

**Code Listings**



**Student Name: Wen Liu**

**ID Number: C00105088**

**Project: Fingerprint Recognition**

**Supervisor: Nigel Whyte (M.Sc. MIEEE)**

**Date: 10/04/2010**

**Contents**

1. **Introduction ……………………………………………………………... 2**
2. **Code Listings ……………………………………………………………... 3**

***MainForm.cs ……………………………………………………………… 3***

***Enhance.cs …………………………………………………………..…. 31***

***Thinning.cs ……………………………………………........................... 33***

***Minutiae.cs ………………………………………….............................. 40***

***CorePoint.cs ……………………………………….................................. 51***

***Match.cs ……………………………………………............……….. 59***

***Recognition.cs ………………………………………………............…….. 67***

1. ***Introduction***

This document contains all essential functional modules of the fingerprint recognition project. The following modules are labeled by C# source file and well commented:

* MainForm.cs
* Enhance.cs
* Thinning.cs
* Minutiae.cs
* CorePoint.cs
* Match.cs
* Recognition.cs

Those code files which generated by IDE (Visual Studio 2008) for GUI construction are not listed in this document. Please see the installation disk of the project for details of those code files, under the source code folder named “src”.

1. ***Code Listings***

***MainForm.cs***

using System;

using System.Globalization;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

using System.IO;

using Microsoft.VisualBasic;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The MainForm class is the entry point of entire project. It contains all

 //functions handle actions of GUI components

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public partial class MainForm : Form

 {

 public static Bitmap fp;

 public static Bitmap originalFP;

 public static String FPN;

 public static int coreX;

 public static int coreY;

 public static int index;

 public static int[,] Orientation;

 public static DataTable table;

 public static String fingerprintDirectory = Environment.CurrentDirectory + "\\Fingerprints";

 public static String databaseDirectory = Environment.CurrentDirectory + "\\EnrolledFingerprints";

 public static String saveFileDirectory = Environment.CurrentDirectory + "\\Match results";

 public static String helpFile = Environment.CurrentDirectory + "\\Document.chm";

 public int[] Minutiae;

 public int[,] matchMatrix1;

 public int[,] matchMatrix2;

 public DataTable Mintable;

 // class constructor is used to initialize all GUI components

 public MainForm()

 {

 InitializeComponent();

 enhanButt.Enabled = false;

 thinButt.Enabled = false;

 minuButt.Enabled = false;

 enrollbutt.Enabled = false;

 cancelButt.Enabled = false;

 SaveCommButt.Enabled = false;

 delButt.Enabled = false;

 loadImage1Butt.Enabled = false;

 loadImage2Butt.Enabled = false;

 reloadButt1.Enabled = false;

 reloadButt2.Enabled = false;

 lblCom1.Enabled = false;

 lblCom2.Enabled = false;

 lblFingerID1.Enabled = false;

 lblFingerID2.Enabled = false;

 lblOrigImage1.Enabled = false;

 lblOrigImage2.Enabled = false;

 lblFID1.Enabled = false;

 lblFID2.Enabled = false;

 txtOI1.Enabled = false;

 txtOI2.Enabled = false;

 txtMComments1.Enabled = false;

 txtMComments2.Enabled = false;

 saveButt.Enabled = false;

 ListUpdate();

 }

 //ListUpdate function is used to update contents of the combo box in Database Management panel.

 public void ListUpdate()

 {

 //set up a connection of fingerprint database

 OleDbConnection conn = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 String sql = "Select \* from Fingerprint";

 OleDbDataAdapter coda = new OleDbDataAdapter();

 coda.SelectCommand = new OleDbCommand(sql, conn);

 DataSet set = new DataSet();

 table = new DataTable();

 try

 {

 //set up a dataTable contains IDs of enrolled fingerprints

 coda.Fill(set, "Fingerprint");

 GC.Collect();

 table = set.Tables["Fingerprint"];

 }

 catch (Exception ee)

 {

 }

 finally

 {

 //close database connection

 conn.Close();

 }

 //update combo box

 comboBoxFP.Items.Clear();

 for (int i = 0; i < table.Rows.Count; i++)

 {

 comboBoxFP.Items.Add(table.Rows[i]["Fingerprint\_ID"]);

 }

 comboBoxFP.Update();

 }

 //this function handles "Browse" button click action

 private void browseButt\_Click(object sender, EventArgs e)

 {

 //open a file dialog to allow user to select a finger to process

 OpenFileDialog openFileDialog = new OpenFileDialog();

 DateTime currentTime = new DateTime();

 currentTime = DateTime.Now;

 //only take bmp file format.

 openFileDialog.Filter = "Bitmap Files(\*.bmp)|\*.bmp";

 openFileDialog.FilterIndex = 0;

 openFileDialog.InitialDirectory = fingerprintDirectory;

 openFileDialog.RestoreDirectory = false;

 //if OK clicked

 if (DialogResult.OK == openFileDialog.ShowDialog())

 {

 Bitmap openedImage = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 //check if image is not with right specification -- size 256\*256, bit depth 24bits

 if ((openedImage.Width == 256) && (openedImage.Height == 256))

 {

 fp = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 originalFP = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 origiImageBox.Image = fp;

 enrollImageBox.Image = null;

 inputImageTextBox.Text = openFileDialog.FileName;

 // get the file name of opened image (not the file path)

 FPN = openFileDialog.FileName.Substring(openFileDialog.FileName.LastIndexOf('\\') + 1, openFileDialog.FileName.LastIndexOf('.') - openFileDialog.FileName.LastIndexOf('\\') - 1);

 enhanButt.Enabled = true;

 thinButt.Enabled = false;

 minuButt.Enabled = false;

 enrollbutt.Enabled = false;

 cancelButt.Enabled = true;

 lblId.Text = "";

 lblNoOfMin.Text = "";

 //set current date and time as default comments of the image

 txtComments.Text = currentTime.ToLongDateString();

 }

 else

 {

 // show message box if the opened image is not with right specification

MessageBox.Show("Please make sure the image size is 256 X 256 pixel and bit depth is 24 bits.", "Input image error!", MessageBoxButtons.OK MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1);

 }

 }

 }

 //this function handles "About" menu item click action

 private void aboutToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 //create and show about box

 AboutBox aboutbox = new AboutBox();

 aboutbox.Show();

 }

 //this function handles "Enhance" button click action

 private void enhanButt\_Click(object sender, EventArgs e)

 {

 //create a instance of Enhance class

 Enhance bmp = new Enhance();

 //call enhance function and set image of enrollImageBox as the enhanced image

 enrollImageBox.Image = bmp.enhance(fp);

 fp = (Bitmap)enrollImageBox.Image;

 thinButt.Enabled = true;

 enhanButt.Enabled = false;

 }

 //this function handles "Thin" button click action

 private void thinButt\_Click(object sender, EventArgs e)

 {

 //create a instance of Thinning class

 Thinning bmp = new Thinning();

 //call ThiningPic function and set image of enrollImageBox as the thinned image

 enrollImageBox.Image = bmp.ThiningPic(fp);

 fp = (Bitmap)enrollImageBox.Image ;

 minuButt.Enabled = true;

 thinButt.Enabled = false;

 }

 //this function handles "Extract Minutiae" button click action

 private void minuButt\_Click(object sender, EventArgs e)

 {

 //create a instance of Minutiae class

 Minutiae bmp = new Minutiae();

 DateTime currentTime = new DateTime();

 currentTime = DateTime.Now;

 // create a instance of CorePoint class

 CorePoint cp = new CorePoint();

 //run normalization process on the original opened fingerprint image

 cp.Normalization(originalFP);

 //run Sobel smooth process on the normalized image

 cp.Sobel();

 //detect the orientation field of the fingerprint

 cp.LocalOrientation();

 //find core point are of the fingerprint

 cp.Point();

 coreX = cp.coX;

 coreY = cp.coY;

 Orientation = cp.Oma;

 //call functions in Minutiae.cs to extract minutiae

 enrollImageBox.Image = bmp.MinuExtract(fp);

 fp = (Bitmap)enrollImageBox.Image;

 minuButt.Enabled = false;

 enhanButt.Enabled = false;

 thinButt.Enabled = false;

 enrollbutt.Enabled = true;

 lblId.Text = FPN+"\_" +currentTime.Day.ToString() + currentTime.Month.ToString()+currentTime.Year.ToString();

 lblNoOfMin.Text = index.ToString();

 //set up Minutiae array to hold minutiae details of the fingerprint

 Minutiae = bmp.tMinu;

 //show coin point are on the image

 Pen p = new Pen(Color.Yellow, 2);

 Graphics g = Graphics.FromImage(enrollImageBox.Image);

 g.DrawEllipse(p, coreX - 5, coreY - 5, 10, 10);

 }

 //this function handles "Cancel" button click action

 private void cancelButt\_Click(object sender, EventArgs e)

 {

 //ask user for confirmation.

