiment, the memory is allocated using the class CFbsBitmap. It is the API defined by the bitmap server. As a result, the memory should be in the server side.

These two experiments seem show that more resource can be allocated in the server.

### **Contribution of these experiments for our project**

In our project, as much memory needed is for the image processing. As a result, these experiments help us to find out that we should use CFbsBitmap class to allocate memory for the image. And we should minimize the use of the memory allocated by malloc and User::Alloc.

### **3.6 Conclusion**

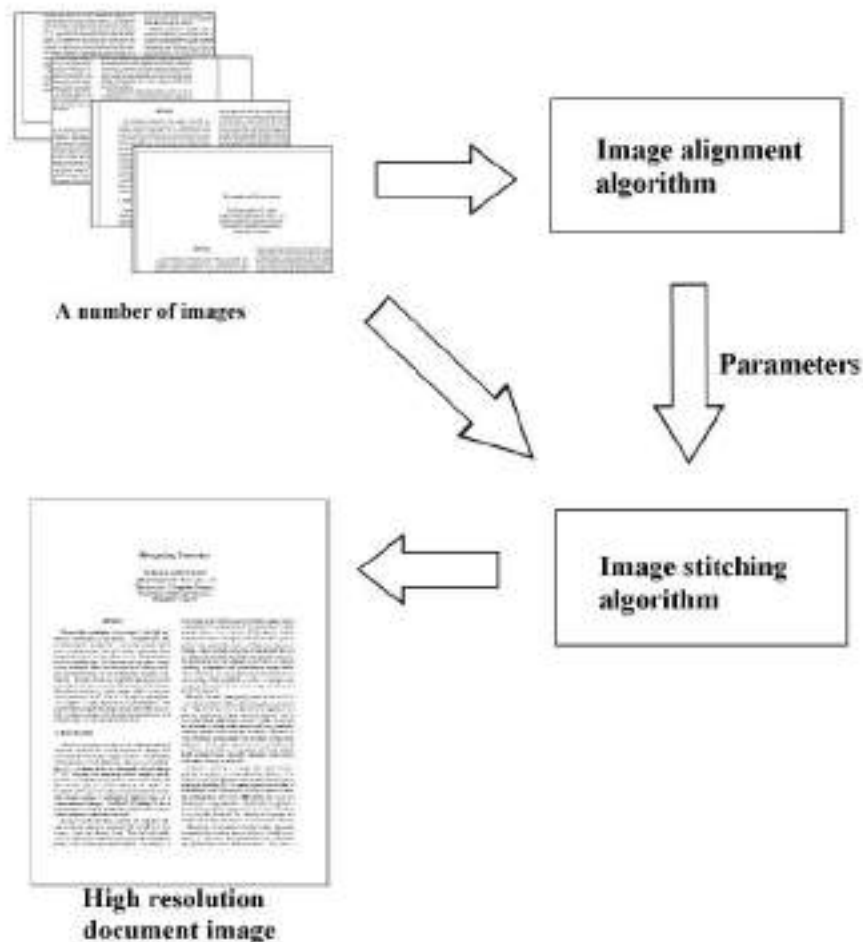
This chapter introduces the overview of Symbian OS and describes the limitations of Symbian phone. In addition, we have successfully implemented program to use Symbian C++ API to take photo using onboard camera, decode and encode image, access raw data of image and do file operation. However, we still fail to implement program to send fax properly. Furthermore, tests on memory had been conducted to find out the limitation of the memory and how should we use the memory.

## **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.



*Fig4.1 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  $(x, y)$

$I_I(x, y)$  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.

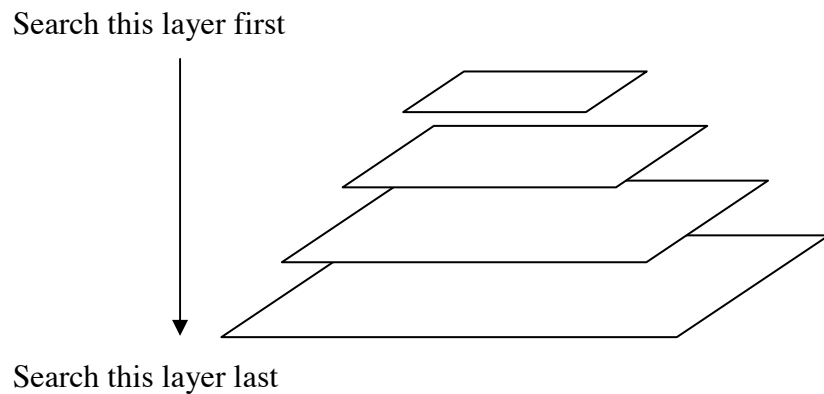
$$E_{SAD}(u,v) = \sum_i |I_1(x_i + u, y_i + v) - I_0(x_i, y_i)|$$

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.



*Fig4.2 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 be written as,

$$\begin{aligned}\mathfrak{F}\{E_{cc}(u, v)\} &= \mathfrak{F}\left\{\sum_i I_0(x_i, y_i) I_1(x_i + u, y_i + v)\right\} \\ &= \mathfrak{F}\{I_0(u, v) \bar{I}_1(u, v)\} = \mathfrak{F}(I_0(u, v)) \mathfrak{F}^*(I_1(u, v))\end{aligned}$$

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.

$$\begin{aligned}\mathfrak{F}\{E_{SSD}(u, v)\} &= \mathfrak{F}\left\{\sum_i [I_1(x_i + u, y_i + v) - I_0(x_i, y_i)]^2\right\} \\ &= \mathfrak{F}\{\delta(u, v)\} \sum_i [I_0^2(x_i, y_i) + I_1^2(x_i, y_i)] - 2\mathfrak{F}(I_0(u, v)) \mathfrak{F}^*(I_1(u, v))\end{aligned}$$

### **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. Several 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

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

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

$$\begin{aligned}
 \mathbf{A} &= \sum_i J_1^T(x_i + u, y_i + v) J_1(x_i + u, y_i + v) \\
 \mathbf{b} &= -\sum_i (I_1(x_i + u, y_i + v) - I_0(x_i + u, y_i + v)) J_1^T(x_i + u, y_i + v)
 \end{aligned}$$

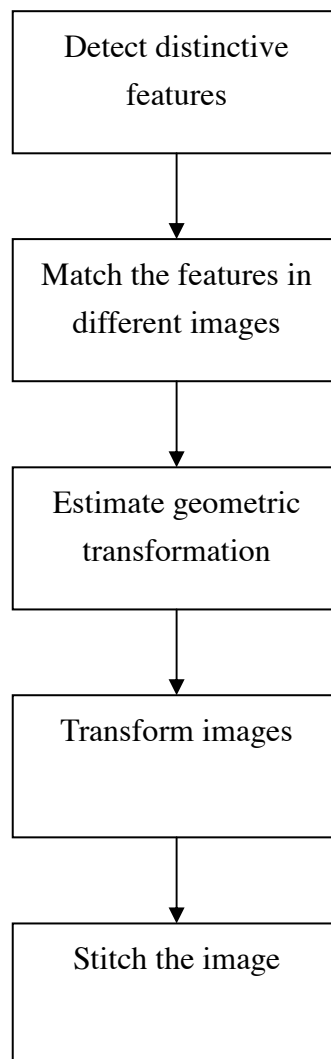
#### **4.2.5 Parametric motion**

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 be matched in the different images. After that,



*Fig4.3 Flow diagram of feature-based registration*

We can estimate geometric transformation for the images. At last, the images need to be transformed and stitched 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.

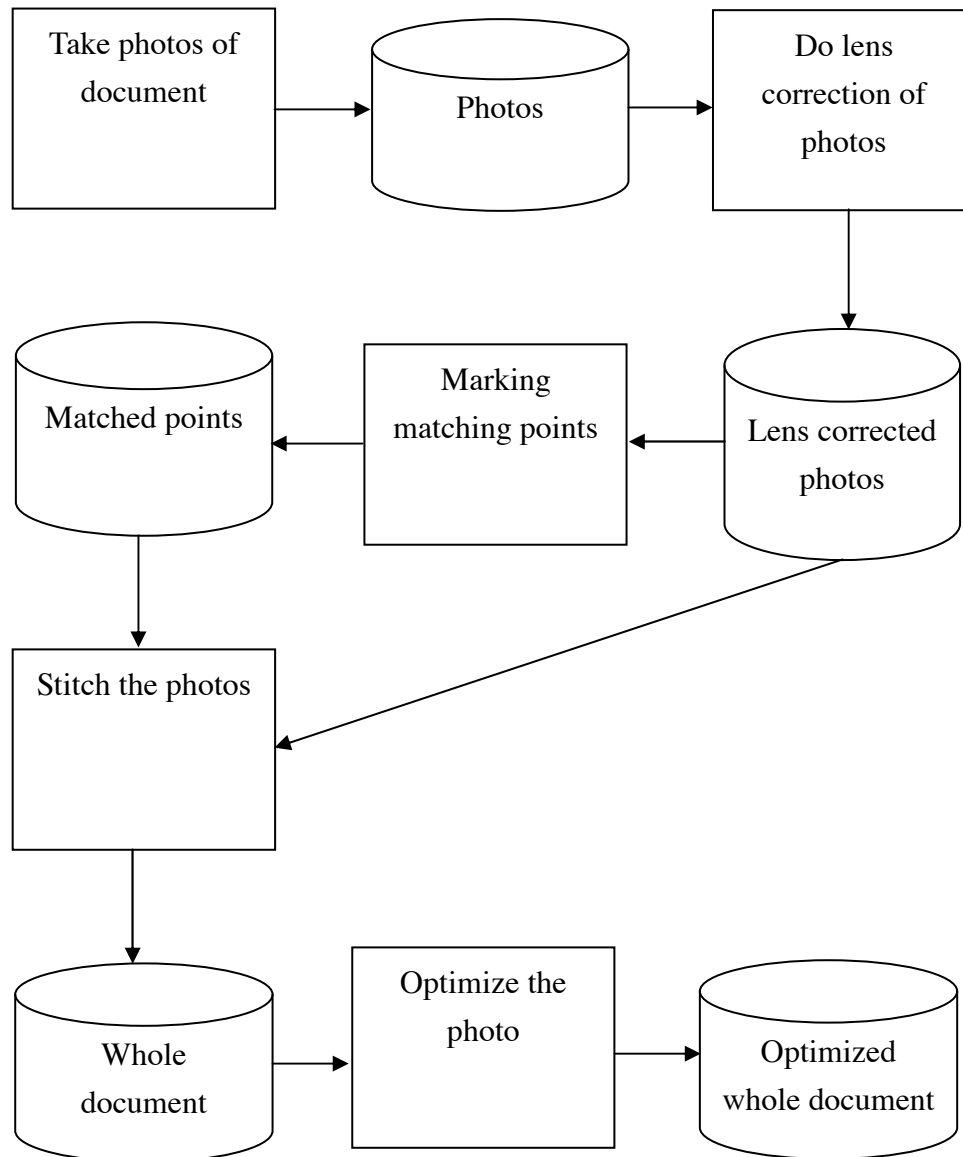
## **Chapter 5 Implementation in Our Project**

### **5.1 Overview**

Our implementation in the Symbian phone is divided into 5 parts. They are,

1. **Take photos of the document.** We have specially designed a graphical user interface to instruct the user to take the several photos of the document. The interface can enable the user to take photo correctly. It can also ensure that the photos taken are already suited for the later stage so that less processing is needed.
2. **Do lens correction of the photos.** After we have taken the required photos, we needed to do lens correction so that the photos are free from lens distortion and ready for the later stage.
3. **Mark matching points of the photos manually.** It is the step of image alignment. In the PC, the matching points are found and matched using SIFT. However, it is not suitable in the Symbian phone because it would need too much time to process.
4. **Stitch the photos.** This step would stitch the photos and produces the whole document. It transforms the photos using the information generated from the step 3.
5. **Optimize the result.** From previous step, the whole document has been already generated. However, we can do some step to improve the document so that the document is more seamless.

The dataflow diagram would illustrate how the data flow in these 5 steps.



