AIU

ARAB INTERNATIONAL UNIVERSITY

Faculty of Informatics & Communication Engineering

Junior Project Report

On

"Vision-Based Arabic Sign Language Recognition"

Submitted to

Department of Informatics Engineering

June 2014

Vision-Based Arabic Sign Language Recognition

A junior project submitted in partial fulfillment of the requirement for the

Degree of Bachelor in

Informatics Engineering

Submitted by

Abdulrahman Khankan

Mohamad Alsioufi

Project Supervisors

Dr. Raouf Hamdan, PhD

Eng. Waed Khwies, MSc

June, 2014

Faculty of Informatics & Communication Engineering

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Computer and Electronics Engineering for acceptance, a project report entitled “Vision-Based Arabic Sign Language Recognition” submitted by Abdulrahman Khankan and Mohamad Alsioufi in partial fulfillment for the Bachelor of Engineering in Informatics

Dr. Raouf Hamdan, PhD

Acknowledgement

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Dr. Raouf Hamdan, PhD and Eng. Waed Khwies, MSc , for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

We are also grateful to our teachers for their constant support and guidance.

At the end we would like to express our sincere thanks to all our friends and family and others who helped us directly or indirectly during this project work.

Abdulrahman Khankan

Mohamad Alsioufi

Abstract

In this project, we facilitate human-machine interaction via hand-gestures captured by a normal webcam. This is accomplished by two subtasks. First, computer vision techniques are used to detect and track a hand, identifying key features such as the location of each fingertip and the center of its palm. Additionally, artificial intelligence is used to efficiently and accurately identify gestures in a predefined vocabulary. The system implements a Neural Network to both train and classify gestures. By using both computer vision techniques and artificial intelligence, enough knowledge is made available to the system to support human-machine interaction via hand-gestures.

Abbreviations

The following list of abbreviations were used in this report:

ANN: Artificial Neural Network

SVM: Support Vector Machine

HCI: Human Computer Interaction

MLP: Multi-Layer Perceptron

RGB: Red, Green, Blue color space

HSV: Hue, Saturation, Value color space

Table of Contents

Introduction

Wherever communities of deaf people exist, sign languages develop. Signing is not only used by the deaf it's also used by people who can hear, but cannot physically speak.

There are many people who use sign languages as their primary language. In the United States alone, approximately 450,000 deaf people use ASL (American Sign Language) as their primary language

In order to facilitate communication between deaf and hearing people, sign language interpreters are often used. Such activities involve considerable effort on the part of the interpreter, since sign languages are distinct natural languages with their own syntax, different from any spoken language.

This report will discuss in detail, a communication system between deaf and hearing people using a webcam and computer science technologies, the steps that were taken in building the system, the design of the system and the state of the art in this field.

Scope

Creating a robust and real-time system that recognize Arabic sign language alphabet characters as a step towards facilitating communication between deaf and hearing people using simple affordable hardware such as webcam and computer science technologies.

Literature Review

Image Processing

Digital image processing is the use of computer algorithms to perform image processing on digital images.

The following techniques were used

1- Skin segmentation using color pixel classification, which includes:
    a. Thresholding, which is the simplest method of image segmentation.
    b. Mathematical morphology operators to eliminate the noise.
    c. Different types of image filtering were used to examine different results.
2- Hand detection and segmentation
3- 2-D Image projection, which was used as a method to extract a feature vector

Machine Learning

Machine learning is a branch of artificial intelligence, concerns the construction and study of systems that can learn from data.

The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. Generalization is the property that the system will perform well on unseen data instances

There are two main types of machine learning:

1- Supervised learning:
    algorithms are trained on labelled examples, i.e., input where the desired output is known. The supervised learning algorithm attempts to generalize a function or mapping from inputs to outputs which can then be used speculatively to generate an output for previously unseen inputs.

2- Unsupervised learning:
   algorithms operate on unlabeled examples, i.e., input where the desired output is unknown. Here the objective is to discover structure in the data (e.g. through a cluster analysis), not to generalize a mapping from inputs to outputs.

This study uses supervised learning method of machine learning.

Neural Networks

Artificial neural networks (ANNs) are computational models inspired by an animal's central nervous systems (in particular the brain) which is capable of machine learning as well as pattern recognition.

Artificial neural networks are generally presented as systems of interconnected "neurons" which can compute values from inputs.

Like other machine learning methods neural networks have been used to solve a wide variety of tasks that are hard to solve using ordinary rule-based programming, including computer vision and speech recognition.

There are many types of ANNs, the type that was used in this study is Multi-Layer Perceptron (MLP) which is a feedforward network. With bipolar sigmoid activation function and backpropagation training method.

Sign language

A sign language is a language which uses manual communication and body language to convey meaning, as opposed to acoustically conveyed sound patterns.

This can involve simultaneously combining hand shapes, orientation and movement of the hands, arms or body, and facial expressions to fluidly express a speaker's thoughts.

Wherever communities of deaf people exist, sign languages develop. Signing is not only used by the deaf it's also used by people who can hear, but cannot physically speak.

There are typically three types of signing (Drew, 2004):

1- Words spelling, which is the most used type of spelling where meanings are represented by hands and body moves (Gestures)
2- Fingerspelling, which is the less used type of spelling where words are made using a manual alphabet (Postures). Fingerspelling is used to complement the vocabulary of the sign language when spelling individual letters of a word is the preferred or only option, such as with proper names or the titles of works, etc.
3- Non-manual features, facial expressions and tongue, mouth and body position

This study only focuses on fingerspelling of the Arabic alphabet.

State of the art

First we have to distinguish between the terms gesture recognition and posture recognition.

A gesture is a way of communication that involves body part movement, and thus, a gesture recognition is the recognition of dynamic related sequence of images (video).

A posture on the other hand, is a static image that involves no movement, and thus, a posture recognition is the recognition of static images that are not related to each other.

This study proposes a solution for fingerspelling of Arabic sign language, which is a posture recognition system.

Sign Language Recognition Systems

The problem of recognizing the sign language in real time was intensively studied in the past in prestigious universities like MIT, University of Milan and the Royal Melbourne Institute of Technology, as well as in private companies such as Fujitsu.

To be able to recognize the signs, a set of measurable features of the body that make difference between signs is needed. The body characteristics that make the difference between the signs are the shape of the hand, the angle from each joint of the fingers and wrist, or arm position and trajectory.