 DialogResult res = MessageBox.Show("Are you sure you want to cancel all works were done ?", "Cancel", MessageBoxButtons.YesNo,MessageBoxIcon.Question,

MessageBoxDefaultButton.Button2);

 // if yes

 if (res == DialogResult.Yes)

 {

 enrollImageBox.Image = null;

 inputImageTextBox.Text = "";

 lblId.Text = "";

 lblNoOfMin.Text = "";

 origiImageBox.Image = null;

 minuButt.Enabled = false;

 enhanButt.Enabled = false;

 thinButt.Enabled = false;

 cancelButt.Enabled = false;

 enrollbutt.Enabled = false;

 }

 }

 //this function handles "Enroll" button click action

 private void enrollbutt\_Click(object sender, EventArgs e)

 {

 //save processed fingerprint image

 Directory.CreateDirectory(databaseDirectory);

 FileAttributes myAttributtes = File.GetAttributes(databaseDirectory);

 File.SetAttributes(databaseDirectory, myAttributtes | FileAttributes.Hidden);

 enrollImageBox.Image.Save(databaseDirectory +"\\"+ lblId.Text + ".bmp",ImageFormat.Bmp);

 //save data into database

 //set up a database connection

 OleDbConnection connection = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 //insert sql query for fingerprint table

 String sql = "INSERT INTO Fingerprint ( Fingerprint\_ID, FP\_image, No\_of\_Min, Comments, Core\_X, Core\_Y )" +"VALUES(?, ?, ?, ?, ?, ?)";

 //set up DbCommand

 OleDbCommand insert = new OleDbCommand(sql, connection);

 insert.Connection = connection;

 connection.Open();

 insert.CommandType = CommandType.Text;

 insert.Parameters.AddWithValue("Fingerprint\_ID", this.lblId.Text);

 insert.Parameters.AddWithValue("FP\_image", this.inputImageTextBox.Text);

 insert.Parameters.AddWithValue("No\_of\_Min",int.Parse(this.lblNoOfMin.Text));

 insert.Parameters.AddWithValue("Comments", this.txtComments.Text);

 insert.Parameters.AddWithValue("Core\_X",coreX);

 insert.Parameters.AddWithValue("Core\_Y",coreY);

 try

 {

 insert.ExecuteNonQuery();

 //select minutiae which the distance to core less than 60 pixel and sort the array

 int k = 0;

 int[] minEnroll = new int[1000];

 for (int i = 4; i < index\*5 ; i += 5)

 {

 if (Minutiae[i - 1] <= 60) // distance less than 60

 {

 int x = Minutiae[i - 4];

 minEnroll[k] =Minutiae[i - 4];

 k++;

 int y = Minutiae[i - 3];

 minEnroll[k] =Minutiae[i - 3];

 k++;

 minEnroll[k] =Minutiae[i - 2];

 k++;

 minEnroll[k] =Minutiae[i - 1];

 k++;

 minEnroll[k] =Minutiae[i];

 k++;

 }

 }

 // sort the minEnroll array

 for (int i = 4; i < k; i += 5)

 {

 for (int j = i; j < k; j += 5)

 {

 if (minEnroll[i - 1] > minEnroll[j - 1])

 {

 int tmp0 = minEnroll[i - 4];

 int tmp1 = minEnroll[i - 3];

 int tmp2 = minEnroll[i - 2];

 int tmp3 = minEnroll[i - 1];

 int tmp4 = minEnroll[i];

 minEnroll[i - 4] = minEnroll[j - 4];

 minEnroll[i - 3] = minEnroll[j - 3];

 minEnroll[i - 2] = minEnroll[j - 2];

 minEnroll[i - 1] = minEnroll[j - 1];

 minEnroll[i] = minEnroll[j];

 minEnroll[j - 4] = tmp0;

 minEnroll[j - 3] = tmp1;

 minEnroll[j - 2] = tmp2;

 minEnroll[j - 1] = tmp3;

 minEnroll[j] = tmp4;

 }

 }

 }

 //the insert sql query for Minutiae table

 OleDbCommand insertM = new OleDbCommand();

 insertM.Connection = connection;

 insertM.Transaction = connection.BeginTransaction();

 // set SQL string array

 String[] minSql = new String [k/5];

 int v = 0;

 for (int i = 4; i < k; i += 5)

 {

 minSql[v] = "INSERT INTO Minutiae ( Fingerprint\_ID, X\_coor, Y\_coor, Type, Distance, Angle )" + "VALUES('"+lblId.Text+"'," + minEnroll[i - 4] + " ," + minEnroll[i - 3] + " ," + minEnroll[i - 2] + " ," + minEnroll[i - 1] + ", "+minEnroll[i]+")";

 v++;

 }

 for (int n = 0; n < minSql.Length; n++)

 {

 String strsql = minSql[n].ToString();

 if (strsql.Trim().Length > 1)

 {

 insertM.CommandText = strsql;

 insertM.ExecuteNonQuery();

 }

 }

 insertM.Transaction.Commit();

 connection.Close();

 // show message box to tell success

 MessageBox.Show("The fingerprint has been saved into database with success.", "Success",

 MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);

 }

 catch (OleDbException ae)

 {

 // show error message if the fingerprint is already exists

 MessageBox.Show("Database error! The fingerprint record is already exists in the database", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 //update combo box

 ListUpdate();

 }

 //this function handles "Load Image 1" button click action

 private void loadImage1Butt\_Click(object sender, EventArgs e)

 {

 if (Directory.Exists(databaseDirectory))

 {

 // open a file dialog to allow user to select a image

 OpenFileDialog openFileDialog = new OpenFileDialog();

 openFileDialog.Filter = "Bitmap Files(\*.bmp)|\*.bmp";

 openFileDialog.FilterIndex = 0;

 openFileDialog.InitialDirectory = databaseDirectory;

 openFileDialog.RestoreDirectory = false;

 if (DialogResult.OK == openFileDialog.ShowDialog())

 {

 // get file name of opened image

 String filename = openFileDialog.FileName.Substring

(openFileDialog.FileName.LastIndexOf('\\') + 1,

 openFileDialog.FileName.LastIndexOf('.') - openFileDialog.FileName.LastIndexOf('\\') - 1);

 Bitmap image = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 String i = openFileDialog.FileName;

 //check if the opened image is get from image database

 if ((i.Substring(0, i.LastIndexOf('\\'))).Equals(databaseDirectory))

 {

 //set image of matchImageBox1 as the opened image

 matchImageBox1.Image = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 //call MinutiaeArray function to get minutiae records of this image

 matchMatrix1 = MinutiaeArray(filename);

 OleDbConnection conn1 = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 String sql1 = "Select \* from Fingerprint where Fingerprint\_ID='" + filename + "'";

 OleDbCommand select = new OleDbCommand(sql1, conn1);

 select.Connection = conn1;

 select.CommandType = CommandType.Text;

 conn1.Open();

 OleDbDataReader reader;

 reader = select.ExecuteReader();

 if (reader.Read())

 {

 //show details of opened image

 lblFID1.Text = reader["Fingerprint\_ID"].ToString();

 txtOI1.Text = reader["FP\_image"].ToString();

 int n = (int)reader["No\_of\_Min"];

 txtMComments1.Text = reader["Comments"].ToString();

 int x = (int)reader["Core\_X"];

 int y = (int)reader["Core\_Y"];

 }

 reader.Close();

 conn1.Close();

 // if the other enrolled fingerprint is selected

 if (matchImageBox2.Image != null)

 {

 //creat a instance of Match class

 Match m = new Match();

 // matching start

 AfterMatch(m.Matching(matchMatrix1, matchMatrix2));

 }

 }

 else

 {

 MessageBox.Show("The image MUST be a fingerprint image in the database!", "Error",

 MessageBoxButtons.OK, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button1);

 }

 }

 }

 else {

 MessageBox.Show("Please make sure you enrolled at least one fingerprint into the database!", "Error",MessageBoxButtons.OK, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button1);

 }

 }

 // this function handle "Load Image 2" Button click action, comments are same with loadImage1Butt\_Click

 private void loadImage2Butt\_Click(object sender, EventArgs e)

 {

 if (Directory.Exists(databaseDirectory))

 {

 OpenFileDialog openFileDialog = new OpenFileDialog();

 openFileDialog.Filter = "Bitmap Files(\*.bmp)|\*.bmp";

 openFileDialog.FilterIndex = 0;

 openFileDialog.InitialDirectory = databaseDirectory;

 openFileDialog.RestoreDirectory = false;

 if (DialogResult.OK == openFileDialog.ShowDialog())

 {

 String filename = openFileDialog.FileName.Substring

(openFileDialog.FileName.LastIndexOf('\\') + 1, openFileDialog.FileName.LastIndexOf('.') - openFileDialog.FileName.LastIndexOf('\\')-1);

 String i = openFileDialog.FileName;

 if ((i.Substring(0, i.LastIndexOf('\\'))).Equals(databaseDirectory))

 {

 matchImageBox2.Image = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 matchMatrix2 = MinutiaeArray(filename);

 OleDbConnection conn2 = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 String sql2 = "Select \* from Fingerprint where Fingerprint\_ID='" + filename + "'";

 OleDbCommand select = new OleDbCommand(sql2, conn2);

 select.Connection = conn2;

 select.CommandType = CommandType.Text;

 conn2.Open();

 OleDbDataReader reader;

 reader = select.ExecuteReader();

 if (reader.Read())

 {

 lblFID2.Text = reader["Fingerprint\_ID"].ToString();

 txtOI2.Text = reader["FP\_image"].ToString();

 int n = (int)reader["No\_of\_Min"];

 txtMComments2.Text = reader["Comments"].ToString();

 int x = (int)reader["Core\_X"];

 int y = (int)reader["Core\_Y"];

 }

 reader.Close();

 conn2.Close();

 // if the other enrolled fingerprint is selected

 if (matchImageBox1.Image != null)

 {

 Match m = new Match();

 // matching start

 AfterMatch(m.Matching(matchMatrix1, matchMatrix2));

 }

 }

 else

 {

 MessageBox.Show("The image MUST be a fingerprint image in the database!", "Error",

 MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 }

 }

 else {

 MessageBox.Show("Please make sure you enrolled at least one fingerprint into the database!", "Error",MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 }

 /\* this function handles actions of user change the selected item of the comb box in "Database Management" panel\*/

 private void comboBoxFP\_SelectedIndexChanged(object sender, EventArgs e)

 {

 String s = databaseDirectory +"\\"+ comboBoxFP.SelectedItem.ToString() + ".bmp";

 // show image of selected record in the image box

 Bitmap image = (Bitmap) Bitmap.FromFile(s,false);

 previewImageBox.Image = image;

 //show details of the record

 lblNOM.Text = (table.Rows[comboBoxFP.SelectedIndex]["No\_of\_Min"]).ToString();

 txtComm.Text = (table.Rows[comboBoxFP.SelectedIndex]["Comments"]).ToString();

 txtOI.Text = (table.Rows[comboBoxFP.SelectedIndex]["FP\_image"]).ToString();

 if (previewImageBox.Image != null)

 {

 //enable buttons if record found

 SaveCommButt.Enabled = true;

 delButt.Enabled = true;

 }

 else

 {

 SaveCommButt.Enabled = false;

 delButt.Enabled = false;

 }

 }

 //this function handles "Save Comments" button click action

 private void SaveCommButt\_Click(object sender, EventArgs e)

 {

 //set up connection with database

 OleDbConnection connec = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 //find and update the record

 String sql = "UPDATE Fingerprint SET Comments = '" + txtComm.Text + "'

