LAMPIRAN

Main Form visual Studio

```
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Data;
    using System.Drawing;
    using System.Linq;
    using System.Text;
    using System.Windows.Forms;
    using System.IO.Ports;

    using Emgu.CV.UI;
    using Emgu.CV;
    using Emgu.CV.Structure;
    using Emgu.CV.CvEnum;

    using System.IO;
    using System.Xml;
    using System.Runtime.InteropServices;
    using System.Threading;
    using System.Security.Principal;
    using System.Threading.Tasks;
    using Microsoft.Win32.SafeHandles;

        public partial class MainForm : Form
        {
            private SerialPort myport;
            #region variables
            int threshold=1;
            Image<Bgr, Byte> currentFrame; //display image dari webcam
            Image<Gray, byte> result, TrainedFace = null; //used to store the result image and
trained face
            Image<Gray, byte> gray_frame = null; //grayscale current image aquired from webcam for
processing

            Capture grabber; //This is our capture variable

            public CascadeClassifier Face = null;
            MCvFont font;//Our fount for writing within the frame

            int NumLabels;

            //Classifier with default training location
            ClassifierTrain Eigen_Recog = new ClassifierTrain();

            #endregion

            public MainForm()
            {
                InitializeComponent();
                trackBar1.Value = threshold;
                //membuka data training dr data sebelumnya
                init();
            }
            private void init()
```

```
            {
                myport = new SerialPort();
                myport.BaudRate = 9600;
                myport.PortName = "COM4";
                myport.Open();
            if (Eigen_Recog.IsTrained)
                {
                    message_bar.Text = "Training Data loaded";
                }
                else
                {
                    message_bar.Text = "No training data found, please train program using Train
menu option";
                }
                initialise_capture();

            }

            //Open training form and pass this
            private void trainToolStripMenuItem_Click(object sender, EventArgs e)
            {
                //Stop Camera
                stop_capture();

                //OpenForm
                TrainingForm TF = new TrainingForm(this); //file data trainingform
                TF.Show();
            }
            public void retrain()
            {

                    message_bar.Text = "Training Data loaded";
                }
                else
                {
                    message_bar.Text = "No training data found, please train program using Train
menu option";
                }
            }

            //Camera Start Stop
            public void initialise_capture()
            {
                //menampung data kamera

                grabber = new Capture();
                grabber.QueryFrame();
                Face = new CascadeClassifier(Application.StartupPath +
                font = new MCvFont(FONT.CV_FONT_HERSHEY_COMPLEX, 0.5, 0.5);
                //Initialize the FrameGraber event

                Application.Idle += new EventHandler(FrameGrabber_Parrellel);

            }
            private void stop_capture()
            {
                Application.Idle -= new EventHandler(FrameGrabber_Parrellel);

                if(grabber!= null)
```

```
            {
            grabber.Dispose();
            }
        }

        void FrameGrabber_Parrellel(object sender, EventArgs e)
        {
            //Get the current frame form capture device
            currentFrame = grabber.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
            Clear_label4();

            //Clear_Faces_Found();

            if (currentFrame != null)
            {
                gray_frame = currentFrame.Convert<Gray, Byte>();
                Rectangle gambar = new Rectangle(20, 60, 80, 80);

                //Rectangle[]  = DetectMultiScale(gray_frame, 1.2, 10, new Size(20, 20),
Size.Empty);

                try
                        {



                            result = currentFrame.Copy(gambar).Convert<Gray,
byte>().Resize(100, 100, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                            result._EqualizeHist();
                            //Image<Gray, Byte> img = result.Convert<Gray, Byte>();
                            //result = result.ThresholdBinary(new Gray(threshold), new
Gray(255));
                            pictureBox1.Image = result.ToBitmap();
                            currentFrame.Draw(gambar, new Bgr(Color.Red), 1);


                                currentFrame.Draw(name + " ", ref font, new Point(gambar.X - 2,
gambar.Y - 2), new Bgr(Color.Red ));
                                label3.Text = name;
                                myport.WriteLine("B");
                               if (name == "mutia")
                                myport.WriteLine("A");

                                //Convert it to Grayscal
                            }

                        }
                        catch
                        {

                        }

                    image_PICBX.Image = currentFrame.ToBitmap();
            }
```

```
            }
            void if_label()
            {
                label3.Text = "";



            }



            void Clear_label4()
            {
                label3.Text = "";

            }

            void ADD_Found(Image<Gray, Byte> img_found, string name_person, int match_value)
            {
                PictureBox PI = new PictureBox();
                PI.Location = new Point(faces_panel_X, faces_panel_Y);
                PI.Height = 80;
                PI.Width = 80;
                PI.SizeMode = PictureBoxSizeMode.StretchImage;
                PI.Image = img_found.ToBitmap();
                Label LB = new Label();
                LB.Text = name_person + " " + match_value.ToString();
                LB.Location = new Point(faces_panel_X, faces_panel_Y + 80);
                //LB.Width = 80;
                LB.Height = 15;


            }

            //Menu Opeartions
            private void exitToolStripMenuItem_Click(object sender, EventArgs e)
            {
                this.Dispose();
            }

            //Unknow Eigen face calibration
            private void Eigne_threshold_txtbxChanged(object sender, EventArgs e)
            {
                try
                {
                    Eigen_Recog.Set_Eigen_Threshold =
Math.Abs(Convert.ToInt32(Eigne_threshold_txtbx.Text));
                    message_bar.Text = "Eigen Threshold Set";
                }
                catch
                {
                    message_bar.Text = "Error in Threshold input please use int";
                }
            }

            private void MainForm_Load(object sender, EventArgs e)
            {
```

```
        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            threshold = trackBar1.Value;
            label2.Text = threshold.ToString();
        }

        private void image_PICBX_Click(object sender, EventArgs e)
        {

        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {

        }


    }
}
```