To implement such a system, researches have been conducted in two main directions (Vamplew, 1990):

1. Systems that use specialized hardware devices for data acquisition: robotic glove to measure finger and hand joint angles, and various mechanical, optical, magnetic and acoustic devices to detect hand position and trajectory
2. Systems that use image processing and computer vision techniques to detect the characteristics of the hand in images taken with a video or web camera

Vision-Based Proposed Solutions

Detection

Haar-Like features and other shape based methods:

The Adaboost learning algorithms are currently one of the fastest and most accurate approaches for object classification.

(Kölsch & Turk, 2004) Exploited the limitations of hand detection using the Viola-Jones detector. A new rectangle feature type was proposed to have more feature combinations than the basic Haar-like features proposed by Viola and

Jones. As the feature pool for learning contains about 107 features, a highly computational cost is needed for training.

(Ong & Bowden, 2004) Applied the Viola-Jones detector to localize/detect human hands, and then exploited shape context to classify differences between hand posture classes.

Other approaches

(Athitsos & Sclaroff, 2003) Formulated the hand posture recognition problem as an image database index problem. A database contains 26 hand shape prototypes, and each prototype has 86 difference viewpoint images. A probabilistic line matching algorithm was applied to measure the similarity between the test image and the database for recognizing hand posture class and estimating hand pose.

Some others used color segmentations combined with other detection methods of hand in binary images.

Recognition

There have been many machine learning methods used in the field of sign language recognition through the years, the authors have studied many previous works and highlighted the most used methods.

Neural Networks

Neural networks are famous learning models in image recognition applications, different types of NNs were used in sign language recognition systems. Some researchers used Hopfield Neural Networks (Huang & Huang, 1998), others preferred Recurrent Neural Networks (Murakami & Taguchi, 1991) and the majority of other researchers used Feedforward Neural Networks (Vamplew, 1990), (Ma & Khorasani, 2004) and (Geman, Bienenstock, & Doursat, 1992).

Support Vector Machines

Support vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Many studies were conducted with SVMs and gave close results to neural networks, and depending on the features that were selected some people got better results using support vector machines (Rashid, Al-Hamadi, & Michaelis, 2010) (Huang, Hu, & Chang, 2009).

Hidden Markov Models

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. A HMM can be presented as the simplest dynamic Bayesian network. Consequently, they seem ideal for visual recognition of complex structured hand gestures such as are found in sign language (Starner T. , 1995). A study by (Starner & Pentland, 1997) conduct real-time HMM-based system for recognizing sentence level American Sign Language (ASL) without explicitly modeling the fingers. The experiment attains a word accuracy of 92% having 40 word lexicon.

System Design and Implementation

System Requirements

Functional requirements

1- Recognition of Arabic sign language alphabet

Non-functional requirements

1- Compatibility
2- Efficiency
3- Effectiveness
4- Portability
5- Response time
6- Robustness

7- Security

8- Testability

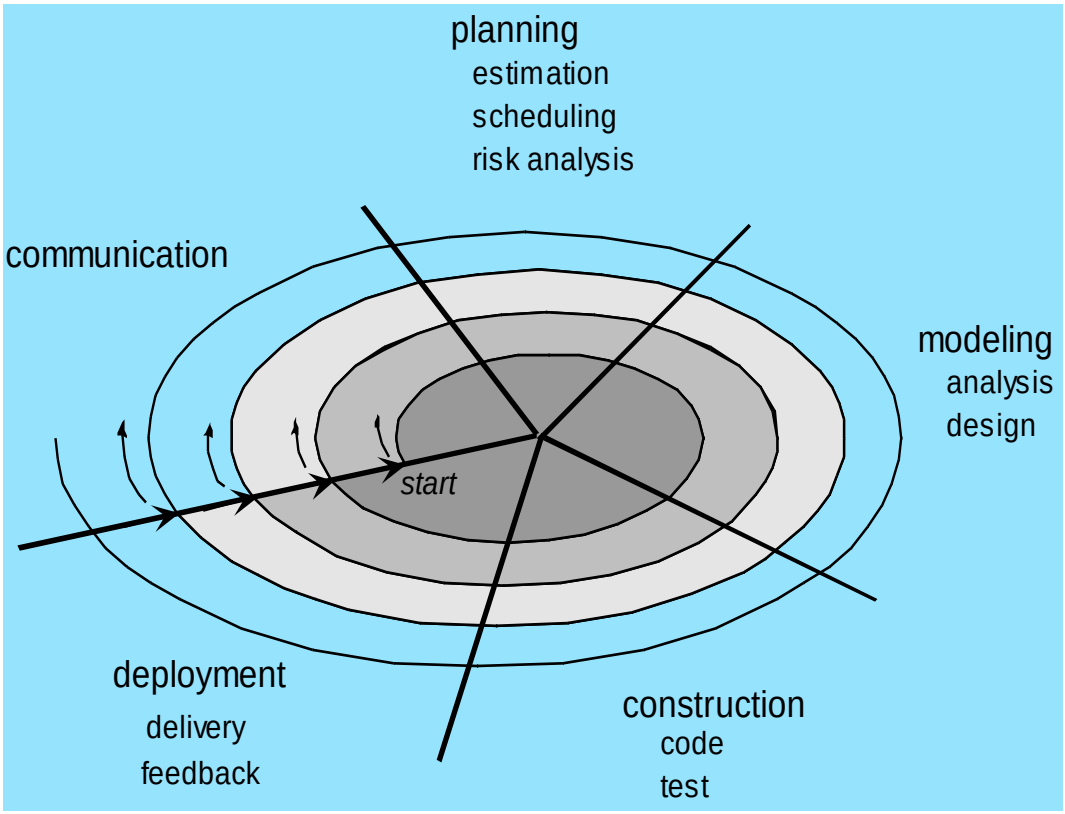
System Analysis

Methodology

In this system spiral model was used. Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis.

It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.



The work went through several iterations each one separated into four phases which are:

1. **Identification**: in the first iteration this phase started with research and studying about the previous work on the same aspect to gather and identify the system requirements. In the next iterations this phase was for determine the next step of the system progress like choosing technics to be used or algorithms to be applied
2. **Design**: this phase includes the conceptual design of the system during the first iteration and the final design in the subsequent spirals.
3. **Build**: Construct phase refers to production of the actual software product at every spiral. In this project in each iteration a version of the application was produced.
4. **Evaluation and Risk Analysis**: Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks.

