IT 12 021
Examensarbete 30 hp
Juni 2012

Integration of face processing functionalities into relational database system Mimer SQL

Qing Gu

UPPSALA UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Integration of face processing functionalities into relational database system Mimer SQL

Qing Gu

In a relational database it is possible to store a multitude of different types of data. It would be an advantageous if the database manager provides functionality for handling specialized data (e.g. images and sounds). Currently, an increasing number of commercial relational databases provide functionalities to support clients and manipulate images. This master's thesis project is a prototype of larger project at Mimer Information Technology AB. And the project has the goal to implement a general framework for adding multimedia functionalities to Mimer SQL.

This project in Mimer SQL focused mainly on face detection and face recognitions instead of general image processing, which is the common focus of its major competitors. This project made use of external image libraries to implement image functionalities, such as face detection and face recognition, into the Mimer relational database. A User-defined-type Image with built-in image functionality methods was designed and implemented to make face detection and face recognition available in Mimer SQL for users with the use of standard SQL language. And this project set up a solid foundation for the future multimedia extensions in Mimer SQL.

Handledare: Lars Berg
Ämnesgranskare: Kjell Orsborn
Examinator: Roland Bol
IT 12 021
Tryckt av: Reprocentralen ITC

1. Introduction

With the development of information technology, multimedia technologies such as image and sound are becoming integrated into people's everyday lives. With the wide use of multimedia technology, major commercial database companies have started to integrate multimedia functionalities into their datbase products. In order to make Mimer SQL stronger and more competitive, a project about integration of face processing functionalities into the Mimer SQL database is proposed from Mimer Information Technology. This project is proposed as the first step for Mimer information Technology to integrate multimedia functionalities into database products, which concentrate on the integration image functionalities. Unlike many major commercial database companies which offer a vast magnitude of general image manipulation and content search functionalities, Mimer SQL focuses on human face processing as its first priority. The implementation for this project is based on muti-platforms: Windows, Windows CE and Linux.

1.1 Overview of Mimer SQL and external libraries

This section gives an overview of the products that Mimer information Technology provides, Mimer SQL and the external image libraries that we need in this project. We integrated two main image libraries into the Mimer SQL database in this project, which were CxImage Library and Omron Library.

1.1.1 Mimer SQL

Mimer SQL is one of most popular SQL-based relational database management systems in the world produced by Mimer information Technology AB. Mimer SQL conforms well to international ISO SQL Standards. The Mimer SQL product family currently includes Mimer SQL Engine, Mimer SQL Embedded, Mimer SQL Mobile and Mimer SQL Real time. These products support Microsoft Windows, Mac OS X, Linux, Symbian OS, Unix ,OpenVMS and others. All of these different products' functionalities are generally very similar based on various platforms.

1.1.2 CxImage Library

In this project, we make use of CxImage Library for the general image processing functionalities such as image formats conversion and image resizing. CxImage is an open source image library that can load, save, display and transform images very simply and quickly, with transparency, multiple layers and selections. In addition, it supports many different image formats such as BMP, GIF and JPEG. Actually, there are many other open source image libraries, such as FreeImage Library, OpenCV Library in addition to CxImage Library. The reason we chose CxImage is its flexibility under different platforms such as Windows, Linux, Windows CE and so on, which helps Mimer SQL to offer database products under different platforms.

1.1.3 Ormron's library

In this project, Mimer SQL focuses primarily on human face processing functionalities--not just some general image manipulation functionalities. The human face processing library is provided by Ormron Company, one of the world's leading companies specializing in image processing. The library that Mimer Information Technology acquired from Ormron provides two main functionalities:

(1) Human face detection
(2) Human face recognition

The face detection and face recognition introduction will be further discussed in the background section. At the same time, this Library supports different platforms such linux, windows and windows CE, which meets the requirement of this project. There will be more detailed descriptions of this library in the Implementation section in this paper.

1.1.4 Windows Mobile

Windows CE system is implemented in this project, and we use Windows Mobile system as the emulation to develop system for Windows CE. Windows Mobile is a compact operating system developed by Microsoft, and designed for use in smartphones and mobile devices. It is based on Windows CE, and features a set of basic applications developed using the Microsoft Win32 API. It is designed to be quite similar to desktop versions of Windows. Windows Mobile can be connected with visual studio .net. So we can debug in visual studio .net under Windows Mobile as an Emulation for Windows CE.

1.2 Synopsis of this paper

This paper describes how to integrate external image libraries into the relational database Mimer SQL. It includes five main sections, which are introduction, background, design and implementations, testing and conclusions.

The first section is the introduction above which gave a general introduction of this project including the database product and the external image libraries. The background section that follows describes the image functionalities offered by database products from the major commercial database companies. The image functionalities were explained in two ways: image manipulations and image content search. The third section was the project design and implementation. This section included two parts. One was the design and implementation of the User-Defined-Types. This part was primarily implemented on the database side. The other part is the external procedures which were implemented in C++ under Visual studio .net 2005. This part mainly describes wrapping the necessary functionalities as external procedures from external image libraries into Mimer SQL. In the fourth section, we demonstrated test cases to make use of the embedded image functionalities. The fifth and final section of this paper was the conclusions, in which we compared the functionalities we implemented for Mimer SQL in this project with other major commercial database products. Potential improvements we can achieve in the future projects for Mimer SQL are proposed in this section as well.

2. Background

With increasing volumes of multimedia data being used in the real world, images have become a common data type saved and manipulated in database management system. In order to satisfy the current requirements, most of the major commercial database management systems currently provide for some image extensions to manage images in databases rather than simply storing it as binary data. The main purpose of these image extensions is to provide image related functions to database users.

For this purpose, major commercial database systems try to encapsulate all extensions in the image data type. The image data type within the database management system is similar to struct in language C and class in C++ and Java. Users can make use of the type just like integer, varchar and float. One of the extensions that database management systems offer is the image attributes. Many of the commercial database systems offer this extension. With this extension, database users can directly make use of the image attributes. In this section, we mainly introduce two parts. First, we will introduce and compare the image extension application situation among the current major commercial database management systems. In this project, however, we mainly focus image extension about the face detection and face recognition. So in the second part, I will specifically give a background introduction about the face processing systems about face detection and face recognition.

2.1 Image extension types

Next, I shall introduce the existing image extensions and types of image extensions. Most of the major database management systems provide image functionalities as image extensions for users. These image functionalities are implemented as methods in image data type. On the current market, they are divided into two types: Image manipulations and Image content searches.

2.1.1 Image manipulation

Image manipulations are defined as the functionalities that change the content of the images. Such as Resize, Rotation, Format Conversion. Some image manipulations also include some metadata extractions. These metadata contain very detailed information about the image, such as the date when the image was taken, camera brand and other details.

2.1.2 Image content search

Image content search supports image searches within the database system. These image searches are based on different image features, such as image color, image texture and shape content. The way to implement the image content search is to compare t one or more features combined into a feature vector between two images in order to gain the similarities between images. The search result is based on the similarities between the images.

2.2 Image extensions status in major DBMS Companies

In this section, we summarize the current status of the major commercial database management systems on the market. The summary is focused on two types of image extensions.

2.2.1 IBM DB2

The image data type defined in DB2 is DB2Image and attributes accessible via the handle include the image characteristics format, height, width, and numColors.

IBM DB2 offers many extenders to strengthen the functionalities of the database server. The still Image Extender is one of the many extenders and it supports image functionalities. Based on the information from [3], [4], DB2 Still Image Extender supports both image manipulation and image intent search functionalities.

Image manipulation offered by DB2 Image Extender is very strong and outstanding. Not only does it support image content conversion such as image rotation, scaling, etc., but it also extracts the metadata from the image showing the image update time, image comment and so on.

DB2 Image Extender also supplies image intent search and it takes four features into consideration: average color, color histogram, positional color2 and texture. Basically, users can get a score for each image based on these four features combined. By comparing the scores, users can detect the similarities between two images.

2.2.2 Informix Dynamic Servers

The image extensions that Informix Dynamic Server (IDS) provides are two DataBlade modules, the Excalibur Image DataBlade Module and the Image Foundation DataBlade Module. These two DataBlade modules are independent and each emphasizes one of the image functionalities that I described above.

2.2.2.1 Excalibur Image DataBlade Module

IfdImageDesc is defined as the image data type for Excalibur Image DataBlade Module. The data type has attributes to store information about the imgformat, imgtype, pixelheight, and pixelwidth. The only image manipulations that Excalibur Image DataBlade Module [1] offers are Image scaling, format conversions, and modification of the color encoding. However, it is quite strong with regard to image content search. It supports 5 features for image intent searches: color content, color structure, texture content, shape content and brightness structure. All these features are combined into a feature vector for consideration. But the Module is not equipped for computing and storing feature vectors.

2.2.2.2 Image Foundation DataBlade Module

Image data type is defined as IfxImage. This data type contains the attributes format, pixeltype, colorspace, compression, width and height. Image Foundation DataBlade Module [2] doesn't support image content search functionality. But it supports rather strong image manipulation functionalities, including scaling, rotation, and color translation.