*Fig5.1 Data flow diagram of the whole implementation*

## **5.2 Take photos of the document**

Taking photos of the document can be divided into 2 major steps.

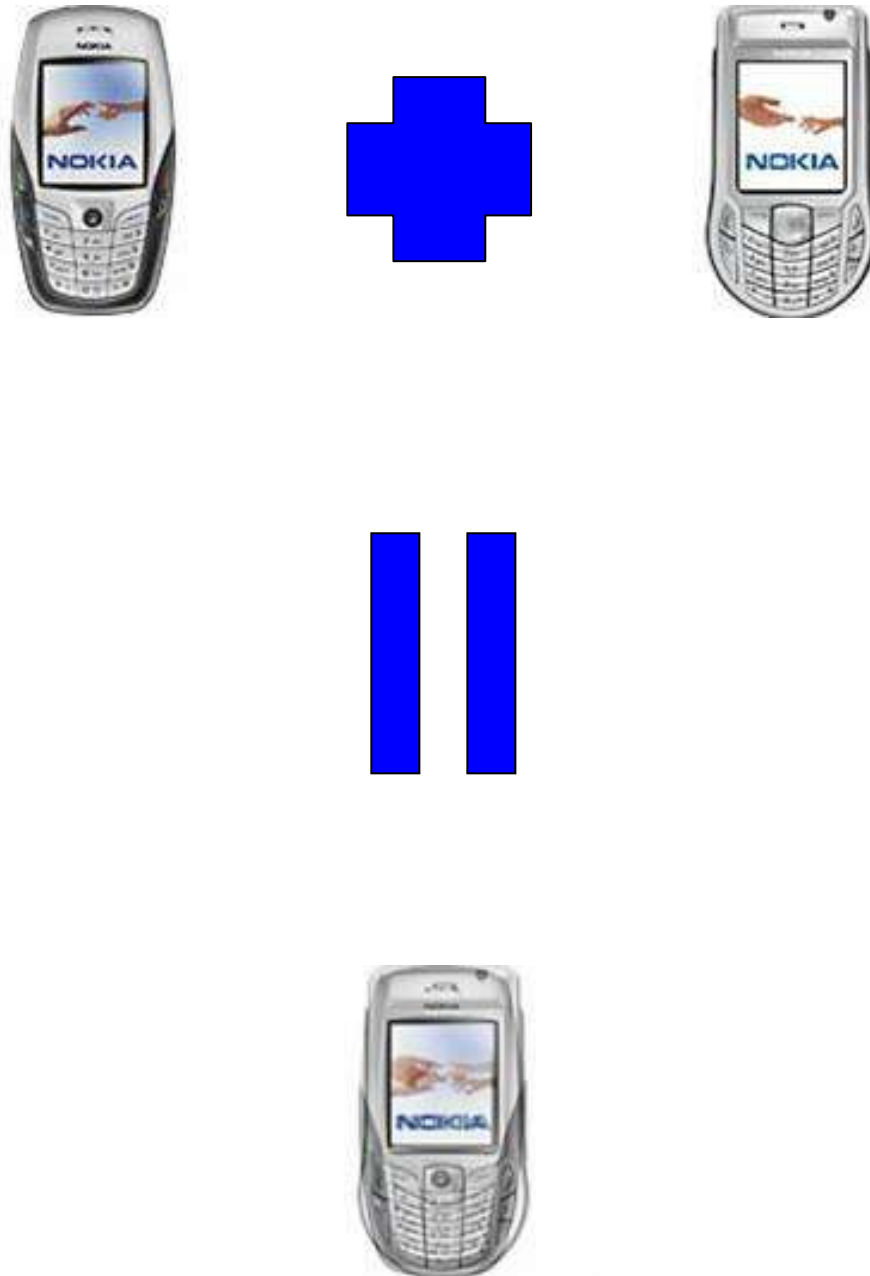
1. Taking photo of the whole document
2. Taking photo of the individual parts of the whole document.

The first part is simple. The user would use the phone to take the photo of the whole document. Although it is simple, the photo of the whole document is very important. The quality of the final output image highly depends on this image because all other individual parts of the document would transform according to this image. When the whole document is taken at a specific orientation, the final output document is at the same orientation. In other words, the final output document in fact is just this image with high resolution.

After the photo of the whole document is taken, the photos of individual parts of the whole document need to be taken. A user interface is designed to instruct the user how to take the correct photos. In the user interface, we use overlaying to do this thing.

### **Overlay**

The idea of the overlaying is displaying two pictures at the same position at the same time. Both of them seem like transparent to each other. There is an example figure showing simple overlaying in the next page. The top-left figure and the top-right figure are two different images. The bottom one is the image after overlaying the two of above images. In this image, you can find features of the top-left figure and top-right figure.



*Fig5.2 Overlaying of 2 images*

In the following, the technique of overlaying is introduced. Here is the formula

Consider there are two input images, picture1 and picture2, and an output image picture3. The red, green and blue components of picture1 at position (x, y) is denoted by picture1[x, y].red, picture2[x, y].green and picture3[x, y].blue.

$$\text{Picture3}[x, y].\text{red} = (\text{picture1}[x, y].\text{red} + \text{picture2}[x, y].\text{red}) / 2$$

$$\text{Picture3}[x, y].\text{green} = (\text{picture1}[x, y].\text{green} + \text{picture2}[x, y].\text{green}) / 2$$

$$\text{Picture3}[x, y].\text{blue} = (\text{picture1}[x, y].\text{blue} + \text{picture2}[x, y].\text{blue}) / 2$$

The above is the calculation of one pixel value at a position. To generate the whole image, all the pixel values are needed to be calculated using the above formula. As the calculation of one pixel value only needs constant time. The complexity to generate an overlay image is  $O(\text{height} * \text{width})$ .

For each color component of each pixel value, the calculation only involves one addition and one division. Although division may take some time, the divider is 2 so that one left shift can be used instead. In addition, the calculation is only integer operation. Therefore, the calculation of one pixel value is simple and efficient for Symbian phone.

The image size needed to do overlay is 400 x 300 (the size would be explained later). As a result, it is possible to do it real-time in Symbian phone.

### **Instruct the user to take correct photos**

After overlaying is introduced, how overlaying would help instruct the user to take correct photo would be discussed. The photo of the whole document has been already taken. This image would help how the individual part of the document. There is an example which is used to illustrate the idea. After taking photo of the



whole document we obtain an image like the following figure.



Fig5.3 Image of whole document

The orientation of the document is landscape because the resolution of the image taken by the photo is 1600 x 1200. The width is longer than the height. To fully make use of the resolution, the document should be better landscape. The image would be divided into 4 sub-images like the following



Fig5.4a Two of the sub-images



*Fig5.4b Another two of the sub-images*

To avoid confusion later, some terms are defined here.

Sub-image of whole document means the divided image of the whole document.

Image of individual part of the whole document means the image taken for individual part of the document.

The sub-image here would be used for the overlaying. Overlaying would be done using the sub-image and the current frame from the camera. The display of the overlaying can enable the user to adjust the camera so that the camera is directed to the correct part of the document. There is an example to illustrate this idea.

Firstly, the two layers are not matched.

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.

*Fig5.5 An example of unmatched layers overlaying*

The camera could be moved according to the misalignment. It is easy for the user to move the phone and get better alignment. The figure below shows the image with better alignment.

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.

*Fig5.6 Another example of unmatched layers overlaying*

In fact, exact match is needed. But the two layers should be matched as much as possible.

### **Careful use of resolution**

So far, the overlaying is discussed to instruct the user to take photos of documents. But there are problems associated with the display. The resolution of the display, the image and the current frame from the camera should be chosen carefully. The reason is that if the resolutions of them are different, resizing is needed. In resizing, interpolation or some other techniques may be needed. Processing using these techniques would take much time. It would make real-time impossible.

Here, we use the Nokia N90 as our equipment.

Firstly, the size to display on the Symbian phone is chosen to be 400 x 300. Although this is not the exact but just approximate resolution of the display, it is more convenient for us. The reason would be discussed later.

Then the resolution of the image taken using the camera should as large as possible as our objective of our document is to reconstruct document with higher resolution. As a result, the resolution is chosen to be 1600 x 1200 which is 2Mega pixel photo.

There are several resolutions of the current frame can be chosen in Nokia N90. It includes 1600 x 1200, 800 x 600 and some other lower resolutions. If 1600 x 1200 is used, the time to getting the current frame from the camera would be too long and the display would lag. The user would find difficult to use the user interface. Therefore, 800 x 600 is used. The lower resolutions should not be chosen due to a reason which would be explained in the following.

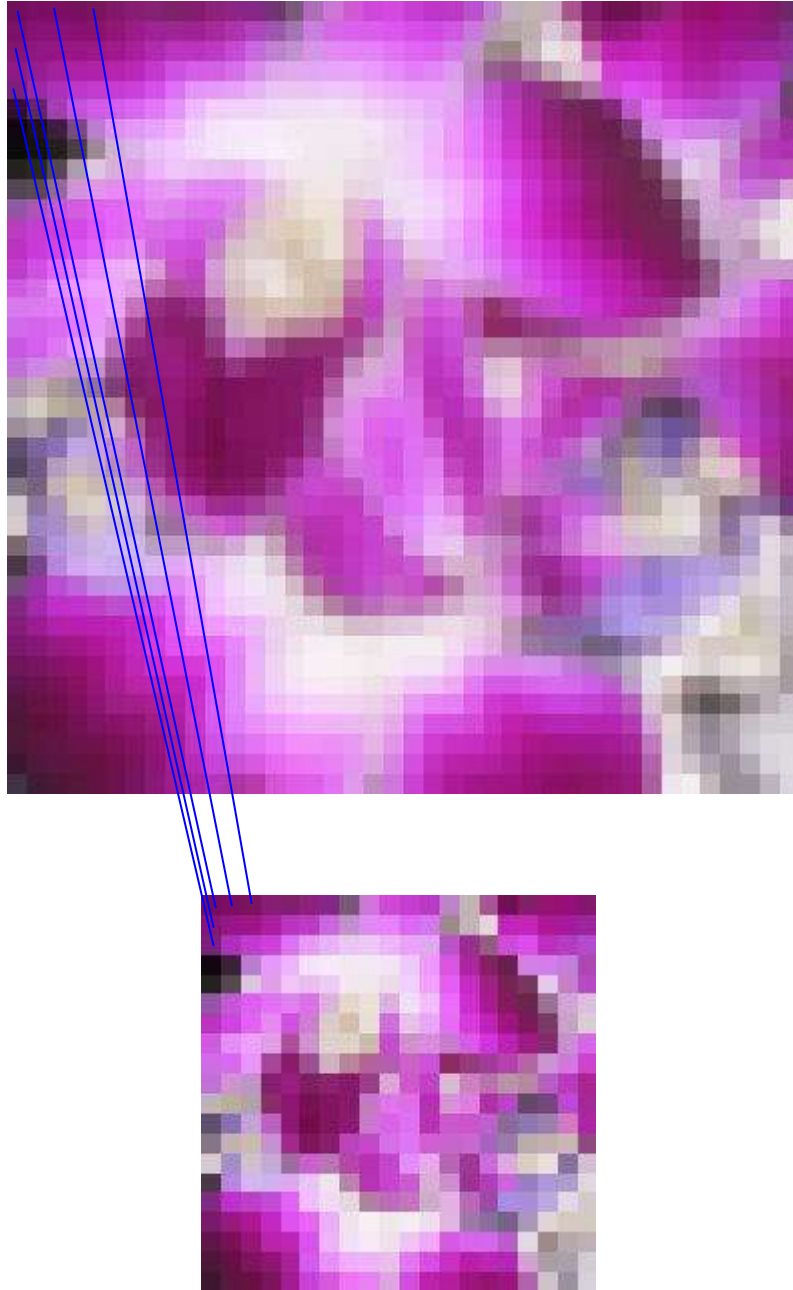
As a result, here are the resolutions of the different types of image:

Types	Resolutions
Display	400 x 300
Photo taken	1600 x 1200
Sub-image of document	$(1600 / 2) \times (1200 / 2)$ 800 x 600
Current frame from camera	800 x 600

There is characteristic of the two kinds of images. They are all multiples of the 400 x 300. Sub-image of document and current frame from camera need to be resized to 400 x 300 for display. However, only sub-sampling is needed.