 WHERE Fingerprint\_ID = '"+ comboBoxFP.SelectedItem.ToString()+"'";

 connec.Open();

 OleDbCommand command = new OleDbCommand(sql, connec);

 command.ExecuteNonQuery();

 connec.Close();

 // update combo box items

 ListUpdate();

 //show message to tell success

 MessageBox.Show("Comments updated with success.", "Success",MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);

 comboBoxFP.Text = "<Select a record from the database>";

 comboBoxFP.Update();

 txtComm.Text = "";

 lblNOM.Text = "";

 txtOI.Text = "";

 previewImageBox.Image.Dispose();

 previewImageBox.Image = null;

 SaveCommButt.Enabled = false;

 delButt.Enabled = false;

 }

 // this function handles "Delete" button click action

 private void delButt\_Click(object sender, EventArgs e)

 {

 // show message box for a confirmation

 DialogResult res = MessageBox.Show("Are you sure you want to delete this record from databases ?", "Delete",MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);

 if (res == DialogResult.Yes)

 {

 FileInfo image = new FileInfo(databaseDirectory + "\\"

+ comboBoxFP.SelectedItem.ToString() + ".bmp");

 // if record exists

 if (image.Exists)

 {

 try

 {

 // reset image box before delete the opened image

 previewImageBox.Image.Dispose();

 previewImageBox.Image = null;

 image.Delete();

 /\*set up connnection with database to delete records from Fingerprint table and Minutiae table\*/

 OleDbConnection connec = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 try

 {

 String sql = "DELETE FROM Fingerprint WHERE Fingerprint\_ID = '" + comboBoxFP.SelectedItem.ToString() + "'";

 OleDbCommand cmd = new OleDbCommand(sql, connec);

 connec.Open();

 cmd.ExecuteNonQuery();

 String sql2 = "DELETE FROM Minutiae WHERE Fingerprint\_ID = '" + comboBoxFP.SelectedItem.ToString() + "'";

 OleDbCommand cmd2 = new OleDbCommand(sql2, connec);

 cmd2.ExecuteNonQuery();

 }

 catch (Exception)

 {

 }

 finally

 {

 connec.Close();

 }

 //update combo box items

 ListUpdate();

 comboBoxFP.Text = "<Select a record from the database>";

 comboBoxFP.Update();

 txtComm.Text = "";

 lblNOM.Text = "";

 txtOI.Text = "";

 SaveCommButt.Enabled = false;

 delButt.Enabled = false;

 }

 // show error message if the record is currently using by other thread.

 catch(Exception)

 {

 previewImageBox.Image = (Bitmap)Bitmap.FromFile(image.ToString(), false) ;

 MessageBox.Show("Delete failed! The image is currently using by other thread.", "Delete Error", MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 }

 }

 }

 //this function handles "Match" button click action

 private void matchButt\_Click(object sender, EventArgs e)

 {

 // reset the GUI components, make them ready for matching

 lblFID1.Text = "";

 lblFID2.Text = "";

 txtOI1.Text = "";

 txtOI2.Text = "";

 txtMComments1.Text = "";

 txtMComments2.Text = "";

 lblMatchedMPairs.Text = "";

 lblResultComments.Text = "";

 lblNoOfTMinu.Text = "";

 lblSimilarity.Text = "";

 saveButt.Enabled = false;

 loadImage1Butt.Enabled = true;

 loadImage2Butt.Enabled = true;

 reloadButt1.Enabled = true;

 reloadButt2.Enabled = true;

 lblCom1.Enabled = true;

 lblCom2.Enabled = true;

 lblFingerID1.Enabled = true;

 lblFingerID2.Enabled = true;

 lblOrigImage1.Enabled = true;

 lblOrigImage2.Enabled = true;

 lblFID1.Enabled = true;

 lblFID2.Enabled = true;

 txtOI1.Enabled = true;

 txtOI2.Enabled = true;

 txtMComments1.Enabled = true;

 txtMComments2.Enabled = true;

 matchImageBox1.Image = null;

 matchImageBox2.Image = null;

 }

 // this function handles "Recognition" button click action

 private void recogButt\_Click(object sender, EventArgs e)

 {

 //initialization

 String[] result = { "", "", "", "" };

 lblFID1.Text = "";

 lblFID2.Text = "";

 txtOI1.Text = "";

 txtOI2.Text = "";

 txtMComments1.Text = "";

 txtMComments2.Text = "";

 lblMatchedMPairs.Text = "";

 lblResultComments.Text = "";

 lblNoOfTMinu.Text = "";

 lblSimilarity.Text = "";

 saveButt.Enabled = false;

 loadImage1Butt.Enabled = false;

 loadImage2Butt.Enabled = false;

 reloadButt1.Enabled = false;

 reloadButt2.Enabled = false;

 lblCom1.Enabled = false;

 lblCom2.Enabled = false;

 lblFingerID1.Enabled = false;

 lblFingerID2.Enabled = false;

 lblOrigImage1.Enabled = false;

 lblOrigImage2.Enabled = false;

 lblFID1.Enabled = false;

 lblFID2.Enabled = false;

 txtOI1.Enabled = false;

 txtOI2.Enabled = false;

 txtMComments1.Enabled = false;

 txtMComments2.Enabled = false;

 matchImageBox1.Image = null;

 matchImageBox2.Image = null;

 //check if database is not empty

 if (comboBoxFP.Items.Count != 0)

 {

 // open a file dialog to allow user to select an unenrolled fingerprint image

 OpenFileDialog openFileDialog = new OpenFileDialog();

 openFileDialog.Filter = "Bitmap Files(\*.bmp)|\*.bmp";

 openFileDialog.FilterIndex = 0;

 openFileDialog.InitialDirectory = fingerprintDirectory;

 openFileDialog.RestoreDirectory = false;

 if (DialogResult.OK == openFileDialog.ShowDialog())

 {

 Bitmap openedImage = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 // check if the opened image fits the specification

 if ((openedImage.Width == 256) || (openedImage.Height == 256))

 {

 //show the image in the image box

 matchImageBox1.Image = (Bitmap)Bitmap.FromFile(openFileDialog.FileName, false);

 txtOI1.Enabled = true;

 txtOI1.Text = openFileDialog.FileName;

 //ask for a confirmation to start recognition

 DialogResult res = MessageBox.Show("Start recognition of this fingerprint ?", "Recognition start",MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);

 if (res == DialogResult.Yes)

 {

 //create an instance of Recognition class

 Recognition ver = new Recognition();

 /\* call Preprocessing function in Recognition class to run image pre-processing on the image and show the output\*/

 matchImageBox1.Image = ver.Preprocessing(openedImage);

 //indicate core point area on the image

 Pen p = new Pen(Color.Yellow, 2);

 Graphics g = Graphics.FromImage(matchImageBox1.Image);

 g.DrawEllipse(p, coreX - 5, coreY - 5, 10, 10);

 //call StartSearch function to start recognition

 result = ver.StartSearch();

 //show the image found

 matchImageBox2.Image = (Bitmap)Bitmap.FromFile(databaseDirectory +

"\\" + result[0] + ".bmp", false);

 //set up a database connection to get detailed information of matched fingerprint

 OleDbConnection conn2 = new OleDbConnection(

"Provider=Microsoft.ACE.OLEDB.12.0;"

 +"Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 String sql2 = "Select \* from Fingerprint where Fingerprint\_ID='"

+ result[0] + "'";

 OleDbCommand select = new OleDbCommand(sql2, conn2);

 select.Connection = conn2;

 select.CommandType = CommandType.Text;

 conn2.Open();

 OleDbDataReader reader;

 reader = select.ExecuteReader();

 if (reader.Read())

 {

 // output detailed information of matched fingerprint

 lblFID2.Enabled = true;

 lblFingerID2.Enabled = true;

 txtOI2.Enabled = true;

 lblOrigImage2.Enabled = true;

 txtMComments2.Enabled = true;

 lblCom2.Enabled = true;

 lblFID2.Text = reader["Fingerprint\_ID"].ToString();

 txtOI2.Text = reader["FP\_image"].ToString();

 txtMComments2.Text = reader["Comments"].ToString();

 }

 reader.Close();

 conn2.Close();

 String[] ar = { result[1], result[2], result[3] };

 AfterMatch(ar);

 // show message to tell user the recognition process is complete

 if (Convert.ToDouble(ar[2]) > 80.0)

 {

 MessageBox.Show("Recognition complete! Matched record found. \r\n\r\nSee result panel for details. ", "Matched record found",

 MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);

 }

 else

 {

 MessageBox.Show("Recognition complete! No certain Matched record found.\r\n\r\nRecord with highest similarity been displayed instead",

 "No certain matched record found",

MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);

 }

 }

 }

 else

 {

 // show message box if the opened image is not with right specification

 MessageBox.Show("Please make sure the image size is 256 X 256 pixel and bit depth is 24 bits.", "Input image error!", MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 }

 }

 else

 {

 //show message box if database is empty

 MessageBox.Show("Fingerprint database is empty! Please enroll at least one fingerprint.",

"Error",MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

 }

 }

/\* this function is used to search into database to find minutiae details of a specific fingerprint and fill them into an array \*/

 public int[,] MinutiaeArray(String filename)

 {

 // set up connection with database and search for minutiae

 OleDbConnection conn = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 String sql = "SELECT \* FROM Minutiae WHERE Fingerprint\_ID = '" + filename + "'";

 OleDbDataAdapter coda = new OleDbDataAdapter(sql, conn);

 // write all minutiae into a data set

 DataSet fillds = new DataSet();

 coda.Fill(fillds, "list");

 GC.Collect();

 Mintable = fillds.Tables["list"];

 int[,] array = new int[Mintable.Rows.Count, 3];

 //fill in the array

 for (int i = 0; i < Mintable.Rows.Count; i++)

 {

 array[i, 0] = Convert.ToInt16(Mintable.Rows[i]["Distance"]);

 array[i, 1] = Convert.ToInt16(Mintable.Rows[i]["Angle"]);

 array[i, 2] = Convert.ToInt16(Mintable.Rows[i]["Type"]);

 }

 //sort the array

 for (int si = 0; si < array.GetLength(0); si++)

 {

 for (int ji = si; ji < array.GetLength(0); ji++)

 {

 if (array[si, 0] > array[ji, 0])