Overall, Infomix's two Image Modules focus on different functionalities but they serve as supplements to each other for the database system.

2.2.3 Oracle

Oracle database management system provides an image data type which is named ORDImage. Attributes in this type include source, height, width, contentLength, contentFormat, compressionFormat and mimeType

Like most commercial database systems, Oracle supports both image manipulations and image content search functionalities. The image extension that is offered by Oracle is Oracle 11g interMedia [6]. However, Oracle is overall more powerful and efficient in these two functionalities than most of the commercial database systems on the market.

Regarding image manipulations, Oracle 11g interMedia offers both image content conversions and metadata extractions, such as image format transcoding, image cutting, image scaling, and generating. And the functions are offered with comparatively more parameters which basically results in more flexibilities for users.

In order to improve the efficiency of image content search, Oracle creates an index for images and stores the index in the database. Global color, local color, texture and structure are combined as considerations in the image content search. In addition to image search based on the image content, with additional software installed, Oracle 11g interMedia also offers image facial search and this search can perform face recognition within the database.

2.2.4 MS SQL Server

As one of the major commercial database systems on the market, the latest version SQL server [8] offers data type for image. However, the data type is unfortunately just binary data and the server doesn't offer any specific image functionalities.

2.3 Face processing systems

Human face processing systems mainly include human face detection and human face recognition algorithm design and implementation. Human face detection is defined as determining the locations and sizes of human faces in images. And human face recognition is defined as identifying or verifying a person from a digital still image or video image. Human face detection is always performed on only one picture. But Human face recognition typically requires a database to store a number of pictures. Human face recognition is based on face detection because in order to identify a person's in a picture, face recognition method has to locate the people's face in the picture. When working with a picture of one person, Human face recognition compares the similarity between the people in the picture with all the people in all the pictures within the database and finds the most similar people. When building a high-efficiency face processing system, the accuracy and performance are often trade-offs and it is hard to achieve both. In response to this challenge, there has been extensive research in these areas, and this search is reviewed below.

2.3.1 Face detection

There have been many detection methods designed and implemented in the last 30 years. Early methods such as Template Matching [11], Eigen Faces [12] and Neural Network [13] were very popular. M. Turk and A. Pentland's Eigen Faces method [12] was relatively easy to implement but the accuracy was much lower under certain lighting conditions and in different sizes. Rowley's ANN method [11] and Yuille's deformable templates [13] achieved very high accuracy in face detection, but were both time-consuming and expensive due to extensive [amount of] calculations required. In recent years, researchers started to propose methods with Combined Facial features [14], [15] and achieved relatively high accuracy in real-time face detection systems. However, the methods discussed above are still limited when faces are posed in different angles. In order to detect faces with different angles, Multi-view face detection (MVFD) is used to detect upright faces in images that with ±90-degree rotation-out-of-plane (ROP) pose changes. Li et al.'s [21] pyramid-structured detector was reported as the first real-time MVFD system. Given the historical research, face detection system research is quite mature in the real time system.

2.3.2 Face recognition

Face recognition also experienced a long history of improvement. The biggest challenges for face recognition are: different illumination levels, different facial expressions and variations in the presence of cosmetics and hair characteristics. Back in 1996, Sobottka and Pitas [16] proposed a method to determine the positions of facial features by evaluating the minimum of the topographic gray level relief of the face region. The method was fast and simple to implement. However, the accuracy of facial features location is highly dependent on the gray level relief of the face region, resulting in limitation of application. Two years later, Yuille [13] proposed the deformable templates method for both facial-feature[s] recognitions, and his method gained a huge success due to its flexibility in templates and high accuracy. Nevertheless, the high-magnitude calculations resulted in slow processing performance. In more recent years, Chan and Abu-Bakar [15] tried to locate eyes regions based on the chrominance components in YCbCr color space. Their eye[s]-detection algorithm performed wavelet transformATION and made use of multilevel shareholding. Utilizing their golden ratio theory for ideal faces, Frakas and Munro can calculate the positions of all the other facial features and compare similarities among faces. The only disadvantage to this method is that threshold in multilevel shareholding algorithm was quite empirically dependent. Principal Component Analysis method [17], [19] was another popular method for face recognition. This method successfully handled both frontal and pose-angled face recognition and also resulted in relatively high performance. In [18], a new algorithm called Orthogonal Laplacianface was proposed. This method managed to not only achieve high accuracy frontal and pose-angled face recognition but also show very impressive performance.

2.3.3 The Omron Library and its face detection and recognition system

Omron's library is used in this thesis project and represents important advances in face detection and face recognition. Omron's library includes both face detection and face recognitions for both PC and embedded systems. Compared to the methods described mainly for real time systems above, Omron managed to also successfully implement a high performance and accuracy face detection and face recognition in the embedded system in cell phones specifically. Based on [21], Omron in [20] proposed a novel nesting-structured detector and succeeded in building a fast multi-view face detection system for embedded systems. This was the first face detection system for embedded system. In addition to face detection, Omron furthered the research about face recognition in [22], and in the proposed processing, illumination and pose variations are compensated to some extent. This method sacrifices some accuracy by adopting relatively fixed threshold for calculations. However, given limited memory in embedded systems, this method for face recognition managed to successfully build a sufficiently accurate and high speed face recognition system for embedded systems. And in the next section, I will introduce how to integrate Omron's library into Mimer relational database.

3. Design and Implementation

In order to fully integrate the image functionalities into Mimer SQL, the design and implementation of this project consist of two main sections. In the first section, I introduce the image data type design and implementation in SQL within the relational database. The second section is the external procedures design and implementation in C++ under .net visual studio. The external procedures basically wrapped up the outside image libraries to produce the face processing functionalities for the implementation of image data type.

Figure 1 below demonstrates the structure of the entire implementation. The Type image is the interface for users to call. It contains seven attributes and four methods. Procedures in SQL are designed to support the methods in Type image, as well as an interface to call external procedures and external procedures wrapped up in? the functionalities of the image processing libraries.

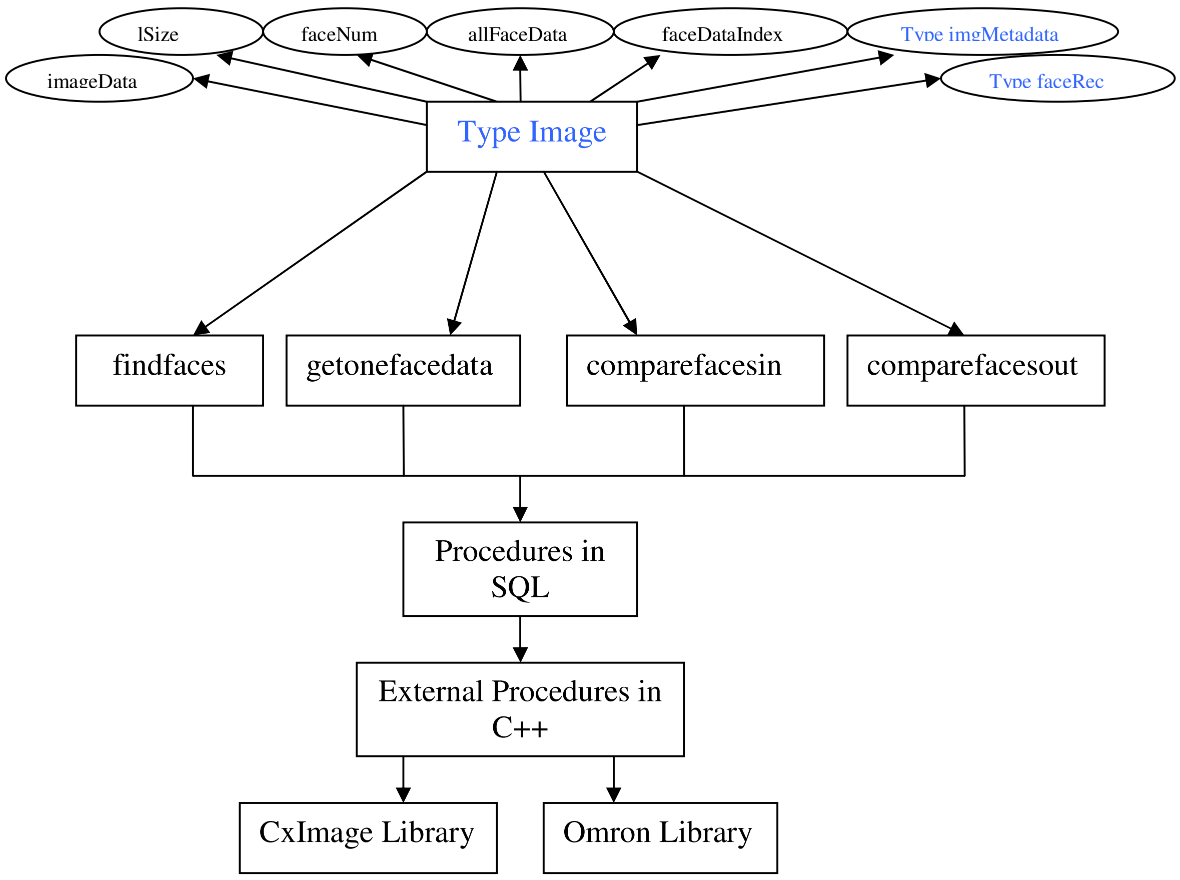


Figure 1: Implementation structure

3.1 Image data type design and implementation

As mentioned in the background, most of the database management systems offer image data type which encapsulates the image attributes and image functionalities. In this project, we also defined a specific image data type for the image attributes and functionalities implemented in? Mimer SQL. The image data type is implemented as user-defined type (UDT). Next, I will give an overview of user defined type (UDT) in Mimer SQL database system.