### **Sub-sampling**

There are figures to illustrate the idea of sub-sampling in the following. Sub-sampling means copying the pixel value according to the ratio of the resolutions of the output image and the input image. In the example of reduce both dimension by 2. Alternate pixel value of the input image is copied to the output image. No interpolation is needed. Interpolation would need float-point operation. So this method can save much time to process.



*Fig5.7 Sub-sampling the upper image to the lower image*

**Take photos of the document** is concluded in the following pseudo code.

```
Take phone of whole document  
Divide the whole document into four  
For each sub document  
    While (there is next frame from camera)  
        Sub-sampled sub-document  
        Sub-sampled of the current frame  
        Overlay the above two image and display  
        if (User take photo of sub-document)  
            Store the photo  
            break
```

## 5.3 Do lens correction of the photos

### 5.3.1 Introduction

After taking photos, we find that all the photos suffered from lens distortion (Fig5.8). It can be obviously seen the bending at the sides of the document.

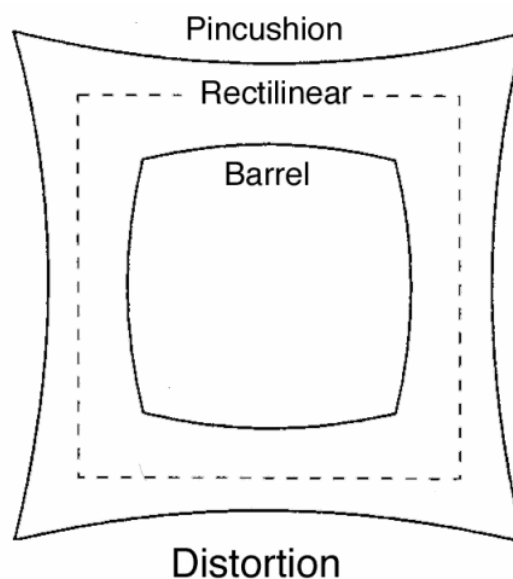
To overcome the problems of lens distortion, we will first perform lens correction before image alignment. This technique is used to ensure the accuracy of matching the sub-images and have correct transform parameters.



Fig5.8 Photo taken using Nokia N90 of resolution 1600x1200.

## **5.3.2 Basic concepts**

### **5.3.2.1 Lens distortions**



*Fig5.9 Common types of lens distortion*

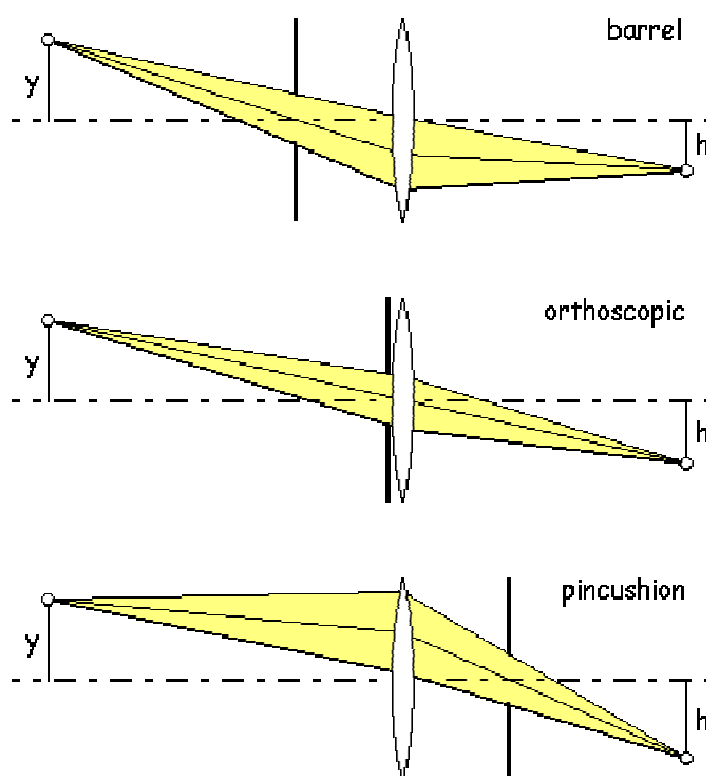
Lens design has been important for photography since its beginnings. Fortunately lenses had been designed for optical instruments such as telescopes and microscopes for a great many years before the invention of photography.

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.

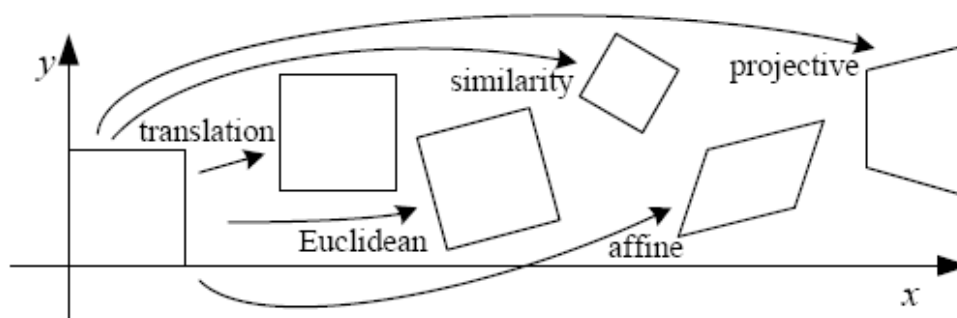


*Fig5.10 Causes of different type of distortion*

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.

### 5.3.2.2 Image Transformation



*Fig5.11 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 Fig5.11. 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:

$$x' = x + tx$$

$$y' = y + ty$$

(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:

$$x' = (S_x)x$$

$$y' = (S_y)y$$

- If  $|S_x|$  and  $|S_y| > 1$ , the effect is increasing the size of an object.
- If  $|S_x|$  and  $|S_y| < 1$ , the size is reduced.

For a symmetric or uniform scaling transformation in which the x and y scale factors are the same ( $S_x = S_y$ ), 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 ( $S_x \neq S_y$ ), 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  $\alpha$  about the origin. A line joining the point with the origin makes an angle  $\beta$  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  $\cos(\alpha + \beta)$  and  $\sin(\alpha + \beta)$ , and rearranging the  $x'$  and  $y'$  is expressed as:

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

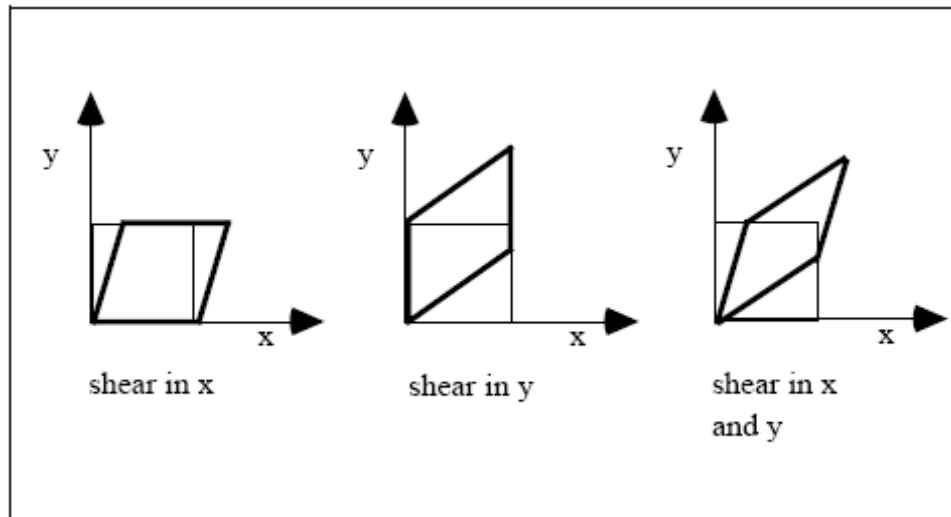
$$y' = R \sin \alpha \cos \beta + R \cos \alpha \sin \beta$$

Finally substituting for  $R \cos \beta$  and  $R \sin \beta$ , the co-ordinate of new point  $(x', y')$  is

simplified to:

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

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



*Fig5.12 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), and then calculate the new coordinates of the pixels inside the region bounded by the 4 tiepoints. The diagram below illustrates such transformation:

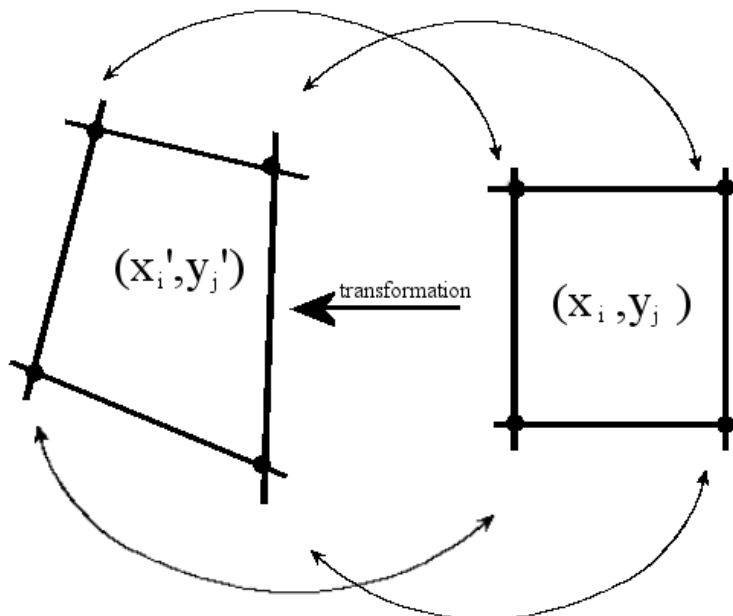


Fig5.13 Image transformation using 4 pairs of tiepoints

$$\begin{bmatrix} x_1 & y_1 & x_1 y_1 & 1 \\ x_2 & y_2 & x_2 y_2 & 1 \\ x_3 & y_3 & x_3 y_3 & 1 \\ x_4 & y_4 & x_4 y_4 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & x_1 y_1 & 1 \\ x_2 & y_2 & x_2 y_2 & 1 \\ x_3 & y_3 & x_3 y_3 & 1 \\ x_4 & y_4 & x_4 y_4 & 1 \end{bmatrix} \begin{bmatrix} c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} = \begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \end{bmatrix}$$

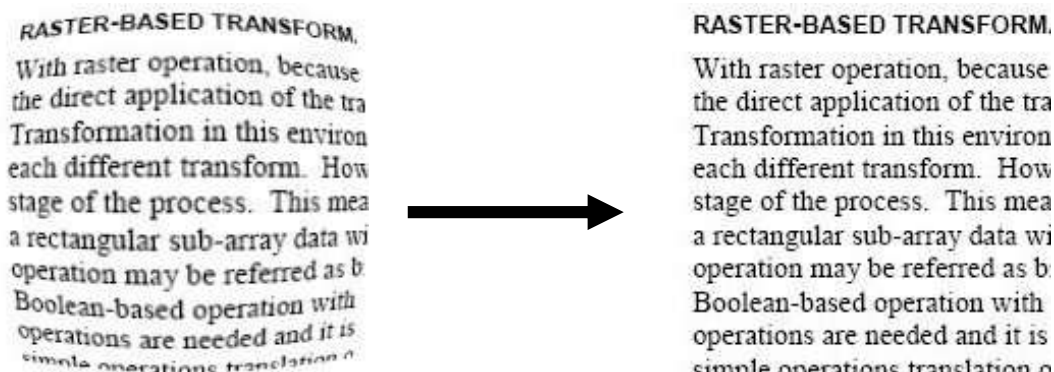
where,

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

$(x'_i, y'_j)$  are the new coordinates after transformation

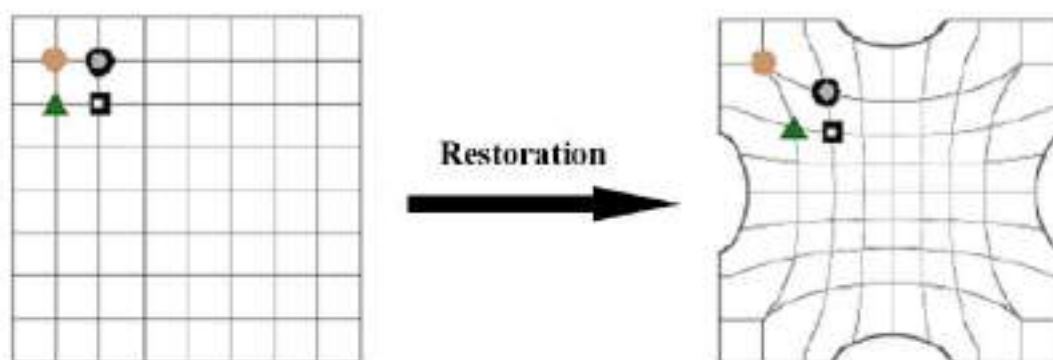
$c_i$  are the transform parameters

### 5.3.3 Implementation



*Fig5.14a 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.