SWOT ANALYSIS

System Design

The system design basically consists of three parts:

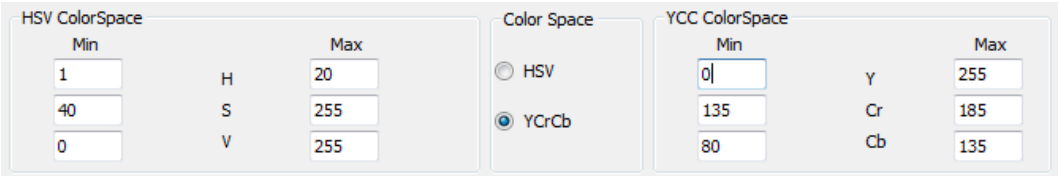
Skin Detection

The first step in skin detection was identifying the human skin color range. The authors of this study have studied different color spaces, mainly, RGB, Normalized RGB, HSV and YCrCb.

After studying the different color spaces the authors concluded that HSV and YCrCb were the least sensitive for illumination variance and identified the skin color range to be:

1- HSV:
   a. $2 < H < 40$
   b. $40 < S < 255$
   c. $0 < V < 255$
2- YCrCb:
   a. $0 < Y < 255$
   b. $135 < Cr < 185$
   c. $80 < Cb < 135$

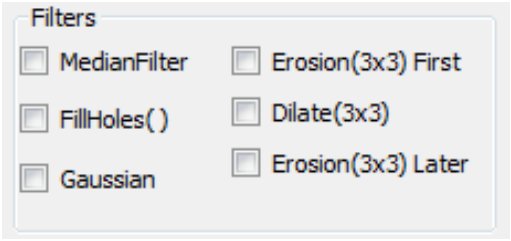
And after studying the various results and by studying the state of the art approaches the study used YCrCb color space as it had the most tolerance for light conditions and gave the best results in detecting the skin and RGB & HSV color spaces were dumped.

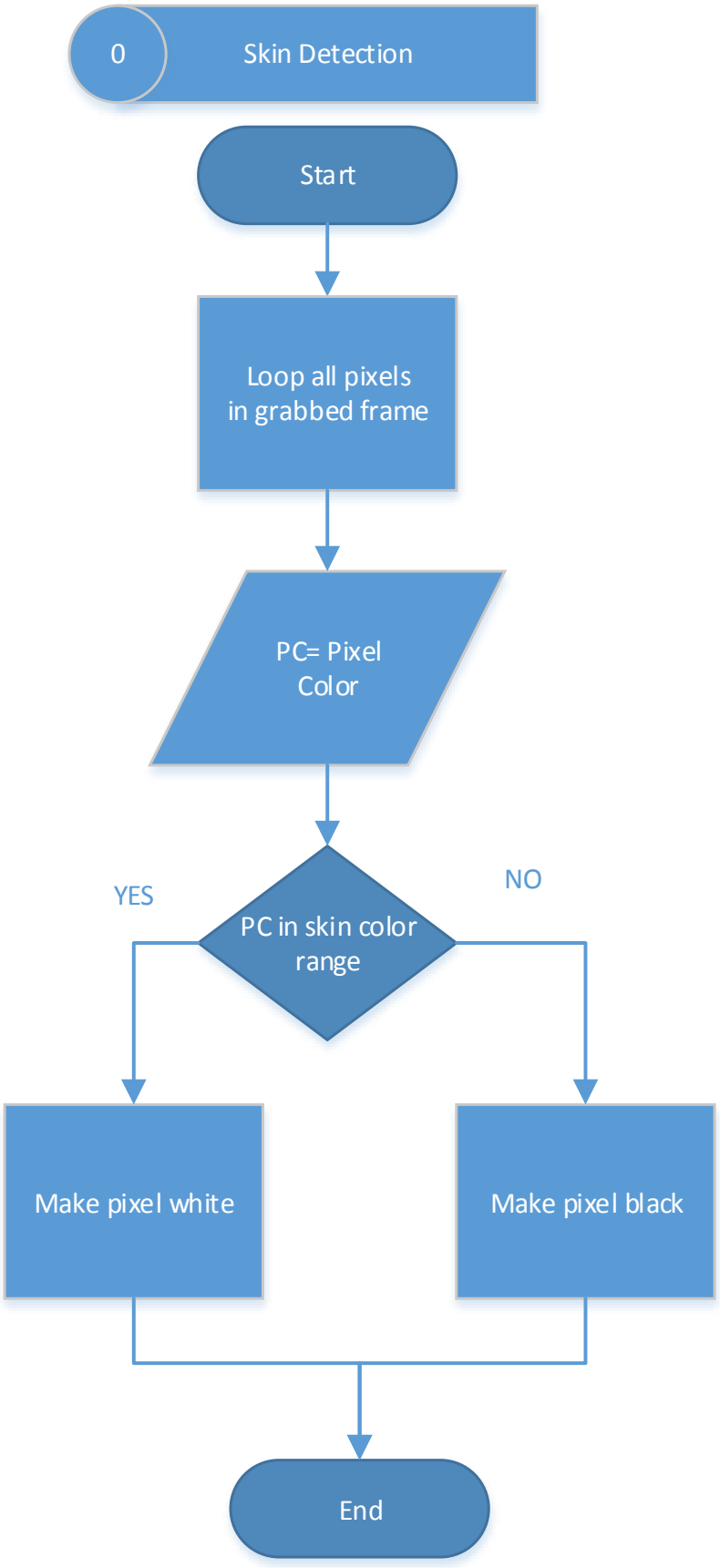


The second step was skin segmentation using the identified skin color range to threshold the image resulting a binary image with the skin colors in white and all other colors in black.

Finally two morphological operations where applied, erosion and dilation, to eliminate the noise.