 {

 int tmp0 = array[si, 0];

 int tmp1 = array[si, 1];

 int tmp2 = array[si, 2];

 array[si, 0] = array[ji, 0];

 array[si, 1] = array[ji, 1];

 array[si, 2] = array[ji, 2];

 array[ji, 0] = tmp0;

 array[ji, 1] = tmp1;

 array[ji, 2] = tmp2;

 }

 }

 }

 //close connection

 conn.Close();

 // return minutiae array

 return array;

 }

 // this function is used to show matching result after fingerprint matching and recognition

 public void AfterMatch(String[] r)

 {

 //set GUI component with corresponding cell in parameter string array

 lblNoOfTMinu.Text = r[0];

 lblMatchedMPairs.Text = r[1];

 lblSimilarity.Text = r[2] + " %";

 //set comments label

 if (Convert.ToDouble(r[2]) == 0)

 {

 lblResultComments.Text = "No Similarity, absolutely from different fingers.";

 }

 else if ((Convert.ToDouble(r[2]) > 0) && (Convert.ToDouble(r[2]) <= 50))

 {

 lblResultComments.Text = "Low Similarity, can be considered as from different fingers.";

 }

 else if ((Convert.ToDouble(r[2]) > 50) && (Convert.ToDouble(r[2]) <= 70))

 {

 lblResultComments.Text = "Moderate Similarity, possibly from same finger.";

 }

 else if ((Convert.ToDouble(r[2]) > 70) && (Convert.ToDouble(r[2]) <= 80))

 {

 lblResultComments.Text = "High Similarity, probably from same finger.";

 }

 else if ((Convert.ToDouble(r[2]) > 80) && (Convert.ToDouble(r[2]) < 100))

 {

 lblResultComments.Text = "Very High Similarity, can be considered as from same finger.";

 }

 else if (Convert.ToDouble(r[2]) == 100)

 {

 lblResultComments.Text = "Perfect match, exactly from same finger";

 }

 loadImage1Butt.Enabled = false;

 loadImage2Butt.Enabled = false;

 reloadButt1.Enabled = false;

 reloadButt2.Enabled = false;

 saveButt.Enabled = true;

 }

 //this function handles "Reload Image"(on the left) button click action

 private void reloadButt1\_Click(object sender, EventArgs e)

 {

 matchImageBox1.Image = null;

 lblFID1.Text = "";

 lblFID2.Text = "";

 txtOI1.Text = "";

 txtOI2.Text = "";

 txtMComments1.Text = "";

 txtMComments2.Text = "";

 }

 //this function handles "Reload Image"(on the right) button click action

 private void reloadButt2\_Click(object sender, EventArgs e)

 {

 matchImageBox2.Image = null;

 lblFID1.Text = "";

 lblFID2.Text = "";

 txtOI1.Text = "";

 txtOI2.Text = "";

 txtMComments1.Text = "";

 txtMComments2.Text = "";

 }

 //this function handles "Save Result" button click action

 private void saveButt\_Click(object sender, EventArgs e)

 {

 // get system current time

 DateTime currentTime = new DateTime();

 currentTime = DateTime.Now;

 //generate a file name

 String sFileName = "\\MR\_" + currentTime.ToString

("dd\_MM\_yy\_HHmmss",DateTimeFormatInfo.InvariantInfo) + ".txt";

 //generate a file stream to write result into the file

 FileStream fs = new FileStream(saveFileDirectory + sFileName, FileMode.Create);

 StreamWriter wf = new StreamWriter(fs);

 //write result into the file

 wf.WriteLine("------------------------------------------------------------------------------------");

 wf.WriteLine("Date : " + currentTime.Date.ToLongDateString());

 wf.WriteLine("Time : " + currentTime.TimeOfDay.Hours.ToString() + ":" +

 currentTime.TimeOfDay.Minutes.ToString() + ":" +

 currentTime.TimeOfDay.Seconds.ToString());

 wf.WriteLine("------------------------------------------------------------------------------------");

 wf.WriteLine("");

 wf.WriteLine("");

 wf.WriteLine("Templete Fingerprint : "+ txtOI1.Text);

 wf.WriteLine("Fingerprint ID : " + lblFID1.Text);

 wf.WriteLine("");

 wf.WriteLine("Sample Fingerprint : " + txtOI2.Text);

 wf.WriteLine("Fingerprint ID : " + lblFID2.Text);

 wf.WriteLine("");

 wf.WriteLine("---------------------------------- Match Result ------------------------------------");

 wf.WriteLine("");

 wf.WriteLine("No of Templete Minutiae : " + lblNoOfTMinu.Text);

 wf.WriteLine("Matched Minutiae Pairs : " + lblMatchedMPairs.Text);

 wf.WriteLine("SIMILARITY : " + lblSimilarity.Text);

 wf.WriteLine("");

 wf.WriteLine("Comments : "+ lblResultComments.Text);

 //close file stream

 wf.Flush();

 wf.Close();

 fs.Close();

 //show compelte message

 MessageBox.Show("The match result saved ! \r\n\r\nFile name : " + sFileName.Substring(1,sFileName.Length-1), "Saved",MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);

 }

 //this function handles "Help Tips" menu item click action

 private void helpTipsToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 //show help document

 System.Diagnostics.Process.Start(helpFile);

 }

 //this function handles "Exit" menu item click action

 private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

 {

 //show message ask for a confirmation

 DialogResult r = MessageBox.Show("Exit FingerprintSpy Ver 1.01? ", "Saved",

 MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);

 if (r == DialogResult.Yes)

 {

 this.Dispose();

 }

 }

 }

}

***Enhance.cs***

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The Enhance class contains functions to enhance the opened image and distinguish

 //fingerprint ridges from vallis

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 class Enhance

 {

 //this function is used to enhance the input image and distinguish ridges from vallis

 public Bitmap enhance(Bitmap x)

 {

 //create a new bitmap

 Bitmap MyBitmap = new Bitmap(x);

 int myWidth, myHeight, i, j, iAvg, iPixel;

 Color myColor, myNewColor;

 RectangleF myRect;

 myWidth = MyBitmap.Width;

 myHeight = MyBitmap.Height;

 myRect = new RectangleF(0, 0, myWidth, myHeight);

 Bitmap bitmap = MyBitmap.Clone(myRect, System.Drawing.Imaging.PixelFormat.DontCare);

 i = 0;

 while (i < myHeight - 1)

 {

 j = 0;

 bool first = true;

 while (j < myWidth - 1)

 {

 // get the color of current pixel

 myColor = bitmap.GetPixel(j, i);

 //check if the red value of current pixel is greater than 120

 if (myColor.R >= 120 && first)

 {

 for (int z = j; z >= 0; z--)

 {

 bitmap.SetPixel(z, i, Color.Black);

 }

 first = false;

 }

 iAvg = myColor.R;

 iPixel = 0;

 if (iAvg >= 100)

 iPixel = 0;

 else

 iPixel = 255;

 myNewColor = Color.FromArgb(255, iPixel, iPixel, iPixel);

 //set smoothed colour to the pixel in the fingerprint image

 bitmap.SetPixel(j, i, myNewColor);

 j = j + 1;

 }

 i = i + 1;

 }

 //return enhanced image

 return bitmap;

 }

 }

}

***Thinning.cs***

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The Thinning class contains functions to thin the enhanced image and return thinned image

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public class Thinning

 {

 //this function is used to thin the image with Zhang-Suen Thinning algorithm

 public Bitmap ThiningPic(Bitmap b)

 {

 // create and show process box to show thinning start message

 ProcessBox pb = new ProcessBox();

 pb.Show();

 Bitmap thinBmp = new Bitmap(b);

 int lWidth = b.Width;

 int lHeight = b.Height;

 int p = 0;

 bool finish= false;

 int i, j, n, m; //indexes

 //four conditions

 bool bCondition1;

 bool bCondition2;

 bool bCondition3;

 bool bCondition4;

 int nCount; //counter

 int[,] neighbour = new int[3, 3]; //3×3 examine matrix

 int[] remove = new int[10000000]; //an array cotains pixels to eliminate.

 do

 {

 for (j = 1; j < lHeight - 1; j++)//row

 {

 for (i = 1; i < lWidth - 1; i++)//column

 {

 //initialize 4 conditions

 bCondition1 = false;

 bCondition2 = false;

 bCondition3 = false;

 bCondition4 = false;

 //get 3X3 parttern

 for (m = 0; m < 3; m++)

 {

 for (n = 0; n < 3; n++)

 {

 //if color of pixel is black,assign 0 to int v, otherwise assign 1.

 int v = (thinBmp.GetPixel(i + m - 1, j + n - 1).R == 255) ? 1 : 0;

 neighbour[m, n] = v;

 }

 }

 if (neighbour[1, 1] == 1)

 {

 //examine conditions

 // 2<=B(P1)<=6

 nCount = (neighbour[0, 0] + neighbour[1, 0] + neighbour[2, 0] +

 neighbour[0, 1] + neighbour[2, 1] +

 neighbour[0, 2] + neighbour[1, 2] + neighbour[2, 2]);

 if (nCount >= 2 && nCount <= 6)

 {

 bCondition1 = true;

 }

 // A01(P1)=1

 nCount = 0;

 if (neighbour[1, 0] == 0 && neighbour[2, 0] == 1)

 nCount++;

 if (neighbour[2, 0] == 0 && neighbour[2, 1] == 1)

 nCount++;

 if (neighbour[2, 1] == 0 && neighbour[2, 2] == 1)

 nCount++;

 if (neighbour[2, 2] == 0 && neighbour[1, 2] == 1)

 nCount++;

 if (neighbour[1, 2] == 0 && neighbour[0, 2] == 1)

 nCount++;

 if (neighbour[0, 2] == 0 && neighbour[0, 1] == 1)

 nCount++;

 if (neighbour[0, 1] == 0 && neighbour[0, 0] == 1)

 nCount++;

 if (neighbour[0, 0] == 0 && neighbour[1, 0] == 1)

 nCount++;

 if (nCount == 1)

 bCondition2 = true;

 //if P2\*P4\*P6=0

 if (neighbour[1, 0] \* neighbour[2, 1] \* neighbour[1, 2] == 0)

 {

 bCondition3 = true;