3.1.1 UDT Overview

A user-defined type (UDT) typically means a named data type that is created in the database by the user. A UDT may be used as the data type for columns in CREATE or ALTER TABLE statements. It can also be used in stored procedures and triggers as the type for variables and parameters. But in this project, we create a user-defined type named image internally for the purpose of providing human face image processing functionalities for customers.

There are two categories of user-defined-types: distinct and structured. A distinct type has a single data type whereas a structured type has a list of attributes. In this project, we primarily use structured types. So I would like to give an overview of structured types. This summary of the definitions and descriptions for using structured types is maily referenced from Mimer information AB document.

The structured type syntax is:

CREATE TYPE type-name *representation* [*type-attributes*] [*method-specification-list*]

A method returns a single value and can thus be used wherever an expression is allowed. There are three types of methods: static, constructor and instance methods. There must exist a method specification for the type that matches the method with regard to parameters and result type. As it is possible to have method specifications with parameter overloading there must be an exact match. If no method modifier is specified, instance is the default. The main difference between these types of methods is how they are invoked.

When a structured type is created, a constructor function is also created. A constructor method must have the same name as the user-defined type to which it belongs and it must have a returns clause that specifies this type. This method is invoked when the type is used and it will initialize all attributes with their default values. If an attribute was not given a default value it will be set to null. Instance methods involve the implementations of certain functionalities regarding attributes of the type, which are quite similar to the non-static member functions of C++ classes.

3.1.2 UDT definitions

In this project, we designed a total of three user-defined types as well as some procedures. These procedures are defined to call external procedures implemented in C++. These external procedures have the same names as the corresponding procedures. They are responsible for wrapping functionalities from external libraries such as CxImage and Ormron. Type image is defined as the major type for image functionality extension in this project and is also the interface for the user to make use of all the face processing functionalities. Type imgMetadata and Type faceRec are defined mainly to support Type image.

3.1.2.1 Type Image

Next, I shall summarize the user-defined types defined in this project. The most important user-defined types and the very first defined one is type image. Type image includes all central attributes. The definition is as below:

create type image(imageData blob, lSize integer, faceNum integer, allFaceData blob, faceDataIndex binary varying(1000), metadata imgMetadata, rectangles faceRec);

Type image includes seven attributes. ImageData is a blob holding the content of the image. In other words, ImageData is the image itself. lSize represents the memory size of the blob ImageData. Attribute faceNum represents the number of faces. Attribute rectangles is a set of detected faces' rectangular coordinates.

Before explaining the meaning allFaceData, face data needs to be explained. Face data from a face holds a combined feature vector of the face, which is used mainly for face recognition. Different faces have different combined feature vectors, so a combined feature vector is used to identify a face. The similarity between two faces can be calculated based on the two faces' combined feature vectors. allFaceData is built like an array to save the face data of all the faces within the image. The face data of all the faces will be saved in order. Attribut faceDataIndex is also built like an index array to store integers representing the size of each face's data. The order of the sizes in FaceDataIndex is in accord with the face data in allFaceData. The last two attributes, metadata and rectangles, are defined as Type imgMetadata and Type faceRec, which are explained below.

3.1.2.2 Type imgMetadata

The second type created is the imgMetadata type. imgMetadata type is used for the metadata of images. Images in different formats tend to have different metadata. These metadata normally contains a lot of information about the image itself. The definition of the imgMetadata type is as below:

create type imgMetadata(dateTime Timestamp, format char(20), camera char(20), height integer, width integer)final;

The attribute Timestamp is the time when the image is taken. Attribute format represents the format of the image. Attribute camera specifies the brand of the camera which was used to take the image. Attribute height and width are the height and width of the image.

3.1.2.3 Type faceRec

The last type we designed and implemented is type faceRec, which is used to hold the face detection result. The definition is as follows:

```
create type faceRec(x integer, y integer ,width integer, height integer)final;
```

In Ormron's Library, after having the image data as input, type faceRec? detects the positions of faces from the data. Then it outputs the number of faces detected and rectangular coordinates (coordinates of endpoints for the face rectangle) of each face. Image's origin (x, y) = (0, 0)) is in the left top corner, and it extends from left to right and from top to bottom. Attribute x and y of type faceRec represent the pixel(x, y) of the left top point of the rectangular for the detected faces. The picture below shows how it looks like when a face is found after the face detection process is complete.

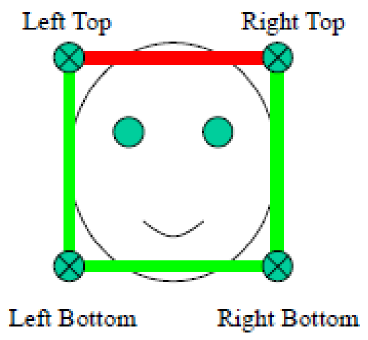


Figure 2: Face detection sample

3.1.3 Methods definitions of Type Image

Methods to type are similar to functions to class in C++. In this project, as to Type image, in addition to the attributes, it has also defined methods to implement the functionalities that we want to support in the Mimer SQL database engine. These methods can be called by Mimer SQL users to obtain the image processing functionalities. Also, within the definition of the methods, it called a few procedures defined as an interface to call external procedures. These procedures will be explained in detail later.

3.1.3.1 Constructor method

The purpose of the constructor method is to initialize the attributes of the types. The declaration for the Constructor method is shown as follows:

```
create constructor method image (imagedata blob, lSize integer,  process Boolean ) for
image
```

In the Constructor method, imagedata is a blob type which holds the image itself and lSize is the size of the image. With regard to the parameter process, imagedata is used to decide whether there is preprocess for the image in the Constructor method.

Memory management and performance are important for a system. Our system supports both windows and windows CE platforms. Especially for the windows CE platform, the memory management and performance are quite essential and critical. Face detection and recognition are very expensive and time-consuming processes for database. So when we consider the design, we proposed a preprocess way to solve the problem. Then what does preprocess mean? The preprocess here means that the system process the face detection or face recognition in the background. To give more detail, the system starts to deal with the process right after the picture is loaded into the database, and the face detection and face recognition result will be saved in the system so that when the face detection and face recognition functions are called, the system only needs to retrieve the saved result.

The Constructor method first initializes imageData and lSize attributes by getting values from the parameters. Second, the metadata attribute of the image type is initialized by calling procedure getMetada (imagedata, lSize) and getWidthHeightFormat (imagedata, lSize, hi, wi, format). Third, if parameter process is set to be 0, it means there is no preprocess. Then the Constructor method is complete and the rest of the attributes are left to be 0 or NULL. However, if parameter process is set to be 1, it means there is preprocess. It will call method findfaces and procedure getallfacedata. Method findfaces will initialize attribute rectangles while getallfacedata will initialize allFaceData, faceDataIndex and faceNum.

3.1.3.2 Instance Method findfaces

Method findfaces is defined to implement face detection for the images. And it is declared as:

instance method findfaces (imagedata blob, lSize integer) returns faceRec

Within method findfaces, it will call procedure findfaces(imagedata, lSize , recs). This procedure will call an external procedure that gets face detection functions from Ormron's Library and the returned result will be a set of rectangles which represent the detected faces and these rectangles will be saved in recs. Then the value of recs will be assigned to attribute rectangles.

3.1.3.3 Instance Method getonefacedata

The purpose of Method getonefacedata is to get face data for a certain face within an image based on its index number.

instance method getonefacedata (index integer) returns varbinary

As I described previously, attribute allFaceData holds face data for all the faces within the image. The index decides which face data will be retrieved. Based on the allFaceData and faceDataIndex, getonefacedata can get the specific face data with the index. But if allFaceData and faceDataIndex are NULL, which means there was no pre-process in the Constructor method, the procedures getallfacedata will be called, and attribute allFaceData and attribute faceDataIndex will be assigned values.

3.1.3.4 Instance Method comparefacesin

Method comparefacesin is an instance method that is used to determine the similarity between two faces within the same image. It's declared as follows:

instance method comparefacesin (index1 integer, index2 integer) returns integer

This method can be called only if two faces that need to be compared are both in the same? image. Basically, it first checks to see if the attributes allFaceData and faceDataIndex exist. If so, it will get face data for both faces based on the index parameters and then call the procedure matchfaces to get a number which indicate the similarity between the two faces. Otherwise, it will first call procedure getallfacedata to initialize the attribute allFaceData and faceDataIndex and then go through the same process to get the similarity number.

