Design and Implementation of a Virtual Smart Board

Abdullah S. Mahmod

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
September 2012
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Mohammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Mohammed Salamah
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Mohammed Salamah _____
2. Asst. Prof. Dr. Adnan Acan _____
3. Asst. Prof. Dr. Ahmet Ünveren _____

ABSTRACT

The Virtual Smart Board implementation is an issue of artificial intelligence to implement most of the functions of the real Smart Board. In this study the functions implemented are mouse actions (left & right click). These actions depend on the hand's position compared with the site of view and the gesture of the hand. Hand tracking and gesture recognition are implemented in three different approaches. The first approach depends on a single camera where the hand is tracked in two dimensional space using a neural network and hand gesture recognition by another network. The second approach depends on two cameras which create a three dimensional space in order to store the position of the view surface and compare the position of skin spots with it. The third approach depends on infrared sensor to detect the infrared LED marker fixed on a glove with switches and a transmitter in order to interact with human action.

Keywords: Smart Board, Skin Detection, Stereo Vision

ÖZ

Sanal Akıllı Tahta uygulaması gerçek Akıllı Tahta'nın içerdiği fonksiyon ve özelliklerin birçoğunu uygulamayı kapsar. Bu araştırmada uygulanan fonksiyonlar fare hareketleridir (sağ ve sol tuşlar). Bu tür hareketler izlenen sayfa ve el hareketiyle oluşan elin pozisyonuna bağlı olmaktadır. El takibi ve hareketi tanımlanması üç farklı yaklaşımla uygulanmaktadır. Birinci yaklaşım el takibini iki boyutlu olarak sinir ağı kullanarak ve el hareketlerini ise farklı bir ağ kullanarak tanımlayan bir adet kameraya bağlıdır. İkinci yaklaşım ise görünüş pozisyonunu kaydedip bununla deri noktalarını karşılaştıran üç boyutlu alan yaratan iki kamerayla desteklenmektedir. Üçüncü yaklaşım insan hareketleriyle etkileşime geçmek amacıyla bir eldivenin üzerine monte edilmiş fiş ve verici cihazlarla kızılötesi LED işaretleyicisini tanımlayan kızılötesi bir alıcıya bağlı olmaktadır.

Anahtar Kelimeler:

ACKNOWLEDGMENT

I would like to thank Assoc. Prof. Dr. Mohammed Salamah for his continuous support and guidance in the preparation of this study. Without his invaluable supervision, all my efforts could have been short-sighted.

Assoc. Prof. Dr. Mohammed Salamah, Chairman of the Department of Computer Engineering, Eastern Mediterranean University has helped me with various issues during this thesis therefore I am truly grateful to him. I owe a great deal of gratitude and thanks to my family who have permitted me to travel from Iraq to Cyprus. They have supported me all through my studies and as a result I would like to dedicate this study to them. I would also like to thank a number of friends who have provided me with moral support and friendship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1. Introduction

The virtual smart board is an artificial intelligence system which relies on a visual device to detect human actions and interpret it to a suitable computer action where the computer video is then displayed by a projector. The Intelligent system connects the human action with a computer action. This study aims to achieve the mouse actions left click, right click and the positioning of the cursor and analyses three approaches to implement virtual smart board system.

The first approach works with a single camera, which takes the spot of the video displayed by the projector. A frame is captured fifteen times every second and every frame is processed to utilize skin pixels. As a result, the skin classification process creates a binary image representing skin spots in the original frame. This binary image is labeled using labeling algorithm by connected components. Every connected label will be cropped to a new resized image and to a constant scale. The resulted images are used as a resource for the neural network which is trained to decide whether the input image is a hand or something else. If the neural network decides that the input image is a hand then the image can be used as an input data to another neural network to categorize the

gesture of the hand. Finally, the computer mouse cursor can be moved to the calibrated place of the hand and takes the action according to the gesture of the hand.

The second approach takes on a stereo vision technique which consists of at least two cameras calibrated to sense a three-dimensional range. Two synchronized frames are taken from the stereo system in a sequence with a constant time interval. Both of the frames processed light the skin pixels and dim the non-skin pixels and result in two binary images which represent the skin spots in the original frames. A calculation is conducted on the resulted binary images which calculates the site of all the skin spots in the three-dimensional range by checking whether the skin touches the site where the video is displayed by the projector. If there is an engagement on it, then the spot of skin, which is in touch with the site will be cropped and resized to a new image. The resulted image will be considered as an input to the neural network which then decides whether it is a hand or not and translates it to the gesture of the hand to take the appropriate action.

The third approach depends on an infrared two-dimensional detector, a wireless transmitter-receiver and a smart glove. An infrared marker is fixed on the smart glove which is detected by the sensor for the positioning of the glove. The sensor detects the infrared marker and sends the information to the computer. The computer then calibrates the position to set the mouse cursor position. The glove also contains switches for right and left clicks. These switches are linked to a micro controller which is connected to a wireless transmitter which transmits the data to the computer.

1.2. Background

1.2.1. Smart board

SMART Technologies Corporation invented the Smart Board device. The Smart board is an interactional projection display that projects the computer's video output. It handles the integration of the interactive whiteboard, a computer, a projector, and the Smart Board computer software. The attached screen is functionable as a large touch screen. This touch screen permits the lecturer to control content by his hand. Some functions which could be handled by the Smart Board are rolling and mouse actions handled the same way as on a computer. Other functions which could be handled are drawing with digital ink pens which do not actually use ink. These pens use optical markers to draw the color on the white board [1].

1.2.2. Hand tracking

Hand tracking is a wide area of research in the image based system community. One of the main reasons is to provide a reliable human interactive system. One of the most resourceful tracking systems was focused on segmented hand motion. In this system the hand could be detected by obtaining point and lines characteristics from grayscale images. The system has difficulties in tracking the existence of a complex background, and it needs a hand-operated initialization before tracing could begin [2].

From a fundamental interaction aspect, the majority of the hand detection task was focused on two dimensional coordinates. The System was focused on tracking a finger across two-dimensional area using a single camera. The system's aim was to control the

GUI without using traditional computer input. Finger identification was carried out using Kalman filtering [3].

1.2.3. Stereo vision

A stereo vision system uses two or more cameras, which captures images of an area from two or more views to create a 3-D space. Distance calculation system requires an increased based line to provide rational deepness resolution [4].

Chapter2

THE VIRTUAL SMART BOARD

2.1. The Two Dimensional Vision Approach

This approach is dependent on a single camera which detects the movement of the hand and tracks the position of it.

2.1.1. Data Acquiring and Representation

Data is acquired using a camera. The Camera is a photo sensor array that senses the light reflected from an element in the detected range. The sensor converts the light in to electric wave and sends it to the computer. The computer receives the wave and represents it in three layers. Every layer is a two dimensional array sized to the resolution of the camera. These layers are red mask, green mask and blue mask. These masks together form a single frame. A frame is acquired every constant time interval to form a video from image sequences. Figure 1. Shows an image acquired from a webcam in RGB color space [5].

Figure 1. Image in RGB Color Space

2.1.2. Preprocess and Noise Elimination

Every frame is processed separately and is represented in RGB color space and later converted into HSV color space. A trained Bayes classifier is used to classify pixels to skin or non-skin pixels according to two feature vectors; hue and saturation of the HSV color space. This classification produces a binary image where the white pixels are skin, and the black pixels are not. The Morphological process could be performed on the binary image to take the interested regions of the image and use it as an input for the neural network. Figure 2 shows a flowchart of the main preprocess steps.

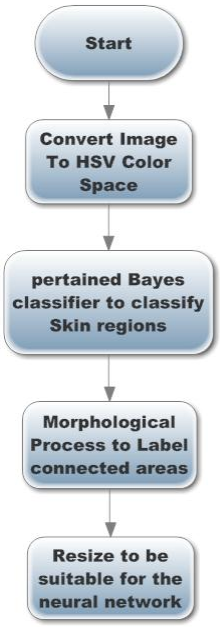

Figure 2.Preprocess Main Steps

2.1.2.1. HSV color space

HSV is a three-dimensional color space representation. It represents colors in hexacone shape. Hue is the angle around the vertical axis. Saturation represents the depth or the clearness of the color. Value or intensity is represented by the vertical axis as shown in Figure 3 [6].

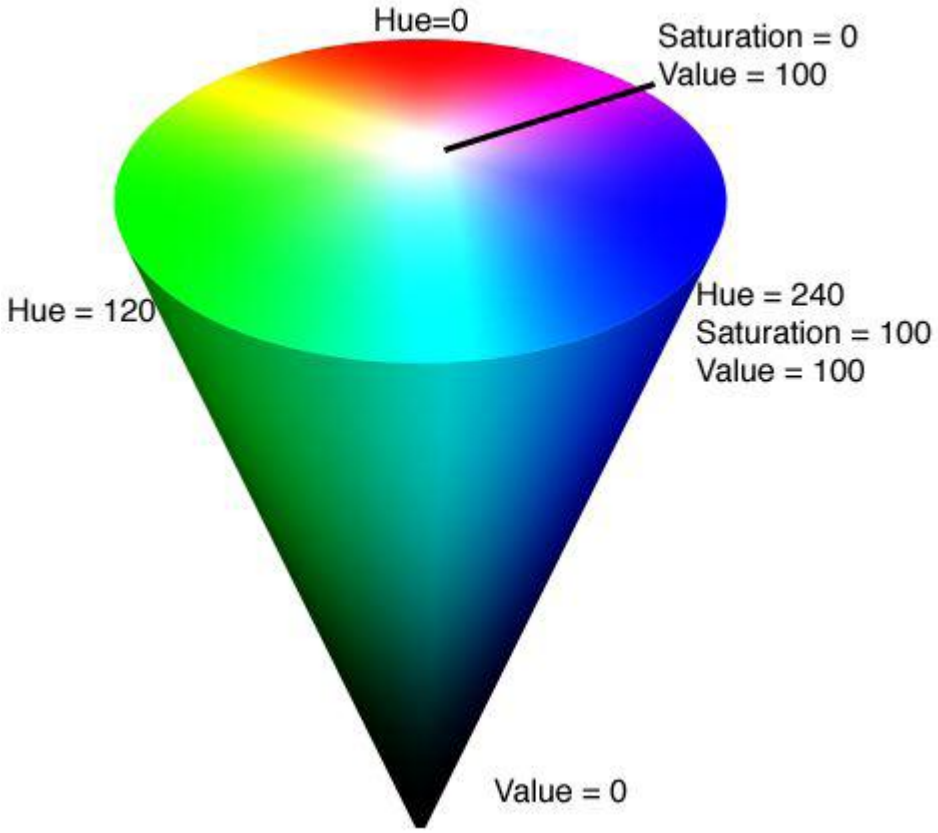

Figure 3.HSV color space representation

Hue could be computed by operating the following equation:

$M = max(R, G, B)$[7]

$m = min(R, G, B)$[7]

$C = M - m$[7]

$$H' = \begin{cases} \frac{G-B}{C} mod 6 & if\ M = R \\ \frac{B-R}{C} + 2 & if\ M = G \\ \frac{R-G}{C} + 4 & if\ M = B \\ undefined & if\ C = 0 \end{cases}$$ [7]

$H = 60° \times H'$[7]

Saturation could be calculated as the following:

$$S = \begin{cases} 0 & if\ C = 0 \\ \frac{C}{V} & otherwise \end{cases}$$[7]

Intensity value could be calculated as the following:

$V = M$[7]

The resulted value of these equations is HSV domain. The purpose of using this domain is that it is easier to segment the colors and analyze them. Figure 4 shows an image in HSV color space.

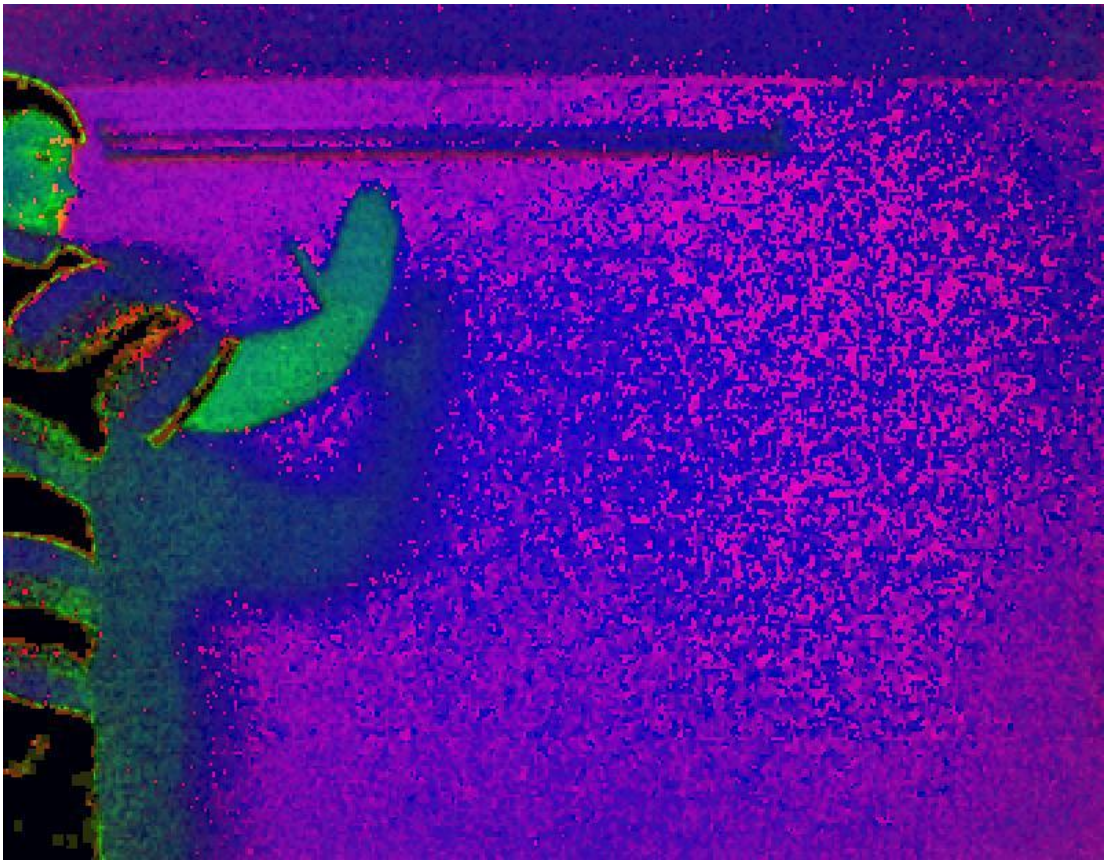

Figure 4. Image In HSV Color Space

2.1.2.2.Bayesian theory

Bayesian decision taking theory is a basic statistical approach to the issue of pattern classification. This approach is dependent on the quantifying of comparisons between different classification decisions using probability[8].

In this project the theory will be used to decide whether the related pixel is skin or not[9].