 }

 //ifP4\*P6\*P8=0

 if (neighbour[2, 1] \* neighbour[1, 2] \* neighbour[0, 1] == 0)

 {

 bCondition4 = true;

 }

 if (bCondition1 && bCondition2 && bCondition3 && bCondition4)

 {

 remove.SetValue(i,p);

 p++;

 remove.SetValue(j,p);

 p++;

 }

 }

 }

 }

 //change color of pixels in remove array into background color

 for (int c = 0; c < p-1; c=c+2)

 {

 int x = (int)remove.GetValue(c);

 int y = (int)remove.GetValue(c+1);

 thinBmp.SetPixel(x, y, Color.Black);

 }

 //if there are pixels to change,do loop 2

 if (p != 0)

 {

 p = 0;

 //3×3matrix

 for (j = 1; j < lHeight - 1; j++)//row

 {

 for (i = 1; i < lWidth - 1; i++)//column

 {

 bCondition1 = false;

 bCondition2 = false;

 bCondition3 = false;

 bCondition4 = false;

 //get 3X3 parttern

 for (m = 0; m < 3; m++)

 {

 for (n = 0; n < 3; n++)

 {

 int v = (thinBmp.GetPixel(i + m - 1, j + n - 1).R == 255) ? 1 : 0;

 neighbour[m, n] = v;

 }

 }

 if (neighbour[1, 1] == 1)

 {

 //test conditions

 // 2<=B(P1)<=6

 nCount = (neighbour[0, 0] + neighbour[1, 0] + neighbour[2, 0] +

 neighbour[0, 1] + neighbour[2, 1] +

 neighbour[0, 2] + neighbour[1, 2] + neighbour[2, 2]);

 if (nCount >= 2 && nCount <= 6)

 {

 bCondition1 = true;

 }

 // A01(P1)=1

 nCount = 0;

 if (neighbour[1, 0] == 0 && neighbour[2, 0] == 1)

 nCount++;

 if (neighbour[2, 0] == 0 && neighbour[2, 1] == 1)

 nCount++;

 if (neighbour[2, 1] == 0 && neighbour[2, 2] == 1)

 nCount++;

 if (neighbour[2, 2] == 0 && neighbour[1, 2] == 1)

 nCount++;

 if (neighbour[1, 2] == 0 && neighbour[0, 2] == 1)

 nCount++;

 if (neighbour[0, 2] == 0 && neighbour[0, 1] == 1)

 nCount++;

 if (neighbour[0, 1] == 0 && neighbour[0, 0] == 1)

 nCount++;

 if (neighbour[0, 0] == 0 && neighbour[1, 0] == 1)

 nCount++;

 if (nCount == 1)

 bCondition2 = true;

 //if P2\*P4\*P8=0

 if (neighbour[1, 0] \* neighbour[2, 1] \* neighbour[0, 1] == 0)

 {

 bCondition3 = true;

 }

 //ifP2\*P6\*P8=0

 if (neighbour[1, 0] \* neighbour[1, 2] \* neighbour[0, 1] == 0)

 {

 bCondition4 = true;

 }

 if (bCondition1 && bCondition2 && bCondition3 && bCondition4)

 {

 remove.SetValue(i, p);

 p++;

 remove.SetValue(j, p);

 p++;

 }

 }

 }

 }

 //change color of pixels in remove array into background color

 for (int c = 0; c < p - 1; c = c + 2)

 {

 int x = (int)remove.GetValue(c);

 int y = (int)remove.GetValue(c + 1);

 thinBmp.SetPixel(x, y, Color.Black);

 }

 if (p != 0)

 {

 finish = false;

 }

 //if no pixel to remove, stop the loop

 else

 {

 finish = true;

 }

 }

 //if no pixel to remove, stop the loop

 else

 {

 finish = true;

 }

 } while (finish != true);

 //close process box

 pb.Dispose();

 return thinBmp;

 }

 }

}

***Minutiae.cs***

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The Minutiae Class contains all functions which are used to extract all

 //minutiae of the fingerprint and then eliminate fake minutiae

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 class Minutiae

 {

 public int[] tMinu = new int[10000];

 Bitmap bt;

 int[] minutiae = new int[10000];

 int p = 0;

 //this function is used to extract all minutiae of the fingerprint

 public Bitmap MinuExtract(Bitmap bm)

 {

 Bitmap cBmp = new Bitmap(bm);

 int width = cBmp.Width;

 int height = cBmp.Height;

 int i, j, n, m; //indexes

 int[,] matrix = new int[3, 3];

 for (j = 1; j < height -1; j++)

 {

 for (i = 1; i < width -1; i++)

 {

 //get 3X3 parttern

 for (m = 0; m < 3; m++)

 {

 for (n = 0; n < 3; n++)

 {

 //set v to 0 if color of the pixel is black, and 1 if is white

 int v = (cBmp.GetPixel(i + m - 1, j + n - 1).R == 255) ? 1 : 0;

 matrix[m, n] = v;

 }

 }

 // if color of the centre pixle is white

 if (matrix[1, 1] == 1)

 {

 int o = Math.Abs(-1-1) + Math.Abs(1);

 int nCount =(Math.Abs(matrix[0, 0] - matrix[1, 0]) +

 Math.Abs(matrix[1, 0] - matrix[2, 0]) +

 Math.Abs(matrix[2, 0] - matrix[2, 1]) +

 Math.Abs(matrix[2, 1] - matrix[2, 2]) +

 Math.Abs(matrix[2, 2] - matrix[1, 2]) +

 Math.Abs(matrix[1, 2] - matrix[0, 2]) +

 Math.Abs(matrix[0, 2] - matrix[0, 1]) +

 Math.Abs(matrix[0, 1] - matrix[0, 0]));

 // check if it is a ridge end

 if (nCount / 2 == 1)

 {

 //add the pixel into minutiae array

 minutiae.SetValue(i, p);

 p++;

 minutiae.SetValue(j, p);

 p++;

 minutiae.SetValue(0, p); // 0 means ridge end;

 p++;

 }

 // check if it is a ridge bifurcation

 if (nCount / 2 == 3)

 {

 //add the pixel into minutiae array

 minutiae.SetValue(i, p);

 p++;

 minutiae.SetValue(j, p);

 p++;

 minutiae.SetValue(1, p); // 1 means ridge bifurcation;

 p++;

 }

 }

 }

 }

 // call MinuEliminate function to eliminate fake minutiae and return processed image

 bt = MinuEliminate(cBmp);

 return bt;

 }

 // this function is used to eliminate fake minutiae and indicate the true minutiae on the image

 public Bitmap MinuEliminate(Bitmap im)

 {

 Bitmap minBmp = new Bitmap(im);

 int x, y;//x,y coordinates of minutiae.

 int i=0; //indexes

 for (int c = 0; c < p - 1; c = c + 3)

 {

 int[,] matchMatrix = new int[11, 11];

 //set central point vlaue as -1

 matchMatrix[5, 5] = -1;

 //set next check point coordinates

 int nextX = 5, nextY = 5;

 //set counters

 int loopCount=0;

 int Count1 = 0;

 int Count2 = 0;

 int Count3 = 0;

 //get minutiae x,y coordinates

 x = (int)minutiae.GetValue(c);

 y = (int)minutiae.GetValue(c + 1);

 //check image boundary, 5~250.

 if ((x >= 5 && x <= 250) && (y >= 5 && y <= 250))

 {

 //initial matchMatrix with 5 and 0, 5 means a ridge pixel

 for (int m = 0; m < 11; m++)

 {

 for (int n = 0; n < 11; n++)

 {

 int v = (im.GetPixel(x + m - 5, y + n - 5).R == 255) ? 5 : 0;

 if (!(m == 5 && n == 5))

 { matchMatrix[m, n] = v; }

 }

 }

 //3X3 neighbor points check matrix

 int[,] matrix = new int[3, 3];

 //check if minutia is a ridge end

 if((int)minutiae.GetValue(c + 2) ==0)

 {

 while (loopCount <= 25 && nextX < 10 && nextX > 0 && nextY < 10 && nextY >0)

 {

 int tNextX=1;

 int tNextY=1;

 //set check matirx value

 for (int a = 0; a < 3; a++)

 {

 for (int b = 0; b < 3; b++)

 {

 int v = matchMatrix[nextX + a - 1, nextY + b - 1];

 matrix[a, b] = v;

 /\* if value is 5, set matchMatrix value to 1 and set the point as centrel point of next check matrix \*/

 if (v == 5)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 1;

 tNextX = nextX + a - 1;

 tNextY = nextY + b - 1;

 }

 }

 }

 nextX = tNextX;

 nextY = tNextY;

 loopCount++;

 }

 for (int c1 = 0; c1 < 10; c1++)

 {

 if ((matchMatrix[c1, 0]==0||matchMatrix[c1, 0]==5||matchMatrix[c1,0]==1) && (matchMatrix[c1+1, 0] == 1))

 Count1++;

 if ((matchMatrix[10, c1] == 0 || matchMatrix[10, c1] == 5 || matchMatrix[10, c1] == 1) && (matchMatrix[10, c1 + 1] == 1))

 Count1++;

 }

 for (int c1 = 10; c1 > 0; c1--)

 {

 if ((matchMatrix[c1, 10] == 0 || matchMatrix[c1, 10] == 5 || matchMatrix[c1, 10] == 1) && (matchMatrix[c1 - 1, 10] == 1))

 Count1++;

 if ((matchMatrix[0, c1] == 0 || matchMatrix[0, c1] == 5 || matchMatrix[0, c1] == 1) && (matchMatrix[0, c1 - 1] == 1))

 Count1++;

 }

 //if count1 is not 0, the minutia is a true end minutia

 if (Count1 != 0)

 {

 //indicate the minutia point on the image with Red colour

 for (int xx = 0; xx < 3; xx++)

 {

 for (int yy = 0; yy < 3; yy++)

 {

 minBmp.SetPixel(x + xx - 1, y + yy - 1, Color.Red);

 }

 }

 tMinu.SetValue(x, i); // add X coor into true minutiae array

 i++;

 tMinu.SetValue(y, i);// add Y coor into true minutiae array

 i++;

 tMinu.SetValue(0, i);// add minutiae type into true minutiae array

 i++;