*Fig5.14b 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). See Fig5.15.

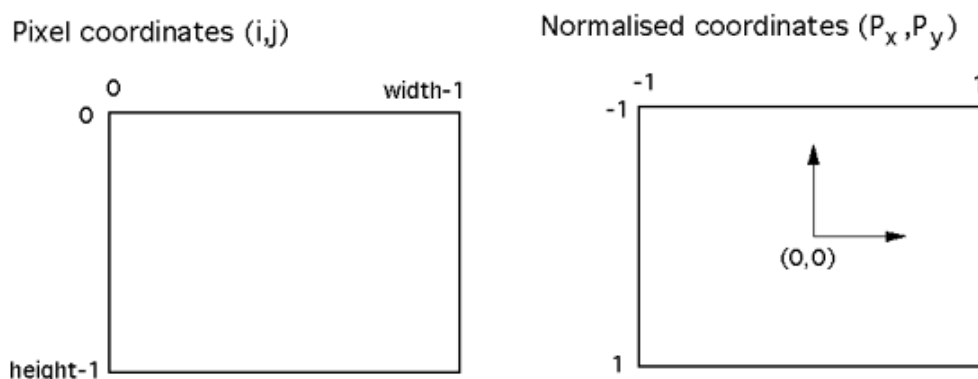


Fig5.15 Pixel coordinates system & normalized coordinate system

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.

### 5.3.4 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 (Fig5.16), but there are still rooms for improvement as the correction factors should be slightly different for different images.



Fig5.16 Lens corrected image of Fig5.8



## **5.4 Mark matching points of the photos manually**

### **Reason of marking matching points manually**

In our project, feature-based registration is used. In PC, we use SIFT to find and match feature points. However, it seems that it is not realistic to use SIFT in Symbian phone.

In PC, it needs 4-5 minutes to find feature points of the images for each image. It needs 4-5 minutes to match the feature points of the image for each pair of images. In our implementation, 5 images (1 image of whole document and 4 individual images of the documents) are needed to find feature points. 4 pairs of images (each individual image of the document and the image of the whole document) are needed to match feature points. As a result the total time is about 45 minutes.

The table below shows the comparison of the processors in Symbian phone and PC.

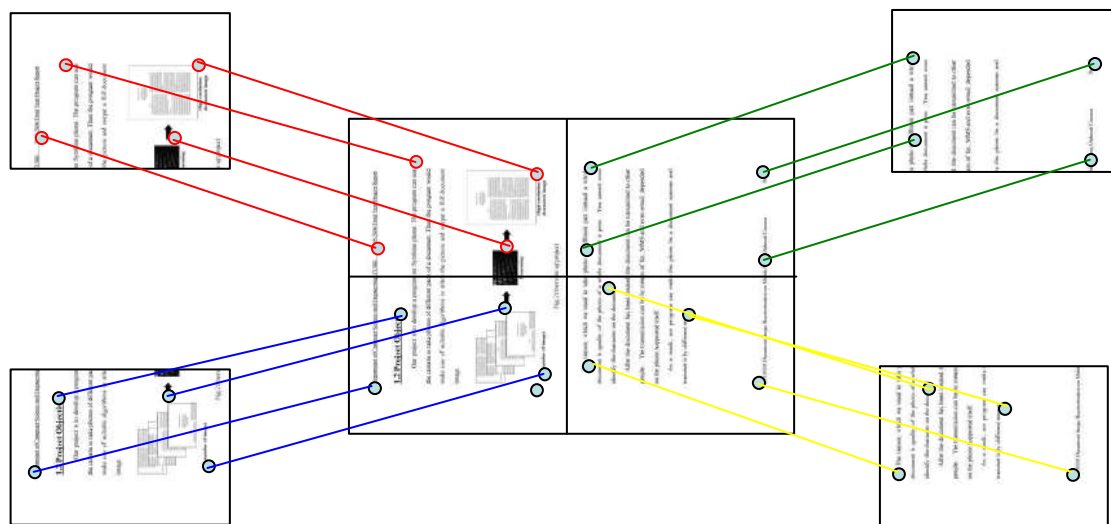
	<b>PC</b>	<b>Symbian phone</b>
<b>Process</b>	Pentium 4 3.0Ghz	RISC 220Mhz
<b>Float-point Unit</b>	Yes	No

There is great different between these two processors. It is obvious that Symbian phone would spend much more time to find and match feature points. As a result, it is designed to mark the match points manually. It would be more realistic. In order to instruct the user to mark the match points easily, a graphical user interface is designed.

### **Requirement of marking matching points**

Before discussing the user interface, the requirement of marking matching

points is first stated. Each individual part of document is transformed according to the whole document before stitching together. Therefore, there are 4 pairs of the images (Each individual part and the whole document) which are needed to find matching points. Each pair needs 4 pairs of matching points. The details are illustrated in the following figure.



*Fig5.17 16 pairs of matching points*

From Fig5.17, 16 pairs of matching points are needed for the later stage.

### **Graphical User Interface**

To enable the user to match the points, suitable image should be displayed on the screen. There should also be a cursor which enables the user to mark the points. The image should be divided and displayed to the screen. The whole document is divided by 8 and the individual part of document is divided by 2. It is illustrated by the following figure. The sub-images of them are marked with number for later explanation.

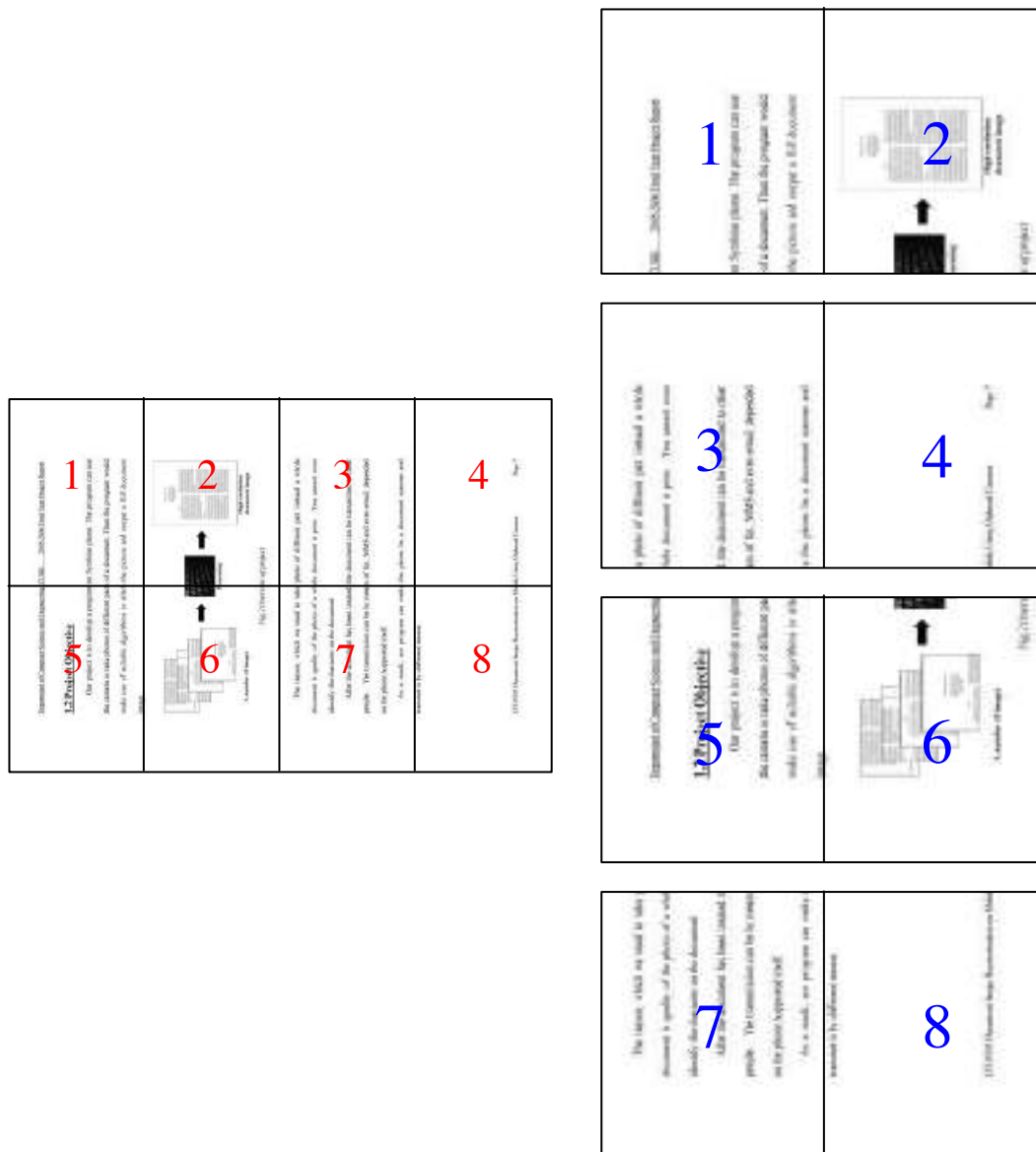
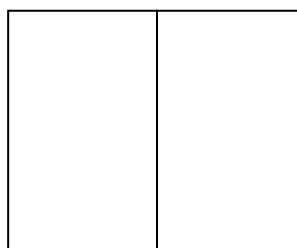


Fig5.18 The images of document is divided accordingly

After the images are divided, the sub-images would be displayed on the screen of the phone at a specific place. The screen on the phone would be split by two equal parts where one part is at the left and one part is at the right.



The left part is used to display the sub-images of the whole document. The right is used to display the sub-images of the individual parts of the document. At the first time, the left part would display the sub-image of the whole document marked with red 1 and the right part would display the sub-images of the individual parts of the document marked with blue 1. The process repeats for 2<sup>nd</sup> – 8<sup>th</sup> time using corresponding marking number.

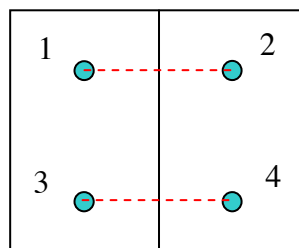
There is also problem associated with size similar in the overlay in the step 1.

Here is the size of different images

Types	Resolutions
Display	400 x 300
One part (either right or left) of the display	$(400 / 2) \times 300$ 200 x 300
Photo taken	1600 x 1200
Divided image of whole document	$(1600 / 4) \times (1200 / 2)$ 400 x 600
Divided image of individual part of document	$(1600 / 2) \times 1200$ 800 x 1200

Both the divided image of whole document (400 x 600) and divided image of individual part of document (800 x 1200) are needed to be resized to 200 x 300 which is one part of display.

At each time, the phone would wait for user to mark the match 2 pairs of matching points. The order of marking is illustrated in the following figure.



*Fig5.19a Starting position of corresponding markers*

The screen would first display the marker in the position 1. The user can move the marker to choose a feature point. After the point is matched, the screen would display the marker in the position 2. The user can move the marker to find a point which is matched with the position 1. The process would repeat for with position 3 and 4. After each point is marked, the coordinates (x, y) with reference with undivided is stored in the files.