To make a decision, the classifier should have a training data for both classes. This data is called training vectors. In this project, the training vectors are two vectors: hue and saturation. The training data is obtained by an image which contains skin regions. These

regions are cropped and considered as a training data for the class skin. Other regions are considered as training data for the non-skin class. In the two-dimensional cases, the Gaussian density function for both classes has the following form:

$$p(\boldsymbol{x}/\omega_j) = \frac{1}{2\pi|C_j|^{1/2}} e^{\frac{-1}{2}(\boldsymbol{x}-\boldsymbol{m_j})^T C_j^{-1}(x-m_j)} \text{ [8]}$$

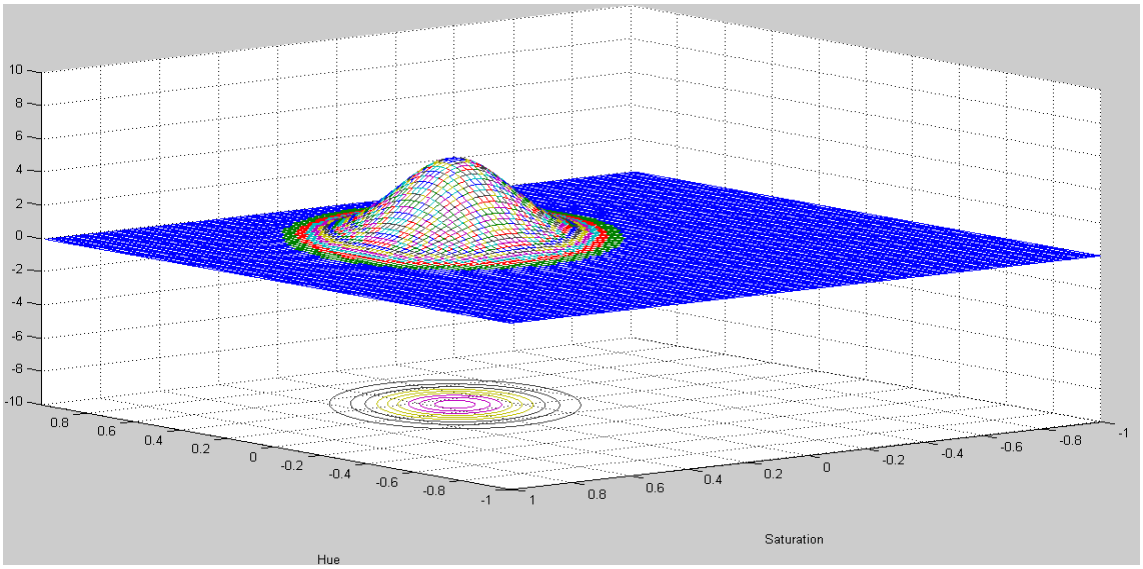

Figure 5. Normal Distribution Of Skin Class

Figure 5shows the normal distribution of skin class, where both density functions are determined by the standard deviation vector $\boldsymbol{m_j}$and covariance matrix, $\boldsymbol{C_j}$, which could be defined as:

$$\boldsymbol{m_j} = \frac{1}{N_j}\sum_{\boldsymbol{x}\in\boldsymbol{\omega_j}} \boldsymbol{x} \text{[8]}$$

and

$$\boldsymbol{C}_j = \frac{1}{N_j}\sum_{\boldsymbol{x}\in\omega_j} \boldsymbol{x}\boldsymbol{x}^{\boldsymbol{T}} - \boldsymbol{m}_j\boldsymbol{m}_j^T$$ [8]

Where $N_j$is the length of the training vector of the class ω_j. These equations are used to estimate the probability density function of the two classes. To obtain the classification process, a decision boundary is required. The numerical arrangement of the decision boundary function for the class ω_j is as the following.[8]

$$d_j(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{m}_j - \frac{1}{2}\boldsymbol{m}_j^T\boldsymbol{m}_j \ldots\ldots\ldots\ldots\ldots j = 1,2.$$[8]

This function is the minimum distance classifier, and by subtracting both the decision function for both classes; the result is the decision boundary function. Figure 6 shows the decision boundary of skin distribution.

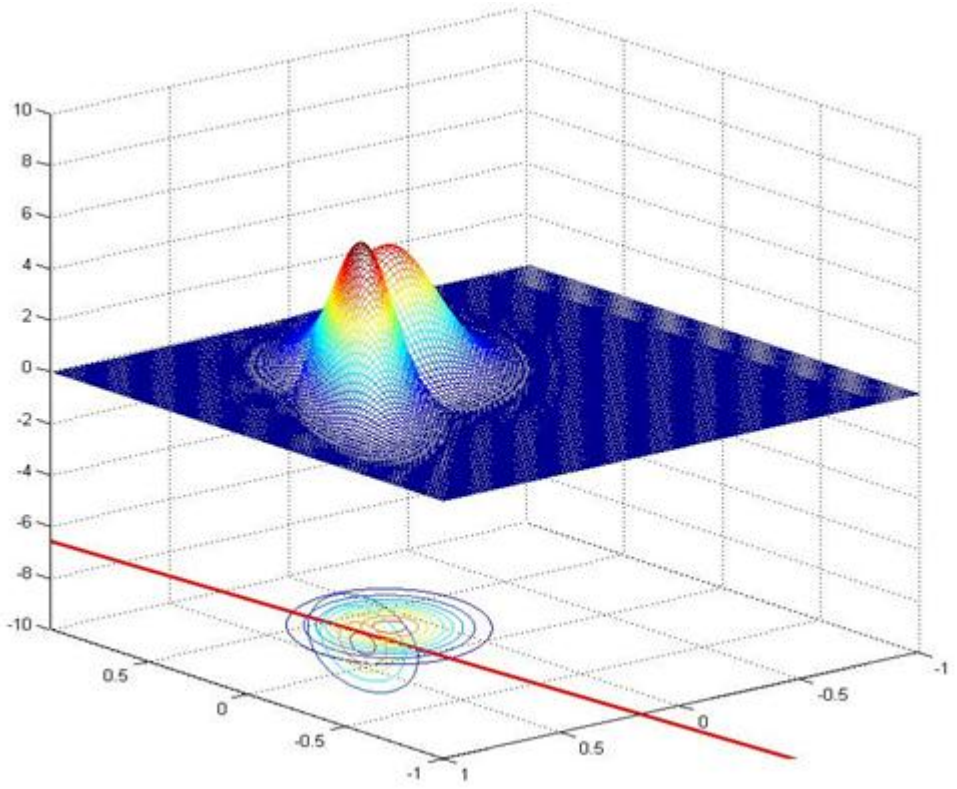

Figure 6. Decision Boundary

The decision boundary is used then to determine whether the related pixel is skin or not. A binary image would be created as the same size as the HSV image. This image would be the result of the classifier. Figure 7 shows the result of the classification. The skin pixels would be marked as the following:

$$binery(i,j) = \begin{cases} true & if\ HSV(i,j)\ is\ skin \\ false & if HSV(i,j) is\ not\ skin \end{cases}$$

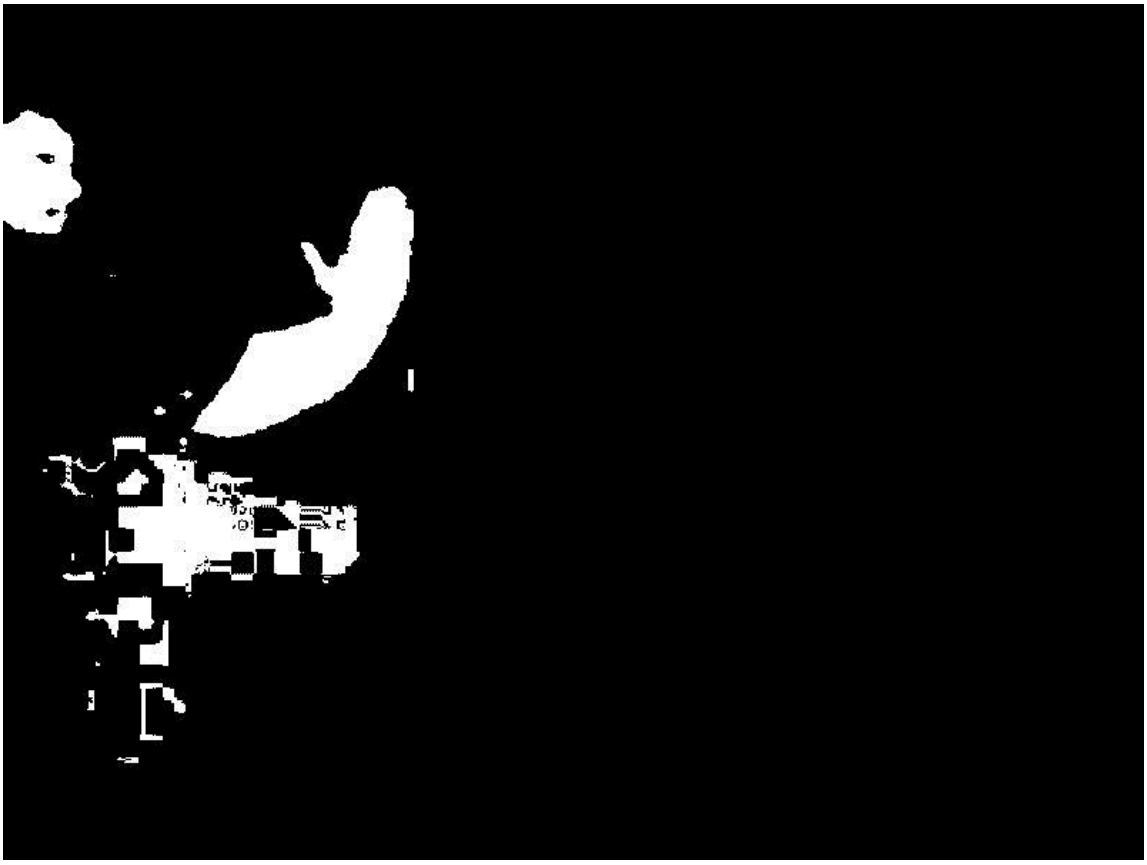

Figure7. Binary Image Representing Skin

2.1.2.3. Labeling Of Connected Regions and Cropping The Interested Labels

Finding the connected component in a binary image is a fundamental process in image segmentation. Each region is labeled with a unique number to separate it from the other regions. It could be used to determine the boundary of regions and recognize the objects in the binary image. [5]

The related pixel P in the binary image (I,J), has four direct neighbors as upper, lower, right and left (N4(p)), and four diagonal neighbors upper left, upper right lower left and lower right (Nd(p)), so N8(p) consists of Nd(p) and N4(p)[5].

First of all, the algorithm checks the whole image from right to left and from top to bottom. Assuming that the current pixel is P, and it has four neighbors (upper, upper left, upper right and left neighbors). If pixel P is marked 0 then it checks the next pixel. If it's marked 1 it then checks the neighbors. If all the neighbors are marked 0, it assigns a new label value for the current pixel P. If one of them is not 0 then it gives the current pixel the label of the neighbor pixel. If two or more of the neighbors are not 0 then it gives pixel P one of the neighbor's labels and marks the labels as equivalent in the labels list.[5]

This step gives the initial labeling to the connected regions in the binary image, but some of the regions have two labels, so another process should be performed as to resolve the equivalent labels.

In the initial labeling process, a list of similar labels has been created. This list contains the label and its equivalent label, so the algorithm checks the list and resolves the labels one by one until every region has one unique label. Figure 8 shows a labeled binary image.

Figure 8.Labeled Binary Image

The labeling algorithm provides the position and calculates area of the regions. It could use the area to filter the regions, if it is not large enough, the region will be ignored [5]. After that the interested regions will be cropped and suitably resized for the neural network. Figure 9 illustrates the cropped region.

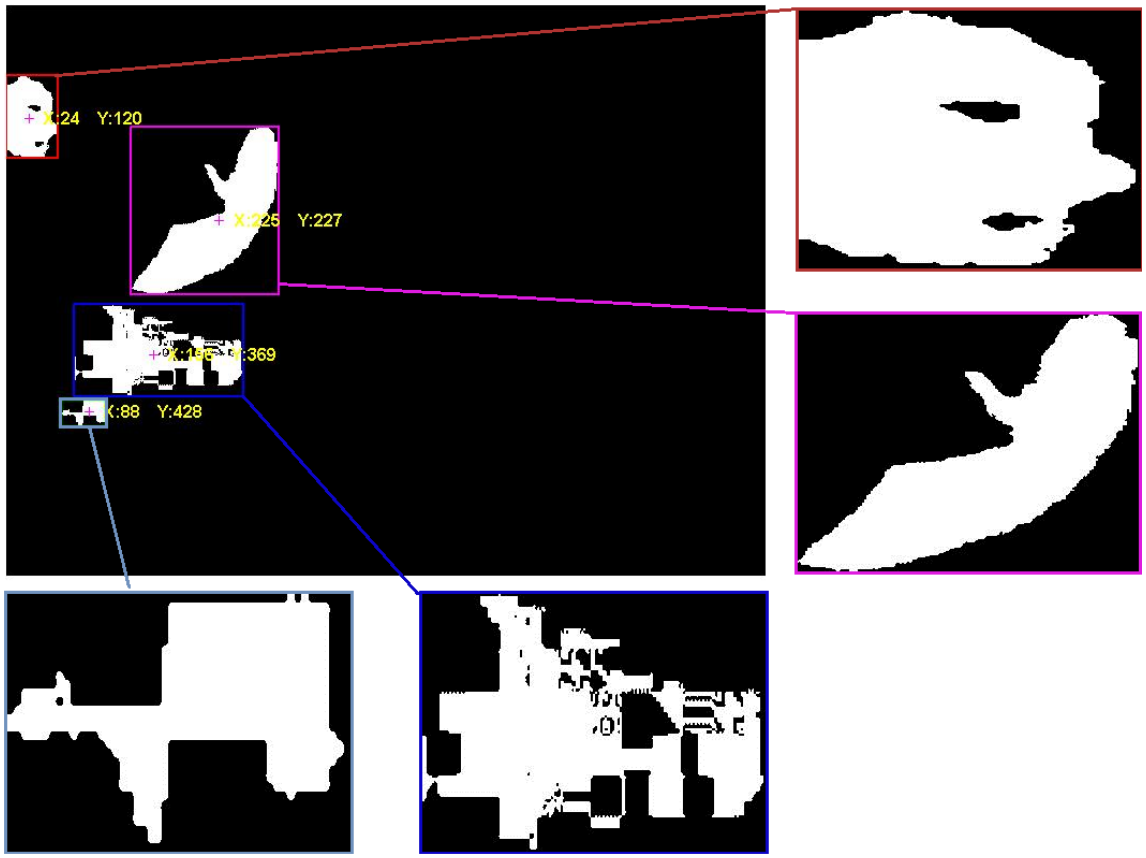

Figure 9. Cropped Image

2.1.3. The Neural network

A large range of problems are solved through the Artificial Neural Networks technology in simple and suitable approaches. The concept of Neural Network (ANNs) is comparable with the human neural network, therefore it has similarity with the word neural networks, as illustrated in Figure 10 [10]

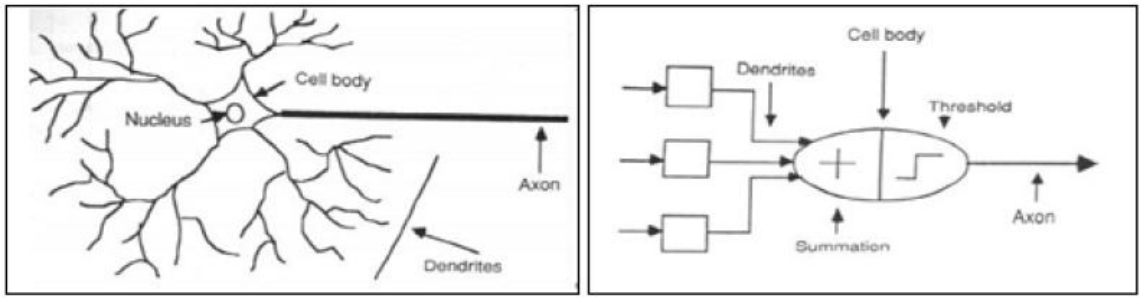

Figure 10. The Human Neuron Cell versus The Mathematical Model [10]

The definition of the Neural network is basically a simple processing unit consisting of a large parallel distributed processor which has the ability to store experimental knowledge and prepare it for use [10].