TrainingForm Visual Studio

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using Emgu.CV.UI;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;

using System.IO;
using System.Drawing.Imaging;
using System.Xml;
using System.Threading;

    public partial class TrainingForm : Form
    {
        #region Variables
        //Camera specific
        Capture grabber;
        int threshold=105;

        //Images for finding face
        Image<Bgr, Byte> currentFrame;
        Image<Gray, byte> result = null;
        Image<Gray, byte> gray_frame = null;

        //Classifier


        //For aquiring 10 images in a row
        List<Image<Gray, byte>> resultImages = new List<Image<Gray, byte>>();
        int results_list_pos = 0
        bool RECORD = false;

        //Saving Jpg
        List<Image<Gray, byte>> ImagestoWrite = new List<Image<Gray, byte>>();
        EncoderParameters ENC_Parameters = new EncoderParameters(1);
        EncoderParameter ENC = new EncoderParameter(System.Drawing.Imaging.Encoder.Quality,
100);
        ImageCodecInfo Image_Encoder_JPG;

        //Saving XAML Data file
        List<string> NamestoWrite = new List<string>();
        List<string> NamesforFile = new List<string>();
        XmlDocument docu = new XmlDocument();

        //Variables
        MainForm Parent;
        #endregion
```

```
        public TrainingForm(MainForm parent)
        {
            InitializeComponent();
            trackBar1.Value = threshold;

            Parent = parent;
            ENC_Parameters.Param[0] = ENC;
            Image_Encoder_JPG = GetEncoder(ImageFormat.Jpeg);
            initialise_capture();
        }

        private void Training_Form_FormClosing(object sender, FormClosingEventArgs e)
        {
            stop_capture();
            Parent.retrain();
            Parent.initialise_capture();
        }

        //Camera Start Stop
        public void initialise_capture()
        {
            grabber = new Capture();
            grabber.QueryFrame();
            //Initialize the FrameGraber event
            Application.Idle += new EventHandler(FrameGrabber);
        }
        private void stop_capture()
        {
            Application.Idle -= new EventHandler(FrameGrabber);
            if (grabber != null)
            {
                grabber.Dispose();
            }
            //Initialize the FrameGraber event
        }

        //Process Frame
        void FrameGrabber(object sender, EventArgs e)
        {
            //Get the current frame form capture device
            currentFrame = grabber.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);

            //Convert it to Grayscale
            if (currentFrame != null)
            {
                gray_frame = currentFrame.Convert<Gray, Byte>();
                Rectangle gambar = new Rectangle(20, 60, 80, 80);

                    result = currentFrame.Copy(gambar).Convert<Gray, byte>().Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                    result._EqualizeHist();
                    //Image<Gray, Byte> img = result .Convert<Gray, Byte>();
                     //result  = result .ThresholdBinary(new Gray(threshold ), new Gray(255));
                     currentFrame.Draw(gambar , new Bgr(Color.Red), 1);
```