Finally, 16 pairs of matching point would be obtained and they are stored in the file for next step.



*Fig5.19b Screen capture of marking the matching points*

**Mark matching points of the photos manually** is concluded in the following pseudocode.

*For each sub-document*

    Display left half of the sub-document and corresponding part of whole document

        Marking two pairs of points

    Display right half of the sub-document and corresponding part of whole document

        Marking two pairs of points

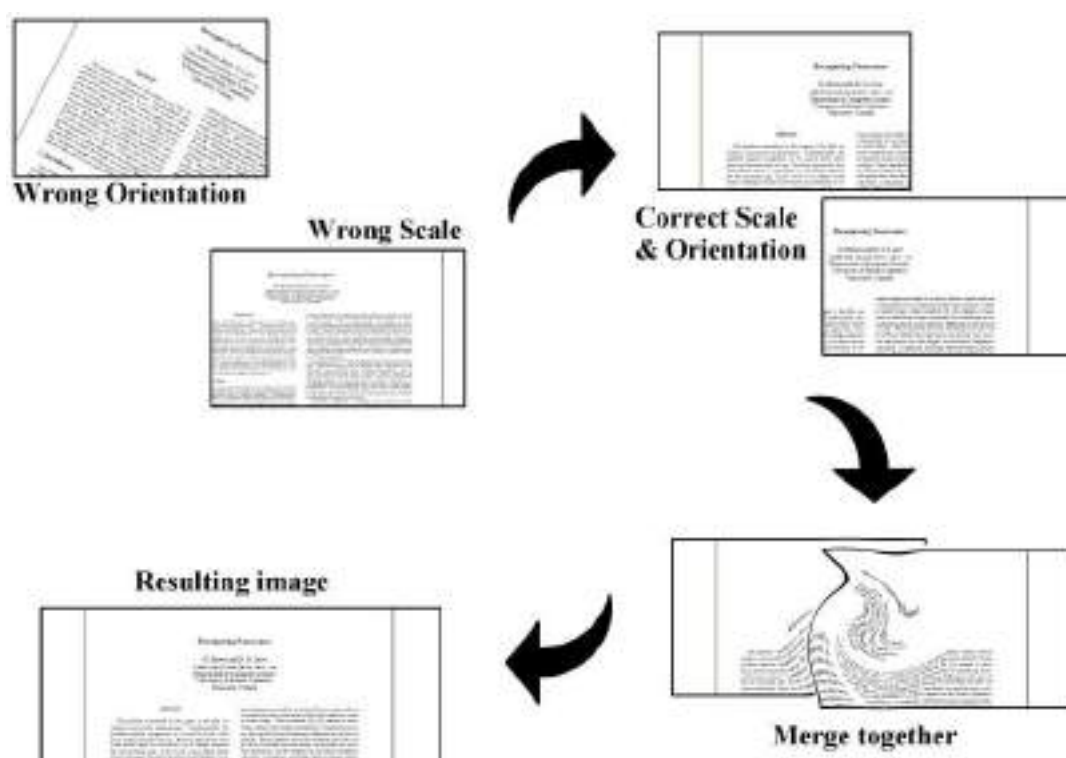
    Store the coordinates into the file.

## **5.5 Stitch the photo**

### **5.5.1 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 an 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.



*Fig5.20 Flow diagram of image stitching*

## 5.5.2 Implementation



Fig5.21a Image1 – Overall Image (Resolution: 1600x1200 pixels)



Fig5.21b Image2 – Any one of the sub-images (Resolution: 1600x1200 pixels)



Fig5.21c Image3 – All sub-images other than Image2 (Resolution: 1600x1200 pixels)



### Step 1:

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:

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

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

### Step 2:

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  $C_{Ai}$ .

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

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

**Step 3:**

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 4:**

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

**Step 5:**

Since after Step 4, 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 6:**

Repeat Step 3 to Step 5 with all other sub-images.

Note:

Images in Fig5.22 are not drawn to scale,

They all are resolution 1600x1200, except “High Resolution Document Image”

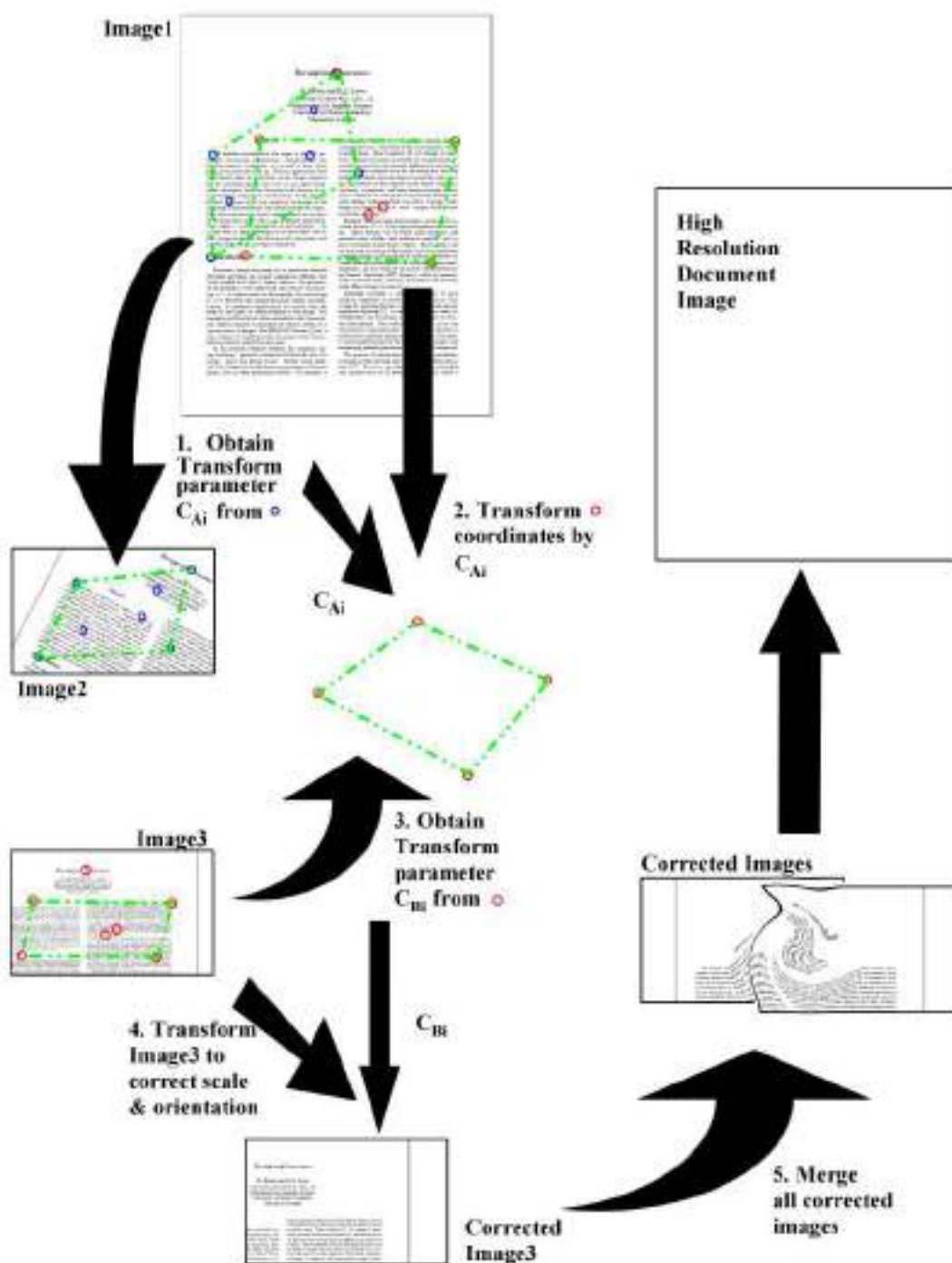


Fig5.22 Flow diagram of image stitching

### **5.5.3 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.

## **5.6 Optimize the photo**

### **5.6.1 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.



*Fig5.23 Image with checkerboard effect*

### **5.6.2 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  $2 \times 2$  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.

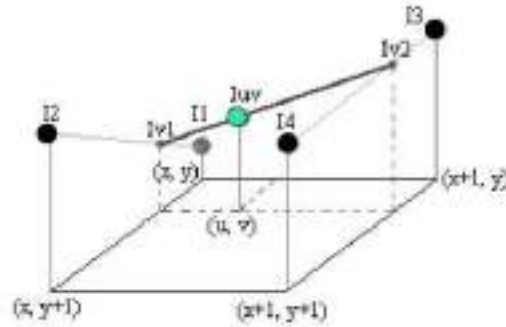
(c) **Bicubic interpolation** - It is more sophisticated and produces smoother edges than bilinear interpolation. Notice for instance the smoother eyelashes in the example below. Here, a new pixel is a bicubic function using 16 pixels in the nearest  $4 \times 4$  neighborhood of the pixel in the original 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.

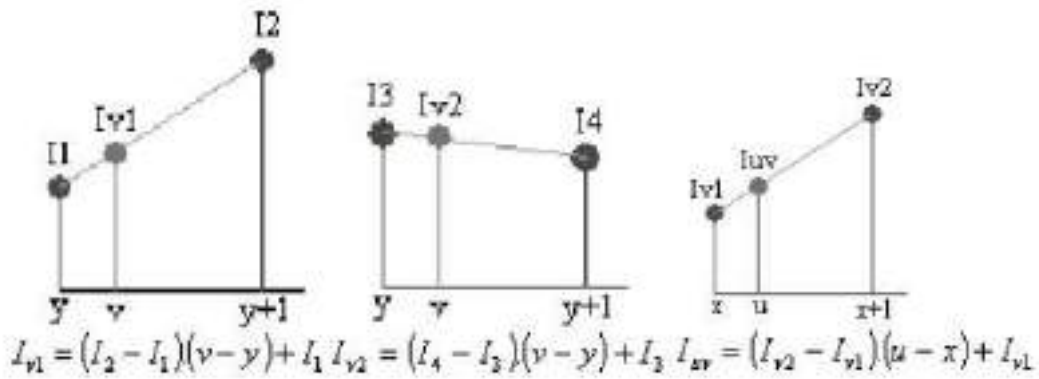
### 5.6.3 Implementation

#### 5.6.3.1 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:



*Fig5.24*  
Pixel value  
calculated by  
bilinear interpolation



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:

$$I_{v1} = (I_2 - I_1)(v - y) + I_1$$

$$I_{v2} = (I_4 - I_3)(v - y) + I_3$$

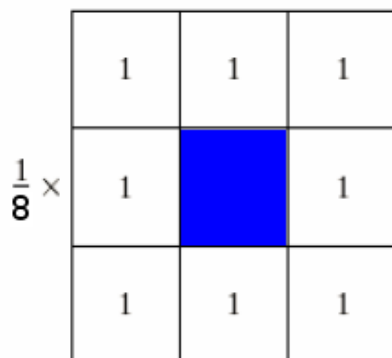
And finally find out the desired pixel value by:

$$I_{uv} = (I_{v2} - I_{v1})(u - x) + I_{v1}$$

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

### 5.6.3.2 Averaging mask

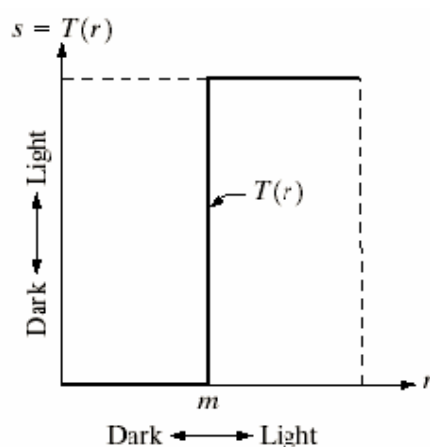
There are also cases that after shearing the sub-images, some pixels are missing its values, but bilinear interpolation is not suitable for these cases. Therefore we applied averaging mask instead of bilinear interpolation.