The output of the mathematical model of neuron relies on the summation of the input multiplied by a weight plus bias, output fired when the total input signal reaches a specific threshold value. The activation function controls the magnitude of the output, and then the output feeds another neuron in the network. Figure 11 describes this process mathematically[11].

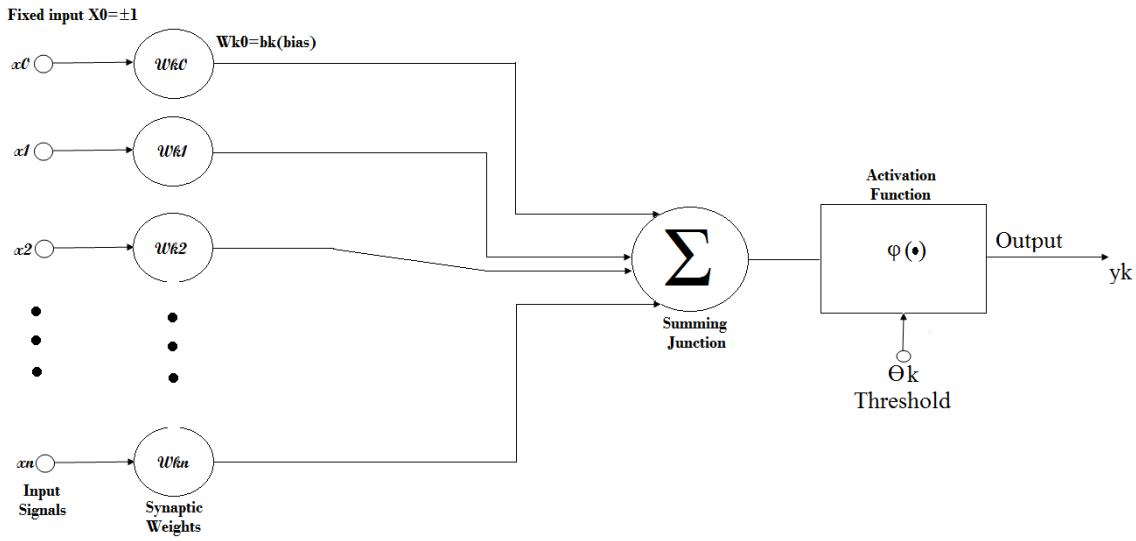

Figure 11.The Mathematical Model Of Neuron

The neural network will accept 200 real values as a summation vector of rows plus summation vector of columns of an image the size of 100X100 pixels. It would then be required to detect the hand by responding with a two elements output vector. The output elements of the network represent hand class and non-hand class. To obtain good results the network should respond with a 1 in the position of the hand to the network. The other value in the output vector should be 0 if the incoming image is representing a hand and vice versa.

2.1.3.1. The Architecture Of The Neural Network

The feed forward network is the simplest type of neural networks; data flows from the input layer through the hidden layers to the output layer [12]. Figure 12 illustrates the architecture of the feed forward network.

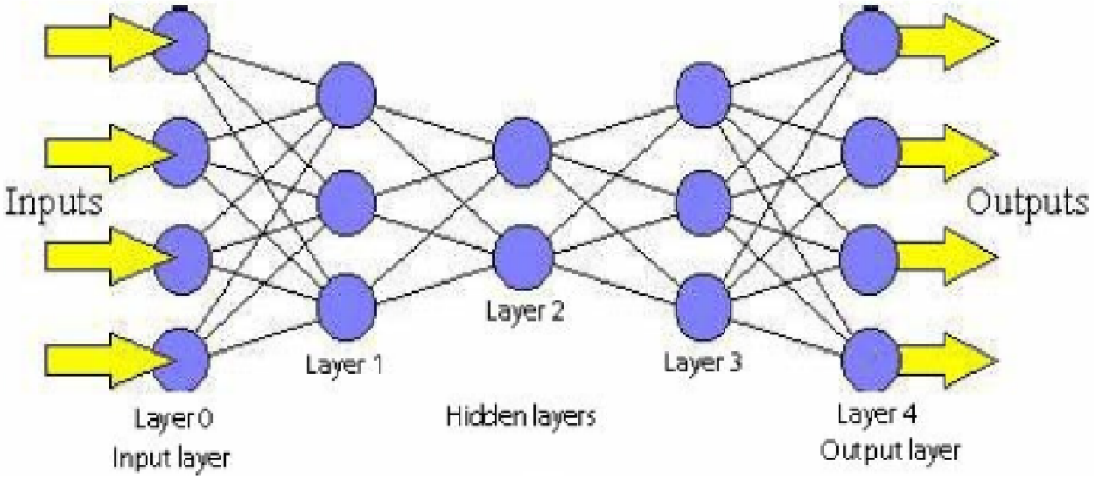

Figure 12.The Feed Forward Network

The neural network used in this project is the feed forward network. It consists of an input layer, a hidden layer, and an output layer. The input layer consists of 200 inputs, and two neurons at the output. The first hidden layer contains 400 neurons. The transfer functions used in the neural network is a log-sigmoid transfer. Function was detected because its output range (0 to 1) was an ideal output Boolean value. Figure 13 shows the input vector preparation method.

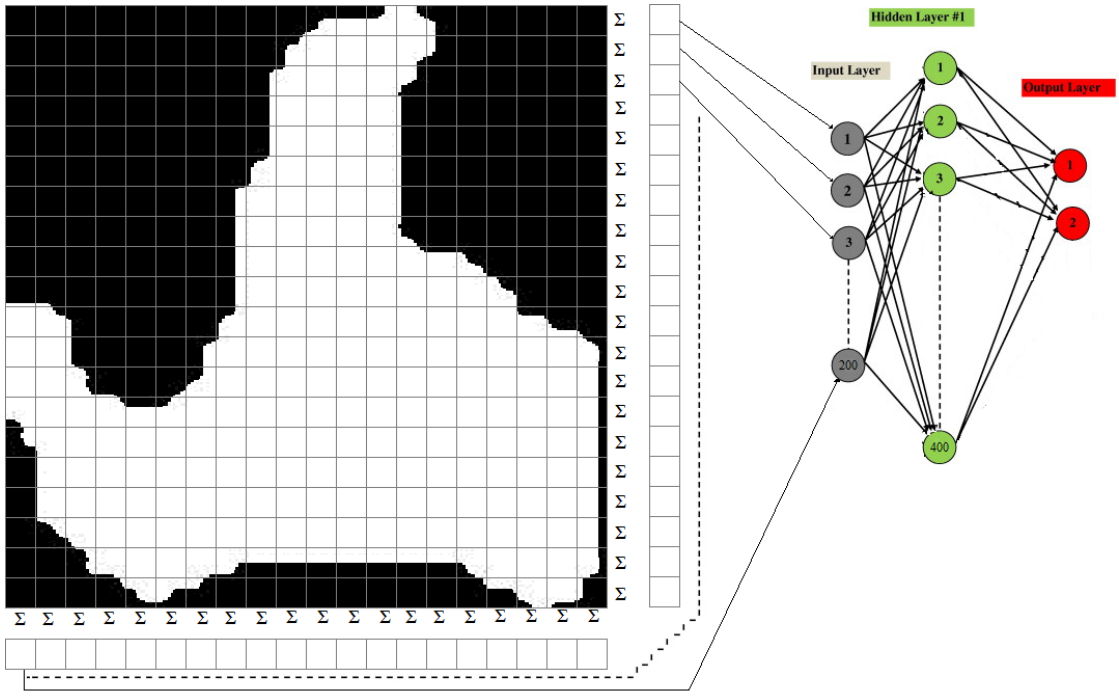

Figure 13. Input Vector Preparation From An Input Image

2.1.3.2. The Neural Network Training

The best widely used learning algorithm in a multilayer neural network is the Back-Propagation algorithm. Back Propagation is a supervised learning network which relies on a gradient descent learning rule. It provides an effective computational method for adjusting weights, with different activation functions. It is a guaranteed method to decrease the total squared error of the computed output of the network. The objective of the algorithm is to train the network to obtain a balance between the ability for a correct response to the input vectors used for the training as well as the ability to respond in the same manner as the input's vectors that have close features to the training vectors.[12]

In order to create a neural network that could recognize the hand from other objects it is essential to train the network on hand samples and non-hand samples.Figure14 shows samples from a hand database.

Figure 14. Positive Hand Samples

The network trained over 400 positive hand samples and 1000 negative samples of other objects. Figure 15 shows the slope of mean square of error.

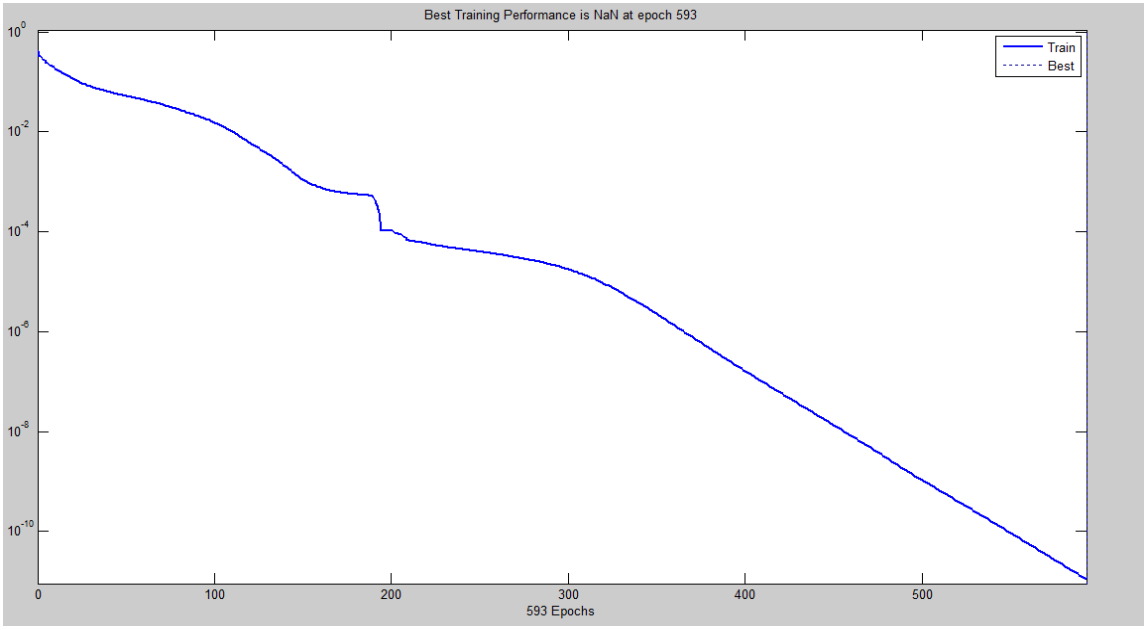

Figure 15. Mean Square Of Error

The training took approximately one hour to train the network and resulted in 1.21e^(-11) mean square of error.

This neural network was used for detecting a hand and differentiating it from other objects. The results of predicating the suspected objects by the network is shown in Figure 16. All other objects are ignored and deleted from the binary image.

Figure 16.Result of Hand Detection

The second neural network has the same architecture as the first but the purpose of it is to differentiate the gesture of the hand. If the first input image is a hand then the same vector will be the input for the second neural network. The training parameters are the same as the first network. Network training took approximately two hours with 10,000 epoch results. Figure 16 shows the performance slope of the second neural network.

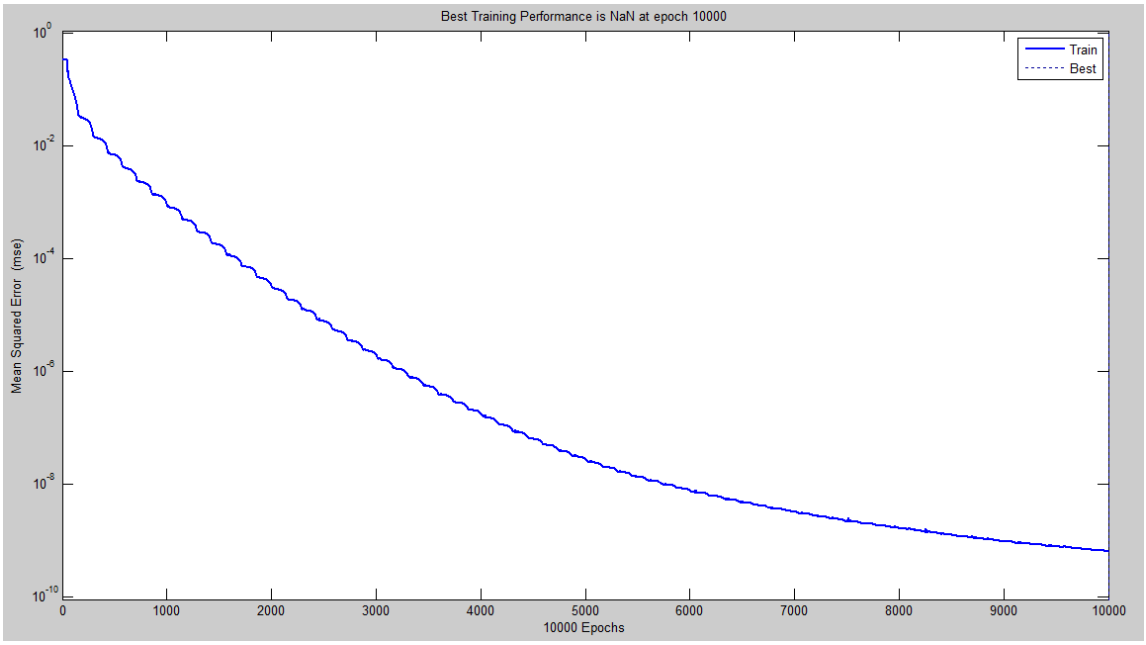

Figure 17. Second Neural Network Performance

The results of the second neural network are of the gestures of the hand interacting with the GUI of the computer by assigning one of the gestures as a left click and the other as a right click and normal position (only movement).

2.1.4. Interacting with GUI

To obtain some kind of interaction the position of the video displayed by the projector according to the camera should be calculated. At the beginning of the process the computer requests the manipulator to give four coordinates of the video (left-top right-top left-bottom right-bottom). The computer shows a full screen image and gives the location where the manipulator should put his hand and give a special gesture . Figure 18 shows the image shown by the projector for calibration.

Figure 18.Calibration Image For Two Directional Approach

The user enters the four co-ordinates which are used for the projection of the image/video according to the camera. These co-ordinates shall be used every time the camera detects a hand to calculate the position of the hand according to the position of the video displayed by the projector. The position of the mouse curser moves to the position of the hand according to the projection. The position of the mouse curser could be calculated as the follows:

Let P_{cam} denotes the center of the crosshair captured from the camera, so there are 4 points $P_{cam1}, P_{cam2}, P_{cam3}$ and P_{cam4}, on the other hand P_{Src} denotes the center of the points of the center of the crosshair according to the computer video, so there are $P_{Src1}P_{Src2}P_{Src3}$ and P_{Src4}. The transformation matrix of P_{cam} is

$$T_{cam} = \begin{bmatrix} a & d & 0 & g \\ b & e & 0 & h \\ 0 & 0 & 1 & 0 \\ c & f & 0 & 1 \end{bmatrix}$$

Where

$$dx1 = X_{cam2} - X_{cam3}, \qquad dy1 = Y_{cam2} - Y_{cam3},$$

$$dx2 = X_{cam4} - X_{cam3}, \qquad dy2 = Y_{cam4} - Y_{cam3}$$

$$sx = X_{cam1} - X_{cam2} + X_{cam3} - X_{cam4}$$

$$sy = Y_{cam1} - Y_{cma2} + Y_{cam3} - Y_{cam4}$$

$$g = (sx * dy2 - dx2 * sy) \Big/ (dx1 * dy2 - dx2 * dy1)$$

$$h = (sx * dy1 - dx1 * sy) \Big/ (dx1 * dy2 - dx2 * dy1)$$

$$a = X_{cam2} - X_{cam1} + g * X_{cam2}$$

$$b = X_{cam4} - X_{cam1} + h * X_{cam4}$$

$$c = X_{cam1}$$

$d = Y_{cam2} - Y_{cam1} + g * Y_{cam2}$

$e = Y_{cam4} - Y_{cam1} + h * Y_{cam4}$

$f = Y_{cam1}$

The same calculation should be performed to obtain the Screen transformation matrix T_{src}. To obtain the system transformation matrix the following equation should be performed:

$$T_{sys} = T_{src} \times inv(T_{cam})$$

This transformation matrix will be used every time the camera detects a hand to determine the appropriate position of the mouse. This position could be determined by performing the following equation:

If it is assumed that P_{hand} is the central point of the hand's position according to the camera, then

$P_{mouse} = T_{sys} * P_{hand}$.