3.1.3.5 Instance Method comparefacesout

Method comparefacesin is an instance method to get the similarity between one face within the image and the other in another image. It's declared as follow:

instance method comparefacesout (index integer, facedata varbinary(1000))returns integer;

comparefacesout method implementation is basically the same as instance method comparefacesin except that one of the faces is not in the same image. The face not in the image needs to provide the face data for this function.

3.1.4 Procedure definitions

At this point, the entire image type definition is complete. There are also some procedures mentioned above that are created to call the external procedures from the other part of the implementation. These external procedures implemented in C++ are named the same as these procedures. As mentioned above, they are called to within the instance methods and serve as the interface for the database to call external procedures which contain the image functionalities. So in order to help understanding, these procedures' definitions are listed below:

create procedure getMetadata(in imagedata blob, size) returns table(name varchar(30), mvalue varchar(30)) external;

```
create procedure getallfacedata (in imagedata blob, out allFaceData blob, out
faceDataIndex binary varying(1000), out faceNum integer ) external;

create procedure getfaces (in imageData blob) returns table(x int, y int, height int, width
int) external;

create procedure findfaces(in imageData blob, in lSize integer,  out  recs faceRec)
external;

create procedure getonefacedata(in allFaceData blob, in faceDataIndex binary
varying(1000), in index integer, in imagedata blob, out facedata varbinary(1000))
external;

create procedure getWidthHeightFormat(in imagedata blob, in lSize int, out height int,
out width int, out format char(20) )external;


create function matchfaces(facedata1 binary varying(1000), facedata2 binary
varying(1000)) returns int external name matchfaces;
```

3.2 External Procedures' design and implementation

As discussed above, in order to design and implement the UDT (User Defined Types) for images and integrate the Face detection and Face recognition Library into Mimer relational database, we need to build external procedures. This section is mainly summary of the design and implementation of the external procedures. The external procedures are designed and implemented based on the integration of the Omron face processing library and CxImage image processing Library. The summary of the external includes two sections: (1) Face detection, (2) Face recognition

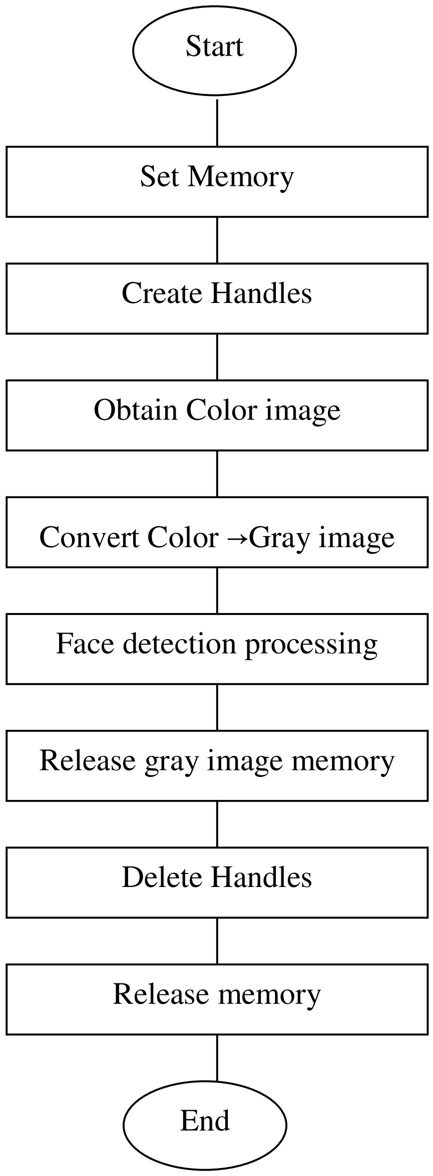


Figure 3: Face detection flow

3.2.1 Face detection

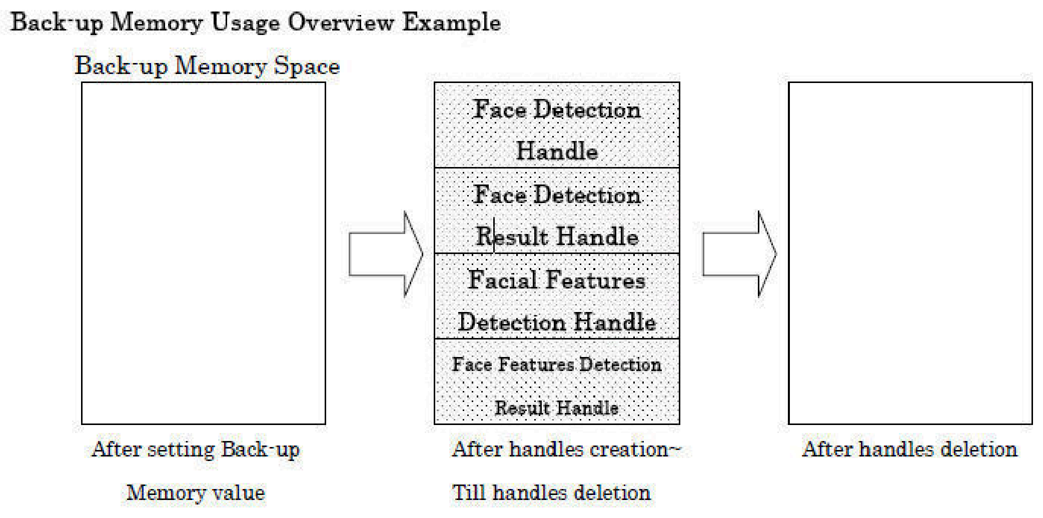
The entire face detection process is implemented within the external procedure findfaces in C++. findfaces will return all the detected faces information. The whole process is described based on the Face detection flowchart (Figure 3). The Face detection flow is quite straight-forward.

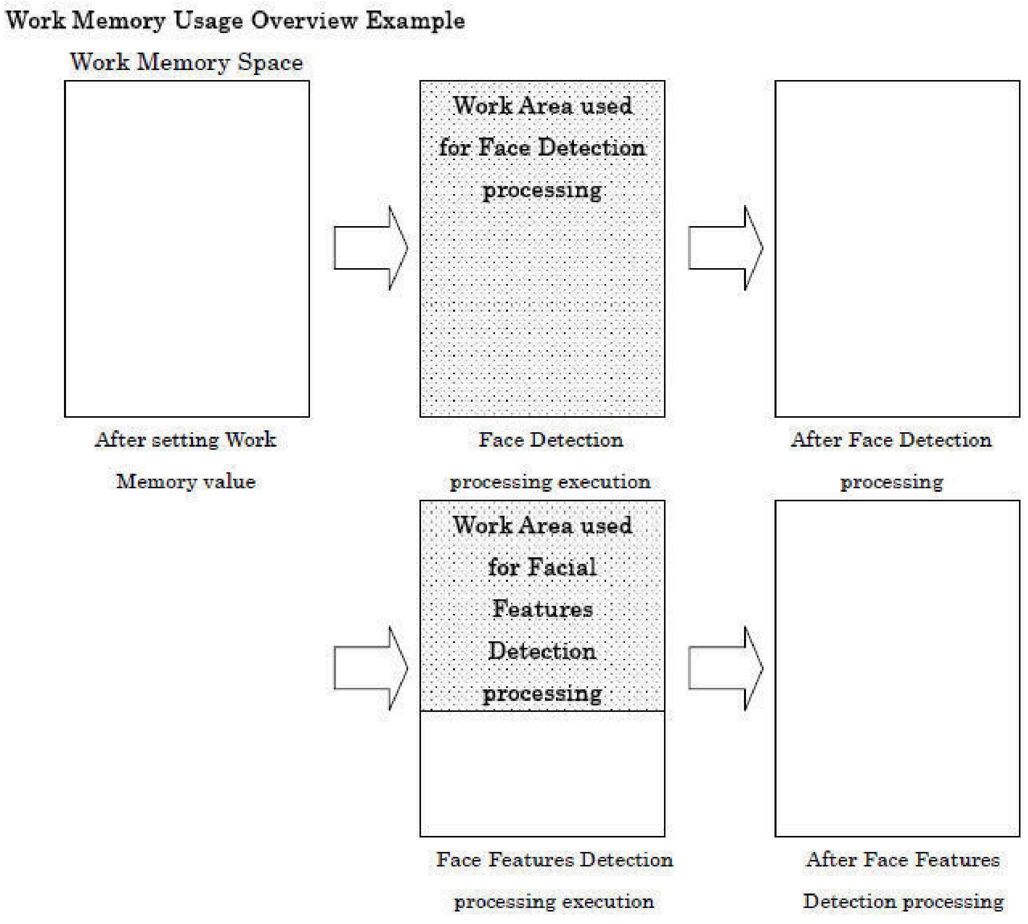
3.2.1.1 Memory Management

In order to make use of the library, the program must first allocate necessary memory for the entire face detection process. Memory allocated in this process can be defined into two types: (1) Back-up Memory (2) Work Memory. Back-up Memory is defined to store handle data while Work Memory area is temporarily used by each function.