*Fig5.25 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.

### 5.6.3.3 Thresholding



*Fig5.26 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.

#### **5.6.4 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.

## **5.7 Experimental Results**

Here are the descriptions of the pictures following.

All photos are taken by the Nokia N90 model, using 2 Megapixels setting.

### **Experimental Set 1:**

It is a black and white document containing words only.

Fig5.27a, b, c, d, e:

They are the photo of whole document and its 4 sub-images which were taken under the guidance of GUI of “Take photos of the document”.

*(Resolution: 1600 x 1200 pixels each)*

Fig5.28a, b, c, d, e:

The image of the whole document and its 4 sub-images after lens correction performed.

Fig5.29: This is used to compare the images before and after lens correction performed. We can see that the edge of image of document is curved before correction and almost follows the straight dotted line after corrected.

Fig5.30a: The high resolution image stitched from the sub-images after

“Mark matching points of the photos manually” and “Stitch the photos”.

*(Resolution: 2400 x 3200 pixels each)*

Fig5.30b: The resulting image after optimization of bilinear interpolation.

Fig5.30c: The resulting image after further optimized by thresholding.

Fig5.31: It is used to compare the original photo of whole document (*1600 x 1200 pixels*) and the resulting image (*2400 x 3200 pixels*) by showing 4 levels of details of the images.

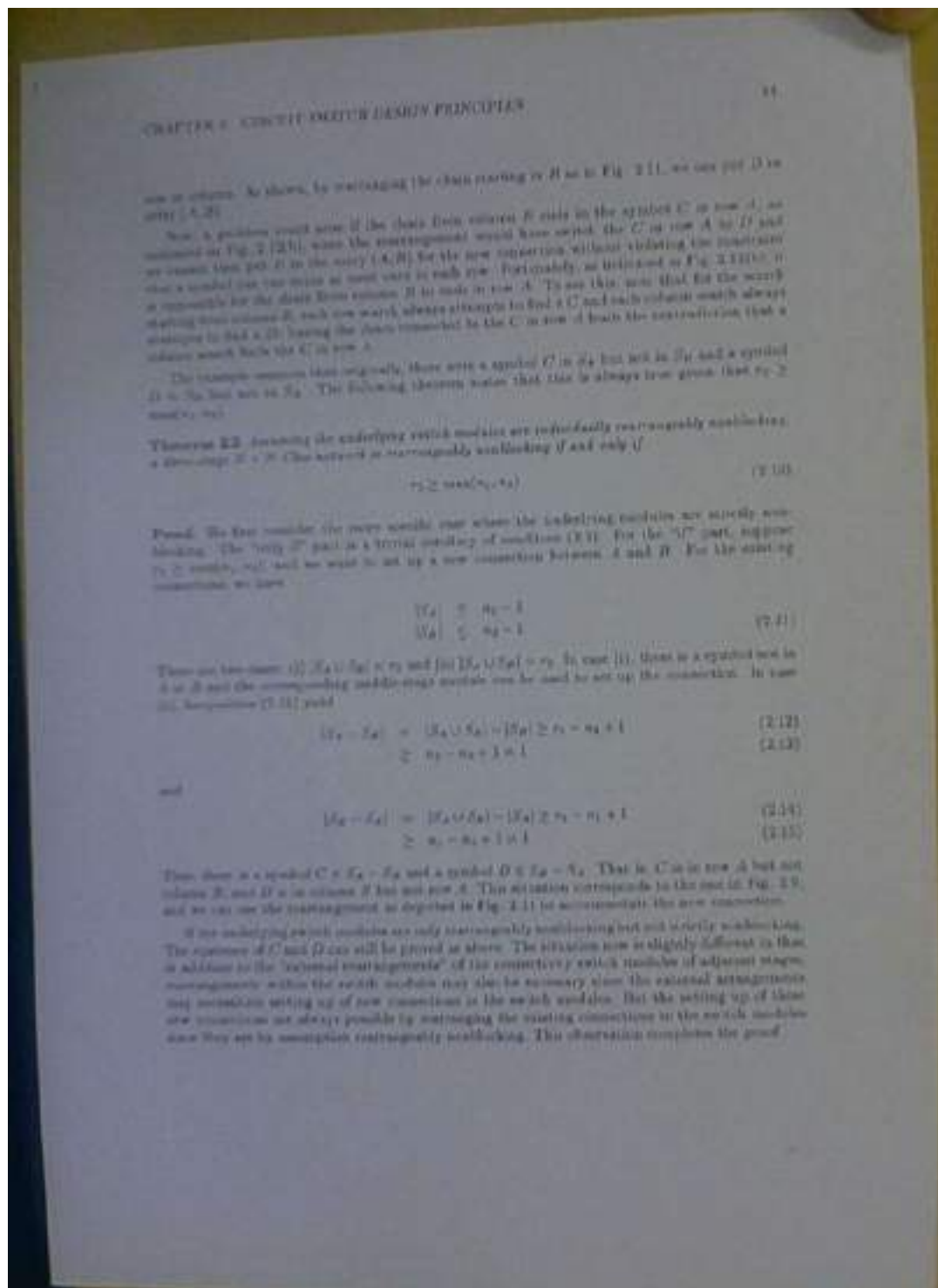


Fig5.27a A photo of the whole document

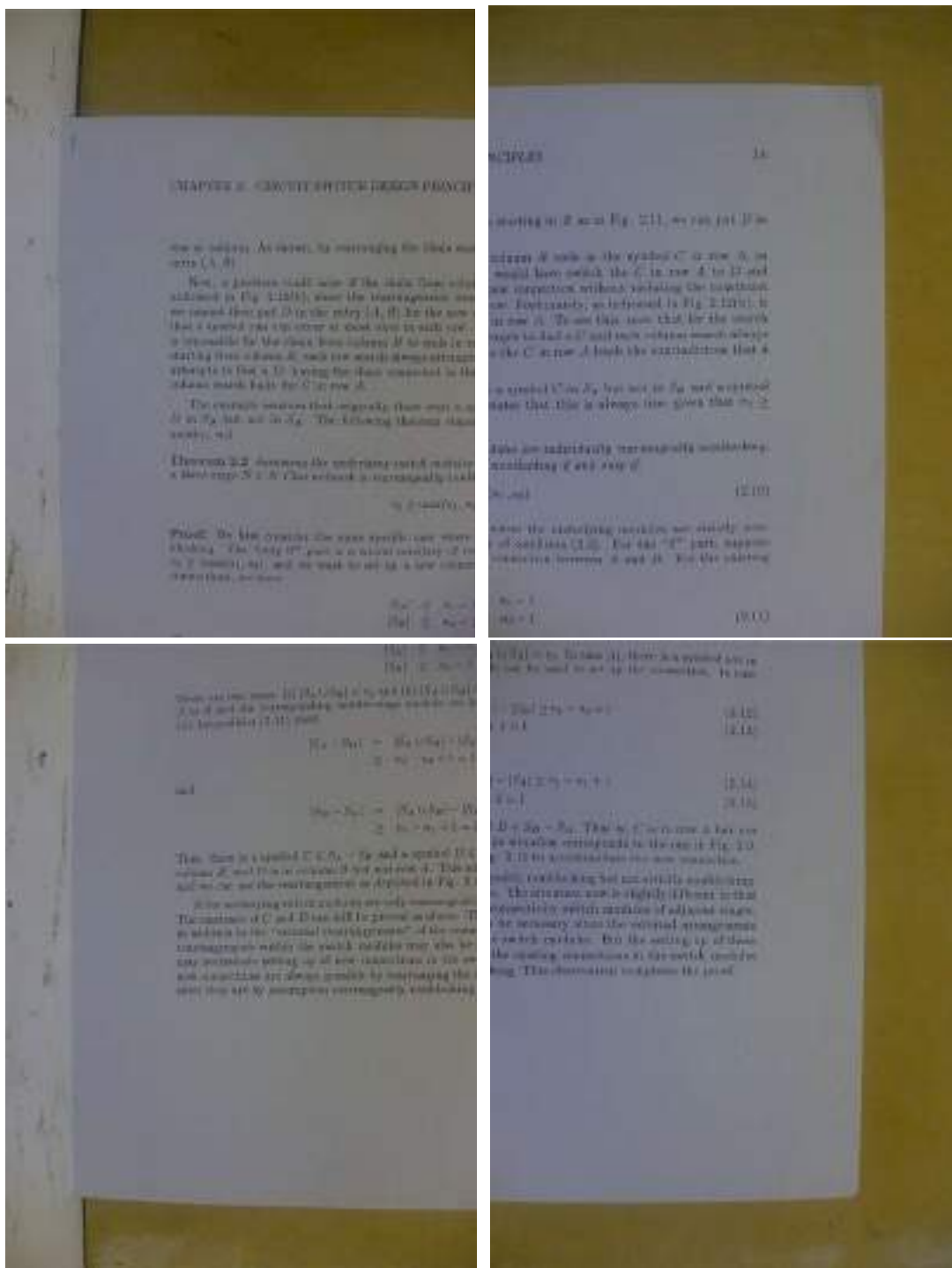
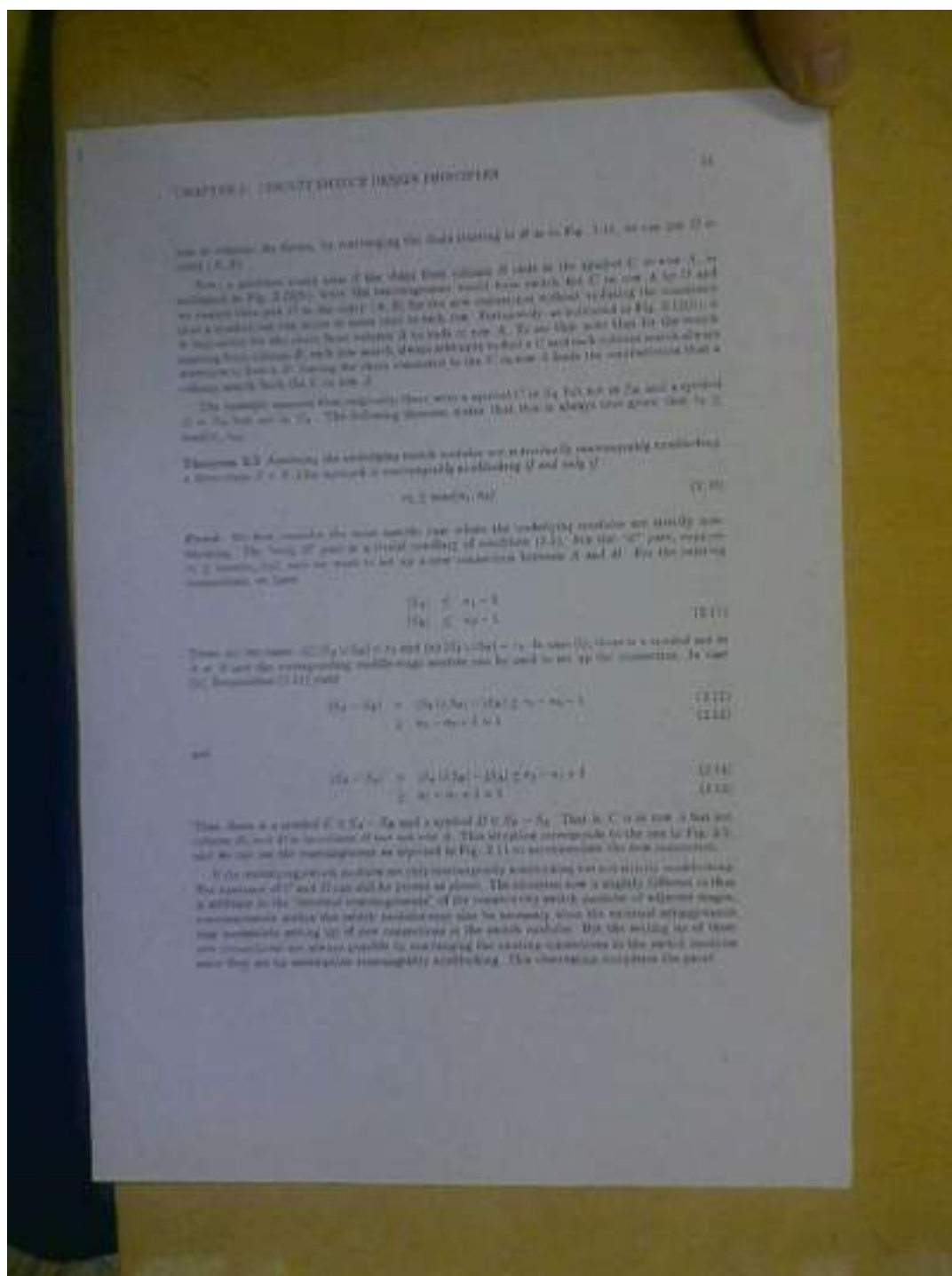