After calculating the position of the mouse cursor, the computer moves the cursor to the new position and takes action according to the hand's gesture.

2.2. The three dimensional vision approach

This approach is based on a stereo camera which builds a 3-dimensional space and measures the distance of the skin regions within that space. The same pre-processing steps of the previous approaches are performed on both the two input videos which detect skin regions. Before the objects enter the neural network their distances are measured. Only the objects that are in touch with the projector's location that views the computer video enters the neural network to recognize the gesture. Figure 19 shows a flow chart regarding this approach.

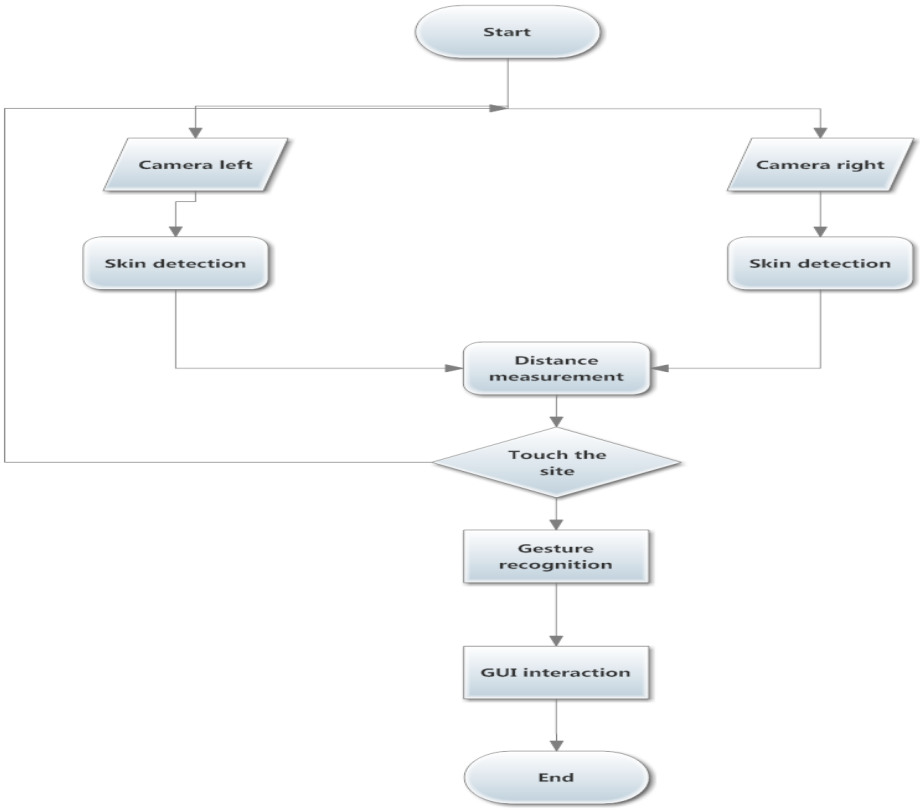

Figure 19.Second Approach Flowchart

2.2.1. Stereo vision

Stereo vision could be defined as two images captured from two cameras simultaneously. Each is overlapped by some amount of vision range with a slight shift. This overlap from two different views is used to detect depth and to build a three dimensional space for the detected regions. Figure 20 illustrates the stereo vision. [13]

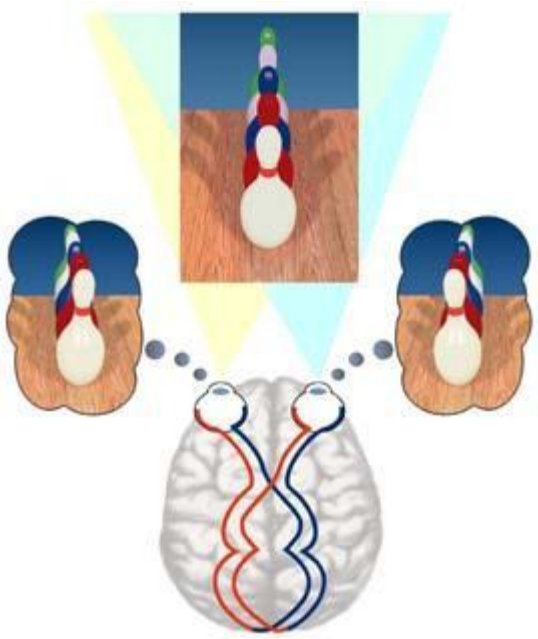

Figure 20. Binocular Vision in Humans[14]

In this system, both of the cameras are horizontally aligned and separated by a distance known as the baseline. Figure 21 illustrates the stereo ranging with simple arrangements. In this virtual system, the optical axes of the two cameras are ideally parallel, both image planes are on the same base line and no lens distortion is present.

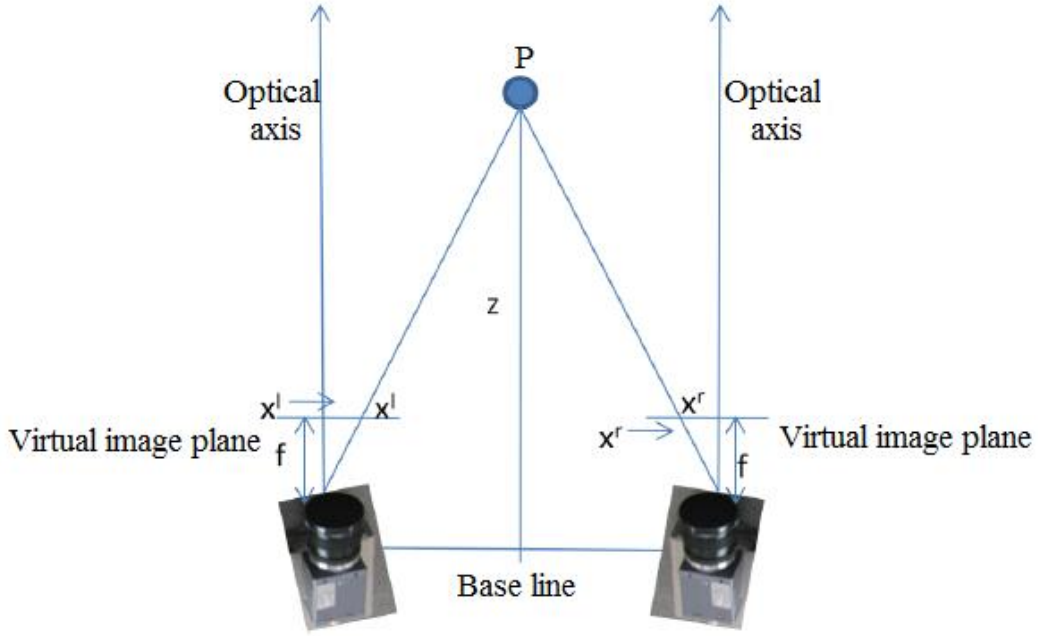

Figure 21.The Virtual Stereo System

In Figure 21 in order to provide the three dimensional co-ordinates of the point P, the distance from the base line to point P in the common vision scene should be determined. The distance Z could be calculated using the following equation:

$$Z = \frac{Bf}{dx}$$

where B is the base line and f is the focal length and dx is distance between the cameras(disparity), which is defined by:

$$dx = |xl - xr|$$

2.2.2. Camera calibration

Camera calibration is an essential step to provide stereo ranging system in reasonable accuracy. The calibration process calculates camera indications such as focal length, and angle between the cameras, etc. The chess board planner method is the easiest method to calibrate the cameras. Both of the cameras are placed to detect the chessboard pattern according to the angle of the cameras. Figure 22 shows a chess pattern captured from two cameras.

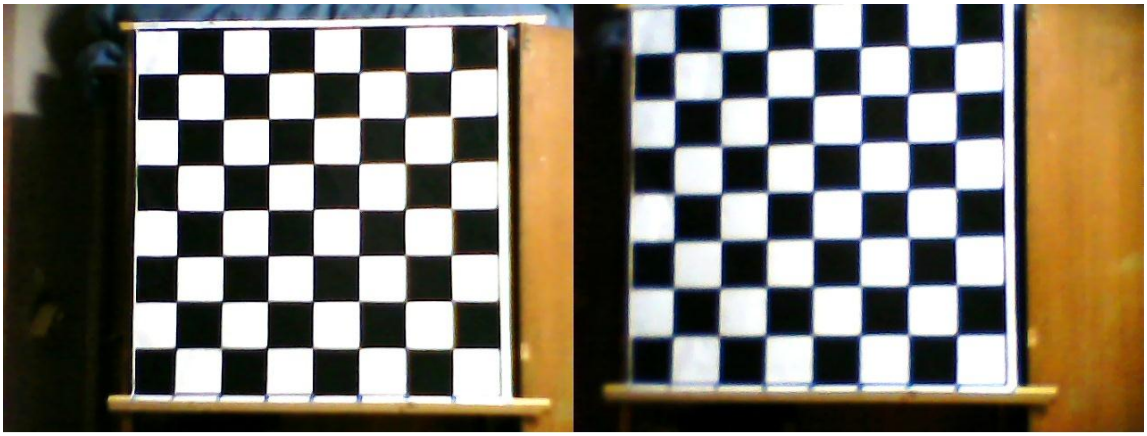

Figure 22. Left And Right Camera Site Of View

The corners are exposed by the calibration system automatically after selecting the border of the chessboard by hand in both of images. For best results it is better to use at least ten images of multiple views of the chessboard. The calibration process provides rotation and translation vectors of both left and right cameras as well as the camera parameters.

2.2.3. Camera Mathematical Model

A point in a three-dimensional scene represented in a two-dimensional image by perspective projection equation q=MQ, where Q is three-dimensional point, M is camera matrix, and q is the resulted two-dimensional image point.

The camera matrix contains the intrinsic parameters of the camera:

$$M = \begin{bmatrix} fx & 0 & ox \\ 0 & fy & oy \\ 0 & 0 & 1 \end{bmatrix}$$

where cx, and cy are the center of the image which is determined by the optical axis. fx, and fy are the horizontal and the vertical scale factors.These two parameters are obtained from focal length, $f$where $fx = sx * f$, and $fy = sy * f$. [15]

The calibration process also produces the camera distortion parameters, where the straight lines in the three-dimensional world appear to be curved in the image due to glass lenses of the camera. This effect is more obviously noticed near to the boundaries of the image. This effect is generally called radial distortion. Figure 23 illustrates the radial distortion.

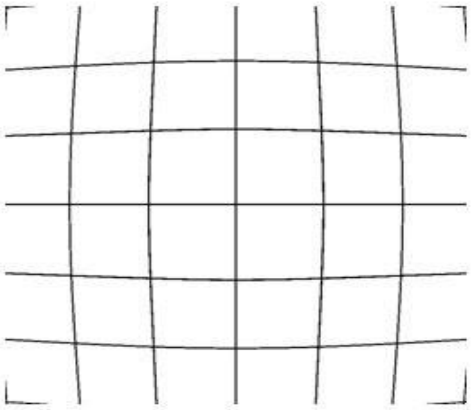

Figure 23.Radial Distortion Effect

The calibration process produces the distortion as a vector as the following form:

$$D = [k_1 \quad k_2 \quad k_3 \quad k_4 \quad 0]$$

The corrected image coordinates could be calculated by the following equations:

$$x_{corected} = x + x \times \sum_{i<4}^{1} k_i r^{i*2}$$

$$y_{corected} = y + y \times \sum_{i<4}^{1} k_i r^{i*2}$$

Where r is the distance from the center of image to the point $P(x, y)$. Figure 24 shows the correlated image.

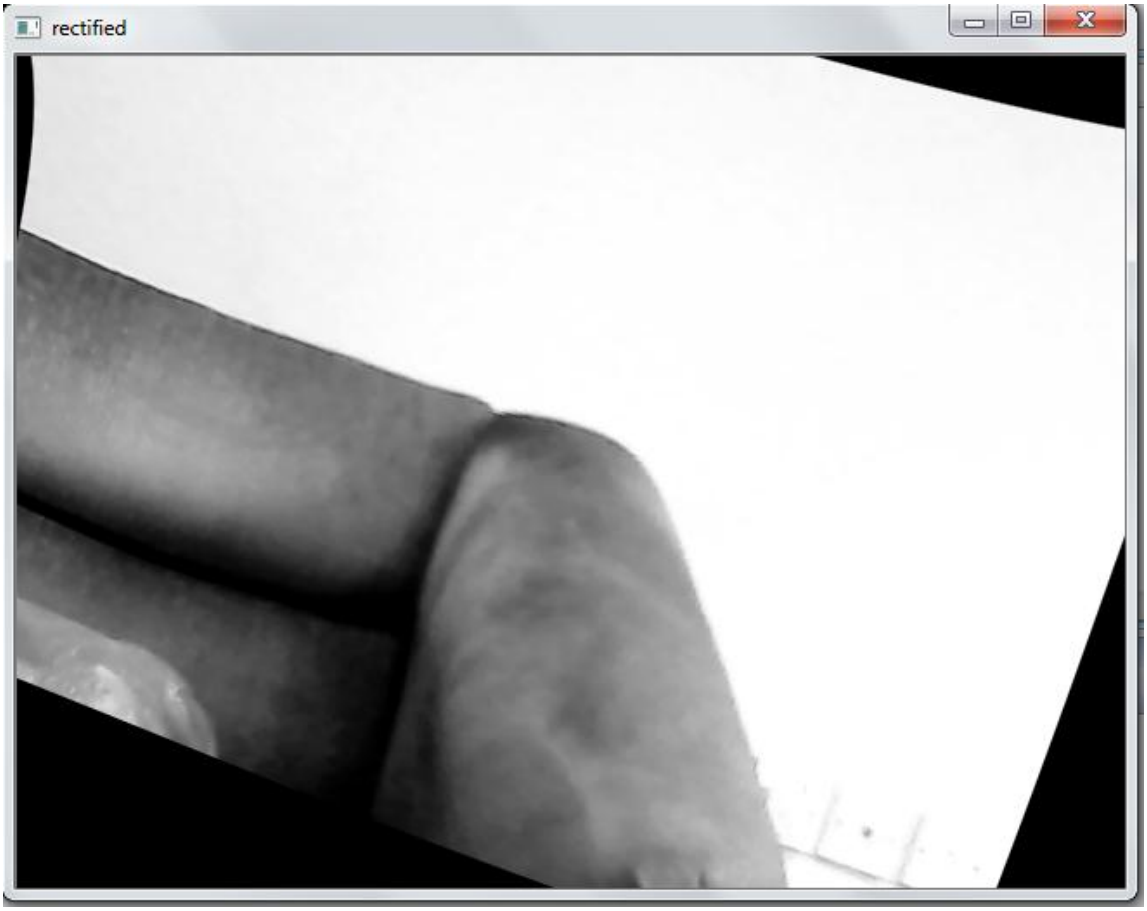

Figure 24. Rectified Image

2.2.4. Rectification

The rectification process is used to improve the speed of finding stereo correspondences by rectifying stereo image pairs and warping these images so that corresponding points become on the equivalent raw, in other words, transforming the epipolar lines to be on the same line.