When we initialize the memory for the process, the starting address and the memory size should be set for each of these memory areas. In the face detection process that follows, the Create Handles and the Obtain Color image steps are preparation steps for the face detection process, which requires the allocated memory. The memory allocation settings should be completed before the handles are created and the allocated memories should be released after deletion of the handles. Handles are defined to hold certain results in the process. For example, in the face detection process, a Face detection handle will be created to save the result of the detected faces.

Figure 4 show the Memory Usage. From the Back-up Memory Usage Overview flow chart, we can clearly see that after backup memory is set, and all the handles needed in the program are created and saved inside the Back-up Memory. After the process is complete, all the handles will be deleted. Whereas, from the Work Memory Usage Overview flow chart, we can see that after the Work memory is set, all the functionalities such as face detection and facial features detections functions are working within the Work Memory. When the entire process is complete, the Work Memory will be deleted.


Back-up Memory Usage Overview Example
Back-up Memory Space
Face Detection Handle
Face Detection Result Handle
Facial Features Detection Handle
Face Features Detection Result Handle
After setting Back-up Memory value
After handles creation~ Till handles deletion
After handles deletion



Work Memory Usage Overview Example
Work Memory Space
Work Area used for Face Detection processing
After setting Work Memory value
Face Detection processing execution
After Face Detection processing
Work Area used for Facial Features Detection processing
Face Features Detection processing execution
After Face Features Detection processing

Figure 4: Memory Usage

3.2.1.2 Convert Color Image to Raw Image

The next step that needs to be explained in more detail is the Obtain Color image. Because when we get a color image, it could be in any format and it could be compressed image and decompressed image. But there is a special requirement for the color image of the image processing Library from Ormron Company. So there needs to be some preprocessing before the real face detection processing. As mentioned in the introduction section, CxImage Library is an open source library for image manipulations such as format conversion, resize, rotation and so on. In this section, we use CxImage to convert image from other formats such as jpeg, png, and tiff to bmp format. After conversion into bmp format, an image is not necessarily a raw image. So we need to do some additional conversion as well. Next, I will give about a summary of the raw image requirement for the face detection processing

3.2.1.2.1 Input requirements of image processing library

Immediately after we obtained the image processing Library from Ormron Company, the very first thing we did was to understand the library. The input image parameter for this Ormron Company's image processing library is a 24 bits raw image. According to the library, a raw image should not contain any other information about the image expect for the bitmap data in bytes. Also, according to Ormron Company's image processing library's requirements, the raw image's origin (x, y) = (0, 0)) extends from left to right and from top to bottom. The picture below gives us a general idea of the pixels layout of a raw image.

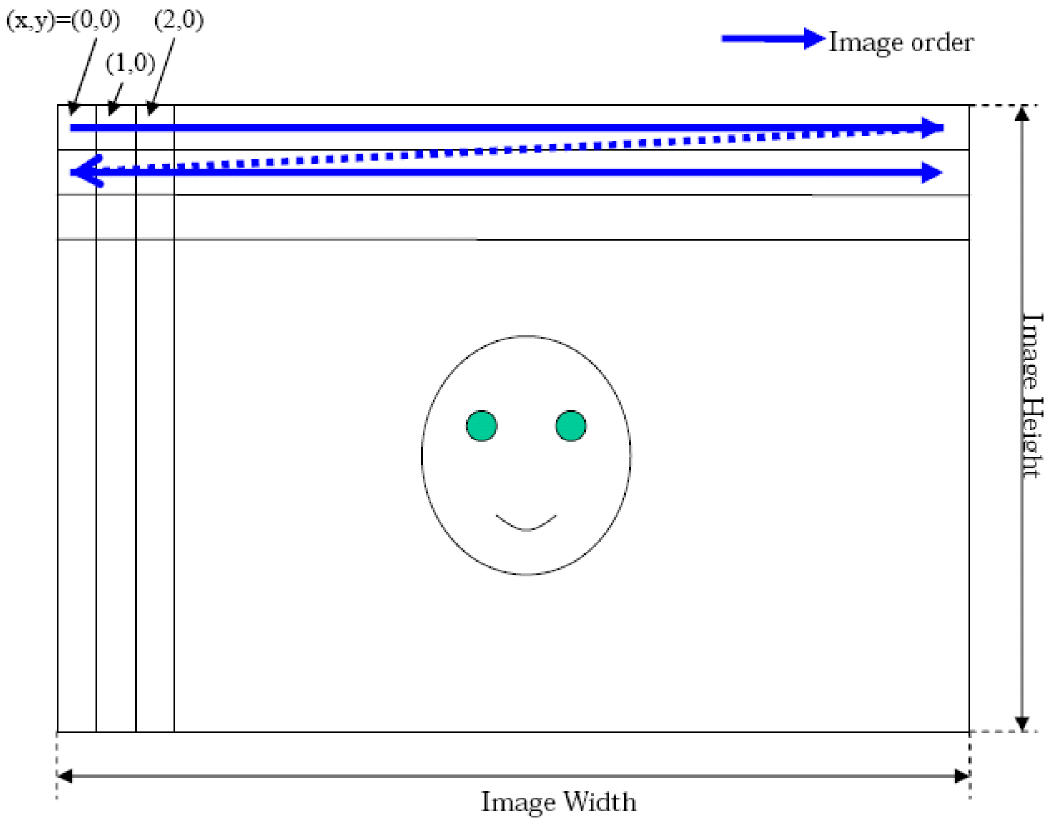


Figure 5: Raw Image

3.2.1.2.2 Introductions about BITMAPINFOHEADER

As mentioned previously in the summary of the input requirements, the library needs a 24 bits raw image to process. After the conversion of the image format by CxImage, we get a bmp format image. The bmp format image is also known as Bitmap. The Bitmap Image File consists of fixed-size structures (headers) as well as variable-size structures appearing in a predetermined sequence. Many different versions of some of these structures can appear in the file, due to the long evolution of this file format. Now we introduce one very common fixed-size structure Header BITMAPINFOHEADER.

BITMAPINFOHEADER structure is defined in C as a structure that contains information about the dimensions and color format of a device-independent bitmap. A device-independent bitmap is used to store bitmap digital images, independently of the display device. So in this project, using windows, 24 bits bitmap actually means that the biBitCount of the BITMAPINFOHEADER structure is 24. The BITMAPINFOHEADER structure is defined as below:

```
struct tagBITMAPINFOHEADER {
        unsigned long  biSize;
        long           biWidth;
        long           biHeight;
        unsigned short biPlanes;
        unsigned short biBitCount;
        unsigned long  biCompression;
        unsigned long  biSizeImage;
        long           biXPelsPerMeter;
        long           biYPelsPerMeter;
        unsigned long  biClrUsed;
        unsigned long  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

From the demonstration of the structure above, biBitCount determines the number of bits that define each pixel and the maximum number of colors in the bitmap. biBitCount specifies the number of bits-per-pixel. Different value of biBitCount has different meanings for the bitmap. The definitions are as follow:

1 Bitmap means monochrome, so the color table contains only two entries. Each bit in the bitmap array represents a pixel.

4 Bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a 4-bit index into the color table

8 Bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a 1-byte index into the color table.

24 Bitmap has a maximum of 2^24 colors. Each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, for a pixel.

In everyday life, images we get are either gray images or color images. Typically, gray images' biBitCount is 8 and the color images' biBitCount is 24.

3.2.1.2.3 Implementation to obtain raw image

As mentioned previously, the input image parameter for this Ormron Company's image processing library is 24 bits raw image and CxImage Library can convert the images from other bits value to 24 bits value as well as convert the image formats. So what we do first is to load the input image as a CxImage object. Then we test the image format, and if it's not bmp format, and we convert it into bmp format. Finally, we test the image's biBitCount to see whether it's 24 bit, and if not, we convert it into 24 bit.

But at this time, when we ran the Ormron library to test pictures for face detections, we still could not get the right result and the result was always no faces detected. So we noticed that there could be some other conversions we would need to do in order to make Ormron image Library work.

What caused the problem? The answer is the zero-padding. In windows bitmap, the bit values for the bitmap are stored as 4 bytes aligned and each byte equals 8 bits, which basically means that each scan is a multiple of 32 bits. But in the real image, each line is not necessarily 4 bytes aligned. So actually, in each scan line, whenever it's not 4 bytes aligned, it will be zero-padded by the end of the scan line. When we do the human face image processing, we need the real image without any zero-padding in the end. So what we need to do is to detect and eliminate all the zero-padding.

Now, the implementation should be described. First of all, we need to calculate the result from a formula in C:

```
remainder = biWidth * nDepth % 4
```

When the bits-per-pixel is 24, nDepth is 3 while when bits-per-pixel is 8, nDepth is 1. But the CxImage Library converted the bits-per-pixel into 24 beforehand. Therefore, nDepth here can only be 3. biWidth specifies the width of the bitmap, in pixels. When nDepth is 3, each pixel takes 3 bytes. So biWidth multiplying nDepth shows the number of bytes the image actually takes in each line. If remainder=0, it means the image data itself is 4 bytes aligned. If remainder is not zero, it means the image data is not 4 bytes aligned and there are zero-padding in the end. If remainder=i, $(0<i<4)$, it has 4-i bytes of zero-padding at the end of each scan line. What we need to do is to get rid of all the zero-padding. After the zero-padding removed, the image becomes the real raw image.

3.2.1.2.4 Face detection processing

With the raw image input ready, the face detection process can start. In the external procedure, a struct is defined as follow to save the results for face detection:

```
Struct OmronFaceDetectionStruct{
   HDTRESULT   hDtResult;
   RAWIMAGE *GrayImg;
   int grayImgWidth;
   int grayImgHeight;
   INT32 nCount;
};
```

Based on the face detection flow chart above, after an image is converted into a raw image, it will be converted into gray image by CxImage. And the gray image will be copied to a piece of memory pointed by pointer GrayImg in Struct OmronFaceDetectionStruct. At the same time, CxImage will return the length and width of the gray image, and the results will be also saved inside Struct OmronFaceDetectionStruct as grayImgHeight and grayImgWidth individually. A face detection process will be performed on the converted gray image. After the process is complete, all the detected face information is saved inside a HDTRESULT handle. The detected face information will be copied from HDTRESULT handle to the handle hDtResult within OmronFaceDetectionStruct as well. In the end, the number of faces detected will be saved into nCount within OmronFaceDetectionStruct. From the hDtResult which holds the detected face information, we can always get very detailed information about all the detected faces. To be detailed, we can loop through the detected faces and get each of the four points representing the rectangle for each detected face. The four points of a detected face are shown in Figure 2. The detected face rectangles will be returned to the methods for image type in SQ by the procedures. After the entire face detection process is complete, both the work memory and the back-up memory should be released in the end.

As we described previously, users can choose whether or not to do preprocess. Face detection is the main part of preprocess. If users choose to do preprocess on a certain image, there is an external procedure findfaces that will be called. The face detection process will be performed on this image and all the detected face information will be saved in a Struct OmronFaceDetectionStruct.

3.2.2 Face recognition

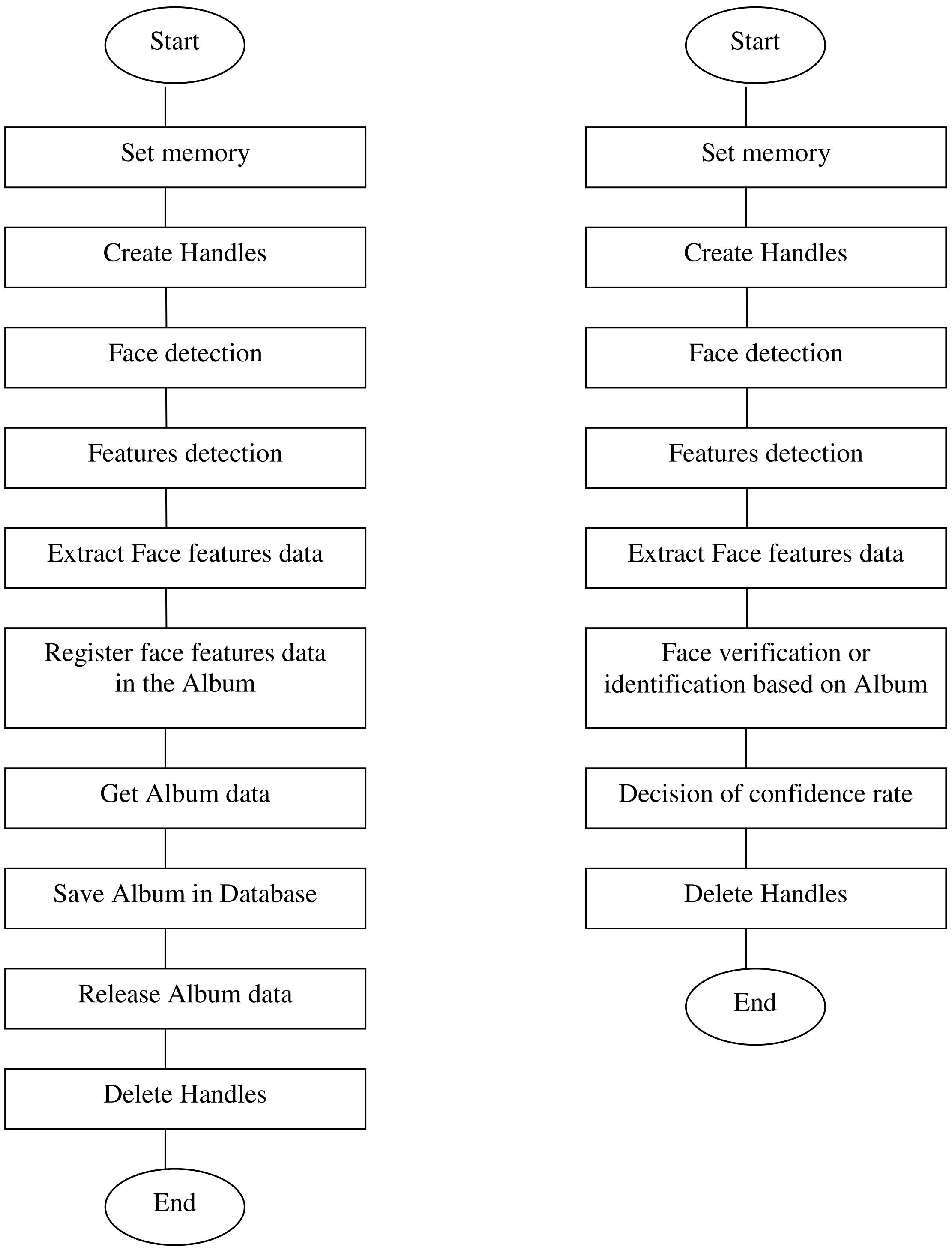


Figure 6: Face registering Flow and Face verification Flow

Face recognition is the identification of or verification of a person in a digital still image or video image based on an existing face image database. Omron library completes two steps in order to perform the face recognition process. The first step is to build a face features database which is called album in the library. So this step is called face registering. The second step is to verify faces based on the existing database. So the second step is called face verification.

3.2.2.1 Face registering

First-, let's look at the Face registering flow. Similar to the face detection flow chart, memory allocation for both Back-up memory and work memory are both necessary. Generally, the Face registering flow builds the Album for face recognitions. From Ormron library's perspective, Face registering basically means registering faces into the Album. The Album defined in this library is treated as a database in this project. So Face registering is building this database defined as the album described in the Ormron's library. We don't save the face data into the Album; instead, we save them in our database. When you need to identify or verify a face in an image, it must be a database with faces first. In a given image, after doing all the preparations for the face processing, the Face detection step involves detecting all the faces within the image. For each detected face, face features data are extracted. When we mention face data in the project, it's actually short for face features data. So face data and face features data mean the same thing. In the end, face data from all the faces will be saved in our database.

In this project, the implementation for the face registering process is basically done in the external procedure getallfacedata. After the face detection process is complete, all detected face information is saved in handle hDtResult within Struct OmronFaceDetectionStruct. Within external procedure getallfacedata, all the face data is gained based on handle hDtResult and saved in an array. In addition, another array is initialized to record the size of each face data. In the end, the array for face data will be passed to attribute allFaceData within image type while the array for the size of each face data will be passed to attribute faceDataIndex within image type.

3.2.2.2 Face verification

Second, the Face verification Flow is illustrated to describe the face verification process. Given an image, we first detect the faces within the image. When we describe face verification, we mean that the system recognizes one face within the database. So we also need to choose one face in the image to do the face recognition. Then for the chosen face, we do face features data extraction. Based on the extracted face features data, we can do verification and identification within the existing Database. To elaborate, according to the specification, we can calculate the similarity between a given face and other faces in the same image or in other images. The degree of similarity between two faces will be represented by a score between 0 and 1000. The higher the score is, the more similar the two faces are. The result of verification and identification should be the most similar faces from the Database. The number of returning faces can be specified by the users.

In this project, most of the implementation for the face verification process has been explained. When we get an image, we first do face detection and features detection to extract the face data and save all the data into a database. One process new here is to calculate the similarity between the images. It’s mainly implemented in the external procedure matchfaces. It takes face data from two faces and returns a score to show the similarity between the two faces. As mentioned above, procedure matchfaces is called within both instance methods comparefacesin and comparefacesout.

Generally, the face registration process is the face data database building process. After face database is built, we do face verification within the face data database based on one face specified.

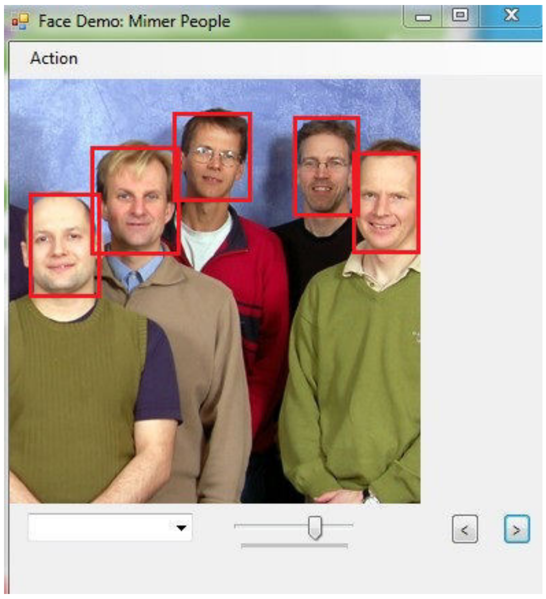
4. Testing Results

4.1 Demo database

We built demos separately under .net to show the results of the face detections and face recognitions. In order to do the testing, we first built a simple database named demo in Mimer SQL. One table named “images” in this database includes three columns: image_id(int), image(blob) and image_Size(float). We inserted about 20 pictures into this database and each picture is one record in Table images. A second table named “contacts” is defined and included columns: contact_ID(int), image_ID(int), facedata(blob), x(int), y(int), width(int), height(int). For pictures in the table images, we initialized each of the pictures as an instance of Type image, call constructor method image (imagedata blob, lSize integer, process Boolean) and set the parameter process to 1. As described previously, when we set process to 1 within constructor method image, the face detection process is initialized and detected faces are saved in Attribute faceRec within Type image. At the same time, the constructor method image will initialize Attribute allFaceData, faceDataIndex in Type image. allFaceData contains the? combined feature vector of all the detected faces. Following the? Instance method of Type image, getonefacedata returned facedata for each detected face in the picture. As a result, all the detected faces from all the pictures in Table images were inserted into the table contacts together with the detected rectangle information and facedata. And each row in the table contacts would be one detected face.

4.2 Face detection

The Procedure Call_findfaces (image_id) was created to retrieve the detected face information for the demo. The stored procedure Call_findfaces retrieved and returned column x, y, width, height from table contacts based on the corresponding image_id. The pictures below were saved in the table images. The application below connected to the database demo we created above. And the pictures were directly retrieved from the table images. When we click the face detection button, the application will call Procedure Call_findfaces. In the left below, a group of people from Mimer were in the picture and the detected faces are marked by red rectangles. We can see that all the faces in the picture were detected. The picture in the right below was randomly downloaded from the internet. Whereas, a couple of faces were not detected due to the angle of the faces.



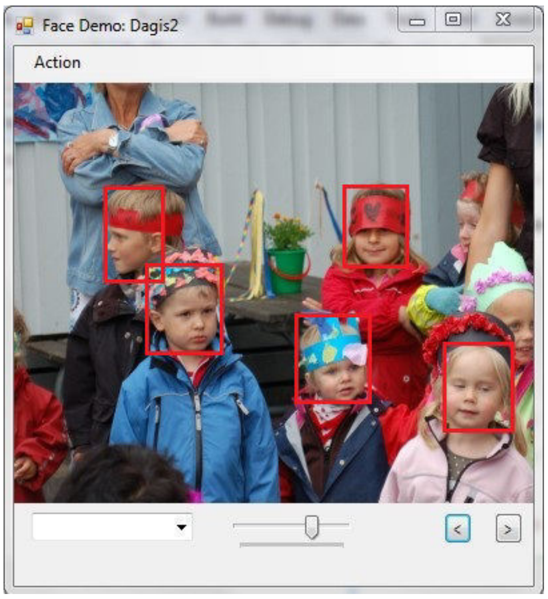


Figure 7: Face detection demo

The demo code to call Call_findfaces implemented in C# and to show the detected faces is displayed below:

```
MimerCommand cmd = null;
DataTable detectionTable;
MimerDataAdapter detectionAdapter;
int x, y, width, height = 0;
this.markFace = false;
            try
        {
          cmd = new MimerCommand("CALL call_findfaces(:image_id)", con);
          cmd.Parameters.Add(":image_id", currentImageId);
          detectionAdapter = new MimerDataAdapter(cmd);
          Image img = this.pictureBox1.Image;
          detectionTable = new DataTable();
          detectionAdapter.Fill(detectionTable);

                    if (detectionTable.Rows.Count > 0)
          {
            foreach (DataRow row in detectionTable.Rows)
            {
              x = (int)row.ItemArray[0];
              y = (int)row.ItemArray[1];
              width = (int)row.ItemArray[2];
```

```
                height = (int)row.ItemArray[3];

                markFaces(img, x, y, width, height);
            }
            this.pictureBox1.Image = img;
        }
        else
        {
            MessageBox.Show("No faces found", "No faces");
        }
    }
    catch (MimerException me)
    {
        ShowError(me);
    }
```

4.3 Face recognition

The procedure search_contacts(contact_id, similarity_threshold) was created to test face recognition. Basically, we tried to pass a contact_id as a face to identify the most similar faces in the database. As mentioned in the previous section, the degree of similarity between two faces will be represented by a score between 0 and 1000. The greater the score, the more similar the two faces. So in order to identify to most similar faces, we specified a parameter similarity_threshold, as a threshold for the similar faces. Within this procedure, we retrieved facedata from the table contacts based on the specified contact_id, and then looped through the entire table contacts to call the function matchfaces comparing the similarity between the specified face and all the other faces. Each call will return one similarity score. So in search_contacts, if any face that has a similarity score greater than similarity_threshold, the face's corresponding information in column x, y, width, height in table contacts will be returned.

We generated two test cases. The test faces we specified in the first test case are shown below at left, and we specified similarity_threshold to be 700. And the results are shown below at right. From the testing, we found out that the same person in the database can be identified.

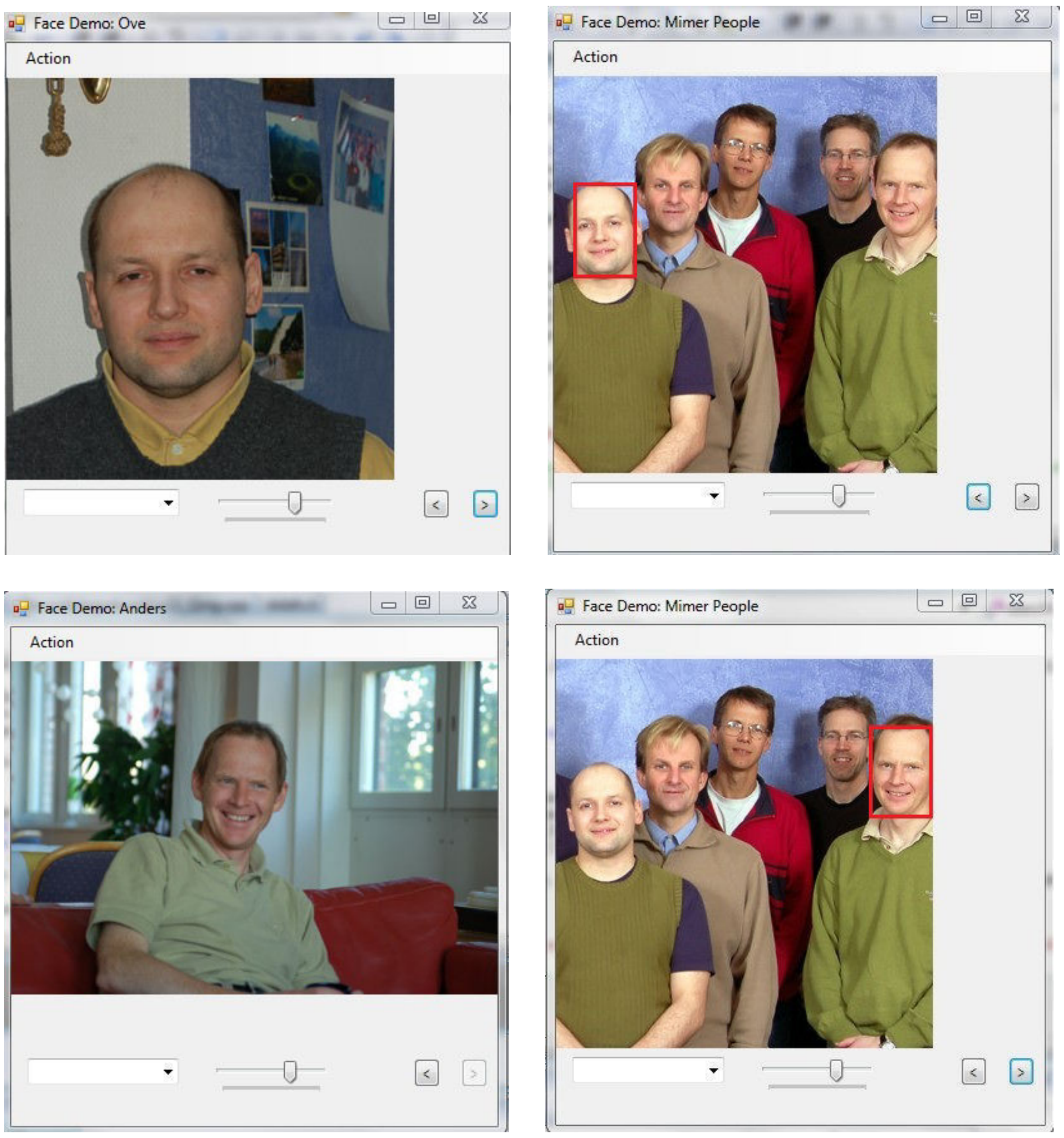


Figure 8: Face recognition demo

The demo code in C# is as follows:

```
MimerCommand searchCmd = null;

        int x, y, width, height = 0;
        this.markFace = true;
        try
        {
            if (searchType == FaceSearch.NonIndexedSearch)
    {
              searchCmd = new MimerCommand("CALL
            search_contact(:contact_id, :similarity_threshold)", con);
    }
    else
    {
        MessageBox.Show("Wrong search type", "Error");
    }
    searchCmd.Parameters.Add(":contact_id", (int)comboBoxContacts.SelectedValue);
    searchCmd.Parameters.Add(":similarity_threshold", this.similarityThreshold);
    imageAdapter = new MimerDataAdapter(searchCmd);

    imageTable = new DataTable();
    imageAdapter.Fill(imageTable);
    currentImagePos = 0;
    if (imageTable.Rows.Count > 0)
    {
        currentImageId = (int)imageTable.Rows[0].ItemArray[0];
        buttonPrevImage.Enabled = false;
        if (imageTable.Rows.Count > 1)
        {
            buttonNextImage.Enabled = true;
        }

        showImage(currentImageId);
        x = (int)imageTable.Rows[0].ItemArray[3];
        y = (int)imageTable.Rows[0].ItemArray[4];
        width = (int)imageTable.Rows[0].ItemArray[5];
        height = (int)imageTable.Rows[0].ItemArray[6];

        markFaces(this.pictureBox1.Image, x, y, width, height);
    }
    else
    {
        MessageBox.Show("No matching faces found", "No match");
    }
}
catch (MimerException me)
{
    ShowError(me);
```

```
}
```

5. Conclusions and future work

5.1 Conclusions

This project was a master's thesis research project designed by Mimer information Technology. This project was also a prototype for a bigger multimedia functionalities integration project within Mimer taking place in the next a few years. In this project, we integrated some image processing functionalities--mainly face detection and face recognition functionalities into relational database Mimer SQL for multi-platforms including windows, windows CE, and Linux.. The project included two primary sections:

1 User Defined Types design and implementations
2 External procedures design and implementations

The first section focused on the database engine side which was implemented in standard SQL language, and we designed and implemented three user defined types for this part. The implementation included the definition of the types, the methods of the types and some procedures. The methods were implemented to call external procedures implemented by the outside image libraries. With the libraries, these methods supported not just some basic image manipulations but also face detection and face recognition functionalities for users of Mimer SQL.

The second section was the external procedures implementation component. This section was implemented in C++ under visual studio .net. To fulfill the functionalities, we made use of two external libraries. The human face processing functionalities are supported by the library from Omron. This library included two parts: face detection and face recognition, which are the main functionalities we want to support within the Mimer SQL database. CxImage library was the second library. In this project, CxImage was primarily responsible for providing image manipulation support to help wrap the face detection and face recognition functionalities. We used it to convert images between formats and from one color bit to another so that we could convert images from normal compressed to raw image.

Unlike many of the database companies who offer general functionalities for both image manipulation and image content search, this project designed by Mimer information Technology was focused on human face processing functionalities. And these human face processing functionalities are not offered by most the database companies. The design for the current project built a structure for the future extended multimedia project mentioned above.

5.2 Future works

This project is prototype for integrating image functionalities into the relational database Mimer Information SQL. Compared with many of the image functionalities offered by

the major commercial database companies in the markets, there is a lot of room for us to expand and improve this project. Generally, we can expand and improve in two areas:
(1) Image manipulation
(2) Image content search

5.2.1 Image manipulations

In this project, we mainly implemented the image functionalities. However, many of the image manipulation functionalities can be extended as a result of this project. In the implementation, we used an external library CxImage which is quite powerful with regard to image manipulations. The external library doesn't have to be CxImage and we can choose a certain library that fits the project the best. What we need to do in the extended project is to add a number of methods to image type such as resize, rotate and format conversions to strengthen the functionalities of image type. At the same time, we also need to implement external procedures for the corresponding methods based on the image library we choose. The design of the extended project would be quite similar to the current one.

5.2.2 Image content search

In this project, the prototype we implemented offers image content search because human face recognition functionality is actually a special kind of image content search functionality. Typically, when we mention image content search, we are referring to image comparison based on the whole image content while Human face recognition is referring to image comparison based on the faces in the image. When you perform image content search, you get the most similar images from the database based on one image content, whereas when you perform face recognition, you get images with the most similar faces based on the face specified.

The face processing functionalities we offer were based on an external library from Company Omron. In this project, even though we did not implement a functionality that supported image content search based on the content of the whole image, we implemented a structure for the image content search. So based on the structure, in the extended project, we can get another external library which supports image content search so that the functionality can be integrated into the existing image type. The implementation for it should be quite similar to the existing human face recognition functionality implementation in Mimer SQL.

5.2.3 Other multimedia functionalities

Overall, these two component areas of functionalities for images have a lot of room to improve. But the project was not just about images but also about the multimedia area. While not just focused on images, this project was also a prototype for other multimedia functionalities in Mimer Information Technology. For example, based on the structure of image type, we can also build voice type so that Mimer can support voice functionalities such as voice recognition. Moreover, we should be able to support video manipulations for different formats as well. There could be also a type named “video”. And we can support many different video manipulations for different video formats. So our final concluding point is that this thesis serves as an excellent prototype that will enable us to build a platform for a number of future improvements and developments.

6. References

[1] Informix Corporation: Excalibur Image DataBlade Module, Version 1.2 – User's Guide, March 1999.

[2] Informix Corporation: Informix Image Foundation DataBlade Module, Version 2.00 – User's Guide, December 2000.

[3] Informix Corporation: Informix Guide to SQL: Tutorial, Version 9.2, December 1999.

[4] Knut Stolze, Still Image Extensions in Database Systems – A Product Overview, 2002

[5] Knut Stolze, A DB2 UDB still image extender Integrate ImageMagick with DB2 UDB, 2005

[6] Oracle Corporation: Oracle Multimedia documentation for Oracle Database 11*g* – User's Guide, 2009

[7] IBM Corporation: DB2 Universal Database Image, Audio, and Video Extenders - Administration and Programming, Version 9.5, 2009.

[8] Microsoft Corporation: Microsoft SQL Server 2008 books online, January 2009

[9] Mimer SQL Engine Document Set, Version 9.3, 2007

[10] OMRON CORPORATION image processing documentary

[11] Ming-Hsuan Yang, David J. Kriegman, and Narendra Ahuja, "Detecting Faces in Images: A Survey", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.24, no.1, January 2002.

[12] M. Turk and A. Pentland, "Eigen Faces for Recognition", Journal of Cognitive. Neuroscience, 3(1), 1991.

[13] H. A. Rowley, S. Baluja, and T. Kanade, "Neural Network Based Face Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 20, January, pp. 23-38, 1998.

[14] Mohamed A. Berbar, Hamdy M. Kelash, and Amany A. Kandeel," Face and Facial Features Detection in Color Images", Proceedings of the Geometric Modeling and Imaging, pp. 209 - 214, July 05-06, 2006.

[15] Y.H. Chan and S.A.R. Abu-Bakar, "Face Detection System Based on Feature-Based Chrominance Color Information", Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (CGIV'04), pp. 153-158, 2004

[16] K. Sobottka and I. Pitas, "Face Localization and Feature Extraction Based on Shape and Color Information," Proc. IEEE Int'l Conf. Image Processing, pp. 483-486, 1996.

[17] Chengjun Liu, "Gabor-based kernel PCA with fractional power polynomial models for face recognition,", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 572-581, 2004.

[18] Deng Cai, Xiaofei He, Jiawei Han and Hong-Jiang Zhang, "Orthogonal Laplacianfaces for Face Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 3608 – 3614, 2006.

[19] C. Liu and H. Wechsler, “Evolutionary pursuit and its application to face recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 6, pp. 570–582, 2000.

[20] B. Wu, H. Ai, C. Huang, and S. Lao. Fast rotation invariant multi-view face detection based on real adaboost. In proceeding of the international conference on automatic face and gesture recognition, pages 79–84, 2004.

[21] S. Z. Li, L. Zhu, Z. Q. Zhang, et al., "Statistical Learning of Multi-View Face Detection". In Proceedings of the 7th European Conference on Computer Vision. Copenhagen, Denmark. May, 2002.

[22] Yoshihisa Ijiri, Miharu Sakuragi and Shihong Lao, "Security Management for Mobile Devices by Face Recognition ". Mobile Data Management, 2006, 7th International Conference.