 //calculate the distance between the minutia to the core

 double d = Math.Sqrt(Math.Abs(MainForm.coreX - x) \* Math.Abs(MainForm.coreX - x) + Math.Abs(MainForm.coreY - y) \* Math.Abs(MainForm.coreY - y));

 d = Math.Round(d, 0);

 tMinu.SetValue((int)d, i); // add distance into true minutiae array

 i++;

 // angle difference between the minutiae and the core

 int an1 = MainForm.Orientation[x, y];

 int an2 = MainForm.Orientation[MainForm.coreX,MainForm.coreY];

// add angle difference into true minutiae array

tMinu.SetValue(Math.Abs(an1 - an2), i);

i++;

 }

 }

 //if minutia is a ridge bifurcation

 if ((int)minutiae.GetValue(c + 2) == 1)

 {

 int t1NextX = 1;

 int t1NextY = 1;

 int t2NextX = 1;

 int t2NextY = 1;

 int t3NextX = 1;

 int t3NextY = 1;

 int sequence = 1;

 //first check central point

 for (int a = 0; a < 3; a++)

 {

 for (int b = 0; b < 3; b++)

 {

 int v = matchMatrix[nextX + a - 1, nextY + b - 1];

 matrix[a, b] = v;

 /\* if value is 5, set matchMatrix value to 1 and set the point as centrel point of next check matrix \*/

 if (v == 5)

 {

 if (sequence == 1)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 1;

 t1NextX = nextX + a - 1;

 t1NextY = nextY + b - 1;

 sequence = 2;

 }

 else if (sequence == 2)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 2;

 t2NextX = nextX + a - 1;

 t2NextY = nextY + b - 1;

 sequence = 3;

 }

 else if (sequence == 3)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 3;

 t3NextX = nextX + a - 1;

 t3NextY = nextY + b - 1;

 }

 }

 }

 }

 //set next check metrix central.

 nextX = t1NextX;

 nextY = t1NextY;

 //set check matirx value for first branch

 while (loopCount <= 25 && nextX < 10 && nextX > 0 && nextY < 10 && nextY > 0)

 {

 for (int a = 0; a < 3; a++)

 {

 for (int b = 0; b < 3; b++)

 {

 int v = matchMatrix[nextX + a - 1, nextY + b - 1];

 matrix[a, b] = v;

/\* if value is 5, set matchMatrix value to 1 and set the point as centrel point of next check matrix \*/

 if (v == 5)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 1;

 t1NextX = nextX + a - 1;

 t1NextY = nextY + b - 1;

 }

 }

 }

 nextX = t1NextX;

 nextY = t1NextY;

 loopCount++;

 }

 //set next check metrix central.

 nextX = t2NextX;

 nextY = t2NextY;

 //set check matirx value for second branch

 while (loopCount <= 25 && nextX < 10 && nextX > 0 && nextY < 10 && nextY > 0)

 {

 for (int a = 0; a < 3; a++)

 {

 for (int b = 0; b < 3; b++)

 {

 int v = matchMatrix[nextX + a - 1, nextY + b - 1];

 matrix[a, b] = v;

 /\*if value is 5, set matchMatrix value to 1 and set the point as centrel point of next check matrix \*/

 if (v == 5)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 2;

 t2NextX = nextX + a - 1;

 t2NextY = nextY + b - 1;

 }

 }

 }

 nextX = t2NextX;

 nextY = t2NextY;

 loopCount++;

 }

 //set next check metrix central.

 nextX = t3NextX;

 nextY = t3NextY;

 //set check matirx value for second branch

 while (loopCount <= 25 && nextX < 10 && nextX > 0 && nextY < 10 && nextY > 0)

 {

 for (int a = 0; a < 3; a++)

 {

 for (int b = 0; b < 3; b++)

 {

 int v = matchMatrix[nextX + a - 1, nextY + b - 1];

 matrix[a, b] = v;

 /\* if value is 5, set matchMatrix value to 1 and set the point as centrel point of next check matrix \*/

 if (v == 5)

 {

 matchMatrix[nextX + a - 1, nextY + b - 1] = 3;

 t3NextX = nextX + a - 1;

 t3NextY = nextY + b - 1;

 }

 }

 }

 nextX = t3NextX;

 nextY = t3NextY;

 loopCount++;

 }

 //counter1,2,3

 for (int c1 = 0; c1 < 10; c1++)

 {

 if ((matchMatrix[c1, 0] == 0 || matchMatrix[c1, 0] == 5 || matchMatrix[c1, 0] == 1) && (matchMatrix[c1 + 1, 0] == 1))

 Count1++;

 if ((matchMatrix[10, c1] == 0 || matchMatrix[10, c1] == 5 || matchMatrix[10, c1] == 1) && (matchMatrix[10, c1 + 1] == 1))

 Count1++;

 if ((matchMatrix[c1, 0] == 0 || matchMatrix[c1, 0] == 5 || matchMatrix[c1, 0] == 2) && (matchMatrix[c1 + 1, 0] == 2))

 Count2++;

 if ((matchMatrix[10, c1] == 0 || matchMatrix[10, c1] == 5 || matchMatrix[10, c1] == 2) && (matchMatrix[10, c1 + 1] == 2))

 Count2++;

 if ((matchMatrix[c1, 0] == 0 || matchMatrix[c1, 0] == 5 || matchMatrix[c1, 0] == 3) && (matchMatrix[c1 + 1, 0] == 3))

 Count3++;

 if ((matchMatrix[10, c1] == 0 || matchMatrix[10, c1] == 5 || matchMatrix[10, c1] == 3) && (matchMatrix[10, c1 + 1] == 3))

 Count3++;

 }

 for (int c1 = 10; c1 > 0; c1--)

 {

 if ((matchMatrix[c1, 10] == 0 || matchMatrix[c1, 10] == 5 || matchMatrix[c1, 10] == 1) && (matchMatrix[c1 - 1, 10] == 1))

 Count1++;

 if ((matchMatrix[0, c1] == 0 || matchMatrix[0, c1] == 5 || matchMatrix[0, c1] == 1) && (matchMatrix[0, c1 - 1] == 1))

 Count1++;

 if ((matchMatrix[c1, 10] == 0 || matchMatrix[c1, 10] == 5 || matchMatrix[c1, 10] == 2) && (matchMatrix[c1 - 1, 10] == 2))

 Count2++;

 if ((matchMatrix[0, c1] == 0 || matchMatrix[0, c1] == 5 || matchMatrix[0, c1] == 2) && (matchMatrix[0, c1 - 1] == 2))

 Count2++;

 if ((matchMatrix[c1, 10] == 0 || matchMatrix[c1, 10] == 5 || matchMatrix[c1, 10] == 3) && (matchMatrix[c1 - 1, 10] == 3))

 Count3++;

 if ((matchMatrix[0, c1] == 0 || matchMatrix[0, c1] == 5 || matchMatrix[0, c1] == 3) && (matchMatrix[0, c1 - 1] == 3))

 Count3++;

 }

 // if couner 1,2,3 are not equal to 0, the minutiae is a true bifurcation minutia

 if (Count1 != 0 && Count2 != 0 && Count3 != 0)

 {

 for (int xx = 0; xx < 3; xx++)

 {

 for (int yy = 0; yy < 3; yy++)

 {

 //indicate the minutia on the image with blue colour

 minBmp.SetPixel(x + xx - 1, y + yy - 1, Color.Blue);

 }

 }

 tMinu.SetValue(x, i);//add x coordinate into true minutiae array

 i++;

 tMinu.SetValue(y, i);//add y coordinate into true minutiae array

 i++;

 tMinu.SetValue(1, i);//1 means a bifurcation minutia

 i++;

 //distance to core point

 double d = Math.Sqrt(Math.Abs(MainForm.coreX - x) \* Math.Abs(MainForm.coreX - x) + Math.Abs(MainForm.coreY - y) \* Math.Abs(MainForm.coreY - y));

 d = Math.Round(d, 0);

 tMinu.SetValue((int)d, i);//add distance into true minutiae array

 i++;

 // angle difference between the minutiae and the core

 int an1 = MainForm.Orientation[x, y];

 int an2 = MainForm.Orientation[MainForm.coreX, MainForm.coreY];

// add angle different into true minutiae array

 tMinu.SetValue(Math.Abs(an1 - an2), i);

i++;

 }

 }

 }

 }

 FingerprintSpy.MainForm.index= i/5;

 return minBmp;

 }

 }

}

***CorePoint.cs***

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The CorePoint Class contains all functions which are used to normalize the original

 //image, get gradients of each pixel use Sobel Operator, calculate location orientation

 //field and determine the core point of the fingerprint

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public class CorePoint

 {

 public int[,] Oma = new int[256, 256]; // Orientation angle of each pixel

 public int coX; //sample pixel X coor

 public int coY; // sample pixel Y coor

 Bitmap image; // Processed image

 int[,,] Gma = new int [256,256,2]; // a 3-dimensional matrix hold gradients of pixels in the image.