Figure 25 illustrates the geometry of stereo imaging system. The point P represented in the left view as Xl and in the right view as Xr. Ol, and/or are the projection points which

determine the epipolar plane with point P. The epipolar lines are the distance between the X point to epipolar intersection points.

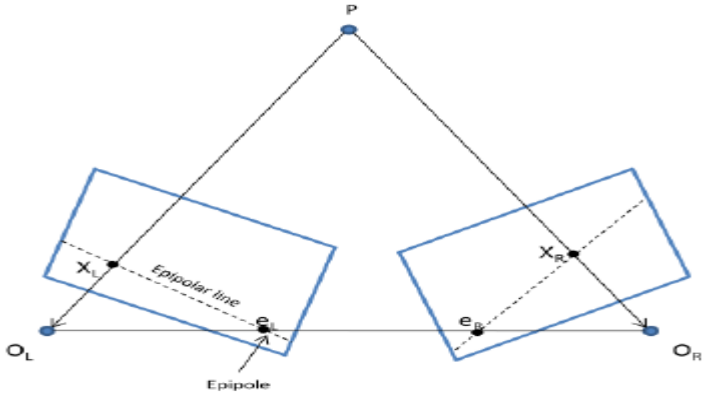

Figure 25. Stereo Image Geometry

Epipolar lines could be calculated by two matrixes; Essential matrix and Fundamental matrix. The essential matrix E is the three dimension physical space to represent the view of image for both the left and right cameras. It contains the rotation and translation from the left camera position to the right camera position in the real coordinates. The fundamental matrix contains left and right epipoles parameters, and the homography of left to right image plane [15].

2.2.5. Correlation

The purpose of correlation is to match the points between stereo images by un-distorting and rectifying both left and right cameras. Calculating the disparity between images is an essential step to determine the distance between the stereo system to the objects. The disparity is the contrast in the position of a certain point in left and right images. The block matching technique is an algorithm to obtain the correlation and calculate disparities. Sum of Absolute Difference windows (S.A.D. windows) are used to

determine the correspondences. SAD algorithm is an area-based correspondence algorithm. It calculates the gray level contrast for the related pixel in a kernel as in the following equation: [16]

$$SAD(i,j) = \sum_{-w}^{y=w} \sum_{-w}^{x=w} |Pr(y+i, x+j+d) - Pl(y+i, x+j)| \text{ [16]}$$

where (i,j) is the related point, d is the disparity of the right to left point, w is the size of the kernel.

This algorithm results in a gray level image where each pixel is the disparity calculated from the right and left images. The resulted image is called a disparity map. The gray levels in the disparity map represent the distance to the objects where the bright intensity represents the close object, and dark area are the far objects. Figure 26 shows an example of disparity map.

Figure 26. Disparity Map

2.2.6. Interacting with GUI

To make the system interact with hand, the system needs the position of the hand and the position of where the projector views the computer video and then starts to detect the skin region in this site. If there is skin region in this site then the process calculates the distance of this region. If it is on the video view's surface then the computer calculates the position of this region according to the surface and moves the mouse curser to that resulted position.

The area where the projector views the computer video could be entered to the computer manually. At the beginning of the program the computer views the images captured from the stereo system and requests the manipulator to enter the corners of the video site view. After that the computer calculates the three dimensional points of the real position

of the corners and calculates the transformation matrix Mv between camera and the view site. This matrix contains the scale and translation information:

$$Mv = \begin{bmatrix} sx & 0 & 0 & tx \\ 0 & sy & 0 & ty \\ 0 & 0 & sz & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position of the cursor could be calculated using the following equation:

$$P_{cur} = Mv \times P_{hand}$$

The mouse action could be decided using the neural network, the skin region which is in touch will be cropped and resized and the same neural network used in the first approach is used to categorize the gesture of the hand to take the action.

2.3. The Infrared sensor approach

This approach is dependent on an infrared two dimensional sensor where data is gathered by an array N*M of infrared receptors to form a binary image. The sensor detects the place of the infrared and then moves the mouse cursor to the equivalent place on the video of the computer and detects and checks the switch to take the suitable action. Figure 27 shows a flow chart of this approach.

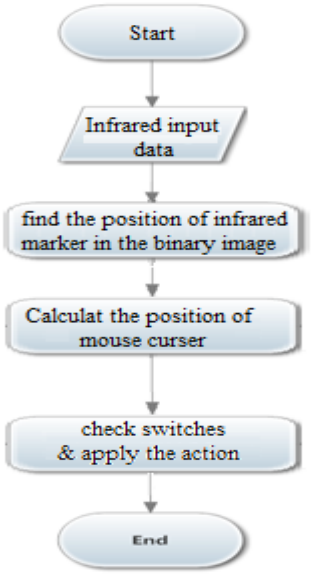

Figure 27. Third Approach Flow Chart

2.3.1. Sensor Data Gathering

The sensor in this image and infrared light spots are represented as white pixels. The sensor will detect the infrared light spots and so there is no need to make the intelligent system to do the detection. Figure 28 shows the binary image acquired from the sensor.

Figure 28. Acquired Image from Infrared Sensor

2.3.2. Electric Glove

The electrical glove used in this project contains an important component to communicate with the computer. These components are infrared LEDs which are used as a signal to detect the position of the glove by the infrared sensor and two switches for left and right click as well as the transmitter to transmit the status of the switches to the computer. Figure 29 shows the electrical glove used in this approach.

Figure 29. Electrical Glove

2.3.3. Interacting with GUI

To interact with the computer, the program must first know the position of the computer video viewed by the projector. At the beginning of the program, the computer view calibration window guides the manipulator to where the operator should click to give the rectangle of the view. Figure 30 shows the calibration window.

Figure 30. Calibration Window For Infrared Sensor Approach

When the manipulator gives a left click on the crosshair shapes shown in figure 30, it then gives the rectangle view of the four corners so it could perform the same calculation of the first approach to obtain the transformation matrix which is used to calculate the mouse cursor position.

Chapter 3

VIRTUAL SMART BOARD TOOLS

3.1. First approach tools

The first approach can be tools found in Appendix As Two Dimensional Tools C++ which is a source code developed in C++ to perform hand tracking using the OpenCV library. The tool kit accepts a data file about the decision boundary of skin detection classifier. The data file contains the trained neural network. The decision boundary was created and trained by using OpenCV and saved in the file (Skin.xml). The neural networks were created by OpenCV and saved in two files (Hand.xml& Gesture.xml). The application is set up to process the video acquired from the camera and the output is the position of the hand.

3.1.1. Hardware Tools

The hardware used in this approach is a low cost USB webcam. Table 1 shows the manufacture specification of the camera used in this approach.

Table 1.Inca Camera Specification

Inca IC 3562 webcam	
Sensor type	1/4 progressive scan CCD sensor

Maximum video resolution	1280 x 960 pixels
Maximum frame rate	30 fps
Focal length	3.85mm
Connectivity	USB 2.0
Price	10$

Figure 31 shows the camera used in this approach.

Figure 31.Camera used in 2D approach

3.1.2. Software Tools

In order to train a Bayes classifier to detect skin pixels, we used $CvNormalBayesClassifier$ class, the training function in this class is as the follows:

$$bool\ CvNormalBayesClassifier :: train(const\ CvMat *\ train\_data,$$

$$const\ CvMat *\ responses$$

$$, const\ CvMat *\ var\_idx\ = 0,$$

$$const\ CvMat *\ sample\_idx = 0\ ,$$

$$bool\ update = false\ );$$

The argument $train\_data$ is a matrix containring the training vectors for both the classes and the required response is the argument $responses$ which is also a matrix that must contain integer numbers. The argument var_idx and $sample\_idx$ are not supported and are received for future. The parameter $update$ identifies whether the model should be trained from scratch or should be updated using the new training data.

To use this classifier for skin detection the function $predict$ needs to be used on all image pixels to estimate the most probable class of the input pixel. $predict$ function is as the following:

$$float\ CvNormalBayesClassifier::predict($$

$$const\ CvMat *\ samples,$$

$$CvMat * results = 0) const;$$

The argument $samples$ is the input matrix containing the features of the related pixel. The argument $results$ is the output of this function.

In order to obtain labeling algorithm, the class $CBlobResult$ is used. The contract function of this class is as the following:

$$CBlobResult :: \text{CBlobResult(IplImage} * \text{source,}$$

$$\text{IplImage} * \text{mask, uchar backgroundColor);}$$

The first argument is the input grayscale image, the second is the mask image. If there is no mask then it is NULL. The last one is the background color.

OpenCV library also contains the neural network class $CvANN\_MLP$. This class implements the feed forward neural network. In order to create a neural network the function $create(\quad)$; should be used. This function is as follows:

$$void\ CvANN\ MLP::create($$

$$const\ CvMat * layer\_sizes,$$

$$int\ activ\_func = SIGMOID\_SYM,$$

$$double\ f\_param1 = 0,$$

$$double\ f\_param2 = 0\ );$$

The argument $layer\_sizes$ is a matrix containing the architecture of the neural network where the length of the matrix is the number of the layers in the neural network, and every element of this matrix is the size of the related layer. For example, if the $layer\_sizes$ matrix elements are (100 200 50 2) then the number of layers is four layers and the input layer containing 100 input elements and the first hidden layer is 200 neurons. The second hidden layer is 50 neurons. The output layer contains 2 output elements. The second argument is the activation function of the neural network where the default activation function is ($SIGMOID\_SYM$). The third argument f_param1 is the free parameter α of the activation function and β is the fourth argument of the function $create$. These arguments determine the network topology.

The function $train$ is used to train the neural network and to adjust the network weights. It returns the number of iterations done. The $train$ function is as follows:

$$int\ CvANN\ MLP::train($$

$$const\ CvMat*\ inputs, const\ CvMat*\ outputs,$$

$$const\ CvMat*\ sample\ weights,$$

$$const\ CvMat * sample\_idx = 0,$$

$$CvANN\ MLP\ TrainParams\ params\ ,$$

$$int\ flags = 0\ );$$

The first argument of this function is to input the training data in a matrix form where every row is a training sample. The second argument is a matrix of the corresponding output, every row is considered as the required output to the related input row. The third argument is optional, it is a vector containing the important order samples. The argument $params$ is a structure containing the training parameters. This structure has the following constructor:

$$CvANN\_MLP\_TrainParams($$

$$CvTermCriteria\ term\_crit,$$

$$int\ train\_method, double\ param1, \quad double\ param2\ );$$

The first argument is the training T$erm\ Criteria$ where it is also a structure containing the type and the max iteration (Epoch) and the $epsilon$ value. The second argument is the $train\ method$ which is either a back propagation (0) or a batch RPROP algorithm 1. The arguments $param1$, and $param2$ are optional to denote the moment scale value.

In order to use this neural network it needs to use the function $redict$. This function estimates the output vector according to the input vector using the neural weights. The argument of this function is as the following:

$$float\ CvANN\ MLP::predict(\ const\ CvMat*\ \_inputs,$$

$$CvMat*\ \_outputs\ )$$

where the input matrix is the input sample and the output matrix is the estimated output.

3.2. Three Dimensional Approach Tools

Stereo Vision System C++ source code can be found in Appendix B as "3D vision tools". The code accepts two cameras and calibration data file as arguments. Matlab stereo vision toolkit is used to create a calibration data file. The application is programmed to calculate the distances of skin regions in both the images.

3.2.1. Hardware Tools

The hardware used in this approach are two low cost USB web cameras. Table 2 shows the manufacture specification for both cameras used in this approach.

Table 2.Poges Camera Specification

Poges pgc-106 webcam	
Sensor type	1/4 progressive scan CCD sensor

Maximum video resolution	800 x 600 pixels
Maximum frame rate	15 fps
Focal length	4.55mm
Price	9$

Figure 32 shows the stereo vision system used in this project.

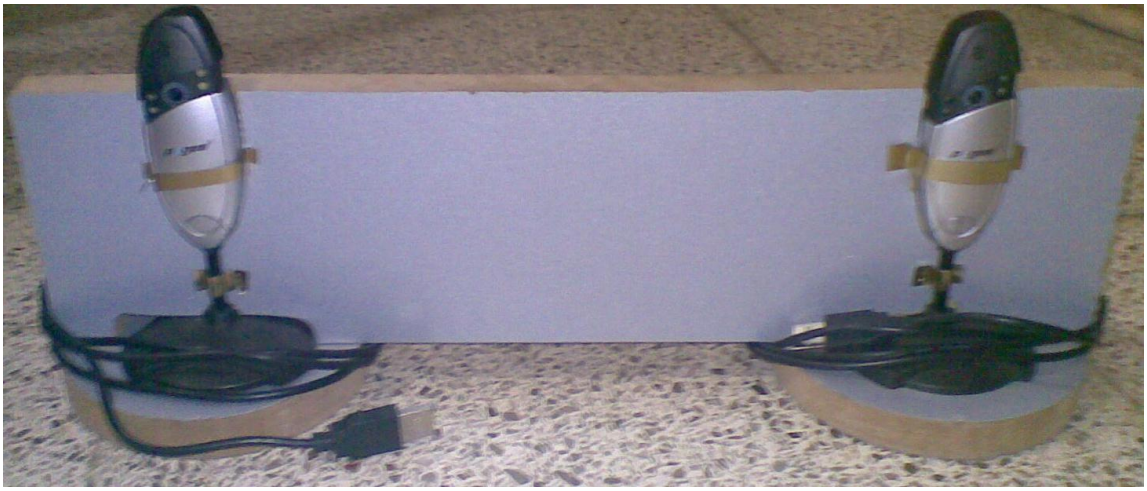

Figure 32.The Stereo Vision System

3.2.2. Software Tools

In order to rectify the two images acquired from stereo camera, the function $cvStereoRectify()$ must be used. This function needs the calibration data of the stereo system to do the rectification. The MATLAB stereo toolkit was used to obtain the calibration data. Appendix B contains the steps to do the calibration. The file resulted from the calibration step contains the following (right and left camera model matrices, right and left distortion effect vectors, rotation and translation information between the cameras). When the program reads the calibration file the function

$cvStereoRectify()$ is used to obtain the rectified rotation between left and right and the projection matrix for both of left and right cameras. The function $cvStereoRectify()$ is as follows:

$$void\ cvStereoRectify(\ const\ CvMat*\ M\_left, const\ CvMat* M\_right, const\ CvMat*\ D\_left, Const\ CvMat* D\_right\ , CvSize\ imageSize, const\ CvMat*\ R, const\ CvMat* T, CvMat*\ Rl, CvMat*\ Rr, CvMat*\ PL, CvMat*\ PR, CvMat*\ Q = 0, Int\ flags = CV\_CALIB\_ZERO\_DISPARITY\ );$$

The first two arguments of this function M_left and M_right are the camera model matrices for the left and right cameras. The third and fourth D_left and D_right arguments are the distortion effect vectors for the left and right lenses, to correct radial distortion. The $imageSize$ argument is the size of incoming image from both of the cameras. The next two arguments$R$ and T are the rotation matrix of the stereo system and the translation vector respectively; these variables are obtained from the calibration process. The function output arguments are left image plane Rl and right image plane Rr, left and right projection matrix (PL and PR).