Fig5.27b, c, d, e

The 4 sub-images of the document taken under the guidance of GUI



*Fig5.28a Image of whole document after lens correction performed.*

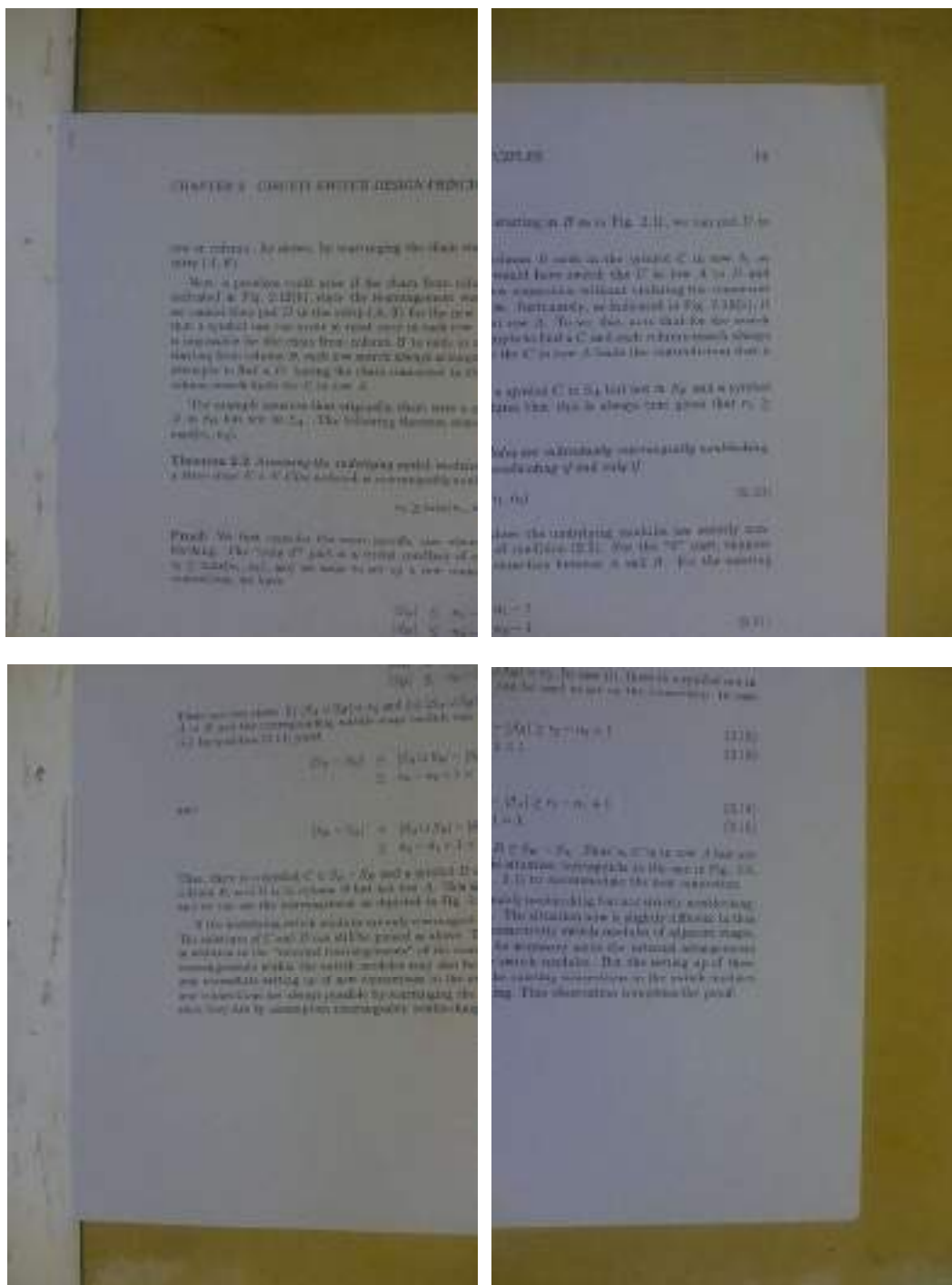


Fig5.28b, c, d, e

The 4 sub-images after lens correction performed.

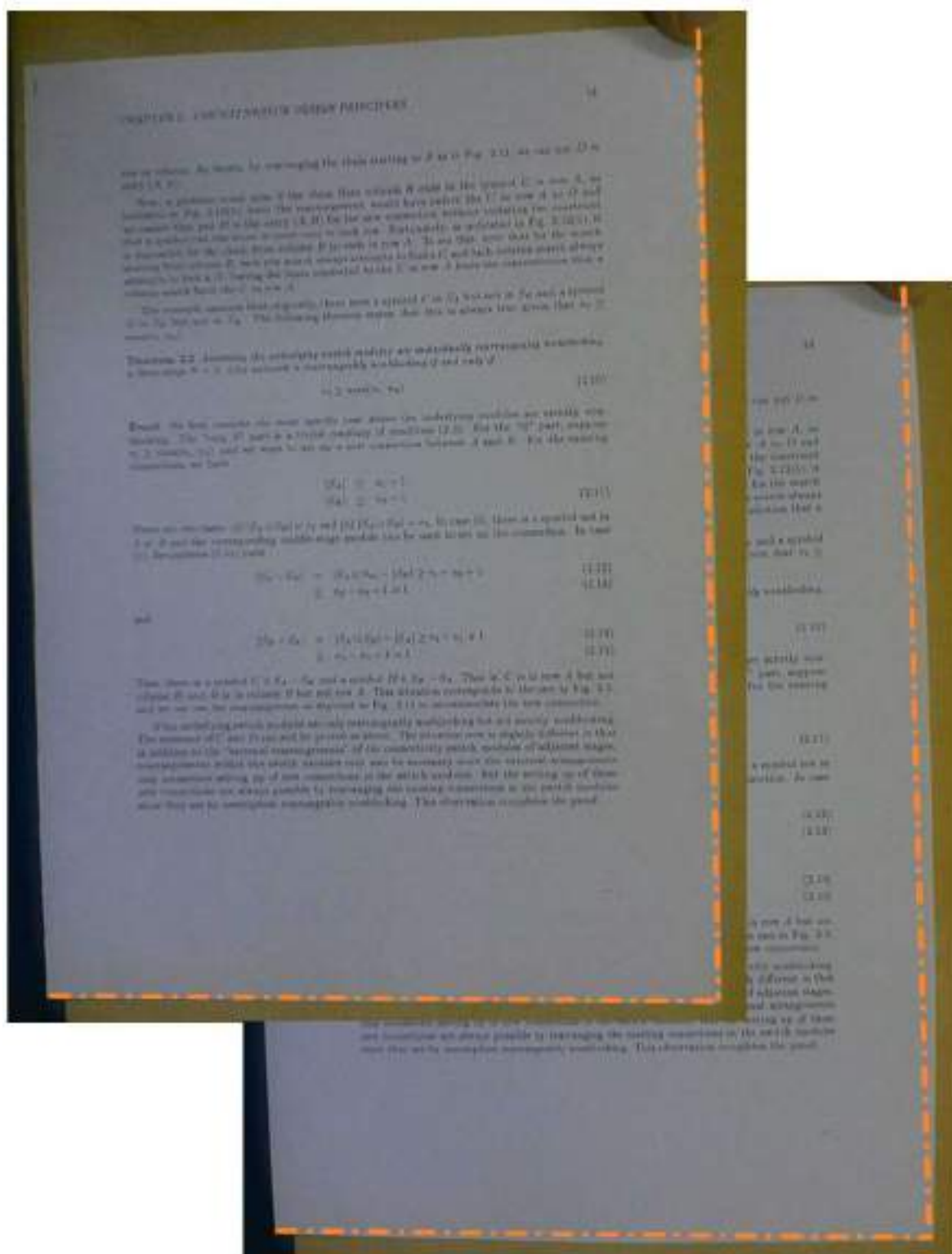


Fig5.29 Comparison of photos after lens correction (left)  
and before lens correction (right)



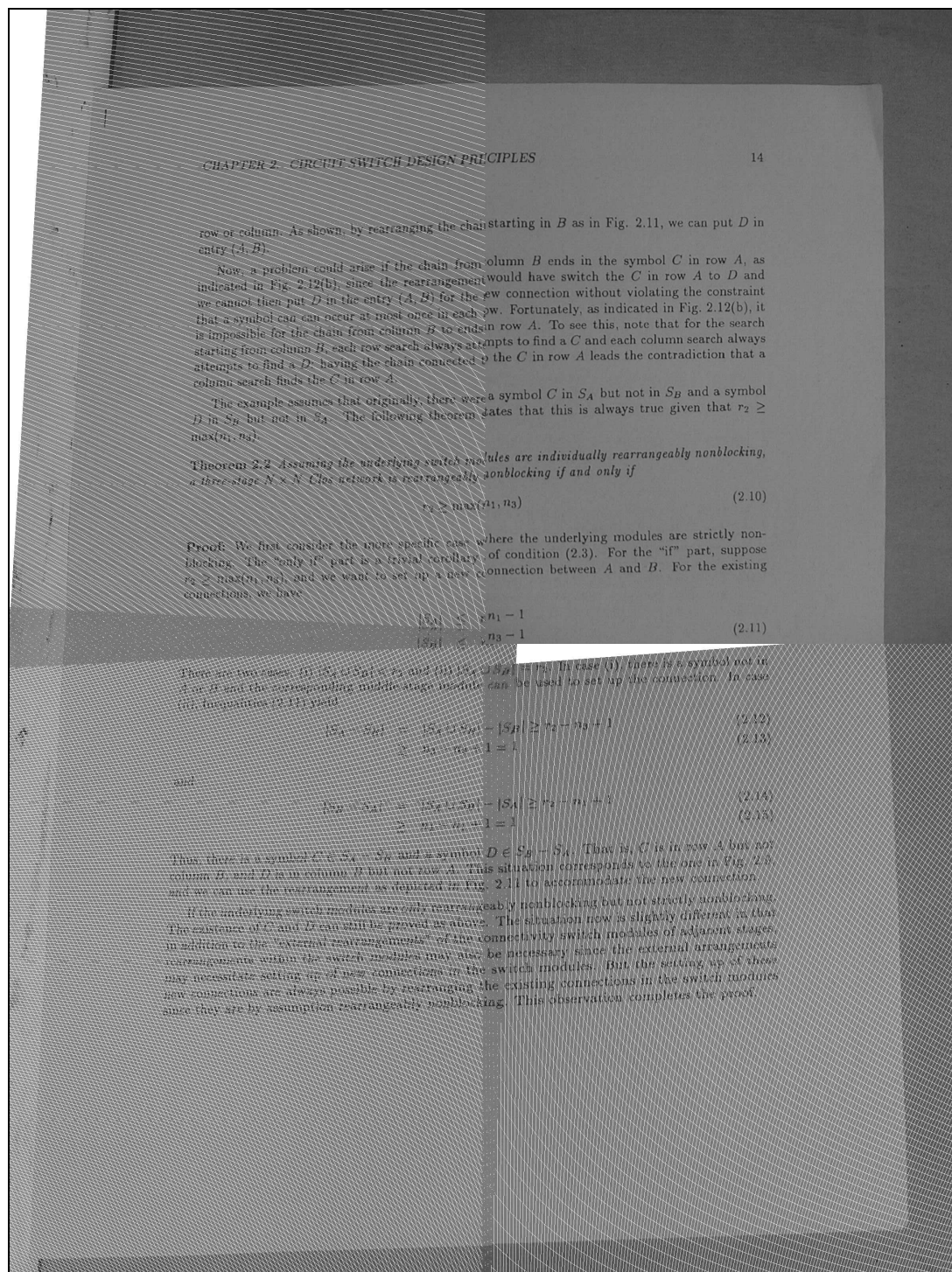


Fig5.30a The high resolution image stitched from the sub-images.