Additional filters and morphological operations were added if necessary




0
Skin Detection
Start
Loop all pixels in grabbed frame
PC= Pixel Color
YES
NO
PC in skin color range
Make pixel white
Make pixel black
End

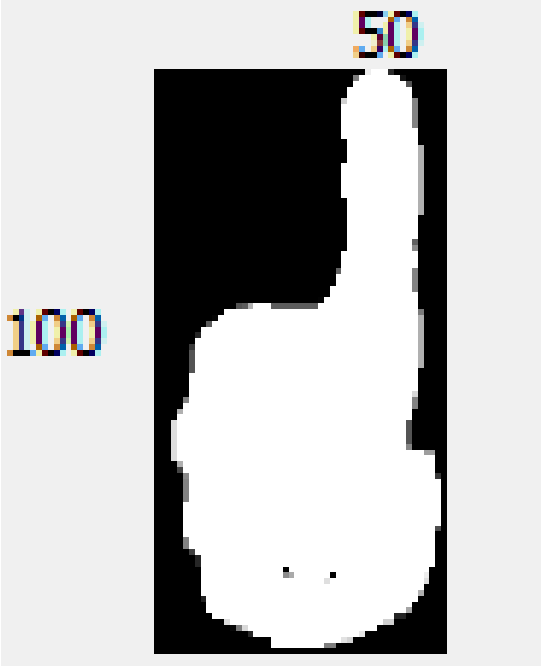
Hand Detection

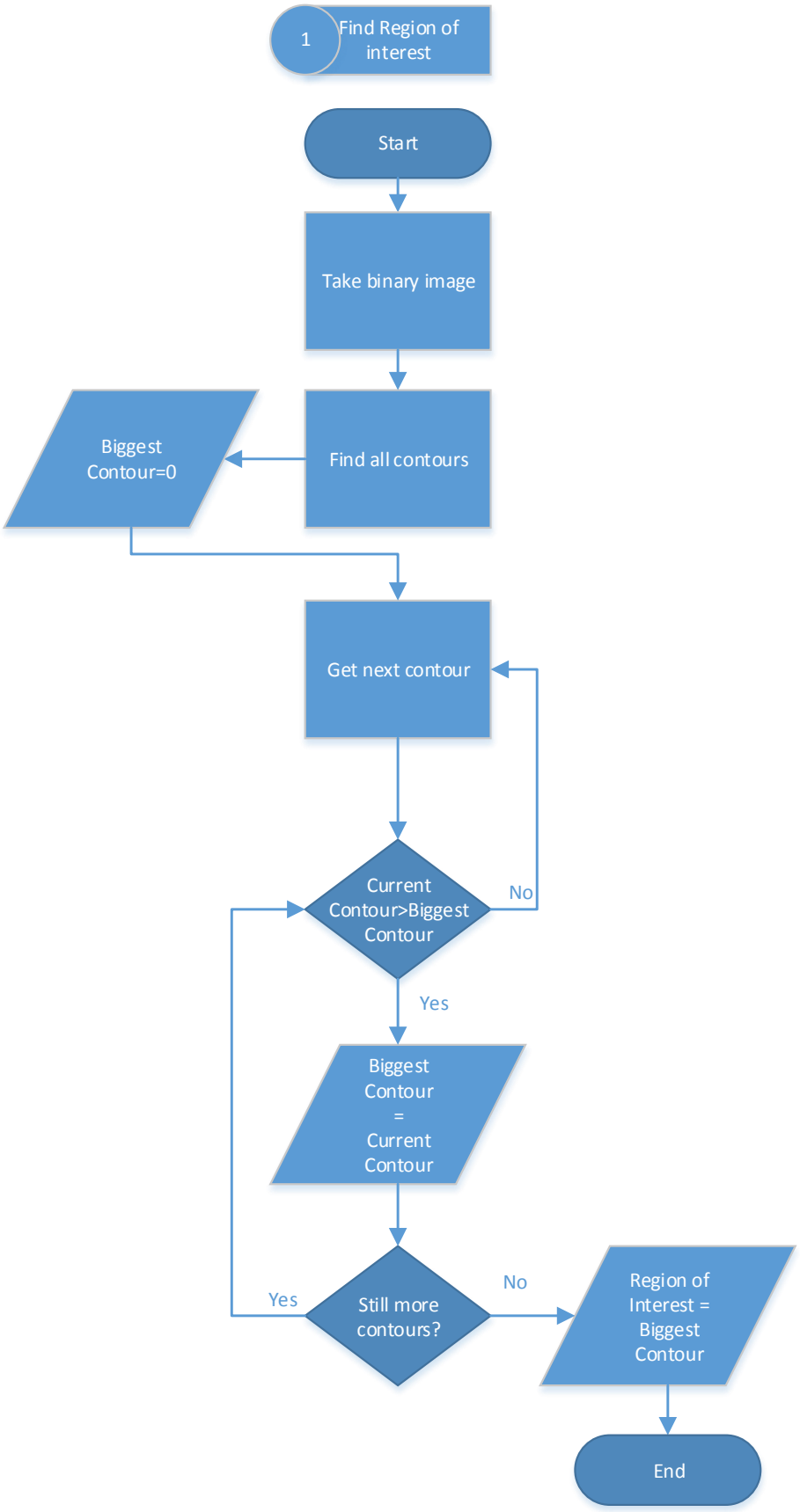
After the previous part produced a binary image showing only skin parts in white, the next step that was conducted was extracting all the contours of the image. Then after finding all the contours a loop was made to extract the biggest contour in the image to find the hand; supposing that the hand had the biggest skin area in the image –closest skin part to the camera.

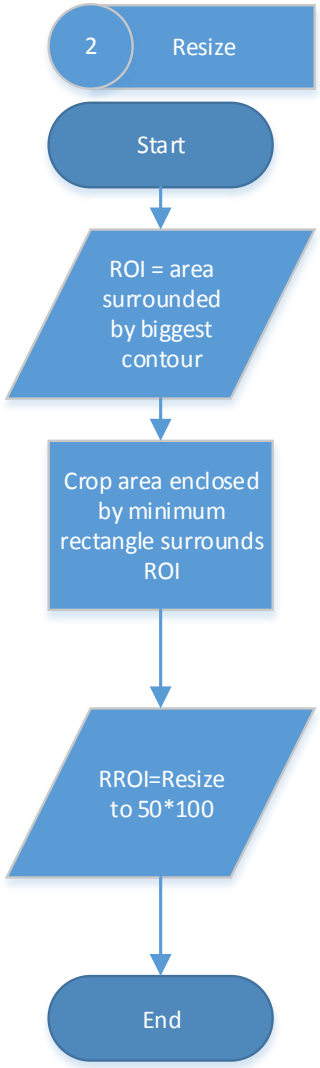
After detecting the hand, the minimum rectangle that enclosed the hand area was extracted. And finally the hand was extracted from the image by extracting the part of the image that was covered by that rectangle.

However, the dimensions of the extracted area varied depending on the distance between the hand and the webcam, to solve that problem the images needed to be resized to fixed dimensions.