To calculate the un-distortion and rectification transformation map for both of incoming images, this function $cvInitUndistortRectifyMap()$ should be applied to both of the images separately. $cvInitUndistortRectifyMap$ has the following form:

$$void\ cvInitUndistortRectifyMap(const\ CvMat*\ M\ ,const\ CvMat*\ D,const\ CvMat*\ R,const\ CvMat*\ P,CvArr*\ mapx,CvArr*\ mapy);$$

The first two arguments are the camera model matrix and the distortion effect vector. The next two arguments are the image plane and projection matrix of the related camera. The output arguments $mapx$ and $mapy$ are pixel mapping of the correlated images.

To obtain the correlated images from both of the incoming images, the function $cvRemap()$ should be applied to both of the incoming images. This function generates generic geometrical transformation to the image. This function has the following form:

$$void\ cvRemap(\ const\ CvArr*\ src,const\ CvArr*\ dst,const\ CvArr*\ mapx,const\ CvArr*\ mapy,int\ flags = CV\_INTER\_LINEAR,CvScalar\ fillval = cvScalarAll(0),);$$

The first argument src is the input image. The next one is the resulted correlated image. The arguments $mapx$ and $mapy$ are the map coordinates of the input image. $flags$ argument is to determine the grouping of interpolation method .

In order to generate distances map (disparity map) the function $cvFindStereoCorrespondenceBM()$ should be used. This function uses the block matching algorithm. This function is as the following:

$$void\ cvFindStereoCorrespondenceBM(\ const\ CvArr*\ leftImage, const\ CvArr$$
$$*\ rightImage, const\ CvArr$$
$$*\ disparityImage, const\ CvStereoBMState*\ BMState,);$$

The first and second argument is the left and right images acquired from the stereo system. The third argument is the result disparity map. The last one is the block matching algorithm parameters. These parameters are used for the filter type of the filter namely preFilterType and it is either $CV\_STEREO\_BM\ NORMALIZED\_RESPONSE$ or the default recommended CV STEREO_BM_XSOBEL. The variable preFilterSize is to determine the size of the filter, and the variable $SADWindowSize$ is to determin the size of window of block matching.

3.3. Infrared Approach Tools

The Infrared approach depends on the Infrared two dimensional sensor and electric glove. Appendix C as "infrared approach tools contains C# source code", sensor, and glove setup.

3.3.1. Hardware Tools

The infrared sensor used in this project is a Wiimote manufactured by Nintendo Co., Ltd. This remote contains an infrared camera in the front which detects only the light in infrared space. It is connected to the computer by Bluetooth adhoc network to transmit infrared sensor and switches data to the computer. This sensor senses a view with 33 degrees horizontally and 23 degrees vertically. Figure 33 shows the Wii remote used in this project.

Figure 33.Wii remote

In order to detect the hand position in the detected site of the sensor, infrared led was used as a marker. This marker is fixed on one of the fingers of a glove with batteries. The glove also contains two switches for (left & right click), and a FM transmitter to send the switches status to the computer. Figure 34 shows the electrical glove and the switches.

Figure 34.(a)Electrical glove front side (b) Electrical glove rear side

3.3.2. Software Tools

The sensor data is stored in buffers which have 10 bits for X coordinate, 10 bits for Y coordinate and 4 bits for intensity. Table 3 shows the bit arrangements in the sensor buffer.

Table 3.Sensor Buffer Arrangements

	Bits							
Byte	7	6	5	4	3	2	1	0
0	X<7:0>							
1	Y<7:0>							
2	Y<9:8>		X<9:8>		S<3:0>			

To import this buffer the class $Wiimote State$ should be defined. In this class there are variables changed automatically by an asynchronous function. These variables are $RawX1$ and $RawY1$ (X coordinate and Y coordinate of the infrared marker).

Chapter 4

RESULT AND DISCUSSION

In this chapter the performances of all approaches are described and an analysis is made between them to decide which approach is optimal to be used as a smart white board application. The first and second approach were implemented using OpenCv Library version 2.1 using Microsoft Visual C++.net. The third approach was implemented by Wiimote library using Microsoft Visual C#.net. All of the three approaches were tested by CORE I7 processor.

4.1. First approach Results and Performance

In this approach there were three steps as mentioned in chapter two, which are the skin segmentation, hand detection and hand gesture recognition.

4.1.1. Skin Segmentation Performance

Since the detection of the hand relies on the segmentation of the skin, It was essential to test the skin detection performance. As mentioned in Chapter 2, the segmentation of skin relies on Bayes classification approach. Figure 35 shows some skin classification results.

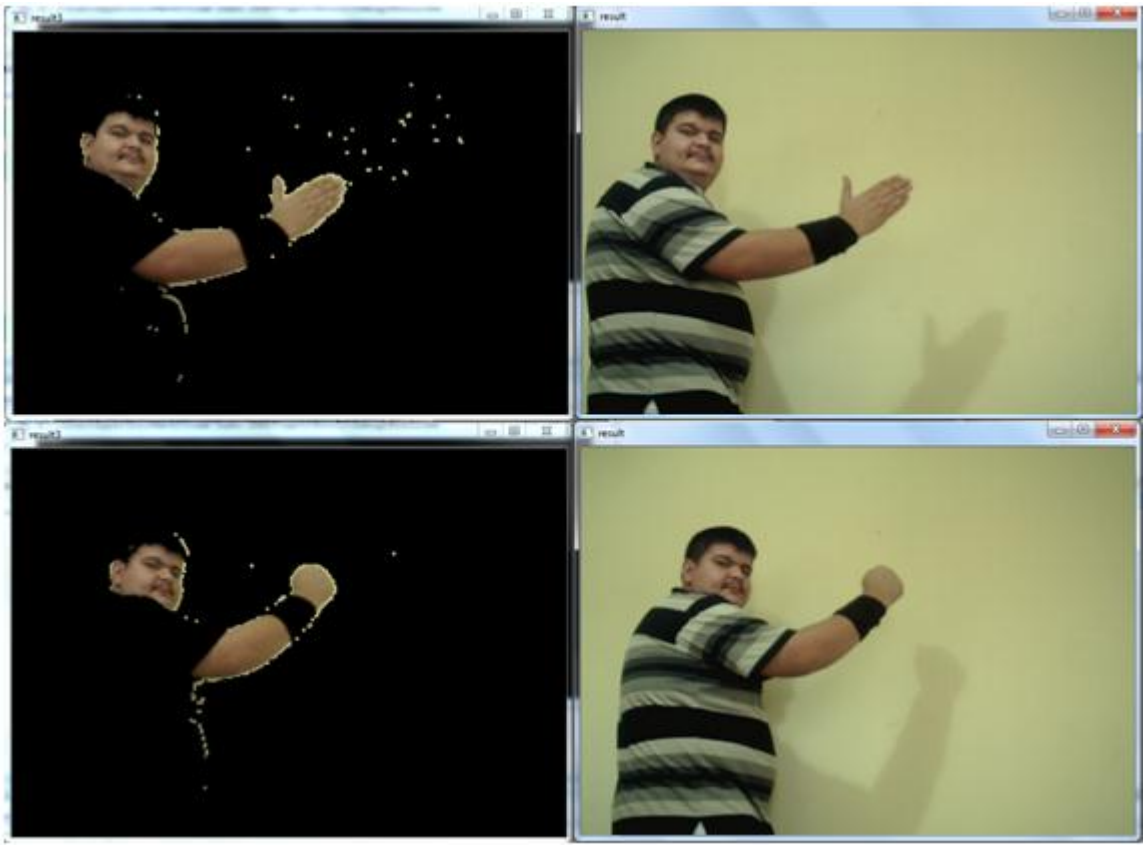

Figure 35.Skin Detection Results

From figure 35 it is possible to see that there are some places classified as skin region however it is not skin which means that there are some errors in the classification. These errors could be measured by calculating the ratio of misclassified area with over all area to get the error ratio. The error ratio of the skin classification is 1.317% for false classification.

4.1.2. Hand detection performance

The hand detection relies on the neural network classifier where skin regions are cropped and sent to the neural network to be classified as a hand or not. Figure 36 shows some results of neural network classification.

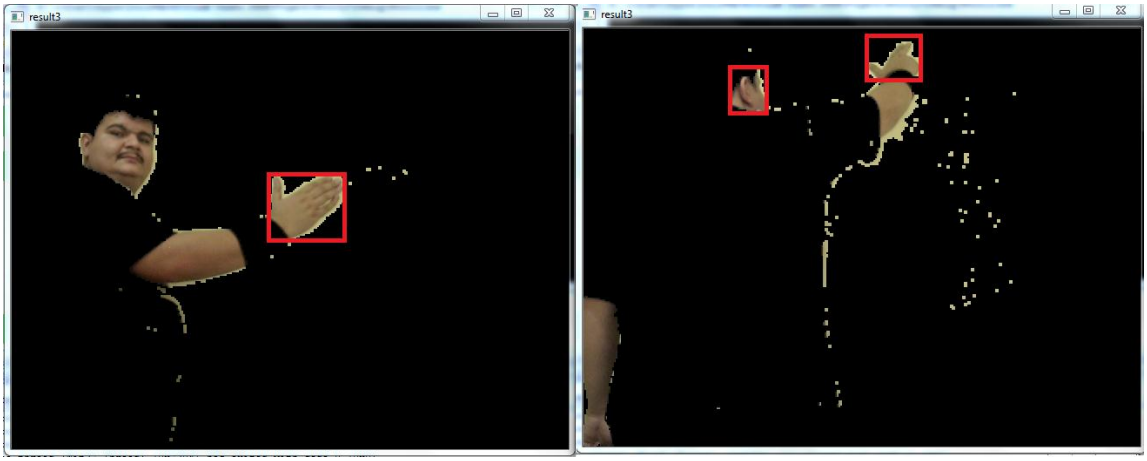

Figure 36. Hand Detection Results

In Figure 36, it can be seen that there is a region falsely classified as a hand. This is occurred because of the similarity in the input vector of the tested image, and it could affect the performance of the overall system. This error could be limited by checking the previous frame results. If there is a hand near by the current place then it is a hand otherwise this region will be ignored.

4.1.3. Hand gesture recognition performance

The gesture recognition relies on the neural network. The detection phase detects the hand and feeds the data to the gesture recognition classifier. In this phase the error ratio was very low and unnoticeable. Figure 37 shows final result of the system, where the hand was detected and recognized according to the gesture.

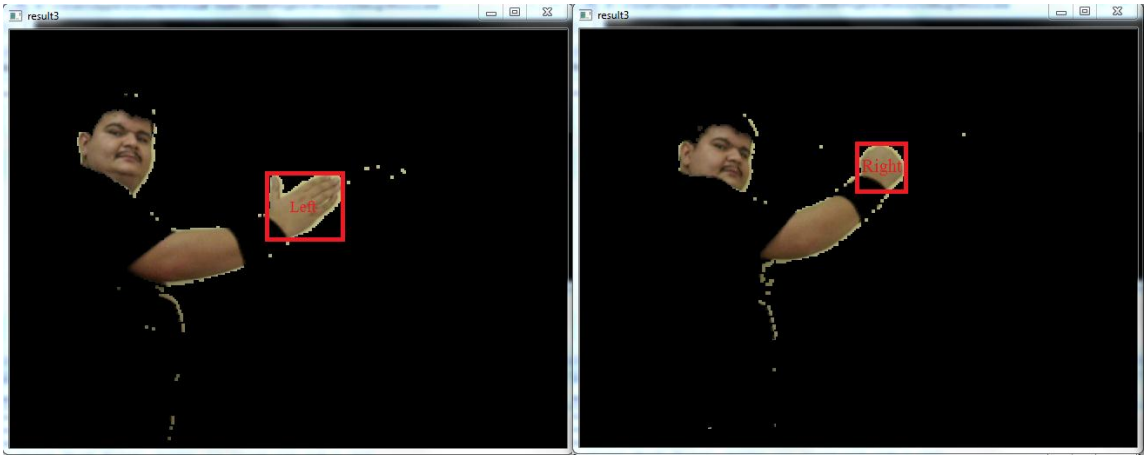

Figure 37.Gusture Recognition Result

4.1.4. Real-time performance

The time to obtain the hand position and to determine the gesture is very important since this types of projects need to be fast enough to interact with the human. In Table 4 there are the time intervals used to detect and recognize the hand.

Table 4. Mean Response Time of First Approach

Resolution	**Response Time**
320x240	154ms
640*480	344ms
800*600	638ms
1280 x 960	1.035 sec

4.2. Second Approach Performance

This approach depend on stereo vision to calculate the distances of skin objects in the detected site.

4.2.1. Real-time Distance Measurements Performance

The purpose of the stereo system is to calculate the skin region's distance. Figure 38 shows the left camera image with the detection of the skin region. Table 5 shows the distances between stereo system and skin regions in Figure 38.

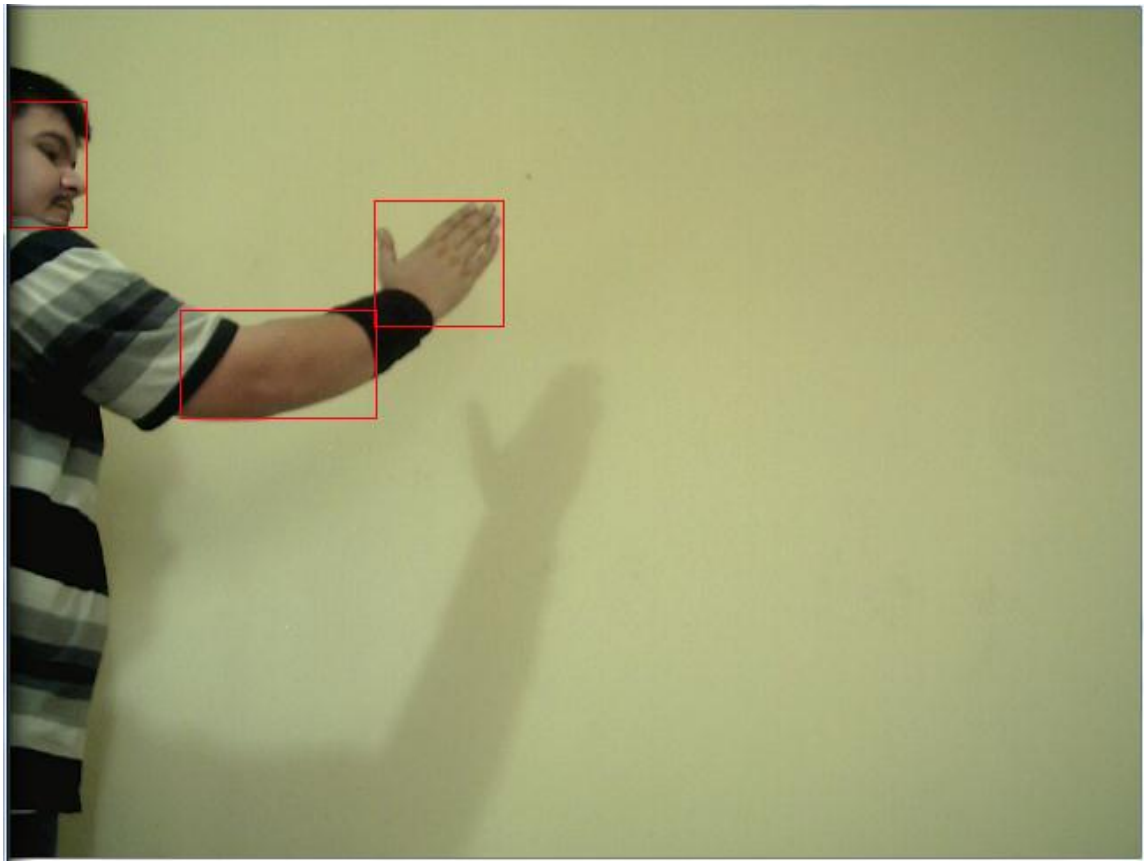

Figure 38. Left Camera Skin Detection

Table 5. Distances between stereo system to objects

Object	Measured Distance	Calculated Distance	Error Ratio
Face	1.34m	1.36m	4%
Hand	1.70m	1.76m	3%
Arm	1.62m	1.69m	4%

The system speed and response times are shown in Table 6 for multiple sized images.