```
                    image_PICBX.Image = currentFrame.ToBitmap();
                    if (RECORD && resultImages.Count < num_to_aquire)
                    {
                        resultImages.Add(result);
                        count_lbl.Text = "Count: " + resultImages.Count.ToString();
                        if (resultImages.Count == num_faces_to_aquire)
                        {
                            ADD_BTN.Enabled = true;
                            NEXT_BTN.Visible = true;
                            PREV_btn.Visible = true;
                            count_lbl.Visible = false;
                            Single_btn.Visible = true;
                            ADD_ALL.Visible = true;
                            RECORD = false;
                            Application.Idle -= new EventHandler(FrameGrabber);
                        }
                    }
                }
            }

            //Saving The Data
            private bool save_training_data(Image face_data)
            {
                try
                {
                    Random rand = new Random();
                    bool file_create = true;
                    string name = "data_" + NAME_PERSON.Text + "_" + rand.Next().ToString() +
".jpg";
                    while (file_create)
                    {
                            file_create = false;
                        }
                        else



                    {
                        //File.AppendAllText(Application.StartupPath +
"/TrainedFaces/TrainedLabels.txt", NAME_PERSON.Text + "\n\r");
                        bool loading = true;
                        while (loading)
                        {
                            try
                            {
                                docu.Load(Application.StartupPath +
"/TrainedFaces/TrainedLabels.xml");
                                loading = false;
                            }
                            catch
                            {
                                docu = null;
                                docu = new XmlDocument();
                                Thread.Sleep(10);
                            }
                        }
```

```
                //Get the root element
                XmlElement root = docu.DocumentElement;



                //Add the values for each nodes
                //name.Value = textBoxName.Text;
                //age.InnerText = textBoxAge.Text;
                //gender.InnerText = textBoxGender.Text;
                name_D.InnerText = NAME_PERSON.Text;
                file_D.InnerText = facename;

                //Construct the Person element
                //person.Attributes.Append(name);
                face_D.AppendChild(name_D);
                face_D.AppendChild(file_D);

                //Add the New person element to the end of the root element
                root.AppendChild(face_D);

                //Save the document
                docu.Save(Application.StartupPath + "/TrainedFaces/TrainedLabels.xml");
                //XmlElement child_element = docu.CreateElement("FACE");
                //docu.AppendChild(child_element);
                //docu.Save("TrainedLabels.xml");
            }
            else
            {


            return true;
        }
        catch (Exception ex)
        {
            return false;
        }

    }
    private ImageCodecInfo GetEncoder(ImageFormat format)
    {
        ImageCodecInfo[] codecs = ImageCodecInfo.GetImageDecoders();
        foreach (ImageCodecInfo codec in codecs)
        {
            if (codec.FormatID == format.Guid)
            {
                return codec;
            }
        }
        return null;
    }

    //Delete all the old training data by simply deleting the folder
    private void Delete_Data_BTN_Click(object sender, EventArgs e)
    {
        if (Directory.Exists(Application.StartupPath + "/TrainedFaces/"))
        {
            Directory.Delete(Application.StartupPath + "/TrainedFaces/", true);
            Directory.CreateDirectory(Application.StartupPath + "/TrainedFaces/");
```

```
    }
        }

        //Add the image to training data
        private void ADD_BTN_Click(object sender, EventArgs e)
        {
            if (resultImages.Count == num_faces_to_aquire)
            {
                if (!save_training_data(face_PICBX.Image)) MessageBox.Show("Error", "Error in
saving file info. Training data not saved", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                stop_capture();
                if (!save_training_data(face_PICBX.Image)) MessageBox.Show("Error", "Error in
saving file info. Training data not saved", MessageBoxButtons.OK, MessageBoxIcon.Error);
                initialise_capture();
            }
        }
        private void Single_btn_Click(object sender, EventArgs e)
        {
            RECORD = false;
            resultImages.Clear();
            NEXT_BTN.Visible = false;
            PREV_btn.Visible = false;
            Application.Idle += new EventHandler(FrameGrabber);
            Single_btn.Visible = false;
            count_lbl.Text = "Count: 0";
            count_lbl.Visible = true;
        }
        //Get 10 image to train
        private void RECORD_BTN_Click(object sender, EventArgs e)
        {
            if (NAME_PERSON.Text != "")
            {
                if (RECORD)
                {
                    RECORD = false;
                }
                else
                {
                    if (resultImages.Count == 10)
                    {
                        resultImages.Clear();
                        timer1.Enabled = false ;
                        Application.Idle += new EventHandler(FrameGrabber);
                    }
                    RECORD = true;
                    ADD_BTN.Enabled = false;
                    timer1.Enabled = true ;
                }
            }
            else
            {
                MessageBox.Show("Write Person Name First!! ");
            }