After studying the hand geometry the authors have concluded that the dimensions of the hand must be Length > Width, and the authors have finally concluded the ratio length = 2*width. Thus the images were resized to 50*100




1
Find Region of interest
Start
Take binary image
Find all contours
Biggest Contour=0
Get next contour
Current Contour>Biggest Contour
No
Yes
Biggest Contour = Current Contour
Still more contours?
Yes
No
Region of Interest = Biggest Contour
End



Recognition

After studying the state of the art in the static image recognition field (posture recognition) and studying the results of ANN and SVM, the authors chose ANN over SVM because it gave relatively better results with respect to the extracted feature vector they chose.

The neural network system that the authors used was MLP ANN neural network implemented in OpenCV which is a feedforward network. The activation function

that was used is symmetric sigmoid. And as a training method the authors used backpropagation with a 0.2 learning rate and 0.1 momentum.

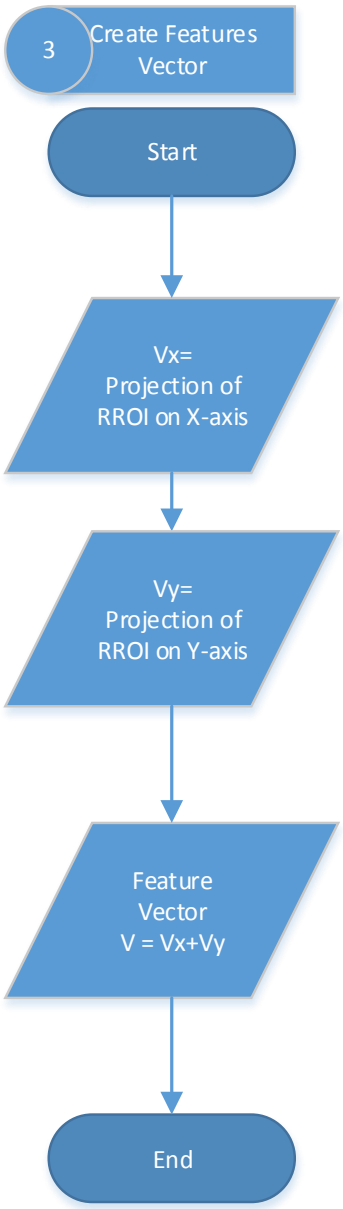
The first step in the recognition part was collecting the dataset. Initially samples were made for the different 28 Arabic letters, 300 samples on average for each letter. The study however, was conducted on the first 7 letters with 30 samples for each letter to train the system with.

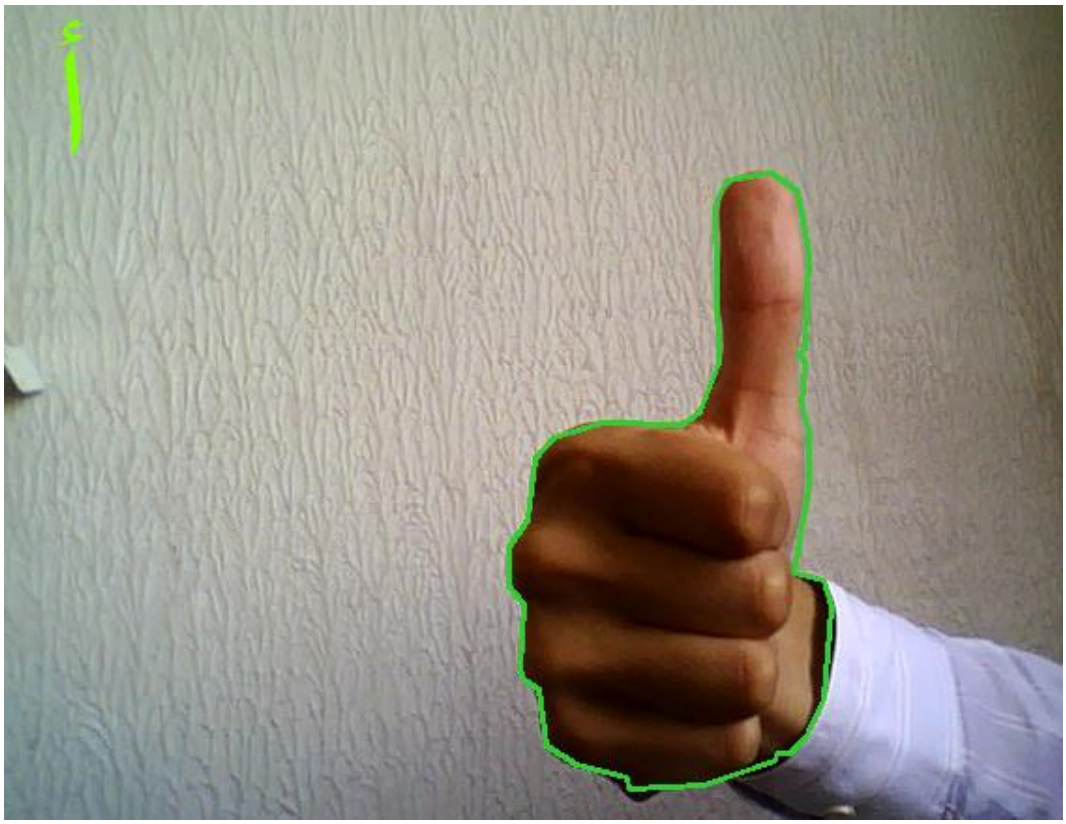
Letter	Desired output
أ	15
ب	45
ت	75
ث	105
ج	135
ح	165
خ	195

The second step was to prepare the input of the system. The system input was a [1*150] feature vector resulting from a 2-D image projection over both axis. The projection on Y (height) resulted a feature vector [1*100] and projection on X (width) resulted another feature vector [1*50], the final [1*150] feature vector was made from the concatenation of both previous vectors.

The final part was training the neural network and evaluating its performance, the test images had the same image processing and input preprocessing that was conducted on the train dataset.

The recognitions results will be included in the results section.


3
Create Features Vector
Start
Vx= Projection of RROI on X-axis
Vy= Projection of RROI on Y-axis
Feature Vector V = Vx+Vy
End



System Implementation

Tools

MS visual studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows superfamily of operating systems, as well as web sites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio supports different programming languages and allows the code editor and debugger to support nearly any programming language, provided a

language-specific service exists. Built-in languages include C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F#.

Visual C#

C# is a programming language that is designed for building a variety of applications that run on the .NET Framework. C# is simple, powerful, type-safe, and object-oriented. The many innovations in C# enable rapid application development while retaining the expressiveness and elegance of C-style languages.