Table 6. Response Time of Second Approach

Resolution	**Response Time**
320x240	334ms
640x480	703ms
800*600	1.564 ms

4.3. Infrared Approach Results

This approach depends on the infrared sensor, this sensor detects the position of infrared marker with no need for programming code. Table 7 shows detection rate over distance. The drawback of this approach is that, it may fail to detect the infrared marker in sunny environments.

Table 7. Detection Ratio Over Distance

Distance	Detection Ratio
1M	100%
2M	100%
5M	100%
10M	98%
15M	94%

4.4. A Comparison between the Approaches

The first approach was not very reliable to detect the hand since it caused a malfunction in the whole system and an unexpected result. Also the time to process the frame was very high and not suitable for human interaction. The second approach was very reliable

to detect the hand but it was very slow to interact with the operator because the system processes two frames in order to detect the hand. The third approach was very reliable and fast because the computer acquired the position of the LED marker in little time. Table 7 shows a comparison between the three approaches.

Table 8. Comparison Between The three Approaches

	Reliability	**Error Rate**	**Speed**	**Response Time image size 800x600**	**Hardware used**	**Price**
Two dimensional approach	Not reliable	36.47%	Slow	638ms	Single webcam	10$
Three dimensional approach	Reliable	11.2%	Very slow	1564 ms	Two Webcams	20$
Infrared sensor approach	Very reliable	2%	Very Fast	2ms	Wiimote and Fm transmitter receiver	40$

Chapter 5

CONCLUSION

Virtual Smart Board implementation is a matter of intelligent system. In this study the functions implemented were mouse actions (positioning, left & right click). These actions relied on the hand's position compared to the site of view and the gesture of the hand. Hand tracking and gesture recognition were implemented in three approaches. The first approach was dependent on a single camera where the hand was tracked in two dimensional space. In this approach the video acquired from camera was processed to obtain the skin regions, these regions entered the neural network. This network recognized the hand and fed another network to recognize the gesture of the hand. The problem of this approach was that it was too slow and less reliable because noise came with the data used to recognize the hand. The second approach was dependent on two cameras which used the stereo vision technique in order to detect the hand's distance and calculate the three dimensional position of the hand. This approach was reliable but very slow because the computer had to process two frames to obtain the hand's position. The third approach was dependent on an infrared sensor to detect the infrared LED marker fixed on a glove with switches and transmitter to interact with human action. This approach was very reliable and fast but needed a glove to detect the hand position. All of the three approaches were very cheap compared with the price of the Smart Board

manufactured by smart technology where highest price of one is 55$ while the Smart Board is in the thousands. The best approach to use was the infrared approach because of the reliability and, response time; and it could be used with slow computers without noticeable processing results.

Future development for implementing virtual smart board may be by using Microsoft accelerator for kinect because it is more easer to detect the motion and hand position in acceptable amount of time, and it is not expensive equipment. This tool kit will be used to implement more function of virtual smart board, such as digital ink pens.

REFERENCES

[1] Ahrenkilde, Mona, ArildsenCarrara, Ann Sofie, SMARTS Boards i undervisningen, Basic Studies in Technological Humanities, (2012).

[2] J. Rehg, T. Kanade. DigitEyes: Vision-Based Human Hand-Tracking.School of Computer Science Technical Report CMU-CS-93-220, CarnegieMellon University, December 1993.

[3] Z. Zhang, Y. Wu, Y. Shan, S. Shafer. Visual panel: Virtual mouse keyboardand 3d controller with an ordinary piece of paper. In Proceedings ofPerceptual User Interfaces, 2001.

[4]TjandranegaraEdwin,Distance Estimation Algorithm for Stereo PairImages, ECE Technical Reports. Paper 64,(2005).

[5] Gonzalez, R. C., Woods, R.E., Digital Image Processing, Addison Wesley,1992.

[6]A. Albiol, L. Torres, and E. J. Delp. "Optimum color spacesforskin detection."in proceedings of the 2001international conference on image processing, volume 1, vol. 1, pp. 122-124 ,2001.

[7] ALBIOL, A., TORRES, L., AND DELP, E. J. 2001. Optimum color spaces for skin detection. In Proceedings of the International Conference on Image Processing, vol. 1, 122–124.

[8] Richard O. Duda, Peter E. Hart and David G. Stork, Pattern Classification, Second ed., Wiley-Interscience, 2000.

[9] CHAI, D., AND BOUZERDOUM, A. 2000. A bayesian approach to skin color classification in ycbcr color space. In Proceedings IEEE Region Ten Conference (TENCON'2000), vol. 2, 421–424.

[10] S. Haykin. (1999) "Neural Networks - A Comprehensive Foundation", Englewood Cliffs, NJ: Prentice-Hall, Second Edition

[11]Shweta K. Yewale, Pankaj K. Bharne. (2011, Apr.) "Artificial Neural Network Approach For Hand Gesture Recognition", International Journal of Engineering Science and Technology (IJEST), vol. 3(4), pp. 2603- 2608.

[12]C.M.Bishop,(1995) "Neural Networks for Pattern Recognition" London, U.K.:Oxford University Press.

[13] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In iccv, page 666. Published by the IEEE Computer Society, 1999.

[14] Cooper, R. (1995). Optometrists Network. Retrieved June 5, 2012, from http://www.vision3d.com/stereo.html

[15] Yi Ma, J.K., Stefano Soatto, Shankar Sastry An Invitation to 3-D Vision From Images to Models. 2001.

[16] B. McKinnon and J. Baltes. Practical region-based matching for stereo visionIn R. Klette and J. D. Zunic, editors, IWCIA, volume 3322

APPENDICES

Appendix A: Two Dimensional Tools C++

DataCollecting.cpp

```
// DataCollecting.cpp
//

#include"stdafx.h"
#include<cv.h>
#include<cxcore.h>
#include<highgui.h>
#include<stdio.h>
#include<ctype.h>
#include<cxcore.h>
#include<blob.h>
#include<BlobResult.h>
#include<iostream>

int _tmain(int argc, _TCHAR* argv[])
{
      double mR,mG,mB,MaxRGB;
      uchar *aPixelIn;
      unsignedint  data[400][560];
      IplImage* imag1;

      imag1=cvLoadImage("image (1).bmp",1);
      IplImage* imag=cvCreateImage(cvSize(320,240), 8, 3 );
      cvResize(imag1,imag,1);
      IplImage* R = cvCreateImage( cvGetSize(imag), 8, 1 );
    IplImage* G = cvCreateImage( cvGetSize(imag), 8, 1 );
      IplImage* B = cvCreateImage( cvGetSize(imag), 8, 1 );
      CvSize Size;
      Size=cvGetSize(imag);
      IplImage* gray=cvCreateImage(cvGetSize(imag),8,1);
      IplImage* BW1=cvCreateImage(cvGetSize(imag),8,1);
      IplImage* HSV=cvCreateImage(cvGetSize(imag),8,3);
      char  name[16];
      for(int Tr=1;Tr<=400;Tr++){
        sprintf(name,"image (%d).bmp",Tr);
            imag1=cvLoadImage(name,1);  // Read Image Data
            cvResize(imag1,imag,1);     //Preprocess
          cvCvtColor(imag,HSV,36);    //Convert To Hsve
          cvSplit(imag,R,G,B,0);
          mR=cvMean(R,0);
          mG=cvMean(G,0);
          mB=cvMean(B,0);
          MaxRGB=mR;
      if(mG>MaxRGB) MaxRGB=mG;
      if(mB>MaxRGB) MaxRGB=mB;
          mR=mR/MaxRGB;
          mB=mB/MaxRGB;
          mG=mG/MaxRGB;
            cvConvertScale(R,R,mR,0);
            cvMerge(R,G,B,0,imag);
```

```
	cvInRangeS(HSV,cvScalar(0,133,77),cvScalar(1000,173,127),BW1);//
Filter Skin Regions
		cvCvtColor(imag,gray,6);
		cvAnd(gray,BW1,gray,0);
		cvCanny(BW1,BW1,1,100,3);
		aPixelIn=(uchar*)BW1->imageData;

		for(int x=0;x<Size.height;x++){
			data[Tr-1][x]=0;
			for(int y=0;y<Size.width;y++){
				data[Tr-1][x]+=aPixelIn[x*gray-
>widthStep+y]/255;//Make Data IN Row and Take The Summation of the
Binary Images
			}
		}
		for(int y=0;y<Size.width;y++){
			data[Tr-1][Size.height+y]=0;
			for(int x=0;x<Size.height;x++){

				data[Tr-1][Size.height+y]+=aPixelIn[x*gray-
>widthStep+y]/255;
			}
		}

	}
	FILE *Fsm=fopen("posative.dat","w");
for(int i=0;i<400;i++)
	{
		for(int j=0;j<560;j++)
		{
			fprintf(Fsm,"%d \n",data[i][j]);// Write Data to File
		}

	}
	fclose(Fsm);
	cvWaitKey(0);
	return 0;
}
```

Neural Network Training

```
// neural.cpp : Defines the entry point for the console application.
//

#include"stdafx.h"
#include<cv.h>
#include<ml.h>
//neural network Declearation
CvANN_MLP machineBrain;
// Read the training data and train the network.
void trainMachine(){
	int i;
	//The number of training samples.
	int train_sample_count;
```

```
	//The data sample consists of 200 inputs and two outputs. 1000 is
the number of trianing samples.
	float td[1000][201];
	//Read the training file
	FILE *fin;   fin = fopen("train.dat", "r");
	//Get the number of samples.
	scanf( "%d", &train_sample_count);
	printf("Found training file with %d samples...\n",
train_sample_count);
	CvMat* trainData = cvCreateMat(train_sample_count, 2, CV_32FC1);
	//Output data samples. Matrix of order (train_sample_count x 1)
	CvMat* trainClasses = cvCreateMat(train_sample_count, 1,
CV_32FC1);
	//The weight of each training data sample. We'll later set all to
equal weights.
	CvMat* sampleWts = cvCreateMat(train_sample_count, 1, CV_32FC1);
	//The matrix representation of our ANN. We'll have four layers.
	CvMat* neuralLayers = cvCreateMat(, 1, CV_32SC1);
	CvMat trainData1, trainClasses1, neuralLayers1, sampleWts1;
	cvGetRows(trainData, &trainData1, 0, train_sample_count);
	cvGetRows(trainClasses, &trainClasses1, 0, train_sample_count);
	cvGetRows(trainClasses, &trainClasses1, 0, train_sample_count);
	cvGetRows(sampleWts, &sampleWts1, 0, train_sample_count);
	cvGetRows(neuralLayers, &neuralLayers1, 0, 4);
	//Setting the number of neurons on each layer of the ANN
	/*     We have in Layer 1: 200 neurons (200 inputs)
                    Layer 2: 400 neurons (hidden layer)
                              Layer 4: 2 neurons (1 output)    */
	cvSet1D(&neuralLayers1, 0, cvScalar(200));
cvSet1D(&neuralLayers1, 1, cvScalar(400));
	cvSet1D(&neuralLayers1, 2, cvScalar(2));
	//Read  samples from file.
	for (i=0;i<train_sample_count;i++)
		for(int j=0;j<202;j++)
			scanf("%f",&td[i][j]);
	fclose(fin);   //Assemble the ML training data.
	for (i=0; i<train_sample_count; i++)   {
		for(j=0;j<202;j++){
			//Input OutPut
		    cvSetReal2D(&trainData1, i, j, td[i][j]);
		//Weight (setting everything to 1)
		    cvSet1D(&sampleWts1, i, cvScalar(1));
	}
	//Create our ANN.
	machineBrain.create(neuralLayers);
	//Train it with our data.
	machineBrain.train(
		trainData,
		trainClasses,
		sampleWts,
		0,

	CvANN_MLP_TrainParams(cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT
_EPS,
		100000,
		1.0
		),
```

```
            CvANN_MLP_TrainParams::BACKPROP,
            0.01,
            0.05
            )
            );
}
```

Warping.cpp

```
class Warper
    {
float* Xsrc = newfloat[4];
      float* Ysrc = newfloat[4];
      float* Xdst = newfloat[4];
      float* Ydst = newfloat[4];
float* MatSrc = newfloat[16];
float* MatDst = newfloat[16];
float* MatCalb = newfloat[16];
      bool dirty;
public Warper()
            {
                TransformationMat();
            }

publicvoid TransformationMat()
        {
           GetSource(0.0f, 0.0f,
                           1.0f, 0.0f,
                           0.0f, 1.0f,
                           1.0f, 1.0f);
            GetDestination(0.0f, 0.0f,
                           1.0f, 0.0f,
                           0.0f, 1.0f,
                           1.0f, 1.0f);
            OutCalc();
        }

publicvoid GetSource(float x0,
                         float y0,
                         float x1,
                         float y1,
                         float x2,
                         float y2,
                         float x3,
                         float y3){
              Xsrc[0] = x0;
              Ysrc[0] = y0;
              Xsrc[1] = x1;
              Ysrc[1] = y1;
              Xsrc[2] = x2;
              Ysrc[2] = y2;
              Xsrc[3] = x3;
              Ysrc[3] = y3;
            dry = true;
        }

publicvoid GetDestination(float x0,
```

```
                        float y0,
float x1,
                              float y1,
                              float x2,
                              float y2,
                              float x3,
                              float y3){
              Xdst[0] = x0;
              Ydst[0] = y0;
              Xdst[1] = x1;
              Ydst[1] = y1;
              Xdst[2] = x2;
              Ydst[2] = y2;
              Xdst[3] = x3;
              Ydst[3] = y3;
            dry = true;
        }


publicvoid OutCalc() {
              DoComputationSquare(  Xsrc[0],Ysrc[0],
                        Xsrc[1],Ysrc[1],
                                    Xsrc[2],Ysrc[2],
                                    Xsrc[3],Ysrc[3],
                                    MatSrc);
              DoComputationQuad(    Xdst[0], Ydst[0],
                                    Xdst[1], Ydst[1],
                                    Xdst[2], Ydst[2],
                                    Xdst[3], Ydst[3],
                                    MatDst);
              multMats(MatSrc, MatDst, MatCalb);
              dirty = false;
            }

publicvoid multMats(float* MatSrc, float* MatDst, float*
                  resMat) {

      for (int r = 0; r < 4; r++) {
      int ri = r * 4;
      for (int c = 0; c < 4; c++) {
                   resMat[ri + c] = (MatSrc[ri    ] * MatDst[c     ] +
                                MatSrc[ri + 1] * MatDst[c +  4] +
                                MatSrc[ri + 2] * MatDst[c +  8] +
                                MatSrc[ri + 3] * MatDst[c + 12]);
                         }
                   }
            }

publicvoid DoComputationQuad( float x0,
                              float y0,
                                          float x1,
                                          float y1,
                                          float x2,
                                          float y2,
                                          float x3,
                                          float y3,
                                          float* mat) {
```

```
      float dx1 = x1 - x2,     dy1 = y1 - y2;
      float dx2 = x3 - x2,     dy2 = y3 - y2;
      float sx = x0 - x1 + x2 - x3;
      float sy = y0 - y1 + y2 - y3;
      float g = (sx * dy2 - dx2 * sy) / (dx1 * dy2 - dx2 * dy1);
      float h = (dx1 * sy - sx * dy1) / (dx1 * dy2 - dx2 * dy1);
      float a = x1 - x0 + g * x1;
      float b = x3 - x0 + h * x3;
      float c = x0;
      float d = y1 - y0 + g * y1;
      float e = y3 - y0 + h * y3;
      float f = y0;

              mat[ 0] = a;    mat[ 1] = d;      mat[ 2] = 0;      mat[
3] = g;
              mat[ 4] = b;    mat[ 5] = e;      mat[ 6] = 0;      mat[
7] = h;
              mat[ 8] = 0;    mat[ 9] = 0;      mat[10] = 1;
      mat[11] = 0;
              mat[12] = c;    mat[13] = f;      mat[14] = 0;
      mat[15] = 1;
            }

publicvoid DoComputationSquare(     float x0,
                                        float y0,
                                        float x1,
                                        float y1,
                                        float x2,
                                        float y2,
                                        float x3,
                                        float y3,
                                        float* mat) {
              DoComputationQuad(x0,y0,x1,y1,x2,y2,x3,y3, mat);

      // invert through adjoint

      float a = mat[ 0],      d = mat[ 1],      /* ignore */
      g = mat[ 3];
      float b = mat[ 4],      e = mat[ 5],      /* 3rd col*/
      h = mat[ 7];
      /* ignore 3rd row */
      float c = mat[12],      f = mat[13];

      float A =     e - f * h;
      float B = c * h - b;
      float C = b * f - c * e;
      float D = f * g - d;
      float E =     a - c * g;
      float F = c * d - a * f;
      float G = d * h - e * g;
      float H = b * g - a * h;
      float I = a * e - b * d;
float idet = 1.0f / (a * A          + b * D          + c * G);

              mat[ 0] = A * idet;   mat[ 1] = D * idet;     mat[ 2] =
0;    mat[ 3] = G * idet;
```

```
                mat[ 4] = B * idet;   mat[ 5] = E * idet;     mat[ 6] =
0;    mat[ 7] = H * idet;
                mat[ 8] = 0       ;   mat[ 9] = 0       ;     mat[10] =
1;    mat[11] = 0       ;
                mat[12] = C * idet;   mat[13] = F * idet;     mat[14] =
0;    mat[15] = I * idet;
          }

publicfloat* getMatCalbrix()
          {
        return MatCalb;
          }

publicvoid warp(float Xsrc, float Ysrc, ref float Xdst, ref float Ydst)
          {
if (dirty)
                  OutCalc();
              Warper::warp(MatCalb, Xsrc, Ysrc, ref Xdst, ref Ydst);
          }

publicstaticvoid warp(float* mat, float Xsrc, float Ysrc, ref     float
Xdst, ref float Ydst){
float* result = newfloat[4];
float z = 0;
              result[0] = (float)(Xsrc * mat[0] + Ysrc*mat[4] + z*mat[8]
+ 1*mat[12]);
              result[1] = (float)(Xsrc * mat[1] + Ysrc*mat[5] + z*mat[9]
+ 1*mat[13]);
              result[2] = (float)(Xsrc * mat[2] + Ysrc*mat[6] + z*mat[10]
+ 1*mat[14]);
              result[3] = (float)(Xsrc * mat[3] + Ysrc*mat[7] + z*mat[11]
+ 1*mat[15]);
              Xdst = result[0]/result[3];
              Ydst = result[1]/result[3];
          }
    }
```