 //this function is used to normalize the unprocessed image

 public void Normalization(Bitmap o)

 {

 Bitmap normalized = new Bitmap(o);

 int i, j, N;

 uint tmp = 0;

 uint M, M1, VAR;

 // Calculating the Mean of the image

 Random r = new Random();

 coX = r.Next(98, 145);

 coY = r.Next(136, 185);

 for (i = 0; i < normalized.Height - 1; i++)

 {

 for (j = 0; j < normalized.Width - 1; j++)

 {

 Color clr = normalized.GetPixel(i, j);

 if ((clr.R == Convert.ToByte(75)) && (clr.G == Convert.ToByte(50)) && (clr.B == Convert.ToByte(74)))

 {

 coX = i;

 coY = j;

 }

 tmp = tmp + Convert.ToUInt32(clr.R);

 }

 }

 N = normalized.Height \* normalized.Width;

 M = Convert.ToUInt32(Math.Abs(tmp / N));

 // Calculating the Variance of the image

 tmp = 0;

 for (i = 0; i < normalized.Height - 1; i++)

 {

 for (j = 0; j < normalized.Width - 1; j++)

 {

 Color clr = normalized.GetPixel(i, j);

 M1 = ((Convert.ToUInt32(clr.R) - M) \* (Convert.ToUInt32(clr.R) - M));

 tmp = tmp + M1;

 }

 }

 VAR = Convert.ToUInt32(Math.Abs(tmp / N));

 uint MO = 50, VARO = 50, newInten;

 Color Inten;

 //normalize the image

 for (i = 0; i < normalized.Height - 1; i++)

 {

 for (j = 0; j < normalized.Width - 1; j++)

 {

 Inten = normalized.GetPixel(i, j);

 //if current pixel's red value is greater than threshold.

 if (Convert.ToUInt32(Inten.R) > M)

 {

 newInten = MO + Convert.ToUInt32(Math.Sqrt((VARO \* ((Convert.ToUInt32(Inten.R) - M) \* (Convert.ToUInt32(Inten.R) - M))) / VAR));

 normalized.SetPixel(i, j, Color.FromArgb(255, Convert.ToInt32(newInten), Convert.ToInt16(newInten), Convert.ToInt16(newInten)));

 }

 //if current pixel's red value is less than threshold.

 else

 {

 newInten = MO - Convert.ToUInt32(Math.Sqrt((VARO \* ((Convert.ToUInt32(Inten.R) - M) \* (Convert.ToUInt32(Inten.R) - M))) / VAR));

 normalized.SetPixel(i, j, Color.FromArgb(255, Convert.ToInt32(newInten), Convert.ToInt16(newInten), Convert.ToInt16(newInten)));

 }

 }

 }

 image = normalized;

 }

 //this function is used to cauclulate gradients of each pixel in the image with Sobel operator

 public void Sobel()

 {

 Bitmap sobelImage = new Bitmap(image);

 int row = sobelImage.Height;

 int col = sobelImage.Width;

 int gradientX = 0;

 int gradientY = 0;

 //Loops to fill pixels into 3X3 Sobel operator

 for (int r = 1; r < row - 1; r++)

 {

 for(int c=1;c < col - 1; c++)

 {

 int[] p = new int[10]; // 9 pixels in the operator,but int [10] is more straightforward.

 p[1] = Convert.ToInt32(sobelImage.GetPixel(r - 1, c - 1).R);

 p[2] = Convert.ToInt32(sobelImage.GetPixel(r - 1, c).R);

 p[3] = Convert.ToInt32(sobelImage.GetPixel(r - 1, c + 1).R);

 p[4] = Convert.ToInt32(sobelImage.GetPixel(r, c - 1).R);

 p[6] = Convert.ToInt32(sobelImage.GetPixel(r, c + 1).R);

 p[7] = Convert.ToInt32(sobelImage.GetPixel(r + 1, c - 1).R);

 p[8] = Convert.ToInt32(sobelImage.GetPixel(r + 1, c).R);

 p[9] = Convert.ToInt32(sobelImage.GetPixel(r + 1, c + 1).R);

 //get gradients of pixel on X and Y coordinates

 gradientX = Math.Abs(p[7] - p[1] + (2 \* p[8]) - (2 \* p[2]) + p[9] - p[3]);

 gradientY = Math.Abs(p[3] - p[1] + (2 \* p[6]) - (2 \* p[4]) + p[9] - p[7]);

 //set graients matrix

 Gma[r,c,0] = gradientX;

 Gma[r, c, 1] = gradientY;

 }

 }

 }

 //this funciton is used to detect local orientation of the fingerprint

 public void LocalOrientation()

 {

 for (int ri = 2; ri <= 252; ri=ri + 3)

 {

 for (int ci = 2; ci <= 252;ci= ci + 3)

 {

 //get gradients value on X coordinate

 double Vx = (2 \* Gma[ri - 1, ci - 1, 0] \* Gma[ri - 1, ci - 1, 1])+

 (2 \* Gma[ri - 1, ci, 0] \* Gma[ri - 1, ci, 1])+

 (2 \* Gma[ri - 1, ci + 1, 0] \* Gma[ri - 1, ci + 1, 1])

 +

 (2 \* Gma[ri , ci - 1, 0] \* Gma[ri , ci - 1, 1]) +

 (2 \* Gma[ri , ci, 0] \* Gma[ri , ci, 1]) +

 (2 \* Gma[ri , ci + 1, 0] \* Gma[ri , ci + 1, 1])

 +

 (2 \* Gma[ri + 1, ci - 1, 0] \* Gma[ri + 1, ci - 1, 1]) +

 (2 \* Gma[ri + 1, ci, 0] \* Gma[ri + 1, ci, 1]) +

 (2 \* Gma[ri + 1, ci + 1, 0] \* Gma[ri + 1, ci + 1, 1]);

 //get gradients value on Y coordinate

 double Vy = (Gma[ri - 1, ci - 1, 0] \* Gma[ri - 1, ci - 1, 1] \* Gma[ri - 1, ci - 1, 0] \* Gma[ri - 1, ci - 1, 1]) +

 (Gma[ri - 1, ci, 0] \* Gma[ri - 1, ci, 1] \* Gma[ri - 1, ci, 0] \* Gma[ri - 1, ci, 1]) +

 (Gma[ri - 1, ci + 1, 0] \* Gma[ri - 1, ci + 1, 1] \* Gma[ri - 1, ci + 1, 0] \* Gma[ri - 1, ci + 1, 1]) +

 (Gma[ri, ci - 1, 0] \* Gma[ri, ci - 1, 1] \* Gma[ri, ci - 1, 0] \* Gma[ri, ci - 1, 1]) +

 (Gma[ri, ci, 0] \* Gma[ri, ci, 1] \* Gma[ri, ci, 0] \* Gma[ri, ci, 1]) +

 (Gma[ri, ci + 1, 0] \* Gma[ri, ci + 1, 1] \* Gma[ri, ci + 1, 0] \* Gma[ri, ci + 1, 1]) +

 (Gma[ri + 1, ci - 1, 0] \* Gma[ri + 1, ci - 1, 1] \* Gma[ri + 1, ci - 1, 0] \* Gma[ri + 1, ci - 1, 1]) +

 (Gma[ri + 1, ci, 0] \* Gma[ri + 1, ci, 1] \* Gma[ri + 1, ci, 0] \* Gma[ri + 1, ci, 1]) +

 (Gma[ri + 1, ci + 1, 0] \* Gma[ri + 1, ci + 1, 1] \* Gma[ri + 1, ci + 1, 0] \* Gma[ri + 1, ci + 1, 1]);

 int angle;

 //check if Vx and Vy are not equal to 0

 if ((Vx != 0) && (Vy != 0))

 {

 //get angle of the centre pixel

 angle =Convert.ToInt16( 180 \* ((Math.Atan(Vy / Vx)) / 2) / Math.PI);

 }

 else

 {

 angle = 0;

 }

 //set local orientation matrix

 Oma[ri - 1, ci - 1] = angle;

 Oma[ri - 1, ci] = angle;

 Oma[ri - 1, ci + 1] = angle;

 Oma[ri, ci - 1] = angle;

 Oma[ri, ci] = angle;

 Oma[ri, ci + 1] = angle;

 Oma[ri + 1, ci - 1] = angle;

 Oma[ri + 1, ci] = angle;

 Oma[ri + 1, ci + 1] = angle;

 }

 }

 }

 //this function is used to get core point area of the image

 public void Point()

 {

 double diffY;

 double diffX;

 int[,] sin = new int [3,3]; // 3 X 3 Sin matrix

 int[,] cos = new int [3,3]; // 3 X 3 Cos matrix

 double sinSum1 = 0;

 double sinSum2 = 0;

 double cosSum1 = 0;

 double cosSum2 = 0;

 for (int row = 2; row <= 252; row += 3)

 {

 for (int col = 2; col <= 252; col += 3)

 {

 sin[0, 0] = Oma[row - 1, col - 1];

 sin[0, 1] = Oma[row - 1,col];

 sin[0, 2] = Oma[row - 1, col + 1];

 sin[1, 0] = Oma[row, col - 1];

 sin[1, 1] = Oma[row, col];

 sin[1, 2] = Oma[row, col + 1];

 sin[2, 0] = Oma[row + 1, col - 1];

 sin[2, 1] = Oma[row + 1, col];

 sin[2, 2] = Oma[row + 1, col + 1];

 cos[0, 0] = Oma[row - 1, col - 1];

 cos[0, 1] = Oma[row - 1, col];

 cos[0, 2] = Oma[row - 1, col + 1];

 cos[1, 0] = Oma[row, col - 1];

 cos[1, 1] = Oma[row, col];

 cos[1, 2] = Oma[row, col + 1];

 cos[2, 0] = Oma[row + 1, col - 1];

 cos[2, 1] = Oma[row + 1, col];

 cos[2, 2] = Oma[row + 1, col + 1];

 for (int rowc = 0; rowc < 3; rowc++)

 {

 sinSum1 += 2 \* Math.Sin((double)sin[rowc, 2]);

 sinSum2 += 2 \* Math.Sin((double)sin[rowc, 0]);

 }

 // set different value of Sin as the different value on Y coordinate

 diffY = sinSum1 - sinSum2;

 for (int colc = 0; colc < 3; colc++)

 {

 cosSum1 += 2 \* Math.Cos((double)cos[2, colc]);

 cosSum2 += 2 \* Math.Cos((double)cos[0, colc]);

 }

 // set different value of Cos as the different value on X coordinate

 diffX = cosSum1 - cosSum2;

 if ((diffX < 0 ) && (diffY < 0))

 {

 // the corrdinates of core point is the pixel with both diffX and diffY greater than 0

 coX = row;

 coY = col;

 }

 }

 }

 }

 }

}

***Match.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FingerprintSpy

{

 class Match

 {

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The Match class contains functions to match two 2 dimentional array which

 //contain distance to core point, angle different and minutiae type.