Visual C# is an implementation of the C# language by Microsoft. Visual Studio supports Visual C# with a full-featured code editor, compiler, project templates, designers, code wizards, a powerful and easy-to-use debugger, and other tools. The .NET Framework class library provides access to many operating system services and other useful, well-designed classes that speed up the development cycle significantly.

EmguCV

EmguCV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled and run on Windows, Linux, Mac OS X, iPhone, iPad and Android devices.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and

recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in some countries, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

Configurations and specifications

There are no specific configurations that were needed for this study. However suitable lighting conditions were taken under consideration.

An off-the-shelf webcam was used to conduct this study

Webcam specifications

Microsoft® LifeCam VX-800 with USB 2.0 interface

Seonsor: CMOS VGA sensor technology

Resolution: 0.31 megapixel (640 x 480 pixels)

Testing

After training the system, the system was tested using 100 samples for each letter. The results are shown in the table below.

	أ	ب	ت	ث	ج	ح	خ
أ	93	7	0	0	0	0	0
ب	2	87	11	0	0	0	0
ت	0	10	60	30	0	0	0
ث	0	0	0	100	0	0	0
ج	0	0	0	0	65	25	10
ح	0	0	0	0	10	70	20
خ	0	0	0	0	0	7	93

Results

The authors have evaluated the system using 100 sample for each letter of the 7 letters and the results were as the following:

Letter	Recognition rate
أ	93%
ب	87%
ت	60%
ث	100%
ج	65%
ح	70%
خ	93%

After analyzing the results, the study concluded that the variations of accuracy of the proposed solution results from changing variables like

- The features that the system was based on.
- The neural network topology and parameter.
- The hand detection algorithm.

The system results can be improved by

- Taking other approaches to extract more discriminant features
- Changing the neural network parameters such as layers sizes and learning parameters, and trying different neural network types
- Using a more robust and accurate hand detection algorithm

Problems and limitations

The authors have faced many problems during their study, the most important problems were the light conditions and the presence of other big skin-like color parts in the camera feed.

Although the proposed solution and selected color space is flexible in its illumination conditions, the irregular improper luminance can paralyze the recognition system, such effects cause poor processed images, thus resulting wrong recognition.

The second major problem is the possibility of having skin-like colors within the camera view that cover bigger continuous binary area than the hand does, making it the better candidate for having the biggest contour and causing improper hand detection and thus wrong recognition

The main limitation of the system is the hand detection part where the authors used a naïve and simple part which is the biggest detected contour.

Conclusion and future work

In this study, we have provided a way to facilitate the communication between deaf and hearing people using a webcam and a neural network approach. As shown by the experimental results, the solution we have proposed in this study can be efficiently used in fingerspelling situations.

The main advantage of our approach over the other attempts is the fact that it requires no additional hardware equipment or special clothes to recognize the signs. The accuracy of the system could be also increased if a more robust skin detection algorithm will be used.

Future work may also be done in order to use another classification model in the supervised learning scenario, such as support vector machines (Steinwart & Christmann, 2008) or Bayesian Learning (Mitchell, 1997).

Future vision & applications

The authors of this study envisioned many probabilities, some of them are:

- A system that detects two hands without being affected by the existence of other body parts like the face
- A system that recognizes dynamic sequential signs (gesture recognition) using another learning scenarios, such as Hidden Markov Models (HMM) and language syntax probabilistic dictionary.
- A system that is robust enough to be used in official and private offices and organizations to facilitate deaf people in their daily life communication difficulties.
- An educational sign language learning system that facilitates learning sign language by evaluating the user's sign and comparing it to the system and correcting the incorrect signs for the user.
- A sign-to-speech system that uses the current sign-to-text system as an intermediate step.

- A mobile application that uses sign-to-speech to allow the deaf individual to communicate with hearing parties by translating the deaf party sign language in front of the camera to voice that is transmitted to the hearing party.

References

Athitsos, V., & Sclaroff, S. (2003). Estimating 3D hand pose from a cluttered image. *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference* (pp. II-432-9 vol. 2). IEEE.

Cutler, R., & Turk, M. (1998). View-based interpretation of real-time optical flow for gesture recognition. *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)* (pp. 416-416). IEEE Computer Society.

Drew, P. (2004, December 22). *Research on Sign Language Recognition*. Retrieved June 19, 2014, from http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, 1-58.

Huang, C., & Huang, W. (1998). Sign language recognition using model-based tracking and a 3D Hopfield neural network. *Machine vision and applications*, 292-307.

Huang, D.-Y., Hu, W.-C., & Chang, S.-H. (2009). Vision-based hand gesture recognition using PCA+ Gabor filters and SVM. *Vision-based hand gesture recognition using PCA+ Gabor filters and SVM* (pp. 1-4). IEEE.

Kölsch, M., & Turk, M. (2004). Robust Hand Detection. *FGR*, (pp. 614-619).

Ma, L., & Khorasani, K. (2004). Facial expression recognition using constructive feedforward neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 1588-1595.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Murakami, K., & Taguchi, H. (1991). Gesture recognition using recurrent neural networks. *SIGCHI conference on Human factors in computing systems*, (pp. 237-242).

Ong, E.-J., & Bowden, R. (2004). A boosted classifier tree for hand shape detection. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference* (pp. 889-894). IEEE.

Rashid, O., Al-Hamadi, A., & Michaelis, B. (2010). Utilizing invariant descriptors for finger spelling American sign language using SVM. In *Advances in Visual Computing* (pp. 253-263). Springer.

Starner, T. (1995). *Visual Recognition of American Sign Language Using Hidden Markov Models.* DTIC Document.

Starner, T., & Pentland, A. (1997). Real-time american sign language recognition from video using hidden markov models. In *Motion-Based Recognition* (pp. 227-243). Springer.

Steinwart, I., & Christmann, A. (2008). *Support vector machines.* Springer.

Vamplew, P. W. (1990). *Recognition of Sign Language Using Neural Networks.* Flinders University of South Australia.