        }
        private void NEXT_BTN_Click(object sender, EventArgs e)
        {
            if (results_list_pos < resultImages.Count - 1)
```

```
    {
                face_PICBX.Image = resultImages[results_list_pos].ToBitmap();
                results_list_pos++;
                PREV_btn.Enabled = true;
            }
            else
            {
                NEXT_BTN.Enabled = false;
            }
        }
        private void PREV_btn_Click(object sender, EventArgs e)
        {
            if (results_list_pos > 0)
            {
                results_list_pos--;
                NEXT_BTN.Enabled = true;
            }
            else
            {
                PREV_btn.Enabled = false;
            }
        }
        private void ADD_ALL_Click(object sender, EventArgs e)
        {
            for(int i = 0; i<resultImages.Count;i++)
            {
                MessageBox.Show("Error", "Error in saving file info. Training data not saved",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                Thread.Sleep(100);
            }
            ADD_ALL.Visible = false;
            //restart single face detection
            Single_btn_Click(null, null);
        }

        private void TrainingForm_Load(object sender, EventArgs e)
        {

        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            threshold = trackBar1.Value;
            label3.Text = threshold.ToString();

        }

        private void timer1_Tick(object sender, EventArgs e)
        {

        }

        private void image_PICBX_Click(object sender, EventArgs e)

    }
}
```

Arduino

```
int in1 = 5;

int in2 = 6;

int ENA = 7;

int SPEED = 200;

int SPEED1 = 0;

int motordata;



void setup() {

  // put your setup code here, to run once:

pinMode(in1,OUTPUT);

pinMode(in2,OUTPUT);

pinMode(ENA,OUTPUT);

digitalWrite(in1,LOW);

digitalWrite(in2,LOW);
```

```
Serial.begin(9600);
}

void loop() {
 if (Serial.available() > 0)
  {
   motordata = Serial.read();

    if(motordata == 'A') // Single Quote! This is a character.
    {
   digitalWrite(in1,HIGH);
   digitalWrite(in2,LOW);
   analogWrite(ENA,SPEED);
delay(1000);
 digitalWrite(in1,LOW);
   digitalWrite(in2,LOW);
   analogWrite(ENA,SPEED1);
```

```
    }



    if(motordata == 'B') // Single Quote! This is a character.

    {

   digitalWrite(in1,LOW);

   digitalWrite(in2,LOW);

   analogWrite(ENA,SPEED1);

    }

  }

}
```

Box Counting Dimensi Fraktal Matlab

```
function varargout = ProgramBOXCounting(varargin)
% PROGRAMBOXCOUNTING M-file for ProgramBOXCounting.fig
%      PROGRAMBOXCOUNTING, by itself, creates a new PROGRAMBOXCOUNTING or raises
the existing
%      singleton*.
%
%      H = PROGRAMBOXCOUNTING returns the handle to a new PROGRAMBOXCOUNTING or the
handle to
%      the existing singleton*.
%
%      PROGRAMBOXCOUNTING('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROGRAMBOXCOUNTING.M with the given input
arguments.
%
%      PROGRAMBOXCOUNTING('Property','Value',...) creates a new PROGRAMBOXCOUNTING
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ProgramBOXCounting_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ProgramBOXCounting_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ProgramBOXCounting

% Last Modified by GUIDE v2.5 13-Jun-2017 03:52:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ProgramBOXCounting_OpeningFcn, ...
                   'gui_OutputFcn',  @ProgramBOXCounting_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before ProgramBOXCounting is made visible.
function ProgramBOXCounting_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ProgramBOXCounting (see VARARGIN)

% Choose default command line output for ProgramBOXCounting
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ProgramBOXCounting wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ProgramBOXCounting_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
[nama_file,nama_path]=uigetfile({'*.jpg';'*.bmp';'*.png';'*.tif'}, ...
 'Buka Citra');
 if~isequal(nama_file,0)
 handles.data1=imread(fullfile(nama_path,nama_file));
  guidata(hObject,handles);
 axes(handles.axes1)
  imshow(handles.data1)
 colormap gray
 axis square
title('Citra Asli');
 [row,col,val]=size(handles.data1);
 else
return
 end
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
[n,r] = boxcount(handles.data1);
 axes(handles.axes2);
```

```
boxcount(handles.data1);
axes(handles.axes3);

boxcount(handles.data1, 'slope');
df=-diff(log(n))./diff(log(r))

nilaiy = get(handles.pushbutton1,'value')

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```