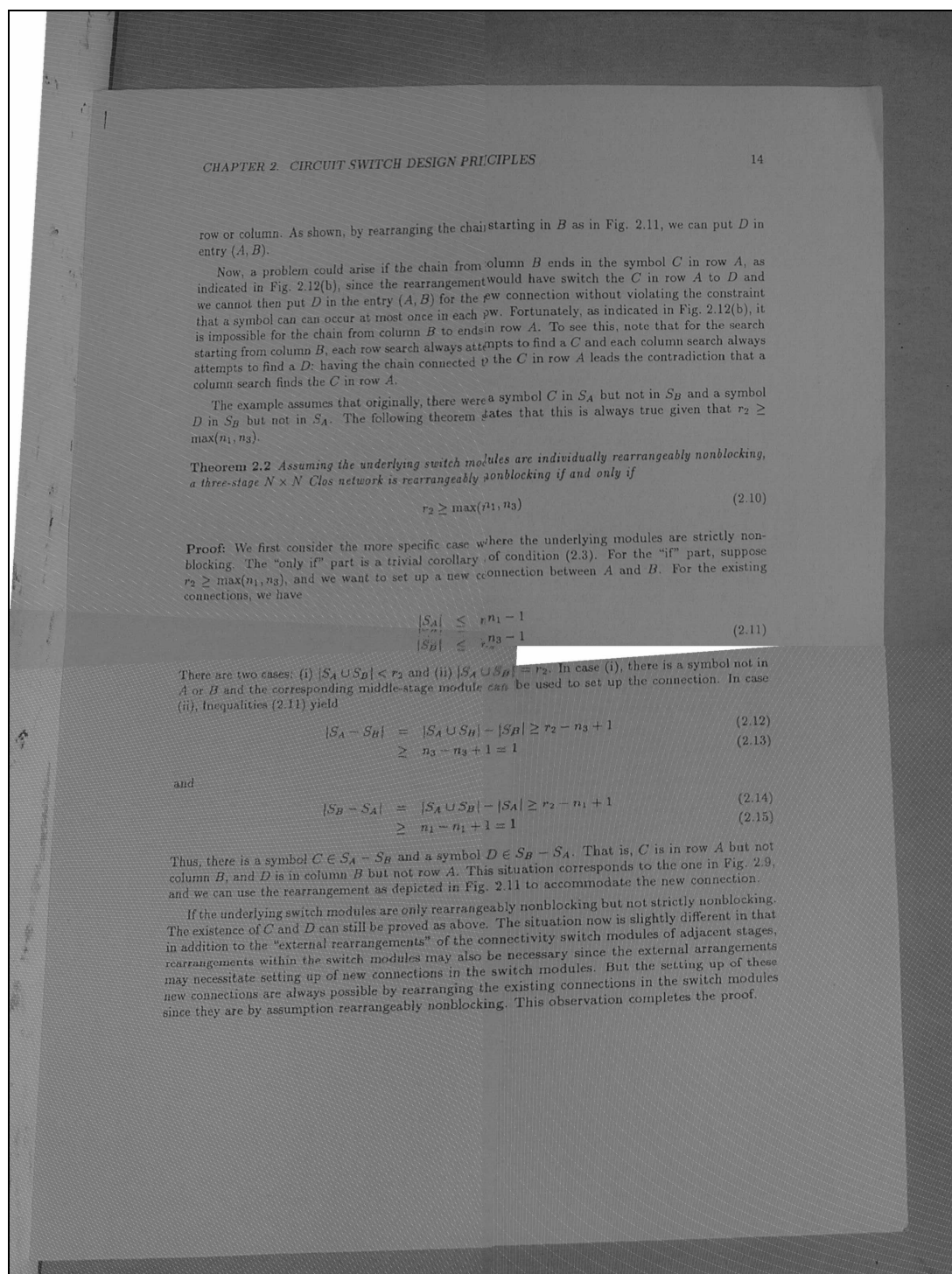


Fig5.30b The resulting image after optimization of bilinear interpolation

## CHAPTER 2. CIRCUIT SWITCH DESIGN PRINCIPLES

14

row or column. As shown, by rearranging the chain starting in  $B$  as in Fig. 2.11, we can put  $D$  in entry  $(A, B)$ .

Now, a problem could arise if the chain from column  $B$  ends in the symbol  $C$  in row  $A$ , as indicated in Fig. 2.12(b), since the rearrangement would have switched the  $C$  in row  $A$  to  $D$  and we cannot then put  $D$  in the entry  $(A, B)$  for the new connection without violating the constraint that a symbol can occur at most once in each row. Fortunately, as indicated in Fig. 2.12(b), it is impossible for the chain from column  $B$  to end in row  $A$ . To see this, note that for the search starting from column  $B$ , each row search always attempts to find a  $C$  and each column search always attempts to find a  $D$ : having the chain connected by the  $C$  in row  $A$  leads to the contradiction that a column search finds the  $C$  in row  $A$ .

The example assumes that originally, there was a symbol  $C$  in  $S_A$  but not in  $S_B$  and a symbol  $D$  in  $S_B$  but not in  $S_A$ . The following theorem states that this is always true given that  $r_2 \geq \max(n_1, n_3)$ .

**Theorem 2.2** Assuming the underlying switch modules are individually rearrangeably nonblocking, a three-stage  $N \times N$  Clos network is rearrangeably nonblocking if and only if

$$r_2 \geq \max(n_1, n_3) \quad (2.10)$$

**Proof:** We first consider the more specific case where the underlying modules are strictly nonblocking. The "only if" part is a trivial corollary of condition (2.3). For the "if" part, suppose  $r_2 \geq \max(n_1, n_3)$ , and we want to set up a new connection between  $A$  and  $B$ . For the existing connections, we have

$$\begin{aligned} |S_A| &\leq r_2 n_1 - 1 \\ |S_B| &\leq r_2 n_3 - 1 \end{aligned} \quad (2.11)$$

There are two cases: (i)  $|S_A \cup S_B| < r_2$  and (ii)  $|S_A \cup S_B| = r_2$ . In case (i), there is a symbol not in  $A$  or  $B$  and the corresponding middle-stage module can be used to set up the connection. In case (ii), Inequalities (2.11) yield

$$|S_A - S_B| = |S_A \cup S_B| - |S_B| \geq r_2 - n_3 + 1 \quad (2.12)$$

$$\geq n_3 - n_3 + 1 = 1 \quad (2.13)$$

and

$$|S_B - S_A| = |S_A \cup S_B| - |S_A| \geq r_2 - n_1 + 1 \quad (2.14)$$

$$\geq n_1 - n_1 + 1 = 1 \quad (2.15)$$

Thus, there is a symbol  $C \in S_A - S_B$  and a symbol  $D \in S_B - S_A$ . That is,  $C$  is in row  $A$  but not column  $B$ , and  $D$  is in column  $B$  but not row  $A$ . This situation corresponds to the one in Fig. 2.9, and we can use the rearrangement as depicted in Fig. 2.11 to accommodate the new connection.

If the underlying switch modules are only rearrangeably nonblocking but not strictly nonblocking. The existence of  $C$  and  $D$  can still be proved as above. The situation now is slightly different in that in addition to the "external rearrangements" of the connectivity switch modules of adjacent stages, rearrangements within the switch modules may also be necessary since the external arrangements may necessitate setting up of new connections in the switch modules. But the setting up of these new connections are always possible by rearranging the existing connections in the switch modules since they are by assumption rearrangeably nonblocking. This observation completes the proof.

Fig5.30c The resulting image after optimization of thresholding.

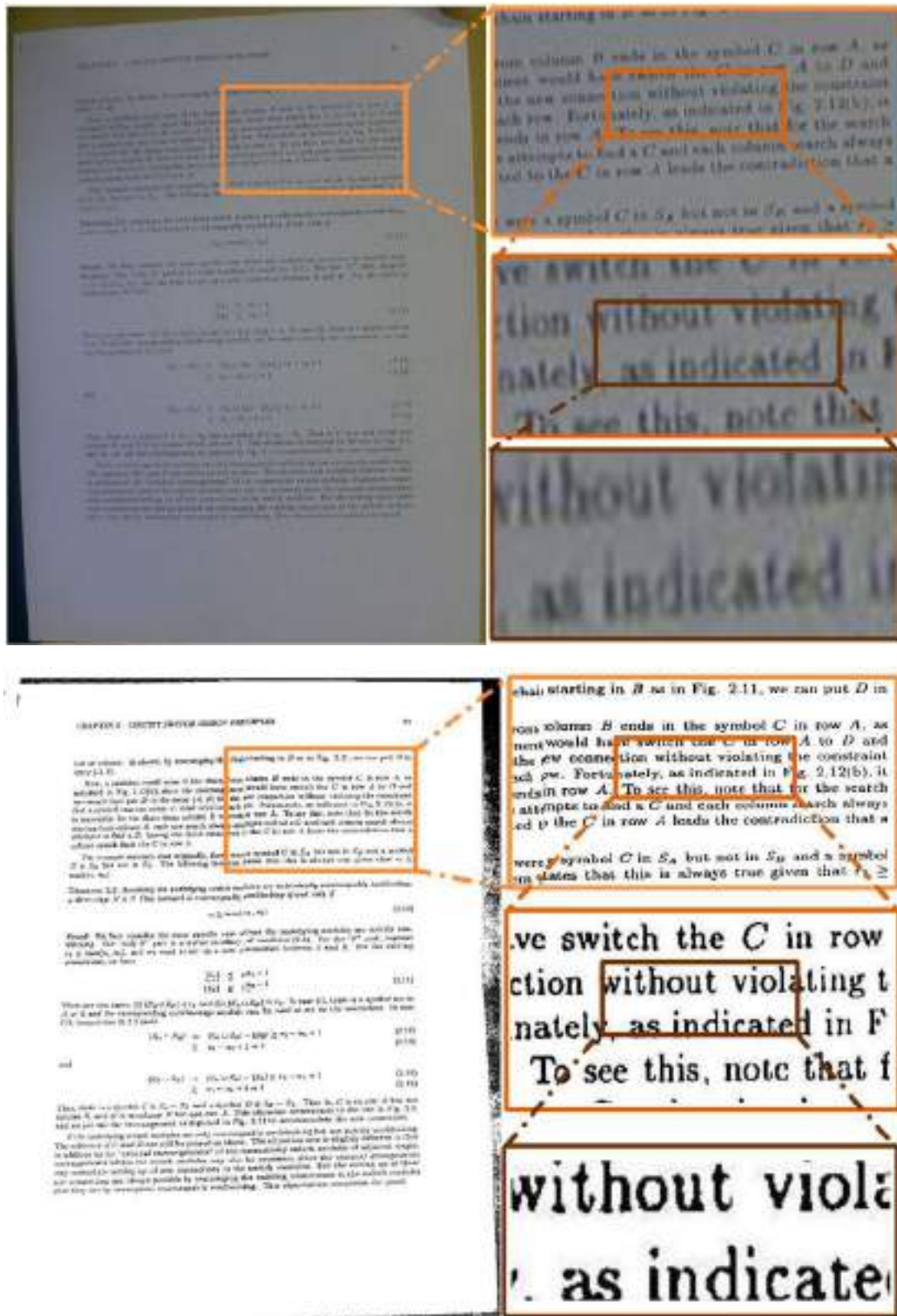


Fig5.31 Comparison of the original photo of whole document and the resulting image by 3 levels of zooming into the images.