Appendices

Appendix A

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.Util;
using System.IO;
using System.Diagnostics;
using System.Threading;
using System.Media;
using Emgu.CV.UI;
using Emgu.CV.CvEnum;
using Emgu.CV.ML;
using Emgu.CV.ML.Structure;



namespace Junior_Alpha
{
    public partial class Form1 : Form
    {

        private Capture frameGrabber; //instance to connect with the camera

        #region colorVariables
        //RGB
        private int MinRed, MinGreen, MinBlue;
        private int MaxRed, MaxGreen, MaxBlue;
        //HSV
        Hsv HSV_min, HSV_max;
        private double MinHue, MinSat, MinVal;
        private double MaxHue, MaxSat, MaxVal;
        //YCrCb
        Ycc YCrCb_min, YCrCb_max;
        private double MinY_var, MinCr_var, MinCb_var;
        private double MaxY_var, MaxCr_var, MaxCb_var;
        #endregion


        Image<Bgr, Byte> currentFrame;
        Image<Hsv, Byte> currentFrameHSV;
        Image<Ycc, Byte> currentFrameYCC;
        Image<Gray, Byte> skinFiltered, filteredHand;
        StructuringElementEx rect_12, rect_6;

        MCvBox2D box;
```

```
        Graphics g;
        Font font;
        SolidBrush solidBrush;


        int counter;
        int framNameCounter;
        String frameName;

        static int resizeWidth = 50;
        static int resizeHeight = 100;

        //needs to be changed to resizeHeight+resizeWidth if projecting on 2
axis
        static int size = resizeWidth + resizeHeight;
        //static int size = resizeHeight*resizeWidth;

        static int letterNum = 7;
        static int sampleNum = 30;
        static int trainSamplesCount = letterNum * sampleNum;

        ANN_MLP network;
        Matrix<int> layerSize = new Matrix<int>(new int[] { 150, 50, 1 });
        MCvANN_MLP_TrainParams parameter = new MCvANN_MLP_TrainParams();
        Matrix<float> traindataMatrix = new Matrix<float>(trainSamplesCount,
size);
        Matrix<float> responseMatrix = new Matrix<float>(trainSamplesCount,
1);
        Matrix<float> sample = new Matrix<float>(1, size);
        Matrix<float> output = new Matrix<float>(1, 1);



        Bitmap[] Sample = new Bitmap[trainSamplesCount];
        Bitmap TestSamplee;

        string letter;

        public Form1()
        {
            InitializeComponent();

            try
            {
                #region colorDefenition

                //calculateRGBColorThreshold();
                calculateHSVColorThreshold();
                calculateYCCColorThreshold();

                HSV_min = new Hsv(MinHue, MinSat, MinVal);
                HSV_max = new Hsv(MaxHue, MaxSat, MaxVal);

                YCrCb_min = new Ycc(MinY_var, MinCr_var, MinCb_var);
                YCrCb_max = new Ycc(MaxY_var, MaxCr_var, MaxCb_var);

                #endregion

                //kernels for Erode & Dilate filters
```

```
                rect_12 = new StructuringElementEx(12, 12, 6, 6,
Emgu.CV.CvEnum.CV_ELEMENT_SHAPE.CV_SHAPE_RECT);
                rect_6 = new StructuringElementEx(6, 6, 3, 3,
Emgu.CV.CvEnum.CV_ELEMENT_SHAPE.CV_SHAPE_RECT);

                //font for drawing recognized letter on frame
                font = new Font("Arial", 72);
                solidBrush = new SolidBrush(Color.Chartreuse);

                //contour box
                box = new MCvBox2D();

                //frameSave variables
                counter = 0;
                framNameCounter = 1;

                // initialize capture instance

                //read from video file
                //frameGrabber = new Capture(@".\..\..\M2U00253.MPG");

                //read from camera id #0
                frameGrabber = new Capture(0);

                // append event handler
                Application.Idle += new EventHandler(processFrame);

                Load_DataSet();
                BuildNeuralNetwork();
                trainNetwork();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }


        private void Load_DataSet()
        {
            //read samples from images
            for (int i = 0; i < trainSamplesCount; i++)
            {
                Sample[i] = new Bitmap("TRAIN SET\\" + i + ".bmp");

                for (int y = 0; y < resizeHeight; y++)
                {
                    for (int x = 0; x < resizeWidth; x++)
                    {
                        Color pixelColor = Sample[i].GetPixel(x, y);
                        if (pixelColor.R != 0)
                        {
                            traindataMatrix[i, y] = traindataMatrix[i, y] +
1;
                            traindataMatrix[i, resizeHeight + x] =
traindataMatrix[i, resizeHeight + x] + 1;
                        }
```

```
                    }
                }

                Console.WriteLine("Loaded pattern: " + i);

                //set expected output to
                //15 if sample = 0:30  (pattern #1)
                //and so on...
                //needs to be checked if choosing smaller values can result
better evaluation/training
                if (i >= 0 && i < 30)
                    responseMatrix[i, 0] = 15;
                if (i >= 30 && i < 60)
                    responseMatrix[i, 0] = 45;
                if (i >= 60 && i < 90)
                    responseMatrix[i, 0] = 75;
                if (i >= 90 && i < 120)
                    responseMatrix[i, 0] = 105;
                if (i >= 120 && i < 150)
                    responseMatrix[i, 0] = 135;
                if (i >= 150 && i < 180)
                    responseMatrix[i, 0] = 165;
                if (i >= 180 && i < 210)
                    responseMatrix[i, 0] = 195;


            }


        }

        //build a new neural network ANN_MLP object and set the learning
parameter object
        private void BuildNeuralNetwork()
        {
            //alpha and beta = 0.1; activation function parameters
            network = new ANN_MLP(layerSize,
Emgu.CV.ML.MlEnum.ANN_MLP_ACTIVATION_FUNCTION.SIGMOID_SYM, 1.0, 1.0);


            //200 iteration; error = 0.0e-6
            parameter.term_crit = new MCvTermCriteria(2000, 0.0e-6);
            //backpropagation training
            parameter.train_method =
Emgu.CV.ML.MlEnum.ANN_MLP_TRAIN_METHOD.BACKPROP;
            //learning rate
            parameter.bp_dw_scale = 0.2;
            //momentum
            parameter.bp_moment_scale = 0.1;

        }

        //train the network using the input/expectedOutput matrices
        //parameter is the object that sets the learning method and
variables
        private void trainNetwork()
        {
            network.Train(traindataMatrix, responseMatrix, null, parameter,
Emgu.CV.ML.MlEnum.ANN_MLP_TRAINING_FLAG.DEFAULT);
```