 //Return matching result after match

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 public String[] Matching(int[,] fingerprint1, int[,] fingerprint2)

 {

 int matchedPairs = 0; // no of matched pairs

 int[,] templete, sample;

 // fingerprint with less minutiae number is the templete

 if (fingerprint1.GetLength(0) <= fingerprint2.GetLength(0))

 {

 templete = fingerprint1;

 sample = fingerprint2;

 }

 else

 {

 templete = fingerprint2;

 sample = fingerprint1;

 }

 bool matched = false; // matched flag

 for (int i = 0; i < templete.GetLength(0); i++)

 {

 matched = false;

 if (i <= sample.GetLength(0) - 3)

 {

 //first match

 if (i == 0)

 {

 // maximum 5 times match for one templete minutia.

 for (int p = i; p <= 4; p++)

 {

 //check if two minutiae are same type and matched flag is false

 if ((templete[i, 2] == sample[p, 2]) && (!matched))

 {

 /\* check if the difference of their distance to core is less than 6 pixels and difference of their angle is less than 6 degree\*/

 if ((Math.Abs(templete[i, 0] - sample[p, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[p,1]) <= 5))

 {

 sample[p, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 }

 }

 //second matches

 else if (i == 1)

 {

 if (templete[i, 2] == sample[i - 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 1, 1]) <= 5))

 {

 sample[i - 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i, 1]) <= 5))

 {

 sample[i, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 1, 1]) <= 5))

 {

 sample[i + 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 2, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 2, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 2, 1]) <= 5))

 {

 sample[i + 2, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 3, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 3, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 3, 1]) <= 5))

 {

 sample[i + 3, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 }

 //other matches

 else

 {

 if (templete[i, 2] == sample[i - 2, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 2, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 2, 1]) <= 5))

 {

 sample[i - 2, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 1, 1]) <= 5))

 {

 sample[i - 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i, 1]) <= 5))

 {

 sample[i, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 1, 1]) <= 5))

 {

 sample[i + 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 2, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 2, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 2, 1]) <= 5))

 {

 sample[i + 2, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 }

 }

 // second last match

 if (i == sample.GetLength(0) - 2)

 {

 if (templete[i, 2] == sample[i - 3, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 1, 1]) <= 5))

 {

 sample[i - 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 2, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 2, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 2, 1]) <= 5))

 {

 sample[i - 2, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 1, 1]) <= 5))

 {

 sample[i - 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i, 1]) <= 5))

 {

 sample[i, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i + 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i + 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i + 1, 1]) <= 5))

 {

 sample[i + 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 }

 // last match

 if (i == sample.GetLength(0) - 1)

 {

 if (templete[i, 2] == sample[i - 4, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 4, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 4, 1]) <= 5))

 {

 sample[i - 4, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 3, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 3, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 3, 1]) <= 5))

 {

 sample[i - 3, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 2, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 2, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 2, 1]) <= 5))

 {

 sample[i - 2, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i - 1, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i - 1, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i - 1, 1]) <= 5))

 {

 sample[i - 1, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 else if (templete[i, 2] == sample[i, 2])

 {

 if ((Math.Abs(templete[i, 0] - sample[i, 0]) <= 5) && (Math.Abs(templete[i, 1] - sample[i, 1]) <= 5))

 {

 sample[i, 2] = -1; // mark as a matched minutia

 matchedPairs++;

 //set matched flag to true

 matched = true;

 }

 }

 }

 }

 double m = matchedPairs;

 double total = templete.GetLength(0);

 double temp = matchedPairs / total;

 double r = Math.Round(temp, 4) \* 100;

 //set up result string array

 String[] result = {templete.GetLength(0).ToString(), matchedPairs.ToString(), r.ToString()};

 return result;

 }

 }

}

***Recognition.cs***

using System;

using System.Globalization;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Threading;

using System.Text;

using System.Windows.Forms;

using System.IO;

namespace FingerprintSpy

{

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 //

 //The Recognition Class contains all functions which are used to recognize a unenrolled

 //image from the database and output matching result and the enrolled fingerprint image

 //

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 class Recognition

 {

 Bitmap Fingerprint;

 int[,] matchMatrix1, matchMatrix2;

 int noOfMin;

 int[] minutiae = new int[10000];

 int[] finalMin = new int[1000];

 int tMinIndex = 0;

 String matchedID;

 double matchScore = 0.00;

 /\* this function is used to run image pre-processing(Enhance,Thinning,Minutiae extraxtion,Core point detection) on the unenrolled image\*/

 public Bitmap Preprocessing(Bitmap fp)

 {

 Fingerprint = fp;

 //Enhance the fingerprint

 Enhance en = new Enhance();

 Fingerprint = en.enhance(fp);

 //Thin the fingerprint

 Thinning th = new Thinning();

 Fingerprint = th.ThiningPic(Fingerprint);

 //Detect core point area of the fingerprint

 CorePoint cp = new CorePoint();

 cp.Normalization(fp);

 cp.Sobel();

 cp.LocalOrientation();

 cp.Point();

 MainForm.coreX = cp.coX;

 MainForm.coreY = cp.coY;

 MainForm.Orientation = cp.Oma;

 //Extract minutiae of the fingerprint

 Minutiae mi = new Minutiae();

 Fingerprint = mi.MinuExtract(Fingerprint);

 minutiae = mi.tMinu;

 tMinIndex = MainForm.index;

 //Eliminate minutiae which distances are greater than 60 pixels

 int k = 0;

 for (int i = 4; i < tMinIndex \* 5; i += 5)

 {

 if (minutiae[i - 1] <= 60) // distance less than 60

 {

 int x = minutiae[i - 4];

 finalMin[k] = minutiae[i - 4];

 k++;

 int y = minutiae[i - 3];

 finalMin[k] = minutiae[i - 3];

 k++;

 finalMin[k] = minutiae[i - 2];

 k++;

 finalMin[k] = minutiae[i - 1];

 k++;

 finalMin[k] = minutiae[i];

 k++;

 }

 }

 noOfMin = k / 5;

 //sort the minutiae array

 for (int i = 4; i < k; i += 5)

 {

 for (int j = i; j < k; j += 5)

 {

 if (finalMin[i - 1] > finalMin[j - 1])

 {

 int tmp0 = finalMin[i - 4];

 int tmp1 = finalMin[i - 3];

 int tmp2 = finalMin[i - 2];

 int tmp3 = finalMin[i - 1];

 int tmp4 = finalMin[i];

 finalMin[i - 4] = finalMin[j - 4];

 finalMin[i - 3] = finalMin[j - 3];

 finalMin[i - 2] = finalMin[j - 2];

 finalMin[i - 1] = finalMin[j - 1];

 finalMin[i] = finalMin[j];

 finalMin[j - 4] = tmp0;

 finalMin[j - 3] = tmp1;

 finalMin[j - 2] = tmp2;

 finalMin[j - 1] = tmp3;

 finalMin[j] = tmp4;

 }

 }

 }

 //Indicate core point area and minutiae on the original fingerprint

 for (int x = 1; x < fp.Height - 1; x++)

 {

 for (int y = 1; y < fp.Width - 1; y++)

 {

 Color c = Fingerprint.GetPixel(x, y);

 if ((c.R == 255)&&(c.B ==0)&&(c.G==0))

 {

 fp.SetPixel(x, y, Color.Red);

 }

 if ((c.R == 0) && (c.B == 255) && (c.G == 0))

 {

 fp.SetPixel(x, y, Color.Blue);

 }

 }

 }

 return fp;

 }

 /\*this function is used to find the most similary enrolled fingerprint

 of the unenrolled fingerprint, it keeps searching database until reach

 the end or find out a enrolled fingerprint which has a very high similarity\*/

 public String[] StartSearch()

 {

 String[] result = { "", "", "" };

 String[] finalResult = { "","","","" };

 OleDbConnection conn = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" +

 "Data Source=" + MainForm.databaseDirectory + "\\Fingerprint.accdb;Jet OLEDB:Database Password=081021");

 DataTable FingerTable;

 DataTable MinTable;

 int i = 0;

 /\* for every enrolled fingerprint record, goto minutiae table to find its minutiae and assign matchMatrix2 with these minutiae records \*/

 do

 {

 /\* set up matchMatrix1 which contains minutiae details of the unenrolled fingerprint with its distance less than 60 pixels \*/

 int[,] matchMatrix1 = new int[noOfMin, 3];

 int index =4;

 for (int ii = 0; ii < noOfMin; ii++)

 {

 matchMatrix1[ii, 0] = finalMin[index - 1];

 int d = matchMatrix1[ii, 0];

 matchMatrix1[ii, 1] = finalMin[index];

 int a = matchMatrix1[ii, 1];

 matchMatrix1[ii, 2] = finalMin[index - 2];

 int t = matchMatrix1[ii, 2];

 index+=5;

 }

 //get all enrolled fingerprint record from database.

 String sqlp = "SELECT \* FROM Fingerprint";

 OleDbDataAdapter fpda = new OleDbDataAdapter(sqlp, conn);

 DataSet fingers = new DataSet();

 fpda.Fill(fingers,"List");

 GC.Collect();

 FingerTable = fingers.Tables["List"];

 String fingerId = FingerTable.Rows[i]["Fingerprint\_ID"].ToString();

 //get minutiae details of the enrolled fingerprint

 String sqlm = "SELECT \* FROM Minutiae WHERE Fingerprint\_ID = '" +fingerId+"'";

OleDbDataAdapter mida = new OleDbDataAdapter(sqlm, conn);

 DataSet min = new DataSet();

 mida.Fill(min, "list");

 GC.Collect();

 MinTable = min.Tables["list"];

 matchMatrix2 = new int[MinTable.Rows.Count, 3];

 //set up matchMatrix2 array with those minutiae details

 for (int p = 0; p < MinTable.Rows.Count; p++)

 {

 matchMatrix2[p, 0] = Convert.ToInt16(MinTable.Rows[p]["Distance"]);

 matchMatrix2[p, 1] = Convert.ToInt16(MinTable.Rows[p]["Angle"]);

 matchMatrix2[p, 2] = Convert.ToInt16(MinTable.Rows[p]["Type"]);

 }

 //create a Match class instance and call Matching function to match these 2 minutiae arrays

 Match runMatch = new Match();

 result = runMatch.Matching(matchMatrix1, matchMatrix2);

 double score = Convert.ToDouble(result[2]);

 //check if the returned score is greater than the highest matching score

 if (score >= matchScore)

 {

 matchScore = score;

 matchedID = fingerId;

 //assign these matching result as elements of final result array

 finalResult[0] = matchedID;

 finalResult[1] = result[0];

 finalResult[2] = result[1];

 finalResult[3] = result[2];

 }

 i++;//index increase by 1

 } while ((i < FingerTable.Rows.Count)&&(matchScore < 95.0)); /\* search loop stop when match score is greater than 95 or no more fingerprint to compare.\*/

 //close database connection

 conn.Close();

 //return the final result

 return finalResult;

 }

 }

}