Appendix B: Three Dimensional Tools C++

Stereo.cpp

```
#include"cv.h"
#include"cxmisc.h"
#include"highgui.h"
#include"cvaux.h"
#include<vector>
#include<string>
#include<algorithm>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

usingnamespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      CvCapture *capture1= cvCaptureFromCAM(2);
    CvCapture* capture2= cvCaptureFromCAM(3);
      IplImage *frame1 = cvQueryFrame( capture1 );
      IplImage *frame2 = cvQueryFrame( capture2 );
      IplImage *gray1=cvCreateImage(cvSize(frame1->width,frame1-
>height),8,1);
      IplImage *gray2=cvCreateImage(cvSize(frame1->width,frame1-
>height),8,1);
      IplImage* Red = cvCreateImage( cvGetSize(frame1), 8, 1 );
    IplImage* Green = cvCreateImage( cvGetSize(frame1), 8, 1 );
      IplImage* Blue = cvCreateImage( cvGetSize(frame1), 8, 1 );
      IplImage* HSV = cvCreateImage( cvGetSize(frame1), 8, 3 );
      IplImage* BW1 = cvCreateImage( cvGetSize(frame1), 8, 1 );
      IplImage* BW2 = cvCreateImage( cvGetSize(frame1), 8, 1 );
      IplImage* gray_fr2 = cvCreateImage( cvGetSize(frame1), 8, 1 );
      CvPoint rect1, rect2;
    cvLoad("M1.xml",&M1);
    cvLoad("D1.xml",&D1);
    cvLoad("R1.xml",&Rl);
    cvLoad("P1.xml",&Pl);
    cvLoad("M2.xml",&M2);
    cvLoad("D2.xml",&D2);
    cvLoad("R2.xml",&Rr);
    cvLoad("P2.xml",&Pr);
    cvLoad("Q.xml",&_Q);
    cvLoad("mx1.xml",mxL);
    cvLoad("my1.xml",myL);
    cvLoad("mx2.xml",mxR);
    cvLoad("my2.xml",myR);



      CvMat* mxL = cvCreateMat( frame1->height,frame1->width, CV_32F );
      CvMat* myL = cvCreateMat( frame1->height,frame1->width, CV_32F );
      CvMat* mxR = cvCreateMat( frame1->height,frame1->width, CV_32F );
      CvMat* myR = cvCreateMat( frame1->height,frame1->width, CV_32F );
      CvMat* pair;
      CvMat part;
```

```
      cvStereoRectify(   &M1,   &M2,  &D1,  &D2, cvSize(frame1-
>width,frame1->height), &R,  &T ,&Rl  ,&Rr   ,&Pl  ,&Pr  ,0 ,0);
      bool isVerticalStereo = fabs(_Pr[1][3]) > fabs(_Pr[0][3]);
      if( !isVerticalStereo )
            pair = cvCreateMat( frame1->height,frame1->width*2,CV_8UC3
);
else
            pair = cvCreateMat( frame1->height*2,frame1->width,CV_8UC3
);
      cvInitUndistortRectifyMap(&M1,&D1,&Rl,&Pl,mxL,myL);
      cvInitUndistortRectifyMap(&M2,&D2,&Rr,&Pr,mxR,myR);
      CvMat* img1r = cvCreateMat( frame1->height,frame1->width, CV_8U
);
      CvMat* img2r = cvCreateMat( frame1->height,frame1->width, CV_8U
);
      CvMat* disp = cvCreateMat( frame1->height,frame1->width, CV_16S
);
       CvMat* vdisp = cvCreateMat( frame1->height,frame1->width, CV_8U
);
      CvStereoBMState *BMState = cvCreateStereoBMState();
        assert(BMState != 0);
        BMState->preFilterSize=41;
        BMState->preFilterCap=31;
        BMState->SADWindowSize=41;
        BMState->minDisparity=-64;
        BMState->numberOfDisparities=128;
        BMState->textureThreshold=10;
        BMState->uniquenessRatio=15;
      int key = 0;
      CBlobResult blobs;
      CBlob currentBlob;
      while(true)
      {
            frame1 = cvQueryFrame( capture1 );
            frame2 = cvQueryFrame( capture2 );
          cvCvtColor(frame1,gray1,CV_BGR2GRAY );
            cvCvtColor(frame2,gray2,CV_BGR2GRAY );
        cvRemap( gray1, img1r, mxL, myL );
        cvRemap( gray2, img2r, mxR, myR );
            cvFindStereoCorrespondenceBM( img1r, img2r, disp, BMState);
        cvNormalize( disp, vdisp, 0, 255, CV_MINMAX );
            if( !isVerticalStereo )
                {
                    cvGetCols( pair, &part, 0, frame1->width );
                    cvCvtColor( img1r, &part, CV_GRAY2BGR );
                    cvGetCols( pair, &part, frame1->width,frame1-
>width*2 );
                    cvCvtColor( img2r, &part, CV_GRAY2BGR );
for( int j = 0; j < frame1->height; j += 16 )
                        cvLine( pair, cvPoint(0,j),
                        cvPoint(frame1->width*2,j),
                        CV_RGB(0,255,0));
                }
else
                {
                    cvGetRows( pair, &part, 0, frame1->height );
```

```
                    cvCvtColor( img1r, &part, CV_GRAY2BGR );
                    cvGetRows( pair, &part, frame1->height,
                        frame1->height*2 );
                    cvCvtColor( img2r, &part, CV_GRAY2BGR );
for( int j = 0; j < frame1->width; j += 16 )
                        cvLine( pair, cvPoint(j,0),
                        cvPoint(j,frame1->height*2),
                        CV_RGB(0,255,0));
                }
        }

      cvReleaseCapture( &capture1 );
      cvReleaseCapture( &capture2 );


}
```

Appendix C: Infrared Approach Tools C#

warper.cs

```
class Warper
    {
float [] Xsrc = newfloat[4];
      float [] Ysrc = newfloat[4];
      float [] Xdst = newfloat[4];
      float [] Ydst = newfloat[4];
float [] MatSrc = newfloat[16];
float [] MatDst = newfloat[16];
float [] MatCalb = newfloat[16];
      bool dirty;

public Warper()
            {
                TransformationMat();
            }

publicvoid TransformationMat()
        {
           GetSource(0.0f, 0.0f,
                           1.0f, 0.0f,
                           0.0f, 1.0f,
                           1.0f, 1.0f);
            GetDestination(0.0f, 0.0f,
                           1.0f, 0.0f,
                           0.0f, 1.0f,
                           1.0f, 1.0f);
            OutCalc();
        }

publicvoid GetSource(   float x0,
                        float y0,
                        float x1,
```

```
                        float y1,
                        float x2,
                        float y2,
                        float x3,
                        float y3){
                Xsrc[0] = x0;
                Ysrc[0] = y0;
                Xsrc[1] = x1;
                Ysrc[1] = y1;
                Xsrc[2] = x2;
                Ysrc[2] = y2;
                Xsrc[3] = x3;
                Ysrc[3] = y3;
               dirty = true;
        }

publicvoid GetDestination(float x0,
                                        float y0,
                                        float x1,
                                        float y1,
                                        float x2,
                                        float y2,
                                        float x3,
                                        float y3){
                Xdst[0] = x0;
                Ydst[0] = y0;
                Xdst[1] = x1;
                Ydst[1] = y1;
                Xdst[2] = x2;
                Ydst[2] = y2;
                Xdst[3] = x3;
                Ydst[3] = y3;
               dirty = true;
        }


publicvoid OutCalc() {
              DoComputationSquare(  Xsrc[0],Ysrc[0],
                                                Xsrc[1],Ysrc[1],
                                                Xsrc[2],Ysrc[2],
                                                Xsrc[3],Ysrc[3],
                                                MatSrc);
              DoComputationQuad(    Xdst[0], Ydst[0],
                                                Xdst[1], Ydst[1],
                                                Xdst[2], Ydst[2],
                                                Xdst[3], Ydst[3],
                                                MatDst);
              multMats(MatSrc, MatDst, MatCalb);
              dirty = false;
            }

publicvoid multMats(float [] MatSrc, float [] MatDst, float [] resMat)
{

      for (int r = 0; r < 4; r++) {
      int ri = r * 4;
      for (int c = 0; c < 4; c++) {
```

```
                    resMat[ri + c] = (MatSrc[ri    ] * MatDst[c     ] +
                                      MatSrc[ri + 1] * MatDst[c +  4] +
                                      MatSrc[ri + 2] * MatDst[c +  8] +
                                      MatSrc[ri + 3] * MatDst[c + 12]);
                            }
                    }
            }

publicvoid DoComputationQuad( float x0,
                                                        float y0,
                                                        float x1,
                                                        float y1,
                                                        float x2,
                                                        float y2,
                                                        float x3,
                                                        float y3,
                                                        float [] mat) {

      float dx1 = x1 - x2,    dy1 = y1 - y2;
      float dx2 = x3 - x2,    dy2 = y3 - y2;
      float sx = x0 - x1 + x2 - x3;
      float sy = y0 - y1 + y2 - y3;
      float g = (sx * dy2 - dx2 * sy) / (dx1 * dy2 - dx2 * dy1);
      float h = (dx1 * sy - sx * dy1) / (dx1 * dy2 - dx2 * dy1);
      float a = x1 - x0 + g * x1;
      float b = x3 - x0 + h * x3;
      float c = x0;
      float d = y1 - y0 + g * y1;
      float e = y3 - y0 + h * y3;
      float f = y0;

              mat[ 0] = a;    mat[ 1] = d;      mat[ 2] = 0;      mat[
3] = g;
              mat[ 4] = b;    mat[ 5] = e;      mat[ 6] = 0;      mat[
7] = h;
              mat[ 8] = 0;    mat[ 9] = 0;      mat[10] = 1;
      mat[11] = 0;
              mat[12] = c;    mat[13] = f;      mat[14] = 0;
      mat[15] = 1;
            }

publicvoid DoComputationSquare(     float x0,
                                                        float y0,
                                                        float x1,
                                                        float y1,
                                                        float x2,
                                                        float y2,
                                                        float x3,
                                                        float y3,
                                                        float [] mat) {
              DoComputationQuad(x0,y0,x1,y1,x2,y2,x3,y3, mat);

      // invert through adjoint

      float a = mat[ 0],      d = mat[ 1],      /* ignore */
      g = mat[ 3];
```

```
        float b = mat[ 4],      e = mat[ 5],      /* 3rd col*/
        h = mat[ 7];
        /* ignore 3rd row */
        float c = mat[12],      f = mat[13];

        float A =     e - f * h;
        float B = c * h - b;
        float C = b * f - c * e;
        float D = f * g - d;
        float E =     a - c * g;
        float F = c * d - a * f;
        float G = d * h - e * g;
        float H = b * g - a * h;
        float I = a * e - b * d;
        float idet = 1.0f / (a * A           + b * D           + c * G);

                mat[ 0] = A * idet;   mat[ 1] = D * idet;     mat[ 2] =
0;     mat[ 3] = G * idet;
                mat[ 4] = B * idet;   mat[ 5] = E * idet;     mat[ 6] =
0;     mat[ 7] = H * idet;
                mat[ 8] = 0       ;   mat[ 9] = 0       ;     mat[10] =
1;     mat[11] = 0       ;
                mat[12] = C * idet;   mat[13] = F * idet;     mat[14] =
0;     mat[15] = I * idet;
          }

publicfloat [] getMatCalbrix()
          {
        return MatCalb;
          }

publicvoid warp(float Xsrc, float Ysrc, ref float Xdst, ref float Ydst)
          {
if (dirty)
                  OutCalc();
                    Warper.warp(MatCalb, Xsrc, Ysrc, ref Xdst, ref Ydst);
          }

publicstaticvoid warp(float [] mat, float Xsrc, float Ysrc, ref float
Xdst, ref float Ydst){
float [] result = newfloat[4];
float z = 0;
              result[0] = (float)(Xsrc * mat[0] + Ysrc*mat[4] + z*mat[8]
+ 1*mat[12]);
              result[1] = (float)(Xsrc * mat[1] + Ysrc*mat[5] + z*mat[9]
+ 1*mat[13]);
              result[2] = (float)(Xsrc * mat[2] + Ysrc*mat[6] + z*mat[10]
+ 1*mat[14]);
              result[3] = (float)(Xsrc * mat[3] + Ysrc*mat[7] + z*mat[11]
+ 1*mat[15]);
              Xdst = result[0]/result[3];
                  Ydst = result[1]/result[3];
          }
    }
```