```
            Console.WriteLine("done");
        }

        //a function to extract vector from the current frame
        //and pass it to evaluation in the neural network
        private void predictCurrentFrame(Image<Gray, Byte> image)
        {
            sample = new Matrix<float>(1, size);
            TestSamplee = image.Bitmap;
            for (int y = 0; y < resizeHeight; y++)
            {
                for (int x = 0; x < resizeWidth; x++)
                {
                    Color pixelColor = TestSamplee.GetPixel(x, y);
                    if (pixelColor.R != 0)
                    {
                        sample[0, y] = sample[0, y] + 1;
                        sample[0, resizeHeight + x] = sample[0, resizeHeight
+ x] + 1;
                    }
                }
            }
            network.Predict(sample, output);

            DrawLetter(output[0, 0]);
        }


        //a function to grab the frames using timer ticks (10ms)
        //replaced in the 2.8 version with
        //Application.Idle += new EventHandler(processFrame);
        private void timer1_Tick(object sender, EventArgs e)
        {
            processFrame(sender, e);
        }

        private void processFrame(object sender, EventArgs e)
        {
            //get the frame from the camera handler
            currentFrame = frameGrabber.QueryFrame();

            if (currentFrame != null)
            {

                //counter to save the current frame with its name set to its
number
                counter++;

                #region imageProcessing
                currentFrameHSV = currentFrame.Convert<Hsv, Byte>();
                currentFrameYCC = currentFrame.Convert<Ycc, Byte>();

                if (radioButtonHSV.Checked)
                    skinFiltered = currentFrameHSV.InRange(HSV_min,
HSV_max);
                else
                    skinFiltered = currentFrameYCC.InRange(YCrCb_min,
YCrCb_max);
```

```
                    CvInvoke.cvErode(skinFiltered, skinFiltered, rect_12, 1);
                    CvInvoke.cvDilate(skinFiltered, skinFiltered, rect_6, 2);
                    //CvInvoke.cvErode(filtered, filtered, rect_6, 2);
                    //CvInvoke.cvDilate(filtered, filtered, rect_6, 2);

                    ExtractContour(skinFiltered);
                    #endregion

                    //used as necessary from the GUI checkboxes
                    #region GUI FILTERS
                    if (checkBox3.Checked)
                        skinFiltered._SmoothGaussian(3);

                    if (checkBox1.Checked)
                        skinFiltered = skinFiltered.Copy().SmoothMedian(3);

                    if (checkBox2.Checked)
                        skinFiltered = FillHoles(skinFiltered.Copy());

                    if (checkBox4.Checked)
                        skinFiltered._Erode(1);

                    if (checkBox5.Checked)
                        skinFiltered._Dilate(1);

                    if (checkBox6.Checked)
                        skinFiltered._Erode(1);

                    #endregion

                    //save current frame if checkbox is checked
                    //save detected hand in bottom right imagebox [50*100]
                    #region DATASET COLLECT
                    if (checkBox_Skin.Checked)
                    {
                        if (counter % 10 == 0)
                        {
                            frameName = @".\data_set\image_" +
letterSelector.Text + "_";
                            frameName = frameName + framNameCounter + ".bmp";
                            filteredHand.Save(@frameName);
                            framNameCounter++;
                        }
                    }
                    #endregion

                    //make a copy of the binary image
                    filteredHand = skinFiltered.Copy();

                    //set the image to the cropped part of the binary image
                    //crop the rectangle that encloses the hand contour
                    filteredHand.ROI = box.MinAreaRect();

                    //resize the image using the defined variables, 50*100
                    filteredHand = filteredHand.Resize(resizeWidth,
resizeHeight, Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR);
```

```
                //debug to confirm detected hand image height and width
[50*100]
                ROIH.Text = filteredHand.Height + "";
                ROIW.Text = filteredHand.Width + "";

                //show final results on their corresponding imageboxes
                pictureBoxCurrentFrame.Image = currentFrame;
                pictureBoxSkinFiltered.Image = skinFiltered;
                imageBoxTracked.Image = filteredHand;

                //pass each frame to the neural network and evaluate it
                predictCurrentFrame(filteredHand);
            }
        }

        //a function to extract all contours in the image and then return
the biggest contour
        //hand is supposed to be the biggest part of skin in the image
        //keep hand closer to camera than other parts of body like face
        private void ExtractContour(Image<Gray, Byte> skin)
        {
            using (MemStorage storage = new MemStorage())
            {

                Contour<Point> contours =
skin.FindContours(Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_SIMPLE,
Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_LIST, storage);

                Contour<Point> biggestContour = null;


                //FIND BIGGEST CONTOUR LOOP

                Double Result1 = 0;
                Double Result2 = 0;
                while (contours != null)
                {
                    Result1 = contours.Area;
                    if (Result1 > Result2)
                    {
                        Result2 = Result1;
                        biggestContour = contours;
                    }
                    contours = contours.HNext;
                }

                if (biggestContour != null)
                {

                    Contour<Point> currentContour =
biggestContour.ApproxPoly(biggestContour.Perimeter * 0.0025, storage);
                    //DRAW CONTOUR
                    currentFrame.Draw(currentContour, new
Bgr(Color.LimeGreen), 2);

                    biggestContour = currentContour;
                    box = biggestContour.GetMinAreaRect();//minimumRectangle
that holds the contours
                }
```

```
            }
        }

        //a function to redraw the frame with the recognized letter on top
left
        private void DrawLetter(float sample)
        {
            if (sample >= 0 && sample < 30)
                letter = "أ";
            if (sample >= 30 && sample < 60)
                letter = "ب";
            if (sample >= 60 && sample < 90)
                letter = "ت";
            if (sample >= 90 && sample < 120)
                letter = "ث";
            if (sample >= 120 && sample < 150)
                letter = "ج";
            if (sample >= 150 && sample < 180)
                letter = "ح";
            if (sample >= 180 && sample < 210)
                letter = "خ";

            g = Graphics.FromImage(pictureBoxCurrentFrame.Image.Bitmap);
            g.DrawString(letter, font, solidBrush, 10, 10);
            //g.DrawString(sample.ToString(), font, solidBrush, 10, 10);
        }

        //a function to fill any holes inside closed contours
        private Image<Gray, byte> FillHoles(Image<Gray, byte> image)
        {
            var resultImage = image.CopyBlank();
            Gray gray = new Gray(255);
            using (var mem = new MemStorage())
            {
                for (var contour = image.FindContours(
                    CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_SIMPLE,
                    RETR_TYPE.CV_RETR_CCOMP,
                    mem); contour != null; contour = contour.HNext)
                {
                    resultImage.Draw(contour, gray, -1);
                }
            }

            return resultImage;
        }

    }
}
```